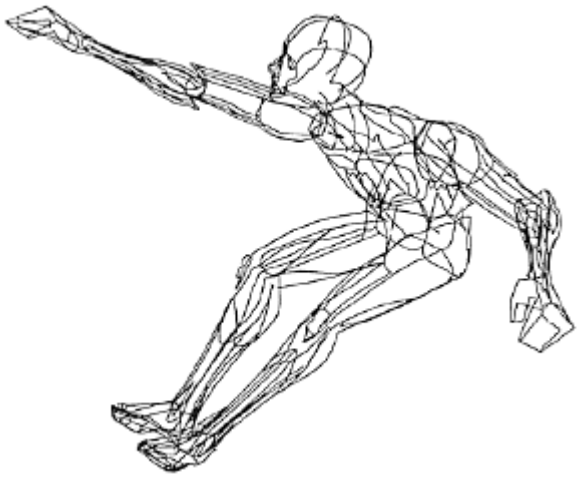
Bài giảng

Đồ họa

Tổng quan về đồ họa máy tính

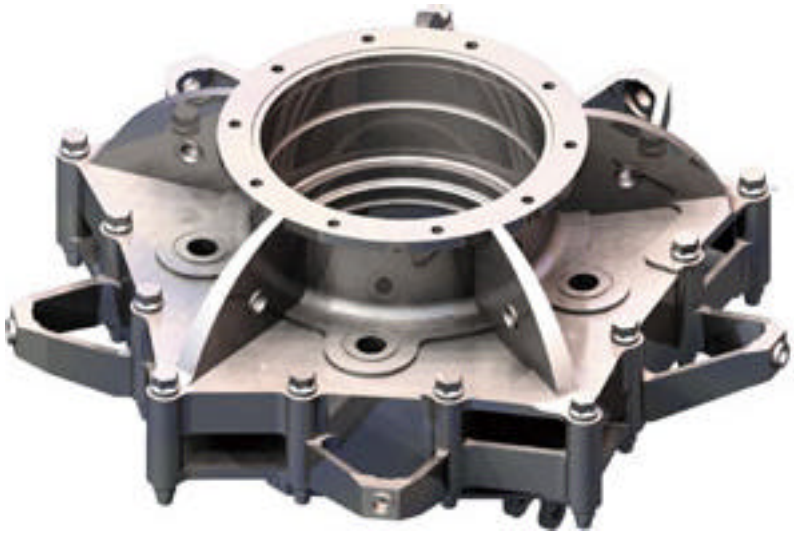
Khái niệm

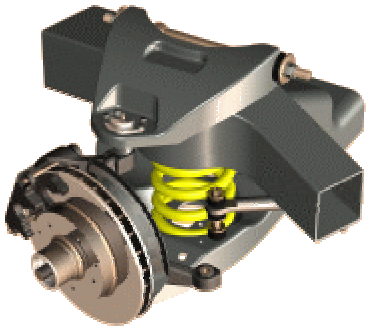
- Đồ họa máy tính có thể được hiểu như là tất cả những gì liên quan đến việc tạo ra ảnh (image) bằng máy tính. Chúng bao gồm : tạo, lưu trữ, thao tác trên các mô hình (model) và các ảnh.
- Thuật ngữ đồ họa máy tính (computer graphics) do William Fetter đặt ra năm 1960 để mô tả một cách thiết kế mới khi đang làm việc tại hãng Boeing.
- Với cách này, anh ta đã tạo nhiều ảnh có thể sử dụng lại để có thể dễ dàng thiết kế buồng lái của phi công theo ý muốn.



Một số ứng dụng của đồ họa máy tính

- Hỗ trợ thiết kế (CAD - Computer Aided Design)

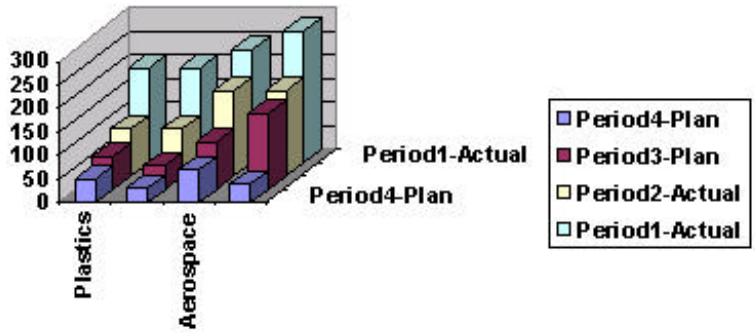


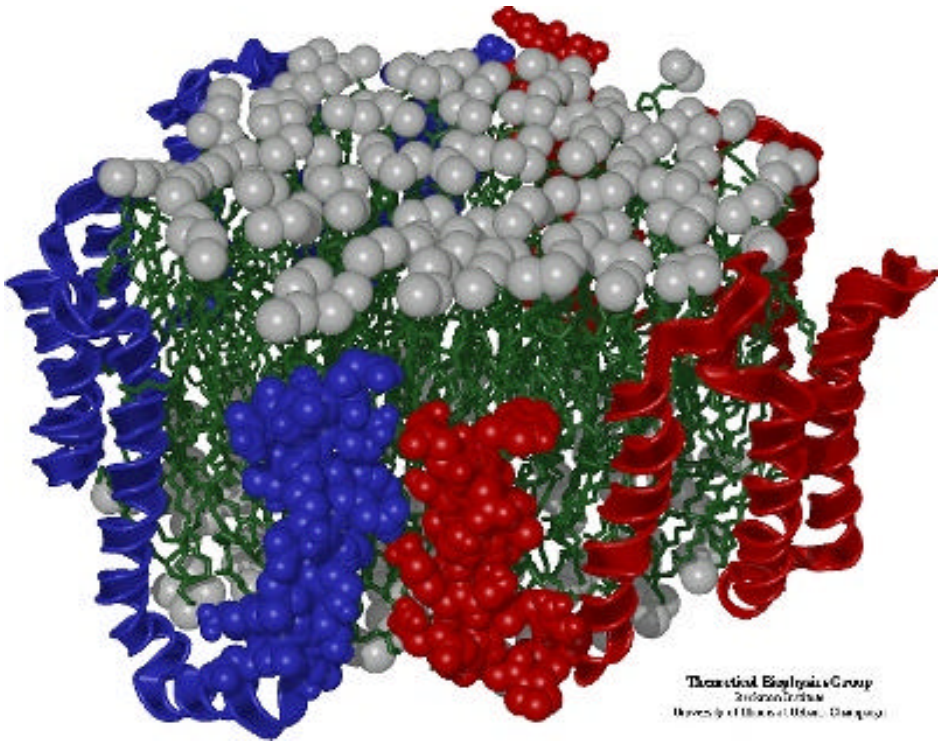


Gồm hai bước chính

- Phác thảo của phần khung(wireframe outline) mà từ đó có thể thấy được toàn bộ hình dạng và các thành phần bên trong của các đối tượng. Sử dụng kĩ thuật này, người thiết kế sẽ dễ dàng nhận thấy ngay các thay đổi của đối tượng khi tiến hành hiệu chỉnh các chi tiết hay thay đổi góc nhìn, ...
- Kết hợp các mô hình chiếu sáng, tô màu và tạo bóng bề mặt để tạo ra kết quả cuối cùng rất gần với thế giới thực.

- Visualization

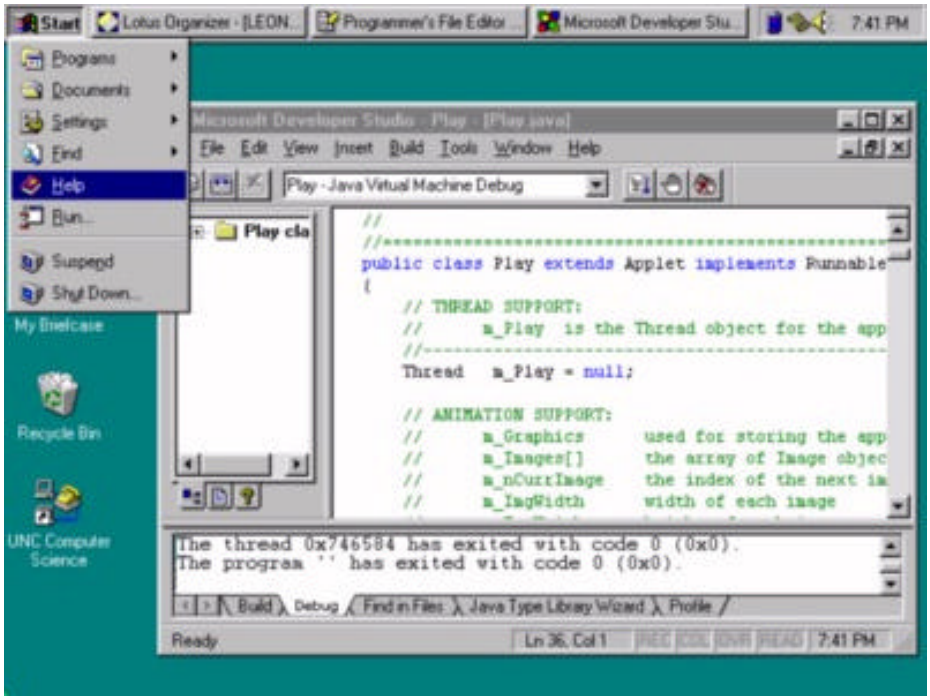




- ♦ Dùng phát sinh các biểu đồ, đồ thị, … trong việc minh họa mối quan hệ giữa nhiều đối tượng với nhau.
- ♦ Tóm lược các dữ liệu về tài chính, thống kê, kinh tế, khoa học, toán học, … giúp cho việc nghiên cứu, quản lí, … một cách có hiệu quả.

- Giải trí

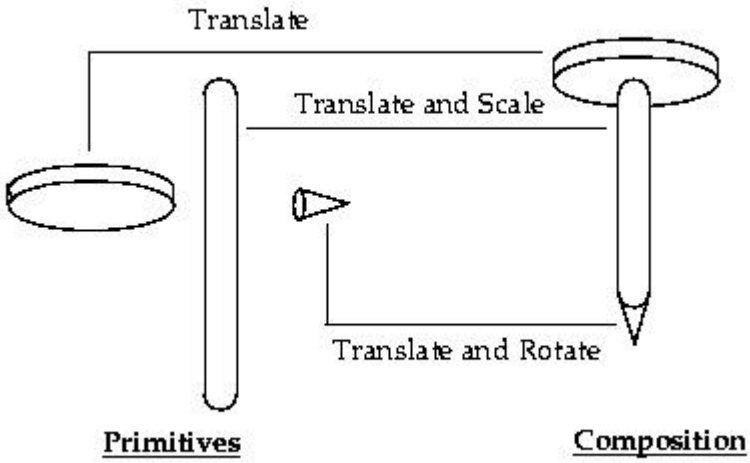
- Tạo giao diện



Tổng quan về một hệ đồ họa

- Các thành phần phần cứng
  - Thiết bị hiển thị : màn hình, máy in, ...
  - Thiết bị nhập : bàn phím, chuột, ...
- Các công cụ phần mềm
  - Công cụ ứng dụng (application package) : Được thiết kế cho các người sử dụng để tạo ra các hình ảnh mà không cần quan tâm tới các thao tác bên trong hoạt động như thế nào. Ví dụ : AutoCAD, Adobe Photoshop, 3D Studio, ...
  - Công cụ lập trình (programming package) : Cung cấp một tập các hàm đồ họa có thể được dùng trong các ngôn ngữ lập trình cấp cao như C, Pascal, ... Ví dụ : GRAPH.TPU, GRAPHICS.LIB, Open GL, ...
- Các chuẩn phần mềm
  - Ra đời để đáp ứng tính tương thích : Nếu các phần mềm được thiết kế với các hàm đồ họa chuẩn chúng có thể dùng được cho nhiều hệ phần cứng và môi trường làm việc khác nhau.
  - GKS (Graphics Kernel System) là chuẩn ra đời đầu tiên cho việc phát triển các phần mềm đồ họa. Ban đầu GKS được thiết kế chỉ dùng cho tập các công cụ đồ họa hai chiều, sau đó mới được mở rộng ra cho đồ họa ba chiều.
  - Các hàm của GKS thực sự chỉ là các mô tả trừu tượng, độc lập với bất kì ngôn ngữ lập trình nào. Để cài đặt một chuẩn đồ họa cho ngôn ngữ cụ thể nào, các cú pháp tương ứng sẽ được xác định và cụ thể hóa.

- Các thành phần của công cụ lập trình
  - Tập các công cụ tạo ra các đối tượng đồ họa cơ sở như điểm, đoạn thẳng, đường cong, vùng tô, kí tự, ...
  - Tập các công cụ thay đổi thuộc tính của các đối tượng cơ sở kể trên như màu sắc, kiểu đường, kiểu chữ, mẫu tô...
  - Tập các công cụ thực hiện các phép biến đổi hình học dùng để thay đổi kích thước, vị trí, hướng, ...
  - Tập các công cụ biến đổi hệ quan sát dùng để xác định vị trí quan sát của các đối tượng và vị trí trên thiết bị hiển thị đối tượng.
  - Tập các công cụ nhập liệu : các ứng dụng đồ họa có thể sử dụng nhiều loại thiết bị nhập khác nhau như chuột, bàn phím, bút vẽ, bảng, ... để điều khiển và xử lí dòng dữ liệu nhập.
  - Tập các công cụ chứa các thao tác dùng cho quản lí và điều khiển như khởi tạo và đóng chế độ đồ họa, xóa toàn bộ màn hình, ...



Hai mô hình cơ bản của ứng dụng đồ họa

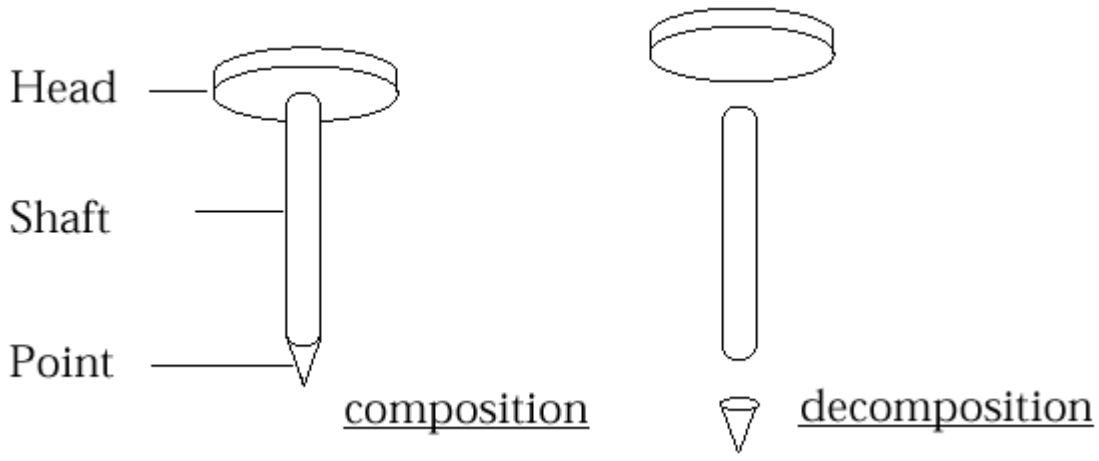
- Ứng dụng đồ họa dựa trên mẫu số hóa (sampled-based graphics)
  - Các pixel (điểm ảnh) được tạo ra bởi thao tác số hóa ảnh bằng cách sử dụng các chương trình vẽ dựa trên mẫu số hóa hay máy quét.
  - Các ứng dụng thuộc dạng này gồm : PaintBrush, Adobe Photoshop, ...
- Ứng dụng đồ họa dựa trên đặc trưng hình học (geometry-based graphics)
  - Dùng các đặc trưng hình học và các thuộc tính để mô tả đối tượng. Sau đó các đối tượng sẽ được số hóa để phục vụ cho hiển thị.
  - Các ứng dụng thuộc dạng này : Adobe Illustrator, AutoCAD, ...

Ứng dụng đồ họa dựa trên mẫu số hóa

- Các đối tượng đồ họa được tạo ra bởi lưới các pixel rời rạc.
- Các pixel này có một mô tả về tọa độ để xác định vị trí và giá trị mẫu (sample values), thông thường là độ sáng hay màu sắc.
- Các pixel này có thể được tạo ra bằng các chương trình vẽ, máy quét, ...
- Khi một ảnh được xác định bởi tập các pixel, chúng có thể có các thao tác :
  - Biên tập ảnh (image editting) : cắt, dán các vùng trên ảnh, sử dụng các công cụ tô màu để hiệu chỉnh, ...
  - Xử lí ảnh (image processing) : sử dụng các thuật toán để thay đổi ảnh mà không có sự can thiệp của người dùng, bao gồm : làm nhòe ảnh (blurring), làm nét ảnh (sharpening), dò đường biên (edge-detection), cân chỉnh màu sắc, ..
- Một số thuận lợi
  - Dễ dàng thay đổi ảnh bằng cách thay đổi màu sắc hay vị trí của các pixel, ví dụ như lấy ảnh âm bản, ...
  - Có thể di chuyển các vùng ảnh từ nơi này sang nơi khác dễ dàng.
- Một số bất lợi
  - Không thể xem xét đối tượng từ các góc nhìn khác nhau.
  - Hiệu chỉnh về thuộc tính hình học, kích thước phức tạp.

Ứng dụng đồ họa dựa trên đặc trưng hình học

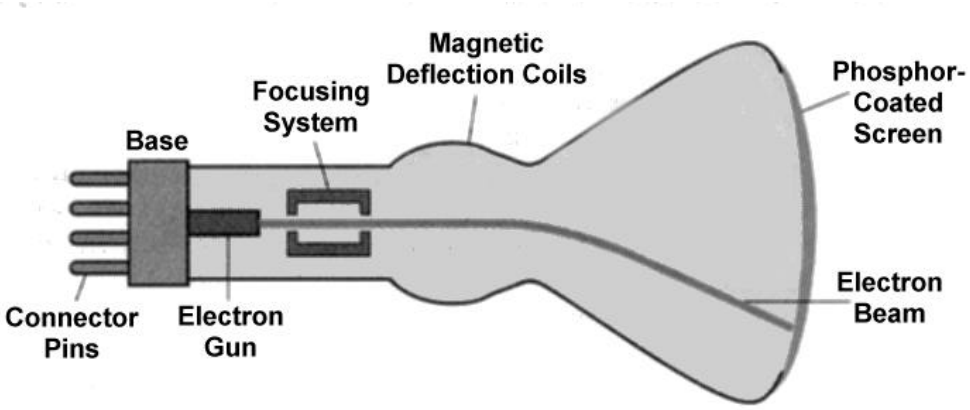
- Các đối tượng đồ họa cơ sở như đoạn thẳng, đa giác, ... được lưu trữ bằng các mô hình (model) và các thuộc tính (attribute) của chúng.
  - ♦ Các mô hình thực chất là các mô tả toán học, ví dụ đoạn thẳng được mô hình bằng hai điểm đầu, cuối, ...
  - ♦ Các thuộc tính được dùng để mô tả cách mà các đối tượng được hiển thị ví dụ như màu sắc, độ dày, ..
- Các ảnh được tạo bởi tập các pixel thông qua việc số hóa các đặc trưng hình học phục vụ cho mỗi yêu cầu hiển thị. Các ảnh có thể khác nhau tùy vào mỗi yêu cầu hiển thị khác nhau, nhưng đều xuất phát từ một mô hình.
- Người dùng không thao tác trực tiếp với từng pixel của ứng dụng dạng này mà thao tác trên các thành phần hình học của đối tượng, sau đó số hóa lại rồi mới hiển thị.



Thiết bị hiển thị : Màn hình

Cấu tạo của CRT

- Một chùm các tia điện tử (tia âm cực) phát ra từ một súng điện tử, vượt qua các hệ thống hội tụ (focusing) và dẫn hướng (deflection) sẽ hướng tới các vị trí xác định trên màn hình được phủ một lớp phosphor.



- Tại mỗi vị trí tương tác với tia điện tử, hạt phosphor sẽ phát ra một chấm sáng nhỏ. Vì ánh sáng phát ra bởi các hạt phosphor mờ dần rất nhanh nên cần phải có một cách nào đó để duy trì ảnh trên màn hình. Một trong các cách đó là lặp đi lặp lại nhiều lần việc vẽ lại ảnh thật nhanh bằng cách hướng các tia điện tử trở lại vị trí cũ. Kiểu hiển thị này gọi là refresh CRT.

- Có nhiều loại phosphor được dùng trong một CRT. Ngoài màu sắc ra, điểm khác nhau chính giữa các loại phosphor là "độ bền" (persistent), đó là khoảng thời gian phát sáng sau khi tia CRT không còn tác động.

- Lớp phosphor có độ bền thấp cần tốc độ làm tươi cao hơn để giữ cho hình ảnh trên màn hình khỏi nhòe. Loại này thường rất tốt cho hoạt hình, rất cần thay đổi hình ảnh liên tục. Lớp phosphor có độ bền cao thường được dùng cho việc hiển thị các ảnh tĩnh, độ phức tạp cao. Mặc dù một số loại phosphor có độ bền lớn hơn 1 giây, tuy nhiên các màn hình đồ họa thường được xây dựng với độ bền dao động từ 10 đến 60 micro giây.

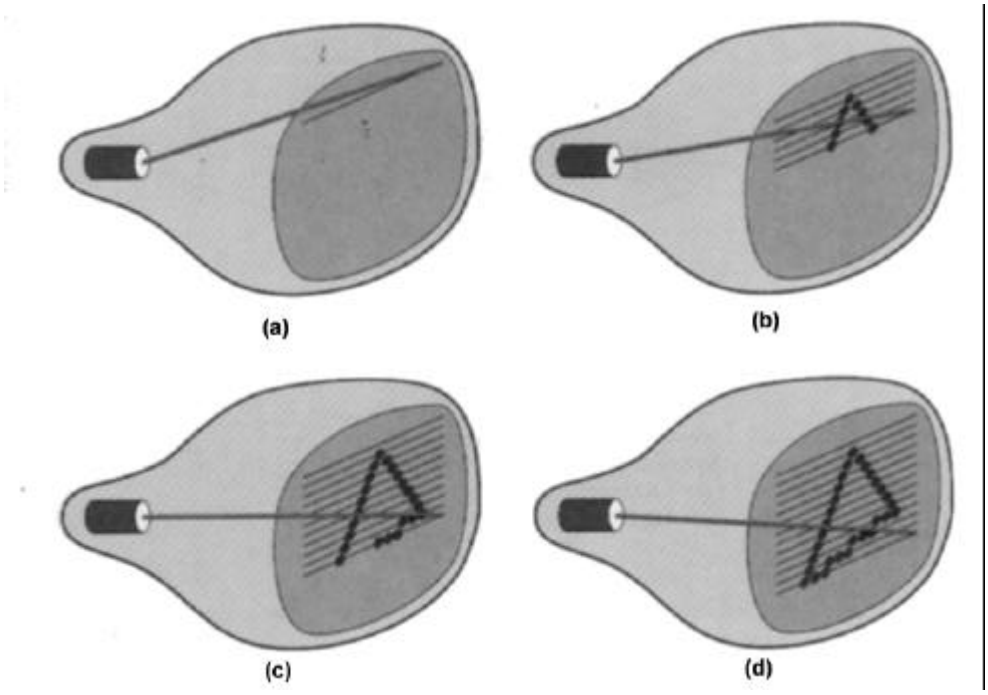
- Số lượng tối đa các điểm có thể hiển thị trên một CRT được gọi là độ phân giải (resolution).

- Kích thước vật lí của màn hình đồ họa được tính từ độ dài của đường chéo màn hình, thường dao động từ 12 đến 27 inch hoặc lớn hơn. Một màn hình CRT có thể được kết hợp với nhiều loại máy khác nhau, do đó số lượng các điểm trên màn hình có thể được vẽ thật sự còn tùy thuộc vào khả năng của hệ thống mà nó kết hợp vào.

- Tỉ số phương là tỉ lệ của các điểm dọc và các điểm ngang cần để phát sinh các đoạn thẳng có độ dài đơn vị theo cả hai hướng trên màn hình (trong một số trường hợp người ta thường dùng tỉ số phương như là tỉ số của các điểm theo chiều ngang so với các điểm theo chiều dọc). Với các màn hình có tỉ số phương khác 1, dễ dàng nhận thấy là các hình vuông hiển thị trên nó sẽ có dạng hình chữ nhật, các hình tròn sẽ có dạng hình ellipse.

Màn hình dạng điểm (raster - scan display):

- Chùm tia điện tử sẽ được quét ngang qua màn hình, mỗi lần một dòng và quét tuần tự từ trên xuống dưới. Sự bật tắt của các điểm sáng trên màn hình phụ thuộc vào cường độ của tia điện tử và đây chính là cơ sở của việc tạo ra hình ảnh trên màn hình.
- Mỗi điểm trên màn hình được gọi là một pixel. Các thông tin về hình ảnh hiển thị trên màn hình được lưu trữ trong một vùng bộ nhớ gọi là vùng đệm làm tươi (refresh buffer) hay là vùng đệm khung (frame buffer). Vùng bộ nhớ này lưu trữ tập các giá trị cường độ sáng của toàn bộ các điểm trên màn hình và luôn luôn tồn tại một song ánh giữa mỗi điểm trên màn hình và mỗi phần tử trong vùng này.
- Để thay đổi các hình ảnh cần hiển thị, các giá trị tương ứng với vị trí và độ sáng phải được đặt vào vùng đệm khung.



- Để tạo ra các ảnh đen trắng, đơn giản chỉ cần lưu thông tin của mỗi pixel bằng 1 bit (các giá trị 0, 1 sẽ tượng trưng cho việc tắt (tối), bật (sáng) pixel trên màn hình). Trong trường hợp ảnh nhiều màu, người ta cần nhiều bit hơn, nếu thông tin của mỗi pixel được lưu bằng b bit, thì ta có thể có 2^b giá trị màu phân biệt cho pixel đó.

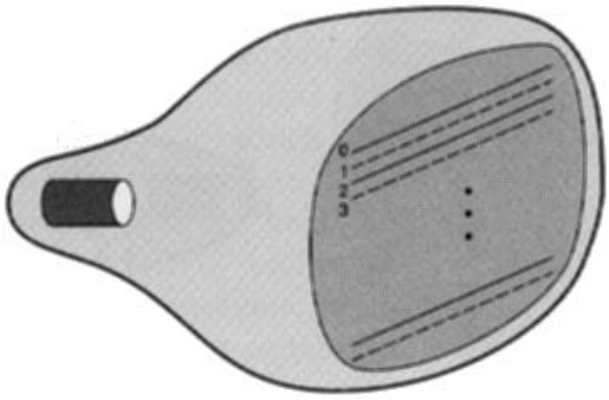
- Trong các màn hình màu, người ta định nghĩa tập các màu làm việc trong một bảng tra (LookUp Table - LUT). Mỗi phần tử của LUT định nghĩa một bộ ba giá trị R (Red), G (Green), B (Blue) mô tả một màu nào đó. Khi cần sử dụng một màu, ta chỉ cần chỉ định số thứ tự (index) tương ứng của màu đó trong LUT. Bảng LUT có thể được thay đổi bởi các ứng dụng và người lập trình có thể can thiệp điều khiển. Với cách làm này chúng ta có thể tiết kiệm không gian lưu trữ cho mỗi phần tử trong vùng đệm khung.

- Số phần tử của LUT được xác định từ số lượng các bits/pixel. Nếu mỗi phần tử của vùng đệm khung dùng b bits để lưu thông tin của một pixel, thì bảng LUT có 2^b phần tử. Nếu b=8, LUT sẽ có 2^8=256 phần tử, đó chính là số màu có thể được hiển thị cùng một lúc trên màn hình.

- Việc làm tươi trên màn hình dạng này được thực hiện ở tốc độ 60 đến 80 frame/giây. Đôi khi tốc độ làm tươi còn được biểu diễn bằng đơn vị Hertz (Hz – số chu kì/ giây), trong đó một chu kì tương ứng với một frame.

- Khi đạt đến cuối mỗi dòng quét, tia điện tử quay trở lại bên trái của màn hình để bắt đầu dòng quét kế tiếp. Việc quay trở lại phía trái màn hình sau khi làm tươi mỗi dòng quét được gọi là tia hồi ngang (horizontal retrace). Và tới cuối mỗi frame, tia điện tử (tia hồi dọc – vertical retrace) quay trở lại góc trên bên trái của màn hình để chuẩn bị bắt đầu frame kế tiếp.

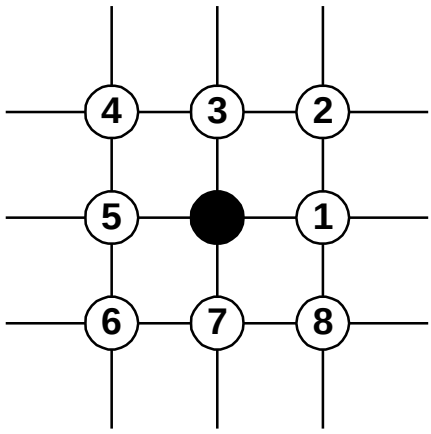
- Trong một số màn hình, mỗi frame được hiển thị thành hai giai đoạn sử dụng kĩ thuật làm tươi đan xen nhau (interlaced refesh). Ở giai đoạn đầu tiên, tia quét sẽ quét một số dòng từ trên xuống dưới, sau tia hồi dọc, các dòng còn lại sẽ được quét. Việc đan xen các dòng quét này cho phép chúng ta thấy được toàn màn hình hiển thị chỉ trong một nửa thời gian so với dùng để quét tất cả các dòng một lần từ trên xuống dưới. Kĩ thuật này thường được dùng cho loại màn hình có tốc độ làm tươi thấp.



Các thuật toán vẽ đường

Dẫn nhập

- Giả sử tọa độ các điểm nguyên sau khi xấp xỉ đối tượng thực lần lượt là $(x_i, y_i), i = 0,\ldots$. Đây là các điểm nguyên sẽ được hiển thị trên màn hình.
- Bài toán đặt ra là nếu biết được (x_i, y_i) là tọa độ nguyên xác định ở bước thứ i, điểm nguyên tiếp theo (x_{i+1}, y_{i+1}) sẽ được xác định như thế nào.
- Đối tượng hiển thị trên lưới nguyên được liền nét, các điểm mà (x_{i+1}, y_{i+1}) có thể chọn chỉ là một trong tám điểm được đánh số từ 1 đến 8 trong hình sau (điểm đen chính là (x_i, y_i)).Hay nói cách khác : $(x_{i+1}, y_{i+1}) = (x_i \pm 1, y_i \pm 1)$.

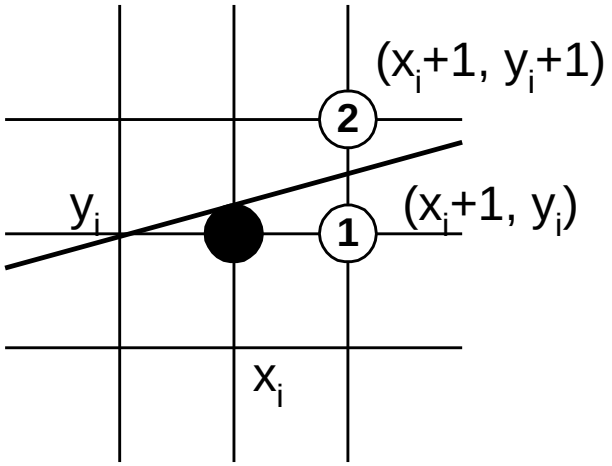


- Dáng điệu của đường sẽ cho ta gợi ý khi chọn một trong tám điểm trên. Cách chọn các điểm như thế nào sẽ tùy thuộc vào từng thuật toán trên cơ sở xem xét tới vấn đề tối ưu tốc độ.

Thuật toán vẽ đường thẳng

- Xét đoạn thẳng có hệ số góc $0 < m < 1$ và $Dx > 0$.
- Với các đoạn thẳng dạng này, nếu (x_i, y_i) là điểm đã xác định được ở bước thứ i (điểm màu đen) thì điểm cần chọn (x_{i+1}, y_{i+1}) ở bước thứ (i+1) sẽ là một trong hai trường hợp như hình vẽ sau :

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} \in \{y_i, y_i + 1\} \end{cases}$$



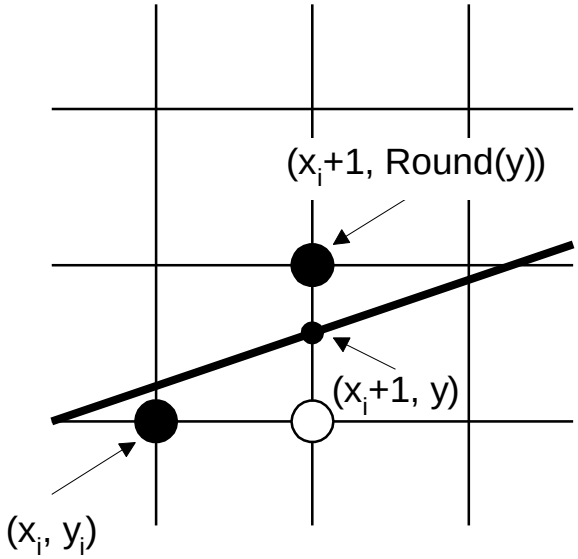
- Vấn đề còn lại, là cách chọn một trong hai điểm trên như thế nào để có thể tối ưu về mặt tốc độ.

Thuật toán DDA (Digital Differential Analyzer)

- Việc quyết định chọn y_{i+1} là y_i hay $y_i + 1$, dựa vào phương trình của đoạn thẳng $y = mx + b$. Nghĩa là, ta sẽ tính tọa độ của điểm $(x_i + 1, y)$ thuộc về đoạn thẳng thực. Tiếp đó, y_{i+1} sẽ là giá trị sau khi làm tròn giá trị tung độ y.

$$y = m(x_i + 1) + b$$

- Như vậy : $y_{i+1} = Round(y)$



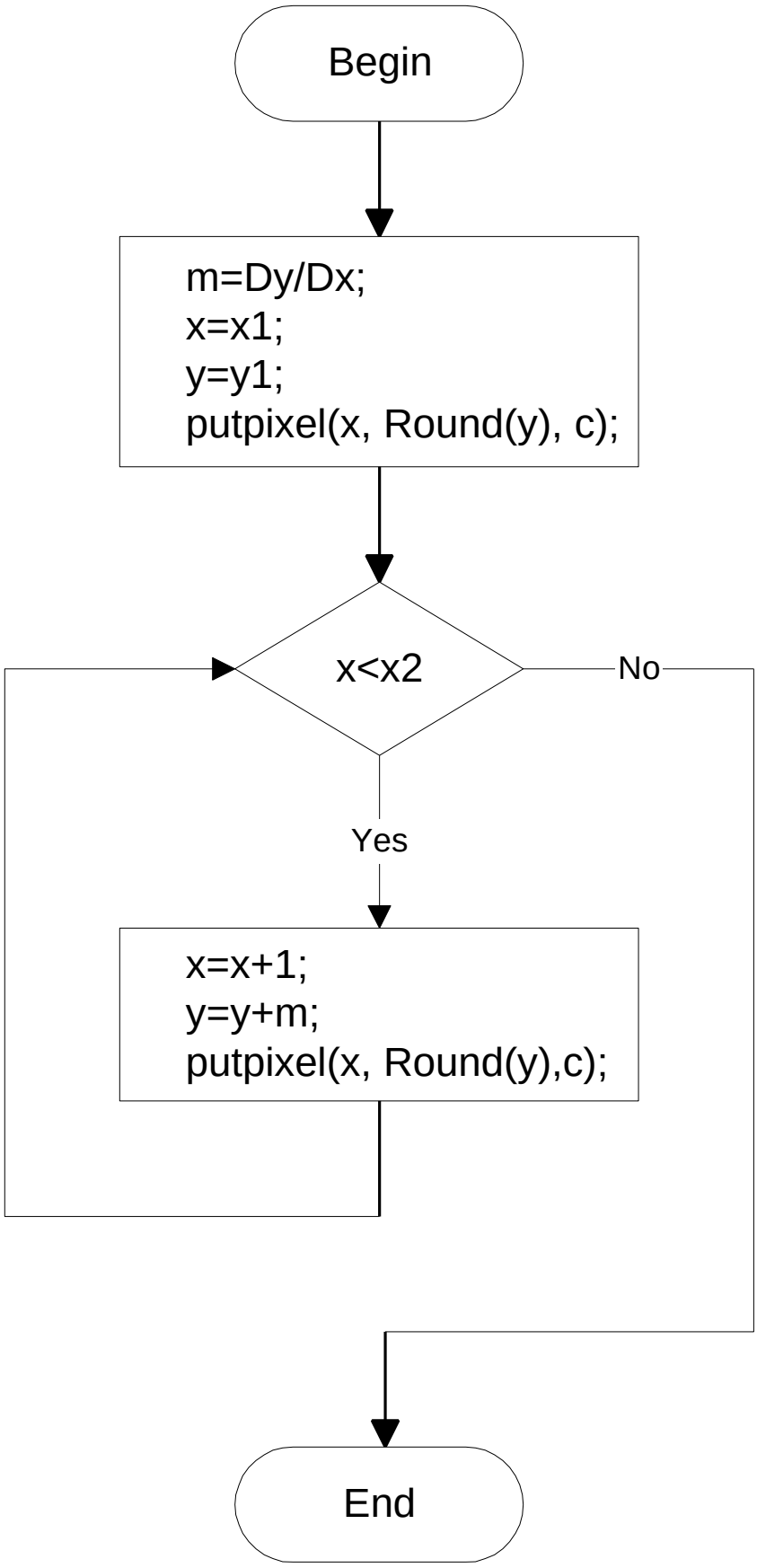
- Nếu tính trực tiếp giá trị thực y ở mỗi bước từ phương trình $y = mx + b$ thì phải cần một phép toán nhân và một phép toán cộng số thực. Để cải thiện tốc độ, người ta tính giá trị thực của y ở mỗi bước theo cách sau để khử phép tính nhân trên số thực :

- Nhận xét rằng : $y_{sau} = mx_{i+1} + b = m(x_i + 1) + b$

$$y_{trước} = mx_i + b$$

$$\Rightarrow y_{sau} = y_{trước} + m$$

Lưu đồ thuật toán DDA



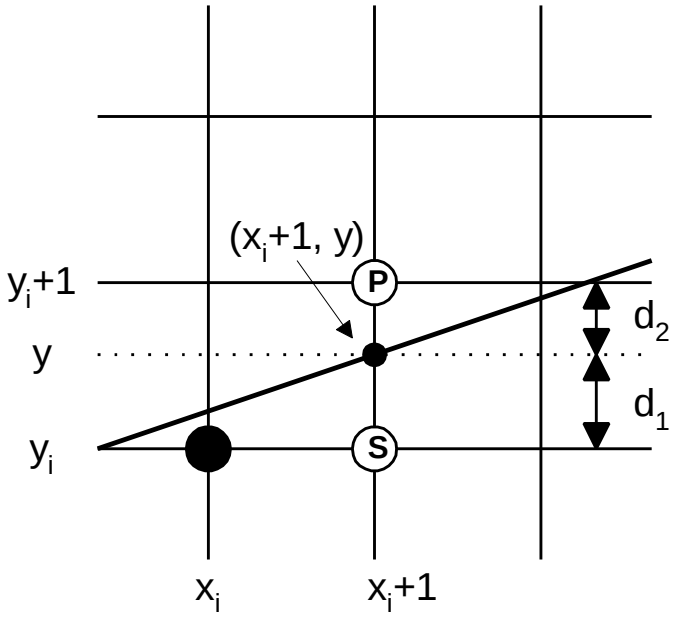
- Ví dụ : Cho A(12, 20) và B(22, 27), ta có m= 0.7

i	x_i	y_i	y
0	12	20	20
1	13	21	20.7
2	14	21	21.4
3	15	22	22.1
4	16		
5	17		
6	18		
7	19		
8	20		
9	21		
10	22	27	

- Cài đặt minh họa thuật toán DDA

```
#define Round(a)  int(a+0.5)
int Color = GREEN;

void LineDDA (int x1, int y1, int x2, int y2)
{
    int x = x1;
    float y = y1;
    float m = float(y2-y1)/(x2-x1);

    putpixel(x, Round(y), Color);
    for(int i=x1; i<x2; i++)
    {
        x++;
        y +=m;
        putpixel(x, Round(y), Color);
    }
} // LineDDA
```

Thuật toán Bresenham



- Gọi $(x_i + 1, y)$ là điểm thuộc đoạn thẳng. Ta có: $y = m(x_i + 1) + b$.

  $d_1 = y - y_i$

- Đặt $d_2 = (y_i + 1) - y$

- Xét tất cả các vị trí tương đối của y so với y_i và $y_i + 1$, việc chọn điểm (x_{i+1}, y_{i+1}) là S hay P phụ thuộc vào việc so sánh d_1 và d_2 hay dấu của $d_1 - d_2$:

  - Nếu $d_1 - d_2 < 0$, ta sẽ chọn điểm S, tức là $y_{i+1} = y_i$.
  - Ngược lại, nếu $d_1 - d_2 \geq 0$, ta sẽ chọn điểm P, tức là $y_{i+1} = y_i + 1$

- Xét $p_i = Dx(d_1 - d_2) = Dx(2y - 2y_i - 1)$

  $\Rightarrow p_i = Dx[2(m(x_i + 1) + b) - 2y_i - 1]$

- Thay $m = \frac{Dy}{Dx}$ vào phương trình trên ta được : $p_i = 2Dyx_i - 2Dxy_i + c$, với $c = 2Dy + (2b-1)Dx$.
- Nhận xét rằng nếu tại bước thứ i ta xác định được dấu của p_i thì xem như ta xác định được điểm cần chọn ở bước (i+1).
- Ta có :

$$p_{i+1} - p_i = (2Dyx_{i+1} - 2Dxy_{i+1} + c) - (2Dyx_i - 2Dxy_i + c)$$

$$\Leftrightarrow p_{i+1} - p_i = 2Dy(x_{i+1} - x_i) - 2Dx(y_{i+1} - y_i)$$

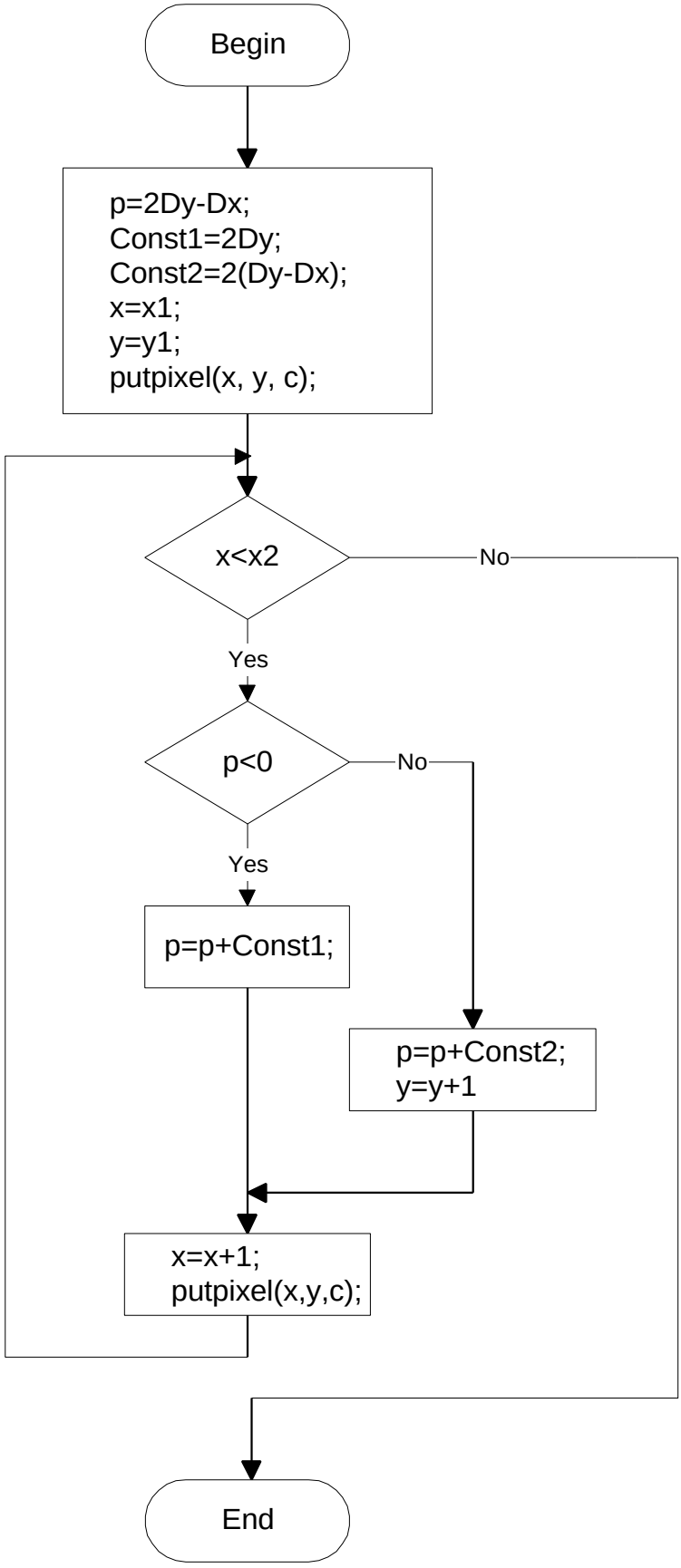
$$\Leftrightarrow p_{i+1} - p_i = 2Dy - 2Dx(y_{i+1} - y_i), \text{ do } x_{i+1} = x_i + 1$$

- Từ đây ta có thể suy ra cách tính p_{i+1} từ p_i như sau :
  - Nếu $p_i < 0$ thì $p_{i+1} = p_i + 2Dy$ do ta chọn $y_{i+1} = y_i$.
  - Ngược lại, nếu $p_i \geq 0$, thì $p_{i+1} = p_i + 2Dy - 2Dx$, do ta chọn $y_{i+1} = y_i + 1$.
- Giá trị p_0 được tính từ điểm vẽ đầu tiên (x_0, y_0) theo công thức :

$$p_0 = 2Dyx_0 - 2Dxy_0 + c = 2Dyx_0 - 2Dxy_0 + 2Dy - (2b-1)Dx$$

- Do (x_0, y_0) là điểm nguyên thuộc về đoạn thẳng nên ta có $y_0 = mx_0 + b = \frac{Dy}{Dx}x_0 + b$. Thế vào phương trình trên ta suy ra : $p_0 = 2Dy - Dx$.

Lưu đồ thuật toán Bresenham



- Ví dụ : Cho A(12, 20) và B(22, 27),
- Ta có
  - Dx = 22-12 = 10, Dy=27-20=7
  - Const1 = 2Dy = 14, Const2 = 2(Dy – Dx) = -6
  - p_0 = 2Dy – Dx = 14-10 = 4

i	x_i	y_i	p_i
0	12	20	4
1	13	21	-2
2	14	21	12
3	15	22	6
4	16	23	0
5	17	24	-6
6	18	24	8
7	19	25	2
8	20	26	-4
9	21	26	10
10	22	27	4

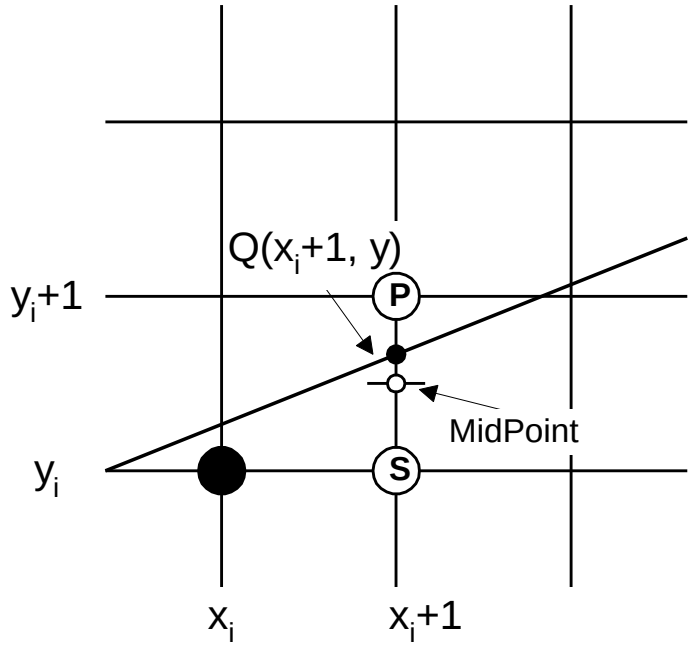
- Nhận xét
  - Thuật toán Bresenham chỉ làm việc trên số nguyên và các thao tác trên số nguyên chỉ là phép cộng và phép dịch bit (phép nhân 2) điều này là một cải tiến làm tăng tốc độ đáng kể so với thuật toán DDA. Ý tưởng chính của thuật toán nằm ở chỗ xét dấu p_i để quyết định điểm kế tiếp, và sử dụng công thức truy hồi $p_{i+1} - p_i$ để tính p_i bằng các phép toán đơn giản trên số nguyên.
  - Thuật toán này cho kết quả tương tự như thuật toán DDA.

- Cài đặt minh họa thuật toán Bresenham

```
void LineBres (int x1, int y1, int x2, int y2)
{
    int Dx, Dy, p, Const1, Const2;
    int x, y;

    Dx     = x2 - x1;
    Dy     = y2 - y1;
    p   = 2*Dy - Dx; // Dy <<1 - Dx
    Const1 = 2*Dy; // Dy <<1
    Const2 = 2*(Dy-Dx); // (Dy-Dx) <<1
    x = x1;
    y = y1;
    putpixel(x, y, Color);
    for(i=x1; i<x2; i++)
    {
        if (p<0)
            p += Const1;
        else
        {
            p += Const2;
            y++;
        }
        x++;
        putpixel(x, y, Color);
    }
} // LineBres
```

Thuật toán MidPoint

- Thuật toán MidPoint đưa ra cách chọn y_{i+1} là y_i hay $y_i + 1$ bằng cách so sánh điểm thực $Q(x_i + 1, y)$ với điểm MidPoint là trung điểm của S và P. Ta có :
  - Nếu điểm Q nằm dưới điểm MidPoint, ta chọn S.
  - Nếu điểm Q nằm trên điểm MidPoint ta chọn P.



- Ta có dạng tổng quát của phương trình đường thẳng :

$Ax + By + C = 0$ với $A = y_2 - y_1, B = -(x_2 - x_1), C = x_2 y_1 - x_1 y_2$

- Đặt $F(x, y) = Ax + By + C$, ta có nhận xét :

$$F(x,y)\begin{cases} < 0, \text{nếu } (x,y) \text{ nằm phía trên đường thẳng} \\ = 0, \text{nếu } (x,y) \text{ thuộc về đường thẳng} \\ > 0, \text{nếu } (x,y) \text{ nằm phía dưới đường thẳng.} \end{cases}$$

- Lúc này việc chọn các điểm S, P ở trên được đưa về việc xét dấu của $p_i = 2F(\text{MidPoint}) = 2F\left(x_i + 1, y_i + \frac{1}{2}\right)$.
  - Nếu $p_i < 0$, điểm MidPoint nằm phía trên đoạn thẳng. Lúc này điểm thực Q nằm dưới điểm MidPoint nên ta chọn S, tức là $y_{i+1} = y_i$.
  - Ngược lại, nếu $p_i \geq 0$, điểm MidPoint nằm phía dưới đoạn thẳng. Lúc này điểm thực Q nằm trên điểm MidPoint nên ta chọn P, tức là $y_{i+1} = y_i + 1$.
- Mặt khác :

$$p_{i+1} - p_i = 2F\left(x_{i+1} + 1, y_{i+1} + \frac{1}{2}\right) - 2F\left(x_i + 1, y_i + \frac{1}{2}\right)$$

$$\Leftrightarrow p_{i+1} - p_i = 2\left[A(x_{i+1} + 1) + B\left(y_{i+1} + \frac{1}{2}\right) + C\right] - 2\left[A(x_i + 1) + B\left(y_i + \frac{1}{2}\right) + C\right]$$

$$\Leftrightarrow p_{i+1} - p_i = 2A + 2B(y_{i+1} - y_i) = 2Dy - 2Dx(y_{i+1} - y_i)$$

- Như vậy :
  - $p_{i+1} = p_i + 2Dy$, nếu $p_i < 0$ do ta chọn $y_{i+1} = y_i$.
  - $p_{i+1} = p_i + 2Dy - 2Dx$, nếu $p_i \geq 0$ do ta chọn $y_{i+1} = y_i + 1$.
- Ta tính giá trị p_0 ứng với điểm ban đầu (x_0, y_0), với nhận xét rằng (x_0, y_0) là điểm thuộc về đoạn thẳng, tức là có : $Ax_0 + By_0 + C = 0$

$$p_0 = 2F\left(x_0 + 1, y_0 + \frac{1}{2}\right) = 2\left[A(x_0 + 1) + B\left(y_0 + \frac{1}{2}\right) + C\right]$$

$$\Rightarrow p_0 = 2(Ax_0 + By_0 + C) + 2A + B = 2A + B = 2Dy - Dx$$

Câu hỏi kiểm tra

- Xét thuật toán Bresenham, với cách đặt d_1 và d_2 như trên, có khi nào d_1 hay d_2 âm hay không ? Cho ví dụ minh họa.

- Tại sao phải so sánh giá trị p_i với 0 trong các thuật toán MidPoint và Bresenham, bản chất của việc so sánh này là gì ?

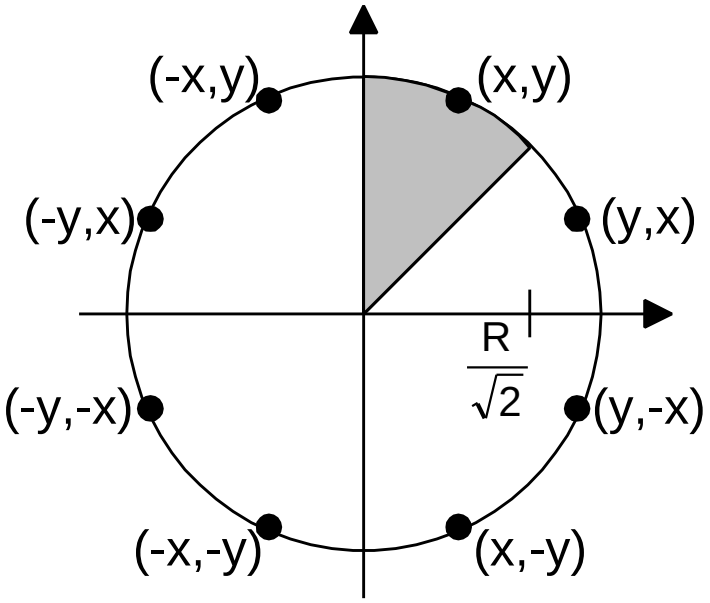
- Tại sao phải nhân F(MidPoint) với 2 khi gán cho p_i theo công thức p_i=2*F(MidPoint) ?

- Cài đặt thuật toán cho trường hợp $0 \le m \le 1$, Dx<0. Ta sử dụng thuật toán với trường hợp $0 \le m \le 1$, Dx>0 đã cài đặt cộng thêm một số thay đổi sau :
  - Thay biểu thức x=x+1 bằng x=x-1 và y=y+1 bằng y=y-1 vì trong trường hợp này x và y đều giảm dần.
  - Nhận xét rằng khi p<0 ta gán p=p+Const1, như vậy để hướng đến sự cân bằng Const1 phải là một giá trị dương. Tương tự như vậy, khi p≥0 ta gán p=p+Const2, Const2 phải là giá trị âm.
  - Từ nhận xét trên, trong các công thức ta sẽ thay Dx bằng abs(Dx), Dy bằng abs(Dy).
- Mở rộng thuật toán trên để vẽ đoạn thẳng trong trường hợp m bất kì.
  - Trường hợp đặc biệt $m=\infty$: Đoạn thẳng song song trục tung nên rất đơn giản khi vẽ.
  - Trường hợp $-1 \le m \le 1$: Sử dụng các công thức của thuật toán vẽ trong trường hợp $0 \le m \le 1$, Dx>0 và thay đổi một số điểm sau :
    - Nếu Dx<0 thì bước nhảy của x sẽ thay bằng −1. Tương tự nếu Dy<0, bước nhảy của y cũng sẽ là −1.
    - Thay Dx bằng abs(Dx), Dy=abs(Dy) trong tất cả các công thức có chứa Dx, Dy.
  - Trường hợp $m \le -1$ hay $m \ge 1$:
    - Thay đổi vai trò của x và y, nghĩa là thay x bằng y, y bằng x, Dx bằng Dy, Dy bằng Dx trong tất cả các công thức.
    - Thực hiện nguyên tắc về bước nhảy, thay đổi công thức Dx, Dy như trong trường hợp $-1 \le m \le 1$

Vẽ đường tròn bằng thuật toán MidPoint

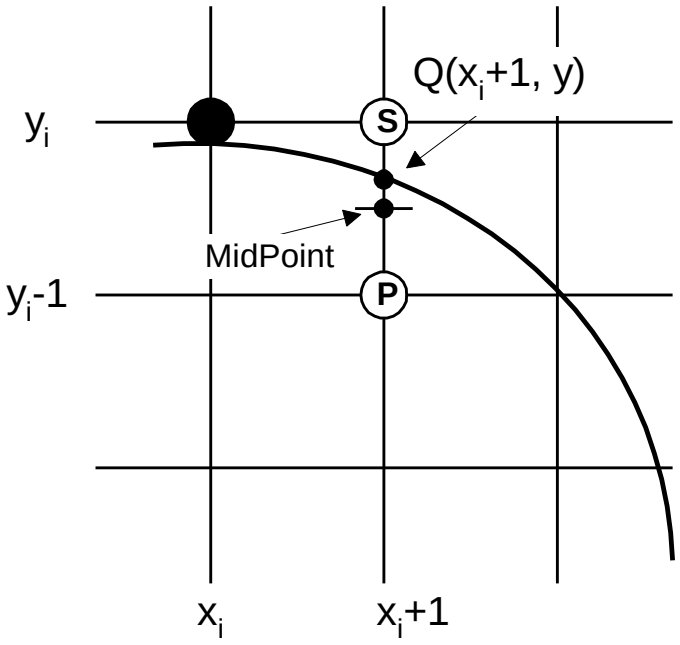
- Do tính đối xứng của đường tròn (C) nên ta chỉ cần vẽ cung ($C^{1/8}$) là cung 1/8 đường tròn, sau đó lấy đối xứng. Cung ($C^{1/8}$) được mô tả như sau (cung của phần tô xám trong hình vẽ) :

$$\begin{cases} 0 \le x \le R\dfrac{\sqrt{2}}{2} \\ R\dfrac{\sqrt{2}}{2} \le y \le R \end{cases}$$



- Như vậy nếu có (x, y) ∈ ($C^{1/8}$) thì các điểm : (y, x), (y,-x), (x,-y), (-x,-y), (-y,-x), (-y,x), (-x,y) sẽ thuộc (C).

- Chọn điểm bắt đầu để vẽ là điểm (0,R).
- Dựa vào hình vẽ, nếu (x_i, y_i) là điểm nguyên đã tìm được ở bước thứ i, thì điểm (x_{i+1}, y_{i+1}) ở bước thứ (i+1) là sự lựa chọn giữa S và P.
- Như vậy : $\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} \in \{y_i, y_i - 1\} \end{cases}$



- Đặt $F(x,y) = x^2 + y^2 - R^2$, ta có :

$$F(x,y)\begin{cases} < 0, \text{nếu } (x,y) \text{ nằm trong đường tròn} \\ = 0, \text{nếu } (x,y) \text{ nằm trên đường tròn} \\ > 0, \text{nếu } (x,y) \text{ nằm ngoài đường tròn.} \end{cases}$$

- Xét $p_i = F(\text{MidPoint}) = F\left(x_i + 1, y_i - \frac{1}{2}\right)$. Ta có :
  - Nếu $p_i < 0$, điểm MidPoint nằm trong đường tròn. Lúc này điểm thực Q gần S hơn nên ta chọn S, tức là $y_{i+1} = y_i$.
  - Ngược lại, nếu $p_i \geq 0$, điểm MidPoint nằm ngoài đường tròn. Lúc này điểm thực Q gần P hơn nên ta chọn P, tức là $y_{i+1} = y_i - 1$.
- Mặt khác :

$$p_{i+1} - p_i = F\left(x_{i+1} + 1, y_{i+1} - \frac{1}{2}\right) - F\left(x_i + 1, y_i - \frac{1}{2}\right)$$

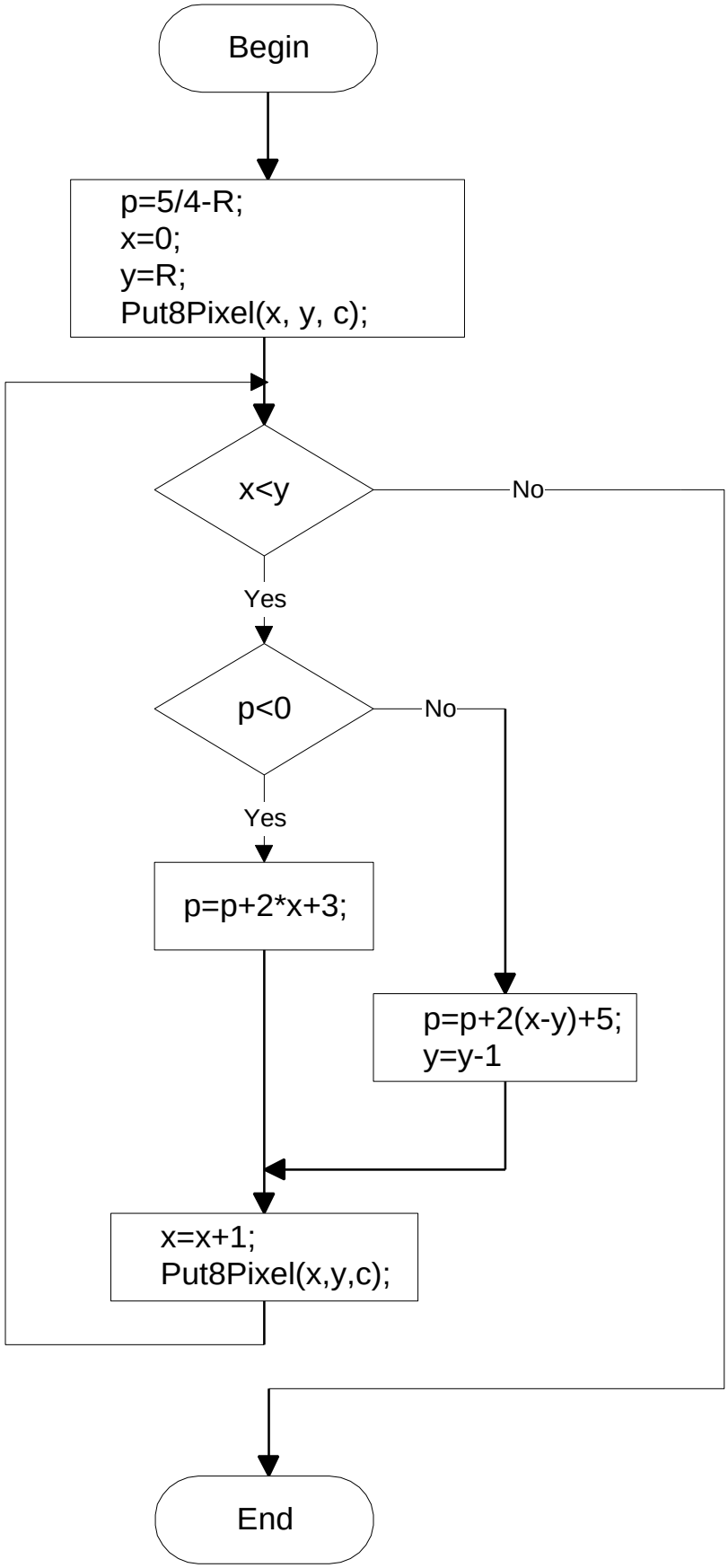
$$\Leftrightarrow p_{i+1} - p_i = \left[(x_{i+1} + 1)^2 + \left(y_{i+1} - \frac{1}{2}\right)^2 - R^2\right] - \left[(x_i + 1)^2 + \left(y_i - \frac{1}{2}\right)^2 - R^2\right]$$

$$\Leftrightarrow p_{i+1} - p_i = 2x_i + 3 + \left(y_{i+1}^2 - y_i^2\right) - \left(y_{i+1} - y_i\right)$$

- Vậy :
  - $p_{i+1} = p_i + 2x_i + 3$, nếu $p_i < 0$ do ta chọn $y_{i+1} = y_i$.
  - $p_{i+1} = p_i + 2x_i - 2y_i + 5$, nếu $p_i \geq 0$ do ta chọn $y_{i+1} = y_i - 1$.
- p_0 ứng với điểm ban đầu $(x_0, y_0) = (0, R)$.

$$p_0 = F\left(x_0 + 1, y_0 - \frac{1}{2}\right) = F\left(1, R - \frac{1}{2}\right) = \frac{5}{4} - R$$

Lưu đồ thuật toán MidPoint vẽ đường tròn



Cài đặt minh họa thuật toán MidPoint vẽ đường tròn

```
void CircleMidPoint (int R)
{
    int x, y;

    x = 0;
    y = R;
    Put8Pixel(x, y);
    p = 1 - R; // 5/4-R
    while (x < y)
    {
        if (p < 0)
            p += 2*x + 3;
        else
        {
            p += 2*(x -y) + 5;
            y--;
        }

        x++;
        Put8Pixel(x, y);
    }

} // CircleMidPoint
```

- Ví dụ : Vẽ đường tròn tâm I(0,0), bán kính R=15.

i	x_i	y_I	p_i		Delta1	Delta2
0	0	15	-14	1-15	3	-25
1	1	15	-11	-14+2*(0)+3	5	-23
2	2	15	-6	-11+2*(1)+3	7	-21
3	3	15	1	-6+2*(2)+3	9	-19
4	4	14	-18	1+2*(3-15)+5	11	-15
5	5	14	-7	-18+2*(4)+3	13	-13
6	6	14	6	-7+2*(5)+3	15	-11
7	7	13	-5	6+2(6-14)+5	17	-7
8	8	13	12	-5+2(7)+3	19	-5
9	9	12	7	12+2(8-13)+5	21	-1
10	10	11	6	7+2(9-12)+5	23	3
11	11	10	9	6+2(10-11)+5	25	7

Nhận xét :

- Nếu đặt Delta1 = 2*x+3, Delta2 = 2*(x-y)+5 thì
  - Do mỗi bước đều tăng x nên sau mỗi lần lặp giá trị Delta1 luôn tăng 2.
  - Do y bị giảm 1 khi gặp p≥0 và giữ nguyên giá trị trong trường hợp ngược lại nên nếu lần lặp trước giá trị p≥0 thì giá trị Delta2 sẽ được tăng 4 và nếu lần lặp trước giá trị p<0 thì giá trị Delta2 sẽ được tăng 2 mà thôi.
- Hãy tối ưu hóa cài đặt thuật toán MidPoint vẽ đường tròn từ nhận xét trên.

BỘ GIAO THÔNG VẬN TẢI

TRƯỜNG ĐẠI HỌC HÀNG HẢI

BỘ MÔN: KHOA HỌC MÁY TÍNH

KHOA: CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG

ĐỒ HỌA MÁY TÍNH

TÊN HỌC PHẦN : **ĐỒ HỌA MÁY TÍNH**

MÃ HỌC PHẦN : **17211**

TRÌNH ĐỘ ĐÀO TẠO : **ĐẠI HỌC CHÍNH QUY**

DÙNG CHO SV NGÀNH : **CÔNG NGHỆ THÔNG TIN**

HẢI PHÒNG - 2010

MỤC LỤC

Tên học phần: Đồ họa máy tính **Loại học phần:** 4

Bộ môn phụ trách giảng dạy: Khoa học Máy tính **Khoa phụ trách:** CNTT

Mã học phần: 17211 **Tổng số TC**: 4

TS tiết	Lý thuyết	Thực hành/Xemina	Tự học	Bài tập lớn	Đồ án môn học
75	45	15	0	15	0

Điều kiện tiên quyết:

Sinh viên phải học xong các học phần sau mới được đăng ký học phần này:

Toán cao cấp, Toán rời rạc, Kỹ thuật lập trình, Tin học đại cương, Cấu trúc dữ liệu, Giải thuật

Mục tiêu của học phần:

Cung cấp cho sinh viên kiến thức và rèn luyện kỹ năng lập trình đồ họa máy tính đồ họa máy tính, các phương pháp dựng hình, xử lý hình ảnh cơ bản trong đồ họa.

Nội dung chủ yếu

- Những vấn đề cơ bản về đồ họa máy tính
- Các thuật toán cơ bản trong đồ họa máy tính.
- Đồ họa 2D/3D
- Các phương pháp biến hình
- Các phương pháp tạo bóng

Nội dung chi tiết của học phần:

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**TH/Xemina**	**BT**	**KT**
Chương I : Giới thiệu đồ họa máy tính.	3	3	0		
1.1. Giới thiệu về đồ họa máy tính 1.2. Tổng quan về một hệ đồ họa					
Chương II : Các đối tượng đồ họa cơ bản	16	9	3	3	1
2.1. Giới thiệu về các đối tượng đồ họa cơ sở 2.1.1. Các đối tượng đồ họa cơ sở : điểm,					

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**TH/Xemina**	**BT**	**KT**
đường, vùng tô, văn bản,... 2.1.2. Các thuộc tính của các đối tượng đồ họa cơ sở 2.2. Các thuật toán vẽ điểm, đường 2.2.1. Vẽ đường thẳng 2.2.2. Vẽ đường tròn, ellipse 2.3. Các thuật toán tô màu 2.3.1. Thuật toán tô màu dựa theo đường biên 2.3.2. Thuật toán tô màu theo dòng quét 2.4. Bài tập áp dụng					
Chương III: Các phép biến đổi đồ họa 2 chiều.	10	8	2		
3.1. Các phép biến đổi cơ sở 3.1.1 Giới thiệu về phép biến đổi, biểu diễn ma trận của phép biến đổi, phép biến đổi affine 3.1.2. Phép tịnh tiến 3.1.3. Phép biến đổi tỉ lệ 3.1.4. Phép quay quanh gốc tọa độ 3.1.5. Biểu diễn ma trận của các phép biến đổi. Hệ tọa độ thuần nhất 3.2. Kết hợp các phép biến đổi 3.2.1. Kết hợp hai hay nhiều phép tịnh tiến, phép biến đổi tỉ lệ, phép quay. 3.2.2. Phép biến đổi tỉ lệ với tâm tỉ lệ bất kì. 3.2.3. Phép quay quanh tâm là một điểm bất kì.					

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**TH/Xemina**	**BT**	**KT**
3.3. Một số phép biến đổi khác 3.3.1. Phép biến dạng 3.3.2. Phép đối xứng 3.3.3. Tính chất của phép biến đổi affine 3.4. Phép biến đổi giữa các hệ tọa độ 3.5. Bài tập áp dụng					
Chương IV: Phép quan sát hai chiều	12	9	2		1
4.1. Cửa sổ, Vùng quan sát 4.2. Phép biến đổi từ cửa sổ - đến – vùng quan sát 4.3. Phép cắt xén 2 chiều 4.3.1 Giải thuật Cohen – Suntherland 4.3.2 Giải thuật chia trung điểm 4.3.3 Giải thuật Liang – Barsky 4.4. Bài tập áp dụng					
Chương V: Đồ họa ba chiều	22	10	5	6	1
5.1. Tổng quan về đồ họa ba chiều 5.1.1. Quy trình hiển thị đồ họa ba chiều 5.1.2. Mô hình hóa đối tượng 5.2. Biểu diễn đối tượng ba chiều 5.2.1. Biểu diễn mặt đa giác 5.2.2. Đường cong và mặt cong , đường cong và mặt cong Bezier, B-Spline 5.3. Các phép biến đổi hình học ba chiều 5.3.1. Phép tịnh tiến 5.3.2. Phép quay quanh một trục					

TÊN CHƯƠNG MỤC	PHÂN PHỐI SỐ TIẾT				
	TS	**LT**	**TH/Xemina**	**BT**	**KT**
5.3.3. Phép biến đổi tỉ lệ 5.3.4. Kết hợp các phép biến đổi 5.3.5. Các phép chuyển đổi giữa các hệ tọa độ 5.4. Bài tập áp dụng					
Chương VI: Quan sát ba chiều	15	6	3	6	
6.1. Các phép chiếu 6.2. Điểm tụ 6.3. Loại bỏ mặt khuất 6.4. Bài tập áp dụng					

Nhiệm vụ của sinh viên :

Tham dự các buổi thuyết trình của giáo viên, tự học, tự làm bài tập do giáo viên giao, tham dự các bài kiểm tra định kỳ và cuối kỳ.

Tài liệu học tập :

- Đặng Văn Đức. *Kỹ thuật đồ họa máy tính*. Viện Công nghệ thông tin. 2002.
- Donald Hearn, M. Pauline Baker, *Computer Graphics*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1986.
- F.S.Hill, *Computer graphics*, 1990

Hình thức và tiêu chuẩn đánh giá sinh viên:

- Hình thức thi cuối kỳ : Thi viết.
- Sinh viên phải đảm bảo các điều kiện theo Quy chế của Nhà trường và của Bộ

Thang điểm: Thang điểm chữ A, B, C, D, F
Điểm đánh giá học phần: Z = 0,4X + 0,6Y.
Bài giảng này là tài liệu **chính thức và thống nhất** của Bộ môn Khoa học máy tính, Khoa Công nghệ thông tin và được dùng để giảng dạy cho sinh viên.

Ngày phê duyệt: / /2010

Trưởng Bộ môn: (ký và ghi rõ họ tên)

CHƯƠNG I GIỚI THIỆU ĐỒ HỌA MÁY TÍNH

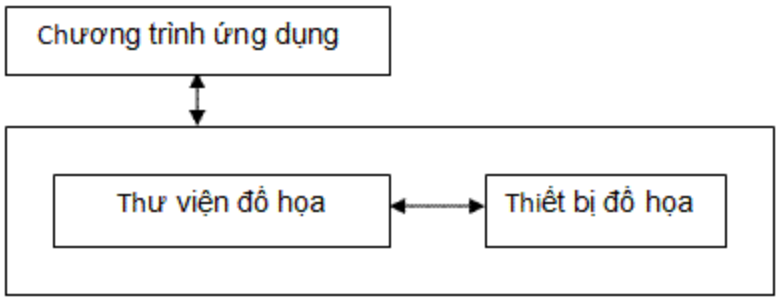
1.1 Giới thiệu về đồ họa máy tính

Ngày nay, đồ họa máy tính được ứng dụng rộng rãi trong ngành công nghệ thông tin. Khó mà tìm được một ứng dụng thương mại của công nghệ thông tin không sử dụng một thành phần nào đó của hệ đồ họa máy tính. Đồ họa máy tính ở vị trí quan trọng trong lĩnh vực thiết kế và giao tiếp kỹ thuật. Nó là cơ sở để chuyển đổi các giải pháp tính toán số sang thể hiện hình ảnh tự nhiên cho thiết kế kỹ thuật hay muốn sáng tỏ một vấn đề phức tạp. Đồ họa máy tính biểu diễn được hình ảnh đối tượng, quan hệ, dữ liệu, vị trí … Đồ họa máy tính còn có chức năng mô tả kích thước của đối tượng, phân tích dữ liệu. Trong những thập niên cuối thế kỷ 20, sự phát triển mạnh của những hệ đồ họa như CAD/CAM đã trở thành chuẩn công nghiệp và trường học.

1.2 Tổng quan về một hệ đồ họa

Mục tiêu của đồ họa máy tính có chức năng tạo ra và thao tác các hình ảnh đồ họa, nên nó phải có khả năng tạo ra và hiệu chỉnh các hình ảnh bằng các tương tác và đáp ứng. Các ứng dụng đồ họa đưa ra các chỉ dẫn thuật ngữ theo yêu cầu đồ họa người dùng. Thư viện đồ họa thực hiện tương tác, làm cầu nối cho giao tiếp giữa người dùng và các thiết bị vật lý đơn giản đi.

Hình 1.1 Hệ thống đồ họa



Một trong các yêu cầu chính của một hệ thống đồ họa là các ứng dụng, áp dụng cho nhiều hệ thiết bị vật lý, phải được phát triển không phụ thuộc vào phần cúng. Để có được điều đó, phải có tiêu chuẩn hóa cho môi trường đồ họa ở mức chức năng, bằng việc cung cấp sự độc lập thiết bị và ngôn ngữ lập trình.

Sự độc lập với thiết bị cho phép các chương trình ứng dụng đồ họa chạy trên các dạng phần cứng khác nhau. Nó được thực hiện thông qua thiết bị nhập xuất logic cung cấp cho phần mềm ứng dụng thông qua thư viện đồ họa và ánh xạ thiết bị vật lý cụ thể.

Cho tới nay, có những tiêu chuẩn đồ họa đã được phát triển trong nhiều năm, bao gồm: GKS(Graphics Kernel System – 1985), được phát triển riêng cho các thiết bị nhập xuất 2 chiều. GKS-3D bổ sung thêm khả năng lập trình 3 chiều. PHIGS (Programmer's Hierarchical Graphics System – 1984) hay PHIGS+ bao gồm khả năng lập trình không gian , tạo thành thao tác dữ liệu đồ họa phức tạp …

Các tiêu chuẩn đồ họa thực tế là kết quả của việc chấp nhận trong công nghiệp các giao diện đặc trưng, được đề xuất bởi nhiều công ty và không nêu ra trong các tiêu chuẩn chính thức. Được nhắc đến trong số này là hệ X-Windows, cung cấp một loạt các chức năng nhập và thao tác đồ họa 2 chiều. Sự mở rộng được bắt đầu vào giữa những năm 80 là hệ X-Windows 3 chiều.

Để đảm bảo sự linh hoạt, các tiêu chuẩn đồ họa thiết lập cho ứng dụng các thay đổi tối thiểu, cho

phép nó định địa chỉ các thiết bị nhập xuất khác nhau. Khởi đầu, người lập trình tạo ra một hệ thống tọa độ mô hình, mô tả đối tượng gọi là hệ thống tọa độ thực. Tiếp theo, là hệ tọa độ tiêu chuẩn và hệ tọa độ thiết bị. Chương trình ứng dụng sẽ giao tiếp với hệ tọa độ chuẩn theo cách thức phù hợp, không quan tâm đến thiết bị xuất được dùng. Do đó, tạo ra sự độc lập với thiết bị trong việc tạo ra hình ảnh của đối tượng.

1.2.1 Hệ tọa độ thực, hệ tọa độ thiết bị và hệ tọa độ chuẩn

Trong lĩnh vực kỹ thuật đồ họa, chúng ta phải hiểu được rằng thực chất của đồ họa là làm thế nào để có thể mô tả và biến đổi được các đối tượng trong thế giới thực trên máy tính. Bởi vì, các đối tượng trong thế giới thực được mô tả bằng tọa độ thực. Trong khi đó, hệ tọa độ thiết bị lại sử dụng hệ tọa độ nguyên để hiển thị các hình ảnh. Đây chính là vấn đề cơ bản cần giải quyết. Ngoài ra, còn có một khó khăn khác là với các thiết bị khác nhau thì có các định nghĩa khác nhau. Do đó, cần có một phương pháp chuyển đổi tương ứng giữa các hệ tọa độ và đối tượng phải được định nghĩa bởi các thành phần đơn giản như thế nào để có thể mô tả gần đúng với hình ảnh thực bên ngoài.

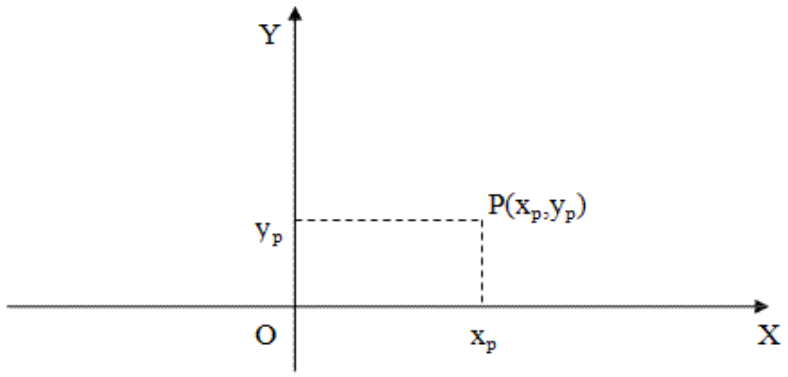
a.Hệ tọa độ thực:

Một trong những hệ tọa độ thực thường được dùng để mô tả các đối tượng trong thế giới thực là hệ tọa độ Descartes. Với hệ tọa độ này, mỗi điểm P được biểu diễn bằng một cặp tọa độ (x_p,y_p) với $x_p, y_p \in R$ (xem hình 1.1).

Hình 2.1 Hệ tọa độ thực

. Ox : gọi là trục hoành; Oy : gọi là trục tung.

. x_p : hoành độ điểm P; y_p : tung độ điểm P.



b. Hệ tọa độ thiết bị:

Hệ tọa độ thiết bị (device coordinates) được dùng cho một thiết bị xuất cụ thể nào đó, ví dụ như máy in, màn hình,.. Trong hệ tọa độ thiết bị thì các điểm cũng được mô tả bởi cặp tọa độ (x,y). Tuy nhiên, khác với hệ tọa độ thực là x, y $\in$ N. Điều này có nghĩa là các điểm trong hệ tọa độ thực được định nghĩa liên tục, còn các điểm trong hệ tọa độ thiết bị là rời rạc. Ngoài ra, các tọa độ x, y của hệ tọa độ thiết bị chỉ biểu diễn được trong một giới hạn nào đó của N.

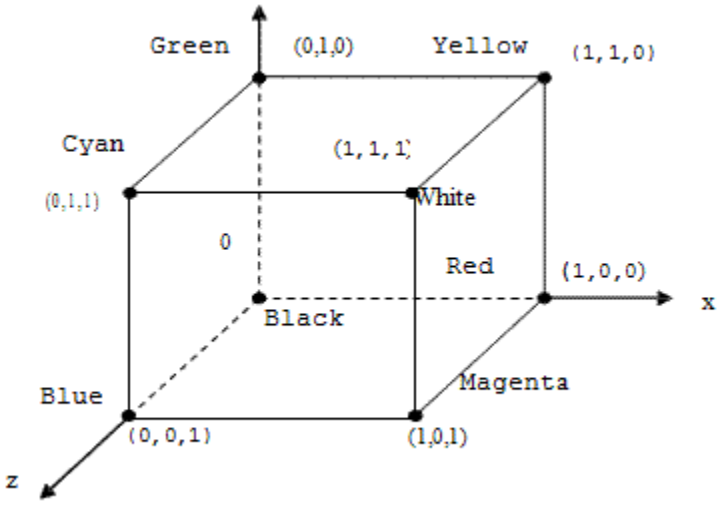
c. Hệ tọa độ thiết bị chuẩn:

Do cách định nghĩa các hệ tọa độ thiết bị khác nhau nên một hình ảnh hiển thị được trên thiết bị này là chính xác thì chưa chắc hiển thị chính xác trên thiết bị khác. Người ta xây dựng một hệ tọa độ thiết bị chuẩn đại diện chung cho tất cả các thiết bị để có thể mô tả các hình ảnh mà không phụ thuộc vào bất kỳ thiết bị nào. Trong hệ tọa độ chuẩn, các tọa độ x, y sẽ được gán các giá trị trong đoạn từ [0,1]. Như vậy, vùng không gian của hệ tọa độ chuẩn chính là hình vuông đơn vị có góc trái dưới (0, 0) và góc phải trên là (1, 1).

1.2.2 Mô hình màu

Màu sắc được sử dụng trong các ứng dụng đồ họa máy tính để giúp người dùng hiểu rõ về đối tượng hình học. Các màn hình đồ họa dựa sử dụng các màu sắc chromatic. Chúng dựa trên lý thuyết về bộ não người là màu sắc ánh sáng được tiếp nhận như sự phối hợp từ 3 màu là đỏ (red), xanh lá cây (green), và xanh dương (blue). Nói chung, màu được mô tả bằng 3 thuộc tính là màu sắc (hue), độ bão hòa (saturation), và độ sáng (brightness); chúng xác định vị trí trong quang phổ màu, độ tinh khiết và cường độ sáng. Có hàng loạt phương pháp được tạo các mô hình màu trong các ứng dụng đồ họa. Trong phần này chỉ đưa ra những mô hình màu tiêu biểu hơn cả, giúp tìm hiểu các ứng dụng đã lựa chọn màu sắc thích hợp như thế nào.

a. RGB (Red - Green - Blue):
Mô hình màu RGB mô tả màu sắc bằng 3 thành phần chính là Red - Green và Blue. Mô hình này được xem như một khối lập phương 3 chiều với màu red là trục x, màu Green là truc y, và màu Blue là trục z. Mỗi màu trong mô hình này được xác định bởi 3 thành phần R, G, B. Ứng với các tổ hợp khác nhau của 3 màu này sẽ cho ta một màu mới .



Trong hình lập phương trên, mỗi màu gốc (R,G,B) có các gốc đối diện là các màu bù với nó. Hai màu được gọi là bù nhau khi kết hợp hai màu này lại với nhau ra màu trắng. Ví dụ : Green - Magenta, Red - Cyan, Blue - Yellow.

b. CMY (Cyan - Magenta - Yellow):
Tương tự như mô hình màu RGB nhưng 3 thành phần chính là Cyan - Magenta - Yellow. Do đó, tọa độ các màu trong mô hình CMY trái ngược với mô hình RGB. Ví dụ : màu White có các thành phần là (0,0,0), màu Black (1,1,1), màu Cyan (1,0,0),....

c. HSV (Hue - Saturation - Value):
Thực chất của mô hình này là sự biến đổi của mô hình RGB. Mô hình HSV được mô tả bằng lệnh lập phương RGB quay trên đỉnh Black. H (Hue) là góc quay trục V (value) qua 2 đỉnh Black và White. Các giá trị biến thiên của H, S, V như sau: (Hue) chỉ sắc thái có giá trị từ 0^0 - 360^0 . S (Saturation) chỉ độ bão hoà. V (Value) có giá trị từ 0 - 1. Các màu đạt giá trị bão hòa khi s = 1 và v = 1.

CHƯƠNG II CÁC ĐỐI TƯỢNG ĐỒ HỌA CƠ BẢN

2.1 Giới thiệu về các đối tượng đồ họa cơ sở

Kỹ thuật đồ họa liên quan đến tin học và toán học bởi vì hầu hết các giải thuật vẽ, tô cùng các phép biến hình đều được xây dựng dựa trên nền tảng của hình học không gian hai chiều và ba chiều. Trong chương này, chúng ta giới thiệu các thuật toán vẽ và tô các đường cơ bản như đường thẳng, đa giác, đường tròn, ellipse. Các thuật toán này giúp cho sinh viên hiểu được quá trình vẽ và tô một đối tượng hình học cơ sở như thế nào.

2.1.1 Các đối tượng đồ họa cơ sở

- Điểm: điểm là thành phần cơ sở được định nghĩa trong một hệ tọa độ, đối với hệ tọa độ 2 chiều mỗi điểm được xác định bởi hoành độ và tung độ. Ngoài thông tin tọa độ, điểm còn có thông tin màu sắc.

- Đoạn thẳng, đường thẳng: đường thẳng xác định qua 2 điểm, đoạn thẳng bị giới hạn bởi 2 điểm đầu và cuối. Phương trình đường thẳng được xác định qua 2 điểm $P(x_1, y_1)$ và $Q(x_2, y_2)$ như sau:

$$\frac{x - x_1}{y - y_1} = \frac{x_2 - x_1}{y_2 - y_1}$$

Hay ở dạng phương trình đoạn chắn $y = mx + b$, trong đó: $D_x = x_2 - x_1$, $D_y = y_2 - y_1$, $m = \frac{D_y}{D_x}$

Hay ở dạng tổng quát: $Ax + By + C = 0$, trong đó: $A = y_2 - y_1$, $B = x_1 - x_2$, $C = x_2y_1 - x_1y_2$

- Đường gấp khúc: là tập các đoạn thẳng nối với nhau một cách tuần tự, các đoạn không nhất thiết phải tạo thành hình khép kín và có thể cắt nhau. Giao của hai đoạn thẳng là đỉnh, đỉnh và đỉnh cuối của đa giác trùng nhau.

- Vùng tô: bao gồm đường biên và vùng bên trong, đường biên là đường khép kín như đa giác lồi.

- Ký tự, chuỗi ký tự: Ký tự cho phép hiển thị thông tin theo ngôn ngữ nào đó.

2.1.2 Các thuộc tính của các đối tượng đồ họa cơ sở

- Điểm có thuốc tính màu sắc.
- Đoạn thẳng, đường thẳng có thuộc tính màu sắc, độ rộng, kiểu nét.
- Vùng tô có thuộc tính của đường thẳng, và thuộc tính riêng là mầu tô và mẫu tô.
- Ký tự có thuộc tính màu sắc, kiểu chữ, cỡ chữ, khoảng cách giữa các ký tự, hướng hiển thị ký tự...

2.2 Các thuật toán vẽ điểm, đường

2.2.1 Thuật toán vẽ đường thẳng

Xét đoạn thẳng có hệ số góc $0<m<=1$ và $\Delta x>0$. Với các đoạn thẳng dạng này, nếu (x_i, y_i) là điểm đã được xác định ở bước thứ i thì điểm kế tiếp (x_{i+1}, y_{i+1}) ở bước thứ i+1 sẽ là một trong hai điểm sau:

$$\begin{cases} x_i = x_{i+1} \\ y_i = \begin{bmatrix} y_i \\ y_{i+1} \end{bmatrix} \end{cases}$$

Vấn đề đặt ra là chọn điểm vẽ như thế nào để đường thẳng được vẽ gần với đường thẳng muốn vẽ nhất và đạt được tối ưu hóa về mặt tốc độ ?

Thuật toán DDA

Là thuật toán tính toán các điểm vẽ dọc theo đường thẳng dựa vào hệ số góc của phương trình đường thẳng y = mx+b. Trong đó: $m = \frac{\Delta_y}{\Delta_x}, \Delta_y = y_{i+1} - y_i, \Delta_x = x_{i+1} - x_i$ Nhận thấy tọa độ của điểm x sẽ tăng 1 đơn vị trên mỗi điểm vẽ, còn việc quyết định chọn $y_i + 1$ là y_{i+1} hay y_i sẽ phụ thuộc vào giá trị sau khi làm tròn của tung độ y. Tuy nhiên, nếu tính trực tiếp giá trị thực của y ở mỗi bước từ phương trình y = mx+b thì cần một phép toán nhân và một phép toán cộng số thực: $y_i + 1 = mx_i + 1 + b = m(x_i + 1) + b = mx_i + b + m$

Để cải thiện tốc độ, người ta khử phép nhân trên số thực. Ta có:

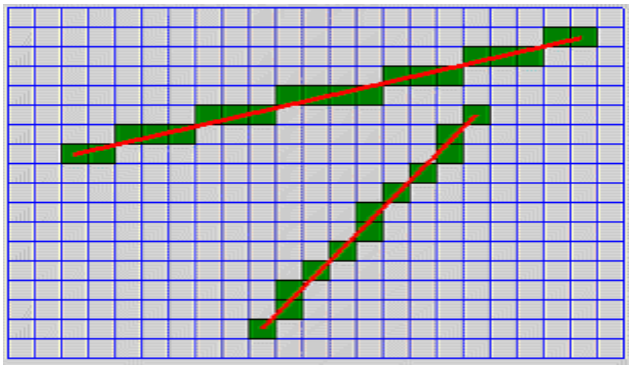
$$y_i = mx_i + b \Rightarrow y_{i+1} = y_i + m \rightarrow \text{int}(y_i + 1) \qquad (2\text{-}1)$$

Có 2 khả năng:

- $0<m<=1$: $x_{i+1} = x_i + 1, y_{i+1} = y_i + m \rightarrow \text{int}(y_{i+1})$ (2-2)
- $m>1$: $x_{i+1} = x_i + 1/m \rightarrow \text{int}(x_{i+1}), y_{i+1} = y_i + 1$

Hai trường hợp này dùng để vẽ một điểm bắt đầu từ bên trái đến điểm cuối cùng bên phải của đường thẳng (xem hình 1.1). Nếu điểm bắt đầu từ bên phải đến điểm cuối cùng bên trái thì xét ngược lại :

- $0<m<=1$: $x_{i+1} = x_i - 1, y_{i+1} = y_i - m \rightarrow \text{int}(y_{i+1})$ (2-3)
- $m>1$: $x_{i+1} = x_i - 1/m \rightarrow \text{int}(x_{i+1}), y_{i+1} = y_i - 1$

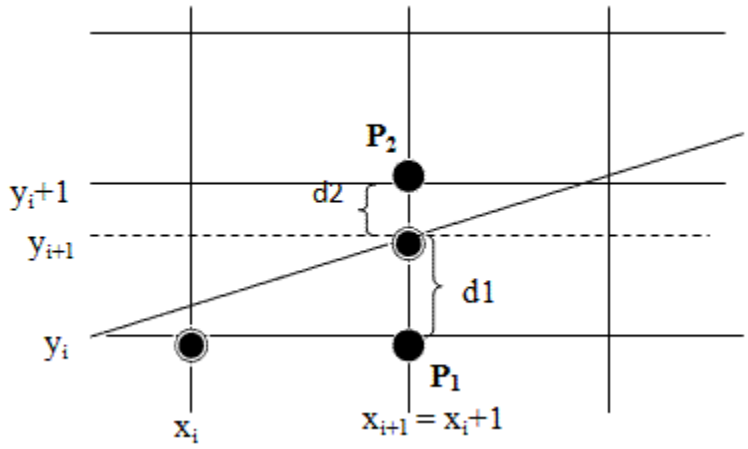


Hình 2.1 Dạng đường thẳng tương ứng 2 khả năng của m.

Tương tự, có thể tính toán các điểm vẽ cho trường hợp m<0: khi |m|<=1 hoặc |m|>1 (sinh viên tự tìm hiểu thêm).

Thuật toán Bresenham

Hình 2.2 Dạng đường thẳng nếu 0<m<=1



Gọi (x_{i+1}, y_{i+1}) là điểm thuộc đoạn thẳng (xem hình 1.2). Ta có $y = m(x_i + 1) + b$

Đặt $d1 = y_{i+1} - y_i$ và $d2 = (y_i + 1) - y_{i+1}$

Việc chọn điểm $(x_i + 1, y_i + 1)$ là P_1 hay P_2 phụ thuộc vào việc so sánh d1 và d2

- Nếu $d1 - d2 < 0$: chọn điểm P_1, tức là $y_{i+1} = y_i$
- Nếu $d1 - d2 \geq 0$: chọn điểm P_2, tức là $y_{i+1} = y_i + 1$

Ta có : $d1 - d2 = 2y_{i+1} - 2y_i - 1 = 2m(x_i+1) + 2b - 2y_i - 1$ (2-4)

$$\Rightarrow P_i = \Delta x\,(d1 - d2) = \Delta x[2m(x_i+1) + 2b - 2y_i - 1] = \Delta x\left[2\frac{\Delta y}{\Delta x}(x_i + 1) + 2b - 2y_i - 1\right]$$

$$= 2\Delta y(x_i+1) - 2\Delta x.y_i + \Delta x(2b - 1) = 2\Delta y.x_i - 2\Delta x.y_i + 2\Delta y + \Delta x(2b - 1)$$

Vậy $C = 2\Delta y + \Delta x(2b - 1) = \text{Const} \Rightarrow P_i = 2\Delta y.x_i - 2\Delta x.y_i + C$ (2-5)

Nhận xét rằng nếu tại bước thứ i ta xác định được dấu của P_i thì xem như ta xác định được điểm cần chọn ở bước (i+1). Ta có :

$$P_{i+1} - P_i = (2\Delta y.x_{i+1} - 2\Delta x.y_{i+1} + C) - (2\Delta y.x_i - 2\Delta x.y_i + C)$$

$$\Leftrightarrow P_{i+1} = P_i + 2\Delta y - 2\Delta x\,(y_{i+1} - y_i)$$

- Nếu $P_i < 0$: chọn điểm P_1, tức là $y_{i+1} = y_i$ và $P_{i+1} = P_i + 2\Delta y$
- Nếu $P_i \geq 0$: chọn điểm P_2, tức là $y_{i+1} = y_i + 1$ và $P_{i+1} = P_i + 2\Delta y - 2\Delta x$ (2-6)

Giá trị P_0 được tính từ điểm vẽ đầu tiên (x_o, y_o) theo công thức: $P_o = 2\Delta y.x_o - 2\Delta x.y_o + C$

Do (x_o, y_o) là điểm nguyên thuộc về đoạn thẳng nên ta có: $y_o = mx_o + b = \frac{\Delta y}{\Delta x}x_o + b$

Thay y_o vào phương trình trên ta được: $P_o = 2\Delta y - \Delta x$

Nhận xét:

Thuật toán Bresenham chỉ thao tác trên số nguyên và chỉ tính toán trên phép cộng và phép nhân 2. Điều này là một cải tiến làm tăng tốc độ đáng kể so với thuật toán DDA. Ý tưởng chính của thuật toán này là ở chổ xét dấu P_i để quyết định điểm kế tiếp, và sử dụng công thức truy hồi $P_{i+1} - P_i$ để tính P_i bằng các phép toán đơn giản trên số nguyên. Tuy nhiên, việc xây dựng trường hợp tổng quát cho thuật toán Bresenham có phức tạp hơn thuật toán DDA.

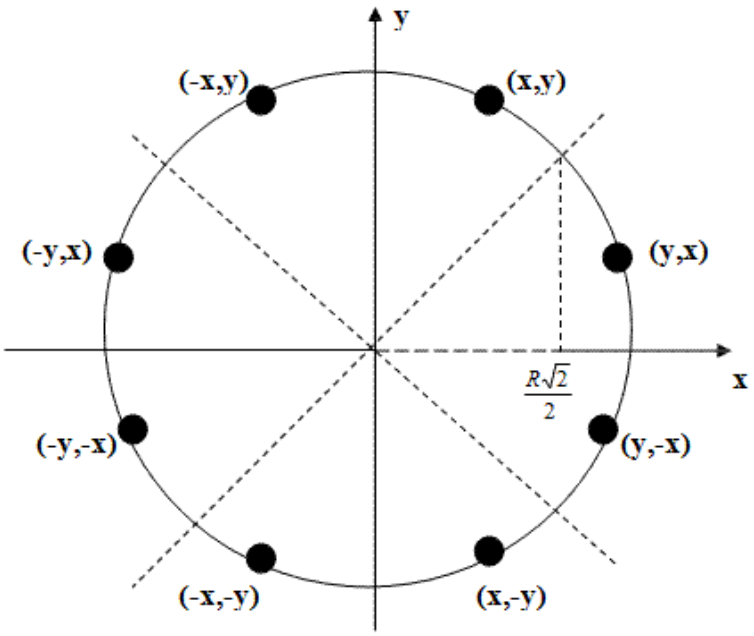
2.2.2 Thuật toán vẽ đường tròn, elip

Trong hệ tọa độ Descartes, phương trình đường tròn bán kính R có dạng:

tâm O(0,0) : $x^2 + y^2 = R^2$; tâm C(x_c,y_c): $(x - x_c)^2 + (y - y_c)^2 = R^2$ (2-7)

Trong hệ tọa độ cực : $\begin{cases} x = x_c + R\cos\theta \\ y = y_c + R\sin\theta \end{cases}, \theta \in [0, 2\pi]$

Do tính đối xứng của đường tròn C, nên ta chỉ cần vẽ 1/8 cung tròn, sau đó lấy đối xứng qua 2 trục tọa độ và 2 đường phân giác thì ta vẽ được cả đường tròn.

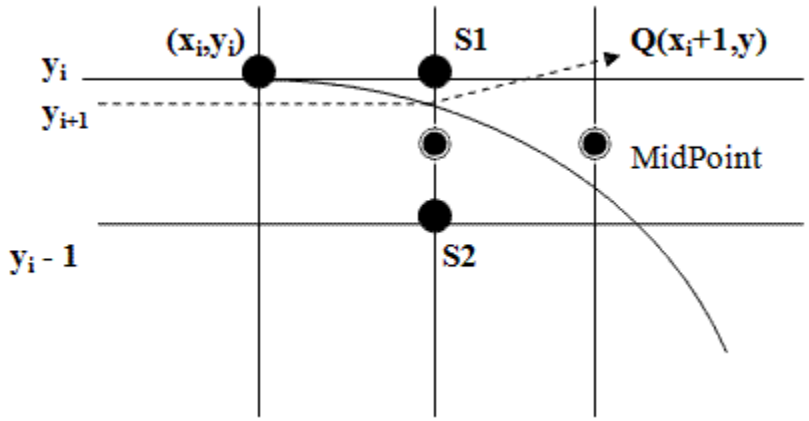


Hình 2.3 Đường tròn với các điểm đối xứng.

Thuật toán xét trung điểm

Do tính đối xứng của đường tròn nên ta chỉ cần vẽ 1/8 cung tròn, sau đó lấy đối xứng là vẽ được cả đường tròn. Thuật toán trung điểm đưa ra cách chọn y_{i+1} là y_i hay y_i-1 bằng cách so sánh điểm Q(x_i+1,y) với điểm giữa là trung điểm (Midpoint) của S1 và S2. Chọn điểm bắt đầu để vẽ là (0,R). Giả sử (x_i, y_i) là điểm nguyên đã tìm được ở bước thứ i (xem hình 1.4), thì điểm (x_i+1, y_i+1) ở bước i+1 là sự lựa chọn giữa S1 và S2.

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{bmatrix} y_i - 1 \\ y_i \end{bmatrix} \end{cases}$$



Hình 2.4 Đường tròn với điểm Q(x_i +1, y) và trung điểm.

Đặt F(x,y) = $x^2 + y^2 - R^2$, ta có :

- F(x,y) < 0 , nếu điểm (x,y) nằm trong đường tròn.
- F(x,y) = 0 , nếu điểm (x,y) nằm trên đường tròn.
- F(x,y) > 0 , nếu điểm (x,y) nằm ngoài đường tròn.

Xét P_i = F(MidPoint) = F(x_i +1, y_i - 1/2). Ta có :

- Nếu P_i < 0 : điểm MidPoint nằm trong đường tròn. Khi đó, điểm Q gần với điểm S1 hơn nên ta chọn $y_{i+1} = y_i$.
- Nếu P_i >= 0 : điểm MidPoint nằm ngoài đường tròn. Khi đó, điểm Q gần với điểm S2 hơn nên ta chọn $y_{i+1} = y_i - 1$.

Mặt khác : $P_{i+1} - P_i$ = F(x_{i+1} +1, y_{i+1} - 1/2) - F(x_i + 1, y_i - 1/2)

= $[(x_{i+1} +1)^2 + (y_{i+1} - 1/2)^2 - R^2] - [(x_i +1)^2 + (y_i - 1/2)^2 - R^2]$

= $2x_i + 3 + ((y_{i+1})^2 - (y_i)^2) - (y_{i+1} - y_i)$

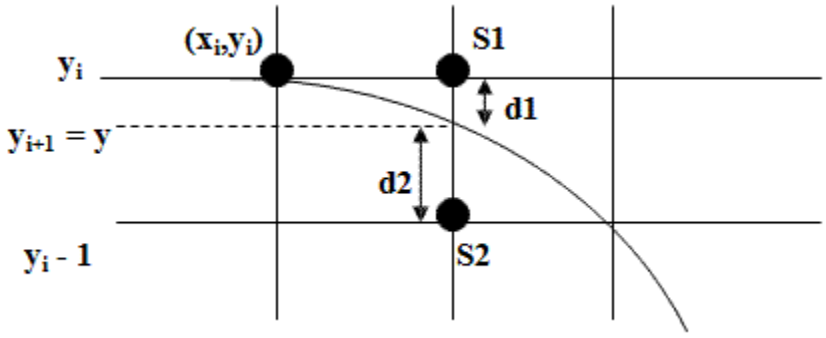
Vậy: Nếu P_i < 0 : chọn $y_{i+1} = y_i$. Khi đó $P_{i+1} = P_i + 2x_i + 3$ (2-8)
Nếu P_i >= 0 : chọn $y_{i+1} = y_i - 1$. Khi đó $P_{i+1} = P_i + 2x_i - 2y_i + 5$

P_i ứng với điểm ban đầu (x_o , y_o) = (0,R) là: P_o = F(x_o + 1, y_o - 1/2) = F(1, R - 1/2) = 5/4 -R

Thuật toán Bresenham

Tương tự thuật toán vẽ đường thẳng Bresenham, các vị trí ứng với các tọa độ nguyên nằm trên đường tròn có thể tính được bằng cách xác định một trong hai pixel gần nhất với đường tròn hơn trong mỗi bước.

Hình 2.5 Đường tròn với khoảng cách d1 và d2.



Ta có: $d1 = (y_i)^2 - y^2 = (y_i)^2 - (R^2 - (x_i + 1)^2)$ và $d2 = y^2 - (y_i - 1)^2 = (R^2 - (x_i + 1)^2) - (y_i - 1)^2$

Đặt $P_i = d1 - d2$, do đó $P_{i+1} = P_i + 4x_i + 6 + 2((y_{i+1})^2 - (y_i)^2) - 2(y_{i+1} - y_i)$

- Nếu $Pi < 0$: chọn $y_{i+1} = y_i$. Khi đó $P_{i+1} = P_i + 4x_i + 6$ (2-9)
- Nếu $P_i >= 0$: chọn $y_{i+1} = y_i - 1$. Khi đó $P_{i+1} = P_i + 4(x_i - y_i) + 10$
- P_o ứng với điểm ban đầu $(x_o , y_o) = (0,R)$ là: $P_o = 3 - 2R$.

Thuật toán vẽ elip

Tương tự thuật toán vẽ đường tròn, sử dụng thuật toán Bresenham để vẽ, ta chỉ cần vẽ 1/4 ellipse, sau đó lấy đối xứng qua các trục tọa độ sẽ vẽ được toàn bộ ellipse.

Xét ellipse có tâm O, các bán kính là a và b, phương trình là: $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$

Chọn tọa độ pixel đầu tiên cần hiển thị là $(x_i , y_i) = (0,b)$. Cần xác định pixel tiếp theo là (x_{i+1} , y_{i+1}). Ta có :

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{bmatrix} y_i \\ y_i - 1 \end{bmatrix} \end{cases} ; \quad d1 = (y_i)^2 - y^2 \; ; \quad d2 = y^2 - (y_i - 1)^2$$

Đặt $P_i = d1 - d2$, có $P_{i+1} = P_i + 2((y_{i+1})^2 - (y_i)^2) - 2(y_{i+1} - y_i) + \frac{2b^2}{a^2}(2x_i + 3)$

Nếu $P_i < 0$: chọn $y_{i+1} = y_i$, $P_{i+1} = P_i + \frac{2b^2}{a^2}(2x_i + 3)$ (2-10)

$$P_{i+1} = P_i + \frac{2b^2}{a^2}(2x_i + 3)$$

Nếu $P_i >= 0$: chọn $y_{i+1} = y_i - 1$,

P_o ứng với điểm ban đầu $(x_o, y_o) = (0, b)$ là:

$$P_o = \frac{2b^2}{a^2} - 2b + 1$$

2.3 Thuật toán tô màu

Tô màu một vùng là thay đổi màu sắc của các điểm vẽ nằm trong vùng cần tô. Một vùng tô thường đượ xác định bởi một đường khép kín nào đó gọi là đường biên. Dạng đường biên đơn giản thường gặp là đa giác. Dạng đường biên đơn giản thường gặp là đa giác. Việc tô màu thường chia làm 2 công đoạn :

- Xác định vị trí các điểm cần tô màu.
- Quyết định tô các điểm trên bằng màu nào. Công đoạn này sẽ trở nên phức tạp khi ta cần tô theo một mẫu tô nào đó chứ không phải tô thuần một màu.

Có 3 cách tiếp cận chính để tô màu. Đó là : tô màu theo từng điểm (có thể gọi là tô đơn giản), tô màu theo dòng quét và tô màu dựa theo đường biên. Chúng ta sẽ giới thiệu 2 loại thuật toán tô mầu đơn giản và tô màu đường biên. Thuật toán tô màu theo dòng quét thì sinh viên tự tìm hiểu.

2.3.1 Tô đơn giản

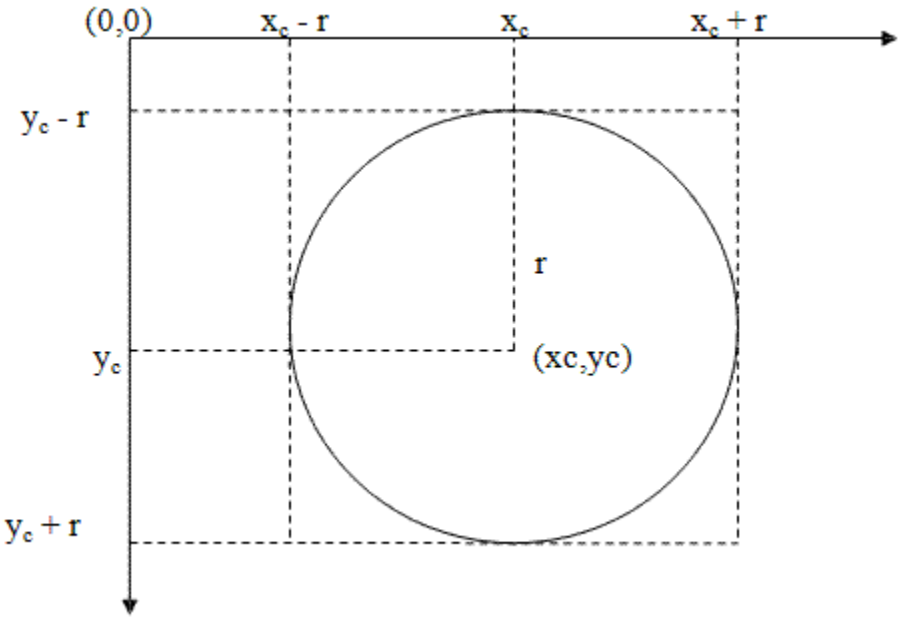
Thuật toán này bắt đầu từ việc xác định một điểm có thuộc vùng cần tô hay không ? Nếu đúng là điểm thuộc vùng cần tô thì sẽ tô với màu muốn tô.

a. Tô đường tròn

Để tô đường tròn thì ta tìm hình vuông nhỏ nhất ngoại tiếp đường tròn bằng cách xác định điểm trên bên trái $(x_c - r, y_c - r)$ và điểm dưới bên phải $(x_c + r, y_c + r)$ của hình vuông .

Giải thuật giả mã như sau:

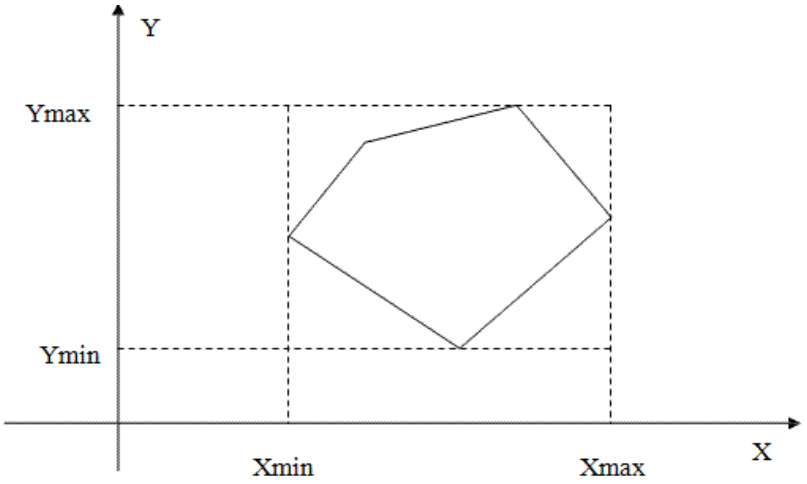
```
i chạy từ  xc-r đến xc+r
   j chạy từ  yc-r  đến  yc+r
    tìm khoảng cách d giữa hai điểm (i,j) và tâm (xc,yc)
       nếu d<r  thì tô điểm (i,j) với màu muốn tô
```



Hình 2.6 đường tròn nội tiếp hình vuông.

b.Tô đa giác

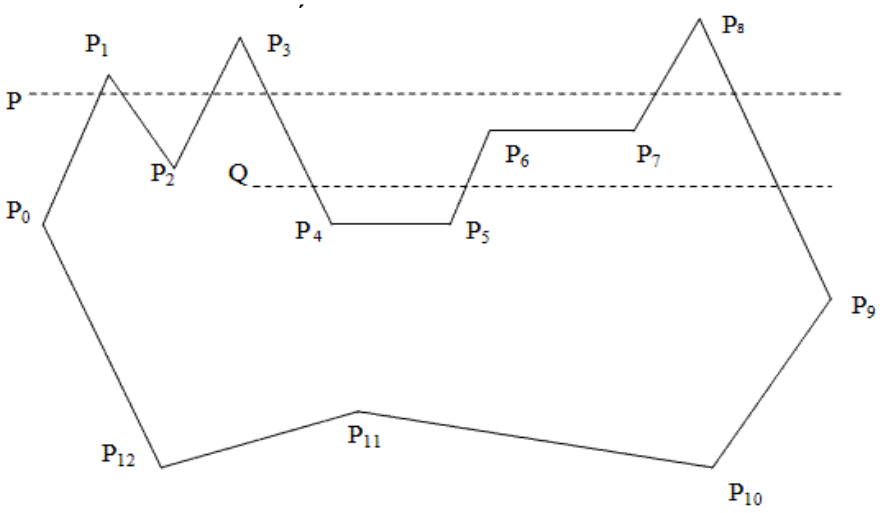
Trước tiên, chúng ta giải quyết bài toán tìm hình chữ nhật nhỏ nhất có các cạnh song song với hai trục tọa độ chứa đa giác cần tô dưa vào hai tọa độ (xmin, ymin), (xmax, ymax).



Hình 2.7 đa giác nội tiếp hình chữ nhật.

Trong đó, xmin, ymin là hoàn
nhất của các đỉnh của đa giác.
ymax (hoặc ngược lại). Xét điể

Thông thường một điểm nằm tr
cắt biên của đa giác phải là một
một giao điểm phải được tính
thường ta chọn tia qua phải.



Hình 2.8 Xét đa giác gồm 13 đỉnh là P_o , P_1 , ..., P_{12} , P_o

Gọi tung độ của đỉnh Pi là Pi.y . Nếu :

- $P_i.y < Min (P_{i+1}.y, P_{i-1}.y)$ hay $P_i.y > Max (P_{i+1}.y, P_{i-1}.y)$ thì P_i là đỉnh cực trị (cực tiểu hay cực đại).
- $P_{i-1}.y < P_i.y < P_{i+1}.y$ hay $P_{i-1} > P_i.y > P_{i+1}.y$ thì P_i là đỉnh đơn điệu.
- $P_i = P_{i+1}$ và $P_i.y < Min (P_{i+2}.y, P_{i-1}.y)$ hay $P_i > Max (P_{i+2}.y, P_{i-1}.y)$ thì đoạn $[P_i,P_{i+1}]$ là đoạn cực trị (cực tiểu hay cực đại).
- $P_i = P_{i+1}$ và $P_{i-1}.y < P_i.y < P_{i+2}.y$ hay $P_{i-1} > P_i.y > P_{i+2}.y$ thì đoạn $[P_i,P_{i+1}]$ là đoạn đơn điệu.

Thuật toán kiểm tra điểm có nằm trong đa giác

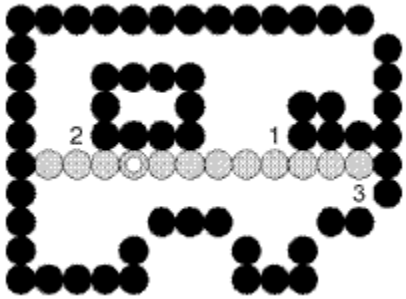
- Với mỗi đỉnh của đa giác ta đánh dấu là 0 hay 1 theo qui ước như sau: nếu là đỉnh cực trị hay đoạn cực trị thì đánh số 0. Nếu là đỉnh đơn điệu hay đoạn đơn điệu thì đánh dấu 1. - Xét số

giao điểm của tia nữa đường thẳng từ P là điểm cần xét với biên của đa giác. Nếu số giao điểm là chẵn thì kết luận điểm không thuộc đa giác. Ngược lại, số giao điểm là lẻ thì điểm thuộc đa giác.

2.3.2 Tô màu đường biên

Bài toán đặt ra : Cần tô màu một vùng nếu biết được màu của đường biên vùng tô và một điểm nằm bên trong vùng tô.

Ý tưởng : Bắt đầu từ một điểm nằm bên trong vùng tô, kiểm tra các điểm lân cận của nó đã được tô với màu muốn tô, hay điểm lân cận có màu trùng với màu biên không ? Nếu cả hai trường hợp đều không phải thì ta sẽ tô điểm đó với màu muốn tô. Quá trình này được lặp lại cho đến khi không còn tô được nữa thì dừng.



(x,y-1)
(x-1,y)(x,y)(x+1,y)
(x,y+1)

Có 2 quan điểm về cách tô này. Đó là dùng 4 điểm lân cận (có thể gọi là 4 liên thông) hay 8 điểm lân cận (8 liên thông)

Cài đặt minh họa thuật toán 4 liên thông

```
Boundary_fill ( x,y, mauto, maubien :integer);
 mau_ht : integer;
  begin
    mau_ht:= getpixel(x, y);
     If  (mau_ht <> mauto) and (mau_ht <> maubien) then
     begin
      putpixel(x,y,color);
      Boundary_fill ( x+1,y, mauto, maubien );
     Boundary_fill ( x-1,y, mauto, maubien );
     Boundary_fill ( x,y+1, mauto, maubien );
     Boundary_fill ( x,y-1, mauto, maubien );
    end;
end;
```

Nhận xét :

- Thuật toán có thể không chính xác khi có một số điểm nằm trong vùng tô có màu là màu cần tô của vùng.
- Việc thực hiện gọi đệ qui làm thuật toán không thể sử dụng cho vùng tô lớn (tràn stack).
- Có thể khắc phục việc tràn stack bằng cách giảm số lần gọi đệ qui. Khởi đầu điểm (x,y) là điểm có vị trí đặc biệt trong vùng tô, sau đó, gọi đệ qui các điểm lân cận của (x,y).

2.4 Bài tập áp dụng

Bài 1 Tại sao phải so sánh giá trị P_i với 0 trong các giải thuật Bresenham hay Midpoint. Bản chất của so sánh là gì?

Gợi ý: giá trị P_i tạo ra không có phép chia nguyên, tính toán nhanh và chính xác hơn là tính trực tiếp.

Bài 2 Cặt đặt các giải thuật Bresenham, MidPoint vẽ đoạn thẳng với hệ số góc trong khoảng [-1, 1], đường tròn.

Gợi ý: Vẽ đường thẳng, ta xét 3 trường hợp $0 < m < 1$, $-1 < m < 0$, $m= 0$.

Bài 3 Giải thích tại sao chỉ chọn cung 1/8 để vẽ rồi lấy đối xứng mà không mở rộng cho cung 1/16 hay 1/32.

Gợi ý: Nếu cung độ dài nhỏ hơn 1/8 thì giải thuật Midpoint không còn đúng nữa, vì không chính xác.

Bài 4 Cài đặt giải thuật vẽ elip có tâm là gốc tọa độ, bán kính trục chính, trục phụ là a, b.

Gợi ý: Dựa trên cơ sở giải thuật Midpoint cho vẽ đường tròn, nhưng vẽ cho cung ¼ rồi lấy đối xứng.

Bài 5 Cài đặt thuật toán tô màu đường biên cho đối tượng hình tròn.

Gợi ý: Sử dụng giả thuật tô mầu đường biên cải tiến, sử dụng giải thuật không đệ quy thì tốt hơn giải thuật đệ quy.

CHƯƠNG III CÁC PHÉP BIẾN ĐỔI ĐỒ HỌA 2 CHIỀU

3.1 Các phép biến đổi cơ sở

Trong lĩnh vực đồ họa máy tính, hình dạng và kích thước của đối tượng 2 chiều đặc trưng bởi một số 2 chiều quan hệ với hệ thống tọa độ Descartes. Một tập hợp các phép biến đổi hình học áp dụng cho đối tượng như: dịch chuyển, thay đổi kích thước, phương chiều của nó . Các hệ CAD luôn có thao tác như: scale, move, rotate, copy … thực hiện những phép biến đổi hình học cơ sở.

3.1.1 Phép tịnh tiến

Khả năng tịnh tiến đối tượng là một đặc điểm cần thiết của mọi hệ thống đồ họa. Phép tịnh tiến làm cho đối tượng dịch chuyển theo một hướng với độ dài xác định. Dưới dạng toán học, mô tả với hệ phương trình sau:

$$\begin{cases} x' = x + T_x \\ y' = y + T_y \end{cases} \qquad (3\text{-}1)$$

3.1.2 Phép biến đổi tỷ lệ

Phép biến đổi tỉ lệ làm thay đổi kích thước đối tượng. Để co hay giãn tọa độ của một điểm P(x,y) theo trục hoành và trục tung lần lượt là S_X và S_Y (gọi là các hệ số tỉ lệ), ta nhân S_X và S_Y lần lượt cho các tọa độ của P.

$$\begin{cases} x' = x.S_X \\ y' = y.S_Y \end{cases} \qquad (3\text{-}2)$$

- Khi các giá trị S_X , S_Y nhỏ hơn 1, phép biến đổi sẽ thu nhỏ đối tượng. Ngược lại, khi các giá trị này lớn hơn 1, phép biến đổi sẽ phóng lớn đối tượng.

- Khi $S_X = S_Y$, người ta gọi đó là phép đồng dạng (uniform scaling). Đây là phép biến đổi bảo toàn tính cân xứng của đối tượng. Ta gọi là phép phóng đại nếu |S|>1 và là phép thu nhỏ nếu |S|<1.

- Nếu hai hệ số tỉ lệ khác nhau thì ta gọi là phép không đồng dạng. Trong trường hợp hoặc S_X hoặc S_Y có giá trị 1, ta gọi đó là phép căng (strain).

3.1.3 Phép đối xứng

Thuật ngữ đối xứng hiểu như hình ảnh trong gương. Phép đối xứng sử dụng trong việc tạo các hình đối xứng. Ví dụ như một nửa hình được tạo ra, sau đó lấy đối xứng để tạo nguyên hình, ví dụ như việc tạo khung đỡ mái nhà. Các ứng dụng như CAD luôn có chỉ thị Mirror thực hiện chức năng trên.

Phép đối xứng qua điểm hay qua trục nào đó. Ma trận đối xứng sẽ có dạng chung như sau:

$$[Tr] = \begin{bmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (3\text{-}3)$$

Các trường hợp khác nhau của đối xứng trục X, Y, Z như sau:

$$[Tr]_X = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad [Tr]_y = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad [Tr]_Z = \begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.1.4 Phép quay

Phép quay làm thay đổi hướng của đối tượng. Một phép quay đòi hỏi phải có tâm quay, góc quay. Góc quay dương thường được qui ước là chiều ngược chiều kim đồng hồ. Ta có công thức biến đổi của phép quay điểm P(x,y) quanh gốc tọa độ góc θ tới vị trí P'(x', y'):

$$\begin{cases} x' = x.\cos\theta - y.\sin\theta \\ y' = x.\sin\theta + y.\cos\theta \end{cases} \qquad (3\text{-}4)$$

hoặc dưới dạng ma trận sau: $[x' \;\; y' \;\; 1] = [x \;\; y \;\; 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$

3.2 Kết hợp các phép biến đổi

Những phép biến hình 2 chiều đòi hỏi không chỉ một mà là chuỗi thứ tự các phép biến hình cơ sở để cuối cùng thu được mục tiêu mong muốn.

3.2.1 Kết hợp các phép tịnh tiến

Nếu ta thực hiện phép tịnh tiến lên điểm P được điểm P', rồi lại thực hiện tiếp một phép tịnh tiến khác lên P' được điểm Q. Như vậy, điểm Q là ảnh của phép biến đổi kết hợp hai phép tịnh tiến liên tiếp.

$$\begin{cases} Q_x = P_x + (T_{x1} + T_{x2}) \\ Q_y = P_y + (T_{y1} + T_{y2}) \end{cases}$$

Vậy kết hợp hai phép tịnh tiến là một phép tịnh tiến. Từ đó, ta có kết hợp của nhiều phép tịnh tiến là một phép tịnh tiến.

3.2.2 Kết hợp các phép biến đổi tỉ lệ

Tương tự như phép tịnh tiến, ta có tọa độ điểm Q là điểm có được sau hai phép tịnh tiến $M_1(S_{x1}, S_{y1})$, $M_2 (S_{x2}, S_{y2})$ là :

$$\begin{cases} Q_x = P_x * S_{x1} * S_{x2} \\ Q_y = P_y * S_{y1} * S_{y2} \end{cases}$$

3.2.3 Kết hợp các phép quay

Tương tự, ta có tọa độ điểm Q là điểm kết quả sau khi kết hợp hai phép quay quanh gốc tọa độ $M_{R1}(\theta_1)$ và $M_{R2}(\theta_2)$ là :

$$\begin{cases} Q_x = P_x*\cos(\theta_1+\theta_2) - P_y*\sin(\theta_1+\theta_2) \\ Q_y = P_x*\sin(\theta_1+\theta_2) + P_y*\cos(\theta_1+\theta_2) \end{cases}$$

3.3 Một số phép biến đổi khác

3.3.1 Phép biến dạng

Phép biến dạng làm thay đổi hình dạng đối tượng, biến dạng theo trục hoành hay trục tung bằng cách thay đổi tọa độ điểm ban đầu theo cách sau đây:

$$[M]_X = \begin{bmatrix} 1 & 0 & 0 \\ s_x & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad [M]_y = \begin{bmatrix} 1 & s_y & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

S_x và S_y là các hệ số biến dạng theo trục hoành và trục tung.

3.3.2 Phép đối xứng

Phép đối xứng xem như phép quay quanh trục đối xứng góc 180^o. Nếu trục đối xứng là trục hoành hay trục tung , ta có các ma trận biến đổi đối xứng qua trục hoành và trục tung như sau:

$$[M]_{RFX} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad [M]_{RFY} = \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

3.3.3 Tính chất của phép biến đổi affine

Phép biến đổi affine là phép biến đổi tuyến tính, như những biến đổi trên. Sau đây là một vài tích chất cơ bản của phép biến đổi affine:

- Phép biến đổi affine bảo toàn đường thẳng.
- Tính song song của các đường thẳng được bảo toàn.
- Tính tỷ lệ khoảng cách được bảo toàn.

3.4 Phép biến đổi giữa các hệ tọa độ

Để thuận tiện cho việc mô tả đối tượng, thông thường đối tượng được mô tả trong hệ tọa độ cục bộ gắn với chúng. Tuy nhiên, muốn hiển thị toàn bộ ảnh bao gồm nhiều đối tượng thành phần, các mô tả phải chuyển về một hệ tọa độ. Quá trình chuyển đổi tọa độ chia thành 2 loại:

- Từ hệ tọa độ cực, tọa độ cầu sang hệ tọa độ đề các.
- Giữa 2 hệ tọa độ đề các.

Trong nội dung trình bày của chương này, ta chỉ xét đối tượng hình học trong không gian tọa độ Đề các, do đó ta khảo sát phép biến đổi giữa 2 hệ tọa độ Đề các. Giả sử ta có hệ tọa độ I có gốc

tọa độ O và các véc tơ đơn vị là i, j . Hệ tọa độ II là ảnh của hệ tọa độ I qua phép biến đổi T(M), có gốc tọa độ O' và các véc tơ đơn vị là tương ứng u, v. Lúc này, một điểm P(x, y) trong hệ tọa độ I sẽ biến đổi thành điểm Q(a, b) trong hệ tọa độ II.

Qua chứng minh, tìm được mối liên hệ giữa 2 điểm Q và P: $Q = P.M^{-1}$

3.5 Bài tập áp dụng

Bài 1 Cho biết phép biến đổi hình vuông thành hình chữ nhật?

Gợi ý: Sử dụng phép biến đổi tỷ lệ

Bài 2 Chứng minh rằng ma trận phép biến đổi lấy đối xứng qua đường thẳng y = x tương đương với kết hợp của phép lấy đối xứng trục hoành và phép quay quanh gốc tọa độ góc 90^o .

Gợi ý: Tìm ma trận M_1 của phép lấy đối xứng qua trục y = x, ma trận M_2 của phép đối xứng qua trục hoành, ma trận M_3 của phép quay 90^o quanh gốc tọa độ. Chứng minh $M_1 = M_2 M_3$

Bài 3 Chứng minh phép quay có tính giao hoán.

Gợi ý: Giả sử với phép quay quanh gốc tọa độ 1 góc α, ta có được kết quả:

$$M_{R1}(\alpha 1)\, M_{R2}(\alpha 2) = M_R (\alpha 1 + \alpha 2) = M_{R2}(\alpha 2)\, M_{R1}(\alpha 1)$$

Bài 4 Chứng minh phép biến đổi affine được phân tích thành tích của các phép tịnh tiến, tỷ lệ, quay.

Gợi ý: Áp dụng quy tắc phân rã phép biến đổi.

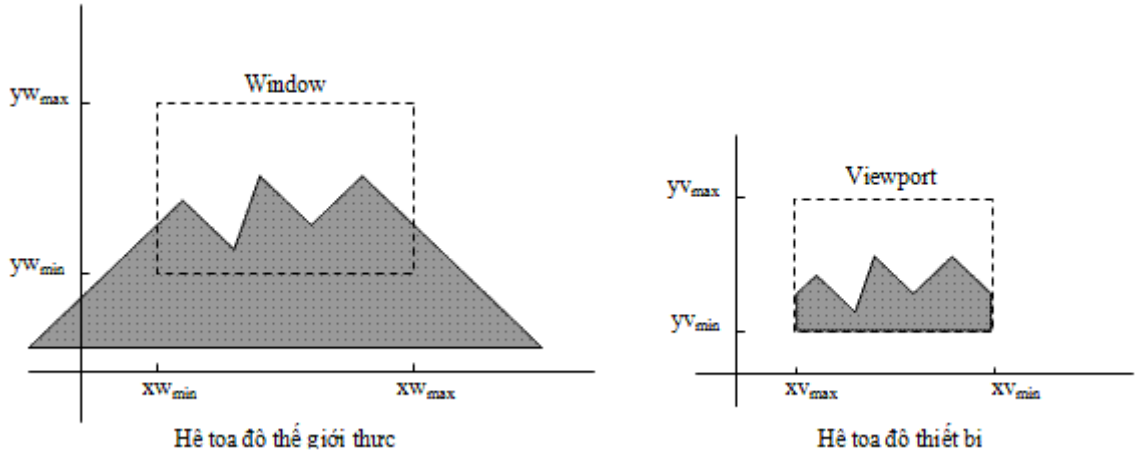
Bài 5 Hệ tọa độ x'O'y' nhận được bằng cách quay quanh gốc tọa độ góc α, rồi tịnh tiến theo véc tơ (tr_x, tr_y) trong hệ tọa độ xOy. Hãy cho biết công thức tọa độ của điểm Q trong hệ tọa độ x'O'y' nếu P(x, y) là tọa độ của điểm P trong xOy.

Gợi ý: Hệ tọa độ x'O'y' là ảnh của hệ tọa độ xOy, cần tìm ma trận biến đổi là tích giữa ma trận quay và ma trận tịnh tiến.

CHƯƠNG IV PHÉP QUAN SÁT 2 CHIỀU

4.1 Cửa sổ, Vùng quan sát

Hệ tọa độ Descartes là dễ thích ứng cho các chương trình ứng dụng để miêu tả các hình ảnh trên hệ tọa độ thực. Các hình ảnh được định nghĩa trên hệ tọa độ thực này sau đó được hệ đồ họa vẽ lên các hệ tọa độ thiết bị. Điển hình, một vùng đồ họa cho phép người sử dụng xác định vùng nào của hình ảnh sẽ được hiển thị và bạn muốn đặt nó ở nơi nào trên hệ tọa độ thiết bị. Một vùng đơn lẻ hoặc vài vùng của hình ảnh có thể được chọn. Những vùng này có thể được đặt ở những vị trí tách biệt, hoặc một vùng có thể được chèn vào một vùng lớn hơn. Quá trình biến đổi này liên quan đến những thao tác như tịnh tiến, biến đổi tỷ lệ vùng được chọn , xóa bỏ vùng không được chọn. Những thao tác trên là Windowing và Clipping.



Hình 4.1 ánh xạ cửa sổ - đến – vùng quan sát

Một vùng có dạng hình chữ nhật được xác định trong hệ tọa độ thực được gọi là một cửa sổ (window). Còn vùng hình chữ nhật trên thiết bị hiển thị để cửa sổ đó ánh xạ đến được gọi là một vùng quan sát (viewport). Hình 3.1 minh họa việc ánh xạ một phần hình ảnh vào trong một vùng quan sát. Việc ánh xạ này gọi là một phép biến đổi hệ quan sát, biến đổi cửa sổ, biến đổi chuẩn hóa. Các lệnh để xây dựng một cửa sổ và vùng quan sát từ một chương trình ứng dụng có thể được định nghĩa như sau:

```
set_window(xw_min, xw_max, yw_min, yw_max)
set_viewport(xv_min, xv_max, yv_min, yv_max)
```

Các tham số trong mỗi hàm được dùng để định nghĩa các giới hạn tọa độ của các vùng chữ nhật. Các giới hạn của cửa sổ được xác định trong hệ tọa độ thực. Hệ tọa độ thiết bị chuẩn thường được dùng nhất cho việc xác định vùng quan sát, dù rằng hệ tọa độ thiết bị có thể được dùng nếu chỉ có một thiết bị xuất duy nhất trong hệ thống. Khi hệ tọa độ thiết bị chuẩn được dùng, lập trình viên xem thiết bị xuất có giá trị tọa độ trong khoảng 0..1. Một sự xác định vùng quan sát được cho với các giá trị trong khoảng này. Các việc xác định sau đây, đặt một phần của sự định nghĩa hệ tọa độ thực vào trong góc trên bên phải của vùng hiển thị, như được minh họa trong hình 3.2:

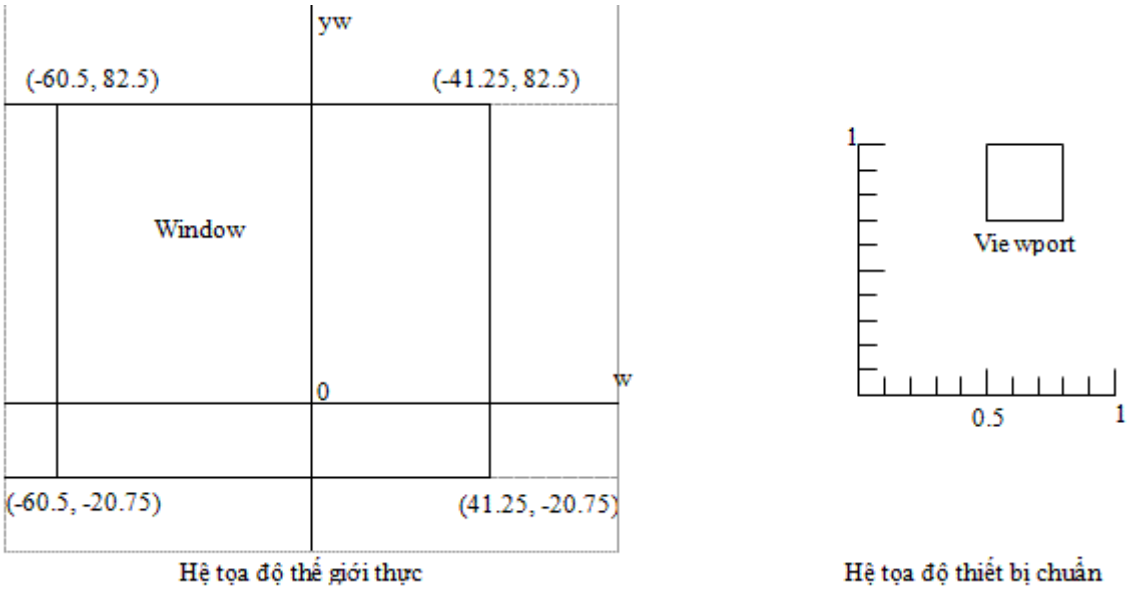
```
set_window(-60.5, 41.25, -20.75, 82.5);

set_viewport(0.5, 0.8, 0.7, 1.0);
```

Nếu một cửa sổ buộc phải được ánh xạ lấp đầy vùng hiển thị, sự xác định vùng quan sát được thiết

lập theo thủ tục:

set_viewport(0,1, 0, 1)

Các vị trí được biểu diễn trên hệ tọa độ thiết bị chuẩn phải được biến đổi sang hệ tọa độ thiết bị trước khi được hiển thị bởi một thiết bị xuất cụ thể. Thông thường một thiết bị xác định được chứa trong các gói đồ họa cho mục đích này. Thuận lợi của việc dùng hệ tọa độ thiết bị chuẩn là để các gói đồ họa độc lập với thiết bị. Các thiết bị xuất khác nhau có thể được dùng nhờ việc cung cấp các trình điều khiển thiết bị thích hợp. Mọi điểm được tham khảo đến trong các gói đồ họa phải được xác định tương ứng trong hệ tọa độ Descartes. Bất kỳ sự định nghĩa hình ảnh nào dùng trong một hệ tọa độ khác, như hệ tọa độ cực, người sử dụng trước tiên phải biến đổi nó sang hệ tọa độ thực. Những hệ tọa độ Descart này sau đó được dùng trong các lệnh cửa sổ để xác định phần nào của hình ảnh muốn được hiển thị.



Hình 4.2: Ánh xạ một cửa sổ vào một vùng quan sát trong hệ tọa độ thiết bị chuẩn

4.2 Phép biến đổi từ cửa sổ - đến – vùng quan sát

Khi tất cả các điểm, đoạn thẳng, và văn bản vừa bị cắt, chúng được ánh xạ lên vùng vùng quan sát để hiển thị. Phép biến đổi đến vùng quan sát này được thực hiện để các vị trí tọa độ liên hệ được giữ lại. Trong hình, một điểm ở vị trí (x_w, y_w) trong một cửa sổ được ánh xạ và trong vị trí (x_v, y_v) trong vùng quan sát. Để duy trì sự sắp đặt liên hệ tương tự trong vùng quan sát như trong cửa sổ, chúng ta cần xác định ánh xạ:

$$\frac{x_v - x_{vmin}}{x_{vmax} - x_{vmin}} = \frac{x_w - x_{wmin}}{x_w - x_{wmin}} \quad , \frac{y_v - y_{vmin}}{y_{vmax} - y_{vmin}} = \frac{y_w - y_{wmin}}{y_w - y_{wmin}} \tag{4-1}$$

vì vậy:

$$x_v = (x_w - x_{wmin}) \frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}} + x_{vmin}$$

$$y_v = (y_w - y_{wmin}) \frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}} + y_{vmin} \quad (4\text{-}2)$$

Các giá trị tỷ lệ $\frac{x_{vmax} - x_{vmin}}{x_{wmax} - x_{wmin}}$ và $\frac{y_{vmax} - y_{vmin}}{y_{wmax} - y_{wmin}}$ là hằng số đối với các điểm được ánh xạ và là hệ số tỷ lệ S_x và S_y theo 2 trục tương ứng. Nếu 2 tỷ lệ khác nhau thì hình vẽ mới trong vùng quan sát bị biến dạng so với hình ban đầu, còn nếu 2 tỷ lệ giống nhau thì hình vẽ mới không biến dạng.

4.3 Phép cắt xén 2 chiều

Ánh xạ một vùng cửa sổ vào trong một vùng quan sát, kết quả là chỉ hiển thị những phần trong phạm vi cửa sổ. Mọi thứ bên ngoài cửa sổ sẽ bị loại bỏ. Các thủ tục để loại bỏ các phần hình ảnh nằm bên ngoài biên cửa sổ được xem như các thuật toán cắt xén (clipping algorithms) hoặc đơn giản được gọi là clipping.

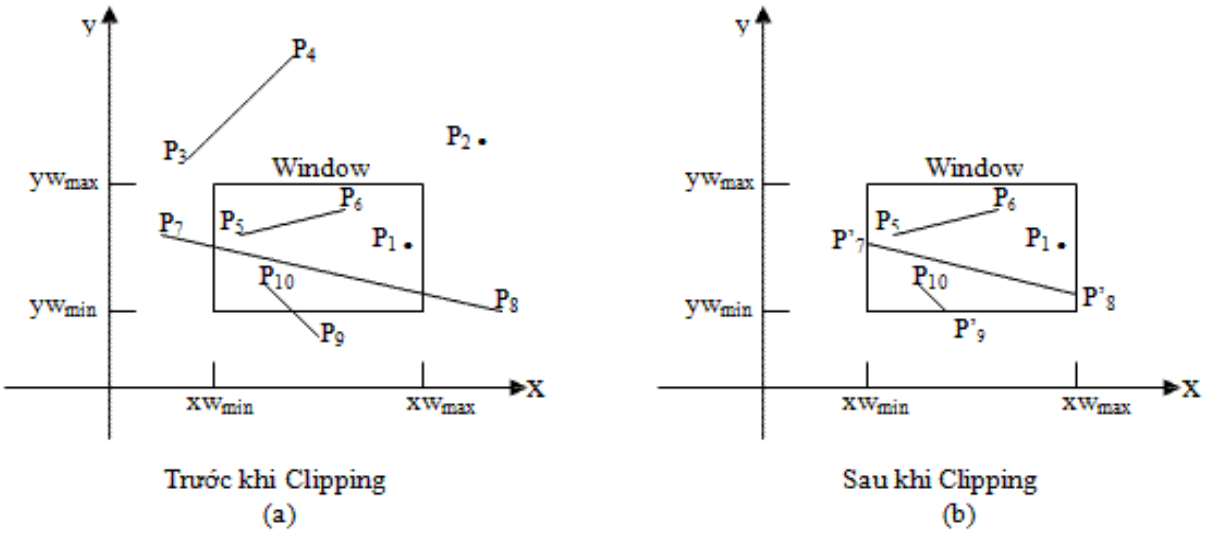
Việc cài đặt phép biến đổi cửa sổ thường được thực hiện bằng việc cắt khỏi cửa sổ, sau đó ánh xạ phần bên trong cửa sổ vào một vùng quan sát . Như một lựa chọn, một vài gói đồ họa đầu tiên ánh xạ sự định nghĩa trong hệ tọa độ thực vào trong hệ tọa độ thiết bị chuẩn và sau đó cắt khỏi biên vùng quan sát. Trong các các phần thảo luận sau, chúng ta giả thiết rằng việc cắt được thực hiện dựa vào đường biên cửa sổ trong hệ tọa độ thực. Sau khi cắt xong, các điểm bên trong cửa sổ mới được ánh xạ đến vùng quan sát.

Việc cắt các điểm khỏi cửa sổ được hiểu đơn giản là chúng ta kiểm tra các giá trị tọa độ để xác định xem chúng có nằm bên trong biên không. Một điểm ở vị trí (x,y) được giữ lại để chuyển đổi sang vùng quan sát nếu nó thỏa các bất phương trình sau:

$$xw_{min} \le x \le xw_{max}, yw_{min} \le y \le yw_{max} \quad (4\text{-}3)$$

Nếu điểm nào không thỏa một trong bốn bất phương trình trên, nó bị cắt bỏ. Trong hình dưới đây, điểm P1 được giữ lại, trong khi điểm P2 bị cắt bỏ.

Hình 4.3 minh họa các quan hệ có thể có giữa các vị trí đoạn thẳng với biên cửa sổ. Chúng ta kiểm tra một đoạn thẳng xem có bị cắt hay không bằng việc xác định xem hai điểm đầu mút đoạn thẳng là nằm trong hay nằm ngoài cửa sổ. Một đoạn thẳng với cả hai đầu nằm trong cửa sổ thì được giữ lại hết, như đoạn từ P5 đến P6. Một đoạn với một đầu nằm ngoài (P9) và một đầu nằm trong (P10) sẽ bị cắt bớt tại giao điểm với biên cửa sổ (P'9). Các đoạn thẳng có cả hai đầu đều nằm ngoài cửa sổ, có thể rơi vào hai trường hợp: toàn bộ đoạn thẳng đều nằm ngoài hoặc đoạn thẳng cắt hai cạnh cửa sổ. Đoạn từ P3 đến P4 bị cắt bỏ hoàn toàn. Nhưng đoạn từ P7 đến P8 sẽ được giữ lại phần từ P'7 đến P'8.

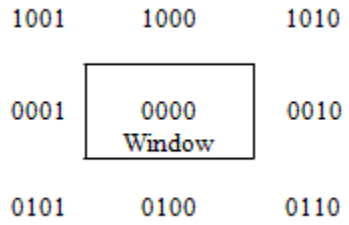


Hình 4.3

Thuật toán cắt xén đường xác định xem đoạn nào toàn bộ nằm trong, đoạn nào bị cắt bỏ hoàn toàn hay bị cắt một phần. Đối với các đoạn bị cắt bỏ một phần, các giao điểm với biên cửa sổ phải được tính. Vì một hình ảnh có thể chứa hàng ngàn đoạn thẳng, việc xử lý clipping nên được thực hiện sao cho có hiệu quả nhất. Trước khi đi tính các giao điểm, một thuật toán nên xác định rõ tất cả các đoạn thẳng được giữ lại hoàn toàn hoặc bị cắt bỏ hoàn toàn. Với những đoạn được xem xét là bị cắt bỏ, việc xác định các giao điểm cho phần được giữ lại nên được thực hiện với sự tính toán ít nhất.

4.3.1 Giải thuật Cohen – Suntherland

Một tiếp cận để cắt các đoạn là dựa trên cơ chế đánh mã được phát triển bởi Cohen và Sutherland. Mọi điểm ở hai đầu mút đoạn thẳng trong hình ảnh sẽ được gán một mã nhị phân 4 bit, được gọi là mã vùng, giúp nhận ra vùng tọa độ của một điểm. Các vùng này được xây dựng dựa trên sự xem xét với biên cửa sổ, như ở hình 4.4. Mỗi vị trí bit trong mã vùng được dùng để chỉ ra một trong bốn vị trí tọa độ tương ứng của điểm so với cửa sổ: bên trái, phải, trên đỉnh, dưới đáy. Việc đánh số theo vị trí bit trong mã vùng từ 1 đến 4 cho từ phải sang trái, các vùng tọa độ có thể liên quan với vị trí bit như sau: Bit 1 – trái ; Bit 2 – phải; Bit 3 – dưới; Bit 4 – trên.



Giá trị 1 ở bất kỳ vị trí nào chỉ ra rằng điểm ở vị trí tương ứng, ngược lại bit ở vị trí đó là 0. Nếu một điểm nằm trong cửa sổ, mã vị trí là 0000. Một điểm bên dưới và bên trái cửa sổ có mã vùng là 0101 .

Các giá trị bit trong mã vùng được xác định bằng cách so sánh giá trị tọa độ (x,y) của điểm đầu mút với biên cửa sổ. Bit 1 đặt lên 1 nếu x < xwmin. Các giá trị của ba bit còn lại được xác định bằng cách so sánh tương tự. Trong các ngôn ngữ lập trình, làm việc trên bit như thế này có thể thực hiện được, các giá trị bit mã vùng có thể được xác định theo các bước sau: (1) Tìm hiệu giữa tọa độ các điểm đầu mút với biên cửa sổ. (2) Dùng bit dấu (kết quả của mỗi hiệu) để đặt giá trị tương ứng trong mã vùng. Bit 1 là bit dấu của x - xwmin; bit 2 là bit dấu của xwmax – x; bit 2 là bit dấu của y - ywmin; và bit 4 là bit dấu của ywmax – y.

Khi chúng ta xây dựng xong các mã vùng cho tất cả các điểm đầu mút, chúng ta có thể xác định nhanh chóng đoạn thẳng nào là hoàn toàn nằm trong cửa sổ, đoạn nào là hoàn toàn nằm ngoài. Bất kỳ đoạn nào có mã vùng của cả 2 đầu mút là 0000 thì nằm trong cửa sổ và chúng ta chấp nhận các đường này. Bất kỳ đường nào mà trong hai mã vùng của hai đầu mút có một số 1 ở cùng vị trí bit thì đoạn hoàn toàn nằm ngoài cửa sổ, và chúng ta loại bỏ các đoạn này. Ví dụ, chúng ta vứt bỏ đoạn có mã vùng ở một đầu là 1001, còn đầu kia là 0101 (có cùng bit 1 ở vị trí 1 nên cả hai đầu mút của đoạn này nằm ở phía bên trái cửa sổ). Một phương pháp có thể được dùng để kiểm tra các đoạn cho việc cắt toàn bộ là thực hiện phép logic and với cả hai mã vùng. Nếu kết quả không phải là 0000 thì đoạn nằm bên ngoài cửa sổ.

Các đường không được nhận dạng là hoàn toàn nằm trong hay hoàn toàn nằm ngoài một cửa sổ thông qua các phép kiểm tra trên sẽ được tìm giao điểm với biên cửa sổ. Như được chỉ ra ở hình dưới đây, các đường thuộc nhóm này có thể cắt hoặc không cắt cửa sổ. Chúng ta có thể xử lý các đoạn này bằng cách so sánh một điểm đầu mút (cái đang nằm ngoài cửa sổ) với một biên cửa sổ để xác định phần nào của đường sẽ bị bỏ. Sau đó, phần đường được giữ lại sẽ được kiểm tra với các biên khác, và chúng ta tiếp tục cho đến khi toàn bộ đường bị bỏ đi hay đến khi một phần đường được xác định là nằm trong cửa sổ. Chúng ta xây dựng thuật toán để kiểm tra các điểm đầu mút tương tác với biên cửa sổ là ở bên trái, bên phải, bên dưới hay trên đỉnh. Để minh họa các bước xác định trong việc cắt các đoạn khỏi biên cửa sổ dùng thuật toán của Cohen-Sutherland, chúng ta xem các đoạn trong hình được xử lý như thế nào. Bắt đầu ở điểm đầu mút bên dưới từ P1 đến P2, ta kiểm tra P1 với biên trái, phải và đáy cửa sổ và thấy rằng điểm này nằm phía dưới cửa sổ. Ta tìm giao điểm P'1 với biên dưới. Sau khi tìm giao điểm P'1, chúng ta vứt bỏ đoạn từ P1 đến P'1. Giao điểm P'2 được tính, và đoạn từ P'1 đến P'2 được giữ lại. Kết thúc quá trình xử lý đoạn P1P2. Bây giờ xét đoạn kế tiếp, P3P4. Điểm P3 nằm bên trái cửa sổ, vì vậy ta xác định giao điểm P'3 và loại bỏ đoạn từ P'3 đến P3. Bằng cách kiểm tra mã vùng phần đoạn thẳng từ P'3 đến P4, chúng ta thấy rằng phần còn lại này nằm phía dưới cửa sổ và cũng bị vứt bỏ luôn.

Các giao điểm với biên cửa sổ có thể được tính bằng cách dùng các tham số của phương trình đường thẳng. Với một đường thẳng đi qua hai điểm (x_1, y_1) và (x_2, y_2), tung độ y của giao điểm với một biên dọc cửa sổ có thể tính được theo phép tính:

$$y = y_1 + m\,(x - x_1) \qquad (4\text{-}4)$$

Ở đây giá trị x được đặt là x_{wmin} hoặc x_{wmax}, và độ dốc m được tính bằng là

$$m = (y_2 - y_1)/(x_2 - x_1)$$

Tương tự, nếu ta tìm giao điểm với biên ngang, hoành độ x có thể được tính như sau:

$$x = x_1 + (y - y_1)/m \qquad (4\text{-}5)$$

với y là y_{wmin} hoặc y_{wmax}.

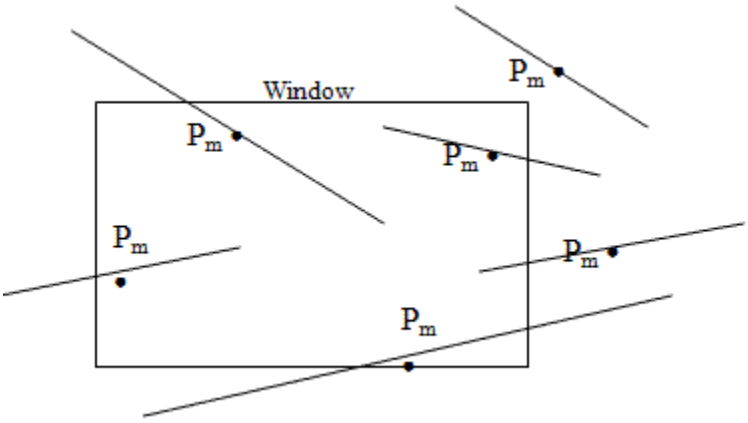
4.3.2 Giải thuật chia tại trung điểm

Một kỹ thuật để xác định giao điểm với biên cửa sổ mà không dùng đến phương trình đường thẳng là dùng thủ tục tìm kiếm nhị phân, được gọi là sự phân chia tại trung điểm. Đầu tiên, việc kiểm tra các đoạn một lần nửa được thực hiện bằng cách dùng mã vùng. Bất kỳ đoạn nào không được chấp nhận hoàn toàn hoặc không bị huỷ bỏ hoàn toàn (nhờ vào kiểm tra mã vùng) thì sẽ được đi tìm giao điểm bằng cách kiểm tra tọa độ trung điểm.

Tiếp cận này được minh họa trong hình dưới đây. Mọi đoạn thẳng với hai điểm đầu mút (x_1,y_1) và (x_2, y_2), trung điểm được tính như sau:

$$x_m = (x_1 + x_2) / 2; \qquad y_m = (y_1 + y_2) / 2 \qquad (4\text{-}6)$$

Mỗi kết quả tính toán cho tọa độ giao điểm liên quan đến một phép cộng và một phép chia 2. Khi tọa độ giao điểm được xác định, mỗi nửa đoạn thẳng được kiểm tra để chấp nhận hay huỷ bỏ toàn bộ. Nếu một nửa đoạn được chấp nhận hoặc bị huỷ bỏ, một nửa kia sau đó sẽ được xử lý theo cách tương tự. Điều này tiếp tục cho đến khi gặp một giao điểm. Nếu một nửa được chấp nhận hoặc bị huỷ bỏ toàn bộ, nửa kia tiếp tục được xử lý cho đến khi toàn bộ nó là bị huỷ bỏ hoặc được giữ lại. Cài đặt phần cứng theo phương pháp này có thể giúp ta clipping khỏi biên vùng quan sát nhanh chóng sau khi các đối tượng vừa được chuyển sang hệ tọa độ thiết bị.



4.3.3 Giải thuật Liang – Barsky

Giải thuật được dựa trên phân tích phương trình tham số đoạn thẳng:

$$\begin{aligned} x &= x_1 + t(x_2 - x_1) = x_1 + tD_x \\ y &= y_1 + t(y_2 - y_1) = y_1 + tD_y \qquad , 0 \le t \le 1 \end{aligned} \qquad (4\text{-}7)$$

Ứng với mỗi giá trị t, ta có một điểm P tương ứng thuộc đường thẳng. Tập hợp các điểm thuộc phần giao của đoạn thẳng và cửa sổ thỏa mãn hệ bất phương trình:

$$\begin{cases} x_{wmin} \le x_1 + tD_x \le x_{wmax} \\ y_{wmin} \le y_1 + tD_y \le y_{wmax} \\ 0 \le t \le 1 \end{cases} \qquad (4\text{-}8)$$

Đặt $p_1 = -D_x$, $q_1 = x_1 - x_{wmin}$

$P_2 = D_x$, $q_2 = x_{wmax} - x_1$

$p_3 = -D_y$, $q_3 = y1 - y_{wmin}$

$p_4 = D_y$, $q_4 = y_{wmax} - y_1$

Hệ bất pt 4-8 có dạng:

$$\begin{cases} p_k t \le q_k \\ 0 \le t \le 1 \end{cases}, \quad k \in \{1,2,3,4\} \qquad (4\text{-}9)$$

Như thế, tìm đoạn giao thực chất là tìm nghiệm của hệ bất pt 4-9. Có các trường hợp sau:

- Nếu $\exists k \in \{1,2,3,4\} : p_k = 0$ và $q_k < 0$ thì hệ vô nghiệm.
- Nếu $\exists k \in \{1,2,3,4\} : p_k \ne 0$ hay $q_k \ge 0$ thì với các bất pt ứng với $p_k = 0$ luôn đúng.

Với $p_k < 0$: ta có $t \ge \frac{q_k}{p_k}$, Với $p_k > 0$: ta có $t \le \frac{q_k}{p_k}$

Vậy nghiệm của hệ 4-9 là $[t_1, t_2]$ thỏa mãn:

$$\begin{cases} t_1 = \max(\{ \frac{q_k}{p_k},\ p_k < 0 \}, 0) \\ t_2 = \min(\{ \frac{q_k}{p_k},\ p_k > 0 \}, 1) \\ t_1 \le t_2 \end{cases}$$

Nếu hệ 4-9 có nghiệm thì đoạn giao giữa đoạn thẳng và cửa sổ là $Q_1(x_1 + t_1 D_x , y_1 + t_1 D_y)$ và $Q_2 (x_1 + t_2 D_x , y_1 + t_2 D_y)$

4.4 Bài tập áp dụng

Bài 1 Ý nghĩa mã vùng trong giải thuật Cohen – Sutherland.

Gợi ý: mã vùng để mô tả vị trí tương đối của vùng nào đó so với cửa sổ.

Bài 2 So sánh hai thuật toán Cohen – Sutherland và Liang – Barsky về số phép tính thực hiện trong các trường hợp chính.

Gợi ý: Giải thuật Liang – Barsky thực hiện ít phép toán hơn.

Bài 3 Cài đặt thuật toán cắt xén hình chữ nhật với 1 đoạn thẳng cho trước theo giải thuật Liang – Barsky hay Cohen – Sutherland.

Gợi ý: Sử dụng giải thuật Liang – Barsky.

Bài 4 Cho biết ma trận của phép biến đổi từ cửa sổ sang vùng quan sát.

Gợi ý: ma trận biến đổi là tích ba ma trận tịnh tiến, tỷ lệ, tịnh tiến.

CHƯƠNG V ĐỒ HỌA 3 CHIỀU

5.1 Tổng quan về đồ họa ba chiều

Khi mô hình hóa và hiển thị một đối tượng ba chiều, ta cần mô tả thông tin cho đối tượng. Các công cụ hỗ trợ đồ họa cung cấp một số hàm hiển thị các thành phần bên trong, những nét tiêu biểu hay một phần của đối tượng. Ngoài ra, các phép biến đổi được dùng đa dạng hơn so với đồ họa hai chiều vì phải chọn nhiều tham số mô tả đối tượng.

Mô tả một đối tượng ba chiều phải qua quy trình xử lý gồm nhiều công đoạn như phép biến đổi hệ tọa độ quan sát và phép chiếu phối cảnh , chuyển đổi từ hệ tọa độ quan sát ba chiều sang hệ tọa độ quan sát hai chiều. Những phần nhìn thấy trong hệ tọa độ quan sát được chọn, xác định và cuối cùng, giải thuật vẽ bề mặt của đối tượng được áp dụng nhằm tạo ra hình ảnh thực tế của đối tượng ba chiều.

5.1.1 Quy trình hiển thị đồ họa ba chiều

Đối tượng được mô tả trong không gian (x, y, z). Có hai dạng mô hình hóa là mô hình thể hiện vật thể (solid) hay bề mặt (boundaries) của đối tượng.

Các mô hình được biểu diễn trong hệ tọa độ cục bộ, hệ tọa độ này chỉ định nghĩa cho đối tượng nên gốc tọa độ và đơn vị đo được chọn sao cho phù hợp với đối tượng.
Bước đầu tiên trong quy trình hiển thị là biến đổi đối tượng từ không gian đối tượng sang không gian thực. Trong không gian này thì đối tượng, nguồn sáng, người quan sát cùng tồn tại. Quá trình này gọi là biến đổi mô hình.

Tiếp theo, ta loại bỏ các phần của đối tượng không nhìn thấy, giúp giảm bớt những thao tác dư thừa của hình ảnh sẽ hiển thị.

Bước tiếp theo là phải chiếu sáng đối tượng , gán cho chúng màu sắc dựa vào đặc điểm hình thành vật.

Sau khi chiếu sáng, phải thực hiện một phép biến đổi hệ tọa độ đặt vị trí quan sát về gốc tọa độ và mặt phẳng quan sát tại vị trí phù hợp. Các đối tượng được đưa về không gian quan sát.

Sau đó, chiếu đối tượng xuống mặt phẳng hai chiều, biến đổi từ không gian quan sát sang không gian thiết bị màn hình. Đối tượng xem như tập hợp các điểm, toàn cảnh đối tượng được hiển thị lên màn hình.

5.1.2 Mô hình hóa đối tượng

Một phương pháp thông dụng để mô hình hóa đối tượng là mô hình khung nối kết. Mô hình khung nối kết gồm tập đỉnh và tập cạnh nối các đỉnh. Khi thể hiện mô hình, các đối tượng là rỗng và không giống thực tế. Để hoàn thiện, ta tạo màu sắc, độ bóng bề mặt, loại bỏ các mặt đường không nhìn thấy.

Hình dạng của đối tượng ba chiều được biểu diễn trên hai danh sách: danh sách đỉnh và danh sách cạnh. Danh sách đỉnh cho thông tin hình học là vị trí các đỉnh, còn danh sách cạnh xác định thông tin kết nối. Có nhiều cách đặc tả mô hình khung nối kết như dùng mảng, xâu … Ở đây, ta biểu diễn mô hình trên theo cấu trúc dữ liệu mảng như sau:

```
typedef struct
{
  float x, y, z;
```

```
}Point3d;
typedef struct
{
  int  NumVertext;
  int  NumEdge;
 Point3d  Vert[50];
  int    Edge[100][2];
}Wireframe;
```

Ngoài ra, đôi khi ta mô tả các mặt phẳng của đối tượng. Mỗi mặt được định nghĩa bởi một đa giác bao. Ví dụ đối tượng hình lập phương có 6 mặt và danh sách đỉnh, cạnh biểu diến mô hình khung nối kết của nó:

Danh sách đỉnh				
Chỉ số	x	y	z	
1	0	0	0	Mặt sau
2	0	0	1	
3	0	1	0	
4	0	1	1	
5	1	0	0	Mặt Trước
6	1	0	1	
7	1	1	0	
8	1	1	1	

Danh sách cạnh		
Cạnh	Đỉnh đầu	Đỉnh
1	1	2
2	1	3
3	2	3
4	1	5
5	2	6
6	3	4
7	3	7
8	4	8
9	5	6
10	5	7
11	2	4
12	7	8

5.2 Biểu diễn đối tượng ba chiều

Các cảnh đồ họa được biểu diễn theo những phương pháp khác nhau sao cho phù hợp với thuộc tính đối tượng. Các mặt đa giác và mặt bậc hai cung cấp cho chúng ta mô tả gần đúng của các đối

tượng elip, hyperbol, …cách tiếp cận thủ tục như Fractal cho phép ta biểu diễn chính xác các đối tượng tự nhiên như bầu trời, không gian thực tế.

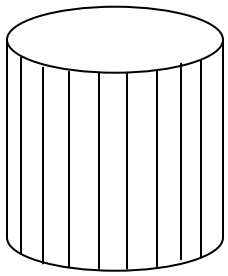
Sơ đồ biểu diễn một đối tượng chia thành 2 dạng chính: Phương pháp biểu diễn bề mặt B – reps và phương pháp biểu diễn theo phân hoạch không gian.

5.2.1 Biểu diễn mặt đa giác

Mô hình hóa bề mặt đối tượng thường được biểu diễn đối tượng, thông qua tập hợp các mặt đa giác xác định bề mặt đối tượng. Với cách biểu diễn trên, ta đơn giản hóa cách lưu trữ dữ liệu và tăng tốc độ hiển thị đối tượng vì bề mặt được mô tả bằng phương trình tuyến tính. Vì nguyên nhân trên mà mô tả các đối tượng thông qua mặt đa giác được dùng cho các đối tượng đồ họa cơ sở.

Trong một vài trường hợp, ta chỉ có thể chọn lựa cách biểu diễn đa giác, việc biểu diễn như thế cung cấp một định nghĩa chính xác về đặc tính của đối tượng. Ví dụ như biểu diễn một hình trụ thì xem như hình trụ là tập các mặt đa giác ghép liền. Nếu cần thể hiện hình trụ thực hơn thì phải biến đổi tạo bóng nội suy cho hình trụ.

Hình 5.1 Mô hình biểu diễn hình trụ thông qua mặt đa giác.



5.2.2 Đường cong và mặt cong , đường cong và mặt cong Bezier, B-Spline

Hình ảnh các đường cong và mặt cong có thể được tạo ra từ một tập các hàm toán học định nghĩa các đối tượng hoặc tập các điểm trên đối tượng. Đối với các đối tượng hình học như hình tròn hay elip thì thư viện đồ họa đã cung cấp sẵn hàm vẽ đối tượng lên mặt phẳng hiển thị. Hình biểu diễn đường cong là tập các điểm dọc theo hình chiếu của đường mô tả bởi hàm số. Nhưng với các đường cong hay mặt cong không có quy luật, thì tập điểm hay lưới đa giác xấp xỉ với đường mặt cong sẽ tạo ra. Hệ đồ họa hay tạo các lưới tam giác để đảm bảo tính đồng phẳng của các cạnh.

Một đường cong hay mặt cong có thể diễn tả bằng phương trình tham số hay không có tham số, tuy nhiên cách biểu diễn thứ nhất được áp dụng trong các bài toán đồ họa sẽ thuận tiện hơn cách kia.

Những đối tượng hình học như hình tròn, elip hình trụ, hình nón, hay hình cầu … được xem là những đường và mặt cong xác định được phương trình tham số chính xác cho nó. Do đó, mọi hệ đồ họa đều hỗ trợ trực tiếp biểu diễn các đối tượng hình học trên. Tuy nhiên, trong thực tế, có những đối tượng hình học không có quy luật chính xác để mô tả những đối tượng như mặt cong của thanh chắn va đập của xe ô tô, hình thân tàu thủy, tay cầm tách chén cafe… Ngoài ra, cũng không thể mô tả đối tượng hình học thông qua phép nội suy.

Yêu cầu đặt ra là: Có một đường cong, xác định theo tập các điểm phân biệt p_1, p_2, .. p_k . Hãy tìm ra giải thuật tạo đường cong ban đầu với độ chính xác nào đó?

Có 2 cách giải quyết:

- Thứ nhất, định vị tọa độ các điểm đã biết thuộc đường cong, tìm ra phương trình tham số để nó đi qua các điểm đó và trùng khớp hình dáng với đường cong ban đầu.

- Thứ hai, xác định một số điểm điều khiển, và tìm giải thuật tạo đường cong dựa trên các điểm đó. Ta phải thay đổi vị trí điểm điều khiển cho tới khi đường cong mới có hình dáng giống như hình dáng đường cong ban đầu thì xem như giải quyết xong.

Trong phần này, chúng ta sẽ giới thiệu phương trình đường cong Bezier và B – Spline, chúng dựa trên cách giải quyết thứ hai.

Đường cong Bezier: đường cong dựa trên L+1 điểm điều khiển p_0, p_1, .. p_L được biểu diễn qua công thức sau đây:

$$P(t) = \sum_{k=0}^{L} P_k B_k^L(t) \qquad (5\text{-}1)$$

$B_k^L(t)$ là đa thức Berntein được xác định theo: $B_k^L(t) = C_L^k (1-t)^{L-k} t^k$ (5-2)

C_L^k là tổ hợp chọn k từ L. Từ công thức 5-2, dẫn đến $\sum_{k=0}^{L} B_k^L(t) = 1$

Đường cong phức tạp thì yêu cầu số lượng điểm điều khiển nhiều, tuy nhiên trong trường hợp đó khiến bậc đa thức càng cao, đòi hỏi tính toán dài và khó. Cách khắc phục là đường cong phức tạp vẫn được ghép từ những đoạn khác nhau, mỗi đoạn xấp xỉ theo một đường cong Bezier bậc nhỏ hơn.

Việc tạo các đường cong phức tạp, bằng cách ghép các đoạn ngắn cũng cho phép kiểm soát sự thay đổi một phần đường cong, trong khi các đoạn khác thuộc tập điểm điều khiển khác nên không bị ảnh hưởng.

Đường cong B – Spline: đường cong cấp m dựa trên véc tơ nút T và L+1 điểm kiểm soát p_k có dạng:

$$P(t) = \sum_{k=0}^{L} P_k N_{k,m}(t) \qquad (5\text{-}3)$$

Trong đó $N_{k,m}(t)$ là đa thức bậc m-1 như sau

$$N_{k,m}(t) = \left(\frac{t - t_k}{t_{k+m-1} - t_k}\right) N_{k,m-1}(t) + \left(\frac{t_{k+m} - t}{t_{k+m} - t_{k+1}}\right) N_{k+1,m-1}(t) \qquad (5\text{-}4)$$

k = 0,1,...,L; $N_{k,1}(t) = 1$ nếu $t_k \le t < t_{k+1}$; $N_{k,1}(t) = 0$ trong các trường hợp khác.

Các khoảng cách $[t_i, t_{i+1}]$ có thể chia đều nhau hoặc theo định nghĩa khác.

Mặt cong Bezier: Xét đường cong Bezier như hàm theo tham số v và các điểm kiểm soát theo tham số u. Công thức cho đường cong sẽ là:

$$P(u, v) = \sum_{k=0}^{L} P_k(u) B_k^L(v)$$

Lúc này, khi u thay đổi kéo theo các điểm kiểm soát thay đổi dẫn tới đường cong thay đổi. Sự biến thiên của đường cong Bezier trong không gian tạo ra mặt cong. Nếu u thay đổi, các điểm

kiểm soát cũng thay đổi trên một đường cong nào đó xem như đường cong Bezier, dựa trên M+1 điểm kiểm soát.

Phương trình mặt cong Bezier là: $P(u,v)=\sum_{i=0}^{M}\sum_{k=0}^{L}P_{i,k}B_i^M(u)B_k^L(v)$ (5-5)

Mặt cong B – Spline: Cũng dựa trên phân tích trên, ta có phương trình mặt cong B – Spline là:

$$P(u,v)=\sum_{i=0}^{M}\sum_{k=0}^{L}P_{i,k}N_{i,m}(u)N_{k,m}(v) \quad (5\text{-}6)$$

Một vấn đề nảy sinh khi tạo các đường, mặt cong Bezier hay B – Spline là nếu đường, mặt cong phức tạp thì làm cách nào để đường cong liên tục tại các điểm điều khiển nối, hay các mảnh ghép tạo mặt cong được trơn tru tại các đường biên. Nguyên nhân là do mỗi phần đường cong, mỗi mảnh ghép mặt cong được kiểm soát bằng tập điểm khác nhau. Dựa vào đặc điểm liên tục bậc nhất, bậc hai, … của các điểm nối sẽ khiến đường cong, mặt cong được liên tục và trơn tru tại các biên.

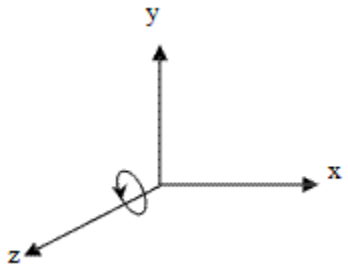
5.3 Các phép biến đổi hình học 3 chiều

Các điểm trong không gian ba chiều được biểu diễn bằng hệ trục tọa độ ba chiều, có thể xem là mở rộng của hệ trục tọa độ hai chiều. Trong thế giới hai chiều, mặt phẳng xy chứa toàn bộ đối tượng. Trong thế giới ba chiều, một trục z vuông góc được đưa ra để tạo thêm hai mặt phẳng chính khác là xz và yz.

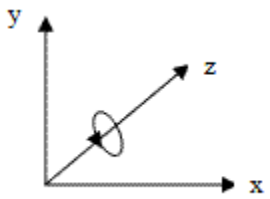
Chiều của các trục tọa độ trong hệ trục tọa độ ba chiều có thể tuân theo quy tắc bàn tay phải hay quy tắc bàn tay trái.

Hệ trục tọa độ tuân theo quy tắc bàn tay phải được mô tả bằng bàn tay phải, với ngón tay cái hướng theo trục z, các ngón tay còn lại xoắn theo chiều từ trục x dương sang trục y dương. Hệ trục tọa độ tuân theo quy tắc bàn tay trái được mô tả bằng bàn tay trái, nếu đặt bàn tay trái sao cho các ngón tay uốn cong theo chiều từ trục x dương tới trục y dương, thì ngón tay cái sẽ chỉ theo chiều trục z dương.

Hình 5.2 Các hệ trục tọa độ



Hệ trục tọa độ theo quy tắc bàn tay phải



Hệ trục tọa độ theo quy tắc bàn tay trái

Hệ tọa độ thuần nhất: Mỗi điểm (x,y,z) trong không gian Descartes được biểu diễn bởi một bộ bốn tọa độ trong không gian 4 chiều thu gọn (hx,hy,hz,h). Người ta thường chọn h=1.

Các phép biến đổi tuyến tính là tổ hợp của các phép biến đổi sau : tỉ lệ, quay, biến dạng và đối xứng. Các phép biến đổi tuyến tính có các tính chất sau:

- Gốc tọa độ là điểm bất động
- Ảnh của đường thẳng là đường thẳng
- Ảnh của các đường thẳng song song là các đường thẳng song song
- Bảo toàn tỉ lệ khoảng cách
- Tổ hợp các phép biến đổi có tính phân phối.

5.3.1 Phép biến đổi tỷ lệ

Phép biến đổi tỷ lệ tạo thành bằng cách gán các giá trị cho đường chéo chính của ma trận biến hình tổng quát 4×4 . Một điểm P(x, y, z) được biến đổi tỷ lệ thành P(x', y', z') bằng phép biến đổi sau:

$$[x'\ \ y'\ \ z'\ \ 1] = [x\ \ y\ \ z\ \ 1]\begin{bmatrix} A & 0 & 0 & 0 \\ 0 & E & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}1)$$

Phép biến đổi được coi là phép tỷ lệ theo gốc tọa độ. Nếu các hệ số A, E khác nhau thì hình ảnh của đối tượng sẽ biến dạng. Ngược lại, thì kích thước đối tượng có thể thay đổi nhưng tỷ lệ với gốc tọa độ vẫn giữ nguyên.

5.3.2 Phép biến dạng

Phép biến dạng ba chiều tạo ra sự biến dạng cho đối tượng bởi việc thay giá trị của một hoặc nhiều tọa độ bằng các hệ số tỷ lệ của cột thứ ba. Cách thực hiện trên là biến dạng theo mặt phẳng tạo ra bởi hai trục tọa độ được điều khiển bởi trục thứ ba. Các thành phần ngoài đường chéo chính của ma trận con 3×3 phía trên của ma trận biến hình tổng quát ảnh hưởng đến phép biến dạng.

Ma trận biến dạng theo trục X:
$$\begin{bmatrix} 1 & Sxy & Sxz & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}2)$$

Ma trận biến dạng theo trục Y:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ Syx & 1 & Syz & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}3)$$

Ma trận biến dạng theo trục Z:
$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ Szx & Szy & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}4)$$

5.3.3 Phép tịnh tiến

Ma trận biến đổi hình sẽ thay đổi điểm P(x, y, z) thành P(x', y', z') bằng phép biến đổi sau:

$$[x'\ \ y'\ \ z'\ \ 1] = [x\ \ y\ \ z\ \ 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ J & K & L & 1 \end{bmatrix} \qquad (5\text{-}5)$$

Các giá trị J, K, L mô tả sự tịnh tiến tương đối theo các hướng x, y, z.

5.3.4 Phép quay hình

Phép quay trong không gian là quan trọng để tìm hiểu hình dạng của đối tượng hoặc kiểm tra quá trình thiết kế ở các góc độ khác nhau. So với phép quay hai chiều, phép quay ba chiều khó hơn vì phải sử dụng trục quay thay thế điểm quay. Các phép quay được phân tích dựa vào các phép quay quanh các trục chính, trong trường hợp này xem hệ trục tọa độ theo quy tắc bàn tay phải.

Ma trận quay quanh trục X: $$[Tr]^{\theta}_{X} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}6)$$

Ma trận quay quanh trục Y: $$[Tr]^{\theta}_{Y} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}7)$$

Ma trận quay quanh trục Z: $$[Tr]^{\theta}_{Z} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (5\text{-}8)$$

Đối với phép quay trong hệ trục tọa độ theo quy tắc bàn tay trái, chỉ cần thay đổi hệ số góc quay θ thành – θ là xong.

5.4 Bài tập áp dụng

Bài 1 Các kiến thức mô tả trong chương xét trong hệ tọa độ Đề các bàn tay trái hay bàn tay phải.

Gợi ý: quy tắc bàn tay phải.

Bài 2 Xác định ma trận của phép đối xứng qua trục X, Y, Z.

Gợi ý: phép đối xứng qua trục X, Y, Z : tọa độ tương ứng với trục nào thì lấy đảo dấu.

Bài 3 Xác định điểm Q là ảnh của điểm P cho trước phép biến dạng theo trục tung, với các hệ số biến dạng là 2.

Gợi ý: tọa độ điểm Q xác định từ phép nhân véc tơ tọa độ P với ma trận biến dạng.

Bài 4 Xác định ma trận của phép quay góc 90^o quanh trục, đi qua 2 điểm $P_o(0, 0)$ P_1 (10, -10)

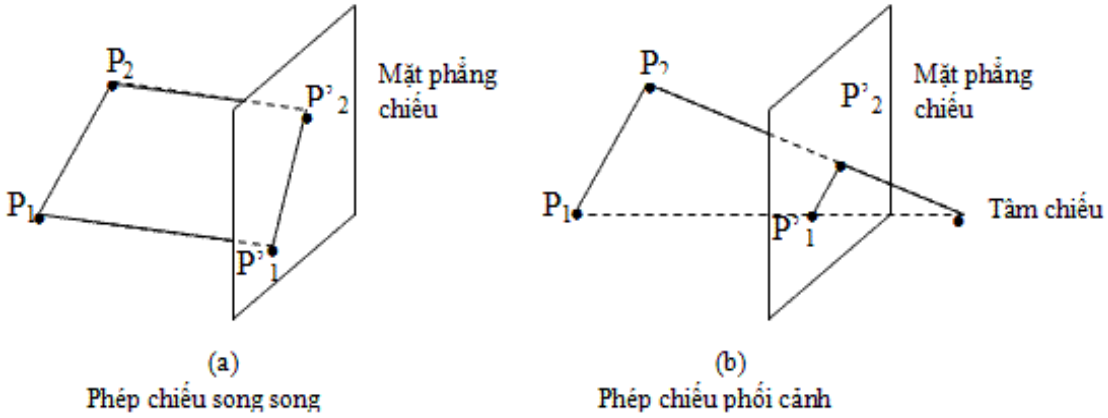
Gợi ý: Ma trận biến đổi là chuỗi các biến đổi: quay quanh X $M_{rot}(X, \alpha)$, quay quanh trục Y $M_{rot}(Y, -\beta)$, quay quanh trục Z $M_{rot}(Z, 90^o)$, quay quanh trục Y $M_{rot}(Y, \beta)$, quay quanh X $M_{rot}(X, -\alpha)$.

CHƯƠNG VI QUAN SÁT 3 CHIỀU

6.1 Các phép chiếu

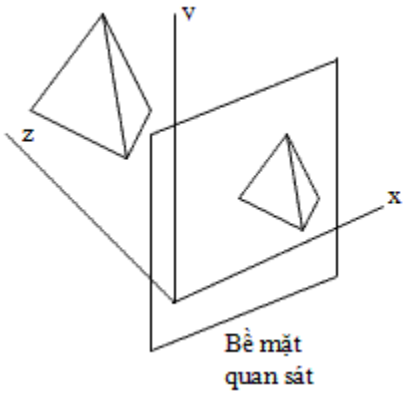
Trong đồ họa hai chiều, các thao tác quan sát biến đổi các điểm hai chiều trong mặt phẳng tọa độ thực thành các điểm hai chiều trong mặt phẳng hệ tọa độ thiết bị. Sự định nghĩa đối tượng, bị cắt bởi một cửa sổ, được ánh xạ vào một vùng quan sát. Các hệ tọa độ thiết bị chuẩn hóa này sau đó được biến đổi sang các hệ tọa độ thiết bị, và đối tượng được hiển thị lên thiết bị kết xuất. Đối với đồ họa ba chiều, việc làm này phức tạp hơn một chút, vì bây giờ có vài chọn lựa để có thể quan sát ảnh như thế nào. Chúng ta có thể quan sát ảnh từ phía trước, từ phía trên, hoặc từ phía sau. Hoặc chúng ta có thể tạo ra quang cảnh về những gì chúng ta có thể thấy nếu chúng ta đang đứng ở trung tâm của một nhóm các đối tượng. Ngoài ra, sự mô tả các đối tượng ba chiều phải được chiếu lên bề mặt quan sát của thiết bị xuất. Trong chương này, trước hết chúng ta sẽ thảo luận các cơ chế của phép chiếu. Sau đó, các thao tác liên quan đến phép biến đổi cách quan sát, và đầy đủ các kỹ thuật quan sát ảnh ba chiều sẽ được phát triển.

Có hai phương pháp cơ bản để chiếu các đối tượng ba chiều lên bề mặt quan sát hai chiều. Tất cả các điểm của đối tượng có thể được chiếu lên bề mặt theo các đường thẳng song song, hoặc các điểm có thể được chiếu theo các đường hội tụ về một điểm được gọi là tâm chiếu. Hai phương pháp này được gọi là phép chiếu song song và phép chiếu phối cảnh.



Hình 6.1 Hai phương pháp chiếu một đoạn thẳng lên bề mặt của mặt phẳng chiếu

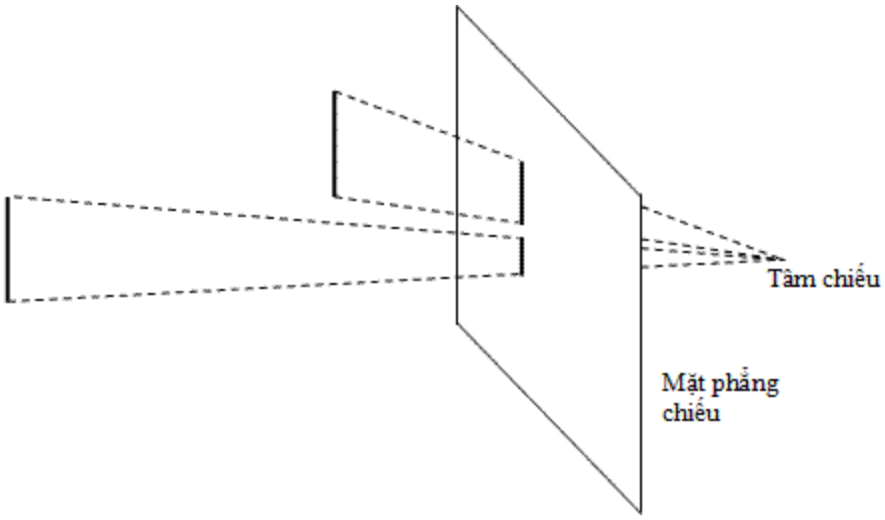
Trong cả hai trường hợp, giao điểm của đường chiếu với bề mặt quan sát xác định các tọa điểm của điểm được chiếu lên mặt phẳng chiếu này.



Hình 6.2 Một bề mặt quan sát được định nghĩa trong mặt z=0 của hệ tọa độ bàn tay trái.

Chúng ta giả sử rằng mặt phẳng chiếu là mặt z = 0 của hệ tọa độ bàn tay trái. Phép chiếu song song bảo tồn mối quan hệ về chiều của các đối tượng, và đây là kỹ thuật được dùng trong việc phác thảo để tạo ra các bức vẽ tỷ lệ của các đối tượng ba chiều. Phương pháp này được dùng để thu các hình

ảnh chính xác ở các phía khác nhau của một đối tượng. Tuy nhiên, phép chiếu song song không cho một hình ảnh thực tế của các đối tượng ba chiều. Ngược lại, phép chiếu phối cảnh tạo ra các hình ảnh thực nhưng không bảo tồn các chiều liên hệ. Các đường ở xa được chiếu sẽ nhỏ hơn các đường ở gần mặt phẳng chiếu, như trong hình

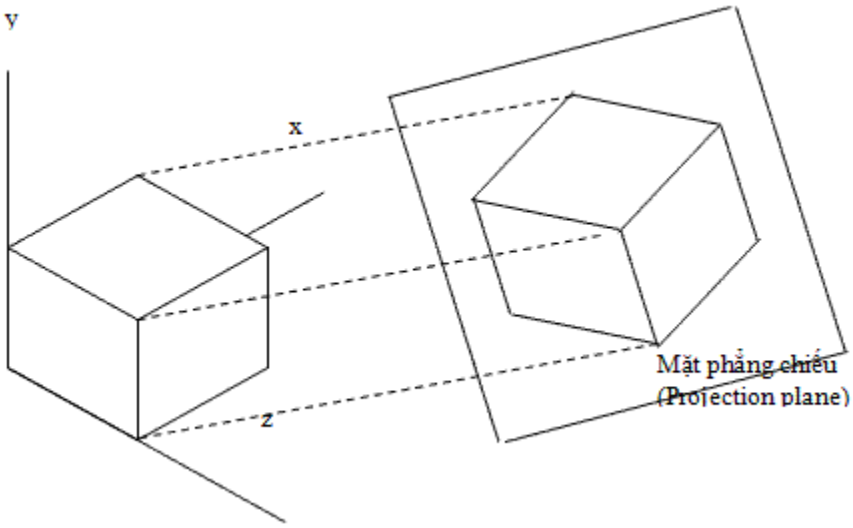


Hình 6.3 Hai đoạn thẳng dài bằng nhau, trong phép chiếu phối cảnh, đoạn nào ở xa mặt phẳng chiếu hơn sẽ có kích thước nhỏ

Hai đoạn thẳng dài bằng nhau, trong phép chiếu phối cảnh, đoạn nào ở xa mặt phẳng chiếu hơn sẽ có kích thước nhỏ.

6.1.1 Phép chiếu song song

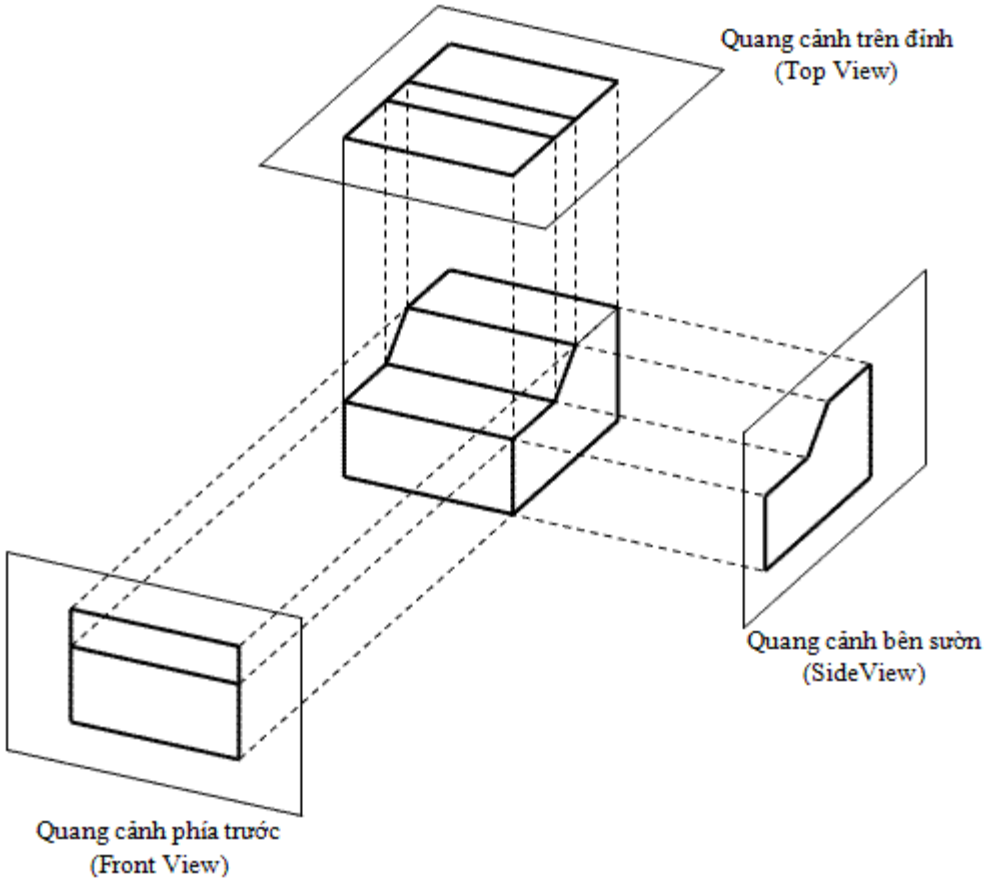
Các hình ảnh được hình thành bằng phép chiếu song song có thể được xác định dựa vào góc hợp bởi hướng của phép chiếu hợp với mặt phẳng chiếu. Khi hướng của phép chiếu vuông góc với mặt phẳng, ta có phép chiếu trực giao (hay phép chiếu vuông góc - orthographic projection). Một phép chiếu có thể không vuông góc với mặt phẳng chiếu được gọi là phép chiếu xiên.



Hình 6.4 Phép chiếu cùng kích thước của một đối tượng lên bề mặt quan sát

Các phép chiếu trực giao đa số được dùng để tạo ra quang cảnh nhìn từ phía trước, bên sườn, và trên đỉnh của đối tượng. Quang cảnh phía trước, bên sườn, và phía sau của đối tượng được gọi là "mặt chiếu", và quang cảnh phía trên được gọi là "mặt phẳng" . Các bản vẽ trong kỹ thuật thường dùng các phép chiếu trực giao này, vì các chiều dài và góc miêu tả chính xác và có thể đo được từ bản vẽ. Chúng ta cũng có thể xây dựng các phép chiếu trực giao để có thể quan sát nhiều hơn một mặt của một đối tượng. Các quang cảnh như thế được gọi là các phép chiếu trực giao trục lượng học. Hầu hết phép chiếu trục lượng học được dùng là phép chiếu cùng kích thước. Một phép chiếu cùng kích thước được thực hiện bằng việc sắp xếp

song song mặt phẳng chiếu mà nó cắt mỗi trục tọa độ ở nơi đối tượng được định nghĩa (được gọi là các trục chính) ở các khoảng cách như nhau từ ảnh gốc. Hình trình bày phép chiếu cùng kích thước. Có tám vị trí, một trong tám mặt, đều có kích thước bằng nhau. Tất cả ba trục chính được vẽ thu gọn bằng nhau trong phép chiếu cùng kích thước để kích thước liên hệ của các đối tượng được bảo tồn. Đây không là trường hợp phép chiếu trực giao trục lượng học tổng quát, khi mà các hệ số tỷ lệ theo ba trục chính có thể khác nhau.

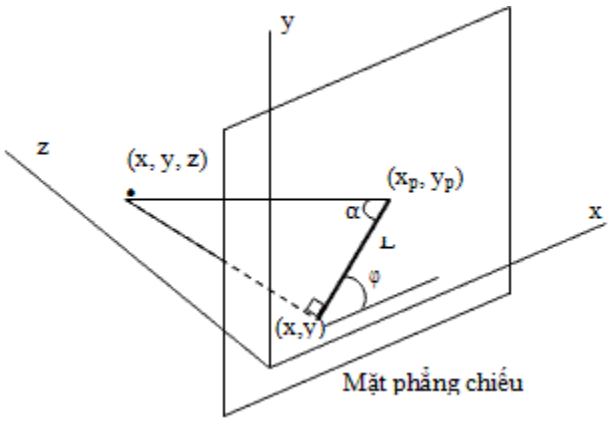


Hình 6.5 Ba phép chiếu trực giao của một đối tượng.

Các phương trình biến đổi để thực hiện một phép chiếu song song trực giao thì dễ hiểu. Đối với điểm bất kỳ (x, y, z), điểm chiếu (x_p, y_p, x_p) trên bề mặt chiếu được tính như sau:

$$x_p = x, \qquad y_p = y, \qquad z_p = 0 \qquad (6\text{-}1)$$

Một phép chiếu xiên đạt được bằng việc chiếu các điểm theo các đường thẳng song song, các đường thẳng này không vuông góc với mặt phẳng chiếu. Hình 5.6 trình bày hình chiếu xiên của điểm (x, y, z) theo một đường thẳng chiếu đến vị trí (x_p, y_p). Các tọa độ chiếu trực giao trên



mặt phẳng chiếu là (x, y). Đường thẳng của phép chiếu xiên tạo một góc α với đường thẳng trên mặt phẳng chiếu (đây là đường nối điểm (xp, yp) với điểm (x, y)). Đường này, có chiều dài L, hợp một góc φ với phương ngang trên mặt phẳng chiếu. Chúng ta có thể diễn tả các tọa độ chiếu qua các số hạng x, y, L, và φ:

$$x_p = x + L\cos\varphi \qquad (6\text{-}2)$$
$$y_p = y + L\sin\varphi$$

Hình 6.6 Phép chiếu vuông góc của điểm (x, y, z) thành điểm (xp, yp) lên mặt phẳng chiếu

Phương chiếu có thể định nghĩa bằng việc chọn các giá trị cho góc α và φ. Các chọn lựa thông thường cho góc φ là 30^o và 45^o, là các góc hiển thị một quang cảnh của mặt trước, bên sườn, và trên đỉnh (hoặc mặt trước, bên sườn, và dưới đáy) của một đối tượng. Chiều dài L là một hàm của tọa dộ z, và chúng ta có thể tính tham số này từ các thành phần liên quan.

$\tan\alpha = \frac{z}{L} = \frac{1}{L_1}$, ở đây L1 là chiều dài của các đường chiếu từ (x, y) đến (x_p, y_p) khi z = 1. Từ phương trình , thu được $L = z\,L_1$ và các phương trình của phép chiếu xiên 6.2 có thể được viết lại như sau

$$x_p = x + z(L_1\cos\varphi) \qquad (6\text{-}3)$$
$$y_p = y + z(L_1\sin\varphi)$$

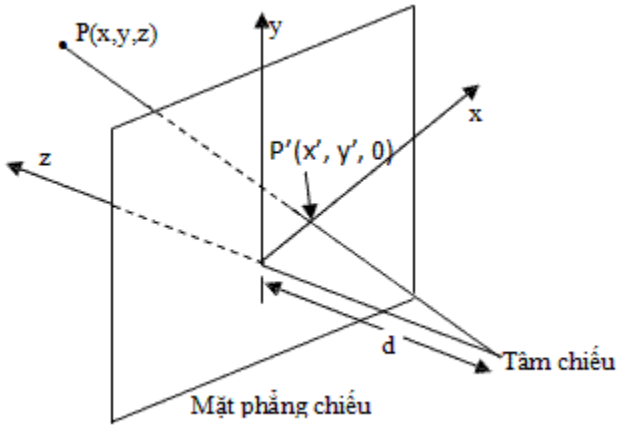
Ma trận biến đổi để tạo ra bất kỳ việc chiếu song song có thể được viết như sau:

$$[M]_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ L_1\cos\varphi & L_1\sin\varphi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (6\text{-}4)$$

Một phép chiếu trực giao có thể đạt được khi $L_1 = 0$ (xảy ra ở góc chiếu $\alpha=90^o$).Các phép chiếu xiên được sinh ra với giá trị L_1 khác không. Ma trận chiếu 5.4 có cấu trúc tương tự ma trận của phép làm biến dạng theo trục z. Thực tế, kết quả của ma trận chiếu này là làm biến dạng mặt phẳng của hằng z và chiếu chúng lên mặt phẳng quan sát. Các giá trị tọa độ x và y trong mỗi mặt của hằng z bị thay đổi bởi một hệ số tỷ lệ đến giá trị z của mặt phẳng để các góc, các khoảng cách, và các đường song song trong mặt phẳng được chiếu chính xác.

6.1.2 Phép chiếu phối cảnh

Để đạt được phép chiếu phối cảnh của đối tượng ba chiều, chúng ta chiếu các điểm theo đường thẳng chiếu để các đường này gặp nhau ở tâm chiếu. Trong dưới đây, tâm chiếu trên trục z và có giá trị âm, cách một khoảng d phía sau mặt phẳng chiếu. Bất kỳ điểm nào cũng có thể được chọn làm tâm của phép chiếu, tuy nhiên việc chọn một điểm dọc theo trục z sẽ làm đơn giản việc tính toán trong các phương trình biến đổi.



Hình 6.7 Phép chiếu phối cảnh của điểm P ở tọa độ (x, y, z) thành điểm (xp, yp, 0) trên mặt phẳng chiếu.

Chúng ta có thể đạt được các phương trình biến đổi cho phép chiếu phối cảnh từ các phương trình tham số mô tả các đường chiếu từ điểm P đến tâm chiếu. Các tham số xây dựng các đường chiếu này là

$$\begin{aligned} x' &= x - xu \\ y' &= y - yu \\ z' &= z - (z + d)u \end{aligned} \qquad (6\text{-}5)$$

Tham số u lấy giá trị từ 0 đến 1, và các tọa độ (x', y', z') thể hiện cho bất kỳ điểm nào dọc theo đường thẳng chiếu. Khi u = 0, điểm P ở tọa độ (x, y, z). Ở đầu mút kia của đường thẳng u =1, và chúng ta có các tọa độ của tâm chiếu, (0, 0, d). Để thu được các tọa độ trên mặt phẳng chiếu, chúng ta đặt z' = 0 và tìm ra tham số u:

$$u = \frac{z}{z+d} \qquad (6\text{-}6)$$

Giá trị của tham số u tạo ra giao điểm của đường chiếu với mặt phẳng chiếu tại (x', y', 0). Thế phương trình 5.6 vào phương trình 5.5, ta thu được các phương trình biến đổi của phép chiếu phối cảnh.

$$x' = x\frac{d}{z+d} = x\frac{1}{z/d+1} \quad ; \quad y' = y\frac{d}{z+d} = y\frac{1}{z/d+1} \quad ; \quad z' = 0 \qquad (6\text{-}7)$$

Dùng biểu diễn hệ tọa độ thuần nhất ba chiều, chúng ta có thể viết phép biến đổi phối cảnh theo hình thức ma trận:

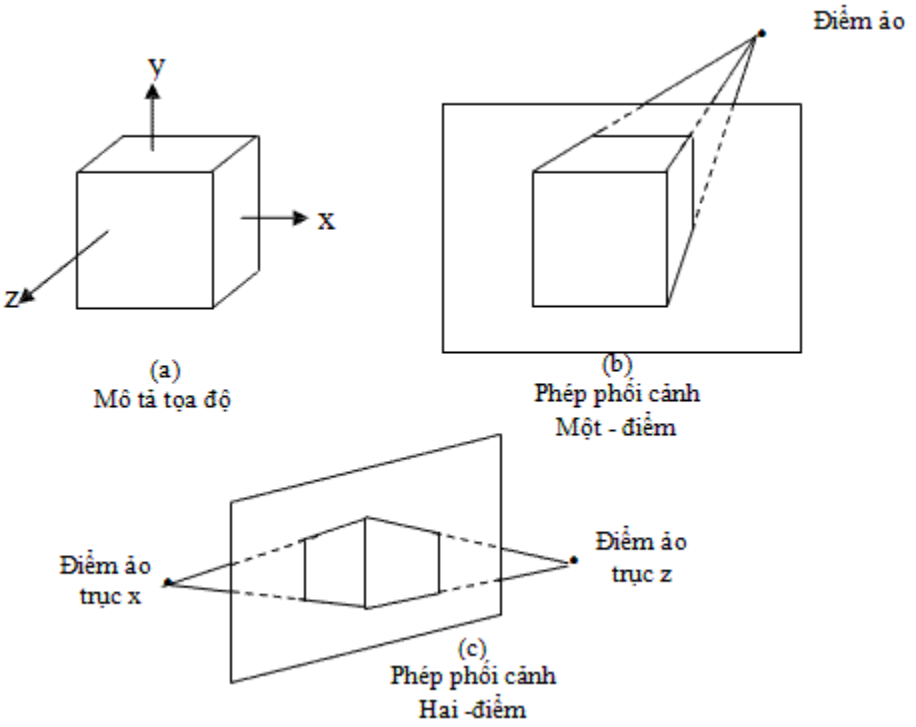
$$\begin{bmatrix} x_h & y_h & z_h & w \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (6\text{-}8)$$

Trong biểu diễn trên, $w = \frac{z}{d} + 1$, và các tọa độ chiếu trên mặt phẳng chiếu được tính từ các tọa độ thuần nhất như sau: $[x' \; y' \; z' \; 1] = [x_h/w \quad y_h/w \quad z_h/w \quad 1]$

6.2 Điểm tụ

Khi các đối tượng ba chiều được chiếu lên một mặt phẳng dùng các phương trình biến đổi phối cảnh, bất kỳ tập hợp các đường thẳng song song nào của đối tượng mà không song song với mặt phẳng chiếu được chiếu thành các đường hội tụ (đồng quy). Các đường thẳng song song với mặt phẳng khi chiếu sẽ tạo ra các đường song song. Điểm mà tại đó tập hợp các đường thẳng song song được chiếu xuất hiện hội tụ về đó được gọi là điểm tụ. Mỗi tập hợp các đường thẳng song song được chiếu như thế sẽ có một điểm tụ riêng .

Hình 6.8 Các quang cảnh phối cảnh của một hình lập phương.



Điểm tụ cho bất kỳ tập các đường thẳng, tức các đường song song với một trong các trục tọa độ thực được nói đến như một điểm tụ chính. Chúng ta quản lý số lượng các điểm tụ chính (một, hai, hoặc ba) với hướng của mặt phẳng chiếu, và các phép chiếu phối cảnh được phân loại dựa vào đó để có các phép chiếu: một-điểm, hai-điểm, hoặc ba-điểm. Số lượng các điểm tụ chính trong một phép chiếu được xác định bởi số lượng các trục của hệ tọa độ thực cắt mặt phẳng chiếu. Hình trên minh họa hình ảnh của các phép chiếu phối cảnh một-điểm và hai-điểm của hình lập phương.

Trong hình 6.8(b), mặt phẳng chiếu có phương song song với mặt xy để chỉ có trục z bị cắt. Phương này tạo ra phép chiếu phối cảnh một-điểm với một điểm tụ trên trục z. Với quang cảnh trong hình 6.8(c), mặt phẳng chiếu cắt cả hai trục x và z nhưng không cắt trục y. Kết quả, phép chiếu phối cảnh hai-điểm này chứa cả hai điểm tụ: trên trục x và trên trục z.

6.3 Loại bỏ mặt khuất

Một vấn đề cần được quan tâm đến trong việc tạo ra các hình ảnh thực là sự xác định và xóa bỏ các phần của đối tượng hình học mà ta không nhìn thấy được từ một vị trí quan sát. Có nhiều tiếp cận chúng ta cần để giải quyết vấn đề này, và cũng có nhiều thuật toán khác nhau đã và đang được phát triển để xóa bỏ các phần bị che khuất một cách hiệu quả cho những loại ứng dụng khác nhau. Có phương pháp tốn bộ nhớ, một số khác cần nhiều thời gian xử lý hay chỉ áp dụng được cho những kiểu đối tượng đặc biệt.

Các thuật toán về đường khuất và mặt khuất dựa vào xử lý trực tiếp định nghĩa đối tượng hay xử lý hình chiếu của các đối tượng đó. Hai tiếp cận này được gọi là các phương pháp không gian đối tượng và các phương pháp không gian ảnh. Phương pháp không gian đối tượng xác định các thành phần của đối tượng được nhìn thấy bằng cách sử dụng các quan hệ hình học và không gian. Nó thực hiện với độ chính xác từ dữ liệu mô tả đối tượng. Trong thuật toán không gian ảnh, tính chất nhìn thấy được của một điểm được quyết định bởi điểm ở vị trí pixel trên mặt phẳng chiếu. Hầu hết các thuật toán khử mặt khuất dùng phương pháp không gian ảnh, tuy nhiên các phương pháp không gian đối tượng vẫn có thể được dùng một cách hiệu quả cho một số trường hợp. Các thuật toán khử đường khuất hầu hết dùng phương pháp không gian đối tượng, dù rằng nhiều thuật toán khử mặt khuất không gian ảnh có thể dễ dàng được chỉnh sửa cho việc khử đường khuất. Trong chương này, chúng ta khảo sát một vài trong số các phương pháp phổ biến để xóa bỏ các đường khuất và mặt khuất.

6.3.1 Phương pháp mặt sau

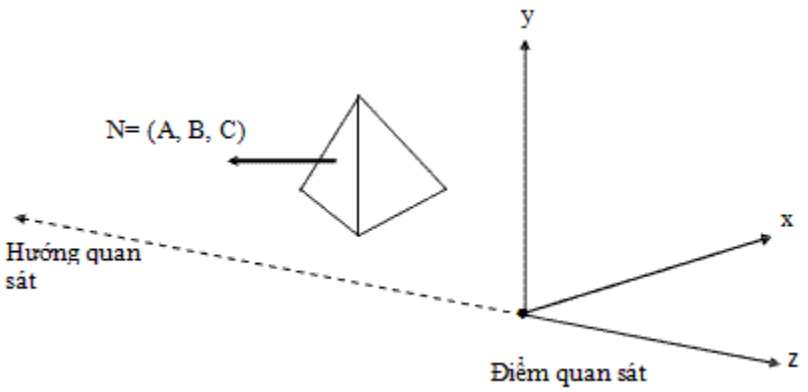
Một phương pháp không gian đối tượng đơn giản là phương pháp mặt sau, dựa vào các phương trình mặt phẳng:

$$Ax + By + Cz + D = 0 \qquad (6\text{-}9)$$

Bất kỳ điểm (x', y', z') trên hệ tọa độ bàn tay trái sẽ ở "phía trong" mặt này nếu nó thỏa bất phương trình:

$$Ax' + By' + Cz' + D < 0 \qquad (6\text{-}10)$$

Nếu điểm (x', y', z') là vị trí quan sát, khi đó bất kỳ mặt phẳng nào làm cho bất phương trình 6-10 đúng phải là một mặt ở đằng sau. Tức là, nó là mặt ta không thể nhìn thấy từ vị trí quan sát.

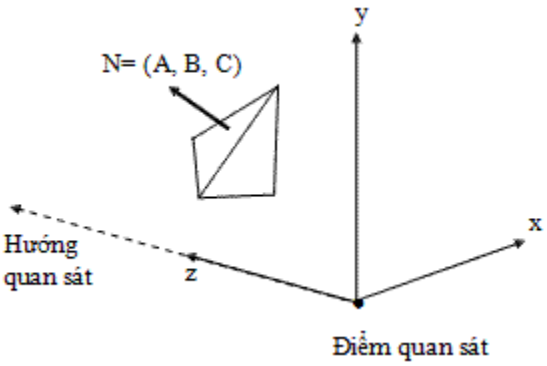


Hình 6.9 Một mặt phẳng với tham số C < 0 trong hệ quan

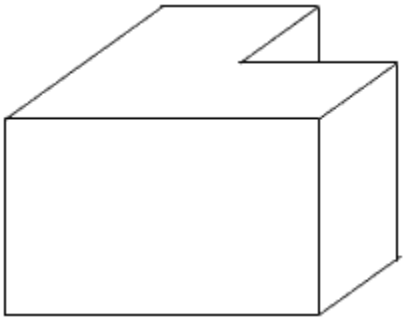
sát bàn tay phải được xác định như mặt ở đằng sau khi hướng quan sát cùng chiều với trục z âm.

Chúng ta có thể thực hiện một cách kiểm tra mặt đằng sau đơn giản hơn bằng cách nhìn ở vector pháp tuyến của mặt có phương trình 7-1, vector này có tọa độ Descartes (A, B, C). Trong hệ tọa độ bàn tay phải với hướng quan sát cùng chiều với trục z âm (xem hình 6.9), vector có tham số C song song với hướng quan sát. Nếu C<0, vector chỉ ra xa khỏi vị trí quan sát, và mặt phải là mặt ở đằng sau. Các tham số A, B, C, và D có thể được tính từ tọa độ các đỉnh được xét theo chiều kim đồng hồ (thay vì hướng ngược chiều kim đồng hồ được dùng trong hệ tọa độ bàn tay phải). Bất phương trình 6-10 sau đó cho một kiểm tra hợp lệ đối với các điểm nằm phía trong. Cũng như vậy, các mặt ở đằng sau có các vector chỉ ra xa khỏi vị trí quan sát và được xác định bởi C>0 khi hướng quan sát cùng hướng với trục z dương. Trong tất cả các thảo luận sau này trong chương, chúng ta giả sử rằng hệ quan sát bàn tay trái được dùng.

Hình 6.10 Trong hệ quan sát bàn tay trái, khi hướng quan sát cùng chiều với trục zv dương, một mặt ở đằng sau là mặt với tham số C>0.



Bằng việc kiểm tra tham số C ở mỗi mặt của đối tượng, ta có thể xác định được ngay tất cả các mặt ở đằng sau. Đối với một khối đa diện lồi đơn lẻ, như hình kim tự tháp trong hình 6.10, việc kiểm tra này xác định tất cả các mặt bị che khuất trên đối tượng, bởi vì mỗi mặt thì là hoàn toàn được nhìn thấy hoặc hoàn toàn bị che khuất. Đối với các đối tượng khác, các kiểm tra phức tạp hơn cần được thực hiện để xác định xem các mặt là bị che khuất hoàn toàn hay chỉ bị che khuất một phần.



Hình 6.11 Ảnh một đối tượng với một mặt bị che khuất một phần

Tương tự, chúng ta cần xác định xem các đối tượng là có một phần hay toàn bộ bị che khuất bởi các đối tượng khác. Một cách tổng quát, việc khử mặt khuất sẽ loại bỏ khoảng một nữa số mặt trong một ảnh khi thực hiện các phép kiểm tra tính nhìn thấy được sau này.

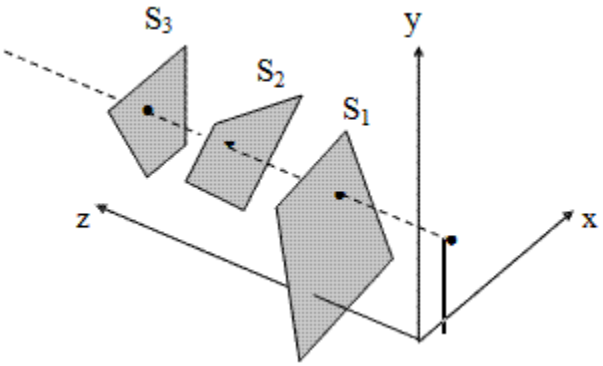
6.3.2 Phương pháp vùng đệm độ sâu

Một tiếp cận không gian ảnh được dùng phổ biến để khử mặt khuất là phương pháp vùng đệm độ sâu. Một cách cơ bản, thuật toán này kiểm tra tính nhìn thấy được của các mặt mỗi lần một điểm. Với mỗi vị trí pixel (x,y) trên mặt phẳng quan sát, mặt nào có giá trị tọa độ z nhỏ nhất ở vị trí pixel đó thì nhìn thấy được. Hình 6.4 trình bày ba mặt có độ sâu khác nhau, với sự quan tâm đến vị trí (x, y) trong hệ quan sát bàn tay trái. Mặt S1 có giá trị z nhỏ nhất ở vị trí này vì vậy giá trị độ sáng ở (x, y) được lưu.

Hai vùng đệm được cần để cài đặt phương pháp này. Một vùng đệm độ sâu được dùng để lưu trữ các giá trị z cho mỗi vị trí (x, y) của các mặt được so sánh. Vùng đệm thứ hai là vùng đệm làm tươi (hay còn gọi là vùng đệm khung), lưu giữ các giá trị độ sáng cho mỗi vị trí (x, y).

Phương pháp này có thể được thực hiện hiệu quả trong các hệ tọa độ chuẩn, với các giá trị độ sâu thay đổi từ 0 đến 1. Giả sử rằng một không gian chiếu được ánh xạ vào một không gian quan sát hình hộp chuẩn, ánh xạ của mỗi mặt lên mặt phẳng quan sát là một phép chiếu trực giao. Độ sâu của các điểm trên bề mặt của một đa giác được tính từ phương trình mặt phẳng. Ban đầu, tất cả các vị trí trong vùng đệm độ sâu được đặt giá trị 1 (độ sâu lớn nhất), và vùng đệm làm tươi được khởi tạo giá trị của độ sáng nền. Mỗi mặt (đã được lập danh sách trong các bảng đa giác sau đó được xử lý. Mỗi lần một đường quét tính độ sâu, hoặc giá trị z, ở mỗi vị trí (x, y). Giá trị z vừa được tính xong sẽ được so sánh với các giá trị lưu trữ trước đó trong vùng đệm độ sâu ở vị trí đó. Nếu giá trị z vừa được tính xong nhỏ hơn các giá trị trước đó, giá trị z mới sẽ được lưu, và độ sáng của mặt ở vị trí đó cũng được cập nhật lại vào vị trí tương ứng trong vùng đệm làm tươi.

Hình 6.12 Ở vị trí (x, y), mặt S1 có giá độ sâu nhỏ nhất và vì thế được nhìn thấy ở ví trí đó



Chúng ta có thể tổng kết các bước của thuật toán vùng đệm độ sâu như sau:

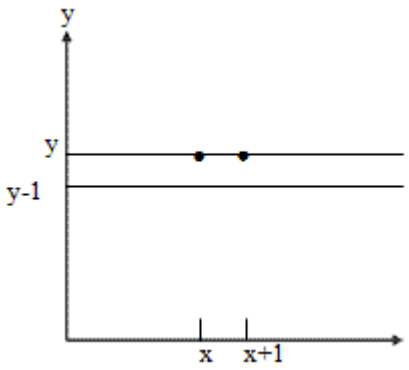
1. Khởi tạo vùng đệm độ sâu và vùng đệm làm tươi để với tất cả các vị trí (x,y), depth(x, y) =1 và refresh(x, y) = background.
2. Đối với mỗi vị trí trên mỗi mặt, so sánh các giá trị độ sâu với các giá trị độ sâu được lưu trước đó trong vùng đệm độ sâu để xác định tính chất nhìn thấy được.

a. Tính giá trị z cho mỗi vị trí (x, y) trên mặt.

b. Nếu z < depth(x, y) thì đặt lại depth(x, y)= z và refresh(x, y) = i , với i là giá trị độ sáng trên mặt ở vị trí (x, y). Trong bước cuối cùng, nếu z không nhỏ hơn giá trị trong vùng đệm độ sâu ở vị trí đó, điểm không được nhìn thấy. Khi quá trình này được hoàn thành cho tất cả các mặt, vùng đệm độ sâu chứa các giá trị z của các mặt nhìn thấy được và vùng đệm làm tươi chỉ chứa các giá trị độ sáng của các mặt nhìn thấy được đó. Các giá trị độ sâu cho một vị trí (x, y) được tính từ phương trình của mỗi mặt:

$$z = \frac{(-Ax - By - D)}{C} \qquad (6\text{-}11)$$

Với mỗi đường quét bất kỳ (xem hình 6.5), các tọa độ x trên cùng đường quét sai khác nhau 1, và các giá trị y giữa hai đường quét cũng sai khác nhau 1. Nếu độ sâu của vị trí (x,y) được xác định là z, khi đó độ sâu z' của vị trí kế tiếp (x+1, y) dọc theo theo đường quét có được từ phương trình



6.4 như sau: Với mỗi đường quét bất kỳ, các tọa độ x trên cùng đường quét sai khác nhau 1, và các giá trị y giữa hai đường quét cũng sai khác nhau 1. Nếu độ sâu của vị trí (x,y) được xác định là z, khi đó độ sâu z' của vị trí kế tiếp (x+1, y) dọc theo theo đường quét có được như sau:
hình 6.13

$$z' = \frac{(-A(x+1) - By - D)}{C} \qquad \text{hoặc} \quad z' = z - \frac{A}{C} \qquad (6\text{-}12)$$

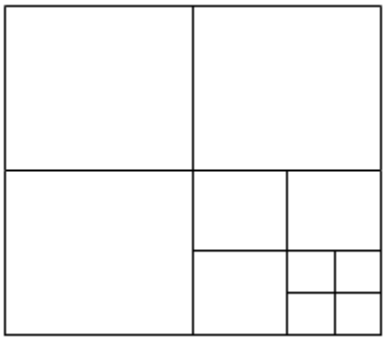
Tỷ số A/C không đổi với mỗi mặt, vì vậy giá trị độ sâu của điểm kế tiếp trên cùng đường quét có được từ giá trị trước đó với một phép trừ. Chúng ta thu được các giá trị độ sâu giữa các đường quét theo cách tương tự. Một lần nữa giả sử rằng vị trí (x, y) có độ sâu z. Khi đó ở vị trí (x, y-1) trên đường quét ngay bên dưới, giá trị độ sâu được tính từ phương trình mặt phẳng như sau:

$$z'' = \frac{(-Ax - B(y-1) - D)}{C} \qquad \text{hoặc} \quad z'' = z + \frac{B}{C} \qquad (6\text{-}13)$$

Độ sâu z được cộng thêm tỷ số B/C. Phương pháp vùng đệm độ sâu thì dễ dàng để cài đặt, và nó không cần sắp xếp các mặt trong ảnh. Nhưng nó cần đến một vùng đệm thứ hai đó là vùng đệm làm tươi. Một hệ thống với độ phân giải 1024 x 1024 có thể cần hơn một triệu vị trí trong vùng đệm độ sâu, với mỗi vị trí cần đủ bit để lưu giữ các tọa độ z tăng. Một cách để giảm bớt không gian lưu giữ cần thiết là tại mỗi thời điểm chỉ xử một phần của ảnh, dùng một vùng độ sâu nhỏ hơn. Sau mỗi phần ảnh được xử lý xong, vùng đệm được dùng lại cho phần kế tiếp.

6.3.3 Phương pháp phân chia vùng

Kỹ thuật khử mặt khuất này thì hiệu quả cho phương pháp không gian ảnh, nhưng các phương pháp không gian đối tượng có thể được dùng để thực hiện việc sắp xếp các mặt theo độ sâu. Phương pháp phân chia vùng tận dụng các thuận lợi của các vùng cố kết trong ảnh bằng cách xác định các vùng quan sát này để tách chúng làm nhiều phần nhỏ, mỗi phần được xem như một mặt đơn lẻ. Chúng ta áp dụng phương pháp này bằng cách phân chia thành công toàn bộ vùng quan sát thành các hình chữ nhật càng lúc càng nhỏ cho đến khi mỗi vùng nhỏ là hình chiếu của một phần của một mặt đơn lẻ nhìn thấy được, hoặc cho đến khi không thể tiếp tục phân chia.



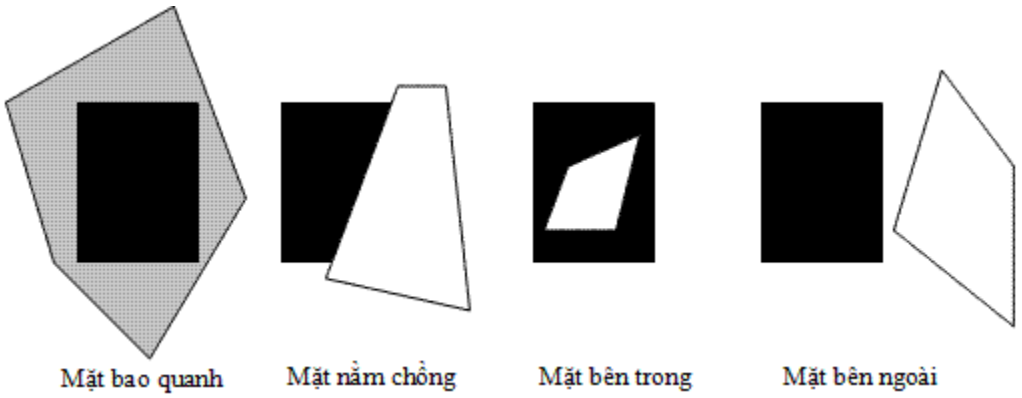
Hình 6.14 Các phần chia được thực hiện thành công với phép chia 2.

Để thực hiện phương pháp này, ta cần xây dựng các phép kiểm tra để xác định nhanh chóng vùng là một phần của một mặt đơn lẻ hoặc cho ta biết vùng thì quá phức tạp để phân tích bình thường. Bắt đầu với cái nhìn tổng thể, ta áp dụng các phép kiểm tra để xác định xem có nên phân chia toàn bộ vùng thành các hình chữ nhật nhỏ hơn không. Nếu các phép kiểm tra chỉ ra rằng mặt quan sát đủ phức tạp, ta phân chia nó. Kế tiếp, chúng ta áp dụng các phép kiểm tra đến mỗi vùng nhỏ hơn, chia nhỏ những vùng này nếu các phép kiểm tra xác định rằng tính nhìn thấy được của một mặt đơn là vẫn chưa chắc chắn. Chúng ta tiếp tục quá trình này đến khi các phần phân chia là dễ dàng được phân tích như là một mặt đơn lẻ hoặc đến

khi chúng được thu giảm kích thước thành một pixel.

Một cách để phân chia một vùng thành công là chia kích thước nó ra làm 2, như trong hình 6.6. Một vùng quan sát với độ phân giải 1024x1024 có thể được chia 10 lần trước khi một phần chia giảm thành 1 điểm. Các phép kiểm tra để xác định tính nhìn thấy được của một mặt đơn trong phạm vi vùng chỉ định được thực hiện bằng cách so sánh các mặt với biên của vùng. Có bốn khả năng có thể xảy ra khi xem xét mối quan hệ giữa một mặt với biên vùng chỉ định. Ta có thể mô tả đặc điểm của các quan hệ này theo các cách sau như hình dưới đây.

Hình 6.15 Các quan hệ có thể xảy ra giữa các mặt đa giác và một vùng chữ nhật.



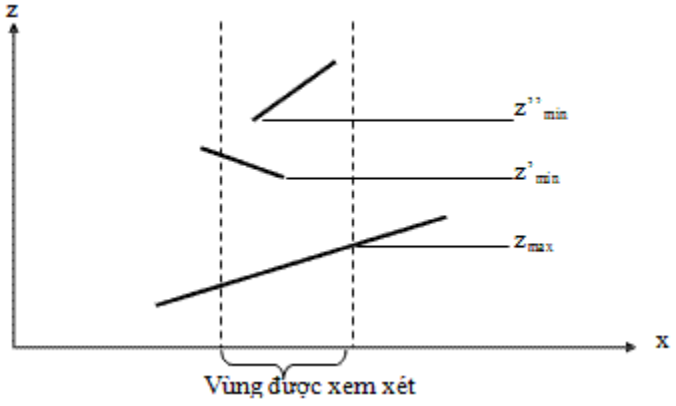
- Mặt bao quanh là mặt hoàn toàn bao quanh một vùng.
- Mặt nằm chồng là mặt có một phần nằm trong và một phần nằm ngoài vùng.
- Mặt bên trong là mặt hoàn toàn nằm bên trong vùng.
- Mặt bên ngoài là mặt hoàn toàn nằm bên ngoài vùng.

Các phép kiểm tra để xác định tính nhìn thấy được của mặt trong phạm vi một vùng có thể được đề cập giới hạn trong bốn loại này. Không có sự phân chia nào thêm nữa cho một vùng nếu một trong các điều kiện sau là đúng:

1. Tất cả các mặt nằm bên ngoài vùng.
2. Chỉ một mặt bên trong, mặt nằm chồng hoặc mặt bao quanh ở trong vùng.
3. Một mặt bao quanh che khuất tất cả các mặt khác trong phạm vi các biên của vùng.

Kiểm tra 1 có thể được thực hiện bằng cách kiểm tra các biên chữ nhật bao quanh các mặt với biên của vùng. Kiểm tra 2 cũng có thể dùng các biên chữ nhật trong mặt xy để xác định mặt nằm trong. Với những kiểu mặt khác, các biên chữ nhật có thể được dùng như một bước kiểm tra ban đầu. Nếu một biên chữ nhật cắt vùng theo cách nào đó, các kiểm tra tiếp theo mới được thực hiện để xác định xem mặt là: mặt bao quanh, mặt nằm chồng, hay mặt bên ngoài. Nếu được xác định là mặt bên trong, mặt nằm chồng, hay mặt bao quanh, các giá trị độ sáng của nó được chuyển đến vùng thích hợp trong vùng đệm khung.

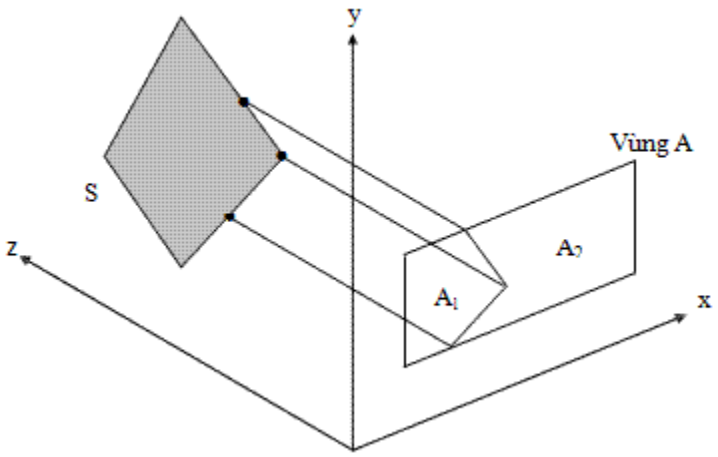
Hình 6.16 Một mặt bao quanh với độ sâu lớn nhất của zmax (xét trong vùng quan sát) che khuất tất cả các mặt mà độ



sâu nhỏ nhất xmin của chúng lớn hơn zmax.

Một phương pháp để thực hiện bước 3 là sắp xếp các mặt dựa theo độ sâu nhỏ nhất của chúng. Sau đó, với mỗi mặt bao quanh, ta đi tính giá trị z lớn nhất trong vùng được xem xét. Nếu giá trị lớn nhất z của một mặt nào (trong số các mặt mặt bao quanh) nhỏ hơn giá trị z nhỏ nhất của các mặt còn lại trong vùng, kiểm tra 3 thỏa mãn không.

Một phương pháp khác để thực hiện kiểm tra 3 mà không cần đến sắp xếp độ sâu là dùng các phương trình mặt phẳng để tính các giá trị z ở bốn đỉnh của vùng cho tất cả các mặt bao quanh, mặt nằm chồng, hay mặt bên trong. Nếu các giá trị z của một trong số các mặt bao quanh mà nhỏ hơn các giá trị z của các mặt còn lại, kiểm tra 3 đúng. Sau đó vùng có thể được tô với các giá trị độ sáng của mặt bao quanh.Trong vài trường hợp, cả hai phương pháp cho kiểm tra 3 trên sẽ thất bại để xác định đúng một mặt bao quanh che khuất tất cả các mặt còn lại khác. Việc kiểm tra thêm nữa sẽ được thực hiện để xác định mặt đơn che phủ vùng, tuy nhiên, thuật toán sẽ nhanh hơn nếu ta phân chia vùng hơn là tiếp tục làm các kiểm tra phức tạp. Khi các mặt bên ngoài và mặt bao quanh vừa được xác định cho một vùng, chúng nó sẽ còn lại các mặt bên ngoài và bao quanh cho tất cả các phần phân chia của vùng. Hơn nữa, vài mặt bên trong và mặt nằm chồng có thể đang chờ để bị loại bỏ khi quá trình phân chia tiếp tục, để các vùng trở nên dễ dàng hơn cho phân tích. Trong trường hợp đã đi đến giới hạn, kích thước vùng chia chỉ còn là 1 pixel, ta đơn giản đi tính độ sâu của mỗi mặt có liên quan ở điểm đó và chuyển giá trị độ sáng của mặt gần nhất vào vùng đệm khung.



Hình 6.17 Vùng A được phân chia thành A_1 và A_2 bằng cách dùng biên của mặt S trên mặt phẳng chiếu.

Như một thay đổi lên quá trình phân chia cơ bản, ta có thể phân chia các vùng dọc theo biên của mặt thay vì chia chúng làm 2. Nếu các mặt vừa được sắp theo độ sâu nhỏ nhất, ta có thể dùng mặt có giá trị z nhỏ nhất để phân chia một vùng được cho.

Hình 6.17 minh họa phương pháp này để phân chia các vùng. Hình chiếu của biên mặt S được dùng để phân chia vùng ban đầu thành các phần A1 và A2. Mặt S sau đó trở thành mặt bao quanh của A1 và các phép kiểm tra 2 và 3 có thể được áp dụng để xác định xem việc phân chia thêm nữa có cần thiết không. Trong trường hợp tổng quát, sự phân chia ít hơn được cần dùng tiếp cận này, tuy nhiên nhiều xử lý thêm nữa sẽ được cần để chia vùng và phân tích mối liên hệ giữa các mặt với các biên vùng chia.

6.4 Bài tập áp dụng

Bài 1 Viết chương trình vẽ đường cong Bezier.
Bài 2 Viết chương trình vẽ mặt lưới B – Spline.
Bài 3 Viết chương trình vẽ phối cảnh hình khối cầu, lăng trụ.. .

Gợi ý: Sử dụng thư viện đồ họa Direct X hoặc OpenGL.

TÀI LIỆU THAM KHẢO

- Đặng Văn Đức. *Kỹ thuật đồ họa máy tính*. Viện Công nghệ thông tin. 2002.
- Donald Hearn, M. Pauline Baker, *Computer Graphics*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1986.
- F.S.Hill, *Computer graphics*, 1990

ĐỀ THI THAM KHẢO

Các đề thi có thời gian làm bài 60 phút

Đề số 1

Câu 1 Cho 3 điểm A, B, C có tọa độ lần lượt là A(0, 40); B(-37, 125); C(40, -81). Hãy tìm tọa độ mới của các điểm trên qua biến đổi:

a. Phép biến đổi tỷ lệ với hệ số tỷ lệ theo trục hoành là 3, trục tung là 2.
b. Phép quay góc 60^o , ngược chiều kim đồng hồ quanh gốc tọa độ.

Câu 2 Hãy tìm các hệ số p và tọa độ các điểm sẽ nảy sinh của đoạn thẳng được vẽ theo giải thuật bresenham, nếu biết đoạn thẳng đi qua hai điểm là A(5, 10) và B(15, 15).

Đề số 2

Câu 1 Cho hệ 3 điểm A, B, C có tọa độ lần lượt là A(43, 0); B(25, -67); C(-40, -34). Hãy tìm tọa độ mới của các điểm trên qua biến đổi:

a. Phép tịnh tiến với hệ số tịnh tiến theo trục hoành là 44, trục tung là -81.
b. Phép đối xứng qua trục tung.
c. Tìm ma trận biến đổi của 2 phép tịnh tiến và phép đối xứng trên.

Câu 2 Hãy tìm các hệ số p và tọa độ các điểm sẽ nảy sinh của đoạn thẳng được vẽ theo giải thuật midpoint, nếu biết đoạn thẳng đi qua hai điểm là A(0, 13) và B(10, 7).

Đề số 3

Câu 1 Cho 3 điểm A, B, C có tọa độ lần lượt là A(43, 0); B(25, -67); C(-40, -34). Hãy tìm tọa độ mới của các điểm trên qua biến đổi:

a. Phép biến dạng với hệ số biến dạng theo trục hoành là 4, trục tung là 3.
b. Phép quay góc 45^o , ngược chiều kim đồng hồ quanh gốc tọa độ.

Câu 2 Hãy tìm hệ số p và tọa độ của 10 điểm sẽ nảy sinh đầu tiên của cung 1/8 đường tròn được vẽ theo giải thuật midpoint, nếu biết đường tròn có tâm (10, -25) và bán kính 90.

Đề số 4

Câu 1 Cho 3 điểm A, B, C có tọa độ lần lượt là A(0, 0); B(67, -46); C(-9, 22). Hãy tìm tọa độ mới của các điểm trên qua biến đổi:

a. Phép đối xứng qua trục hoành.
b. Phép biến đổi tỷ lệ với hệ số tỷ lệ theo trục hoành là 2, trục tung là 1.
c. Tìm ma trận biến đổi của 2 phép đối xứng và phép biến đổi tỷ lệ trên.

Câu 2 Hãy tìm hệ số p và tọa độ của 32 điểm sẽ nảy sinh đầu tiên của đường tròn được vẽ theo giải thuật midpoint, nếu biết đường tròn có tâm (-30, -12) và bán kính 60.

Đề số 5

Câu 1 Cho 3 điểm A, B, C có tọa độ lần lượt là A(0, 1, 18); B(7, -16, 9); C(-9, 12, 4). Hãy tìm tọa độ mới của các điểm trên qua biến đổi:

a. Phép biến dạng theo trục z, hệ số biến dạng tác động lên hoành độ là 3, lên tung độ là 2.
b. Tìm ma trận biến đổi của phép đối xứng qua đường y = x.

Câu 2 Cho cửa sổ giới hạn bởi điểm trên trái có tọa độ (10, 76) và điểm dưới phải có tọa độ (60, 24). Áp dụng giải thuật Cohen-Sutherland thực hiện cắt xén các đoạn thẳng sau với biên cửa sổ:

√ Đoạn thẳng AB: A(4, 35); A(50, 80)
√ Đoạn thẳng CD: C(47, 29); D(55, 3)
√ Đoạn thẳng EF: E(-10, 29); F(0, 3)

Đề số 6

Câu 1 Cho 3 điểm A, B, C có tọa độ lần lượt là A(1, -6, -18); B(16, 7, -9); C(0, 4, -3). Hãy tìm tọa độ mới của các điểm trên qua biến đổi:

a. Phép biến đổi tỷ lệ, hệ số biến đổi theo trục x là 1, trục y là 3, trục z là 2.
b. Tìm ma trận biến đổi của phép đối xứng qua đường y = -x.

Câu 2 Cho cửa sổ giới hạn bởi điểm trên trái có tọa độ (4, 50) và điểm dưới phải có tọa độ (60, -30). Áp dụng giải thuật Cohen-Sutherland thực hiện cắt xén các đoạn thẳng sau với biên cửa sổ:

√ Đoạn thẳng AB: A(24, 0); B(40, 90)
√ Đoạn thẳng CD: C(38, -35); D(55, 0)
√ Đoạn thẳng EF: E(-10, 29); F(0, 3)

Đề số 7

Câu 1 Cho 3 điểm A, B, C có tọa độ lần lượt là A(0, 1, 18); B(7, -16, 9); C(-9, 12, 4). Hãy tìm tọa độ mới của các điểm trên qua biến đổi:

a. Phép tịnh tiến, hệ số tịnh tiến theo trục x là 10, trục y là -15, trục z là 30.
b. Tìm ma trận biến đổi của phép đối xứng qua đường y = x.

Câu 2 Cho cửa sổ ở hệ tọa độ thực giới hạn bởi điểm trên trái có tọa độ (10, 76) và điểm dưới phải có tọa độ (60, 24); cửa sổ quan sát giới hạn bởi điểm trên trái có tọa độ (-15, 15) và điểm dưới phải có tọa độ (-10, 24). Cho tam giác ABC xác định trong hệ tọa độ như sau: A(14, 35); B(50, 29); C(47, 39). Tìm tọa độ mới của tam giác trong cửa sổ quan sát.

Đề số 8

Câu 1 Cho 3 điểm A, B, C có tọa độ lần lượt là A(-1, -6, 8); B(6, 0, -39); C(4, 0, -2). Hãy tìm tọa độ mới của các điểm trên qua biến đổi:

a. Phép biến đổi tỷ lệ, hệ số biến đổi theo trục x là 2, trục y là 3, trục z là 1.
b. Tìm ma trận biến đổi của phép đối xứng qua đường y = -x.

Câu 2 Cho cửa sổ ở hệ tọa độ thực giới hạn bởi điểm trên trái có tọa độ (20, 80) và điểm dưới phải có tọa độ (50, 34); cửa sổ quan sát giới hạn bởi điểm trên trái có tọa độ (-15, 20) và điểm dưới phải có tọa độ (-12, 25). Cho tam giác ABC xác định trong hệ tọa độ như sau: A(14, 35); B(50, 29); C(47, 39). Tìm tọa độ mới của tam giác trong cửa sổ quan sát.

BÀI GIẢNG ĐỒ HỌA MÁY TÍNH

CÁC KHÁI NIỆM CƠ BẢN

NGÔ QUỐC VIỆT

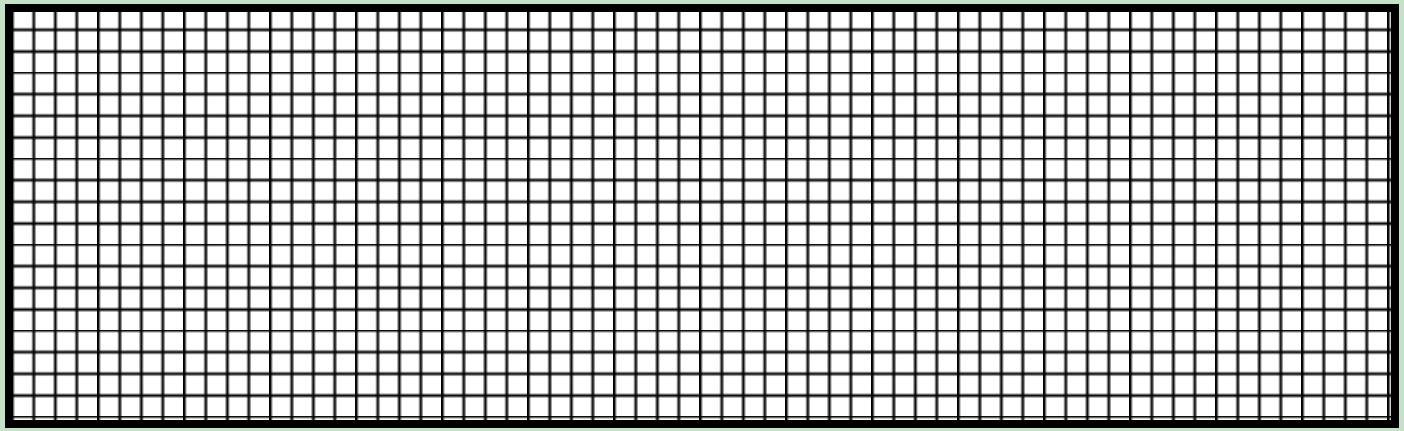
2011

Nội dung

1. Thiết bị hiển thị
2. Các đối tượng đồ họa cơ bản
3. Không gian màu
4. Nhắc lại đại số tuyến tính
5. Hệ toạ độ
6. Cách sử dụng OpenGL

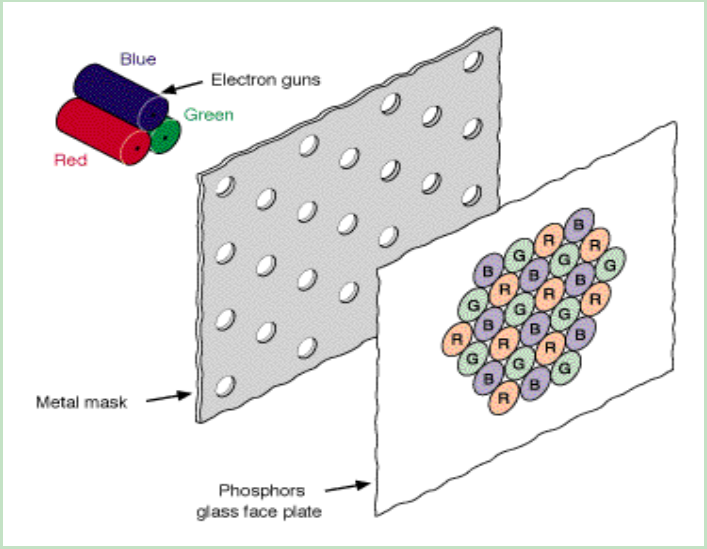
Thiết bị hiển thị

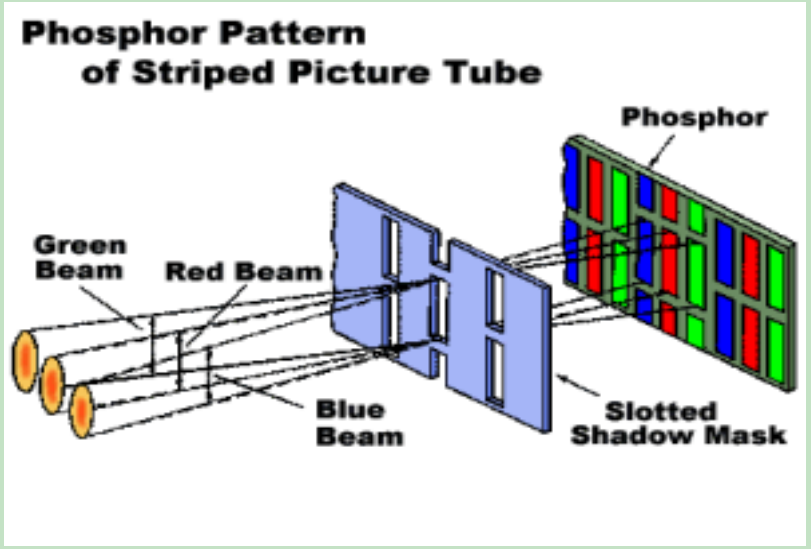
- Vector: plotter.
- Raster: màn hình, máy in.



Thiết bị hiển thị

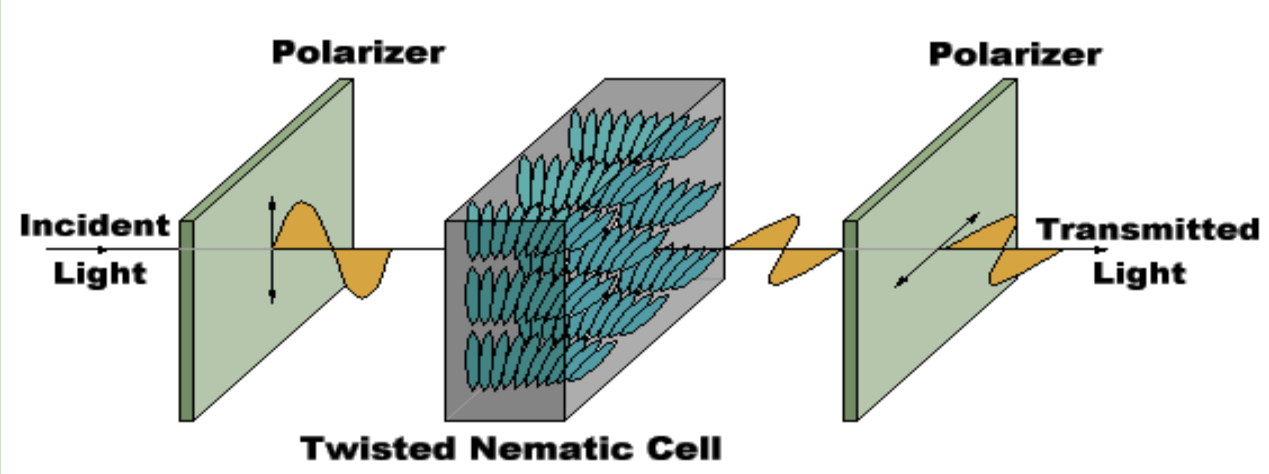
- CRT





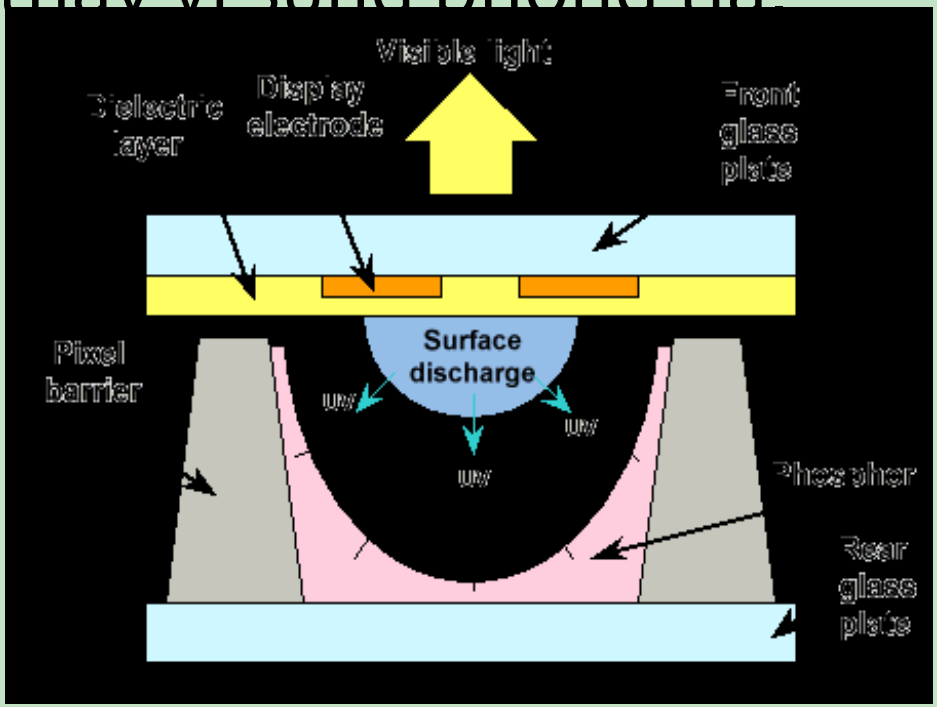
Thiết bị hiển thị

- LCD (Liquid Crystal Displays)



Thiết bị hiển thị

- Plasma: nguyên lý giống CRT nhưng tác động bởi gas thay vì súng phóng tia

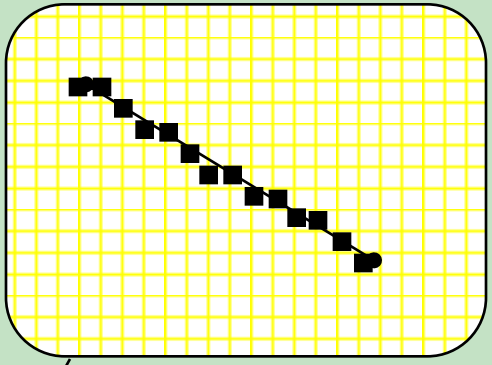


Các đối tượng đồ họa cơ bản

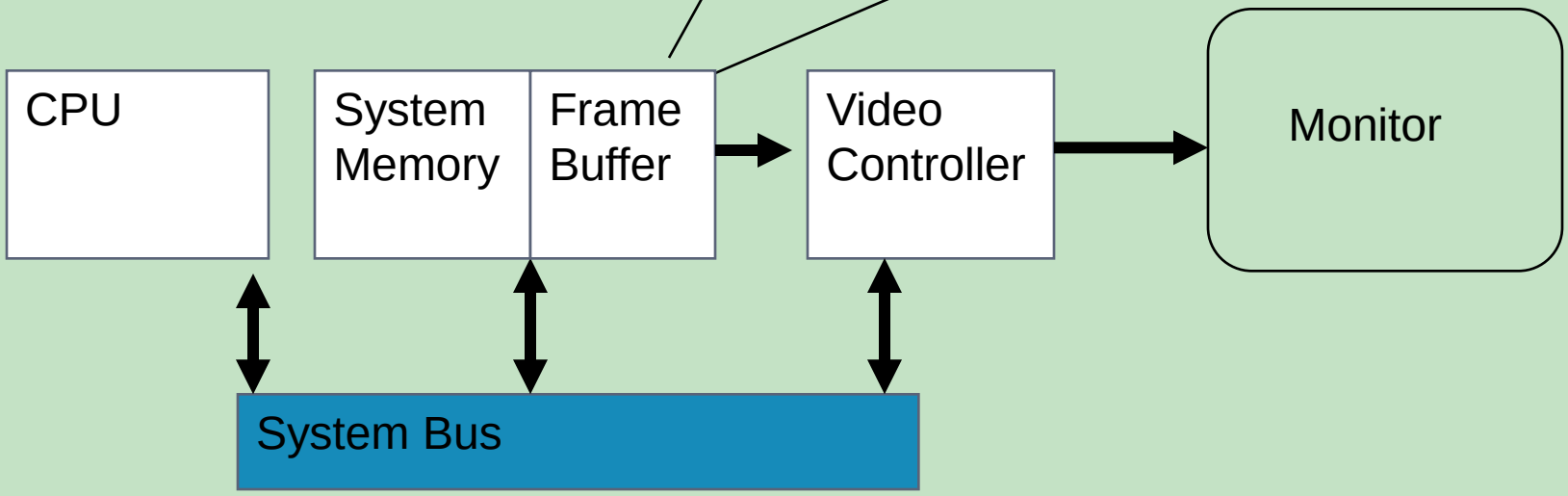
- Pixel: điểm ảnh trong thiết bị raster.
  - Sắc xám (từ 0→ 255), hoặc màu thể hiện bởi bộ ba (hay bộ bốn) giá trị 0→255.
- Độ phân giải: từ 320x200 →2000x1500.
- Khái niệm điểm (point): tọa độ (x, y, z) → vị trí trong không gian.
- Đường gấp khúc (polyline): dãy tọa độ {(x, y, z)}.
- Polygon: đường gấp khúc khép kín.

Video Controllers

- Phần cứng đồ hoạ
  - Chứa frame buffer
  - Liên kết trực tiếp với video controller



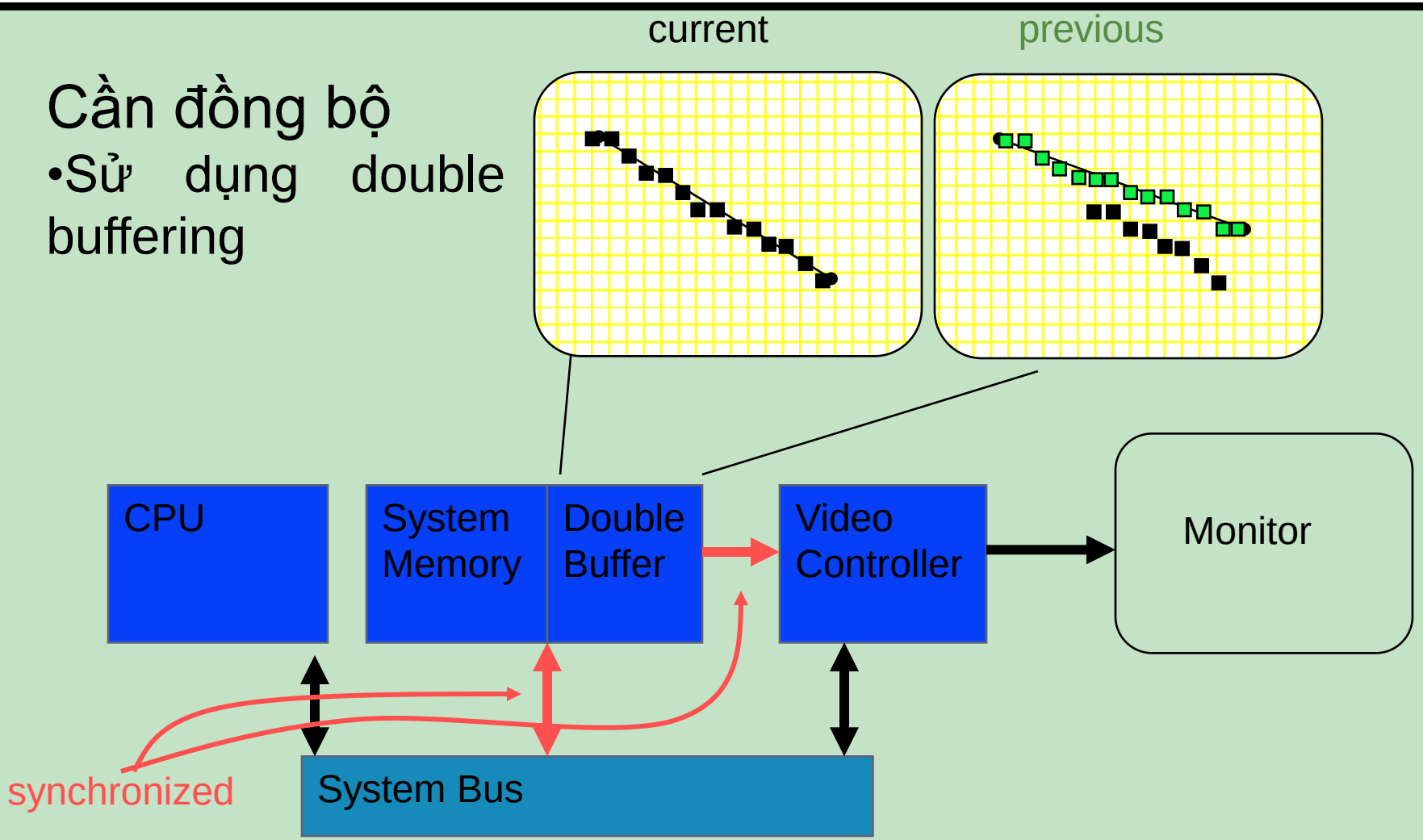
Frame buffer
Cartesian
Coordinates



Video Controllers

Cần đồng bộ

•Sử dụng double buffering



Raster Graphics Systems

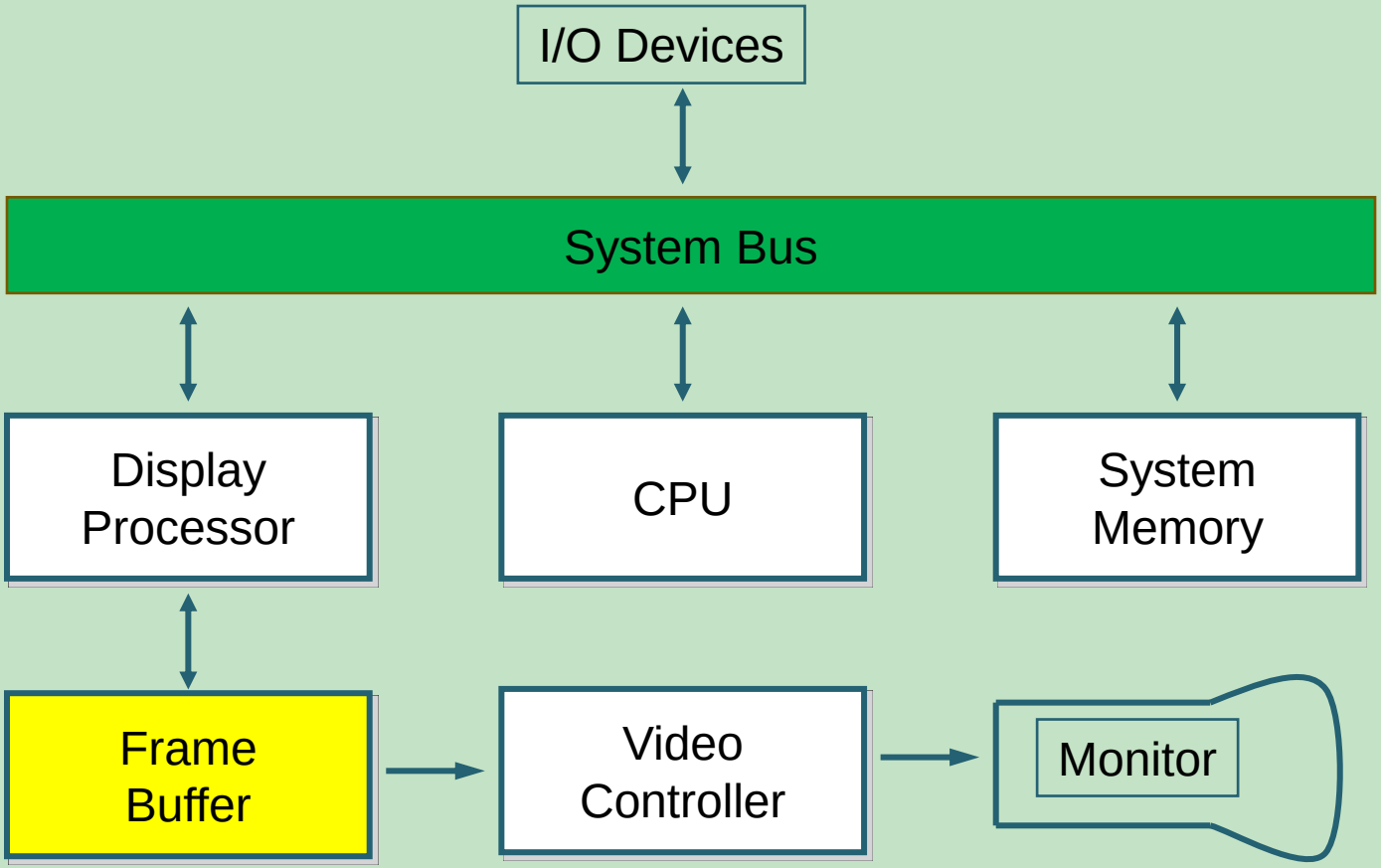
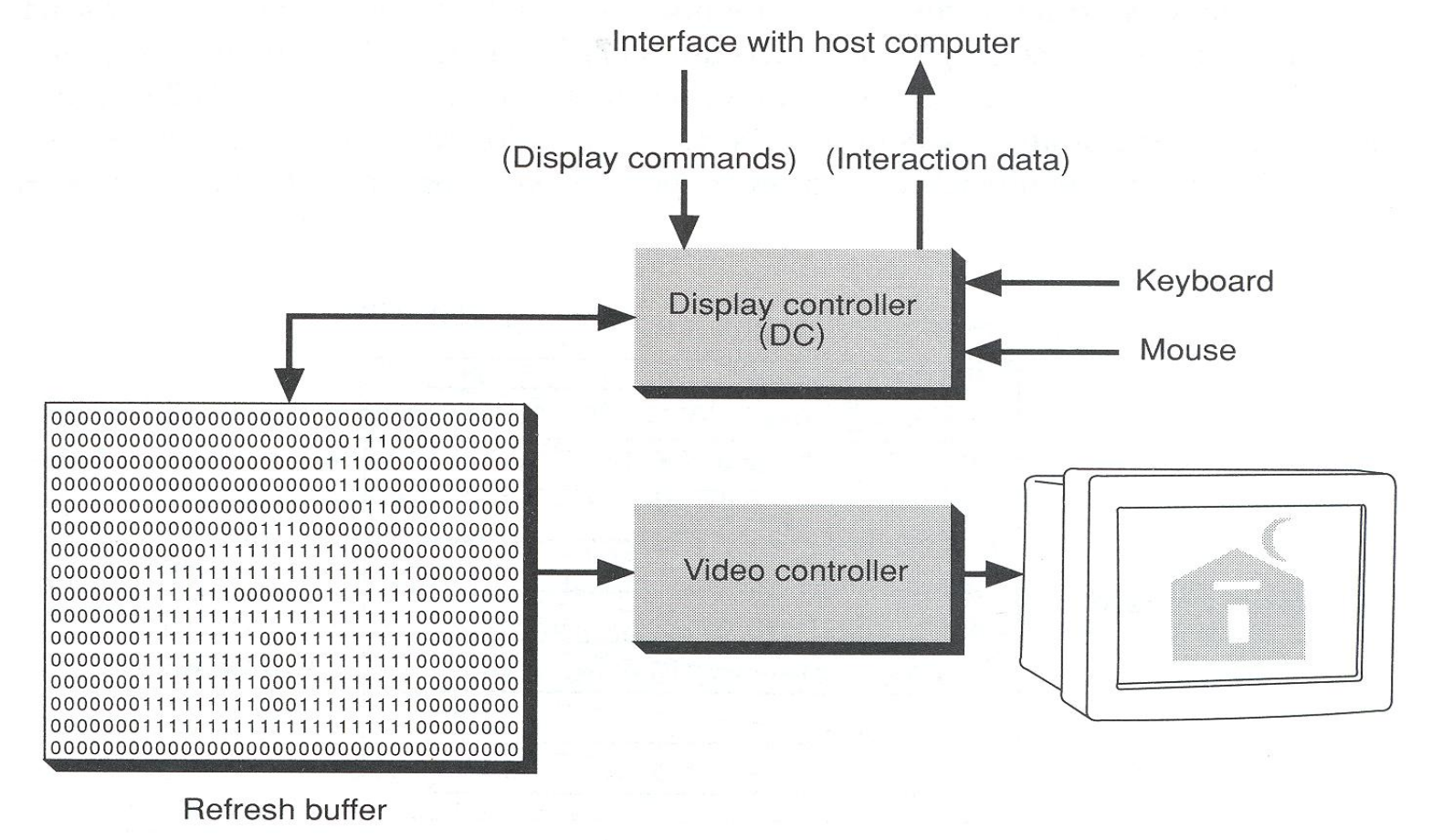


Figure 2.29 from Hearn and Baker

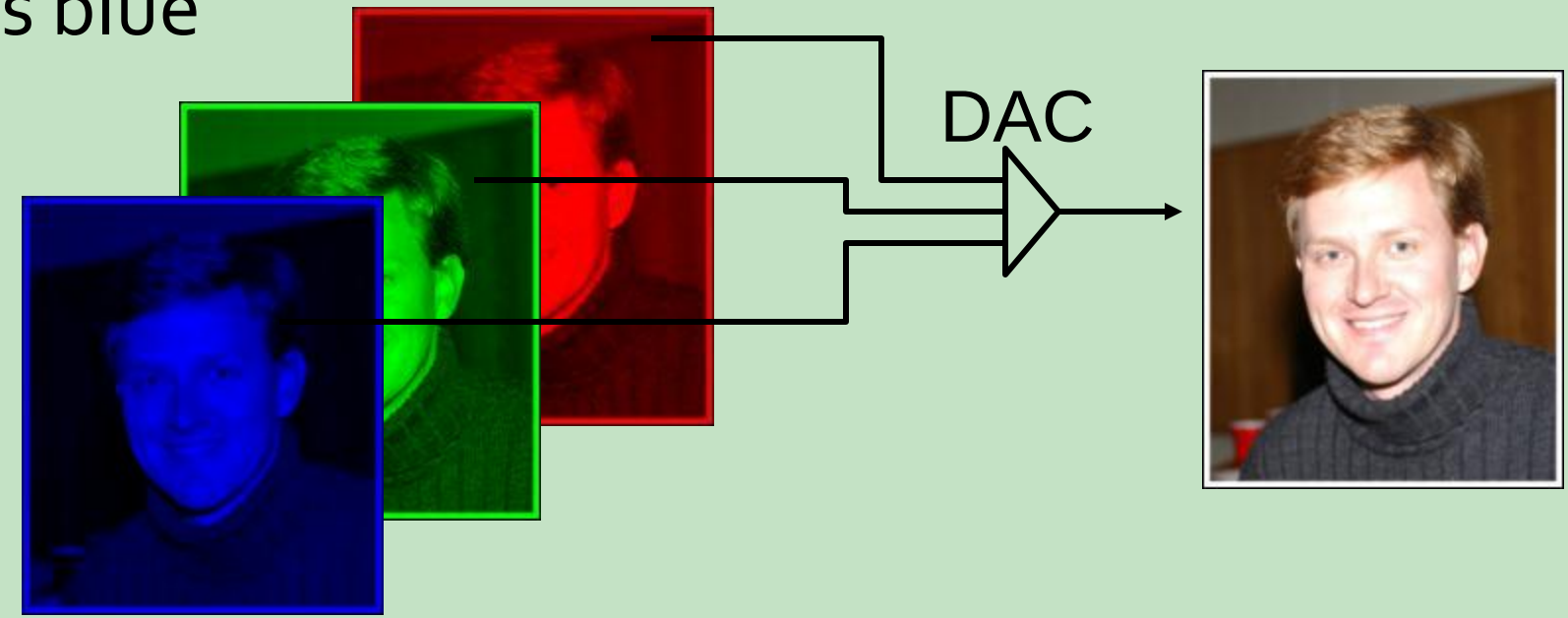
Frame buffer



Trích từ Foley et al

Direct Color Frame buffer

- Lưu trữ cường độ của R, G, và B trong frame buffer.
- 24 bits per pixel = 8 bits red, 8 bits green, 8 bits blue



Không gian màu

- Là phương pháp để xác định màu
- Ví dụ: người cảm nhận màu sắc qua
  - *Màu (Hue)*: phân biệt đỏ, vàng, xanh, v.v..
  - *Độ bão hòa (Saturation)*: nhạt/đậm hơn bao nhiêu so với màu bão hòa.
  - *Độ sáng (Lightness)*: cường độ sáng. Đôi khi còn gọi là *brightness* nếu đối tượng phát (như TV luôn gọi là brightness)
- Máy tính mô tả màu với (R, G, B)

Tại sao có nhiều không gian màu

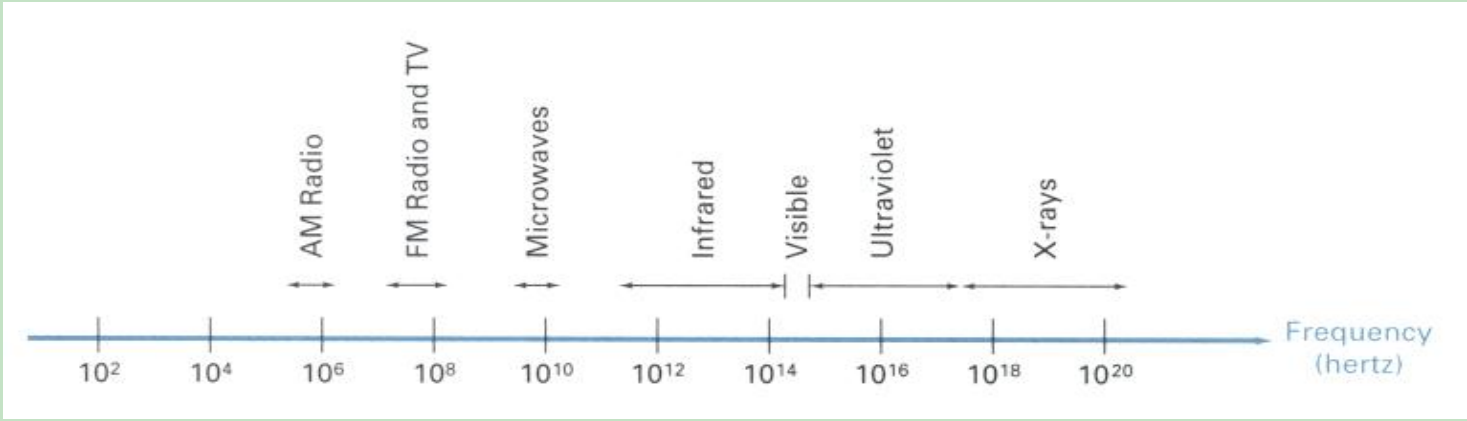
- Do đáp ứng tuỳ thuộc vào những ứng dụng khác nhau.
- Do gắn chặt với thiết bị
- Một không gian màu số sử dụng độ đo tuyến tính. Máy tính không sử dụng màu tuyến tính.
- Một số cần biểu diễn trực giác.
- Các không gian màu phổ biến: RGB, CMYK, HSL, YIQ, YUV, YCbCr, YCC, CIE (CIELab, CIELuv).

Tại sao có nhiều không gian màu

- Biết rõ các không gian màu → sẽ có ưu thế khi thực hiện các ứng dụng đồ hoạ, xử lý ảnh
- Ví dụ: nén ảnh có thể chuyển về không gian thích hợp sẽ nén tốt hơn.
- Nhận dạng: với không gian màu thích hợp sẽ thuận lợi hơn (bài toán color matching).
- Không phải không gian màu nào cũng phủ tối ưu màu.

Không gian màu

- Tần số ánh sáng có thể thấy được trong khoảng
  - Đỏ = 4.3×10^{14} hertz (700nm)
  - Tím = 7.5×10^{14} hertz (400nm)

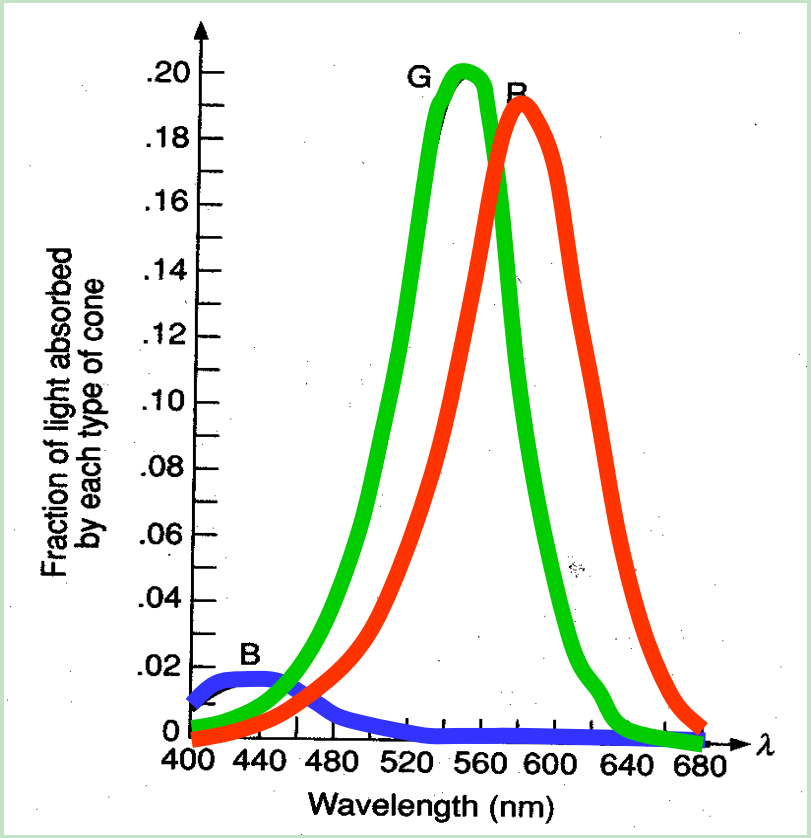


- Thấy rõ nhất: yellow-green ở 550 nm
- Thấy yếu nhất: blue ở 440nm.

Cảm nhận màu sắc của người

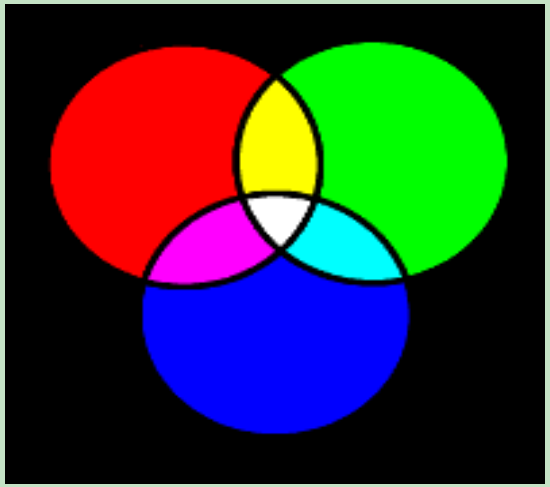
- Humans có sắc tố cảm nhận ánh sáng được gọi là L, M, and S
- Mỗi cái có phổ khác nhau

$$L = \int L(\lambda)E(\lambda)d\lambda$$
$$M = \int M(\lambda)E(\lambda)d\lambda$$
$$S = \int S(\lambda)E(\lambda)d\lambda$$



Kết hợp màu

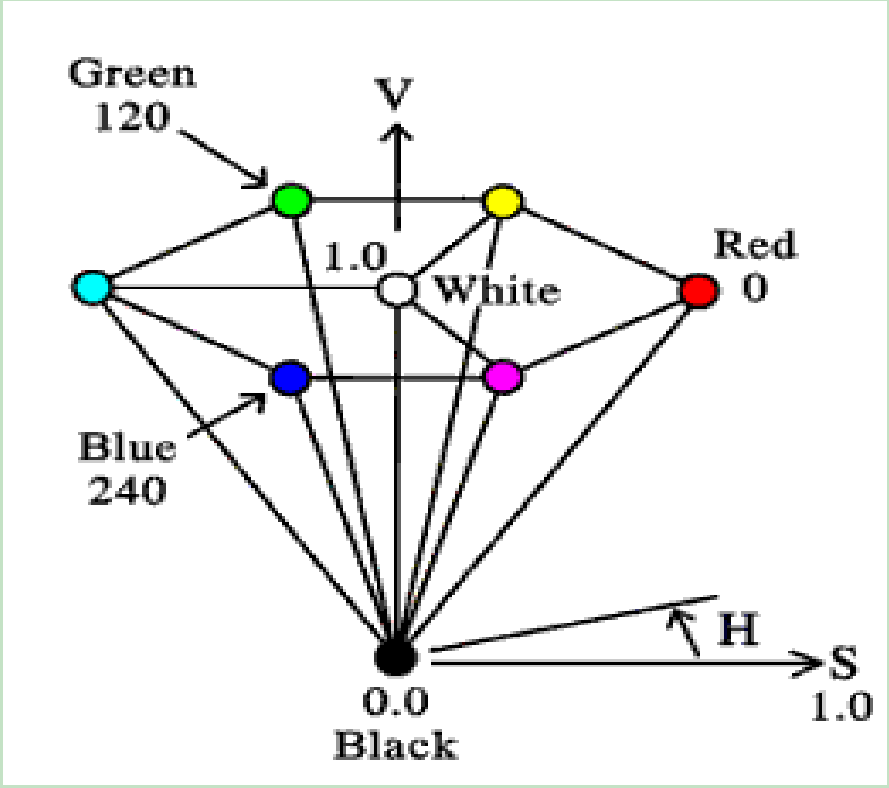
Màu cộng (RGB)
Shining colored lights
on a white ball



Màu trừ (CMYK)
Mixing paint colors and
illuminating with white light

Không gian HSV

- Gần với cảm nhận mắt



Hue (H): góc quay quanh trục đứng

Saturation (S): giá trị từ 0 đến 1 chỉ ra độ bão hòa.

Value (V): chiều cao của hình nón

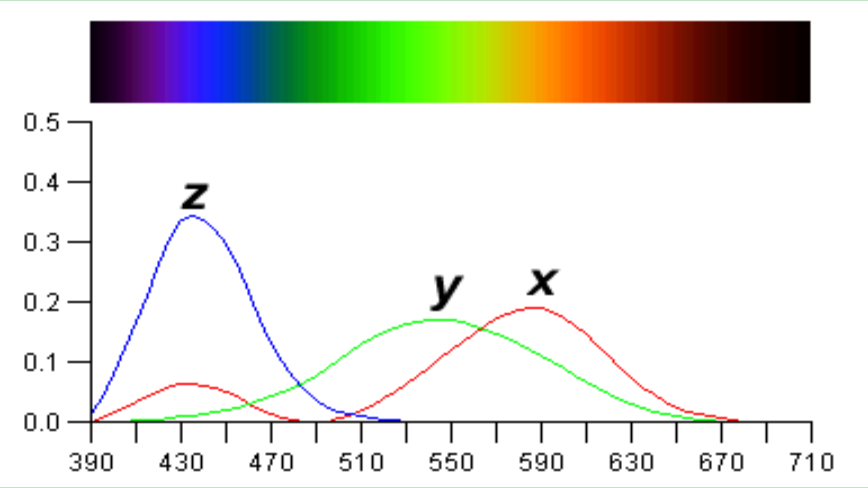
Không gian HSV

- Quan sát nón HSV từ trên xuống

H	S	V	Color
0	1.0	1.0	Red
120	1.0	1.0	Green
240	1.0	1.0	Blue
*	0.0	1.0	White
*	0.0	0.5	Gray
*	*	0.0	Black
60	1.0	1.0	?
270	0.5	1.0	?
270	0.0	0.7	?

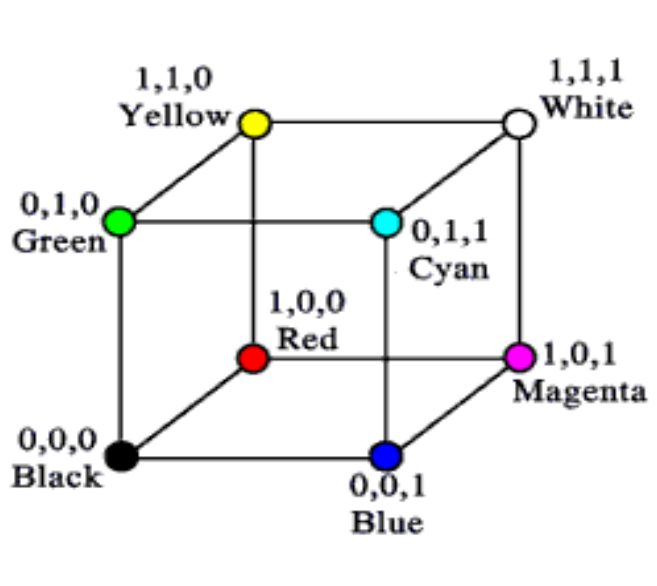
Không gian CIE (1931)

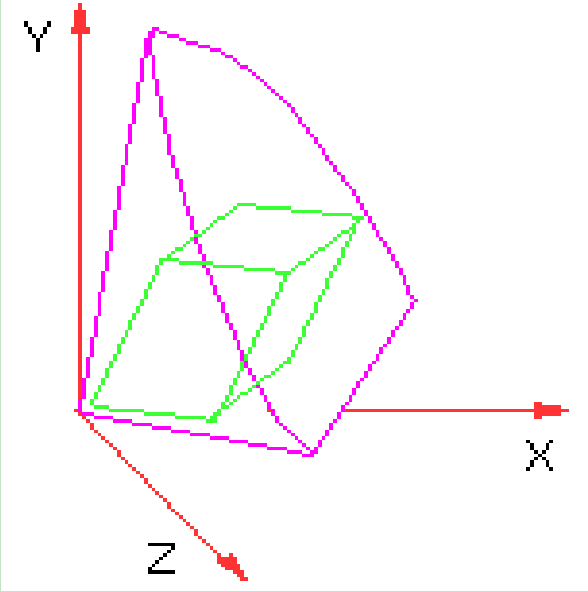
- Tập 3 bước sóng có thể được kết hợp để tạo ra các bước sóng khác.
- The CIE (Commission Internationale d'Eclairage) xác định 3 d nguồn sáng X, Y, Z:
- Ý tưởng: bước sóng λ có thể được tạo ra từ sự kết hợp của X, Y, và Z. Gần như tuyến tính với cảm nhận màu sắc.



Không gian màu RGB

- Định nghĩa theo hình khối vuông dạng
- Khối vuông RGB nằm trong không gian CIE theo dạng





Không gian màu YIQ

- YIQ mô hình màu dùng cho ti vi màu ở America. Y: độ sáng; I (orange-cyan); Q (green-magenta).
- Nếu chỉ dùng Y => ti vi trắng đen.
- Chuyển từ màu sang trắng/đen băng cách chỉ xét giá trị R.
- Chuyển đổi từ RGB sang YIQ

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.28 & -0.32 \\ 0.21 & -0.52 & 0.31 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Không gian màu CMY

- Cyan, magenta, và yellow là phần bù của red, green, và blue
  - Có dạng là mầu trắng trừ đi thành phần màu tương ứng.
  - Giống như RGB, nhưng gốc ở màu white thay vì black.
- Thuận tiện cho thiết bị hardcopy như máy in laser
  - Nếu thêm mực cyan vào tran in, ánh sáng đỏ không bị phản chiếu
  - Thêm màu đen là một thành phần (CMYK) để tạo cân bằng giữa CMY.

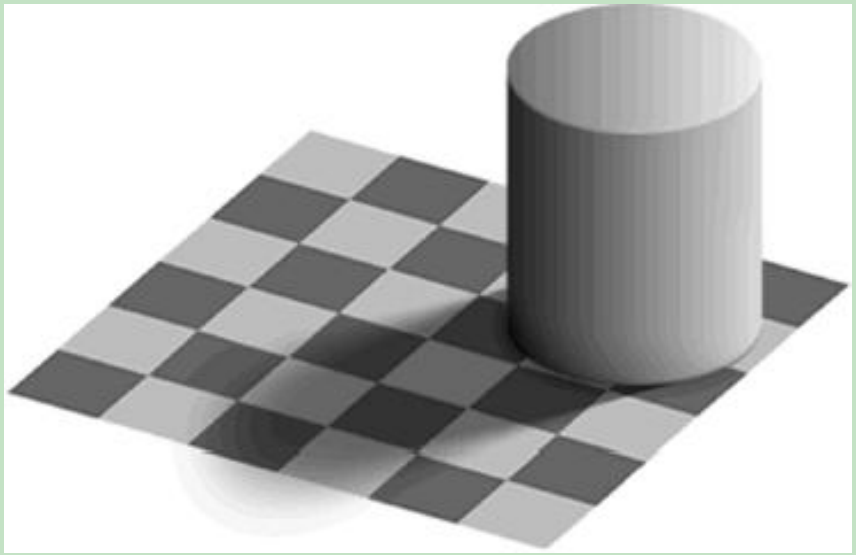
$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Tổ chức màu theo index

- Sử dụng chỉ số màu thay vì RGBA.
  - 2màu: chỉ số 0, 1.
  - 4 màu: chỉ số 0, 1, 2, 3.
  - 256 màu: chỉ số 0, 1, .., 255.
  - .....
- Sử dụng bảng màu ánh xạ cho các chỉ số màu.
  - Ví dụ: ảnh 256 sắc màu, sử dụng bảng 256 màu.
- Ví dụ: ảnh sắc xám, ảnh "trắng đen".
- Ưu điểm và nhược điểm?

Brightness và Lightness

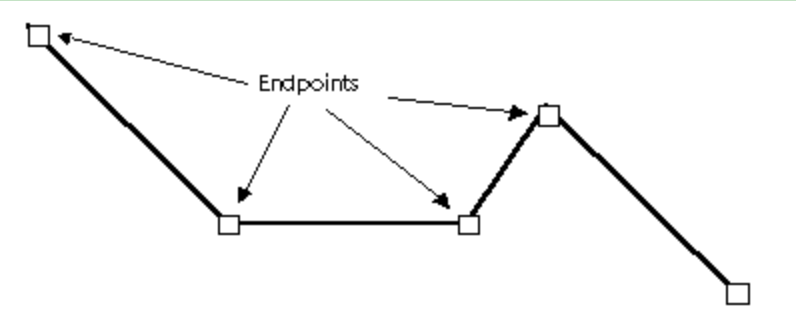
- Brightness: lượng ánh sáng (phát ra)
- Lightness: 'trắng' cỡ nào (nhận được)

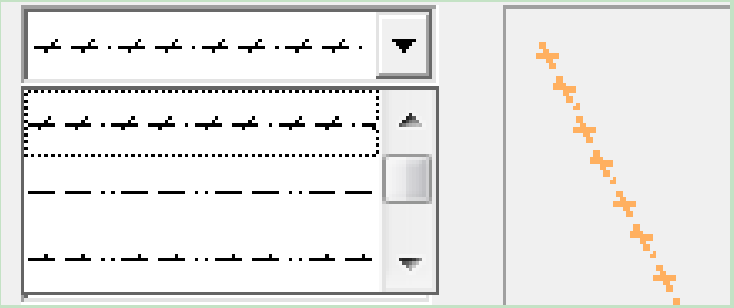


- Ô trắng sẽ bị tối khi bị bóng đen chiếu vào nó.

Các đối tượng đồ hoạ

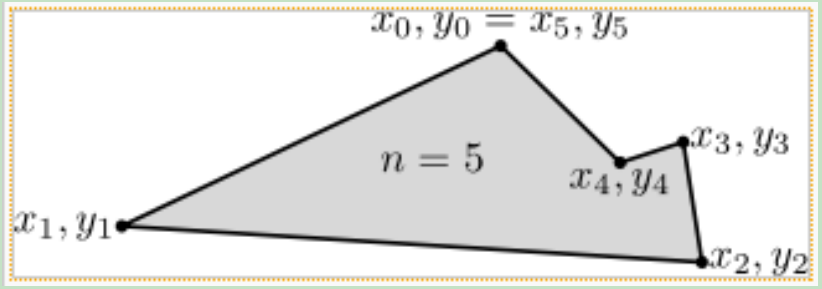
- Điểm: toạ độ (x, y).
  - Vị trí trong hệ toạ độ: điểm mốc, cột đèn, v.v.
- Đoạn thẳng: nối giữa hai điểm.
- Đường gấp khúc (polyline): nối nhiều đỉnh (vertex, endpoint).
  - Kiểu đường: biên giới, hàng rào.

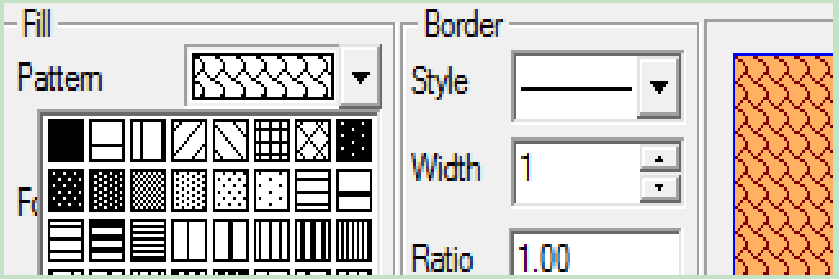




Các đối tượng đồ hoạ

- Đường gấp khúc khép kín (polygon)

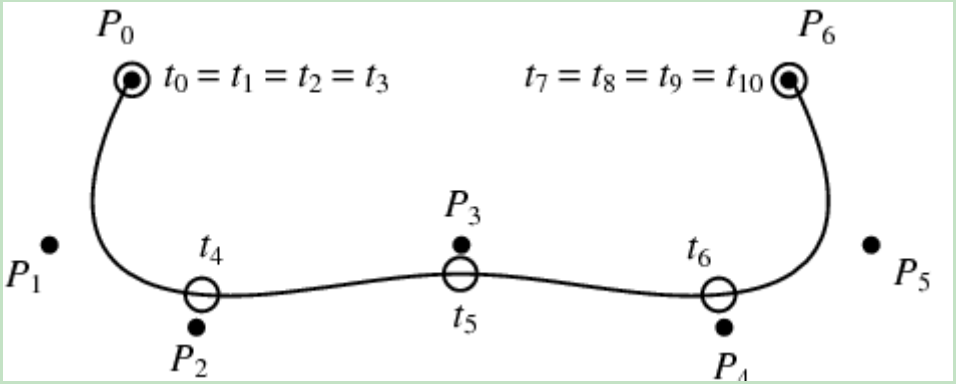


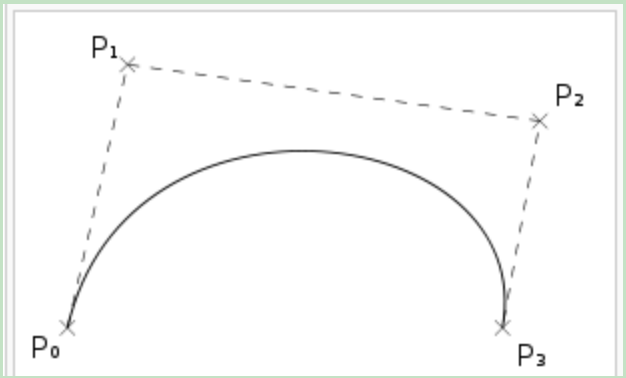


- Câu hỏi : tính diện tích đa giác và thuật giải thực hiện.
- Câu hỏi: tính tâm của đa giác.
- Viết cấu trúc thể hiện các đối tượng: điểm, polyline, polygon.

Các đối tượng đồ hoạ khác

- Đường ellipse, arc, đường bậc 2, bậc 3, ..





Trích: Wikipedia

- Chú ý: các đối tượng đặc biệt đều được chuyển thành polyline hay polygon khi vẽ lên thiết bị.

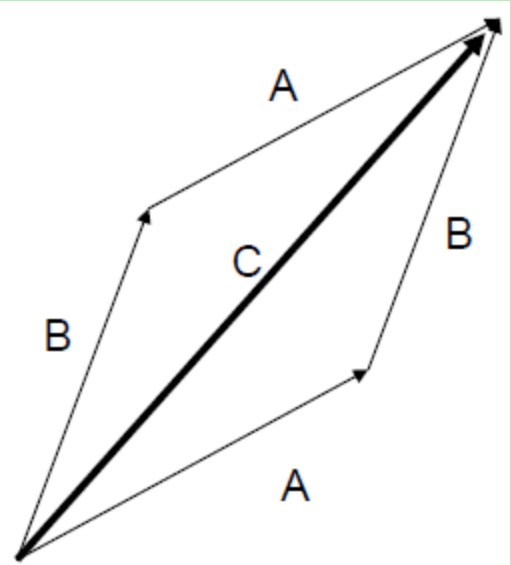
Nhắc lại đại số tuyến tính

- Dot product: đo độ tương quan hai vector

$$a \cdot b = ab^T = \begin{bmatrix} a & b & c \end{bmatrix} \begin{bmatrix} d \\ e \\ f \end{bmatrix} = ad + be + cf$$

$$\|a\|^2 = aa^T = \sqrt{aa + bb + cc}$$

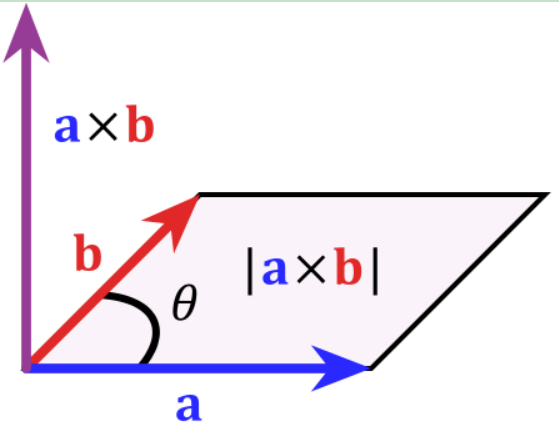
$$a \cdot b = \|a\| \|b\| \cos(\theta)$$

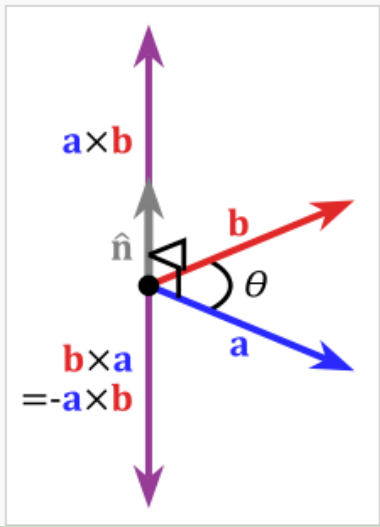


Nhắc lại đại số tuyến tính

- Cross product của hai vector A, B là vector C trực giao với mặt phẳng chứa A và B.
- Hướng của C theo nguyên tác bàn tay phải

$$a \times b = \|a\| \|b\| \sin\theta . \hat{n}$$



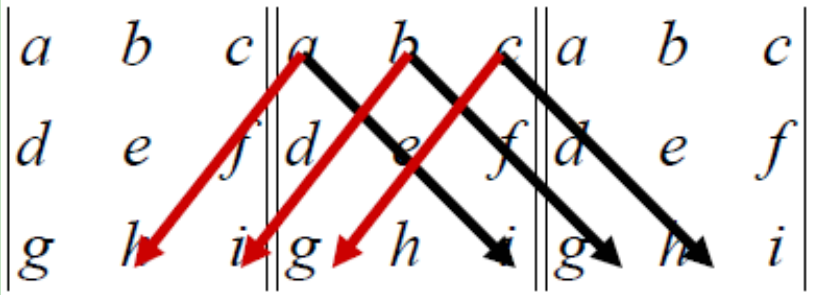


- Chuẩn của C: $\|a \times b\| = \|a\| \|b\| \sin(\theta)$

Nhắc lại đại số tuyến tính

- Định thức ma trận:

$$\begin{vmatrix} a & b & c \\ d & e & f \\ g & h & i \end{vmatrix} = aei + bfg + cdh - afh - bdi - ceg$$



- Cộng từ trái sang phải. Trừ từ phải sang trái

Các hệ toạ độ thường dùng

- Thế giới thực: kinh độ, vĩ độ, mét, kilômét;
- Toạ độ đối tượng: xe hơi, máy bay, nhà (đâu cần biết kinh, vĩ độ).
- Hệ toạ độ nhìn: mình quan sát mọi thứ ra sao (từ đâu, góc nhìn thế nào).
- Hệ toạ độ chiếu: phóng to, thu nhỏ, v.v.
- Toạ độ thiết bị: pixel.

Tóm tắt: thế giới được vẽ thế nào

- Mọi vật thể đều quy về các toạ độ, màu sắc, ánh sáng.
- Qua các bước trung gian nhằm: tăng tốc độ, vẽ cho đẹp, thực tế, đa dạng.
- Render ra toạ độ thiết bị (pixel).

Sử dụng OpenGL với C++.

- Download GLUT: http://www.opengl.org/resources/libraries/glut.html
- Kết hợp với Visual C++. Copy files vào các folder sau:
  - glut.h → VC/include/gl/
  - glut32.lib → VC/lib/
  - glut32.dll → windows/system32/
- Header Files, luôn đưa các include sau vào đầu file header có sử dụng OpenGL.
  - #include <GL/glut.h>
- Lib Files: Glu32.lib; GLaux.lib và OpenGL32.lib (nếu là GLUT cũ).

Đọc thêm tài liệu và bài tập

- Đọc thêm về các không gian màu và cách chuyển đổi. Trình bày tham luận chi tiết về một không gian màu (định nghĩa, ưu nhược điểm, ứng dụng).
- Đọc thêm thuật giải Bresemham vẽ đường thẳng/cong trong thiết bị raster.

Danh sách gợi ý các đề tài miễn thi

1. **Cài đặt**: trò chơi nổ bóng (hay ảnh)gần nhau.
2. **Cài đặt**: minh hoạ một vài vấn đề của hình không gian (2 nhóm) dựa trên thư viện 3D OpenGL.
3. **Cài đặt**: lập trình minh họa các công thức phân tử hoá học với thư viện PaperVision 3D.
4. **Cài đặt:** viết một ứng dụng web (dùng Flex hoặc SilverLight) minh hoạ ứng dụng đồ hoạ 3D bất kỳ.
5. **Cài đặt**: tìm hiểu và cài đặt đồ hoạ 3D trên thiết bị di động (Android hoặc Windows Mobile)
6. **Lý thuyết:** radiosity - thuật giải render.
7. **Lý thuyết:** morphing-kỹ thuật biến hình.

Tài liệu: Đồ họa máy tính

Mục lục

PHẦN TỔNG QUAN

1. Mục đích yêu cầu

Sau khi học xong môn này, sinh viên cần đạt được các yêu cầu sau:

- Hiểu thế nào là đồ họa trên máy tính.
- Thiết kế và cài đặt được các thuật toán vẽ các đường cơ bản như đường thẳng, đường tròn,...
- Thiết kế và cài đặt được các thuật toán tô một hình.
- Sử dụng được các phép biến hình trong không gian 2 chiều, 3 chiều để làm thay đổi một hình ảnh đã có sẵn.
- Có thể tạo một cửa sổ để cắt - dán một hình.
- Hiểu khái niệm về các tiếp cận để mô phỏng được một hình ảnh trong không gian 3 chiều trên máy tính.

2. Đối tượng sử dụng

Môn kỹ thuật đồ họa được giảng dạy cho sinh viên năm thứ tư của các khoa sau:

- Chuyên ngành công nghệ thông tin.
- Chuyên ngành điện tử (viễn thông, tự động hóa,...)
- Chuyên ngành sư phạm (Toán tin, Lý tin)

3. Nội dung cốt lõi

Giáo trình Kỹ thuật đồ họa bao gồm 7 chương.

- Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
- Chương 2: Các thuật toán tô màu
- Chương 3: Phép biến đổi trong đồ họa 2 chiều
- Chương 4: Tạo cửa sổ và cắt hình
- Chương 5: Đồ họa 3 chiều
- Chương 6: Quan sát ảnh 3 chiều
- Chương 7: Khử các mặt khuất và đường khuất

4. Kiến thức tiên quyết

- Kiến thức về hình học không gian và hình giải tích
- Kiến thức lập trình căn bản, lập trình đồ họa
- Kiến thức về cấu trúc dữ liệu, lập trình đệ qui

Trang 4 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản

5. Danh mục tài liệu tham khảo

- Donald Hearn, M. Pauline Baker; Computer Graphics; Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986.
- F.S.Hill; Computer graphics ; 1990
- Vũ Mạnh Tường, Dương Anh Đức, Trần Đan Thư, Lý Quốc Ngọc. Giáo trình Nhập môn đồ họa & xử lý ảnh.1995.
- VERA B.ANAND, người dịch TS Nguyễn Hữu Lộc. Đồ họa máy tính và Mô hình hóa hình học. Nhà xuất bản Thành Phố Hồ Chí Minh - 2000.
- Foley, Van Darn, Feiner, Hughes, Phillips. Introduction à L'Infographie. 1995.
- Lê Tấn Hùng, Huỳnh Quyết Thắng. Kỹ thuật đồ họa. Nhà xuất bản khoa học và kỹ thuật, Hà nội - 2000.

Chương 1: GIỚI THIỆU THUẬT TOÁN VẼ VÀ TÔ CÁC ĐƯỜNG CƠ BẢN

1.1 Tổng quan

• Mục tiêu của chương 1

Học xong chương này, sinh viên phải nắm bắt được các vấn đề sau:

- Thế nào là hệ đồ họa
- Thiết kế và cài đặt được các thủ tục vẽ và tô các đường cơ bản như đường thẳng, đường tròn, elip, và các đường cong khác.

• Kiến thức cơ bản cần thiết

Các kiến thức cơ bản cần thiết để học chương này bao gồm :

- Các khái niệm toán học về đường thẳng như : đường thẳng là gì : dạng tổng quát phương trình đường thẳng, hệ số góc, tung độ dốc.
- Hiểu rõ hình dáng của đường thẳng phụ thuộc vào hệ số góc như thế nào.
- Phương trình tổng quát của đường tròn, ellippse (không có tham số và có tham số).
- Kĩ thuật lập trình: thiết lập thủ tục, hàm (lưu ý truyền qui chiếu và truyền giá trị).

• Tài liệu tham khảo

Donald Hearn, M. Pauline Baker. Computer Graphics . Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986 (chapters 3, 55-76).

• Nội dung cốt lõi

Thiết lập thủ tục vẽ :

- Đường thẳng bằng giải thuật DDA
- Đường thẳng bằng giải thuật Bresenham
- Đường tròn bằng giải thuật đối xứng
- Đường tròn bằng giải thuật Bresenham
- Đường tròn bằng giải thuật MidPoint
- Ellippse
- Đa giác

1.2. Hệ tọa độ thế giới thực, hệ tọa độ thiết bị và hệ tọa độ chuẩn

Một hệ mềm đồ họa được mô tả bao gồm 3 miền như sau :
- Miền điều khiển : bao bọc toàn bộ hệ thống.
- Miền thực : nằm trong miền điều khiển. Khi một số nào đó thâm nhập vào miền thực, nó sẽ được chuyển thành số thực dấu phẩy động, và khi có một số rời khỏi miền này thì nó sẽ được chuyển thành số nguyên có dấu 16 bits.
- Miền hiển thị : nằm trong miền điều khiển nhưng phân biệt với miền thực. Chỉ có số nguyên 16 bits mới nằm trong miền hiển thị.

Trong lĩnh vực kỹ thuật đồ họa, chúng ta phải hiểu được rằng thực chất của đồ họa là làm thế nào để có thể mô tả và biến đổi được các đối tượng trong thế giới thực trên máy tính. Bởi vì, các đối tượng trong thế giới thực được mô tả bằng tọa độ thực. Trong khi đó, hệ tọa độ thiết bị lại sử dụng hệ tọa độ nguyên để hiển thị các hình ảnh. Đây chính là vấn đề cơ bản cần giải quyết. Ngoài ra, còn có một khó khăn khác nữa là với các thiết bị khác nhau thì có các định nghĩa khác nhau. Do đó, cần có một phương pháp chuyển đổi tương ứng giữa các hệ tọa độ và đối tượng phải được định nghĩa bởi các thành phần đơn giản như thế nào để có thể mô tả gần đúng với hình ảnh thực bên ngoài.

Hai mô hình cơ bản của ứng dụng đồ họa là dựa trên mẫu số hóa và dựa trên đặc trưng hình học. Trong ứng dụng đồ họa dựa trên mẫu số hóa thì các đối tượng đồ họa được tạo ra bởi lưới các pixel rời rạc. Các pixel này có thể đuợc tạo ra bằng các chương trình vẽ, máy quét, ... Các pixel này mô tả tọa độ xác định vị trí và giá trị mẫu. Thuận lợi của ứng dụng này là dể dàng thay đổi ảnh bằng cách thay đổi màu sắc hay vị trí của các pixel, hoặc di chuyển vùng ảnh từ nơi này sang nơi khác. Tuy nhiên, điều bất lợi là không thể xem xét đối tượng từ các góc nhìn khác nhau. Ứng dụng đồ họa dựa trên đặc trưng hình học bao gồm các đối tượng đồ họa cơ sở như đoạn thẳng, đa giác,.... Chúng được lưu trữ bằng các mô hình và các thuộc tính. Ví dụ : đoạn thẳng được mô hình bằng hai điểm đầu và cuối, có thuộc tính như màu sắc, độ dày. Người sử dụng không thao tác trực tiếp trên các pixel mà thao tác trên các thành phần hình học của đối tượng.

a. Hệ tọa độ thế giới thực:

Một trong những hệ tọa độ thực thường được dùng để mô tả các đối tượng trong thế giới thực là hệ tọa độ Descartes. Với hệ tọa độ này, mỗi điểm P được biểu diễn bằng một cặp tọa độ (xp,yp) với xp, yp $\in$ R (xem hình 1.1).

Y
X
yp
xp O
P(xp,yp)

Hình 1.1 : Hệ tọa độ thực.

. Ox : gọi là trục hoành.
. Oy : gọi là trục tung.
. xp : hoành độ điểm P.
. yp : tung độ điểm P.

b. Hệ tọa độ thiết bị

Hệ tọa độ thiết bị (device coordinates) được dùng cho một thiết bị xuất cụ thể nào đó, ví dụ như máy in, màn hình,..

Trong hệ tọa độ thiết bị thì các điểm cũng được mô tả bởi cặp tọa độ (x,y). Tuy nhiên, khác với hệ tọa độ thực là x, y ∈ N. Điều này có nghĩa là các điểm trong hệ tọa độ thực được định nghĩa liên tục, còn các điểm trong hệ tọa độ thiết bị là rời rạc. Ngoài ra, các tọa độ x, y của hệ tọa độ thiết bị chỉ biểu diễn được trong một giới hạn nào đó của N. Ví dụ : Độ phân giải của màn hình trong chế độ đồ họa là 640x480. Khi đó, x∈(0,640) và y∈(0,480) (xem hình 1.2).

(0,0) (640,0)
(0, 480) (640,480)
Hình 1.2 : Hệ tọa độ trên màn hình.

c. Hệ tọa độ thiết bị chuẩn (Normalized device coordinates)
Do cách định nghĩa các hệ tọa độ thiết bị khác nhau nên một hình ảnh hiển thị được trên thiết bị này là chính xác thì chưa chắc hiển thị chính xác trên thiết bị khác. Người ta xây dựng một hệ tọa độ thiết bị chuẩn đại diện chung cho tất cả các thiết bị để có thể mô tả các hình ảnh mà không phụ thuộc vào bất kỳ thiết bị nào.
Trong hệ tọa độ chuẩn, các tọa độ x, y sẽ được gán các giá trị trong đoạn từ [0,1]. Như vậy, vùng không gian của hệ tọa độ chuẩn chính là hình vuông đơn vị có góc trái dưới (0, 0) và góc phải trên là (1, 1).
Quá trình mô tả các đối tượng thực như sau (xem hình 1.3):
Ảnh định nghĩa
trên tọa độ thế
giới thực.
Tọa độ chuẩn hóa Tọa độ thiết bị
màn hình
máy in
thiết bị
khác

Hình 1.3 : Hệ tọa độ trên màn hình.
1.3. Thuật toán vẽ đoạn thẳng
Xét đoạn thẳng có hệ số góc 0<m<=1 và Δx>0. Với các đoạn thẳng dạng này, nếu (xi, yi) là điểm đã được xác định ở bước thứ i thì điểm kế tiếp (xi+1, yi+1) ở bước thứ i+1 sẽ
là một trong hai điểm sau (xem hình vẽ 1.4) :
xi+1= xi + 1

yi+1= yi + 1
yi

(xi,yi)
(xi+1,yi+1)
(xi+2,yi+2)

(xi+3,yi+2)
(xi+4,yi+3)
Hình 1.4 : Các điểm vẽ gần với điểm muốn vẽ.
Vấn đề đặt ra là chọn điểm vẽ như thế nào để đường thẳng được vẽ gần với đường thẳng muốn vẽ nhất và đạt được tối ưu hóa về mặt tốc độ ?

1.3.1. Thuật toán DDA (Digital DifferentialAnalyzer)

Là thuật toán tính toán các điểm vẽ dọc theo đường thẳng dựa vào hệ số góc của phương trình đường thẳng y=mx+b.
Trong đó, m=
x
y
Δ
Δ
, Δy = yi+1 - yi , Δx = xi+1 - xi
Nhận thấy trong hình vẽ 1.4 thì tọa độ của điểm x sẽ tăng 1 đơn vị trên mỗi điểm vẽ, còn việc quyết định chọn yi +1 là yi +1 hay yi sẽ phụ thuộc vào giá trị sau khi làm tròn
của tung độ y. Tuy nhiên, nếu tính trực tiếp giá trị thực của y ở mỗi bước từ phương trình y=mx+b thì cần một phép toán nhân và một phép toán cộng số thực.
yi +1 = mxi +1 + b = m(xi + 1) + b = mxi + b + m
Để cải thiện tốc độ, người ta khử phép nhân trên số thực.
Ta có : yi = mxi + b
⇒ yi +1 = yi + m → int(yi +1)
• Tóm lại khi 0<m<=1 :
xi +1 = xi + 1
yi +1 = yi + m → int(yi +1)
• Trường hợp m>1: chọn bước tăng trên trục y một đơn vị.
xi +1 = xi + 1/m → int(xi +1)
yi +1 = yi + 1
Trang 10 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
Hai trường hợp này dùng để vẽ một điểm bắt đầu từ bên trái đến điểm cuối cùng bên phải của đường thẳng (xem hình 1.5). Nếu điểm bắt đầu từ bên phải đến điểm cuối cùng bên trái thì xét ngược lại :
• 0<m<=1: xi +1 := xi - 1
yi +1:= yi - m → int(yi+1)

• m>1: xi +1:= xi – 1/m → int(xi+1)
yi +1:= yi – 1

Hình 1.5 : Hai dạng đường thẳng có 0<m<1 và m>1.

Tương tự, có thể tính toán các điểm vẽ cho trường hợp m<0: khi |m|<=1 hoặc |m|>1 (sinh viên tự tìm hiểu thêm).
Trang 11 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
Lưu đồ thuật toán DDA

Begin
dx=x2-x1
dy=y2-y1
abs(dx)>abs(dy)
step=abs(dx) step=abs(dy)
x_inc=dx/step
y_inc=dy/step
x=x1;y= y1
putpixel(x1,y1,c)
k<=step
x = x+x_inc
y = y+y_inc
putpixel(round(x),round(y),c)
End
No Yes
No
Yes

Cài đặt minh họa thuật toán DDA

```
Procedure DDA ( x1, y1, x2, y2, color : integer );
Var  dx, dy, step : integer;
X_inc, y_inc , x, y : real ;
Begin
dx:=x2-x1;
dy:=y2-y1;
if abs(dx)>abs(dy) then   steps:=abs(dx)
else  steps:=abs(dy);
x_inc:=dx/steps;
y_inc:=dy/steps;
x:=x1;  y:=y1;
putpixel(round(x),round(y), color);
for k:=1 to steps do
begin
x:=x+x_inc;
y:=y+y_inc;
putpixel(round(x),round(y),  color);
end;
end;
```

1.3.2. Thuật toán Bresenham

xi
yi+1
yi
P2
yi+1
d2
d1
xi+1 = xi+1
P1

Hình 1.6 : Dạng đường thẳng có 0<=m<=1.

Trang 13 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản

Gọi (xi +1,yi +1) là điểm thuộc đoạn thẳng (xem hình 1.6). Ta có y:= m(xi +1)+b.

Đặt d1 = yi +1 - yi

d2 = (yi +1) - yi +1

Việc chọn điểm (xi +1, yi +1) là P1 hay P2 phụ thuộc vào việc so sánh d1 và d2 hay dấu của

d1-d2.

- Nếu d1-d2<0 : chọn điểm P1, tức là yi +1= yi
- Nếu d1-d2 ≥0 : chọn điểm P2, tức là yi +1= yi +1

Xét Pi = Δx (d1 - d2)

Ta có : d1 - d2 = 2 yi+1 - 2yi - 1

= 2m(xi+1) + 2b - 2yi - 1

⇒ Pi = Δx (d1 - d2) = Δx[2m(xi+1) + 2b - 2yi - 1]

= Δx[2

Δ

Δ

y

x

(xi+1) + 2b - 2yi - 1]

= 2Δy(xi+1) - 2Δx.yi + Δx(2b - 1)

= 2Δy.xi - 2Δx.yi + 2Δy + Δx(2b - 1)

Vậy C = 2Δy + Δx(2b - 1) = Const

⇒ Pi = 2Δy.xi - 2Δx.yi + C

Nhận xét rằng nếu tại bước thứ i ta xác định được dấu của Pi thì xem như ta xác định được điểm cần chọn ở bước (i+1). Ta có :

Pi +1 - Pi = (2Δy.xi+1 - 2Δx.yi+1 + C) - (2Δy.xi - 2Δx.yi + C)

⇔ Pi +1 = Pi + 2Δy - 2Δx (yi+1 - .yi)

- Nếu Pi < 0 : chọn điểm P1, tức là yi +1= yi và Pi +1 = Pi + 2Δy.
- Nếu Pi ≥ 0 : chọn điểm P2, tức là yi +1= yi +1 và Pi +1 = Pi + 2Δy - 2Δx
- Giá trị P0 được tính từ điểm vẽ đầu tiên (x0 ,y0) theo công thức :

P0 = 2Δy.x0 - 2Δx.y0 + C

Do (x0 ,y0) là điểm nguyên thuộc về đoạn thẳng nên ta có :

y0 = m .x0 + b =

Δ
Δ
y
x
.x0 + b
Thế vào phương trình trên ta được :
P0 = 2Δy - Δx

Trang 14 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
Lưu đồ thuật toán Bresenham

Begin
dx = x2-x1; dy = y2 - y1;
P = 2dy-dx; c1 = 2dy; c2 = 2(dy-dx);
x = x1; y = y1;
putpixel (x,y,color);
x < x2
P < 0
P = P + c1
End
No
Yes
No
Yes
P = P + c2
y = y + 1
x = x +1
putpixel(x,y,color)
Trang 15 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
Cài đặt minh họa thuật toán Bresenham

```
Procedure Bres_Line (x1,y1,x2,y2 : integer);
Var dx, dy, x, y, P, const1, const2 : integer;
Begin
dx : = x2 - x1;  dy : = y2 - y1;
P : = 2*dy - dx;
Const1 : = 2*dy ; const2 : = 2*(dy - dx) ;
x:= x1; y:=y1;
Putpixel ( x, y, Color);
while (x < x-2 ) do
begin
x : = x +1 ;
if (P < 0) then P : = P + const1
else
begin
y : = y+1 ;
```

P : = P + const2
end ;
putpixel (x, y, color) ;
end ;
End ;

Nhận xét :
Thuật toán Bresenham chỉ thao tác trên số nguyên và chỉ tính toán trên phép cộng và phép nhân 2 (phép dịch bit). Điều này là một cải tiến làm tăng tốc độ đáng kể so với thuật toán DDA.
Ý tưởng chính của thuật toán này là ở chổ xét dấu Pi để quyết định điểm kế tiếp, và sử dụng công thức truy hồi Pi +1 - Pi để tính Pi bằng các phép toán đơn giản trên số nguyên.
Tuy nhiên, việc xây dựng trường hợp tổng quát cho thuật toán Bresenham có phức tạp hơn thuật toán DDA.

Trang 16 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
1.4. Thuật toán vẽ đường tròn
Trong hệ tọa độ Descartes, phương trình đường tròn bán kính R có dạng:
Với tâm O(0,0) : x2
+ y2
= R2

Với tâm C(xc,yc): (x - xc)
2
+ (y - yc)
2
= R2

Trong hệ tọa độ cực :
x = xc + R.cosθ
y = yc + Y.sinθ
với θ ∈ [0, 2π].
Do tính đối xứng của đường tròn C (xem hình 1.7) nên ta chỉ cần vẽ 1/8 cung tròn, sau đó lấy đối xứng qua 2 trục tọa độ và 2 đường phân giác thì ta vẽ được cả đường tròn.
(x,y)
(y,x)
(y,-x)
(x,-y) (-x,-y)
(-y,-x)
(-y,x)
(-x,y)
x
y
2
2 R

Hình 1.7 : Đường tròn với các điểm đối xứng.

1.4.1. Thuật toán đơn giản

Cho x = 0, 1, 2, ..., int(
2
2 R) với R>1.

- Tại mỗi giá trị x, tính int(y =
22
xR −).
- Vẽ điểm (x,y) cùng 7 điểm đối xứng của nó.

Cài đặt minh họa thuật toán đơn giản.


```
Procedure Circle (xc, yc, R : integer) ;
;
xc + x , yc +y, color) ;

End
0 to round(R*Sqrt(2)/2) do

und(Sqrt(R*R - x*x)) ;
```

1.4.2. Thuật toán xét điểm giữa (MidPoint)

chỉ cần vẽ 1/8 cung tròn, sau đó lấy đối

```
Var x, y : integer ;
Procedure DOIXUNG
Begin
putpixel (
putpixel (xc - x , yc + y, color) ;
putpixel (xc + x , yc - y, color) ;
putpixel (xc -  x , yc- y, color) ;
putpixel (xc + y , yc + x, color) ;
putpixel (xc - y , yc  + x, color) ;
putpixel (xc + y , yc - x, color) ;
putpixel (xc - y , yc  - x, color) ;
;
Begin
For x : =
Begin
y : = ro
DOIXUNG;
End ;
End ;
```

Do tính đối xứng của đường tròn nên ta
xứng là vẽ được cả đường tròn. Thuật toán MidPoint đưa ra cách chọn yi+1 là yi hay yi-1
bằng cách so sánh điểm thực Q(xi+1,y) với điểm giữa MidPoind là trung điểm của S1 và S2. Chọn điểm bắt đầu để vẽ là (0,R). Giả sử (xi, yi) là điểm nguyên đã tìm được ở bước thứ i (xem hình 1.8), thì điểm (xi+1, yi+1) ở bước i+1 là sự lựa chọn giữa S1 và S2.
x = x + 1 i+1 i

yi+1= yi - 1
yi
Trang 18 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
(xi,yi) S1
S2
yi
yi+1
yi - 1
Q(xi+1,y)
MidPoint

Hình 1.8 : Đường tròn với điểm Q(xi+1, y) và điểm MidPoint.
Đặt F(x,y) = x2
+ y2
- R2
, ta có :
. F(x,y) < 0 , nếu điểm (x,y) nằm trong đường tròn.
. F(x,y) = 0 , nếu điểm (x,y) nằm trên đường tròn.
. F(x,y) > 0 , n
i - 1/2). Ta có :
n. Khi đó, điểm thực Q gần với điểm
n. Khi đó, điểm thực Q gần với điểm
1 - 1/2) - F(xi + 1, yi - 1/2)
+ (yi)
2
) - (yi+1 - yi)
+3
Nếu
Pi ứn
P0 = F(x0 + 1, y0 - 1/2) = F(1, R - 1/2) =
ếu điểm (x,y) nằm ngoài đường tròn.
Xét Pi = F(MidPoint) = F(xi +1, y
- Nếu Pi < 0 : điểm MidPoint nằm trong đường trò
S1 hơn nên ta chọn yi+1 = yi .
- Nếu Pi >= 0 : điểm MidPoint nằm ngòai đường trò
S2 hơn nên ta chọn yi+1 = yi - 1.
Mặt khác :
Pi+1 - Pi = F(xi+1 +1, yi+

= [(xi+1 +1)
2
+ (yi+1 - 1/2)
2
- R2
] - [(xi +1)
2
+ (yi - 1/2)
2
- R2
]
= 2xi + 3 + ((yi+1)
2
Vậy :
- Nếu Pi < 0 : chọn yi+1 = yi. Khi đó Pi+1 = Pi + 2xi
- Pi >= 0 : chọn yi+1 = yi - 1. Khi đó Pi+1 = Pi + 2xi - 2yi +5.
- g với điểm ban đầu (x0 , y0) = (0,R) là:

4
5
-R

Lưu đồ thuật toán MidPoint vẽ đường tròn

Begin
P = 5/4 - R;
x=0 ; y= R;
Putpixel(x,y,c);
x < y
P < 0
P = P + 2*x + 3
End
No
Yes
No
Yes
P = P + 2*(x-y)+5
y = y -1
x = x +1
putpixel(x,y,color)

Minh họa thuật toán MidPoint:

```
Procedure DTR(xc, yc, r, mau : integer);
var  x, y, p : integer ;
begin
x:=0 ;  y:=r;
p:=1 - r;
while ( y > x)  do
begin
doi_xung;
if (p<0) then p:=p+2*x+3
else begin
p:=p+2*(x-y)+5 ;
y:=y-1;
end;
x:=x+1;
end; {while}
end;
```

1.4.3. Vẽ đường tròn bằng thuật toán Bresenham

Tương tự thuật toán vẽ đường thẳng Bresenham, các vị trí ứng với các tọa độ nguyên nằm trên đường tròn có thể tính được bằng cách xác định một trong hai pixel gần nhất với đường tròn thực hơn trong mỗi bước (xem hình 1.9).

(xi,yi) S1
S2
yi
yi+1 = y
yi - 1
d1
d2

Hình 1.9 : Đường tròn với khoảng cách d1 và d2.

Ta có :

d1 = (yi)
2
- y2
= (yi)
2
- (R2
- (xi + 1)
2
)
d2 = y2
- (yi - 1)
2
= (R2

- (xi + 1)
2
) - (yi - 1)
2

Pi = d1 - d2
Tính Pi+1 - Pi
⇒ Pi+1 = Pi + 4xi + 6 + 2((yi+1)
2
- (yi)
2
) - 2(yi+1 - yi)
- Nếu Pi < 0 : chọn yi+1 = yi. Khi đó Pi+1 = Pi + 4xi +6
- Nếu Pi >= 0 : chọn yi+1 = yi - 1. Khi đó Pi+1 = Pi + 4(xi - yi) + 10.
- P0 ứng với điểm ban đầu (x0 , y0) = (0,R) là: P0= 3 - 2R.

Minh họa thuật toán vẽ đường tròn bằng Bresenham
P Pr ro oc ce ed du ur re e D DT TR R_ _B BR RE ES S((x xc c, ,y yc c, ,r r, ,m ma au u : : i in nt te eg ge er r)); ;
v va ar r x x, ,y y, ,p p: :i in nt te eg ge er r; ;
b be eg gi in n
x x: := =0 0 ; ; y y: := =r r; ;
p p: := = 3 3 – – 2 2* *r r ; ;
w wh hi il le e ((x x< <y y)) d do o
b be eg gi in n
d do oi i_ _x xu un ng g; ;
i if f ((p p< <0 0)) t th he en n p p: := = p p + + 4 4* *x x + + 6 6
e el ls se e b be eg gi in n
p p: := = p p + + 4 4* *((x x- -y y)) + + 1 10 0 ; ;
y y: := =y y- -1 1; ;
e en nd d; ;
x x: := =x x+ +1 1; ;
e en nd d; ;{ {w wh hi il le e} }
e en nd d; ;
1.4.4. Thuật toán vẽ Ellipse
Tương tự thuật toán vẽ đường tròn, sử dụng thuật toán Bresenham để vẽ, ta chỉ cần vẽ 1/4 ellipse, sau đó lấy đối xứng qua các trục tọa độ sẽ vẽ được toàn bộ ellipse.
Xét ellipse có tâm O, các bán kính là a và b, phương trình là : 1 2
2
2
2
=+
b
y
a
x

Chọn tọa độ pixel đầu tiên cần hiển thị là (xi ,yi) = (0,b). Cần xác định pixel tiếp theo là (xi+1 ,yi+1). Ta có :
xi+1= xi + 1

yi+1= yi - 1
yi

d1 = (yi)
2
- y2

d2 = y2
- (yi - 1)
2
Pi = d1 - d2
Tính Pi+1 - Pi
⇒ Pi+1 = Pi + 2((yi+1)
2
- (yi)
2
) - 2(yi+1 - yi) +
2
2
2
a
b
(2xi + 3)
- Nếu Pi < 0 : chọn yi+1 = yi. Khi đó Pi+1 = Pi +
2
2
2
a
b
(2xi +3)
- Nếu Pi >= 0 : chọn yi+1 = yi - 1. Khi đó Pi+1 = Pi + 2
2
2
a
b
(2xi +3) +4(1- yi)
- Pi ứng với điểm ban đầu (x0 , y0) = (0,b) là: P0 =
2
2
2

a
b
- 2b + 1

Minh họa thuật toán vẽ Ellipse

```
Procedure Ellipse(xc,yc,a,b : integer);
var x,y : integer;
z1, z2, P : real;
procedure dx;
begin
putpixel (xc + x , yc +y, color) ;
putpixel (xc - x , yc + y, color) ;
putpixel (xc + x , yc - y, color) ;
putpixel (xc -  x , yc- y, color) ;
end;
begin
x:=0 y:=b;
```



```
z1:= (b*b)/(a*a);
z2:= 1/ z1;
P:= 2*z1 - 2*b +1;
while (z1* (x/y) ≤ 1) do
begin
dx;
if  P < 0 then P:= P + 2*z1*(2*x+3)
else begin
P:= P + 2*z1*(2*x+3) + 4*(1-y);
y:= y -1;
end;
x:= x+1;
end;
x:=a ; y:= 0;
P:= 2*z2 - 2*a +1;
while (z2* (y/x) < 1) do
begin
dx;
if  P < 0 then P:= P + 2*z2*(2*y+3)
else begin
P:= P + 2*z2*(2*y+3) + 4*(1-x);
x:= x -1;
end;
y:= y +1;
end;
end;
```

1.4.5. Vẽ đường conics và một số đường cong khác

Phương trình tổng quát của các đường conics có dạng :

Ax2
+ Bxy + Cy2
+ Dx + Ey + F = 0
Trang 24 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
Giá trị của các hằng số A, B, C, D, E, F sẽ quyết định dạng của đường conics, cụ thể là nếu:
B2
- 4AC < 0 : dạng đường tròn (nếu A=C và B=0) hay ellipse.
B2
- 4AC = 0 : dạng parabol.
B2
- 4AC > 0 : dạng hyperbol.
Áp dụng ý tưởng của thuật toán Midpoint để vẽ các đường conics và một số đường cong khác theo các bước theo các bước tuần tự sau:
- Bước 1: Dựa vào dáng điệu và phương trình đường cong, để xem thử có thể rút gọn phần đường cong cần vẽ hay không.
- Bước 2: Tính đạo hàm, từ đó phân thành các vùng vẽ.
. Nếu $0 \le f'(x) \le 1$: xi+1 = xi + 1; yi+1 = yi (hoặc = yi +1)
. Nếu $-1 \le f'(x) \le 0$: xi+1 = xi + 1; yi+1 = yi (hoặc = yi - 1)
. Nếu $f'(x) > 1$: yi+1 = yi + 1; xi+1 = xi (hoặc = xi +1)
. Nếu $f'(x) < -1$: yi+1 = yi + 1; xi+1 = xi (hoặc = xi +1)
- Bước 3 : Tính Pi cho từng trường hợp để quyết định f '(x) dựa trên dấu của Pi. Pi thường là hàm được xây dựng từ phương trình đường cong. Cho Pi=0 nếu (xi , yi) thuộc về đường cong. Việc chọn Pi cần chú ý sao cho các thao tác tínn Pi sau này hạn chế phép toán trên số thực.
- Bước 4 : Tìm mối liên quan của Pi+1 và Pi bằng cách xét hiệu Pi+1 - Pi
- Bước 5 : Tính P0 và hoàn chỉnh thuật toán.

1.4.6. Vẽ đa giác

Đường gấp khúc hở Đường gấp khúc kín
Hình 1.10 : Hai dạng của đường gấp khúc.

Trang 25 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
• Định nghĩa đa giác (Polygone): Đa giác là một đường gấp khúc kín có đỉnh đầu và đỉnh cuối trùng nhau (xem hình 1.10)
• Xây dựng cấu trúc dữ liệu để vẽ đa giác

```
Type
d_dinh = record
x,y: longint;
end;
```

dinh = array[0..10] of d_dinh;
var
d: dinh;
Với cách xây dựng cấu trúc dữ liệu như thế này thì chúng ta chỉ cần nhập vào tọa độ các đỉnh và sau đó gọi thủ tục vẽ đường thẳng lần lượt qua 2 đỉnh như (0, 1), (1,2), ..., (n-1, n), trong đó đỉnh n trùng với đỉnh 0 thì ta sẽ vẽ được toàn bộ đa giác.

- Đa giác được gọi là lồi nếu bất kỳ đường thẳng nào đi qua một cạnh của đa giác thì toàn bộ đa giác nằm về một phía của đường thẳng đó. Ngược lại, nếu tồn tại ít nhất một cạnh của đa giác chia đa giác làm 2 phần thì gọi là đa giác lõm (xem hình 1.11).

d3
d2 d1
d0
P2 P3
P0
P3
P4
P5
P1 P2
d4
P0
P4

Hình 1.11 : Đa giác lồi và đa giác lõm

- Thuật toán kiểm tra một đa giác là lồi hay lõm

Thuật toán 1: Lần lượt thiết lập phương trình đường thẳng đi qua các cạnh của đa giác. Ứng với từng phương trình đường thẳng, xét xem các đỉnh còn lại có nằm về một

phía đối với đường thẳng đó hay không ? Nếu đúng thì kết luận đa giác lồi, ngược lại là đa giác lõm.
Nhận xét : Phương trình đường thẳng y = ax + b chia mặt phẳng ra làm 2 phần.
Các điểm nằm C(xc,yc) trên đường thẳng sẽ có yc > axc + b và các điểm D(xd,yd) nằm phía dưới đường thẳng sẽ có yd < axd + b.
Ví dụ : Cho đường thẳng AB có phương trình y =
2
1
x + 1 và hai điểm C, D có tọa

độ là C(0,4), D(2,0) (xem hình 1.12).
Y
C(0,4)
D(2,0)
A
B
X

Hình 1.12 : Đường thẳng AB và 2 điểm C, D.
Ta có : Yc = 4 > axc + b =
2
1
.0 + 1
và Yd = 0 < axd + b =
2
1
.2 + 1
Vậy hai điểm C, D nằm về hai phía đối với đường thẳng AB.
Thuật toán 2 :
Nhận xét :
Trong mặt phẳng Oxy, cho 2 véc tơ a và b , Tích vô hướng của 2 véc tơ là :
T(a , b) =
yx
yx
bb
aa
= ax* by - ay * bx
Khi đó :
a quẹo trái sang b nếu T ≥0
a quẹo phải sang b nếu T < 0
Trang 27 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
Một đa giác là lồi khi đi dọc theo biên của nó thì chỉ đi theo một hướng mà thôi. Nghĩa là chỉ quẹo phải hay quẹo trái. Ngược lại là đa giác lõm (xem hình 1.13).

Hình 1.13 : Đa giác lồi có 5 đỉnh.

Xét đa giác gồm các đỉnh P0, P1 ,.... Pn , (P0 = Pn) , n ≥ 3 (xem hình 1.13).
Tính Vi = Pi+1 - Pi , ∀ i = 0, 1, ..., n-1.
Tính Ti = T(Vi , Vi+1)

Nếu với mọi Ti đều cùng dấu thì kết luận đa giác lồi.
Ngược lại, là đa giác lõm.

1.4.7. Tổng kết chương 1
- Chương 1 đã trình bày khái niệm về một hệ độ họa, sự hiển thị của điểm trên màn hình với tọa độ phải là số nguyên.
- Phân biệt thế nào là hệ tọa độ thế giới thực, hệ tọa độ thiết bị và hệ tọa độ chuẩn.
- Cần lưu ý về hệ số góc của đường thẳng. Bởi vì, với hệ số góc khác nhau thì giải thuật có thay đổi. Nhất là trong giải thuật Bresenham.
- Chú ý hơn trong cách xây dựng cấu trúc dữ liệu để lưu tọa độ của các đỉnh đa giác.
- So sánh các trường hợp sử dụng công thức của các đường cong (có tham số và không có tham số).

1.4.8. Bài tập chương 1
1. Viết chương trình vẽ bầu trời có 10.000 điểm sao, mỗi điểm sao xuất hiện với một màu ngẫu nhiên. Những điểm sao này hiện lên rồi từ từ tắt cũng rất ngẫu nhiên.

0 V
v

P1 P0
P4
P3
P2

2 V
v

3 V
v

4 V
v

1 V
v

2. Viết chương trình thực hiện 2 thao tác sau :
- Khởi tạo chế độ đồ họa, đặt màu nền, đặt màu chữ, định dạng chữ (settextstyle(f,d,s)), xuất một chuỗi ký tự ra màn hình. Đổi font, hướng, kích thước.
- Xuất một chuỗi ra màn hình, chuỗi này có tô bóng.
(lưu ý rằng nội dung chuỗi ký tự, màu tô, màu bóng là được nhập từ bàn phím).
3. Viết chương trình vẽ đoạn thẳng AB với màu color theo giải thuật DDA. Biết rằng tọa độ A,B, color được nhập từ bàn phím. Trang trí màu nền, ghi chú các tọa độ A, B ở hai đầu đoạn thẳng.
4. Tương tự như bài tập 3 nhưng sử dụng giải thuật Bresenham. Lưu ý các trường

hợp đặc biệt của hệ số góc.
5. Tổng hợp bài tập 4, viết chương trình vẽ đường thẳng bằng giải thuật Bresenham cho tất cả các trường hợp của hệ số góc. Lưu ý xét trường hợp đặc biệt khi đường thẳng song song với trục tung hay với trục hoành.
6. Viết chương trình nhập tọa độ 3 điểm A, B, C từ bàn phím. Tìm tọa độ điểm D thuộc AB sao cho CD vuộng góc AB. Vẽ đoạn thẳng AB và CD.
7. Viết chương trình xét vị trí tương đối của 2 đoạn thẳng AB và CD. Biết rằng trong màn hình đồ họa đoạn thẳng AB và CD được gọi là cắt nhau khi hai điểm A, B ở về hai phía của CD và ngược lại.
8. Viết chương trình vẽ đường tròn theo giải thuật đơn giản (đối xứng).
9. Viết chương trình vẽ đường tròn theo giải thuật Bresenham.
10. Viết chương trình vẽ đường tròn theo giải thuật MidPoint.
11. Viết chương trình vẽ một đường tròn tâm O bán kính R. Vẽ các đường tròn đồng tâm với O, có bán kính chạy từ 1 đến R. Sau đó xoá các đường tròn đồng tâm này và vẽ các đường tròn đồng tâm khác đi từ R đến 1.
12. Viết chương trình vẽ một đường tròn tâm O bán kính R. Hãy vẽ một đoạn thẳng từ tâm O độ dài R. Hãy quay đoạn thẳng này quanh đường tròn.
13. Viết chương trình vẽ Elippse.
14. Viết chương trình vẽ Elippse có bán kính lớn là a, bán kính nhỏ là b và một đường tròn nội tiếp Elippse. Tô đường tròn bằng các đường tròn đồng tâm. Sau đó tô elippse bằng các elippse đồng tâm có bán kính lớn chạy từ b đến a, bán kính nhỏ là b.
Trang 29 Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
15. Viết chương trình vẽ một hình chữ nhật, một hình vuông và một hình bình hành. Yêu cầu chú thích tọa độ các đỉnh.
16. Viết chương trình vẽ một tam giác. Tọa độ các đỉnh được nhập từ bàn phím, mỗi cạnh có một màu khác nhau.
17. Viết chương trình vẽ một đa giác có n đỉnh.
18. Viết chương trình xét tính lồi lõm của một đa giác bằng cách thiết lập phương trình đường thẳng đi qua các cạnh của đa giác.
19. Viết chương trình xét tính lồi lõm của một đa giác bằng cách thiết lập các véc tơ chỉ phương của các cạnh.
Trang 30 Chương 2: Các thuật toán tô màu
Chương 2 : CÁC THUẬT TOÁN TÔ MÀU
2.1. Tổng quan
• Mục tiêu
Học xong chương này, sinh viên phải nắm bắt được các vấn đề sau:
- Hiểu được khái niệm về không gian màu RGB,CMY, HSV.
- Thiết kế và cài đặt được các giải thuật tô màu.
• Kiến thức cơ bản cần thiết
Kiến thức tin học : lập trình cấu trúc dữ liệu, cách lưu trữ và xây dựng mãng dữ liệu chứa các giao điểm của đường thẳng và đa giác.
Kỹ năng lập trình đệ qui, tạo stack khử đệ qui.
• Tài liệu tham khảo
Computer Graphics . Donald Hearn, M. Pauline Baker. Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986 (chapters 4, 78-103)

• Nội dung cốt lõi
- Trình bày các không gian màu RGB, CMY, HSV
- Giới thiệu các thuật toán tô màu bao gồm : tô đơn giản, tô theo đường biên và tô scan-line

2.2. Các không gian màu

2.2.1. Không gian màu RGB (Red - Green - Blue)

Không gian màu RGB mô tả màu sắc bằng 3 thành phần chính là Red - Green và Blue. Không gian này được xem như một khối lập phương 3 chiều với màu red là trục x, màu Green là truc y, và màu Blue là trục z. Mỗi màu trong không gian này được xác định bởi 3 thành phần R, G, B. Ứng với các tổ hợp khác nhau của 3 màu này sẽ cho ta một màu mới (xem hình 2.1).

Hình 2.1 : Không gian màu RGB.

Green Yellow
White
Cyan
(0,1,0) (1,1,0)
(0,1,1)
(1,1,1)
Black
Blue Magenta
Red
0
(0,0,1) (1,0,1)
(1,0,0)
x
z

Nhận xét :

Trong hình lập phương trên (xem hình 2.1), mỗi màu gốc (R,G,B) có các gốc đối diện là các màu bù với nó. Hai màu được gọi là bù nhau khi kết hợp hai màu này

lại với nhau ra màu trắng. Ví dụ : Green - Magenta, Red - Cyan, Blue - Yellow.

2.2.2. Không gian màu CMY (Cyan - Magenta - Yellow)

Tương tự như không gian màu RGB nhưng 3 thành phần chính là Cyan - Magenta - Yellow. Do đó, tọa độ các màu trong không gian CMY trái ngược với không gian RGB. Ví dụ : màu White có các thành phần là (0,0,0), màu Black (1,1,1), màu Cyan (1,0,0),....

2.2.3. Không gian màu HSV (Hue - Saturation - Value)

Thực chất của không gian này là sự biến đổi của không gian RGB. Không gian HSV được mô tả bằng lệnh lập phương RGB quay trên đỉnh Black. H (Hue) là góc quay trục V (value) qua 2 đỉnh Black và White (xem hình 2.2).

Các gía trị biến thiên của H, S, V như sau :

H (Hue) chỉ sắc thái có giá trị từ 00

- 3600

.

S (Saturation) chỉ độ bảo hoà.

V (Value) có giá trị từ 0 - 1. Các màu đạt giá trị bảo hòa khi s = 1 và v = 1.

Trang 32 Chương 2: Các thuật toán tô màu

V V= =1 1

H H

S S

C Cy ya an n

B Bl lu ue e M Ma ag ge en nt ta a

R Re ed d

Y Ye el ll lo ow w G Gr re ee en n

W Wh hi il le e

B Bl la ac ck k

R RG GB B H HS SV V

R Re ed d ((1 1, ,0 0, ,0 0)) ((0 00 0

, ,1 1, ,1 1))

Y Ye el ll lo ow w ((1 1, ,1 1, ,0 0)) ((6 60 00 0

, ,1 1, ,1 1))

Hình 2.2 : Không gian màu HSV.

2.3. Các thuật toán tô màu

Tô màu một vùng là thay đổi màu sắc của các điểm vẽ nằm trong vùng cần tô.

Một vùng tô thường đượcc xác định bởi một đường khép kín nào đó gọi là đường biên.

Dạng đường biên đơn giản thường gặp là đa giác.

Việc tô màu thường chia làm 2 công đoạn :

. Xác định vị trí các điểm cần tô màu.

. Quyết định tô các điểm trên bằng màu nào. Công đoạn này sẽ trở nên phức tạp khi ta cần tô theo một mẫu tô nào đó chứ không phải tô thuần một màu.

Có 3 cách tiếp cận chính để tô màu. Đó là : tô màu theo từng điểm (có thể gọi là tô đơn giản), tô màu theo dòng quét và tô màu dựa theo đường biên.

2.3.1. Tô đơn giản

Thuật toán này bắt đầu từ việc xác định một điểm có thuộc vùng cần tô hay không ? Nếu đúng là điểm thuộc vùng cần tô thì sẽ tô với màu muốn tô.

Trang 33 Chương 2: Các thuật toán tô màu

• Tô đường tròn

- Để tô đường tròn thì ta tìm hình vuông nhỏ nhất ngoại tiếp đường tròn bằng cách xác định điểm trên bên trái (xc-r, yc-r) và điểm dưới bên phải (xc+r, yc+r) của hình vuông (xem hình 2.2).
- Cho i đi từ xc-r đến xc+r

Cho j đi từ yc-r đến yc+r

Tính khoảng cách d giữa hai điểm (i,j) và tâm (xc,yc)

Nếu d<r thì tô điểm (i,j) với màu muốn tô

(0,0) xc - r

yc - r

xc xc + r

r

yc

yc + r

(xc,yc)

Hình 2.3 : đường tròn nội tiếp hình vuông.

• Tô đa giác

- Tìm hình chữ nhật nhỏ nhất có các cạnh song song với hai trục tọa độ chứa đa giác cần tô dưa vào hai tọa độ (xmin, ymin), (xmax, ymax). Trong đó, xmin, ymin là hoành độ và tung độ nhỏ nhất, xmax, ymax là hoành độ và tung độ lớn nhất của các đỉnh của đa giác.
- Cho x đi từ xmin đến xmax, y đi từ ymin đến ymax (hoặc ngược lai). Xét điểm P(x,y) có thuộc đa giác không ? Nếu có thì tô với màu cần tô (xem hình 2.4).

Trang 34 Chương 2: Các thuật toán tô màu

Ymax

Ymin

Y

Xmin Xmax
X

Hình 2.4 : đa giác nội tiếp hình chữ nhật.

Thông thường một điểm nằm trong đa giác thì số giao điểm từ một tia bất kỳ xuất phát từ điểm đó cắt biên của đa giác phải là một số lẻ lần. Đặc biệt, tại các đỉnh cực trị (cực đại hay cực tiểu) thì một giao điểm phải được tính 2 lần (xem hình 2.5). Tia có thể qua phải hay qua trái. Thông thường ta chọn tia qua phải.
Ví dụ : Xét đa giác gồm 13 đỉnh là P0 , P1 , ..., P12 = P0
(xem hình 2.5).
P0
P2
P3
P4 P5
P7 P6
P8
P9
P10
P11
P12
P1
P
Q

Hình 2.5 : Đa giác có 13 đỉnh.
Lưu ý :
Trang 35 Chương 2: Các thuật toán tô màu
Gọi tung độ của đỉnh Pi là Pi.y . Nếu :
- Pi.y < Min (Pi+1.y, Pi-1.y) hay Pi.y > Max (Pi+1.y, Pi-1.y) thì Pi là đỉnh cực trị (cực
tiểu hay cực đại).
- Pi-1.y < Pi.y < Pi+1.y hay Pi-1 > Pi.y > Pi+1.y thì Pi là đỉnh đơn điệu.
- Pi = Pi+1 và Pi.y < Min (Pi+2.y, Pi-1.y) hay Pi > Max (Pi+2.y, Pi-1.y) thì đoạn [Pi,Pi+1] là đoạn cực trị (cực tiểu hay cực đại).
- Pi = Pi+1 và Pi-1.y < Pi.y < Pi+2.y hay Pi-1 > Pi.y > Pi+2.y thì đoạn [Pi,Pi+1] là
đoạn đơn điệu.

• Thuật toán kiểm tra điểm có nằm trong đa giác
- Với mỗi đỉnh của đa giác ta đánh dấu là 0 hay 1 theo qui ước như sau: nếu là đỉnh cực trị hay đoạn cực trị thì đánh số 0. Nếu là đỉnh đơn điệu hay đoạn đơn điệu thì đánh dấu 1.
- Xét số giao điểm của tia nữa đường thẳng từ P là điểm cần xét với biên của đa giác. Nếu số giao điểm là chẵn thì kết luận điểm không thụộc đa giác. Ngược lại, số giao

điểm là lẻ thì điểm thuộc đa giác.

• Minh họa thuật toán xét điểm thuộc đa giác

```
function  PointInpoly(d: dinh;  P: d_dinh; n: integer): boolean;
var  count, i: integer;
x_cut: longint;
function  next(i: integer): integer;
begin
next := (i + n + 1) mod n
end;
function  prev(i: integer): integer;
begin
prev := (i + n - 1) mod n
end;
begin
count := 0;
for i := 0 to n-1 do
```



```
if   d[i].y = P.y   then
begin
if    d[i].x > P.x    then
begin
if ((d[prev(i)].y < P.y) and (P.y < d[next(i)].y)) or
((d[prev(i)].y > P.y) and (P.y > d[next(i)].y)) then
count := count + 1;
if  d[next(i)].y = P.y then
if ((d[prev(i)].y < P.y) and (P.y < d[next(next(i))].y)) or
((d[prev(i)].y > P.y  and (P.y > d[next(next(i))].y)) then
count := count + 1;
end;
end   else   {d[i].y = P.y}
if ((d[i].y < P.y) and (P.y < d[next(i)].y)) or
((d[i].y > P.y) and (P.y > d[next(i)].y)) then
begin
x_cut := d[i].x + Round((d[next(i)].x - d[i].x)
/ (d[next(i)].y - d[i].y) * (P.y - d[i].y));
if x_cut >= P.x then count := count + 1;
end;
if  (count mod 2 = 0) then  PointInPoly :=  false
else PointInpoly := true;
end;
```

• Minh họa thuật toán tô đa giác

(xmin, ymin, xmax, ymax: đã khai báo trong chương trình chính.)

```
Procedure Todg ( d:dinh; n,maubien : integer ; d: dinh; n:integer ) ;
var  x, y:integer;
```

```
P: d_dinh;
begin
for x:=xmin  to  xmax do
for   y:= ymin to ymax  do
```

```
begin
P.x:= x;   P.y := y;
if  pointInpoly (d, P, n) then
if getpixel(x,y)<>maubien then putpixel(x,y,color);
end;
end;
```

• Nhận xét:

Thuật toán tô đơn giản có ưu điểm là tô rất mịn và có thể sử dụng được cho đa giác lồi hay đa giác lõm, hoặc đa giác tự cắt, đường tròn, ellipse.

Tuy nhiên, giải thuật này sẽ trở nên chậm khi ta phải gọi hàm PointInpoly nhiều lần. Để khắc phục nhược điểm này người ta đưa ra thuật toán tô màu theo dòng quét.

2.3.2. Tô màu theo dòng quét (scan - line)

Phương pháp này sẽ xác định phần giao của các dòng quét kế tiếp nhau với đường biên của vùng tô. Sau đó, sẽ tiến hành tô màu các điểm thuộc phần giao này. Phương pháp này thường được dùng để tô màu đa giác lồi , lõm hay đa giác tự cắt, đường tròn, ellipse, và một số đường cong đơn giản khác.

• Các bước chính của thuật toán

- Tìm ymin, ymax lần lượt là giá trị nhỏ nhất, lớn nhất của tập các tung độ của các đỉnh của đa giác đã cho.
- Ứng với mỗi dòng quét y = k với k thay đổi từ ymin đến ymax, lặp :

. Tìm tất cả các hoành độ giao điểm của dòng quét y = k với các cạnh của đa giác.

. Sắp xếp các hoành độ giao điểm theo thứ tự tăng dần : x0 ,x1 ,..., xn ,...

. Tô màu các đoạn thẳng trên đường thẳng y = k lần lượt được giới hạn bởi các cặp (x0, x1), (x1 ,x2), ...(xem hình 2.6).

Hình 2.6 : Tô đa giác bằng giải thuật scan -line.

• Các vấn đề cần lưu ý:

- Hạn chế được số cạnh cần tìm giao điểm ứng với mỗi dòng quét vì ứng với mỗi dòng quét không phải lúc nào cũng giao điểm với các cạnh của đa giác.
- Xác định nhanh hoàn độ giao điểm vì nếu lặp lại thao tác tìm giao điểm của cạnh đa giác với mỗi dòng quét sẽ tốn rất nhiều thời gian.
- Giải quyết trường hợp số giao điểm đi qua đỉnh đơn điệu thì tính số giao điểm là 1 hay đi qua đỉnh cực trị.thì tính số giao điểm là 0 (hoặc 2).

• Tổ chức cấu trúc dữ liệu và thuật toán

- Danh sách các cạnh (Edge Table - ET) : chứa toàn bộ các cạnh của đa giác (loại các cạnh song song với trục Ox) được sắp theo thứ tự tăng dần của trục y. Xem hình 2.5 ta có thể sắp xếp các cạnh trong ET là : AB, AI, HG, BC, GF, DC, EF (loại IH và DE)
- Danh sách các cạnh đang kích hoạt (Active Edge Table - AET) : chứa các cạnh của đa giác có thể cắt ứng với dòng quét hiện hành, các cạnh này được sắp theo thứ tự tăng dần của hoành độ giao điểm của hoành độ giao điểm giữa cạnh và dòng quét.
- Khi dòng quét đi từ ymin đến ymax, các cạnh thoả điều kiện sẽ được chuyển từ ET sang AET. Nghĩa là, khi dòng quét y=k bắt đầu cắt một cạnh, khi đó k ≥ ymin, cạnh này sẽ được chuyển từ ET sang AET. Khi dòng quét không còn cắt cạnh này nữa, khi

đó, k > ymax, cạnh này sẽ bị loại khỏi AET. Khi không còn cạnh nào trong ET hay AET thì quá trình tô màu kết thúc (xem hình 2.5).

B
D E
F
G
I H
A
yH+1
yH
ymin
C
P
ymax

Hình 2.7 : Tô đa giác bằng giải thuật scan -line.

- Để tìm giao điểm giữa cạnh đa giác và dòng quét, ta có nhận xét sau :
y = k+1
y = k
xk
x k+1

x k+1 - x k =
m
1
((k+1) - k) =
m
1
hay x k+1 = x k +
m
1

Trong đó m là hệ số góc của cạnh.

Lưu đồ thuật toán scan - line

Begin
Tạo danh sách tất cả các cạnh (ET) của đa giác.
i < ymax
Tìm hoành độ giao điểm và sắp xếp theo thứ
tự tăng dần
End
No
Yes
Tô mẫu các đoạn giao được tạo bởi từng
cặp hoành độ kế tiếp nhau
Cập nhật lại thông tin của các cạnh để sử
dụng cho dòng quét kế tiếp
i = ymin
Cập nhật danh sách các cạnh kích hoạt AET
i = i + 1

2.3.3. Phương pháp tô màu dựa theo đường biên

Bài toán đặt ra : Cần tô màu một vùng nếu biết được màu của đường biên vùng tô và một điểm nằm bên trong vùng tô.

Ý tưởng : Bắt đầu từ một điểm nằm bên trong vùng tô, kiểm tra các điểm lân cận của nó đã được tô với màu muốn tô, hay điểm lân cận có màu trùng với màu biên không ? Nếu cả hai trường hợp đều không phải thì ta sẽ tô điểm đó với màu muốn tô. Quá trình này được lặp lại cho đến khi không còn tô được nữa thì dừng (xem hình 2.8).

Hình 2.8 : Tô màu theo đường biên.

Có 2 quan điểm về cách tô này. Đó là dùng 4 điểm lân cận (có thể gọi là 4 liên thông) hay 8 điểm lân cận (8 liên thông) (xem hình 2.9).

(x,y-1)
(x-1,y) (x,y)(x+1,y)

(x,y+1)

Hình 2.9 : 4 liên thông và 8 liên thông.

Cài đặt minh họa thuật toán 4 liên thông

```
Procedure Boundary_fill ( x,y, mauto, maubien :integer);
var mau_ht : integer;
begin
mau_ht:= getpixel(x, y);
if  (mau_ht <> mauto) and (mau_ht <> maubien) then
begin
```



```
putpixel(x,y,color);
Boundary_fill ( x+1,y, mauto, maubien );
Boundary_fill ( x-1,y, mauto, maubien );
Boundary_fill ( x,y+1, mauto, maubien );
Boundary_fill ( x,y-1, mauto, maubien );
end;
end;
```

Nhận xét :
- Thuật toán có thể không chính xác khi có một số điểm nằm trong vùng tô có màu là màu cần tô của vùng.
- Việc thực hiện gọi đệ qui làm thuật toán không thể sử dụng cho vùng tô lớn (tràn stack).
- Có thể khắc phục việc tràn stack bằng cách giảm số lần gọi đệ qui. Khởi đầu điểm (x,y) là điểm có vị trí đặc biệt trong vùng tô, sau đó, gọi đệ qui các điểm lân cận của (x,y) (xem hình 2.8).

Hình 2.10: Tam giác với 3 tọa độ đỉnh.
(100,100)
(100,400)
(500,200)

Ví dụ 1: Trong hình 2.10, ta có thể xét điểm (x,y) có tọa độ là (498, 200). Với điểm khởi đầu này thì chỉ cần xét 3 điểm lân cận là (x-1,y), (x,y-1), (x,y+1). Khi đó thủ tục tô màu theo đường biên được viết lại như sau :

```
Procedure Boundary_fill ( x,y,mauto, maubien :integer);
var mau_ht : integer;
```



```
begin
mau_ht:= getpixel(x,y);
if  (mau_ht <> mauto) and (mau_ht <> maubien) then
begin
putpixel(x,y,color);
Boundary_fill ( x-1,y, mauto, maubien );
Boundary_fill ( x,y+1, mauto, maubien );
```

```
Boundary_fill ( x,y-1, mauto, maubien );
end;
end;
```
Ví dụ 2: Trong hình 2.10, ta có thể xét điểm (x,y) có tọa độ là (102, 102). Với điểm khởi đầu này thì chỉ cần xét 2 điểm lân cận là (x+1,y), (x,y+1). Khi đó thủ tục tô màu theo đường biên được viết lại như sau :

```
Procedure Boundary_fill ( x,y,mauto, maubien :integer);
var mau_ht : integer;
begin
mau_ht:= getpixel(x,y);
if  (mau_ht <> mauto) and (mau_ht <> maubien) then
begin
putpixel(x,y,color);
Boundary_fill ( x+1,y, mauto, maubien );
Boundary_fill ( x,y+1, mauto, maubien );
end;
end;
```


- Một cải tiến khác : không cài đặt đệ qui mà tô theo từng dòng (xem hình 2.11).

Hình 2.10 : Tô theo từng dòng.

2.4. Tổng kết chương 2

- Sinh viên cần hiểu được khái niệm về các không gian màu.

Lưu ý nhiều ở giải thuật tô biên và scan-line.

- Trong scan-line phải đánh dấu các đỉnh đơn điệu và đỉnh cực trị.
- Trong giải thuật tô biên, việc thực hiện gọi đệ qui nhiều lần làm thuật toán không thể sử dụng cho vùng tô lớn (tràn stack). Có thể khắc phục việc tràn stack bằng cách giảm số lần gọi đệ qui. Thực hiện gọi đệ qui tại đỉnh đặc biệt của đa giác.

2.5. Bài tập chương 2

20. Viết chương trình vẽ một đa giác n đỉnh, xét xem một điểm P nào đó có thuộc

đa giác không ?
21. Viết chương trình vẽ một đa giác n đỉnh. Tô đa giác bằng giải thuật tô đơn giản (Tìm xmin, ymin, xmax, ymax).
22. Viết chương trình vẽ một đường tròn. Tô đường tròn bằng giải thuật tô đơn giản.
23. Viết chương trình vẽ một đa giác n đỉnh. Tô đa giác bằng giải thuật tô biên. Lưu ý cho các trường hợp của đa giác : hình chữ nhật, đa giác lồi, đa giác lõm.
24. Viết chương trình vẽ một đường tròn. Tô đường tròn bằng giải thuật tô biên.
25. Viết chương trình vẽ một đa giác n đỉnh. Tô đa giác bằng giải thuật scan-line.
26. Viết chương trình vẽ một đường tròn. Tô đường tròn bằng giải thuật tô scanline.
27. Viết chương trình vẽ hai đường tròn C1 và C2 cắt nhau. Tô phần giao của hai đường tròn đó. Tô phần bù của C2. Tô phần bù của C1. Lưu ý rằng 3 màu tô này phải khác nhau.

Trang 46 Chương 3: Phép biến đổi trong đồ họa hai chiều

Chương 3 : PHÉP BIẾN ĐỔI TRONG ĐỒ HỌA HAI CHIỀU

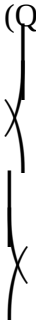
3.1. Tổng quan

• Mục tiêu
- Sinh viên cần hiểu được các phép biến đổi cơ bản trong không gian hai chiều. Nắm vững công thức tổng quát của phép biến đổi Affine, từ đó suy ra các phép tịnh tiến, quay...
- Có khả năng lập trình tạo một hình ảnh động trên máy tính
• Kiến thức cơ bản cần thiết
Kiến thức toán học : hiểu biết về ma trận, định thức. Các phép toán trên ma trận.
• Tài liệu tham khảo
Computer Graphics . Donald Hearn, M. Pauline Baker. Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986 (chapters 5, 106-122).
• Nội dung cốt lõi
Bản chất của phép biến đổi hình học là thay đổi các mô tả về tọa độ của đối tượng như thay đổi về hướng, kích thước, hình dạng. Do đó, chương này trình bày các phép biến đổi như tịnh tiến, tỉ lệ, phép quay, đối xứng, biến dạng.

3.2. Phép tịnh tiến (translation)

Có hai quan điểm về phép biến đổi hình học, đó là :
- Biến đổi đối tượng : thay đổi tọa độ của các điểm mô tả đối tượng theo một qui tắc nào đó.
- Biến đổi hệ tọa độ : Tạo ra một hệ tọa độ mới và tất cả các điểm mô tả đối tượng sẽ được chuyển về hệ tọa độ mới.
Các phép biến đổi hình học cơ sở là : tịnh tiến, quay, biến đổi tỉ lệ.
Phép biến đổi Affine hai chiều (gọi tắc là phép biến đổi) là một ánh xạ T biến đổi điểm P(Px, Py) thành điểm Q(Qx, Qy) theo hệ phương trình sau:
Qx = a*Px + c*Py + trx

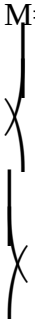
Qy = b*Px + d*Py + try
Trang 47 Chương 3: Phép biến đổi trong đồ họa hai chiều
Hay
(Qx, Qy) = (Px, Py). + (tr



dc
ba
x, try)
⇒ Q = P.M + tr

Dùng để dịch chuyển đối tượng từ vị trì này sang vị trí khác.
Nếu gọi trx và try lần lượt là độ dời theo trục hoành và trục tung thì tọa độ điểm mới Q(x', y') sau khi tịnh tiến điểm P(x,y) sẽ là :
x' = x + trx
y' = y + try
(trx, try) được gọi là vector tịnh tiến hay vector độ dời (xem hình 3.1).
Hay Q = P*M +tr

M= , tr = (tr



10
01
x, try)

Hình 3.1 : Phép biến đổi tịnh tiến điểm P thành Q.
3.3. Phép biến đổi tỷ lệ

Phép biến đổi tỉ lệ làm thay đổi kích thước đối tượng. Để co hay giãn tọa độ của một điểm P(x,y) theo trục hoành và trục tung lần lượt là Sx và Sy (gọi là các hệ số tỉ lệ), ta nhân Sx và Sy lần lượt cho các tọa độ của P.

x' = x.Sx

y' = y.Sy

Q(x',y')

P(x,y)

x' x

y

y'

O

trx

try

- Khi các giá trị Sx , Sy nhỏ hơn 1, phép biển đổi sẽ thu nhỏ đối tượng. Ngược lại, khi các giá trị này lớn hơn 1, phép biến đổi sẽ phóng lớn đối tượng.

- Khi Sx = Sy , người ta gọi đó là phép đồng dạng (uniform scaling). Đây là phép biến đổi bảo toàn tính cân xứng của đối tượng. Ta gọi là phép phóng đại nếu |S|>1 và là phép thu nhỏ nếu |S|<1.
- Nếu hai hệ số tỉ lệ khác nhau thì ta gọi là phép không đồng dạng. Trong trường hợp hoặc Sx hoặc Sy có giá trị 1, ta gọi đó là phép căng (strain).

3.4. Phép quay

Phép quay làm thay đổi hướng của đối tượng. Một phép quay đòi hỏi phải có tâm quay, góc quay. Góc quay dương thường được qui ước là chiều ngược chiều kim đồng hồ.

- Phép quay quanh gốc tọa độ

Ta có công thức biến đổi của phép quay điểm P(x,y) quanh gốc tọa độ góc θ (xem hình 3.2):

x' = x.cosθ - y.sinθ

y' = x.sinθ + y.cosθ

Hay Q = P*M

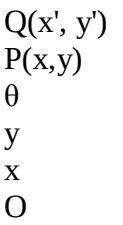
với M =

− θθ

θθ

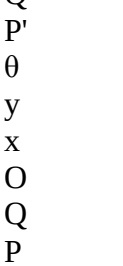
cossin

sincos



Hình 3.2 : Phép quay quanh gốc tọa độ.

- Phép quay quanh một điểm bất kỳ



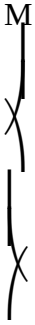
Hình 3.3 : Phép quay quanh một điểm bất kỳ.

Xét điểm P(P.x,P.y) quay quanh điểm V(V.x, V.y) một góc θ đến điểm Q(Q.x,Q.y). Ta có thể xem phép quay quanh tâm V được kết hợp từ phép các biến đổi cơ bản sau:

- Phép tịnh tiến (-V.x, -V.y) để dịch chuyển tâm quay về gốc tọa độ
- Quay quanh gốc tọa độ O một góc θ
- Phép tịnh tiến (+V.x, +V.y) để đưa tâm quay về vị trí ban đầu

Ta cần xác định tọa độ của điểm Q (xem hình 3.3).
- Từ phép tịnh tiến (-V.x,-V.y) biến đổi điểm P thành P' ta được:
P' = P + V
hay P'.x = P.x - V.x
P'.y = P.y - V.y
- Phép quay quanh gốc tọa độ biến đổi điểm P' thành Q'
Q' = P'.M
Q'.x = P'.x*cosθ - P'.y*sinθ
Q'.y = P'.x*sinθ + P'.y*cosθ
- Phép tịnh tiến (+V.x, +V.y) biến đổi điểm Q' thành Q ta được
Q = Q' + V
hay Q.x = Q'.x + V.x
Q.y = Q'.y + V.y
Q.x = (P.x - V.x)*cosθ - (P.y - V.y)*sinθ + V.x

Q.y = (P.x - V.x)*sinθ + (P.y - V.y)*cosθ + V.y
Q.x = P.x*cosθ - P.y*sinθ + V.x*(1- cosθ) + V.y*sinθ
Q.y = P.x*sinθ + P.y*cosθ - V.x*sinθ + V.y*(1- cosθ)
Vậy Q = P.M + tr.
Với
M =



− θθ
θθ
cossin
sincos
tr = (V.x*(1- cosθ) + V.y*sinθ , - V.x*sinθ + V.y*(1- cosθ))
3.5. Phép đối xứng
Phép đối xứng trục có thể xem là phép quay quanh trục đối xứng mõt góc 1800
.
Phương trình ban đầu :
Q.x = a*P.x + c*P.y + trx
Qy = b*P.x + d*P.y + try
Hay
(Q.x, Q.y) = (P.x, P.y). + (tr

dc
ba
x, try)
Trục đối xứng là trục hoành :
M =

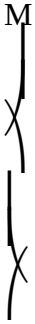
−10
01
Ta có :
Q.x = P.x
Q.y = - P.y
Tương tự trục đối xứng là trục tung :
Ta có :
Q.x = - P.x
Q.y = P.y

M =

–
10
01
3.6. Phép biến dạng
Phép biến dạng biến đổi làm thay đổi, méo mó hình dạng của các đối tượng.
- Biến dạng theo phương trục x sẽ làm thay đổi hoành độ còn tung độ giữ nguyên.
Trang 51 Chương 3: Phép biến đổi trong đồ họa hai chiều
Ví dụ : biến đổi điểm P(P.x, P.y) thành điểm Q(Q.x, Q.y) theo phương trục x là phép biến đổi được biểu diễn bởi phương trình sau :

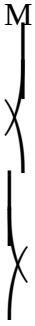
Q.x = P.x + h*P.y

Q.y = P.y
M =



1
01
h
- Biến dạng theo phương trục y sẽ làm thay đổi tung độ còn hoành độ giữ nguyên.
Q.x = P.x
Q.y = g*P.x + P.y

M =



10
1 g
3.7. Phép biến đổi Affine ngược (The inverse of an Affine transformation)
Phép biến đổi ngược dùng để undo một phép biến đổi đã thực hiện.
Gọi Q là ảnh của P qua phép biến đổi T có ma trận biến đổi M là : P.M.
Phép biến đổi ngược T
-1
sẽ có ma trận biến đổi là M
-1
là ma trận nghịch đảo của ma trận M.
Nếu M = thì M

dc
ba -1
=
bcad −
1

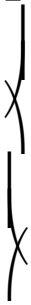
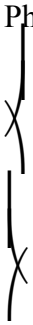

–
–
ac
bd

Ta có :
Phép tịnh tiến : M = thì M


10
01 -1
=


10
01
Phép quay : M = thì M

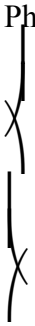


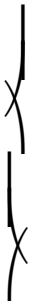
– θθ
θθ
cossin
sincos -1
=

–
θθ
θθ
cossin
sincos
Phép biến đổi tỉ lệ : M = thì M

y
x
S
S
0
0 -1
=



y
x
S
S
1
0
0
1

Phép biến dạng : M = thì M


1
1
h
g -1
=


–
–
1
1
h
g

3.8. Một số tính chất của phép biến đổi affine

- Bảo toàn đường thẳng : ảnh của đường thẳng qua phép biến đổi affine là đường thẳng.

Ví dụ : Để biến đổi một đoạn thẳng qua hai điểm A và B, chỉ cần thực hiện phép biến đổi cho A và B. Do vậy, để biến đổi một đa giác, chỉ cần thực hiện phép biến đổi đối với các đỉnh của đa giác.
- Bảo toàn tính song song : ảnh của hai đường thẳng song song là song song.
Ví dụ : ảnh của hình vuông, hình chữ nhật, hình bình hành, hình thoi sau phép biến đổi là hình bình hành.
- Bảo toàn tỉ lệ khoảng cách : Nếu điểm M chia đoạn AB theo tỉ số m thì ảnh của M là M' cũng chia đoạn AB theo tỉ số m.
Ví dụ : Trong hình vuông, các đường chéo cắt nhau tại trung điểm của mỗi đường nên các đường chéo của bất kỳ hình bình hành nào cũng cắy nhau tại trung điểm của mỗi đường.
Trong tam giác đều, giao điểm của 3 đường trung tuyến chia mỗi đường theo tỉ số 1:2. Do ảnh của tam giác đều qua phép biến đổi affine la một tam giác nên giao điểm của các đường trung tuyến trong một tam giác cũng sẽ chia chúng theo tỉ lệ 1:2.

3.9. Hệ tọa độ thuần nhất

Tọa độ thuần nhất của một điểm trên mặt phẳng được biểu diễn bằng bộ ba số tỉ lệ (xh , yh , h) không đồng thời bằng 0 và liên hệ với các tọa độ (x, y) của điểm đó bởi công thức :

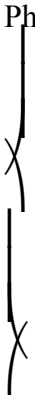
x =
h
xh
và y =
h
yh

Nếu một điểm có tọa độ thuần nhất là (x,y,z) thì nó cũng có tọa độ thuần nhất là (h.x, h.y, h.z) trong đó h là số thực khác 0 bất kỳ.
Một điểm P(x,y) sẽ được biểu diễn dưới dạng tọa độ thuần nhất là (x,y,1).
Trong hệ tọa độ thuần nhất các ma trận của phép biến đổi được biểu diễn như sau :

Phép tịnh tiến : M =

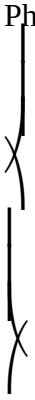
1
010
001
yx trtr

Phép biến đổi tỉ lệ : M =



100
00
00
Y
X
S
S

Phép quay : M =



–
100
0cossin
0sincos
θθ
θθ

Thuận lợi của hệ tọa độ thuần nhất là khi ta kết hợp hai hay nhiều phép biến đổi affine thi ma trận hợp của nhiều phép biến đổi được tính bằng cách nhân các ma trận của các phép biến đổi thành phần.

3.10. Kết hợp các phép biến đổi (composing transformation)

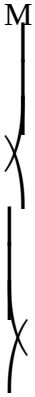
Quá trình áp dụng các phép biến đổi liên tiếp để tạo nên một phép biến đổi tổng thể được gọi là sự kết hợp các phép biến đổi.

• Kết hợp các phép tịnh tiến

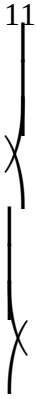
Nếu ta thực hiện phép tịnh tiến lên điểm P được điểm P', rồi lại thực hiện tiếp một phép tịnh tiến khác lên P' được điểm Q. Như vậy, điểm Q là ảnh của phép biến

đổi kết hợp hai phép tịnh tiến liên tiếp.
Q.x = P.x + (trx1+t rx2)
Q.y = P.y + (try1+try2)

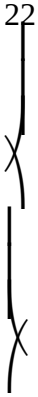
M = * =



1
010
001
11 yx trtr



1
010
001
22 yx trtr



+ + 1
01 0
00 1

2121 yyxx trtrtrtr

Vậy kết hợp hai phép tịnh tiến là một phép tịnh tiến. Từ đó, ta có kết hợp của nhiều phép tịnh tiến là một phép tịnh tiến.

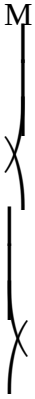
- Kết hợp các phép biến đổi tỉ lệ

Tương tự như phép tịnh tiến, ta có tọa độ điểm Q là điểm có được sau hai phép tịnh tiến M1(Sx1, Sy1), M2 (Sx2, Sy2)là :

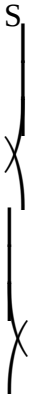
Q.x = P.x* Sx1* Sx2

Q.y = P.y* Sy1* Sy2

M = * =

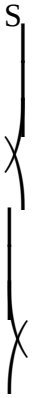


100
00
00
1
1
Y
X
S
S



100
00
00
2
2

Y
X
S
S



10 0
0* 0
00 *
21
21
YY
XX
SS
SS

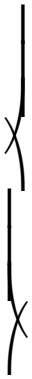
• Kết hợp các phép quay

Tương tự, ta có tọa độ điểm Q là điểm kết quả sau khi kết hợp hai phép quay quanh gốc tọa độ MR1(θ1) và MR2(θ2) là :

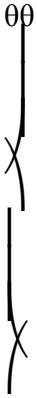
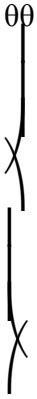
Q.x = P.x*cos(θ1+θ2) - P.y*sin (θ1+θ2)

Q.y = P.x*sin(θ1+θ2) + P.y*cos(θ1+θ2)

M = * =



–
100
01cos1sin
01sin1cos

θθ
θθ

–
100
02cos2sin
02sin2cos
θθ
θθ

+ +–
+ +
10 0
0)21cos()21sin(
0)21sin()21cos(
θθθθ
θθθθ

3.11. Tổng kết chương 3

Sinh viên cần nắm bắt được vấn đề cơ bản của phép biến đổi 2 chiều là phép biến đổi Affine biến đổi điểm P(P.x, P.y) thành điểm Q(Q.x, Q.y) là hàm tuyến tính có dạng :

(Q.x, Q.y) = (P.x, P.y). + (tr


dc
ba
x, try)
⇒ Q = P.M + tr
Từ công thức cơ bản này ta suy ra được các công thức biến đổi khác.
3.12. Bài tập chương 3
1. Vẽ một hình bình hành bằng cách sử dụng phép tịnh tiến. (Vẽ đoạn thẳng AB, sau đó tịnh tiến AB thành đoạn thẳng CD//AB, vẽ AD, Tịnh tiến AD thành BC (xem hình vẽ).
Trang 55 Chương 3: Phép biến đổi trong đồ họa hai chiều
A(xa,ya)
D(xa+k1,ya+k
2)
C(xb+k1,yb+k2)
B(xb,yb)

2. Viết chương trình vẽ một hình vuông ABCD (xem hình vẽ).
- Tịnh tiến hình vuông đó đến vị trí khác.
- Phóng to hình vuông ABCD.
- Biến dạng hình vuông thành hình thoi.

3. Vẽ một elip, sau đó vẽ thêm 3 elip khác có cùng tâm với elip đã cho, có độ dãn ở trục Ox là K và Oy là 1.
4. Vẽ một elip nghiêng một góc G độ có các trục không song song với các trục tọa độ.
5. Vẽ một bông hoa bằng cách vẽ các elip nghiêng một góc G độ với các màu khác nhau. Vẽ đến khi nào ấn phím bất kỳ thì ngưng.
6. Viết chương trình mô phỏng sự chuyển động của elip bằng cách cho elip này quay quanh tâm của nó.
7. Viết chương trình mô phỏng sự chuyển động của trái đất quay quanh mặt trời.
8. Viết chương trình vẽ một đường tròn tâm O bán kính R. Vẽ một đường kính AB. Quay đường kính này quanh tâm đường tròn.
Trang 56 Chương 3: Phép biến đổi trong đồ họa hai chiều
9. Viết chương trình vẽ đoạn thẳng AB.
Trang 57 Chương 4: Windowing và Clipping
Chương 4
TẠO CỬA SỔ VÀ CẮT HÌNH
(WINDOWING AND CLIPPING)

4.1. Tổng quan
• Mục tiêu
Học xong chương này, sinh viên cần phải nắm bắt được các vấn đề sau:
- Thế nào là window ?

- Hiểu rõ các thao tác loại bỏ phần hình ảnh nằm ngoài một vùng cho trước (thao tác này được gọi là xén hình).
- Thiết kế và cài đặt được các thuật toán xén hình.

• Kiến thức cơ bản cần thiết

Kiến thức tin học bao gồm kỹ thuật lập trình và cấu trúc dữ liệu

• Tài liệu tham khảo

Computer Graphics . Donald Hearn, M. Pauline Baker. Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986 (chapters 6, 123-153)

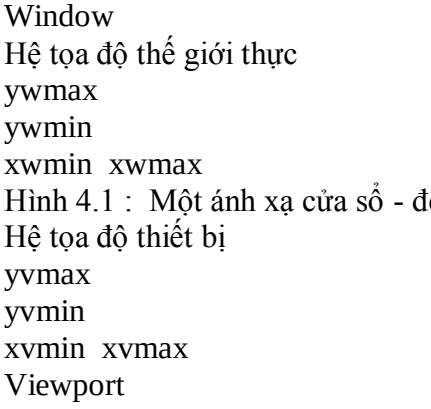
• Nội dung cốt lõi

- Trình bày các khái niệm về window.
- Các thuật toán clipping : Cohen-Sutherland, Liang-Barsky
- Phép biến đổi từ cửa sổ

4.2. Các khái niệm về Windowing

Hệ tọa độ Descartes là dễ thích ứng cho các chương trình ứng dụng để miêu tả các hình ảnh (picture) trên hệ tọa độ thế giới thực (world coordinate system). Các hình ảnh được định nghĩa trên hệ tọa độ thế giới thực này sau đó được hệ đồ họa vẽ lên các hệ tọa độ thiết bị (device coordinate). Điển hình, một vùng đồ họa cho phép người sử dụng xác định vùng nào của hình ảnh sẽ được hiển thị và bạn muốn đặt nó ở nơi nào trên hệ tọa độ thiết bị. Một vùng đơn lẻ hoặc vài vùng của hình ảnh có thể được chọn. Những vùng này có thể được đặt ở những vị trí tách biệt, hoặc một vùng có thể được chèn vào một vùng lớn hơn. Quá trình biến đổi này liên quan đến những thao tác như

tịnh tiến, biến đổi tỷ lệ vùng được chọn và xóa bỏ những phần bên ngoài vùng được chọn. Những thao tác này được gọi là windowing và clipping (xem hình 4.1).



Hình 4.1 : Một ánh xạ cửa sổ - đến – vùng quan sát

Một vùng có dạng hình chữ nhật được xác định trong hệ tọa độ thế giới thực được gọi là một cửa sổ (window). Còn vùng hình chữ nhật trên thiết bị hiển thị để cửa sổ đó ánh xạ đến được gọi là một vùng quan sát (viewport). Hình 4.1 minh họa việc ánh xạ một phần hình ảnh vào trong một viewport. Việc ánh xạ này gọi là một phép biến đổi hệ quan sát (viewing transformation), biến đổi cửa sổ (windowing tranformation), biến đổi chuẩn hóa (normalization transformation).

Các lệnh để xây dựng một cửa sổ và vùng quan sát từ một chương trình ứng dụng có thể được định nghĩa như sau:

set_window(xw_min, xw_max, yw_min, yw_max)

set_viewport(xv_min, xv_max, yv_min, yv_max)

Các tham số trong mỗi hàm được dùng để định nghĩa các giới hạn tọa độ của các vùng chữ nhật. Các giới hạn của cửa sổ được xác định trong hệ tọa độ thế giới thực. Hệ tọa độ thiết bị chuẩn thường được dùng nhất cho việc xác định vùng quan sát, dù rằng hệ tọa độ thiết bị có thể được dùng nếu chỉ có một thiết bị xuất (output device) duy nhất trong hệ thống. Khi hệ tọa độ thiết bị chuẩn được dùng, lập trình viên xem thiết bị xuất có giá trị tọa độ trong khoảng 0..1. Một sự xác định vùng quan sát được cho với các giá trị trong khoảng này. Các việc xác định sau đây, đặt một phần

của sự định nghĩa hệ tọa độ thế giới thực vào trong góc trên bên phải của vùng hiển thị, như được minh họa trong hình 4-2:

set_window(-60.5, 41.25, -20.75, 82.5);

set_viewport(0.5, 0.8, 0.7, 1.0);

Nếu một cửa sổ buộc phải được ánh xạ lấp đầy vùng hiển thị, sự xác định viewport được cho là:

Set_viewport(0,1, 0, 1)

Các vị trí được biểu diễn trên hệ tọa độ thiết bị chuẩn phải được biến đổi sang hệ tọa độ thiết bị trước khi được hiển thị bởi một thiết bị xuất cụ thể. Thông thường một thiết bị xác định được chứa trong các gói đồ họa cho mục đích này. Thuận lợi của việc dùng hệ tọa độ thiết bị chuẩn là để các gói đồ họa độc lập với thiết bị. Các thiết bị xuất khác nhau có thể được dùng nhờ việc cung cấp các trình điều khiển thiết bị thích hợp. Mọi điểm được tham khảo đến trong các gói đồ họa phải được xác định tương ứng trong hệ tọa độ Descartes. Bất kỳ sự định nghĩa hình ảnh nào dùng trong một hệ tọa độ khác, như hệ tọa độ cực, người sử dụng trước tiên phải biến đổi nó sang hệ tọa độ thế giới thực. Những hệ tọa độ Descart này sau đó được dùng trong các lệnh cửa sổ để xác định phần nào của hình ảnh muốn được hiển thị (xem hình 4.2).

(-60.5, 82.5)
(-60.5, -20.75)
Hệ tọa độ thế giới thực
(41.25, -20.75)
(-41.25, 82.5)
Window
0
yw
xw
Hình 4-2: Ánh xạ một cửa sổ vào một vùng quan sát trong hệ tọa độ thiết bị chuẩn
1
1 0.5
Viewport
Hệ tọa độ thiết bị chuẩn

Các lệnh về cửa sổ và vùng quan sát được phát biểu trước khi gọi các thủ tục vẽ ảnh. Các sự xác lập cho cửa sổ và vùng quan sát sẽ ảnh hưởng đến bất kỳ lệnh xuất theo sau nào cho đến khi có một sự xác lập mới.
Bằng việc thay đổi vị trí vùng quan sát, các đối tượng có thể được hiển thị ở bất kỳ vị trí nào trên thiết bị xuất. Cũng như vậy, bằng việc thay đổi kích thước vùng quan sát, kích thước các phần của đối tượng có thể bị thay đổi. Khi các cửa sổ được đặt lại các kích thước khác được ánh xạ thành công vào một vùng quan sát, các hiệu ứng về phóng to (zooming) có thể thực hiện được.

Hình 4-3: Hiển thị đồng thời hai biểu đồ, dùng đa cửa sổ và sự xác định vùng quan sát.

Khi các cửa sổ được làm nhỏ hơn, người dùng có thể phóng to vài nơi trên ảnh để xem chi tiết hơn mà không cần phóng to toàn bộ cửa sổ. Các hiệu ứng panning có thể được tạo ra bằng cách di chuyển một cửa sổ có kích thước xác định ngang qua một hình ảnh lớn.
Một ví dụ của việc dùng đa cửa sổ và các lệnh về vùng quan sát được cho trong các thủ tục sau đây. Hai biểu đồ được hiển thị trên hai phần đều nhau của một thiết bị hiển thị (xem hình 4-3).

```
type
points = array[1..max_points] of real;
procedure two_graphs;
var x,y : points; k: integer;
begin
set_window(0, 1, 0, 1); {vẽ đường chia ở trung tâm}
set_viewport(0, 1, 0, 1);
x[1]:=0.5; y[1]:=0; x[2]:=0.5; y[2]:=1;
polyline(2, x, y);
```


```
for k:=1 to 9 do begin  {đọc dữ liệu cho đồ thị thứ nhất}
{các giá trị dữ liệu từ 300 đến 700}   x[k]:=k;
readln(y[k]);
end; {for k}
set_window(1, 9, 300, 700);
set_viewport(0.1, 0.4, 0.2, 0.8);{đặt vào phần bên trái màn hình}
polyline(9, x, y);

for k:=1 to 13 do begin  {đọc dữ liệu cho đồ thị thứ hai}
x[k]:=k;
readln(y[k]);
end;
set_window(1, 13, 10, 100);     {các giá trị dữ liệu từ 10 đến 100}
set_viewport(0.6, 0.9, 0.2, 0.8);{đặt dữ liệu vào phần bên phải màn hình}
polyline(13, x, y);
end;{two graph}
```

Một phương pháp khác để xây dựng các vùng đa cửa sổ và vùng quan sát trong gói đồ họa là gán nhãn đến mỗi sự xác định. Điều này có thể được làm bằng việc thêm đối số thứ năm vào các lệnh về cửa sổ và vùng quan sát để xác định vùng chỉ định. Các tham số có thể là một chỉ số nguyên (0, 1, 2, 3, …). Các lệnh xuất sau đó dùng các chỉ số này để chỉ định sự chuyển đổi từ cửa sổ đến vùng quan sát nào. Cơ chế đánh số này cũng có thể được dùng để gắn kết một độ ưu tiên với mỗi vùng quan sát, đây là cơ sở để cài đặt tính chất nhìn thấy được của các cửa sổ nằm đè lên nhau. Các vùng quan sát được hiển thị theo độ ưu tiên được trình bày ở hình 4-4:

1 0
Hình 4-4: Hiển thị các vùng
quan sát theo thứ tự ưu tiên.
Các vùng quan sát có số thứ tự
nhỏ hơn sẽ có quyền ưu tiên cao
hơn.

Để cài đặt cách làm việc đa trạm (multiple workstation) , một tập bổ sung các lệnh về cửa sổ và vùng quan sát sẽ được định nghĩa. Các lệnh này có chứa số của trạm, giúp xây dựng các cửa sổ và vùng quan sát trên các trạm làm việc khác nhau. Điều này cho phép một người dùng hiển thị các phần khác nhau của ảnh kết quả lên các thiết bị xuất khác nhau. Ví dụ, một kiến trúc sư có thể hiển thị tổng thể bản vẽ của một căn nhà lên một màn hình, còn chi tiết tầng 2 sẽ được hiển thị lên màn hình thứ hai (xem hình 4.5)

Window
a
Hình 4-5
Quay cửa sổ, được xác
định bởi một góc a.
Các lệnh về cửa sổ và vùng quan sát vừa được giới thiệu được dùng cho các vùng hình chữ nhật, các đường biên của chúng song song với các trục tọa độ. Vài gói đồ họa cho phép người dùng chọn kiểu cửa sổ và vùng quan sát khác. Một cửa sổ bị quay, như hình 4-5, có thể được xác định với tham số là góc a trong một lệnh về cửa sổ. Một khả năng khác là chỉ định rõ một đa giác nào đó như một cửa sổ bằng việc cho một chuỗi các đỉnh. Chúng ta sẽ bắt đầu bằng việc trình bày các thuật toán cài đặt các cửa sổ và vùng quan sát hình chữ nhật, biên của chúng song song với trục x và y. Các cửa sổ có hình dạng đặc biệt khác sẽ được thảo luận sau đó như các thuật toán mở rộng (xem hình 4-6).

Thủ tục
Clipping
Ánh xạ vùng cửa
sổ vào vùng quan
sát trong hệ tọa độ
thiết bị chuẩn

Chuyển đổi
vùng vùng quan
sát sang hệ tọa
độ thiết bị
Input một hình ảnh trên
hệ tọa độ thế giới thực
nhờ một chương trình
ứng dụng
Hiển thị lên
thiết bị xuất
vật lý
Hình 4-6 Quá trình chuyển đổi các cửa sổ vào trong các vùng quan sát.

4.3. Các thuật toán Clipping

Ánh xạ một vùng cửa sổ vào trong một vùng quan sát, kết quả là chỉ hiển thị những phần trong phạm vi cửa sổ. Mọi thứ bên ngoài cửa sổ sẽ bị loại bỏ. Các thủ tục để loại bỏ các phần hình ảnh nằm bên ngoài biên cửa sổ được xem như các thuật toán clipping (clipping algorithms) hoặc đơn giản được gọi là clipping.

Việc cài đặt phép biến đổi cửa sổ thường được thực hiện bằng việc cắt (clipping) khỏi cửa sổ, sau đó ánh xạ phần bên trong cửa sổ vào một vùng quan sát (hình 6-6). Như một lựa chọn, một vài gói đồ họa đầu tiên ánh xạ sự định nghĩa trong hệ tọa độ thế giới thực vào trong hệ tọa độ thiết bị chuẩn và sau đó cắt khỏi biên vùng quan sát. Trong các các phần thảo luận sau, chúng ta giả thiết rằng việc cắt được thực hiện dựa vào đường biên cửa sổ trong hệ tọa độ thế giới thực. Sau khi cắt xong, các điểm bên trong cửa sổ mới được ánh xạ đến vùng quan sát.

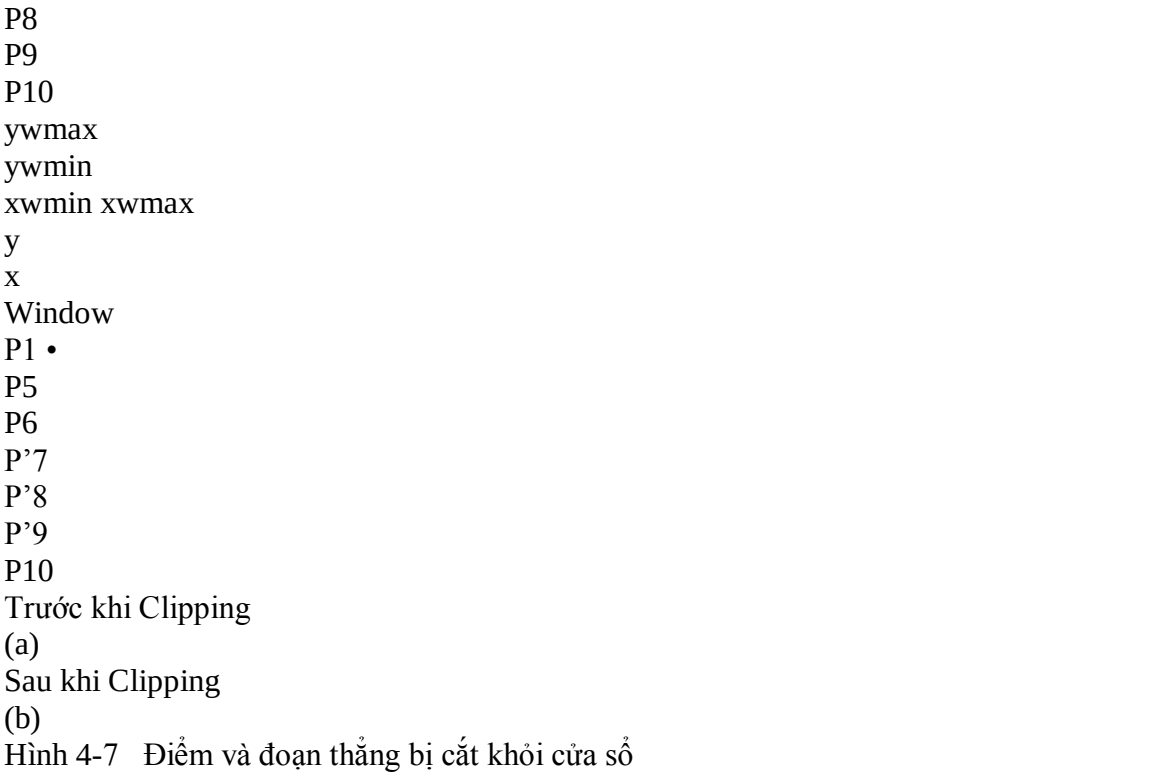
Việc cắt các điểm khỏi cửa sổ được hiểu đơn giản là chúng ta kiểm tra các giá trị tọa độ để xác định xem chúng có nằm bên trong biên không. Một điểm ở vị trí (x,y) được giữ lại để chuyển đổi sang vùng quan sát nếu nó thỏa các bất phương trình sau:

$$xwmin \leq x \leq xwmax,\ ywmin \leq y \leq ywmax \quad (4\text{-}1)$$

Nếu điểm nào không thỏa một trong bốn bất phương trình trên, nó bị cắt bỏ.

Trong hình 4-7, điểm P1 được giữ lại, trong khi điểm P2 bị cắt bỏ.

ywmax
ywmin
xwmin xwmax
y
x
Window
P2 •
P1 •
P4
P3
P5
P6
P7



Hình 4-7 Điểm và đoạn thẳng bị cắt khỏi cửa sổ

Hình 4-7 minh họa các quan hệ có thể có giữa các vị trí đoạn thẳng với biên cửa sổ. Chúng ta kiểm tra một đoạn thẳng xem có bị cắt hay không bằng việc xác định xem hai điểm đầu mút đoạn thẳng là nằm trong hay nằm ngoài cửa sổ. Một đoạn thẳng với cả hai đầu nằm trong cửa sổ thì được giữ lại hết, như đoạn từ P5 đến P6. Một đoạn với một đầu nằm ngoài (P9) và một đầu nằm trong (P10) sẽ bị cắt bớt tại giao điểm với biên

cửa sổ (P’9). Các đoạn thẳng có cả hai đầu đều nằm ngoài cửa sổ, có thể rơi vào hai trường hợp: toàn bộ đoạn thẳng đều nằm ngoài hoặc đoạn thẳng cắt hai cạnh cửa sổ.

Đoạn từ P3 đến P4 bị cắt bỏ hoàn toàn. Nhưng đoạn từ P7 đến P8 sẽ được giữ lại phần từ P’7 đến P’8.
Thuật toán clipping đường (line-clipping) xác định xem đoạn nào toàn bộ nằm trong, đoạn nào bị cắt bỏ hoàn toàn hay bị cắt một phần. Đối với các đoạn bị cắt bỏ một phần, các giao điểm với biên cửa sổ phải được tính. Vì một hình ảnh có thể chứa hàng ngàn đoạn thẳng, việc xử lý clipping nên được thực hiện sao cho có hiệu quả nhất. Trước khi đi tính các giao điểm, một thuật toán nên xác định rõ tất cả các đoạn thẳng được giữ lại hoàn toàn hoặc bị cắt bỏ hoàn toàn. Với những đoạn được xem xét là bị cắt bỏ, việc xác định các giao điểm cho phần được giữ lại nên được thực hiện với sự tính toán ít nhất.
Một tiếp cận để cắt các đoạn là dựa trên cơ chế đánh mã được phát triển bởi Cohen và Sutherland. Mọi điểm ở hai đầu mút đoạn thẳng trong hình ảnh sẽ được gán một mã nhị phân 4 bit, được gọi là mã vùng (region code), giúp nhận ra vùng tọa độ của một điểm. Các vùng này được xây dựng dựa trên sự xem xét với biên cửa sổ, như ở hình 6-8. Mỗi vị trí bit trong mã vùng được dùng để chỉ ra một trong bốn vị trí tọa độ tương ứng của điểm so với cửa sổ: bên trái (left), phải (right), trên đỉnh (top), dưới đáy (bottom). Việc đánh số theo vị trí bit trong mã vùng từ 1 đến 4 cho từ phải sang trái, các vùng tọa độ có thể liên quan với vị trí bit như sau:
Bit 1 – left
Bit 2 – right
Bit 3 – below
Bit 4 – above
Giá trị 1 ở bất kỳ vị trí nào chỉ ra rằng điểm ở vị trí tương ứng, ngược lại bit ở vị trí đó là 0. Nếu một điểm nằm trong cửa sổ, mã vị trí là 0000. Một điểm bên dưới và bên trái cửa sổ có mã vùng là 0101 (xem hình 4-8).

1001

1000 1010
Hình 4-8
Các mã vùng nhị phân cho các điểm đầu mút đoạn thẳng, được dùng để định nghĩa các vùng tọa độ liên hệ với một cửa sổ.

0001

0000
Window

0010

0101

0100

0110

Trang 65 Chương 4: Windowing và Clipping
Các giá trị bit trong mã vùng được xác định bằng cách so sánh giá trị tọa độ (x,y) của điểm đầu mút với biên cửa sổ. Bit 1 đặt lên 1 nếu x < xwmin. Các giá trị của ba bit còn lại được xác định bằng cách so sánh tương tự. Trong các ngôn ngữ lập trình, làm việc trên bit như thế này có thể thực hiện được, các giá trị bit mã vùng có thể được xác định theo các bước sau: (1) Tìm hiệu giữa tọa độ các điểm đầu mút với biên cửa sổ. (2) Dùng bit dấu (kết quả của mỗi hiệu) để đặt giá trị tương ứng trong mã vùng. Bit 1 là bit dấu của x - xwmin; bit 2 là bit dấu của xwmax – x; bit 2 là bit dấu của y - ywmin; và bit
4 là bit dấu của ywmax – y.
Khi chúng ta xây dựng xong các mã vùng cho tất cả các điểm đầu mút, chúng ta có thể xác định nhanh chóng đoạn thẳng nào là hoàn toàn nằm trong cửa sổ, đoạn nào là hoàn toàn nằm ngoài. Bất kỳ đoạn nào có mã vùng của cả 2 đầu mút là 0000 thì nằm trong cửa sổ và chúng ta chấp nhận các đường này. Bất kỳ đường nào mà trong hai mã vùng của hai đầu mút có một số 1 ở cùng vị trí bit thì đoạn hoàn toàn nằm ngoài cửa sổ, và chúng ta loại bỏ các đoạn này. Ví dụ, chúng ta vứt bỏ đoạn có mã vùng ở một đầu là 1001, còn đầu kia là 0101 (có cùng bit 1 ở vị trí 1 nên cả hai đầu mút của đoạn này nằm ở phía bên trái cửa sổ). Một phương pháp có thể được dùng để kiểm tra các đoạn cho việc cắt toàn bộ là thực hiện phép logic and với cả hai mã vùng. Nếu kết quả không phải là 0000 thì đoạn nằm bên ngoài cửa sổ (xem hình 4-9).

P3
P4
P’2
P’1
P1
P2
P’3
Window Hình 4-9
Các đoạn từ một điểm này đến một điểm khác có thể cắt cửa sổ hoặc giao điểm với các biên nằm ngoài cửa sổ.

Các đường không được nhận dạng là hoàn toàn nằm trong hay hoàn toàn nằm ngoài một cửa sổ thông qua các phép kiểm tra trên sẽ được tìm giao điểm với biên cửa sổ. Như được chỉ ra ở hình 4-9, các đường thuộc nhóm này có thể cắt hoặc không cắt cửa sổ. Chúng ta có thể xử lý các đoạn này bằng cách so sánh một điểm đầu mút (cái đang nằm ngoài cửa sổ) với một biên cửa sổ để xác định phần nào của đường sẽ bị bỏ. Sau đó, phần đường được giữ lại sẽ được kiểm tra với các biên khác, và chúng ta tiếp tục cho đến khi toàn bộ đường bị bỏ đi hay đến khi một phần đường được xác định là

nằm trong cửa sổ. Chúng ta xây dựng thuật toán để kiểm tra các điểm đầu mút tương tác với biên cửa sổ là ở bên trái, bên phải, bên dưới hay trên đỉnh.

Để minh họa các bước xác định trong việc cắt các đoạn khỏi biên cửa sổ dùng thuật toán của Cohen-Sutherland, chúng ta xem các đoạn trong hình 4-9 được xử lý như thế nào. Bắt đầu ở điểm đầu mút bên dưới từ P1 đến P2, ta kiểm tra P1 với biên trái, phải và đáy cửa sổ và thấy rằng điểm này nằm phía dưới cửa sổ. Ta tìm giao điểm P'1 với biên dưới. Sau khi tìm giao điểm P'1, chúng ta vứt bỏ đoạn từ P1 đến P'1. Tương tự, vì P2 bên ngoài cửa sổ, chúng ta kiểm tra và thấy rằng điểm này nằm phía trên cửa sổ. Giao điểm P'2 được tính, và đoạn từ P'1 đến P'2 được giữ lại. Kết thúc quá trình xử lý đoạn P1P2. Bây giờ xét đoạn kế tiếp, P3P4. Điểm P3 nằm bên trái cửa sổ, vì vậy ta xác định giao điểm P'3 và loại bỏ đoạn từ P'3 đến P3. Bằng cách kiểm tra mã vùng phần đoạn thẳng từ P'3 đến P4, chúng ta thấy rằng phần còn lại này nằm phía dưới cửa sổ và cũng bị vứt bỏ luôn.

Các giao điểm với biên cửa sổ có thể được tính bằng cách dùng các tham số của phương trình đường thẳng. Với một đường thẳng đi qua hai điểm (x1, y1) và (x2, y2), tung độ y của giao điểm với một biên dọc cửa sổ có thể tính được theo phép tính:

$$y = y1 + m (x - x1) \qquad (4\text{-}2)$$

Ở đây giá trị x được đặt là xwmin hoặc xwmax, và độ dốc m được tính bằng là

$$m = (y2 - y1)/ (x2 - x1)$$

Tương tự, nếu ta tìm giao điểm với biên ngang, hoành độ x có thể được tính như sau:

$$x = x1 + (y - y1)/m \qquad (4\text{-}3)$$

với y là ywmin hoặc ywmax.

Thủ tục sau đây minh họa thuật toán clipping đường (line-clipping) của Cohen-Sutherland. Các mã cho mỗi điểm đầu mút được chứa trong các mảng Boolean bốn phần tử.

```
var
xw_min, xw_max, yw_min, yw_max: real;
procedure clip_a_line (x1, y1, x2, y2: real);
type
```



```
boundaries = (left, right, bottom, top);
code = array [boundaries] of boolean;
var
```

```
code1, code2 : code;
done, display: boolean;
m: real;
procedure encode (x, y : real; var c: code);
begin
if x < xw_min then c[left]:= true
else c[left]:= false;
if x > xw_max then c[right]:= true
else c[right]:= false;
if y < yw_min then c[bottom]:= true
else c[bottom]:= false;
if y > yw_max then c[top]:= true
else c[top]:= false
end; {encode}

function accept (c1, c2 : code) : boolean;
var k : boundaries;
begin
{nếu điểm có trị "true" ở bất kỳ vị trí nào trong mã của nó,
một chấp nhận bình thường là không thể}
accept :=true;
for k:= left to top do
if c1[k] or c2[k] then accept :=false
end; {accept}
function reject (c1, c2 : code) : boolean;
var k : boundaries;
begin
{nếu hai điểm đầu mút có trị 'true' ở cùng vị trí tương ứng,
đoạn thẳng bị xóa bỏ}
```

```
reject:=false;
for  k:= left to top do
if c1[k] and c2[k] then reject :=true
end; {reject}

procedure swap_if_needed (var x1, y1, x2, y2: real;
var c1, c2: code);
begin
{đảm bảo rằng x1, y1 là điểm nằm ngoài cửa sổ và c1 chứa mã đó}
end; {swap_if_needed}

begin
done :=false;
display :=false;
while not done do begin
encode (x1, y1, code1);
```

```
encode (x2, y2, code2);
if accept (code1, code2) then begin
done :=true;
display :=true;
end {if accept}
else
if reject (code1, code2) then done :=true
else begin {tìm giao điểm}
{bảo đảm rằng  x1, y1 nằm ngoài cửa sổ}
swap_if_needed (x1, y1, x2, y2, code1, code2);
m := (y2-y1) / (x2-x1);
if code1[left] then begin
y1 := y1 + (xw_min – x1) * m;
x1 :=xw_min
end {cắt biên phải}
else
if code1[right] then begin
```

```
y1 := y1 + (xw_max – x1)*m;
x1 := xw_max
end {cắt biên trái}
else
if code1[bottom] then begin
x1 := x1 + (yw_min – y1) / m;
y1 := yw_min
end {cắt biên dưới đáy}
else
if code1[top] then begin
x1 := x1 + (yw_max – y1) / m;
y1 := yw_max
end {cắt biên đỉnh}
end {ngược lại tìm giao điểm}
end; {while not done}
if display then {draw x1, y1, to x2, y2}
end; {clip_a_line}
```
Một kỹ thuật để xác định giao điểm với biên cửa sổ mà không dùng đến phương trình đường thẳng là dùng thủ tục tìm kiếm nhị phân, được gọi là sự phân chia tại trung điểm. Đầu tiên, việc kiểm tra các đoạn một lần nữa được thực hiện bằng cách dùng mã vùng. Bất kỳ đoạn nào không được chấp nhận hoàn toàn hoặc không bị huỷ bỏ hoàn toàn (nhờ vào kiểm tra mã vùng) thì sẽ được đi tìm giao điểm bằng cách kiểm tra tọa độ trung điểm.

Tiếp cận này được minh họa trong hình 4-10 (xem hình 4-10). Mọi đoạn thẳng với hai điểm đầu mút (x1,y1) và (x2, y2), trung điểm được tính như sau:

$$xm = (x1 + x2) / 2; \quad ym = (y1 + y2) / 2 \quad (4\text{-}4)$$

Mỗi kết quả tính toán cho tọa độ giao điểm liên quan đến một phép cộng và một phép chia 2. Khi tọa độ giao điểm được xác định, mỗi nữa đoạn thẳng được kiểm tra

để chấp nhận hay huỷ bỏ toàn bộ. Nếu một nữa đoạn được chấp nhận hoặc bị huỷ bỏ, một nữa kia sau đó sẽ được xử lý theo cách tương tự. Điều này tiếp tục cho đến khi gặp một giao điểm. Nếu một nữa được chấp nhận hoặc bị huỷ bỏ toàn bộ, nữa kia tiếp

tục được xử lý cho đến khi toàn bộ nó là bị huỷ bỏ hoặc được giữ lại. Cài đặt phần cứng theo phương pháp này có thể giúp ta clipping khỏi biên vùng quan sát nhanh chóng sau khi các đối tượng vừa được chuyển sang hệ tọa độ thiết bị.

Pm
•
Pm •
Pm •
Pm •
Pm •
Pm
•
Window
Hình 4-10
Các trung điểm, Pm
được dùng trong
thuật toán clipping

Các kỹ thuật khác cho việc clipping đoạn dùng phương trình tham số của đường thẳng. Chúng ta có thể viết phương trình đường thẳng qua 2 điểm (x1, y1) và (x2, y2) theo hình thức tham số:

$x = x1 + (x2 – x1)u = x1 + \Delta x\, u$ (4-5)

$y = y1 + (y2 – y1)u = y1 + \Delta y\, u$

Với $\Delta x = x2 – x1$ và $\Delta y = y2 – y1$. Tham số u được gán các giá trị từ 0 đến 1, và các tọa độ (x,y) là tọa độ các điểm trên đường ứng với các giá trị cụ thể của u trong đoạn [0,1]. Khi u = 0, (x, y) = (x1, y1). Ở đầu kia của đoạn, u = 1 và (x, y) = (x2, y2).
Một thuật toán clipping đường hiệu quả dùng phương trình tham số đã được phát triển bởi Liang và Barsky. Họ ghi chú rằng nếu một điểm (x, y) dọc theo đường mà nằm trong cửa sổ được định nghĩa bởi các tọa độ (xwmin, ywmin) và (xwmax, ywmax),
thì các điều kiện sau đây phải được thỏa:

$xwmin \leq x1 + \Delta x\, u \leq xwmax$ (4-6)
$ywmin \leq y1 + \Delta y\, u \leq ywmax$
Bốn bất phương trình trên có thể được viết lại theo hình thức sau:
$pk\, u \leq qk,\ k = 1, 2, 3, 4$ (4-7)
ở đây p và q được định nghĩa như sau:
$p1 = -\Delta x,\quad q1 = x1 - xwmin$
$p2 = -\Delta x,\quad q2 = xwmax - x1$ (4-8)

$p3 = -\Delta y,\quad q3 = y1 - ywmin$
$p4 = \Delta y,\quad q4 = ywmax - y1$
Bất kỳ đoạn thẳng nào song song với một trong các biên cửa sổ sẽ có pk = 0, giá trị k phụ thuộc vào biên cửa sổ (k = 1, 2, 3, và 4 tương ứng với biên trái, phải, dưới, trên). Nếu với các giá trị đó của k, chúng ta có thể gặp $qk < 0$, khi đó đoạn thẳng sẽ hoàn toàn nằm ngoài biên và có thể bị loại bỏ khi xét sau này. Nếu $qk \geq 0$, đường thẳng tương ứng nằm trong biên.
Khi $pk < 0$, sự kéo dài không giới hạn của đoạn thẳng từ bên ngoài vào bên trong của biên cửa sổ kéo dài. Nếu $pk > 0$, đoạn thẳng tiến từ bên trong ra bên ngoài. Với pk khác 0, chúng ta có thể tính giá trị của u tương ứng với điểm mà tại đó đoạn thẳng kéo dài cắt biên k kéo dài của cửa sổ:
$u = qk/pk$ (4-9)
Đối với mỗi đoạn thẳng, chúng ta có thể tính các giá trị cho các tham số u1 và u2 để xác định phần nào của đoạn nằm bên trong cửa sổ. Giá trị của u1 được xác định bằng cách nhìn ở các cạnh của cửa sổ xem đoạn kéo dài nào từ ngoài vào trong ($p<0$). Đối với các cạnh cửa sổ, chúng ta tính $rk = qk/pk$. Giá trị của u1 là lớn nhất trong tập chứa 0 và các giá trị khác của r. Ngược lại, giá trị của u2 được xác định bằng cách kiểm tra các biên xem đoạn nào kéo dài nào từ bên trong ra bên ngoài ($p>0$). Một giá trị của rk được tính cho mỗi biên cửa sổ, và giá trị của u2 là nhỏ nhất trong tập chứa 1 và các giá trị đã được tính của r.
Nếu $u1 > u2$, đoạn hoàn toàn nằm ngoài cửa sổ và có thể bị vứt bỏ. Ngược lại, các điểm đầu mút của đoạn bị cắt được tính từ hai giá trị của tham số u.
Thuật toán này được trình bày trong thủ tục sau đây. Các tham số giao điểm của đoạn được khởi tạo các giá trị $u1 = 0$ và $u2 = 1$. Đối với mỗi biên cửa sổ, các giá trị thích hợp cho p và q được tính và được dùng bởi hàm cliptest để xác định xem đoạn nào có thể bị loại bỏ hoặc xem các tham số giao điểm sắp sửa bị thay đổi không. Khi $p < 0$, tham số r được dùng để cập nhật u1; khi $p>0$, tham số r được dùng để cập nhật u2. Nếu việc cập nhật u1 hoặc u2 đưa đến kết quả $u1 > u2$, chúng ta loại bỏ đoạn thẳng. Ngược lại, chúng ta cập nhật tham số u thích hợp chỉ nếu giá trị mới đưa đến kết quả làm ngắn đoạn thẳng. Khi $p=0$ và $q<0$, chúng ta vứt bỏ đoạn thẳng bởi vì nó song song và ở bên ngoài biên. Nếu đoạn thẳng vẫn chưa bị loại bỏ sau tất cả bốn giá trị của p và

q vừa được kiểm tra xong, các điểm đầu mút của đoạn bị cắt được xác định từ các giá trị của u1 và u2.

```
var
xwmin, xwmax, ywmin, ywmax : real;
procedure clipper (var x1, y1, x2, y2 : real);
var
```

```
u1, u2, dx, dy : real;
function cliptest (p, q : real; var u1, u2 : real);
var
r : real;
result : boolean;
begin
result := true;
if  p < 0 then begin  {đoạn từ bên ngoài vào bên trong biên }
r := q / p;
if  r > u2 then  result := false
{huỷ bỏ đoạn hoặc cập nhật u1 nếu thích hợp}
else if  r > u1 then u1 :=r
end {if p < 0}
else
if p > 0 then begin {đoạn từ bên trong ra bên ngoài của biên}
r := q / p;
if  r < u1 then  result := false
else if  r < u2  then u2 := r
end {if p > 0}
else
if  q < 0 then result := fasle;
cliptest := result
end; {cliptest}
begin {clipper}
u1 := 0;
u2 := 1;
```

```
dx := x2 – x1;
if  cliptest (-dx, x1 – xwmin, u1, u2) then
if  cliptest (dx, xwmax – x1, u1, u2) then begin
dy := y2 - y1;
if cliptest (-dy, y1 – ywmin, u1, u2) then
if cliptest(dy, ywmax – y1, u1, u2) then begin
{nếu u1 và u2 nằm  trong đoạn [0,1],
dùng để tính các điểm đầu mút mới}
if u2 < 1 then begin
x2 := x1 + u2 * dx;
y2 := y1 + u2 * dy
end; {if u2 < 1}
if u1 > 0 then begin
x1 := x1 + u1 * dx;
y1 := y1 + u1 * dy
end; {if u1 >  0}
end {if cliptest}
end {if cliptest}
end; {clipper}
```

Thuật toán clipping đường của Liang và Barsky giảm bớt các tính toán cần thiết để cắt các đoạn. Mỗi lần cập nhật u1 và u2 cần chỉ một phép chia, và các giao điểm với cửa sổ được tính chỉ một lần, khi mà các giá trị u1 và u2 vừa hoàn thành. Trái lại, thuật toán của Cohen và Sutherland lặp lại việc tính giao điểm của đoạn với các biên cửa sổ, và mỗi phép tính giao điểm cần cả hai phép chia và nhân (xem hình 4-11).

Window
Hình chữ nhật bao quanh
Hình 4-11
Cửa sổ bị quay được bao quanh bởi một biên chữ nhật lớn hơn (có các cạnh song song với hệ trục tọa độ)

Khi các cửa sổ bị quay hay các đa giác có hình dạng bất kỳ (được dùng làm cửa sổ và vùng quan sát), các thuật toán clipping đã được thảo luận sẽ cần vài sự thay đổi. Nó vẫn có thể được dùng để che chắn các đoạn thẳng. Một cửa sổ bị quay, hoặc một đa giác bất kỳ nào khác, có thể bị bao quanh trong một hình chữ nhật lớn hơn (hình chữ nhật này có các trục song song với các trục tọa độ) (hình 4 -11). Bất kỳ đoạn thẳng nào nằm bên ngoài hình chữ nhật bao quanh lớn hơn (bounding rectangle) thì cũng nằm bên ngoài cửa sổ (window). Các kiểm tra nằm trong cũng không dễ dàng, và các giao điểm phải được tính dùng phương trình đường thẳng của các biên cửa sổ và của các đoạn thẳng bị cắt.

Clipping một vùng (Area clipping)

Làm thế nào các đa giác được dùng trong các ứng dụng vẽ đường (line-drawing application) có thể bị cắt bằng cách xử lý các đoạn thẳng thành phần thông qua các thuật toán clipping đường đã được thảo luận. Một đa giác được xử lý theo cách này sẽ được thu giảm một loạt các đoạn sẽ bị cắt (xem hình 4-12).

Hình 4-12: Đa giác bị cắt bởi một thuật toán clipping đường.
Trước khi clipping Sau khi clipping

Khi một một biên đa giác định nghĩa một vùng tô, như ở hình 4-13. Một version thay đổi của thuật toán clipping đường được cần đến. Trong trường hợp này, một hoặc nhiều vùng kép kín phải được tạo ra để định nghĩa các biên cho vùng tô (xem hình 4-13).

Hình 4 –13: Một vùng có hình dạng, trước và sau khi clipping.

Trước khi clipping Sau khi clipping

Một kỹ thuật cho việc clipping đa giác, được phát triển bởi Sutherland và Hodgman, thực hiện việc clipping bằng cách so sánh một đa giác với lần lượt mỗi biên cửa sổ. Kết quả trả về của thuật toán là một tập các đỉnh định nghĩa vùng bị cắt (vùng này được tô với một màu hay một mẫu tô nào đó). Phương pháp căn bản được thể hiện trong hình 4-14.
Các vùng đa giác được định nghĩa bằng việc xác định một dãy có thứ tự các đỉnh. Để cắt một đa giác, chúng ta so sánh lần lượt mỗi đỉnh với biên một cửa sổ. Các đỉnh nằm bên trong cạnh cửa sổ này được giữ lại cho việc clipping với biên kế tiếp của cửa sổ (xem hình 4-15).

Hình
4-14
Clipping một
vùng đa giác
bằng cách dùng
các biên cửa sổ.

Cắt bên
trái
Cắt bên
phải
Cắt bên
dưới
Đa giác
gốc
Cắt bên
trên

Lưu P
(a)
•S
• P
Không điểm
nào được lưu
(c)
S•
P•
Hình 4-15
clipping.
Lưu I
(b)
•S
P•
• I
Lưu I, P
(d)
•P S• •
I

Quá trình xử lí các đỉnh của một dâ giác liên quan đến biên của cửa sổ. Từ đỉnh S, đỉnh kế tiếp được xét (P) có thể sinh ra một điểm, không điểm nào, hoặc hai điểm sẽ được lưu bởi thuật toán các đỉnh bên ngoài cạnh cửa sổ bị vứt bỏ. Nếu chúng ta

khởi hành từ một điểm bên trong cạnh cửa sổ đi đến một điểm bên ngoài, chúng ta lưu lại giao điểm của đoạn thẳng với biên cửa sổ. Cả hai giao điểm và đỉnh đa giác được lưu lại nếu chúng ta đi từ ngoài cạnh cửa sổ vào bên trong. Khả năng thứ tư có thể xảy ra khi chúng ta xử lí một điểm (P) và điểm trước đó (S) với biên cửa sổ được minh họa trong hình 4-15. Một điểm bên trong biên cửa sổ được lưu lại (trường hợp a), trong khi một điểm bên ngoài thì không (trường hợp c). Nếu một điểm P và điểm trước đó S nằm trên các phía đối diện nhau qua một biên (P ở trong, S ở ngoài và ngược lại), giao điểm I được tính và được lưu (trường hợp b và d). Trong trường hợp d, điểm P nằm trong và điểm trước đó S nằm ngoài, vì vậy cả hai giao điểm I và P được lưu. Khi tất cả các đỉnh vừa được xử lí với biên trái cửa sổ, tập các điểm được lưu sẽ tiếp tục bị cắt khi xem xét với biên kế tiếp của cửa sổ.

Hình 4-16
Clipping một đa giác khỏi cạ trái cửa sổ, bắt đầu với đỉnh 1. số có phẩy được dùng để đánh các điểm được lưu bởi thuậ clipping.

1
2
3
4
5
6
1’
2’
3’
4’
5’
Window
nh bên
Các
nhãn
t toán

Chúng ta minh họa phương pháp này bằng việc xử lí vùng trong hình 4-16 khi xem xét với biên bên trái của cửa sổ. Đỉnh 1 và 2 được xác định là nằm bên ngoài của biên. Đi qua đến đỉnh 3, đang nằm bên trong, chúng ta tính giao điểm và lưu lại cả hai giao điểm và đỉnh 3. Đỉnh 4 và 5 được xác định là nằm trong, và chúng nó cũng được lưu lại. Đỉnh thứ sáu và đỉnh cuối cùng thì nằm ngoài, vì vậy chúng ta tính và lưu giao điểm. Dùng năm điểm vừa được lưu, chúng ta lặp lại quá trình này khi xem xét với biên kế tiếp của cửa sổ.

Cài đặt các thuật toán vừa được mô tả đòi hỏi phải dùng không gian lưu trữ ngoài để lưu các điểm. Điều có thể tránh được nếu chúng ta quản lý được mỗi điểm (điểm sắp sửa được lưu và đi nhanh qua nó để kiểm tra tiếp), cùng với các lệnh (instructions) để cắt nó khỏi biên kế tiếp của cửa sổ. Chúng ta lưu một điểm (dù là một

đỉnh nguyên thuỷ của đa giác hay một đỉnh có được khi tính giao điểm) chỉ sau khi nó được xử lí khi xem xét với tất cả các biên. Như thể chúng ta có một đường ống chứa

một chuỗi các động tác clipping. Một điểm nằm bên trong hay nằm trên biên cửa sổ ở một giai đoạn sẽ được đi qua để đến giai đoạn kế tiếp.
Thủ tục sau đây thể hiện tiếp cận này . Một mảng s, lưu những điểm mới nhất vừa bị cắt cho với mỗi biên của cửa sổ. Quá trình chính đi qua mỗi đỉnh p đi vào quá trình clip_this để xem xét việc cắt với cạnh đầu tiên của cửa sổ . Nếu đoạn thẳng được định nghĩa bởi điểm đầu mút p và s[edge] cắt cạnh cửa sổ này, giao điểm được xác định và được đi qua để đến giai đoạn kế tiếp. Nếu p nằm bên trong cửa sổ, nó bị bỏ qua để đến giai đoạn clipping kế tiếp. Bất kì điểm nào còn được giữ lại sau khi xem xét với tất cả các cạnh của cửa sổ thì sau đó được gia nhập vào mảng kết quả kết xuất x_out và y_out. Mảng first_point lưu giữ cho mỗi cạnh cửa sổ điểm đầu tiên bị cắt bởi cạnh đó. Sau khi tất cả các đỉnh của đa giác vừa được xem xét xong, một quá trình kết thúc cắt các đoạn (đoạn đã được định nghĩa bởi các điểm đầu và cuối (các điểm bị cắt khỏi mỗi mỗi cạnh)).

```
type
point = array [1..max_points] of real;
procedure polygon_clip (n : integer; x, y : points; var m : integer;
var x_out, y_out : points);
const
boundary_count = 4;
type
vertex  =  array [1..2] of real;
boundary_range  = 1..boundary_count;
var
k :  integer;
p :  vertex;
s, first_point : array [1..boudary_count] of vertex;
new_edge  :  array [1..boundary_count] of boolean;

function inside (p : vertex; edge : boundary_range) : boolean;
begin
{trả về true nếu đỉnh p nằm trong cạnh edge cửa sổ}
end; { inside}
```



```
function cross (p, s : vertex;  edge : integer) : boolean;
begin
{trả về true nếu cạnh đa giác ps cắt biên cửa sổ}
end; {cross}

procedure output_vertex (p : vertex);
begin
m := m +1;
x_out[m] := p[1]; y_out[m] := p[2];
```

```
end; { output_vertex }

procedure find_intersection (p, s : vertex;
edge : boundary_range; var i;  vertex);
begin
{trả về trong tham số i giao điểm của ps với biên edge cửa sổ }
end; { intersection }

procedure clip_this (p : vertex; edge : boundary_range);
var i : vertex;
begin{ clip_this }
{lưu điểm đầu tiên cắt biên cửa sổ}
if new_edge[edge] then begin
first_point[edge] := p;
new_edge[edge] := false
end  {new_edge}
else
{nếu ps cắt biên cửa sổ, tìm giao điểm,
cắt giao điểm khỏi cạnh kế tiếp của cửa sổ}
if cross (p, s[edge], edge) then begin
find_intersection (p, s[edge], edge , i);
if edge < boundary_count then clip_this (i, edge +1)
else output_vertex (i)
Trang 79 Chương 4: Windowing và Clipping
end; {nếu ps cắt cạnh}
{cập nhật các đỉnh đã được lưu}
s[edge] := p;
{nếu p nằm bên trong cạnh cửa sổ này,
cắt nó khỏi cạnh kế tiếp của cửa sổ}
if inside (p, edge) then
if edge  < boundary_count  then clip_this (p, edge +1)
else output_vertex (p)
end; {clip_this}
procedure clip_closer;
{đóng quá trình. Đối với mỗi cạnh của cửa sổ,
cắt đường (đang nối với đỉnh được lưu sau cùng và điểm first_point
bị xử lý khỏi cạnh)}
var
i : vertex;
edge : integer;
begin
for edge := 1 to boundary_count do
if cross (s[edge], first_point[edge], edge) then begin
find_intersection (s[edge], first_point[edge], edge, i);
if edge < boundary_count then clip_this (i, edge +1)
else output_vertex (i)
```

```
end {nếu s và first_point cắt cạnh}
end; {clip_closer}

begin {polygon_clip}
m :=0;     {số các đỉnh kết xuất}
for k := 1 to boundary_count do
new_edge[k] := true;
for  k:= 1 to  n do begin   {đặt mỗi đỉnh vào đường ống (pipeline)}
p[1] := x[k]; p[2] := y[k];
clip_this (p, 1)    {cắt khỏi cạnh đầu tiên của cửa sổ}
end; {for k}
```
Trang 80 Chương 4: Windowing và Clipping
```
clip_closer     {đóng đa giác}
end; { polygon_clip }
```

Khi một đa giác lõm bị cắt bởi một cửa sổ hình chữ nhật, vùng bị cắt sau cùng có thể hình thành hai đa giác riêng biệt thật sự. Vì thuật toán cắt vùng này chỉ tạo ra một danh sách các đỉnh, các vùng riêng biệt này sẽ được nối lại bằng các đoạn thẳng nối. Một ví dụ của hiệu ứng này được thể hiện trong hình 4-17. Sự xem xét đặt biệt có thể được thực hiện đối với trường hợp như thế để gỡ bỏ các đoạn nối dư thừa, hoặc các thuật toán clipping tổng quát hơn sẽ được phát triển (xem hình 4-17).

Hình 4-17: Clipping đa giác lõm trong hình (a) bởi một cửa sổ tạo ra hai vùng nối nhau trong hình (b)

Window
(a) (b)

Dù chúng ta đã và đang giới hạn việc thảo luận của chúng ta đối với các cửa sổ chữ nhật có các cạnh song song với trục x và trục y., chúng ta có thể cài đặt thuật toán này với cửa sổ có hình đa giác bất kì. Chúng ta có thể cần lưu trữ thông tin về mỗi biên cửa sổ, và chúng ta có thể cần thay đổi thủ tục inside và find_intersection để quản lý thuộc tính của các biên tuỳ ý.
Một tiếp cận khác để clipping các vùng đa giác là dùng các phương pháp phương trình tham số. Các cửa sổ hình dạng tuỳ ý sau đó có thể được xử lí bằng cách

dùng phương trình tham số của đường thẳng để mô tả cả hai: biên cửa sổ và các biên của vùng bị cắt.
Các vùng bị clipping hình dạng khác đa giác cần thực hiện nhiều công việc hơn một chút, vì biên của các vùng này không được định nghĩa bằng các phương trình

đường thẳng. Ví dụ, trong hình 4-18, phương trình đường tròn được cần để tìm hai giao điểm trên biên cửa sổ.
Hình 4-18: Clipping một vùng có hình dạng tròn.

Clipping văn bản (Text Clipping)
Có vài kỹ thuật có thể được dùng để clipping văn bản trong gói đồ họa. Việc chọn lựa phương pháp cụ thể để cài đặt phụ thuộc vào các phương pháp đã được dùng để sinh ra các kí tự và mức độ tinh vi được đòi hỏi bởi người dùng trong việc xử lí văn bản (xem hình 4-19).
Trước khi clipping
Trước khi clipping Sau khi clipping
Hình 4-19
Clipping văn bản dùng các biên chữ nhật. Bất kỳ hình chữ nhật nào mà nằm đè lên biên cửa sổ đều bị vứt bỏ hoàn toàn.

Phương pháp đơn giản nhất để xử lí các chuỗi kí tự có liên quan đến một biên cửa sổ là dùng chiến lượt "clipping tất cả văn bản hoặc không clipping gì cả" (all-or-none text-clipping), được trình bày trong hình 6-19. Nếu tất cả chuỗi kí tự nằm bên trong

một cửa sổ, chúng ta giữ lại nó. Ngược lại, chuỗi vứt bỏ. Thủ tục này có thể được cài đặt bằng việc xem xét một hình chữ nhật bao quanh mẫu văn bản. Các vị trí biên của hình chữ nhật sau đó được so sánh với các biên cửa sổ, và chuỗi bị huỷ bỏ nếu có bất kì sự nằm đè nào. Phương pháp này cho ta clipping nhanh nhất.
Một sự chọn lựa để loại bỏ toàn bộ chuỗi kí tự nếu nó nằm đè lên biên một cửa sổ là dùng chiến lược "clipping kí tự toàn bộ hoặc không" (all-or-none character-clipping). Ở đây chúng ta vứt bỏ chỉ những kí tự nào không hoàn toàn nằm trong cửa

sổ (xem hình 4-20). Trong trường hợp này, các giới hạn biên của các kí tự đơn lẻ được so sánh với cửa sổ. Bất kì kí tự nào hoặc nằm đè lên hoặc nằm bên ngoài biên cửa sổ đều bị cắt bỏ.

Hình 4-20
Các chuỗi kí tự có thể hoàn toàn bị cắt để mà chỉ những kí tự hoàn nằm bên trong cửa sổ mới được giữ lại.
Trước khi clipping Sau khi clipping

Phương pháp sau cùng cho việc quản lí việc cắt văn bản là cắt các kí tự riêng lẻ.
Bây giờ chúng ta xem các kí tự cũng tương tự như các đoạn thẳng. Nếu một kí tự riêng lẻ nằm đè lên biên cửa sổ, chúng ta cắt bỏ phần nằm ngoài cửa sổ (xem hình 4-21). Các kí tự được hình thành với các đoạn thẳng có thể được xử lí theo cách này, bằng cách dùng thuật toán clipping đường. Việc xử lí các kí tự được hình thành bởi các bản đồ bit cần clipping những pixel đơn lẻ bằng cách so sánh các vị trí liên hệ của các mẫu lưới (patern grid) với các biên cửa sổ.

Hình 4-21
Clipping các kí tự

đơn lẻ.
Trước khi clipping Sau khi clipping

Tẩy xoá (banking)
Thay vì lưu giữ lại thông tin trong một vùng được định nghĩa,, một vùng cửa sổ có thể được dùng để xóa bỏ bất kì thứ gì bên trong biên của nó. Những gì nằm bên ngoài được giữ lại.
Việc xoá bỏ tất cả các màu kết xuất trong một vùng chỉ định có ý nghĩa thuận lợi cho việc nạp chồng các hình ảnh khác. Các kỹ thuật này thường được dùng để thiết kế các trang trình bày (layout) trong quảng cáo hoặc trong các ứng dụng xuất bản (publishing) hoặc cho việc thêm các nhãn hoặc mẫu thiết kế đến một hình ảnh. Kỹ

thuật cũng được dùng để nối kết các biểu đồ, bản đồ, hoặc giản đồ. Hình 4-22 minh họa vài ứng dụng của tẩy xóa.
Khi hai hiển thị che phủ lên nhau dùng đến các phương pháp tẩy xóa, một cái có thể được nghĩ đến như cận cảnh (ảnh ở gần-foreground) và những cái còn lại được xem như ảnh nền (background). Một cửa sổ xóa, cái đang bao quanh vùng hiển thị cận ảnh, được đặt lên trên ảnh nền, và các phần hình ảnh nằm trong vùng cửa sổ bị xóa sạch. Hai hiển thị được nối kết lại, với các thông tin của cận ảnh được đặt vào vùng cửa sổ bị xóa.

(a)
(b)
Hình 4-22
Các ví dụ về tẩy xóa: (a
vùng được cung cấp để
nhãn; (b) Một vùng
dùng để xóa một phần c
hiển thị trước đó để tạ
một vùng trống cho n
chồng ảnh mới lên.

) Một
dán
được
ủa
o ra
ạp

4.4. Phép biến đổi từ cửa sổ - đến – vùng quan sát
Khi tất cả các điểm, đoạn thẳng, và văn bản vừa bị cắt, chúng được ánh xạ lên vùng vùng quan sát để hiển thị. Phép biến đổi đến vùng quan sát này được thực hiện để các vị trí tọa độ liên hệ được giữ lại.
Trong hình 4-23, một điểm ở vị trí (xw, yw) trong một cửa sổ được ánh xạ và trong vị trí (xv, yv) trong vùng quan sát. Để duy trì sự sắp đặt liên hệ tương tự trong vùng quan sát như trong cửa sổ, chúng ta cần:

min max
min
min max
min
xvxv
xvxv
xwxw
xwxw
–
–
=
–
–
(4-10)

và
min max
min
min max
min
yvyv
yvyv
ywyw
ywyw
–
–
=
–
–
(4-11)

Ta viết lại phương trình (4-10) và (4-11) như các phép tính biến đổi rõ ràng cho các tọa độ xv và yv:
min min

min max
min max
) (xvxwxw
xwxw
xvxv
xv +−
−
−
= (4-12)

min min
min max
min max
) (yvywyw
ywyw
yvyv
yv +−
−
−
=

Các phép tính biến đổi từ cửa sổ - đến - vùng quan sát có thể được viết chặt chẽ hơn như sau:

xv = sx(xw - xwmin) + xvmin (4-13)
yv = sy(yw - ywmin) + yvmin

Phép biến đổi này bao gồm cả hai phép biến đổi tỉ lệ và tịnh tiến. Các hệ số tỉ lệ sx và sy phụ thuộc vào kích thước liên hệ của cửa sổ và vùng quan sát. Các hệ số tỉ lệ này phải bằng nhau nếu các đối tượng muốn được bảo tồn sự cân đối (đồng dạng) khi chúng được ánh xạ đến vùng quan sát. Khi cửa sổ và vùng quan sát có kích thước bằng nhau (sx = sy = 1), không có sự thay đổi nào về kích thước của các đối tượng được biến đổi. Giá trị của xvmin và yvmin cho biết các hệ số tịnh tiến để di chuyển các đối tượng vào vùng quan sát.

Các chuỗi kí tự có thể được quản lí theo hai cách khi chúng được ánh xạ đến vùng quan sát. Việc ánh xạ đơn giản nhất bảo tồn kích thước kí tự, thậm chí khi vùng quan sát được mở rộng hay thu nhỏ lại so với cửa sổ. Phương pháp này có thể được dùng đến khi văn bản được tạo ra với các font chuẩn – không thể bị thay đổi. Trong các hệ thống khi mà có sự cho phép thay đổi kích thước kí tự chuẩn, sự định nghĩa chuỗi có thể được đặt trong cửa sổ tương tự như các từ gốc. Đối với các kí tự được hình thành bởi các đoạn thẳng, việc ánh xạ đến vùng quan sát có thể được thực hiện như một dãy tuần tự các phép biến đổi đường (xem hình 4-23).

x
Hình 4-23: Một điểm ở vị trí (xw, yw) trong cửa sổ được ánh xạ đến điểm (xv, yv) trong vùng quan sát. Việc ánh xạ được thực hiện sao cho tỷ lệ tương quan trong hai vùng tương tự nhau.
yvmax
yvmin
xvmin xvmax

y
(xv, yv)
•
x
ywmax
ywmin
xwmin xwmax
y
(xw, yw)
•

4.5. Tổng kết chương 4

- Cần nắm vững khái niệm Window, cách mã vùng theo giải thuật Cohen-Sutherland. Phân biệt điểm thuộc và không thuộc window.
- Lưu ý cách sử dụng phương trình tham số của đường thẳng trong giải thuật Liang-Barsky.
- Có thể hiệu chỉnh các thuật toán xén đoạn thẳng để xén đa giác bằng cách xem đa giác như là một tập các đoạn thẳng liên tiếp nối với nhau. Tuy nhiên, kết quả xén được là tập các đoạn thẳng rời nhau.
- Lưu ý điều chúng ta mong muốn là kết quả sau khi xén một đa giác phải là một hoặc các đa giác để có thể chuyển thành các vùng tô.

4.6. Bài tập chương 4

1. Viết chương trình tạo cửa sổ hình chữ nhật có tọa độ các điểm dưới bên trái và điểm trên bên phải lần lượt là (Xmin, Ymin) và (Xmax, Ymax).
2. Tiếp tục bài 1, hãy xét một điểm P(x,y) có nằm bên trong cửa sổ không ? Biết

rằng nếu P(x,y) nằm bên trong cửa sổ thì P sẽ thoả hệ bất phương trình sau :

$Xmin \leq x \leq Xmax$
$Ymin \leq y \leq Ymax$

3. Tiếp tục bài tập 2, xét bài toán xén đoạn thẳng được cho bởi các điểm P1(x1, y1), P2(x2, y2) bất kỳ.
4. Tiếp tục bài tập 3, sử dụng thuật toán Cohen - Sutherland (phân chia mã vùng) xét bài toán xén các đoạn thẳng được cho bởi các điểm P1(x1, y1), P2(x2, y2), P3(x3, y3), P4(x4, y4), P5(x5, y5), P6(x6, y6), P7(x7, y7), và P8(x8, y8) vào cửa sổ chữ nhật trên xem hình vẽ (a) và (b)).
5. Thảo luận kỹ nhân tố căn bản đằng sau các kiểm tra và phương khác nhau để tính các tham số giao nhau u1 và u2 trong thuật toán clipping đường Liang-Barsky.
Trang 86 Chương 4: Windowing và Clipping
6. So sánh số lượng các phép tính toán học được thực hiện trong các thuật toán clipping đường Cohen-Sutherland và Liang-Barsky đối với vài hướng đoạn thẳng khác nhau liên quan đến cửa sổ clipping.
7. Cài đặt thuật toán thuật toán clipping đường Liang-Barsky lên hệ thống của bạn.
8. Hãy nghĩ ra một thuật toán để thực hiện việc clipping đường bằng cách dùng phương pháp phân chia điểm ở giữa. Sự cài đặt phần mềm của thuật toán này có thuận lợi hơn hai thuật toán clipping đường đã được thảo luận trong chương không?
9. Cài đặt một thuật toán cắt các đoạn thẳng bằng cách dùng một cửa sổ bị quay, được định nghĩa bởi các giá trị tọa độ nhỏ nhất và lớn nhất và bị quay một góc như trong hình 6-5.
10. Thay đổi thuật toán clipping đa giác để cắt các vùng đa giác lõm một cách hợp lý. (Một phương pháp để thực hiện điều này là chia đa giác lõm ra làm các đa giác lồi.)
11. Sửa lại cho hợp lí thuật toán clipping đường Liang-Barsky để clipping đa giác.
12. Viết thủ tục để cắt một ellipse bằng cách dùng cửa sổ chữ nhật.
13. Giả sử rằng các kí tự được định nghĩa trong một lưới điểm (pixel grid), hãy phát triển một thuật toán clipping văn bản để cắt các kí tự đơn lẻ theo chiến lược "tất cả - hoặc - không".
14. Hãy phát triển một thuật toán clipping văn bản để cắt các kí tự đơn lẻ, giả sử rằng các kí tự được định nghĩa trong một lưới điểm (pixel grid).
15. Viết một thủ tục thực hiện xóa một phần bất kì của hình ảnh đã được định nghĩa, dùng kích thước cửa sổ xóa được xác định bất kỳ.
16. Viết các thủ tục để cài đặt các lệnh của cửa sổ và vùng quan sát. Tức là, các thủ tục có chứa tham số về hệ tọa độ trong các lệnh để thực hiện biến đổi sang vùng quan sát cho các cảnh cụ thể: clipping trong hệ tọa độ thế giới thực, chuyển đổi sang hệ tọa độ chuẩn hóa, sau cùng biến đổi đến hệ tọa độ thiết bị.
Trang 87 Chương 5: Đồ họa ba chiều
Chương 5 : ĐỒ HỌA BA CHIỀU

5.1. Tổng quan

• Mục tiêu

Học xong chương này sinh viên cần phải nắm bắt được các vấn đề sau:

- Thế nào là đồ họa 3 chiều ?
- Viết được chương trình vẽ một hình trong không gian 3 chiều

• Kiến thức cơ bản

Hình giải tích và hình học không gian : tích vô hướng của hai véc tơ. Ma trận cùng các phép toán

• Tài liệu tham khảo

Computer Graphics . Donald Hearn, M. Pauline Baker. Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986 (chapters 9, 181-233)

• Nội dung cốt lõi

- Trình bày cách biểu diễn đối tương 3 chiều: biểu diễn các đối tương cơ bản qua mô hỉnh khung nối kết.
- Các phép biến đổi trong không gian 3 chiều.

5.2. Giới thiệu đồ họa 3 chiều

Các đối tượng trong thế giới thực phần lớn là các đối tượng 3 chiều còn thiết bị hiển thị chỉ 2 chiều. Do vậy, muốn có hình ảnh 3 chiều ta cần phải giả lập.

Chiến lược cơ bản là chuyển đổi từng bước. Hình ảnh sẽ được hình thành từ từ, ngày càng chi tiết hơn.

Qui trình hiển thị ảnh 3 chiều như sau

• Biến đổi từ hệ tọa độ đối tượng sang hệ tọa độ thế giới thực (Modelling transformation).

Mỗi đối tượng được mô tả trong một hệ tọa độ riêng được gọi là Hệ tọa độ đối tượng.

Có 2 cách mô hình hóa đối tượng:

- Solid modeling : mô tả các vật thể (kể cả bên trong).
- Boudary representation : chỉ quan tâm đến bề mặt đối tượng.

Các đối tượng có thể được biểu diễn bằng mô hình Wire-Frame.

Nhận thấy rằng khi biểu diễn đối tượng, ta có thể chọn gốc tọa độ và đơn vị đo lường sao cho việc biểu diễn là thuận lợi nhất. Thường thì người ta chuẩn hóa kích thước của đối tượng khi biểu diễn.

Boudary representation cho phép xử lý nhanh còn silid modeling cho hình ảnh đầy đủ và xác thực hơn.

• Loại bỏ các đối tượng không nhìn thấy được (Trivial Rejection).

Loại bỏ các đối tượng hoàn toàn không thể nhìn thấy trong cảnh.

Thao tác này giúp ta lược bỏ bớt các đối tượng không cần thiết do đó giảm chi phí xử lý.

• Chiếu sáng các đối tượng (Illumination).

Gán cho các đối tượng màu sắc dựa trên các đặc tính của các chất tạo nên chúng và các nguồn sáng tồn tại trong cảnh.

Có nhiều mô hình chiếu sáng và tạo bóng : constant-intensity, Interpolate,...

• Chuyển từ word space sang eye space (Viewing Transformation).

Thực hiện một phép biến đổi hệ tọa độ để đặt vị trí quan sát (viewing position) về gốc tọa độ và mặt phẳng quan sát (viewing plane) về một vị trí mong ước.

Hình ảnh hiển thị phụ thuộc vào vị trí quan sát và góc nhìn.
Hệ qui chiếu có gốc đặt tại vị trí quan sát và phù hợp với hướng nhìn sẽ thuận lợi cho các xử lý thật.

• Loại bỏ phần nằm ngoài viewing frusturn (Clipping).

Thực hiện việc xén đối tượng trong cảnh để cảnh nằm gọn trong một phần không gian hình chóp cụt giới hạn vùng quan sát mà ta gọi là viewing frustum.
Viewung frustum có trục trùng với tia nhìn, kích thước giới hạn bởi vùng ta muốn quan sát.

• Chiếu từ eye space xuống screen space (Projection).

Thực hiện việc chiếu cảnh 3 chiều từ không gian quan sát xuống không gian màn hình.
Có 2 phương pháp chiếu:
- Chiếu song song
- Chiếu phối cảnh

Trang 89 Chương 5: Đồ họa ba chiều

Khi chiếu ta phải tiến hành việc khử mặt khuất để có thể nhận được hình ảnh trung thực.
Khử mặt khuất cho phép xác định vị trí (x,y) trên màn hình thuộc về đối tượng nào trong cảnh.

• Chuyển đối tượng sang dạng pixel (Rasterization).
• Hiển thị đối tượng (Display).

5.3. Biểu diễn đối tượng 3 chiều

Trong đồ họa máy tính, các đối tượng lập thể có thể được mô tả bằng các bề mặt (surface) của chúng. Ví dụ : một hình lập phương được xây dựng từ sáu mặt phẳng, một hình trụ được xây dựng từ sự kết hợp của một mặt cong và hai mặt phẳng và hình cầu được xây dựng từ chỉ một mặt cong.
Thông thường để biểu diễn một đối tượng bất kỳ, người ta dùng phương pháp xấp xỉ để đưa các mặt về dạng các mặt đa giác (polygon faces).

• Điểm trong không gian 3 chiều có tọa độ (x,y,z) mô tả một vị trí trong không gian.

```
typedef  struct  {
int     x;
int  y;
int  z;
}  Point _3D ;
```

• Vectơ : xác định bởi 3 tọa độ dx, dy, dz mô tả một hướng và độ dài của véc tơ.

Véc tơ không có vị trí trong không gian.

$$| V | = \sqrt{dx^2 + dy^2 + dz^2}$$

Tích vô hướng của hai véc tơ

$$V1 * V2 = dx1dx2 + dy1dy2 + dz1dz2$$

Hay $V1 * V2 = |V1||V2| \cos \theta$

```
typedef    struct   {
```

```
int  dx;
int  dy;
int dz;
}    Vector ;
```

• Đoạn thẳng trong không gian 3 chiều: biểu diễn tổ hợp tuyến tính của 2 điểm
Để biểu diễn dạng tham số của đoạn thẳng, ta có :
$P = P1 + t*(P2 - P1) , \quad (0 \leq t \leq 1)$
```
typedef    struct   {
Point  P1;
Point  P2;
}   Segment ;
```

• Tia (Ray) : là một đoạn thẳng với một đầu nằm ở vô cực.
Biểu diễn dạng tham số của tia :
$P = P1 + t*V \quad , \quad (0 \leq t < \infty)$
```
typedef   struct   {
Point P1;
Vector V;
}   Ray;
```

• Đường thẳng (Line): là một đoạn thẳng với cả hai đầu nằm ở vô cực
Biểu diễn dạng tham số của đường thẳng
$P = P1 + t*V \quad , \quad (\infty \leq t < \infty)$
```
typedef   struct   {
Point P1;
Vector V;
}   Line;
```

• Đa giác (Polygon) : là một vùng giới hạn bởi hạn dãy các điểm đồng phẳng .
(Các điểm được cho theo thứ tự ngược chiều kim đồng hồ)
```
typedef    struct   {
```

```
Point *Points;
int  nPoints;
}   Polygon;
```
Có thể biểu diễn một mặt đa giác bằng một tập hợp các đỉnh và các thuộc tính kèm theo. Khi thông tin của mỗi mặt đa giác được nhập, dữ liệu sẽ được điền vào các bảng (mãng dữ liệu) sẽ được dùng cho các xử lý tiếp theo, hiển thị và biến đổi. Các bảng dữ liệu mô tả mặt đa giác có thể tổ chức thành hai nhóm : bảng hình học và bảng thuộc tính. Các bảng lưu trữ dữ liệu hình học chứa tọa độ các đỉnh và các tham số cho biết về định hướng trong không gian của mặt đa giác. Thông tin về thuộc tính của các đối tượng chứa các tham số mô tả độ trong suốt, tính phản xạ và

các thuộc tính kết cấu của đối tượng. Một cách tổ chức thuận tiện để lưu trữ các dữ liệu hình học là tạo ra 3 danh sách : một bảng lưu đỉnh, một bảng lưu cạnh và một bảng lưu đa giác. Trong đó:
- Các giá trị tọa độ cho mỗi đỉnh trong đối tượng được chứa trong bảng lưu đỉnh.
- Bảng cạnh chứa các con trỏ trỏ đến bảng đỉnh cho biết đỉnh nào được nối với một cạnh của đa giác .
- Cuối cùng là bảng lưu đa giác chứa các con trỏ trỏ đến bảng lưu cạnh cho biết những cạnh nào tạo nên đa giác.

• Mặt phẳng (Plane) :

```
typedef   struct  {
Vector  N;
int  d;
}   Plane;
```

Phương trình biểu diễn mặt phẳng có dạng : Ax + By + Cz + D = 0 (5-1)
Trong đó (x,y,z) là một điểm bất kỳ của mặt phẳng và A, B, C, D là các hằng số diễn tả thông tin không gian của mặt phẳng.
Để xác định phương trình mặt phẳng, ta chỉ cần xác định 3 điểm không thẳng hàng của mặt phẳng này. Như vậy, để xác định phương trình mặt phẳng qua một đa giác, ta sẽ sử dụng tọa độ của 3 đỉnh đầu tiên (x1,y1), (x2,y2), (x3,y3) trong đa giác này.
Từ phương trình (5-1) ta có :
Trang 92 Chương 5: Đồ họa ba chiều
Axk + Byk + Czk + D = 0 , k= 0,1,2,3. (5-2)
Trong đó :
A =
33
23
11
1
1
1
zy
zy
zy
B =
3 3
2 2
1 1
1
1
1
zx
zx
zx

C =
1
1
1
33
32
11
yx
yx
yx
C =
333
232
111
zyx
zyx
zyx

Khai triển các định thức trên ta có :
A = y1(z2 - z3) + y2(z3 - z1) + y3(z1 - z2)
B = z1(x2 - x3) + z2(x3 - x1) + z3(x1 - x2)
C = x1(y2 - y3) + x2(y3 - y1) + x3(y1 - y2)
A = - x1(y2z3 - y3z2) - x2(y3z1 - y1z3) - x3(y1z2 - y2z1)
Hướng của mặt phẳng thường được xác định thông qua véc tơ pháp tuyến của nó. Véc tơ pháp tuyến n = (A,B,C) (xem hình 5-1)

n=(A,B,C)

.

Hình 5.1 : Vec tơ pháp tuyến của mặt phẳng.

- Mô hình khung nối kết (Wireframe-Model)

Một phương pháp thông dụng và đơn giản để mô hình hóa đối tượng là mô hình khung nối kết. Một mô hình khung nối kết gồm có một tập các đỉnh và tập các cạnh nối các đỉnh đó. Khi thể hiện bằng mô hình này, các đối tượng 3 chiều có vẻ rỗng và không giống thực tế lắm. Tuy nhiên, vẽ bằng mô hình này thì nhanh nên người ta

thường dùng nó trong việc xem phác thảo các đối tượng. Để hoàn thiện hơn, người ta dùng các kỹ thuật tạo bóng và loại bỏ các đường khuất, mặt khuất.
Với mô hình khung nối kết, hình dạng của đối tượng 3 chiều được biểu diễn

bằng hai danh sách (list) : danh sách các đỉnh (vertices) và danh sách các cạnh (edges) nối các đỉnh đó. Danh sách các đỉnh cho biết thông tin hình học (đó là vị trí các đỉnh), còn danh sách các cạnh xác định thông tin về sự kết nối (cho biết cặp các đỉnh tạo ra cạnh). Chúng ta hãy quan sát một vật thể ba chiều (xem hình 5-2) được biểu diễn bằng mô hình khung nối kết như sau:

z
10
1
x
y
54
9
6
8
3
2
7
1
1 1

Hình 5.2 :
Vật thể 3 chiều
được biểu diễn
bằng khung nối
kết.

Bảng danh sách các cạnh và đỉnh biểu diễn vật thể

Vertex List
Vertex x y z
1 0 0 0 back side
2 0 1 0
3 0 1 1
4 0 0.5 1.5
5 0 0 1
6 1 0 0 front side
7 1 1 0
8 1 1 1
9 1 0.5 1.5
10 1 0 1

Edge List
Edge Vertex1 Vertex2
1 1 2
2 2 3
3 3 4
4 4 5
5 5 1
6 6 7
7 7 8
8 8 9
9 9 10
10 10 6
11 1 6
12 2 7
13 3 8
14 4 9
15 5 10
16 2 5
17 1 3

Người ta có thể vẽ các đối tương theo mô hình khung nối kết bằng cách sử dụng các phép chiếu song song hay phép chiếu phối cảnh sẽ được giới thiệu ở chương 6.

5.4. Các phép biến đổi 3 chiều

5.4.1. Hệ tọa độ bàn tay phải - bàn tay trái

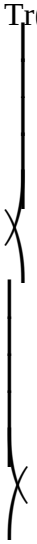
- Hệ tọa độ theo qui ước bàn tay phải : để bàn tay phải sao cho ngón cái hướng theo trục z, khi nắm tay lại, các tay chuyển động theo hướng từ trục x đến trục y.
- Hệ tọa tọa độ theo qui ước bàn tay trái : để bàn tay phải sao cho ngón cái hướng theo trục z, khi nắm tay lại, các ngón tay chuyển động theo hướng từ trục x đến trục y.
- Hệ tọa độ thuần nhất (Homogeneous Coordinates) : Mỗi điểm (x,y,z) trong không gian Descartes được biểu diễn bởi một bộ bốn tọa độ trong không gian 4 chiều thu gọn (hx,hy,hz,h). Người ta thường chọn h=1.

• Các phép biến đổi tuyến tính là tổ hợp của các phép biến đổi sau : tỉ lệ, quay, biến dạng và đối xứng. Các phép biến đổi tuyến tính có các tính chất sau :
- Gốc tọa độ là điểm bất động
- Ảnh của đường thẳng là đường thẳng
- Ảnh của các đường thẳng song song là các đường thẳng song song
- Bảo toàn tỉ lệ khoảng cách
- Tổ hợp các phép biến đổi có tính phân phối

5.4.2. Các phép biến đổi Affine cơ sở

• Phép tịnh tiến

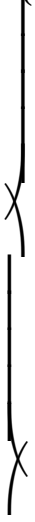
Tr(trx, try, trz) =



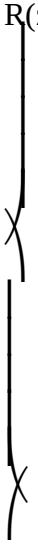
1
0100
0010
0001
zyx trtrtr

• Phép biến đổi tỉ lệ

S((sx, sy, sz) =



1000

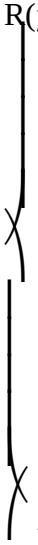
000
000
000
z
y
x
s
x
s
Khi Sx = Sy = Sz ta có phép biến đổi đồng dạng.
Trang 95 Chương 5: Đồ họa ba chiều
• Phép quay quanh trục Z
R(z,θ) =


–
1000
0100
00cossin
00sincos
θθ
θθ
• Phép quay quanh trục X
R(x,θ) =


–
1000
0cossin0
0sincos0
0001
θθ
θθ

• Phép quay quanh trục Y

R(y,θ) =



–
1000
0cos0sin
0010
0sin0cos
θ θ
θ θ

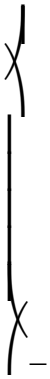
• Cách xác định chiều dương trong các phép quay

Định nghĩa về chiều quay được dùng chung cho cả hệ tọa độ theo qui ước bàn tay phải và bàn tay trái. Cụ thể chiều dương được định nghĩa như sau :

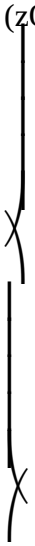
- Quay quanh truc x : từ trục dương y đến trục dương x
- Quay quanh trục y : từ trục dương z đến trục dương x
- Quay quanh trục x : từ trục dương x đến trục dương y

• Phép đối xứng qua mặt phẳng tọa độ

(yOx) : Mr(x) =



1000
0100
0010
0001
(zOx) : Mr(y) =



1000
0100
0010
0001

(xOy) Mr(x) =

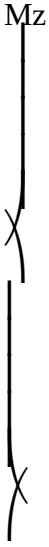
–
1000
0100
0010
0001

• Phép đối xứng qua trục x, y và z

Mx =

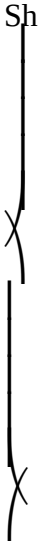
–
–
1000
0100
0010
0001

My =

–
–
1000
0100
0010
0001
Mz =


–
–
1000
0100
0010
0001

- Phép biến dạng

Sh =


1000
01
01
0 1
yzxz
zy xy
zxyx
hh
hh
hh
5.5. Tổng kết chương 5
- Trong đồ họa máy tính, các đối tượng được mô tả bằng bề mặt của chúng. Khi đó, người ta dùng phương pháp xấp xỉ để đưa các bề mặt về dạng các mặt đa giác.
- Lưu ý khi sử dụng phương pháp mô hình khung nối kết, bao gồm một tập các đỉnh và một tập các cạnh nối các đỉnh đó. Phương pháp này thì nhanh nhưng có khuyết điểm là không giống thực tế. Để cải thiện, cần dùng các kỹ thuật tạo bóng và khử các mặt khuất, đường khuất.
Trang 97 Chương 6: Quan sát ảnh ba chiều
Chương 6 : QUAN SÁT ẢNH BA CHIỀU

6.1. Tổng quan
• Mục tiêu
Học xong chương này sinh viên cần phải nắm bắt được các vấn đề sau:
- Cơ chế của phép chiếu
- Các thao tác liên quan đến phép biến đổi cách quan sát.
- Kỹ thuật quan sát ảnh 3 chiều
• Kiến thức cơ bản
Kiến thức toán học : các khái niệm cơ bản về vị trí tương đối của đường thẳng và mặt phẳng trong hình học không gian.
• Tài liệu tham khảo
Computer Graphics . Donald Hearn, M. Pauline Baker. Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986 (chapters 12, 235-257)
• Nội dung cốt lõi
- Khái niệm phép chiếu
- Phép chiếu song song
- Phép chiếu phối cảnh
- Biến đổi hệ tọa độ quan sát
- Lập trình xem ảnh 3 chiều
6.2. Các phép chiếu
Trong đồ họa hai chiều, các thao tác quan sát biến đổi các điểm hai chiều trong mặt phẳng tọa độ thế giới thực thành các điểm hai chiều trong mặt phẳng hệ tọa độ thiết bị. Sự định nghĩa đối tượng, bị cắt bởi một cửa sổ, được ánh xạ vào một vùng quan sát. Các hệ tọa độ thiết bị chuẩn hóa này sau đó được biến đổi sang các hệ tọa độ thiết bị, và đối tượng được hiển thị lên thiết bị kết xuất. Đối với đồ họa ba chiều, việc làm này phức tạp hơn một chút, vì bây giờ có vài chọn lựa để có thể quan sát ảnh như thế nào. Chúng ta có thể quan sát ảnh từ phía trước, từ phía trên, hoặc từ phía sau. Hoặc chúng ta có thể tạo

ra quang cảnh về những gì chúng ta có thể thấy nếu chúng ta đang đứng ở trung tâm của

một nhóm các đối tượng. Ngoài ra, sự mô tả các đối tượng ba chiều phải được chiếu lên bề mặt quan sát của thiết bị xuất. Trong chương này, trước hết chúng ta sẽ thảo luận các cơ chế của phép chiếu. Sau đó, các thao tác liên quan đến phép biến đổi cách quan sát, và đầy đủ các kỹ thuật quan sát ảnh ba chiều sẽ được phát triển.

Có hai phương pháp cơ bản để chiếu các đối tượng ba chiều lên bề mặt quan sát hai chiều. Tất cả các điểm của đối tượng có thể được chiếu lên bề mặt theo các đường thẳng song song, hoặc các điểm có thể được chiếu theo các đường hội tụ về một điểm được gọi là tâm chiếu (the center of projection). Hai phương pháp này được gọi là phép chiếu song song (parallel projection) và phép chiếu phối cảnh (perspective projection) (xem hình 6-1). Trong cả hai trường hợp, giao điểm của đường chiếu với bề mặt quan sát xác định các tọa điểm của điểm được chiếu lên mặt phẳng chiếu này. Chúng ta giả sử rằng mặt phẳng chiếu là mặt z = 0 của hệ tọa độ bàn tay trái (left-handed coordinate system) (xem hình 6-2).

(a)
Phép chiếu song song
P2
P1
P’2
P’1
Mặt phẳng
chiếu
•
•
•
•
(b)
Phép chiếu phối cảnh
P2
P1
P’2
P’1
Mặt phẳng
chiếu
•
• •
•
•

Tâm chiếu
Hình 6-1 Hai phương pháp chiếu một đoạn thẳng lên bề mặt của mặt phẳng chiếu

Bề mặt
quan sát
y
z
Hình 6-2
Một bề mặt quan sát được
định nghĩa trong mặt z=0 của
hệ tọa độ bàn tay trái.
x

Phép chiếu song song bảo tồn mối quan hệ về chiều của các đối tượng, và đây là kỹ thuật được dùng trong việc phác thảo để tạo ra các bức vẽ tỷ lệ của các đối tượng ba chiều. Phương pháp này được dùng để thu các hình ảnh chính xác ở các phía khác nhau của một đối tượng. Tuy nhiên, phép chiếu song song không cho một hình ảnh thực tế của các đối tượng ba chiều. Ngược lại, phép chiếu phối cảnh tạo ra các hình ảnh thực nhưng không bảo tồn các chiều liên hệ. Các đường ở xa được chiếu sẽ nhỏ hơn các đường ở gần mặt phẳng chiếu, như trong hình 6-3 (xem hình 6-3).

Hình 6-3
Hai đoạn thẳng dài bằng nhau, trong
phép chiếu phối cảnh, đoạn nào ở xa mặt
phẳng chiếu hơn sẽ có kích thước nhỏ
Mặt phẳng
chiếu
Tâm chiếu

6.2.1. Các phép chiếu song song

Các hình ảnh được hình thành bằng phép chiếu song song có thể được xác định dựa vào góc hợp bởi hướng của phép chiếu hợp với mặt phẳng chiếu. Khi hướng của phép chiếu vuông góc với mặt phẳng, ta có phép chiếu trực giao (hay phép chiếu vuông góc - orthographic projection). Một phép chiếu có thể không vuông góc với mặt phẳng chiếu được gọi là phép chiếu xiên (oblique projection).

Các phép chiếu trực giao hầu như được dùng để tạo ra quang cảnh nhìn từ phía trước, bên sườn, và trên đỉnh của đối tượng (xem hình 6-4). Quang cảnh phía trước, bên sườn, và phía sau của đối tượng được gọi là "mặt chiếu" (elevation), và quang cảnh phía trên được gọi là "mặt phẳng" (plane). Các bản vẽ trong kỹ thuật thường dùng các phép chiếu trực giao này, vì các chiều dài và góc miêu tả chính xác và có thể đo được từ bản vẽ.

Quang cảnh phía trước
(Front View)
Quang cảnh bên sườn
(SideView)
Quang cảnh trên đỉnh
(Top View)
Hình 6-4
Ba phép chiếu trực giao
của một đối tượng.

Chúng ta cũng có thể xây dựng các phép chiếu trực giao để có thể quan sát nhiều hơn một mặt của một đối tượng. Các quang cảnh như thế được gọi là các phép chiếu trực giao trục lượng học (axonometric orthographic projection). Hầu hết phép chiếu trục lượng học được dùng là phép chiếu cùng kích thước (isometric projection). Một phép chiếu cùng kích thước được thực hiện bằng việc sắp xếp song song mặt phẳng chiếu mà nó cắt mỗi trục tọa độ ở nơi đối tượng được định nghĩa (được gọi là các trục chính) ở các khoảng cách như nhau từ ảnh gốc. Hình 6-5 trình bày phép chiếu cùng kích thước. Có tám vị trí, một trong tám mặt, đều có kích thước bằng nhau. Tất cả ba trục chính được vẽ thu gọn bằng nhau trong phép chiếu cùng kích thước để kích thước liên hệ của các đối tượng được bảo tồn. Đây không là trường hợp phép chiếu trực giao trục lượng học tổng quát, khi mà các hệ số tỷ lệ theo ba trục chính có thể khác nhau.

Các phương trình biến đổi để thực hiện một phép chiếu song song trực giao thì dễ hiểu. Đối với điểm bất kỳ (x, y, z), điểm chiếu (xp, yp, xp) trên bề mặt chiếu được tính như
sau:

$$xp = x, \quad yp = y, \quad zp = 0 \qquad (6\text{-}1)$$

z
x
y
Mặt phẳng chiếu
(Projection plane)
Hình 6-5 Phép chiếu cùng kích thước
của một đối tượng lên bề mặt quan sát

Một phép chiếu xiên đạt được bằng việc chiếu các điểm theo các đường thẳng song song, các đường thẳng này không vuông góc với mặt phẳng chiếu. Hình 6-6 trình bày hình chiếu xiên của điểm (x, y, z) theo một đường thẳng chiếu đến vị trí (xp, yp). Các tọa độ chiếu trực giao trên mặt phẳng chiếu là (x, y). Đường thẳng của phép chiếu xiên tạo một góc α với đường thẳng trên mặt phẳng chiếu (đây là đường nối điểm (xp, yp) với điểm (x, y)). Đường này, có chiều dài L, hợp một góc φ với phương ngang trên mặt phẳng chiếu. Chúng ta có thể diễn tả các tọa độ chiếu qua các số hạng x, y, L, và φ:

$$xp = x + L \cos\varphi \quad (6\text{-}2)$$
$$yp = y + L \sin\varphi$$

Hình 6-6 Phép chiếu vuông góc của
điểm (x, y, z) thành điểm (xp, yp) lên
mặt phẳng chiếu
x
Mặt phẳng chiếu
z
(x,y)
y
α
φ
(x, y, z)
•
(xp, yp)
L

Phương chiếu có thể định nghĩa bằng việc chọn các giá trị cho góc α và φ. Các chọn lựa thông thường cho góc φ là 30o

và 45o
, là các góc hiển thị một quang cảnh của
mặt trước, bên sườn, và trên đỉnh (hoặc mặt trước, bên sườn, và dưới đáy) của một đối

tượng. Chiều dài L là một hàm của tọa độ z, và chúng ta có thể tính tham số này từ các
thành phần liên quan.
tan α =
L
z
=
1
1
L
(6-3)

ở đây L1 là chiều dài của các đường chiếu từ (x, y) đến (xp, yp) khi z = 1.
Từ phương trình 6-3, chúng ta có
L = z L1 (6-4)
và các phương trình của phép chiếu xiên 6-2 có thể được viết lại như sau
xp = x + z(L1 cosφ) (6-5)
yp = y + z(L1 sinφ)

Ma trận biến đổi để tạo ra bất kỳ việc chiếu song song có thể được viết như sau

Pparallel = (6-6)

1000
00sin cos
0010
0001

1 1 φφ LL

Một phép chiếu trực giao có thể đạt được khi L1 = 0 (xảy ra ở góc chiếu α=90o
).
Các phép chiếu xiên được sinh ra với giá trị L1 khác không. Ma trận chiếu 6-6 có cấu trúc
tương tự ma trận của phép làm biến dạng theo trục z. Thực tế, kết quả của ma trận chiếu này là làm biến dạng mặt phẳng của hằng z và chiếu chúng lên mặt phẳng quan sát. Các giá trị tọa độ x và y trong mỗi mặt của hằng z bị thay đổi bởi một hệ số tỷ lệ đến giá trị z của mặt phẳng để các góc, các khoảng cách, và các đường song song trong mặt phẳng được chiếu chính xác. Hiệu quả này được thể hiện trong hình 6-7, ở đây mặt sau của hình hộp bị biến dạng và bị nằm đè bởi mặt trước trong phép chiếu đến bề mặt quan sát. Một cạnh của hình hộp, cái nối mặt trước với mặt sau, được chiếu thành đoạn chiều dài L1, cái
hợp thành một góc φ với đường ngang trong mặt phẳng chiếu.

z
y
L1
Hình 6-7
Phép chiếu xiên
của một hình
hộp lên bề mặt
quan sát tại mặt Chương 6: Quan sát ảnh ba chiều

Hai góc được dùng phổ biến trong phép chiếu xiên là các góc có tgφ =1 và tgφ=2.
Trường hợp đầu, φ = 45o
và quang cảnh đạt được được gọi là phép chiếu cavalier. Tất cả
các đường vuông góc v ới mặt phẳng chiếu được chiếu với chiều dài không thay đổi. Các ví dụ của phép chiếu cavalier đối với một hình lập phương được cho trong hình 6-8.

Khi góc chiếu được chọn để tgφ = 2, kết quả quang cảnh được gọi là phép chiếu cabinet. Góc phép chiếu này xấp xỉ 63.4o
làm cho các đường chiếu vuông góc với bề mặt
chiếu được chiếu ở một nữa chiều dài của chúng.
Các phép chiếu cabinet cho hình ảnh thực hơn phép chiếu cavalier vì sự thu giảm chiều dài của các đường song song. Hình 6-9 trình bày phép chiếu cabinet cho hình lập phương.

(a)
φ=45o
(b)
φ=30o
Hình 6-8
Phép chiếu cavalier của
một hình lập phương lên
bề mặt chiếu với hai giá
trị góc φ. Độ sâu của
phép chiếu bằng với
chiều rộng và chiều cao.

(a)

φ=45o
Hình 6-9
Phép chiếu cabinet của một hình lập phương lên bề mặt chiếu với hai giá trị góc φ. Độ sâu của phép chiếu bằng 1/2 chiều rộng và chiều cao.
(b)
φ=30o

6.2.2. Các phép chiếu phối cảnh

Để đạt được phép chiếu phối cảnh của đối tượng ba chiều, chúng ta chiếu các điểm theo đường thẳng chiếu để các đường này gặp nhau ở tâm chiếu. Trong hình 6-10, tâm chiếu trên trục z và có giá trị âm, cách một khoảng d phía sau mặt phẳng chiếu. Bất kỳ điểm nào cũng có thể được chọn làm tâm của phép chiếu, tuy nhiên việc chọn một điểm dọc theo trục z sẽ làm đơn giản việc tính toán trong các phương trình biến đổi.

Hình 6-10
Phép chiếu phối cảnh của điểm P ở tọa độ (x, y, z) thành điểm (xp, yp, 0) trên mặt phẳng chiếu.
x
Mặt phẳng chiếu
z
(xp,yp)
y
•
•
•
P(x,y,z)
Tâm chiếu
d

Chúng ta có thể đạt được các phương trình biến đổi cho phép chiếu phối cảnh từ các phương trình tham số mô tả các đường chiếu từ điểm P đến tâm chiếu (xem hình 6-10). Các tham số xây dựng các đường chiếu này là

$$x' = x - xu$$
$$y' = y - yu \qquad (6\text{-}7)$$
$$z' = z - (z + d)u$$

Tham số u lấy giá trị từ 0 đến 1, và các tọa độ (x', y', z') thể hiện cho bất kỳ điểm nào dọc theo đường thẳng chiếu. Khi u = 0, phương trình 12-7 làm cho điểm P ở tọa độ (x, y, z). Ở đầu mút kia của đường thẳng u =1, và chúng ta có các tọa độ của tâm chiếu,

(0, 0, d). Để thu được các tọa độ trên mặt phẳng chiếu, chúng ta đặt z' = 0 và tìm ra tham số u:

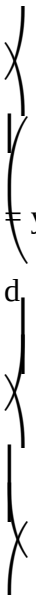
$$u = \frac{z}{z + dz} \qquad (6\text{-}8)$$

Giá trị của tham số u tạo ra giao điểm của đường chiếu với mặt phẳng chiếu tại (xp, yp, 0). Thế phương trình 6-8 vào phương trình 6-7, ta thu được các phương trình biến đổi của phép chiếu phối cảnh.

$$xp = x - \left(\frac{z}{dz + 1}\right)$$

$$= x + \left(\frac{z}{dz + 1}\right)$$

$$yp = y$$


= y
+ dz
d

+1
1
dz
(6-9)

zp = 0

Dùng biểu diễn hệ tọa độ thuần nhất ba chiều (three-dimentional homogeneous coordinate representation), chúng ta có thể viết phép biến đổi phối cảnh theo hình thức ma trận:

Trong biểu diễn này,
[xh yh xh w] = [x y z 1] (6-10)

1000
/1000
0 010

0001
d

và các tọa độ chiếu trên mặt phẳng chiếu được tính từ các tọa độ thuần nhất như sau
[xp yp zp 1] = [xh/w yh/w zh/w 1] (6-12)
Khi các đối tượng ba chiều được chiếu lên một mặt phẳng dùng các phương trình biến đổi phối cảnh, bất kỳ tập hợp các đường thẳng song song nào của đối tượng mà không song song với mặt phẳng chiếu được chiếu thành các đường hội tụ (đồng quy). Các đường thẳng song song với mặt phẳng khi chiếu sẽ tạo ra các đường song song. Điểm mà tại đó tập hợp các đường thẳng song song được chiếu xuất hiện hội tụ về đó được gọi là điểm ảo (vanishing point). Mỗi tập hợp các đường thẳng song song được chiếu như thế sẽ có một điểm ảo riêng (xem hình 6.11).
w =
z
+ 1 (6-11)
d

Hình 6-11
Các quang cảnh phối cảnh của một hình lập phương.
(a)

Mô tả tọa độ
(c)
Phép phối cảnh
Hai -điểm
•
• Điểm ảo
trục x
Điểm ảo
trục z
(b)
Phép phối cảnh
Một - điểm
Điểm ảo
(Vanishing
•
x
y
z
Điểm ảo cho bất kỳ tập các đường thẳng, tức các đường song song với một trong các trục tọa độ thế giới thực được nói đến như một điểm ảo chính (principal vanishing point). Chúng ta quản lý số lượng các điểm ảo chính (một, hai, hoặc ba) với hướng của mặt phẳng chiếu, và các phép chiếu phối cảnh được phân loại dựa vào đó để có các phép chiếu: một-điểm (one-point), hai-điểm (two-point), hoặc ba-điểm (three-point). Số lượng các điểm ảo chính trong một phép chiếu được xác định bởi số lượng các trục của hệ tọa độ thế giới thực cắt mặt phẳng chiếu. Hình 6-11 minh họa hình ảnh của các phép chiếu phối cảnh một-điểm và hai-điểm của hình lập phương. Trong hình 6-11(b), mặt phẳng chiếu có phương song song với mặt xy để chỉ có trục z bị cắt. Phương này tạo ra phép chiếu phối cảnh một-điểm với một điểm ảo trên trục z. Với quang cảnh trong hình 6-11(c), mặt phẳng chiếu cắt cả hai trục x và z nhưng không cắt trục y. Kết quả, phép chiếu phối cảnh hai-điểm này chứa cả hai điểm ảo: trên trục x và trên trục z.

6.3. Biến đổi hệ tọa độ quan sát (hệ quan sát)

Việc tạo ra quang cảnh của một đối tượng trong không gian ba chiều thì tương tự như việc chụp ảnh. Chúng ta có thể đi vòng quanh và chụp các bức ảnh từ bất kỳ góc

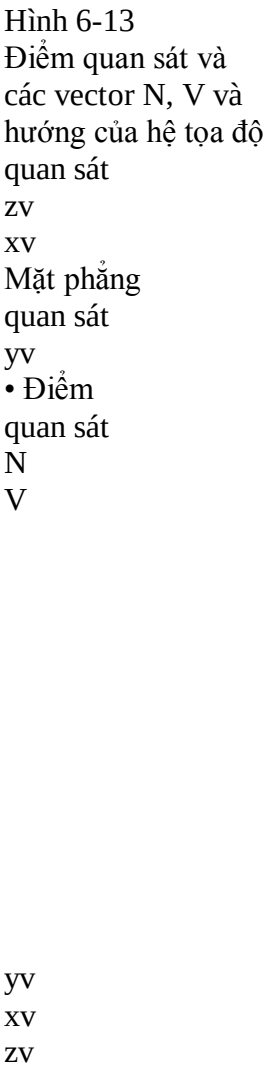
nhìn nào, ở các khoảng cách khác nhau, và với các hướng camera khác nhau. Những gì xuất hiện trong kính ngắm được chiếu lên bề mặt film phẳng. Kiểu len của camera, cái mà chúng ta dùng để xác định phần nào của đối tượng hoặc cảnh vật xuất hiện trên bức ảnh sau cùng. Các ý tưởng này được kết hợp chặt chẽ trong một gói đồ họa. Chúng ta yêu cầu người sử dụng chỉ rõ một điểm để từ đó quan sát các đối tượng và chỉ ra bao nhiêu cảnh cần được chứa đựng vào trong hiển thị sau cùng.

6.3.1. Xác định mặt phẳng quan sát

Người dùng chỉ định rõ cách nhìn cụ thể cảnh bằng việc định nghĩa một mặt phẳng quan sát (view plane). Mặt phẳng quan sát là bề mặt để ta chiếu quang cảnh của một đối tượng lên đó. Chúng ta có thể nghĩ về nó như film trong một camera, cái được bố trí và được định hướng để đặt các bức ảnh được yêu cầu vào. Mặt phẳng quan sát được

xây dựng bằng việc định rõ hệ quan sát (view coordinate system), như được trình bày trong hình 6-12. Các vị trí trên hệ tọa độ thế giới thực sẽ được định nghĩa lại và diễn tả mối liên hệ tương ứng đến hệ tọa độ này.
Để xây dựng các hệ quan sát, người sử dụng chọn một vị trí trên hệ tọa độ thế giới thực để dùng nó như điểm quan sát (view reference point). Đây sẽ là gốc của hệ quan sát. Hướng của mặt phẳng quan sát được định nghĩa bằng việc xác định vector pháp tuyến của mặt phẳng quan sát (view plane normal vector), N. Vector này xây dựng hướng cho trục z dương của hệ quan sát. Một vector dựng đứng V, được gọi là vector nhìn lên (view up vector), được dùng để định nghĩa hướng cho trục y dương. Hình 6-13 minh họa hướng của hệ quan sát, ở đó mặt phẳng quan sát là mặt xy.

Trang 108
Hình 6-12
Hệ quan sát với các trục xv, yv, và zv. Mô tả đối tượng trong tọa độ thế giới thực được chuyển sang hệ tọa độ quan sát.
yw
zR
w
xw
zv
xv
yvChương 6: Quan sát ảnh ba chiều



Hình 6-13
Điểm quan sát và
các vector N, V và
hướng của hệ tọa độ

(-1, -1, 0)
N•
Mặt quan sát
(0, 0, 0)

(b)
N
yv
xv
zv
(-1, 0, 0)
N•
Mặt quan sát
(0, 0, 0)
(a)
N
Hình 6-14
Hướng của mặt phẳng quan sát để xác định các tọa độ vector pháp tuyến. Vị trí (-1, 0, 0) định hướng mặt phẳng quan sát trong (a), trong khi đó vị trí (-1, -1, 0) cho hướng trong (b).

Vector pháp tuyến của mặt phẳng quan sát N có thể được xây dựng bằng việc xác định một vị trí tọa độ liên hệ với gốc tọa độ thế giới thực. Việc làm này định nghĩa hướng của vector pháp tuyến như đường thẳng từ gốc (của tọa độ thế giới thực) đến vị trí tọa độ

được chỉ định (gốc hệ quan sát). Hình 6-14 cho hai hướng của mặt phẳng quan sát để các tọa độ vector pháp tuyến được xác định. Vector V có thể được xác định theo cách tương tự. Người sử dụng thường khó khăn để xác định chính xác hai vector vuông góc này, vì vậy một vài gói đồ họa thay đổi cách xác định vector V của người dùng. Như được thể hiện trong hình 6-15, V được chiếu đến vị trí để vuông gốc với pháp vector.

Vị trí đượ
chiếu
Vị trí được
xác định theo
lý thuyết
Hình 6-15 Thay đổi sự xác định theo l

c

ý
thuyết của vector V đến vị trí vuôn g góc
với vector N.
V
N

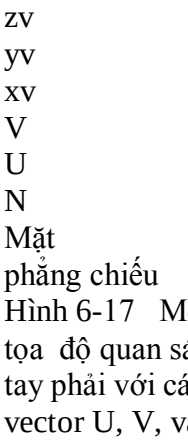
Đôi khi vector thứ ba U, được dùng để chỉ rõ hướng x của hệ quan sát. Hệ quan sát sau đó có thể được mô tả như hệ uvn, và mặt phẳng quan sát được gọi là mặt uv. Chúng ta
giả thuyết rằng vị trị x theo hướng như ở hình 6-16. Hướng của U và V trong bức ảnh này thì không đổi so với hướng chuẩn của trục x và y trên thiết bị hiển thị. Chúng ta có thể nghĩ về mặt phẳng quan sát trong hệ quan sát này như một thiết bị logic (logical device) làm cơ sở cho việc hiển thị ảnh.

zv
yv
xv
V
U
N
Mặt phẳng chiếu
Hình 6-16 Hệ uvn
định nghĩa các
hướng cho các trục
của một hệ quan sát
bàn tay trái.

Dù là hệ tọa độ bàn tay trái (xem hình 6-16) hay hệ tọa độ bàn tay phải (xem hình 6-17) đều có thể được dùng làm hệ quan sát. Trong các thảo luận sau này, chúng ta sẽ dùng hệ tọa độ bàn tay trái, vì nó trực quan hơn một chút. Các đối tượng xa hơn từ người quan sát có các giá trị theo trục z lớn. Tuy nhiên, hệ tọa độ bàn tay phải thường được

dùng, vì nó có hướng tương tụ như hệ tọa độ thế giới thực. Do đó, sự biến đổi giữa hai hệ

này được làm đơn giản.



Hình 6-17 Một hệ tọa độ quan sát bàn tay phải với các vector U, V, và N.

Trong việc xây dựng mặt phẳng quan sát, vài vùng đồ họa sử dụng các tham số bổ sung được gọi là khoảng cách quan sát. Mặt phẳng quan sát được định nghĩa như mặt phẳng song song với mặt phẳng xy, cái nằm ở một khoảng cách xác định từ điểm quan sát. Đối với thảo luận của ta, chúng ta giả thuyết rằng mặt phẳng quan sát là mặt xy ở gốc tọa độ của hệ quan sát. Điều này cho phép chúng ta chiếu lên mặt z = 0.

Để tạo ra một quang cảnh từ một điểm quan sát thuận lợi do người dùng chọn, các vị trí được định nghĩa liên hệ với gốc của hệ tọa độ thế giới thực phải được định nghĩa lại liên hệ với gốc của hệ quan sát. Tức là, chúng ta phải biến đổi các tọa độ từ hệ tọa độ thế giới thực sang hệ tọa độ quan sát. Sự biến đổi này được thực hiện bằng một dãy biến đổi tuần tự của phép tịnh tiến và phép quay để ánh xạ các trục của hệ tọa độ quan sát lên trên các trục của hệ tọa độ thế giới thực. Khi được áp dụng đến định nghĩa hệ tọa độ thế giới thực của các đối tượng trong ảnh, dãy biến đổi tuần tự này biến đổi chúng đến vị trí mới trong hệ tọa độ quan sát. Ma trận biểu diễn dãy biến đổi tuần tự này có thể được thu được bằng việc kết hợp các ma trận biến đổi như sau (xem hình 6-18):

1. Phản chiếu liên hệ đến mặt xy, đảo ngược dấu mỗi tọa độ z. Điều này thay đổi hệ quan sát bàn tay trái thành hệ quan sát bàn tay phải.
2. Tịnh tiến điểm quán sát đến gốc của hệ tọa độ thế giới thực.

3. Quay quanh trục tọa độ thế giới thực x để mang trục tọa độ quan sát z vào mặt phẳng xz của hệ tọa độ thế giới thực.
Trang 111 Chương 6: Quan sát ảnh ba chiều
4. Quay quanh trục tọa độ thế giới thực y cho đến khi trục z của cả hai hệ trùng nhau.
5. Quay quanh trục tọa độ thế giới thực z để trục y của hệ quan sát và hệ thế giới thực trùng nhau.

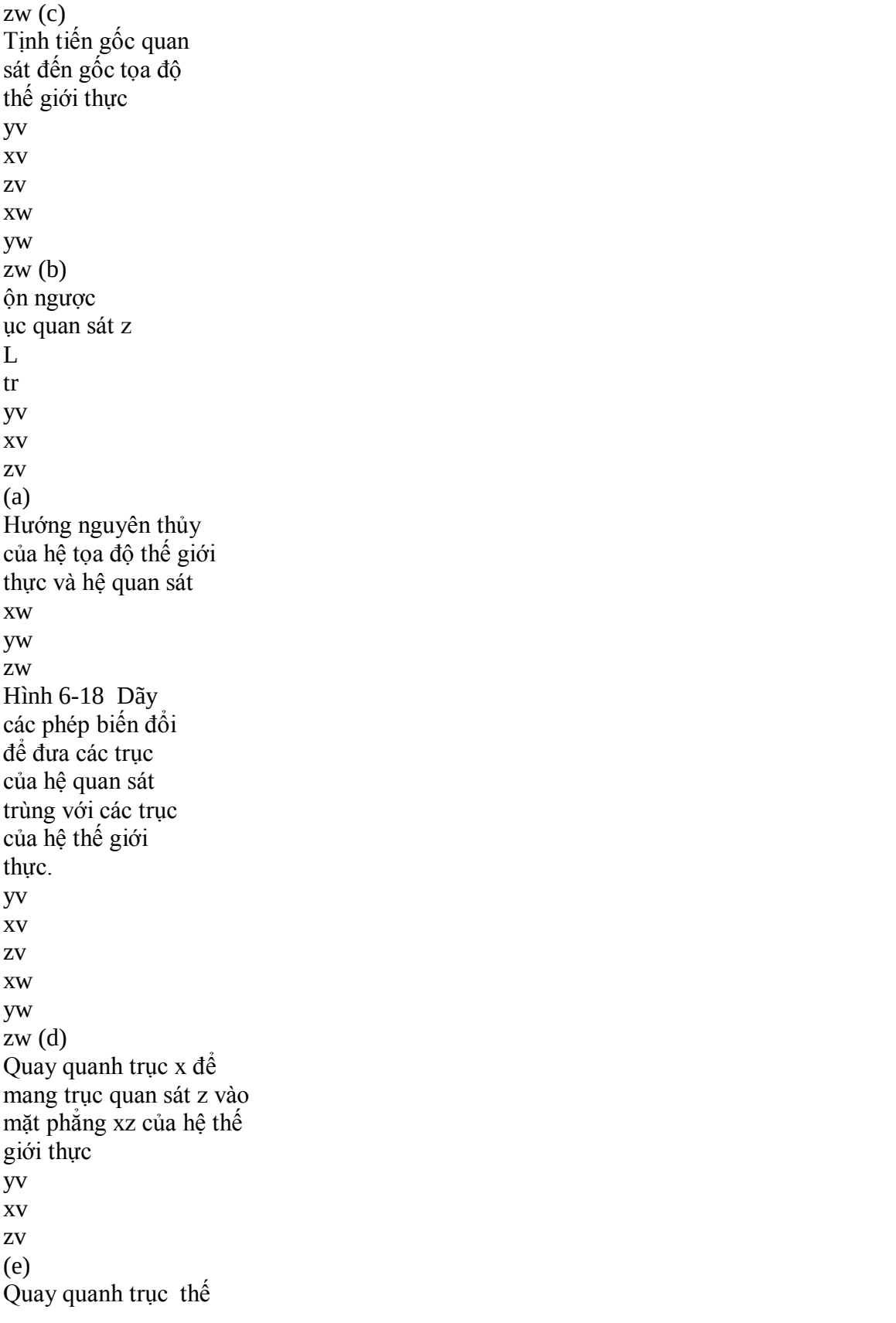
Kết quả của mỗi phép biến đổi trên được thể hiện trong hình 6-18. Dãy tuần tự các biến đổi này có nhiều điểm chung với dãy các biến đổi để quay một đối tượng xung quanh một trục bất kỳ, và các thành phần của ma trận quan sát có thể được xác định bằng cách dùng các kỹ thuật tương tự kỹ thuật quay quanh một trục bất kỳ. Đối với các gói dùng hệ quan sát bàn tay phải, phép nghịch đảo giá trị z ở bước 1 là không cần thiết.

6.3.2. Không gian quan sát

Trong camera tương tự (analogy), kiểu len được dùng trên camera là yếu tố quyết định bao nhiêu cảnh được bắt trên film. Một len góc rộng (wide-angle len) giữ nhiều cảnh hơn len bình thường (regular len). Trong quan sát ba chiều, một cửa sổ chiếu được dùng với hiệu quả tương tự. Cửa sổ được định nghĩa bằng các giá trị nhỏ nhất và lớn nhất của x và y trên mặt quan sát (xem hình 6 -19). Hệ quan sát được dùng để tạo ra giới hạn của cửa sổ, cái có thể xuất hiện ở bất kỳ đâu trên mặt phẳng quan sát.

xw

yw


zw (c)
Tịnh tiến gốc quan
sát đến gốc tọa độ
thế giới thực
yv
xv
zv
xw
yw
zw (b)
ộn ngược
ục quan sát z
L
tr
yv
xv
zv
(a)
Hướng nguyên thủy
của hệ tọa độ thế giới
thực và hệ quan sát
xw
yw
zw
Hình 6-18 Dãy
các phép biến đổi
để đưa các trục
của hệ quan sát
trùng với các trục
của hệ thế giới
thực.
yv
xv
zv
xw
yw
zw (d)
Quay quanh trục x để
mang trục quan sát z vào
mặt phẳng xz của hệ thế
giới thực
yv
xv
zv
(e)
Quay quanh trục thế

giới thực y để hai
trục z trùng nhau
xw
w
zw
y yw
(f)
Quay quanh trục thế
giới thực z để hai hệ
trùng nhau
yv xv
zv
xw
zw
yv
xv
zv

Cứa sổ chiếu được dùng để định nghĩa một không quan sát (view volume). Chỉ những đối tượng nằm trong không quan sát mới được chiếu và hiển thị lên mặt phẳng chiếu. Hình dạng chính xác của không quan sát dựa vào kiểu phép chiếu được yêu cầu bởi người dùng. Trong bất kỳ trường hợp nào, bốn mặt của không gian quan sát đi xuyên qua các cạnh của cửa sổ. Với phép chiếu song song, bốn mặt của không gian quan sát này hình thành một hình hộp không giới hạn (xem hình 6-20). Một hình chóp bị cắt cụt (hình kim tự tháp), với đỉnh nằm ở tâm chiếu (xem hình 6-21), được dùng như không gian quan sát cho phép chiếu phối cảnh. Hình chóp bị cắt cụt này được gọi là một hình cụt (frustum).

Hình 6-21 Không gian quan sát
cho phép chiếu phối cảnh.
• Tâm chi
Không gian
quan sát
Cửa sổ
ếu

Hình 6-19
Sự xác định cửa sổ trên mặt phẳng chiếu, tọa độ điểm thấp nhất và cao nhất được cho trong hệ quan sát
Mặt phẳng
chiếu
(xwmin, ywmin)
xv zv
Cửa sổ
yv
•
•
(xwmax, ywmax)
Cửa sổ
(Window)
Không gian quan sát
(View Volume)
Hình 6-20 Không gian quan sát cho phép chiếu song song.

Vài vùng đồ họa giới hạn tọa độ của tâm chiếu là các vị trí dọc theo trục z của hệ quan sát. Chúng ta cần một tiếp cận tổng quát hơn là cho phép tâm chiếu được đặt ở bất

kỳ vị trí nào trong hệ quan sát. Hình 6-22 trình bày hai hướng của không gian quan sát hình chóp liên hệ với các trục quan sát. Trong hình 6-22 (b), không điểm nào chiếu đến mặt phẳng quan sát, vì tâm chiếu và các đối tượng được quan sát thì ở cùng phía với mặt quan sát. Trong trường hợp này, không có cái gì nào được hiển thị.

yv
xv
zv
Phía trước mặt
phẳng quan sát
(b)
•
yv
xv
zv
Phía sau mặt
phẳng quan sát

(a)
•
Hình 6-22
Hai vị trí tâm chiếu
của không gian
quan sát trong phép
chiếu phối cảnh.

Trong các phép chiếu song song, hướng của phép chiếu định nghĩa hướng của không gian quan sát. Bằng cách cho một vị trí liên hệ đến gốc hệ quan sát, người dùng định nghĩa được một vector xác định hướng của không gian quan sát liên hệ với mặt phẳng quan sát. Hình 6-23 trình bày hình dạng của các không gian quan sát cho cả hai: phép chiếu song song trực giao và phép chiếu song song xiên. Thông thường, một hoặc hai mặt phẳng bổ sung được dùng để định nghĩa rõ hơn không gian quan sát. Gồm một mặt gần (near plane) và một mặt xa (far plane) tạo ra không gian quan sát có giới hạn, được bao quanh bởi sáu mặt phẳng, (xem hình 6-24). Các mặt gần và xa thì luôn song song với mặt phẳng quan sát, và chúng được xác định bởi các khoảng cách với mặt phẳng quan sát trong hệ quan sát. Các tên lần lượt cho các mặt gần và mặt xa là các mặt ở đây, ở đằng kia hay các mặt ở phía trước, phía sau.

Không gian
quan sát
Cửa sổ
Phép chiếu xiên
(a)
Hướng chiếu
zv
xv
Phép chiếu trực giao
(b)
Không gian
quan sát
Cửa sổ
Hướng chiếu
zv
xv

Hình 6-23 Các không gian quan sát cho các phép chiếu song song xiên và trực giao, được quan sát tại mặt xz.

Với các mặt phẳng này, người dùng có thể loại bỏ một số phần của cảnh khi thực hiện quan sát dựa trên độ sâu của chúng. Đây là một ý tưởng độc đáo khi dùng đến phép chiếu phối cảnh. Các đối tượng ở rất xa mặt phẳng quan sát khi chiếu đến có thể chỉ còn là một điểm đơn. Các đối tượng ở rất gần có thể che khuất các đối tượng khác mà người dùng muốn xem. Hoặc, khi được chiếu, các đối tượng ở gần có thể lớn đến nỗi mà chúng nó vượt quá các biên cửa sổ và không thể được nhận ra.

6.3.3. Clipping

Một thuật toán clipping ba chiều xác định và lưu giữ tất cả các đoạn thẳng trong phạm vi không gian quan sát để sau đó chiếu lên mặt phẳng quan sát. Tất cả các đoạn thẳng bên ngoài không gian quan sát sẽ bị vứt bỏ. Việc clipping này có thể được thực hiện bằng cách dùng một sự mở rộng thuật toán clipping đường hai chiều hoặc dùng các phương pháp clipping đa giác. Các phương trình mặt phẳng định nghĩa các biên của không gian quan sát có thể được dùng đến để kiểm tra các vị trí liên hệ của các điểm đầu mút đoạn thẳng và để định vị các giao điểm.

Bằng cách thay thế các tọa độ của một điểm đầu mút đoạn thẳng vào trong phương trình mặt của biên, chúng ta có thể xác định được điểm đầu mút đó thì ở trong hay ở ngoài biên. Một điểm đầu mút (x, y, z) của đoạn thẳng thì ở ngoài một mặt phẳng biên nếu $Ax + By + Cz + D > 0$, với A, B, C, và D là các tham số mặt của biên đó. Tương tự, điểm ở trong biên nếu $Ax + By + Cz + D < 0$. Các đoạn thẳng có cả hai điểm đầu mút nằm bên ngoài một mặt phẳng biên sẽ bị vứt bỏ, và các đoạn thẳng nào có cả hai điểm đầu mút nằm bên trong tất cả các mặt biên sẽ được giữ lại. Giao điểm của một đoạn thẳng với một mặt biên được tìm bằng cách dùng các phương trình đường thẳng và phương trình mặt. Tọa độ giao điểm (x1, y1, z1) là các giá trị trên đường thẳng và thỏa phương trình mặt $Ax1 + By1 + Cz1 + D = 0$.

Khi hệ thống đã xác định được các đối tượng, mỗi đối tượng có độ ưu tiên riêng trong không gian quan sát, chúng nó được chiếu đến mặt phẳng quan sát. Tất cả các đối tượng trong không gian quan sát sẽ rơi nằm vào phạm vi cửa sổ chiếu. Cũng như trong không gian hai chiều, nội dung của cửa sổ sẽ được ánh xạ đến một vùng quan sát do người dùng chỉ định. Điều này làm chuẩn hóa các hệ tọa độ, sau đó, chúng được chuyển đổi đến các hệ tọa độ thiết bị thích hợp để hiển thị (xem hình 6-24).

Hình 6-24
Các không gian quan sát được bao bởi các mặt gần , mặt xa và bởi các mặt trên đỉnh và dưới đáy. Các khoảng cách đến các mặt gần và xa được xác định bởi dn và df.

6.4. Cài đặt các thao tác quan sát (Implementation of Viewing Operations)

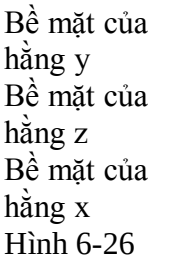
Hình 6-25 Các thao tác logic trong việc xem ảnh ba chiều

Chúng ta có thể khái niệm hóa một dãy các thao tác để thực hiện quan sát như trong hình 6-25. Đầu tiên, các mô tả hệ tọa độ thế giới thực được biến đổi sang hệ tọa độ quan sát. Tiếp đến, cảnh được quan sát bị cắt bởi một không gian quan sát và được chiếu vào vùng cửa sổ được định nghĩa trên mặt phẳng quan sát. Cửa sổ này sau đó được ánh xạ lên một vùng quan sát (vùng này đã được định rõ trong hệ tọa độ thiết bị chuẩn). Bước

Chuyển sang
các hệ tọa độ
quan sát
Cắt khỏi không
gian quan sát
Chiếu đến cửa
sổ
Biến đổi đến
vùng quan sát
hai chiều
Biến đổi đến các
hệ tọa độ thiết bị
CÁC HỆ
TỌA ĐỘ
THẾ GIỚI
THỰC BA
CHIỀU
CÁC HỆ
TỌA ĐỘ
QUAN SÁT
BA CHIỀU
CÁC HỆ
TỌA ĐỘ
QUAN SÁT
BA CHIỀU

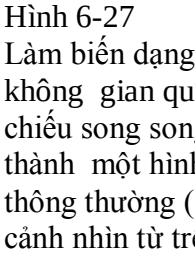
CÁC HỆ
TỌA ĐỘ
QUAN SÁT
HAI CHIỀU
CÁC HỆ
TỌA ĐỘ
THIẾT BỊ
CHUẨN
HAI
CHIỀU
Trang 116 Chương 6: Quan sát ảnh ba chiều
cuối cùng là phải biến đổi mô tả trong hệ tọa độ thiết bị chuẩn vào trong các hệ tọa độ thiết bị và hiển thị quang cảnh lên một thiết bị xuất.
Mô hình được trình bày ở hình 6-25 thì hữu ích như mô hình cho lập trình viên hoặc cho việc khái niệm hóa các thao tác quan sát ba chiều. Tuy nhiên, để hiệu quả, việc cài đặt thật sự của việc quan sát ba chiều trong một gói đồ họa cần một hình thức khác hơn nhiều. Trong phần này, chúng ta nhìn vào những vấn đề, nơi mà các lo lắng về cài đặt làm cho chúng ra xa rời với mô hình cơ bản của việc quan sát ba chiều
Các không gian quan sát được chuẩn hóa (Normalized View Volumes)
Clipping trong không gian hai chiều được thực hiện một cách tổng quát bởi một hình chữ nhật có các cạnh song song với trục x và y. Điều này làm đơn giản rất nhiều các tính toán clipping, vì mỗi biên cửa sổ được xác định bởi chỉ một giá trị tọa độ. Ví dụ, các giao điểm của các đoạn cắt biên trái cửa sổ đều có giá trị tọa độ x bằng với biên trái.
Trong mô hình của lập trình viên ba chiều, việc clipping được thực hiện bởi một không gian quan sát được xác định bởi cửa sổ chiếu, kiểu chiếu, và các mặt gần, xa. Bởi vì các mặt gần và xa song song với mặt phẳng chiếu, mỗi mặt có giá trị tọa độ z không đổi. Tọa độ z của các giao điểm của các đoạn với các mặt này thì đơn giản là tọa độ z của các mặt phẳng tương ứng. Nhưng bốn mặt còn lại của không gian quan sát có thể có hướng không gian tùy ý. Để tìm giao điểm của một đoạn với một trong các mặt đó ta cần tìm phương trình cho mặt phẳng chứa các mặt của không gian quan sát. Tuy nhiên, điều này trở nên không cần thiết nếu chúng ta biến đổi không gian quan sát trước khi clipping đến một hình hộp thông thường.
Clipping bởi một hình hộp thông thường thì đơn giản hơn bởi vì mỗi mặt bây giờ vuông góc với một trong các trục tọa độ. Như được trình bày trong hình 6-26, đỉnh và đáy của một không gian quan sát như thế là các mặt phẳng của hằng y, các mặt là các mặt phẳng của hằng x, và các mặt gần và xa có một giá trị z được xác định trước. Ví dụ, tất cả các đoạn thẳng cắt mặt trên đỉnh của hình hộp, bây giờ sẽ có giá trị tọa độ y của mặt đó. Thêm vào đó, để làm đơn giản hóa các thao tác clipping, việc biến đổi thành một hình hộp thông thường làm rút ngắn quá trình xử lý chiếu thành một phép chiếu trực giao đơn giản. Chúng ta đầu tiên xem xét làm thế nào để biến đổi một không gian quan sát thành một hình hộp thông thường, sau đó thảo luận về thao tác chiếu.
Trang 117 Chương 6: Quan sát ảnh ba chiều

xv
yv
zv



Hình 6-26
Một không gian quan sát hình hộp thông thường.

Trong trường hợp của một phép chiếu song song trực giao, không gian quan sát đã là một hình hộp chữ nhật ngay từ đầu. Đối với phép chiếu song song xiên, chúng ta làm biến dạng không gian quan sát để làm cho cùng phương hướng chiếu với hướng vector pháp tuyến của mặt phẳng quan sát, N. Phép biến dạng này mang các mặt của không gian quan sát hình hộp thành mặt quan sát, như trong hình 6-27.


Hình 6-27
Làm biến dạng một không gian quan sát chiếu song song xiên thành một hình hộp thông thường (quang cảnh nhìn từ trên đỉnh

xuống).
Mặt gần
Mặt xa
(a)
Hướng
nguyên thủy
Không
gian quan
sát
Hướng của
phép chiếu
Cửa sổ
N
Mặ
Mặ
(b)
Sau khi
biến dạng
Không
gian quan
sát
Hướng của
phép chiếu
Cử
t gần
t xa
a sổ
N

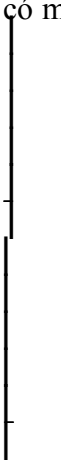
Đối với không gian quan sát trong một phép chiếu phối cảnh, phép biến dạng và biến đổi tỷ lệ được cần để tạo ra hình hộp chữ nhật. Chúng ta đầu tiên làm biến dạng theo phương x và y để mang tâm chiếu đặt lên đường thẳng vuông góc tại tâm cửa sổ (Hình 6-28). Với đỉnh tại điểm này, các mặt đối diện của hình chóp cụt (trừ hai mặt gần, xa, ta có trái đối với phải, đỉnh đối với đáy) có cùng kích thước. Chúng ta sau đó áp dụng phép biến đổi tỷ lệ để biến đổi các mặt của hình chóp cụt hành các mặt chữ nhật của một hình hộp thông thường.

Hình 6-29 trình bày một quang cảnh bên sườn của hình chóp cụt đối với phép chiếu song song. Để biến đổi hình chóp cụt này thành hình chữ nhật thông thường với chiều cao bằng với chiều cao cửa sổ, chúng ta áp dụng một phép biến đổi tỷ lệ liên hệ với điểm cố định (xF, yF, 0) ở tâm cửa sổ. Phép biến đổi này phải biến đổi theo tỷ lệ giữa các điểm trong hình chóp ở xa cửa sổ hơn so với các điểm ở gần cửa sổ hơn để mang chúng vào trong vùng hình hộp. Thực tế, hệ số tỷ lệ được cần sẽ tỷ lệ nghịch với

khoảng cách đến cửa sổ. Đối với một tâm chiếu ở khoảng cách d phía sau cửa sổ, hệ số tỷ lệ được cần là d/(z+d), với z được xem như khoảng cách từ điểm đến cửa sổ. Ma trận cho phép biến đổi tỷ lệ này là

Ở đây hệ số tỷ lệ là S = d(z + d). Tất cả các giá trị tọa độ x và y trong cảnh được biến đổi tỷ lệ bởi phép biến đổi này. Các điểm trong không gian này được ánh xạ thành các điểm trong hình hộp mà không có sự thay đổi giá trị của z (hình 6-29)

Việc chuyển đổi thành một hình hộp còn có một lợi ích quan trọng khác. Các phép biến đổi



− − 10)1()1(
010 0
00 0
000
F F ySxS

S
S
(6-13)
Không
gian
quan sát
M
xa
ặt
Mặt
gần
Cửa
sổ
z z

Tâm
chiếu
z
d
(x, y, z)
(xF, yF, 0)
Hướng z
Hình hộp th
thường
M

ặ
gần
Cửa
sổ
z z
Tâm
chiếu

d
xv
yv
zv
Hình 6-30
Biến đổi một hình hộp thông thường (a)
thành một hình lập phương đơn vị (b).
xv
yv
zv
1
1

•
•
•
(a)
(b)
1
tâm
Hình 6-28
Biến dạng một không gian quan
sát chiếu phối cảnh để mang
chiếu lên đường vuông góc vớ
cửa sổ (quang cảnh trên đỉnh)
Tâm chiếu

Sau khi biến đổi
(b)
i tâm
•
Xa
Cửa sổ
Không gian quan
sát
Gần
N
Tâm chiếu

Hướng nguyên
thủy
(a)
•
Cửa sổ
Không gian quan
sát
Gần
N
Xa

được áp dụng thực hiện một số lượng lớn công việc cần thiết để chiếu các điểm nguyên thủy lên mặt phẳng chiếu. Ví dụ, phép biến đổi để chuyển hình chóp thành một hình hộp nhất thiết phải thực hiện phép chiếu phối cảnh. Các vị trí tọa độ được biến đổi sang các giá trị chiếu x và y, tuy nhiên ta giữ nguyên các giá trị khác không z. Các điểm nằm trong phạm vi các giá trị z của hai mặt gần và xa có thể được chiếu bằng cách đơn giản vứt bỏ tọa độ z. Vì thế bằng việc biến đổi không gian quan sát thành hình hộp thông thường, thao tác chiếu được rút gọn thành một phép chiếu trực giao đơn giản.
Chúng ta có thể làm các thao tác quan sát hiệu quả hơn. Việc biến đổi không gian quan sát thành hình hộp thông thường (cái được làm tương tự như thao tác chiếu) xảy ra

liền sau việc ánh xạ từ tọa độ thế giới thực sang hệ tọa độ quan sát. Nếu chúng ta kết hợp các ma trận lại để làm một lúc một dãy các thao tác này, mỗi vị trí tọa độ có thể được chuyển từ vị trí của nó trong tọa độ thế giới thực sang vị trí tương ứng trong hình hộp chỉ là một bước thực hiện.

Vài gói đồ họa thực hiện việc clipping bằng cách dùng hình hộp thông thường như vừa được trình bày. Các phần của đối tượng trong phạm vi hình hộp được chiếu đến mặt trước (front plane) và sau đó được ánh xạ đến vùng quan sát hai chiều.

Các gói đồ họa khác thì ánh xạ hình hộp này vào một hình lập phương đơn vị (unit cube) (hình 6-30) trước khi clipping và chiếu.

Hình lập phương đơn vị là một không gian được xác định bởi các mặt sau:

x =0, x= 1, y = 0, y = 1, z = 0, z = 1 (6-14)

Vì hình lập phương đơn vị được định nghĩa bởi các giá trị trong đoạn [0..1], nó có thể được xem như một không gian quan sát chuẩn hóa (normalized view volume). Cũng như với hình hộp, khi các thành phần nằm trong không gian vừa được ánh xạ đến mặt trước (front plane), các điểm đó sẽ được ánh xạ đến một vùng quan sát hai chiều.

Như một chọn lựa khác, hình hộp thông thường, được xác định bởi cửa sổ mặt quan sát, có thể được ánh xạ đến một vùng quan sát ba chiều (three-dimensional viewport) trước khi clipping. Vùng quan sát này là một hình hộp thông thường được định nghĩa trong hệ tọa độ chuẩn hóa. Việc ánh xạ từ cửa sổ-đến–vùng quan sát trong không gian ba chiều cần được thực hiện với một phép biến đổi kết hợp tỷ lệ và tịnh tiến tương tự như với việc ánh xạ từ cửa sổ-đến–vùng quan sát trong không gian hai chiều. Chúng ta có thể biểu diễn ma trận biến đổi ba chiều của tập các thao tác này như sau:

000
000
000
z
y
x
D
D
D

Các tham số Dx, Dy, và Dz là các tỷ lệ về kích thước của vùng quan sát so với không gian quan sát hình hộp theo các hướng x, y, và z (xem hình 6-31):

ywmax - ywmin
df - dn
xwmax - xwmin

yvmax - yvmin min xvmax - xv
zvmax - zvmin
y
z
x
Hướng các trục
tọa độ
Không gian quan sát
được xác định bởi các
tọa độ cửa sổ và các
mặt gần và xa
Vùng quan
sát ba chiều
Hình 6-31
Các kích thước
của không gian
quan sát và
vùng quan sát

min max

min max
xwxw
xvxv
Dx
–
–
=

min max
min max
ywyw
yvyv
Dy
–
–
= (6-16)
nf
z
dd
zvzv
D
–
–
= min max

ở đây các biên của không gian quan sát được xây dựng bởi các giới hạn cửa sổ (xwmin, xwmax, ywmin, ywmax), và các vị trí dn và df của các mặt gần và xa. Các biên của vùng quan
sát được thiết đặt với các giá trị tọa độ xvmin, xvmax, yvmin, yvmax, zvmin, và zvmax. Các tham
số bổ sung Kx, Ky, và Kz trong phép biến đổi là:

Kx = xvmin – xwmin * Dx
Ky = yvmin – ywmin * Dy (6-17)
Kz = zvmin – dn * Dz

Việc ánh xạ từ cửa sổ-đến-vùng quan sát được thực hiện trước khi clipping như
trong hình 6-32.

Hình 6-32 Thực hiện các phép biến đổi hệ quan sát để các thao tác có thể được nối kết vào
một ma trận biến đổi đơn, được áp dụng trước khi clipping.

Chuyển sang
các hệ tọa độ
quan sát
Biến đổi thành
một hình hộp
thông thường
Biến đổi đến
vùng quan sát
ba chiều

Thuận lợi của cách làm này là ma trận biến đổi chuẩn hóa (từ không gian quan sát-đến-ánh xạ vào vùng quan sát) có thể được kết hợp với ma trận biến đổi các tọa độ trong hệ thế giới thực sang các vị trí trong hình hộp. Ma trận kết quả biến đổi các vị trí trong phạm vi hệ tọa độ thế thực thành các điểm chiếu x và y trong vùng quan sát. Mỗi tọa độ của cảnh gốc cần được tịnh tiến chỉ một lần. Các điểm được tịnh tiến này bị clipping bởi vùng quan sát. Các giá trị x và y của các điểm trong không gian quan sát sau đó được biến đổi đến các hệ tọa độ thiết bị để hiển thị (xem hình 6-33).

Cắt khỏi vùng
quan sát ba
chiều
Thực hiện
chiếu trực giao
đến vùng quan
sát hai chiều
CÁC HỆ TỌA
ĐỘ THẾ GIỚI
THỰC BA
CHIỀU
CÁC HỆ
TỌA ĐỘ
QUAN SÁT
BA CHIỀU
CÁC HỆ
TỌA ĐỘ
QUAN
SÁT BA
CHIỀU
CÁC HỆ
TỌA ĐỘ
CHUẨN
HÓABA
CHIỀU
CÁC HỆ
TỌA ĐỘ
CHUẨN
HÓA BA
CHIỀU
CÁC HỆ
TỌA ĐỘ
QUAN
SÁT HAI
CHIỀU
Biến đổi đến
hệ tọa độ thiết
bị
xv
zv
yv
Vùng quan sát

ba chiều
Vùng quan sát
hai chiều
Hình 6-33
Ánh xạ phần bên trong của
một vùng quan sát ba chiều
(trong hệ tọa độ chuẩn hóa)
đến các tọa độ trên thiết bị.
Trang 122
Hệ tọa độ thiết bị chuẩn Thiết bị hiển thị Chương 6: Quan sát ảnh ba chiều

Clipping dựa vào một không gian quan sát được chuẩn hóa
Các bề mặt có thể bị cắt khỏi các biên vùng quan sát bằng các thủ đơn giản hơn trong đồ họa hai chiều. Dù là các thủ tục clipping đường hay cliping đa giác đều có thể được sửa lại cho thích hợp với clipping một vùng quan sát trong ba chiều. Các mặt cong được xử lý bằng cách dùng các phương trình mặt biên kết hợp với việc xác định đường cắt với các mặt của hình hộp. Bây giờ chúng ta xem các thủ tục clipping hai chiều được thay đổi thế nào để dùng cho ba chiều.
Các khái niệm trong hai chiều về các mã vùng có thể được mở rộng cho ba chiều bằng việc xem xét các vị trí phía trước và phía sau vùng quan sát ba chiều, cũng như các vị trị bên trái, bên phải, phía dưới, hoặc phía trên không gian. Đối với clipping hai chiều, chúng ta đã dùng mã vùng nhị phân bốn bit để xác định vị trí của các điểm đầu mút đoạn thẳng có quan hệ với các biên cửa sổ thế nào. Đối với các điểm ba chiều, chúng ta cần mở rộng mã vùng thành sáu bit. Mỗi điểm trong cảnh khi đó được gán một mã vùng sáu bit để xác định mối quan hệ với các mặt biên của vùng quan sát. Với một điểm đầu mút đoạn thẳng ở vị trí (x, y, z), ta gán các vị trí bit trong mã vùng từ phải sang trái như sau:
bit 1 =1 nếu x < xvmin (left)
bit 2 =1 nếu x > xvmax (right)
bit 3 =1 nếu y < yvmin (below)
bit 4 =1 nếu y > yvmax (above)
bit 5 =1 nếu z < zvmin (front)
bit 6 =1 nếu z > zvmax (back)
Ví dụ, một mã vùng 101000 chỉ ra rằng một điểm thì ở trên và phía sau vùng quan sát, trong khi đó mã vùng 000000 chỉ ra rằng một điểm nằm trong không gian quan sát. Một đoạn thẳng có thể được xác định ngay là hoàn toàn nằm trong vùng quan sát nếu cả hai điểm đầu mút của nó đều có mã vùng là 000000. Nếu điểm đầu mút nào không có mã vùng 000000, chúng ta thực hiện phép logic and lên hai mã đầu mút. Kết quả phép toán and sẽ khác 0 đối với các đoạn thẳng hoàn toàn nằm ngoài không gian quan sát. Nếu
Trang 123 Chương 6: Quan sát ảnh ba chiều
chúng ta không thể xác định được một đoạn thẳng là hoàn toàn nằm trong hay hoàn toàn nằm ngoài không gian, ta sẽ đi tìm giao điểm với các mặt biên của không gian.
Như trong clipping đường hai chiều, chúng ta dùng các giao điểm được tính của

đường với các mặt của vùng quan sát để xác định xem phần nào của đoạn thẳng bị vứt bỏ. Phần được giữ lại của đoạn sẽ được kiểm tra với các mặt khác, và chúng ta tiếp tục đến khi xác định được là đoạn bị vứt bỏ hoàn toàn hay đến khi thấy nó nằm bên trong không gian.

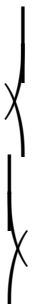
Việc xác định các giao điểm trong clipping đường, cũng như trong các thủ tục clipping đa giác, nên được làm sao cho hiệu quả. Các phương trình của các đoạn ba chiều được biểu diễn thuận tiện theo dạng tham số. Với một đoạn có hai điểm đầu mút P1 = (x1,
y1, z1) và P2 = (x2, y2, z2), chúng ta có thể viết phương trình tham số là

$$x = x1 + (x2 - x1)u$$
$$y = y1 + (y2 - y1)u \quad (6\text{-}18)$$
$$z = z1 + (z2 - z1)u$$

Tọa độ (x, y, z) biểu diễn cho một điểm bất kỳ trên đoạn thẳng giữa hai điểm đầu mút, và các tham số u thay đổi từ 0 đến 1. Giá trị u =0 tạo ra điểm P1, u=1 cho điểm P2. Để tìm giao điểm của một đường với một mặt của vùng quan sát, chúng ta thay thế giá trị tọa độ, cái là giá trị hằng của mặt đó, vào phương trình tham số 12-18 và giải tìm u. Cho trường hợp này, giả sử chúng ta đang xét một đường với mặt trước (front plane) của vùng quan sát. Khi đó z = zvmin, và

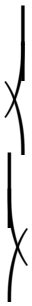
$$u = \frac{zz_{1min} - zz_{zv}}{-} \quad (6\text{-}19)$$

Khi giá trị u được tính bởi phương trình 12-19 không nằm trong đoạn [0..1], đều này có nghĩa là đoạn thẳng không cắt mặt trước ở bất kỳ điểm nào nằm giữa hai đầu mút P1 và P2 (đường A trong hình 6-34). Nếu giá trị u được tính nằm trong đoạn [0..1], chúng ta tính tọa độ giao điểm x và y như sau



$$-+= {}_{12}{}^{1min}$$

1211)(
zz
zzv
xxxx (6-20)



_
_
−+=
12
1min
1211)(
zz
zzv
yyyy

Nếu x1 hoặc y1 không nằm trong phạm vi các biên của vùng quan sát, khi đó đường thẳng này cắt mặt trước ở một điểm ở xa nào đó trên biên của không gian (đường B trong hình 6-34).
Thuật toán clipping đường Liang-Basky được thảo luận trong Chương 6 có thể được mở rộng cho ba chiều bằng việc xem xét các hiệu ứng (effect) của các mặt gần và xa. Các mặt này kết hợp với hai phép kiểm tra bổ sung trong quá trình xử lý tham số giao điểm u1 và u2.

Hình 6-34
Một quanh cảnh bên sườn ở mặt yz của hai đoạn thẳng bị cắt bởi mặt trước của khung nhìn. Với đoạn A, phương trình (12-19) tạo ra một giá trị của u ngoài đoạn [0..1]. Với đoạn B, phương trình (12-20) tạo ra một giao điểm bên ngoài đoạn từ yvmin đến yvmax.
yvmax
•
Tiết diện cắt của vùng quan sát ba chiều
y
z
yvmin
zvmin zvmax
•
•
•
A
B
P2
P1
P1
P2

6.5. Cài đặt phần cứng
Các chip đồ họa, dùng các kỹ thuật mạch điện VLSI (very large scale integration), được dùng trong nhiều hệ thống để thực hiện các thao tác xem ảnh. Các chip theo yêu cầu khách hàng này được thiết kế để biến đổi, clipping, và chiếu các đối tượng đến thiết bị xuất cho cả hai ứng dụng: hai chiều và ba chiều.

Hình 6-35 Một tập gồm 12 chip đồ họa giúp thực hiện các thao tác xem ảnh khác nhau.
ĐỊNH NGHĨA BỨC ẢNH Ở
HỆ TỌA ĐỘ THẾ GIỚI THỰC
Biến đổi đến các hệ tọa độ thiết bị

Các thao tác clipping
Các thao tác biến đổi Chương 6: Quan sát ảnh ba chiều

Hình 6-35 trình bày các thành phần của một loại chip đồ họa. Các chip được tổ chức vào một đường ống (pipeline) để thực hiện các thao tác biến đổi, clipping, và biến đổi hệ tọa độ. Bốn chip đầu tiên được cung cấp cho các phép toán ma trận liên quan đến biến đổi tỷ lệ, tịnh tiến, quay, và các phép biến đổi cần cho các phép chiếu trực giao và phối cảnh. Mỗi trong số sáu chip kế tiếp thực hiện clipping bởi các biên của vùng quan sát. Bốn trong số các chip này được dùng trong các ứng dụng hai chiều, và hai cái còn lại được cần cho việc clipping bởi các mặt gần và xa của vùng quan sát ba chiều. Hai chip sau cùng trong đường ống biến đổi hệ tọa độ vùng quan sát sang hệ tọa độ thiết bị xuất.

6.6. Lập trình xem ảnh ba chiều

Chương trình ví dụ sau đây minh họa việc sinh ra ảnh bằng chiếu phối cảnh và phép chiếu song song của một đối tượng.

```
Program Polycon;
Uses Crt, Graph, Graph3d;

Const MaxSommet = 50;
MaxFaces  = 30;
MaxAretes = 10;
IncAng    = 5;
IncRho    = 1;
IncEcran  = 20;

Var St : Array [1..MaxSommet, 1..3] of real;
Fc : Array [1..MaxFaces,0..MaxAretes] of integer;
fff : Array [1..MaxFaces] of boolean;
O1, O2, O3 : Real;
NF : Integer;
Pointille : Boolean;
```

```
Procedure VueDeDepart;
Begin
Projection := Perspective;
Rho := 15;  Theta := 0; Phi := 0;
DE := 400; Pointille  := True;
End;

Procedure LectureSommets;
Begin
St[1,1] :=  2;    St[1,2] :=  2.7;    St[1,3] :=  -2;
```

```
St[2,1] :=  2;    St[2,2] :=  2.7;    St[2,3] :=   0;
St[3,1] :=  2;    St[3,2] := -2.7;    St[3,3] :=   0;
St[4,1] :=  2;    St[4,2] := -2.7;    St[4,3] :=  -2;
St[5,1] := -2;    St[5,2] := -2.7;    St[5,3] :=  -2;
St[6,1] := -2;    St[6,2] :=  2.7;    St[6,3] :=  -2;
St[7,1] := -2;    St[7,2] :=  2.7;    St[7,3] :=   0;
St[8,1] :=  0;    St[8,2] :=  1.7;    St[8,3] :=   2;
St[9,1] :=  0;    St[9,2] := -1.7;    St[9,3] :=   2;
St[10,1] := -2;    St[10,2] := -2.7;    St[10,3] :=   0;
End;

Procedure LectureFaces;
Begin
NF := 9;
FC[1,0] := 4; FC[1,1] := 1; FC[1,2] := 2; FC[1,3] := 3; FC[1,4] := 4;
FC[2,0] := 4; FC[2,1] := 1; FC[2,2] := 6; FC[2,3] := 7; FC[2,4] := 2;
FC[3,0] := 3; FC[3,1] := 2; FC[3,2] := 7; FC[3,3] := 8;
FC[4,0] := 4; FC[4,1] := 2; FC[4,2] := 8; FC[4,3] := 9; FC[4,4] := 3;
FC[5,0] := 4; FC[5,1] := 1; FC[5,2] := 4; FC[5,3] := 5; FC[5,4] := 6;
FC[6,0] := 4; FC[6,1] := 7; FC[6,2] :=10; FC[6,3] := 9; FC[6,4] := 8;
FC[7,0] := 3; FC[7,1] := 3; FC[7,2] := 9; FC[7,3] :=10;
FC[8,0] := 4; FC[8,1] :=10; FC[8,2] := 5; FC[8,3] := 4; FC[8,4] := 3;
FC[9,0] := 4; FC[9,1] := 5; FC[9,2] :=10; FC[9,3] := 7; FC[9,4] := 6;
End;

ProCedure VecteurVision(St1, St2, St3:integer;  Var V1, V2, V3 : rEal);
Begin
V1 := O1 - St[St1,1]; V2 := O2 - St[St1,2]; V3 := O3 - St[St1,3];
End;

Procedure VecteurNormal(St1, ST2, St3:integer; Var N1, N2, N3 : Real);
Var P1, P2, P3, Q1, Q2, Q3 : Real;
Begin
```

```
P1 := ST[St2,1] - ST[St1,1];
P2 := ST[St2,2] - ST[St1,2];
P3 := ST[St2,3] - ST[St1,3];
Q1 := ST[St3,1] - ST[St1,1];
Q2 := ST[St3,2] - ST[St1,2];
Q3 := ST[St3,3] - ST[St1,3];
N1 := P2*Q3 - Q2*P3;
N2 := P3*Q1 - P1*Q3;
N3 := P1*Q2 - Q1*P2;
End;

Function ProDuitScalaire(V1, V2, V3, N1, N2, N3: Real):Real;
Begin
ProDuitScalaire := V1*N1 + V2*N2 + V3*N3
End;

Procedure DessineObject;
Var F, St1, St2, St3, NS, No : Integer;
V1, V2, V3, N1, N2, N3 : Real;
X, Y, Z,  XO, YO, ZO : Real;

Procedure DessineFace;
Var S : Integer;
Begin
Trang 127 Chương 6: Quan sát ảnh ba chiều
NS := FC[f,0];
For S := 1 To NS Do
Begin
No := FC[F,S]; X := ST[No,1]; Y := ST[No,2]; Z := ST[No,3];
If S = 1  Then Begin
DePlaceEn(X, Y, Z);
XO := X; YO := Y; ZO := Z;
End
Else Tracevers(X, Y, Z);
End;
TraceVers(XO, YO, ZO);
End;

Begin
FillChar(FFF, Sizeof(fff), #0);
SetLineStyle(DottedLn, 0, NormWidth);
SetColor(LightRed);
For F := 1 to NF Do
Begin
St1 := Fc[F,1]; St2 := Fc[F,2]; St3 := Fc[F,3];
VecteurVision(St1, St2, St3, V1, V2, V3);
```

```
VecteurNormal(St1, St2, St3, N1, N2, N3);
If ProDuitScalaire(V1, V2, V3, N1, N2,  N3) <= 0 then
Begin
If Pointille Then DessineFace;
FFF[f] := True;
End;
End;
SetLineStyle(SolidLn, 0, NormWidth);
SetColor(White);
For F := 1 to Nf Do
If Not FFF[F] Then DessineFace;
End;

Procedure CoordonneeOeil;
Begin
InitialiseProjection;
O1 := Rho * Aux7; O2 := Rho * Aux8; O3 := Rho * Aux2;
End;

Procedure Affichage;
Var S1, S2, S3, S4, S5 : String;
Begin
Cloture(0, MaxX, 0, 23);  ClearViewPort;
SetTextStyle(SmallFont, HorizDir, 4);
SetColor(14);
Str(Theta:3:1, S2); Str(Phi:3:1, S3); Str(DE:3:1, S4);
IF Projection = Perspective
Then Begin
Str(Rho:3:1, S1);
OutTextXY(80, 0,'Chieu Phoi Canh: Rho = '+S1+'   Theta = '+S2+
+'   Phi = '+S3+'   Ecran = '+S4);
End
Else OutTextXY(80,0, 'Chieu Song Song: Rho = infini  Theta = '+S2+
+'   Phi = '+S3+'   Ecran = '+S4);
Str(MaxX, S1); Str(MaxY, S2);
OutTextXY(5,  0, S1+' x '+S2);
OutTextXY(5, 12, 'Control: ArrowKey, E, A, +, -, T, C, F-Fine');
Cloture(0, MaxX, 24, MaxY);
End;

Procedure Commandes;
Trang 128 Chương 6: Quan sát ảnh ba chiều
Const RhoPara = 1e20;
Var RhoPersp, DEPersp, DEPara : Real;
Ch : Char;
Begin
```

```
VueDeDepart;
DEPara := 30;
CoorDonneeOeil;
DessineObject;
Affichage;
RePeat
Ch := UpCase(Readkey);
IF Ch = #0 Then Ch := UpCase(Readkey);
If Ord(Ch)  In [72,80,75,77,69,65,43,45,84,67,61,95]
Then
Begin
ClearDevice;
Case Ord(Ch) Of
72 : Phi := Phi + IncAng;
80 : Phi := Phi - IncAng;
75 : Theta := Theta + IncAng;
77 : Theta := Theta - IncAng;
69 : Rho := Rho + IncRho;
65 : Rho := Rho - IncRho;
43,61 : DE :=  DE + IncEcran;
45,95 : DE :=  DE - IncEcran;
84 : Pointille := not (Pointille);
67 : If Projection = Perspective
Then Begin
RhoPersp := Rho;
DEPersp := DE;
Projection := Parallele;
Rho := RhoPara;
DE := DEPara;
End
Else Begin
DEPara := DE;
Projection := Perspective;
Rho := RhoPersp;
De := DePersp;
End;
End;
CoordonneeOeil;
DessineObject;
Affichage
End;
Until (CH = 'F') Or (ch = #13) Or (ch =#27);
EcranTexte;
End;

Begin
```

```
EcranGraphique('');
Cloture(0, MaxX, 0, MaxY);
LectureSommets;
LectureFaces;
VueDeDepart;
Commandes;
End.
```


6.7. Các mở rộng đến Đường ống quan sát (Viewing Pipeline)

Ở điểm này, chúng ta thảo luận tập trung vào phần trung tâm của thao tác xem ảnh (viewing operation), thường được nói đến như phép biến đổi hệ quan sát (viewing transformation). Điều này gồm việc ánh xạ đến hệ quan sát, chiếu, và clipping. Chúng ta bây giờ đề cập đến các thao tác mà chúng có thể đến trước hoặc sau phép biến đổi hệ quan sát và làm ảnh hưởng đến hình ảnh sau cùng của một đối tượng.

Các gói đồ họa (cái cho phép các biến đổi ma trận) được kết hợp với các giai đoạn dùng đến ma trận trước khi thực biến đổi hệ quan sát. Chúng ta có thể nghĩ về điều này như việc quay hoặc bố trí lại đối tượng trước camera. Nếu một số giai đoạn phải được biến đổi, mỗi giai đoạn được biến đổi bằng các ma trận thích hợp, và tập hợp các đối tượng sau đó được chiếu bởi phép biến đổi hệ quan sát để hình thành ảnh cuối cùng. Khi ma trận biến đổi liên hệ đến bất kỳ giai đoạn bị thay đổi nào, toàn bộ quá trình xử lý xem ảnh phải được lặp lại.

Trong vài trường hợp, người dùng chỉ muốn thay đổi hình dạng bên ngoài (appearance) của cảnh trên thiết bị xuất. Có thể là các kỹ sư muốn quay mô hình (không có mặt trước để lộ rõ cấu trúc bên trong - cutaway) của vài bộ phận ba chiều vừa được chiếu. Hoặc có thể một ứng dụng hoạt hình cần di chuyển một đối tượng từ vùng này đến vùng khác trên màn ảnh. Kỹ sư có thể dùng các giai đoạn biến đổi hoặc yêu cầu một cảnh mới của phần dùng các tham số quan sát mới. Trong các trường hợp này, một cuộc hành trình thứ hai xuyên qua các đường ống quan sát được cần đến. Trong các trường hợp như thế, các hệ đồ họa đôi khi cung cấp các phép biến đổi ảnh (image transformation): các thay đổi được áp dụng đến phép chiếu hai chiều cuối cùng. Các phép biến đổi ảnh được áp dụng trong hai chiều, cho phép người dùng đặt lại vị trí một đối tượng trên màn hình mà không làm thay đổi hoàn toàn hình ảnh khi muốn xem mặt sau của nó. Vì những thay đổi này không cần thiết biến đổi hệ quan sát hay clipping ba chiều, nên chúng được thực hiện nhanh chóng.

6.8. Tổng kết chương 6

Sinh viên cần nắm được các nội dung cốt lõi của chương bao gồm các phép chiếu song song và phối cảnh. Ưu điểm của phép chiếu song song là có thể xác định được kích thước chính xác của các đối tượng trên ảnh thông qua các thông tin 2 chiều còn lại. Nhược điểm của phép chiếu song song là hình ảnh không thật do không có độ sâu. Ngược

lại, phép chiếu phối cảnh tạo ra các hình ảnh thực hơn nhưng không bảo toàn về chiều của các mối liên hệ.

6.9. Bài tập chương 6

1. Cài đặt một khối đa diện (ba chiều với các mặt phẳng) nằm trong góc 1/8 đầu tiên

của hệ tọa độ theo quy tắc bàn tay trái (tất cả các giá trị định nghĩa các đỉnh của đối tượng là dương). Phát triển một thủ tục (procedure) để thực hiện phép chiếu song song (được xác định bất kỳ) lên mặt phẳng xy.
2. Mở rộng thủ tục của bài tập 1 để cài đặt được các quang cảnh khác nhau của đối tượng bằng cách: đầu tiên, thực hiện các phép quay đối tượng quanh các trục quay (là các đường thẳng song song với với các mặt chiếu), sau đó chiếu đối tượng lên bề mặt quan sát.
3. Cài đặt thủ tục trong bài lập 1 để sinh ra một phép chiếu phối cảnh một điểm của đối tượng lên bề mặt chiếu, dùng phương trình 6-10 và khoảng cách quang sát d được xác định tùy ý dọc theo trục z âm.
4. Mở rộng ma trận biến đổi trong phương trình 6-10 để tâm chiếu có thể được chọn ở vị trí (x, y, -d) bất kỳ phía sau mặt phẳng chiếu.
5. Cài đặt thủ tục trong bài tập 1 để sinh ra một phép chiếu phối cảnh một điểm của đối tượng lên mặt phẳng chiếu, dùng ma trận biến đổi của bài tập 4.
6. Giả sử rằng một bề mặt chiếu được định nghĩa là mặt xy của hệ tọa độ quy tắc bàn tay trái, cài đặt sự định nghĩa tọa độ của một hình hộp chữ nhật trong hệ tọa độ này để nó nằm phía trước mặt phẳng chiếu. Dùng ma trận biến đổi của bài tập 4, hướng của khối sao cho thu được phép chiếu phối cảnh một điểm và hai điểm. Viết một chương trình để hiển thị hai quang cảnh phối cảnh này. Cái nào trong hai quang cảnh trên thực hơn.
7. Mở rộng thủ tục trong bài tập 6 để thu được một phép chiếu phối cảnh ba điểm. Bạn có thể phát hiện ra sự khác nhau lớn giữa phép chiếu hai điểm và ba điểm khống?
8. Phát triển một tập các thủ tục để biến đổi một mô tả đối tượng trong các hệ tọa độ thế giới thực sang các hệ quan sát (đã được xác định). Tức là, cài đặt hàm

(function) view_matrix, được cung cấp các tọa độ của điểm quan sát, pháp vector, và vector nhìn lên (view up vector).
9. Mở rộng các thủ tục trong bài tập 8 để thu được một phép chiếu song song (đã được xác định) của đối tượng lên một cửa sổ được định nghĩa trên mặt xy của hệ quan sát. Sau đó biến đổi cửa sổ đến một vùng quan sát trên màn ảnh. Giả sử rằng các đối tượng thì ở phía trước mặt phẳng quan sát và rằng không có việc clipping bởi một không gian quan sát nào được thực hiện.
10. Mở rộng các thủ tục trong bài tập 8 để thu được một phép chiếu phối cảnh (đã được xác định) của đối tượng lên một cửa sổ được định nghĩa trên mặt xy của hệ quan sát. Sau đó biến đổi cửa sổ đến một vùng vùng quan sát trên màn ảnh. Giả sử rằng các đối tượng thì ở phía trước mặt phẳng quan sát và rằng không có việc clipping bởi một không gian quan sát nào được thực hiện.
11. Nghĩ ra một thuật toán để cắt (clip) các đối tượng trong một quang cảnh bởi một hình chóp cụt đã được định nghĩa. So sánh các phép toán được cần trong thuật toán này với các phép toán được cần trong thuật toán cắt quang cảnh bởi một hình hộp thông thường.
12. Viết một chương trình thực hiện chiếu phối cảnh một hình chóp cụt thành một hình hộp thông thường.
13. Thay đổi thuật toán clipping đường Liang-Barsky hai chiều để cắt (clip) các đường ba chiều bởi một bởi một hình hộp (đã được xác định).

14. Mở rộng thuật toán của bài tập 13 để cắt một khối đa diện (đã được xác định) bởi một hình hộp.
15. Đối với cả hai phép chiếu song song và phối cảnh, hãy thảo luận các điều kiện để việc clipping ba chiều được thực hiện trước, phép chiếu lên mặt phẳng chiếu được thực hiện sau có thể tương đương với việc chiếu trước rồi thực hiện clipping sau.
16. Dùng bất kỳ thủ tục clipping nào, viết một chương trình thực hiện một phép biến đổi hệ quan sát hoàn chỉnh từ tọa độ thế giới thực sang vùng quan sát cho một phép chiếu song song trực giao của một đối tượng.
17. Mở rộng thủ tục của bài tập 16 để thực hiện một phép chiếu song song (được xác định bất kỳ) của một đối tượng lên một vùng quan sát đã được định nghĩa.
Trang 132 Chương 6: Quan sát ảnh ba chiều
18. Phát triển một chương trình để cài đặt một hướng quan sát hoàn chỉnh cho một phép chiếu phối cảnh. Chương trình phải biến đổi sự xác định hệ tọa độ thế giới thực của một đối tượng lên một vùng quan sát hai chiều đã được định nghĩa để hiển thị lên một phần của màn hình video.
19. Cài đặt các hàm set_view_representation và set_view_index để thực hiện một hướng chiếu (được xác định bất kỳ) trên một đối tượng được định nghĩa trong hệ tọa độ thế giới thực để thu được sự hiển thị vùng quan sát trên màn hình.
20. Thay đổi các thủ tục trong bài tập 19 để cho phép clipping bởi mặt phẳng của không gian quan sát bất kỳ. Điều này có thể được thực hiện với các tham số bổ sung đến tập các điều kiện clipping cho mỗi mặt phẳng là cắt (clip) hoặc không cắt (noclip).
Trang 133 Chương 7: Khử các mặt kuất và đường khuất
Chương 7
KHỬ CÁC MẶT KHUẤT VÀ ĐƯỜNG KHUẤT

7.1. Tổng quan
• Mục tiêu
Học xong chương này sinh viên cần phải nắm bắt được các vấn đề sau:
- Việc tạo ra các hình ảnh thực là sự xác định và xóa bỏ các phần của ảnh mà ta không nhìn thấy được từ một vị trí quan sát.
- Nắm vững các tiếp cận khử mặt khuất và đường khuất.
• Kiến thức cơ bản
Kiến thức toán học : kiến thức cơ bản về cách vẽ hình trong hình học không gian
Kiến thức tin học : kỹ thuật lập trình và cấu trúc dữ liệu.
• Tài liệu tham khảo
Computer Graphics . Donald Hearn, M. Pauline Baker. Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986 (chapters 13, 260-284)
• Nội dung cốt lõi
Các tiếp cận khử các mặt khuất, đường khuất bao gồm :
- Phương pháp dùng vùng đệm độ sâu
- Phương pháp đường quét
- Phương pháp sắp xếp theo độ sâu
- Phương pháp phân chia vùng
Trang 134 Chương 7: Khử các mặt kuất và đường khuất
7.2. Khử các mặt nằm sau (Back-Face Removal)

Một vấn đề lớn cần được quan tâm đến trong việc tạo ra các hình ảnh thực là sự xác định và xóa bỏ các phần của bức ảnh mà ta không nhìn thấy được từ một vị trí quan sát.

Có nhiều tiếp cận chúng ta cần để giải quyết vấn đề này, và cũng có nhiều thuật toán khác nhau đã và đang được phát triển để xóa bỏ các phần bị che khuất một cách hiệu quả cho những loại ứng dụng khác nhau. Vài phương pháp đòi hỏi nhiều bộ nhớ hơn, một vài cần nhiều thời gian xử lý hơn, một số khác lại chỉ áp dụng được cho những kiểu đối tượng đặc biệt. Phương pháp nào được chọn cho một ứng dụng cụ thể dựa vào các nhân tố như độ phức tạp của ảnh, kiểu đối tượng được hiển thị, các thiết bị hiện có, và các hình ảnh cần hiển thị là tĩnh hay động. Trong chương này, chúng ta khảo sát tỉ mỉ một vài trong số các phương pháp được dùng biến nhất để xóa bỏ các đường khuất và mặt khuất.

Phân loại các thuật toán

Các thuật toán về đường khuất và mặt khuất thường được phân loại dựa theo chúng nó được dùng để xử lý trực tiếp định nghĩa đối tượng hay xử lý hình chiếu của các đối tượng đó. Hai tiếp cận này được gọi là các phương pháp không gian đối tượng (object-space) và các phương pháp không gian ảnh (image-space). Phương pháp không gian đối tượng so sánh các đối tượng, cũng như các thành của chúng với mỗi cái khác để xác định xem các mặt và đường nào sẽ được đánh nhãn là không nhìn thấy được. Trong một thuật toán không gian ảnh, tính chất nhìn thấy được của một điểm được quyết định bởi điểm ở vị trí pixel trên mặt phẳng chiếu. Hầu hết các thuật toán khử mặt khuất dùng phương pháp không gian ảnh, tuy nhiên các phương pháp không gian đối tượng vẫn có thể được dùng một cách hiệu quả cho một số trường hợp. Các thuật toán khử đường khuất hầu hết dùng phương pháp không gian đối tượng, dù rằng nhiều thuật toán khử mặt khuất không gian ảnh có thể dễ dàng được chỉnh sửa cho việc khử đường khuất.

Dù có có sự khác nhau lớn trong tiếp cận cơ bản được cần bởi các thuật toán khử mặt khuất và đường khuất, nhưng hầu hết chúng đều dùng đến phương pháp sắp xếp (sorting) và cố kết (coherence) để cải thiện sự thực hiện. Sắp xếp sẽ mang đến sự dễ dàng cho việc so sánh độ sâu sau này, điều này được thực hiện bằng cách sắp xếp

thứ tự các đường, mặt, và các đối tượng trong ảnh dựa vào khoảng cách từ chúng đến mặt phẳng quan sát. Phương pháp cố kết được dùng để thu được thuận lợi của sự cân đối trong ảnh. Một đường quét riêng lẻ có thể được dùng để chứa đựng các giá trị về độ sáng của các pixel, và các mẫu đường quét (scan-line patterns) thường thay đổi ít từ đường này đến đường kế tiếp. Các khung nối kết động chứa các thay đổi chỉ trong vùng lân cận của các đối tượng di chuyển. Và các mối quan hệ cố định thường được xây dựng giữa các đối tượng và các mặt trong ảnh.

Một phương pháp không gian đối tượng đơn giản để xác định các mặt sau (back faces) đối tượng là dựa vào các phương trình mặt:

$Ax + By + Cz + D = 0$ (7-1)

Bất kỳ điểm (x', y', z') trên hệ tọa độ bàn tay trái sẽ ở "phía trong" mặt này nếu nó thỏa bất phương trình:

$Ax' + By' + Cz' + D < 0$ (7-2)

Nếu điểm (x', y', z') là vị trí quan sát (viewing position), khi đó bất kỳ mặt phẳng nào làm cho bất phương trình 7-2 đúng phải là một mặt ở đằng sau. Tức là, nó

là mặt ta không thể nhìn thấy từ vị trí quan sát.
Chúng ta
có thể thực hiện
một cách kiểm
tra mặt đằng sau
đơn giản hơn
bằng cách nhìn
ở pháp vector
(normal vector)
của mặt có
phương trình
7-1. Pháp vector
này có tọa độ Descartes (A, B, C). Trong hệ tọa độ bàn tay phải với hướng quan sát cùng chiều với trục zv âm (xem hình 7-1), pháp vector có tham số C song song với hướng quan sát. Nếu C<0, pháp vector chỉ ra xa khỏi vị trí quan sát, và mặt phải là mặt ở đằng sau.
Hình 7-1
Một mặt phẳng với tham số C < 0 trong hệ quan sát bàn tay phải được xác
như mặt ở đằng sau khi hướng quan sát cùng chiều với trục zv âm.
định
yv
xv
zv
Điểm quan
sát
Hướng quan
sát
N= (A, B, C)
•
Trang 136 Chương 7: Khử các mặt kuất và đường khuất
Các phương pháp tương tự có thể được dùng trong các gói đồ họa, nơi sử dụng hệ quan sát bày tay trái. Trong các gói đồ họa này, các tham số A, B, C, và D có thể được tính từ tọa độ các đỉnh được xét theo chiều kim đồng hồ (thay vì hướng ngược chiều kim đồng hồ được dùng trong hệ tọa độ bàn tay phải). Bất phương trình 7-2 sau đó cho một kiểm tra hợp lệ đối với các điểm nằm phía trong. Cũng như vậy, các mặt ở đằng sau có các pháp vector chỉ ra xa khỏi vị trí quan sát và được xác định bởi C>0 khi hướng quan sát cùng hướng với trục zv dương (xem hình 7-2). Trong tất cả các thảo luận sau này trong chương, chúng ta giả sử rằng hệ quan sát bàn tay trái được dùng.

yv
zv
Điểm
quan sát
Hướng
quan sát

xv
N= (A, B, C)
•
Hình 7-2
Trong hệ quan sát bàn tay trái, khi hướng quan sát cùng chiều với trục zv dương, một mặt ở đằng sau là mặt với tham số C>0.

Bằng việc kiểm tra tham số C ở mỗi mặt của đối tượng, ta có thể xác định được ngay tất cả các mặt ở đằng sau. Đối với một khối đa diện lồi đơn lẽ, như hình kim tự tháp trong hình 7-1, việc kiểm tra này xác định tất cả các mặt bị che khuất trên đối tượng, bởi vì mỗi mặt thì là hoàn toàn được nhìn thấy hoặc hoàn toàn bị che khuất. Đối với các đối tượng khác, các kiểm tra phức tạp hơn cần được thực hiện để xác định xem các mặt là bị che khuất hoàn toàn hay chỉ bị che khuất một phần (xem hình 7-3). Tương tự, chúng ta cần xác định xem các đối tượng là có một phần hay toàn bộ bị che khuất bởi các đối tượng khác. Một cách tổng quát, việc khử mặt khuất sẽ loại bỏ khoảng một nữa số mặt trong một ảnh khi thực hiện các phép kiểm tra tính nhìn thấy được sau này.
Hình 7-3
Ảnh một đối tượng với một mặt b
che khuất một phần
ị

Trang 137 Chương 7: Khử các mặt kuất và đường khuất
7.3. Phương pháp dùng vùng đệm độ sâu (Depth-Buffer Method)
Một tiếp cận không gian ảnh được dùng phổ biến để khử mặt khuất là phương pháp vùng đệm độ sâu, còn được gọi là phương pháp z-buffer. Một cách cơ bản, thuật toán này kiểm tra tính nhìn thấy được của các mặt mỗi lần một điểm. Với mỗi vị trí pixel (x,y) trên mặt phẳng quan sát, mặt nào có giá trị tọa độ z nhỏ nhất ở vị trí pixel đó thì nhìn thấy được. Hình 7-4 trình bày ba mặt có độ sâu khác nhau, với sự quan tâm đến vị trí (x, y) trong hệ quan sát bàn tay trái. Mặt S1 có giá trị z nhỏ nhất ở vị trí này vì vậy giá trị độ sáng ở (x, y) được lưu.
Hai vùng đệm được cần để cài đặt phương pháp này. Một vùng đệm độ sâu (depth buffer) được dùng để lưu trữ các giá trị z cho mỗi vị trí (x, y) của các mặt được so sánh. Vùng đệm thứ hai là vùng đệm làm tươi (refresh buffer) (hay còn gọi là vùng

đệm khung), lưu giữ các giá trị độ sáng cho mỗi vị trí (x, y).
Phương pháp này có thể được thực hiện hiệu quả trong các hệ tọa độ chuẩn, với các giá trị độ sâu thay đổi từ 0 đến 1. Giả sử rằng một không gian chiếu (projection volume) được ánh xạ vào một không gian quan sát hình hộp chuẩn, ánh xạ của mỗi mặt lên mặt phẳng quan sát là một phép chiếu trực giao. Độ sâu của các điểm trên bề mặt của một đa giác được tính từ phương trình mặt phẳng. Ban đầu, tất cả các vị trí trong vùng đệm độ sâu được đặt giá trị 1 (độ sâu lớn nhất), và vùng đệm làm tươi được khởi tạo giá trị của độ sáng nền. Mỗi mặt (đã được lập danh sách trong các bảng đa giác (polygon tables)) sau đó được xử lý. Mỗi lần một đường quét (scane line), tính độ sâu, hoặc giá trị z, ở mỗi vị trí (x, y). Giá trị z vừa được tính xong sẽ được so sánh với các giá trị lưu trữ trước đó trong vùng đệm độ sâu ở vị trí đó. Nếu giá trị z vừa được tính xong nhỏ hơn các giá trị trước đó, giá trị z mới sẽ được lưu, và độ sáng của mặt ở vị trí đó cũng được cập nhật lại vào vị trí tương ứng trong vùng đệm làm tươi.

Hình 7-4
Ở ví trí (x, y), mặt S1 có giá độ sâu nhỏ nhất và
thế được nhìn thấy ở ví trí đó
vì
•
•
•
•
x
yv
zv v
S3
S2
S1

Chúng ta có thể tổng kết các bước của thuật toán vùng đệm độ sâu như sau:
1. Khởi tạo vùng đệm độ sâu và vùng đệm làm tươi để với tất cả các vị trí (x,y), depth(x, y) =1 và refresh(x, y) = background.
2. Đối với mỗi vị trí trên mỗi mặt, so sánh các giá trị độ sâu với các giá trị độ sâu được lưu trước đó trong vùng đệm độ sâu để xác định tính chất nhìn thấy được.
a. Tính giá trị z cho mỗi vị trí (x, y) trên mặt.
b. Nếu z < depth(x, y) thì đặt lại depth(x, y)= z và refresh(x, y) = i , với i là giá trị độ sáng trên mặt ở vị trí (x, y).
Trong bước cuối cùng, nếu z không nhỏ hơn giá trị trong vùng đệm độ sâu ở vị trí đó, điểm không được nhìn thấy. Khi quá trình này được hoàn thành cho tất cả các mặt, vùng đệm độ sâu chứa các giá trị z của các mặt nhìn thấy được và vùng đệm làm tươi

chỉ chứa các giá trị độ sáng của các mặt nhìn thấy được đó.
Các giá trị độ sâu cho một vị trí (x, y) được tính từ phương trình của mỗi mặt:
C
DByAx
z
– ––
= (7-3)
Với mỗi đường quét bất kỳ (xem hình 7-5), các tọa độ x trên cùng đường quét sai khác nhau 1, và các giá trị y giữa hai đường quét cũng sai khác nhau 1. Nếu độ sâu của vị trí (x,y) được xác định là z, khi đó độ sâu z' của vị trí kế tiếp (x+1, y) dọc theo theo đường quét có được từ phương trình 13-3 như
sau:
Hình 7-5
Từ vị trí (x, y) trên một đường quét, vị
trí kế tiếp qua phải có tọa độ (x+1, y), và
vị trí liền ngay bên dưới trên dòng kế
tiếp có tọa độ (x, y-1)
y
x
y
y-1
x x+1
•
• •
C
DByxA
z
– – +–
=
)1(
'
hoặc

C
A
zz –= '
(7-4)

Tỷ số A/C không đổi với mỗi mặt, vì vậy giá trị độ của điểm kế tiếp trên cùng đường quét có được từ giá trị trước đó với một phép trừ.
Chúng ta thu được các giá trị độ sâu giữa các đường quét theo cách tương tự.
Một lần nữa giả sử rằng vị trí (x, y) có độ sâu z. Khi đó ở vị trí (x, y-1) trên đường quét ngay bên dưới, giá trị độ sâu được tính từ phương trình mặt phẳng như sau:

C
DyBAx
z
– – ––
=
)1(
"
hoặc (7-5)
C
B
zz += "
ở đây cần một phép cộng hằng B/C với giá trị độ sâu z trước đó.
Phương pháp vùng đệm độ sâu thì dễ dàng để cài đặt, và nó không cần sắp xếp các mặt trong ảnh. Nhưng nó cần đến một vùng đệm thứ hai đó là vùng đệm làm tươi. Một hệ thống với độ phân giải 1024 x 1024 có thể cần hơn một triệu vị trí trong vùng đệm độ sâu, với mỗi vị trí cần đủ bit để lưu giữ các tọa độ z tăng. Một cách để giảm bớt không gian lưu giữ cần thiết là tại mỗi thời điểm chỉ xử một phần của ảnh, dùng một vùng độ sâu nhỏ hơn. Sau mỗi phần ảnh được xử lý xong, vùng đệm được dùng lại cho phần kế tiếp.

7.4. Phương pháp đường quét (Scan-Line Method)

Phương pháp không gian ảnh để khử các mặt bị che khuất này là sự mở rộng của thuật toán scan-line để tô phần bên trong của một đa giác. Thay vì chỉ tô một mặt, bây giờ chúng ta xử lý với nhiều mặt. Khi mỗi đường quét được xử lý, tất cả các mặt đa giác cắt đường quét đó sẽ được kiểm tra để xác định xem mặt nào nhìn thấy được. Ở mỗi vị trí trên cùng đường quét các tính toán độ sâu được thực hiện cho mỗi mặt để xác định mặt gần mặt phẳng quan sát nhất. Khi mặt mặt nhìn thấy được được xác định, giá trị độ sáng cho vị trí đó được nhập vào vùng đệm làm tươi (refresh buffer).
Một đa giác trong không gian ba chiều được cài đặt có thể bao gồm cả hai bảng: bảng các cạnh (edge table) và bảng đa giác (polygon table), tương tự như trong hình 7-23 ở cuối chương này. Bảng các cạnh (edge table) chứa tọa độ các đỉnh đầu mút của mỗi cạnh, đảo hệ số góc của mỗi đường thẳng qua cạnh, và các chỉ diểm (pointer) đến

bảng đa giác để xác định mặt nào chứa mỗi cạnh. Bảng đa giác chứa các hệ số của phương trình mỗi mặt, thông tin về độ sáng cho các mặt, và có thể chỉ đến bảng các cạnh.
Để dễ dàng nghiên cứu các mặt cắt một đường quét được cho, chúng ta cài đặt một danh sách động chứa các cạnh lấy thông tin trong bảng cạnh. Danh sách động này sẽ chỉ chứa các cạnh cắt đường quét hiện hành, được sắp xếp theo thứ thự x tăng. Và, chúng ta định nghĩa một cờ (flag) cho mỗi mặt, cờ này được đặt là on hay off để chỉ ra mỗi vị trí nằm dọc trên đường quét là nằm trong hay nằm ngoài mặt. Các đường quét được xử lý từ trái sang phải. Ở biên bên trái nhất của một mặt, cờ của mặt là on; và ở biên bên phải nhất cờ là off.
Hình 7-6 minh họa phương pháp scan-line để xác định vị trí các phần nhìn thấy được dọc theo một đường quét. Danh sách động cho đường quét 1 (scan line 1) lấy thông tin từ bảng các cạnh đối với các cạnh AB, BC, HE, và FG. Đối với các vị trí dọc theo đường quét này giữa các cạnh AB và BC, chỉ cờ mặt S1 là on. Do đó, không phép

tính độ sâu nào là cần thiết, và thông tin độ sáng của mặt S1 được lấy từ bảng đa giác để nhập vào vùng làm tươi. Tương tự, các cạnh HE và FG, chỉ cờ cho mặt S2 là on. Không vị trí nào khác dọc theo đường quét 1 cắt các mặt, vì vậy các giá trị độ sáng trong các vùng khác được đặt là độ sáng nền.Độ sáng nền có thể được nạp vào trong vùng đệm trong một thủ tục khởi tạo.

Đường quét
Đường quét
Đường quét
yv

1
2
3
xv
S1 S2
A
B
C
D
E
F
H
G
Hình 7-6
Các đường quét cắt hình chiếu của hai mặt S1 và S2 trên mặt phẳng chiếu. Các đường nét đứt chỉ ra rằng đó là biên của mặt bị che khuất.

Danh sách động cho đường quét 2 và 3 trong hình 7-6 chứa các cạnh DA, HE, BC, và FG. Dọc theo đường quét 2 từ cạnh DA đến cạnh EH, chỉ cờ của mặt S1 là on.

Nhưng giữa HE và BC, các cờ cho cả hai mặt là on. Trong đoạn này, các tính toán độ về độ sâu phải được thực hiện bằng cách dùng tham số mặt của các mặt. Trong ví dụ

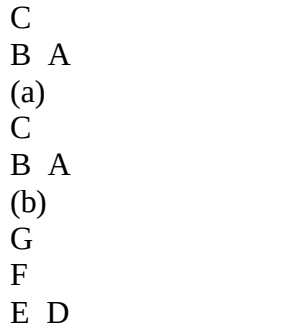
này, độ sâu của mặt S1 được được giả thiết là nhỏ hơn của mặt S2, vì vậy độ sáng của mặt S1 được nạp vào trong vùng đệm làm tươi đến khi biên BC được gặp. Sau đó cờ của mặt S1 trở thành off, và độ sáng của mặt S2 được lưu cho đến cạnh FG được đi qua.

Chúng ta có thể tận dụng các thuận lợi có được từ quan hệ cố kết dọc theo các đường quét khi chúng ta đi từ đường này đến đường kế tiếp. Trong hình 7-6, đường quét 3 có danh sách động giống như của dòng 2. Bởi vì không có thay đổi nào xảy ra tại các giao điểm đường, ta không cần tính lại độ sâu giữa các cạnh HE và BC. Hai mặt phải có hướng tương tự như được xác định trên đường quét 2, vì vậy độ sáng cho mặt S1 không cần nhập lại.

Dù có bao nhiêu mặt được xếp chồng lên nhau, ta cũng có thể dùng phương pháp scan-line này. Các cờ cho các mặt được đặt để chỉ rõ xem một vị trí là bên trong hay bên ngoài, và các tính toán về độ sâu được thực hiện khi các mặt xếp chồng lên nhau. Trong vài trường hợp, các mặt có thể che khuất nhau một cách luân phiên (xem hình 7-7). Khi các phương pháp cố kết được dùng, ta cần cẩn thận để lưu vết phần nào của mặt là thấy được trên mỗi đường quét. Một cách để xử lý trường hợp này là phân chia các mặt. Cụ thể, mặt ABC trong hình 7-8 có thể được chia làm ba mặt ABED, DEGF, và CFG. Mỗi mặt nhỏ có thể được xét như một mặt riêng biệt, để mà không có hai mặt nào là bị che khuất và nhìn thấy một cách luân phiên.

Đường cắt
(a)
Đường cắt
(b)
Hình 7-7
Các ví dụ về hướng các mặt
che khuất lẫn nhau.

7.5. Phương pháp sắp xếp theo độ sâu (Depth- Sorting Method)



Hình 7-8

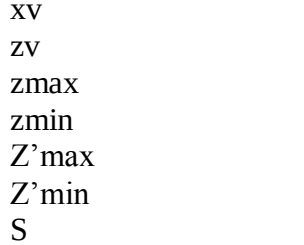
Chia một mặt ra làm nhiều mặt để tránh các vấn đề nhìn thấy và không nhìn thấy luân phiên giữa hai mặt.

Ta có thể sử dụng cả hai phương pháp không gian ảnh và không gian đối tượng trong một thuật toán khử mặt khuất. Phương pháp sắp xếp theo độ sâu (depth-sorting method) là một sự nối kết của hai tiếp cận trên, nó thực hiện các công việc cơ bản sau:

1. Các mặt được sắp theo thứ tự giảm dần của độ sâu.
2. Các mặt được vẽ theo thứ tự từ mặt có độ sâu lớn nhất đến mặt có độ sâu nhỏ nhất (vẽ từ mặt xa nhất đến mặt gần nhất).

Các các thao tác sắp xếp được thực hiện trong không gian đối tượng, còn sự chuyển đổi dòng quét (scan conversion) được thực hiện trong không gian ảnh.

Phương pháp giải quyết vấn đề mặt khuất này đôi khi còn được gọi là thuật toán của họa sĩ (painter's algorithm). Để tạo ra một bức sơn dầu (oil painting), đầu tiên họa sĩ sơn các độ sáng nền. Kế tiếp, các đối tượng ở xa nhất được thêm vào. Sau cùng, các đối tượng ở gần được vẽ phủ lên các đối tượng ở xa đó. Mỗi lớp vẽ sau phủ lên lớp vẽ trước đó. Dùng kỹ thuật tương tự, chúng ta đầu tiên sắp xếp các mặt theo khoảng cách từ chúng đến mặt quan sát. Các giá trị độ sáng của mặt xa nhất được nhập vào vùng



S’
Hình 7-9
Hai mặt không có sự nạp chồng độ
sâu.

vùng đệm làm tươi. Với mỗi mặt kế tiếp (xét theo thứ tự độ sâu giảm dần), ta “sơn” các độ sáng của mặt lên vùng đệm làm tươi (phủ lên các độ sáng của mặt được xử lý trước đó).
Việc sơn các mặt đa giác lên vùng đệm làm tươi dựa theo độ sâu được thực hiện trong vài bước. Đầu tiên, các mặt được sắp xếp dựa vào giá trị z lớn nhất của mỗi mặt. Mặt với độ sâu lớn nhất (gọi là S) sau đó được so sánh với các mặt còn lại trong danh sách để xác định xem có bất kỳ sự chồng độ sâu nào không (nằm chồng lên nhau). Nếu không có sự chồng độ sâu nào xảy ra, S được vẽ ra (vẽ ra theo từng đường quét).
Trong hình 7-9 trình bày hai mặt không có sự chồng độ sâu (hai mặt không nằm chồng nhau), hình chiếu của chúng lên mặt phẳng xz. Xử lý này sau đó được lặp lại cho mặt kế tiếp trong danh sách. Khi không có sự chồng độ sâu nào xảy ra, mỗi mặt sẽ được xử lý theo thứ tự độ sâu đó cho đến khi tất cả đều được quét qua. Nếu có một sự chồng độ sâu được phát hiện ở bất kỳ điểm nào trong danh sách, ta cần làm vài so sánh bổ sung để xác định xem mặt nào nên được sắp xếp lại.
Với mỗi mặt nằm chồng với S, ta thực hiện các phép kiểm tra sau. Chỉ cần một trong số các phép kiểm tra này là đúng (true), ta không cần sắp lại vị trí mặt đó. Các phép kiểm tra được lập danh sách theo mức độ khó tăng dần:
1. Trên mặt phẳng xy, các hình chữ nhật bao quanh hai mặt không chồng lên nhau.

xv
zv
xmax xmin x’max x’min
Hình 7-10
Hai mặt không có sự chồng độ sâu
theo hướng x.
S’
S
2. Mặt S thì ở “phía ngoài” mặt nằm chồng, so sánh dựa vào mặt phẳng quan sát.
3. Mặt nằm chồng thì ở “phía trong ” mặt S, so sánh dựa vào mặt phẳng quan sát.
4. Các hình chiếu của hai mặt lên mặt phẳng quan sát không nằm chồng lên nhau.

Vừa khi một phép kiểm tra được phát hiện là đúng cho một mặt nằm chồng, ta

biết rằng mặt không nằm phía sau S. Vì vậy ta chuyển đến mặt chồng S kế tiếp. Nếu tất cả các mặt mặt nằm chồng vượt qua được ít nhất một trong các phép kiểm tra trên, ta không phải sắp xếp và S có thể được vẽ ra.

Phép kiểm tra 1 được thực hiện trong hai phần: Chúng ta kiểm tra sự nằm chồng theo hướng x, sau đó kiểm tra nằm chồng theo hướng y. Nếu cái nào trong hai hướng này được phát hiện là không có nằm chồng, hai mặt phẳng không che khuất nhau. Một ví dụ về hai mặt có nằm chồng theo hướng z nhưng không chồng theo hướng x được cho trong hình 7-10.

Chúng ta có thể thực hiện phép kiểm tra 2 bằng cách thế tọa độ tất cả các đỉnh của S vào phương trình mặt của mặt nằm chồng và kiểm tra dấu của kết quả. Giả sử rằng mặt nằm chồng có hệ số A', B', C', và D'. Nếu $A'x + B'y + C'z + D' > 0$ với mỗi đỉnh có tọa độ (x, y, z) của S, mặt S sẽ ở "phía ngoài" mặt nằm chồng S' (xem hình 7-11). Như được đề cập trước đây, các hệ số A', B', C', và D' phải được xác định trước để pháp vector của mặt nằm chồng S' chỉ ra xa khỏi mặt phẳng quan sát.

Hình 7-11
Mặt S hoàn toàn ở "phía ngoài" mặt nằm chồng S' khi nhìn từ mặt quan sát xy.
xv
zv
S
S'

Hình 7-12
Mặt nằm chồng S' hoàn toàn ở "phía trong" mặt S, khi nhìn từ mặt quan sát xy.
xv
zv
S
S'

Phép kiểm tra 3 được thực hiện dùng các hệ số A, B, C, và D của mặt S. Nếu tọa độ (x, y, z) của tất cả các đỉnh của mặt nằm chồng S' thỏa điều kiện $Ax + By + Cz$

+ D < 0, khi đó mặt nằm chồng S’ sẽ ở “phía trong” mặt S (cung cấp pháp vector của mặt S hướng ra xa mặt phẳng quang sát). Hình 7-12 trình bày một mặt nằm chồng S’ thỏa phép kiểm tra này. Trong ví dụ này, mặt S thì không ở “phía ngoài” S’ (phép kiểm tra 2 không đúng).
Trang 145 Chương 7: Khử các mặt kuất và đường khuất
Nếu tất cả các phép kiểm tra từ 1 đến 3 đều thất bại (sai), chúng ta thử đến phép kiểm tra 4 bằng cách kiểm tra sự cắt nhau giữa các cạnh biên của hai mặt, dùng các phương trình đường thẳng trong mặt xy. Như được minh họa trong hình 7-13, hai mặt có thể cắt hoặc không cắt nhau thậm chí khi các không gian bao quanh chồng nhau theo các hướng x, y, và z (xem hình 7-13).

Các mặt không cắt nhau
Hình 7-13
Hai mặt với các
chữ nhật nằm ch
biên
ồng
. nhau trong mặt xy
Các mặt cắt nhau

Nếu tất cả bốn phép kiểm tra trên đều thất bại với một mặt nằm chồng cụ thể S’, ta đổi chỗ hai mặt S và S’ cho nhau trong danh sách đã được sắp. Một ví dụ của hai mặt sẽ được sắp xếp lại với thủ tục này được cho trong hình 7-14. Tuy nhiên, ta vẫn không biết chắc rằng ta đã tìm gặp mặt xa nhất tính từ mặt phẳng quan sát chưa. Hình 7-15 minh họa một trường hợp mà tại đó đầu tiên chúng ta đổi chỗ S và S’’ với nhau.
Nhưng vì S’’
che khuất một phần của S’ (nhìn lên từ mặt xy), chúng ta cần đổi chỗ S’’ và S’ với nhau để có ba mặt được sắp hợp lý theo độ sâu. Do đó, chúng ta cần lặp lại quá trình kiểm tra cho mỗi mặt, cái vừa được sắp lại trong danh sách.

Hình 7-14
Mặt S có độ sâu z lớn hơn
xv

zv
S S’
Hình 7-15
Ba mặt ban đầu đã được sắp theo thứ tự độ sâu z :

S,
xv
zv
S’’ S’
S

Thuật toán vừa được phác thảo có thể đi vào một vòng lặp vô tận nếu hai hay nhiều mặt che khuất lẫn nhau một cách luân phiên như trong hình 7-7 (xem hình 7-7).

Trong các trường hợp như thế, thuật toán sẽ lặp đi lặp lại không ngừng việc đổi chỗ vị trí của các mặt nằm chồng nhau. Để tránh các vòng lặp như thế, chúng ta có thể đặt cờ trạng thái cho mặt nào vừa được sắp đến vị trí xa hơn để nó không bị di chuyển lại nữa. Nếu có một sự cố gắng được làm để đổi chỗ các mặt lần thứ hai, ta chia nó ra làm hai phần tại đường cắt (đường giao) của hai mặt. Mặt ban đầu sau đó được thay thế bởi hai mặt mới, và ta lại tiếp tục quá trình xử lý như trước đây.

7.6. Phương pháp phân chia vùng (Area- Subdivision Method)

Kỹ thuật khử mặt khuất này thì hiệu quả cho phương pháp không gian ảnh, nhưng các phương pháp không gian đối tượng có thể được dùng để thực hiện việc sắp xếp các mặt theo độ sâu. Phương pháp phân chia vùng tận dụng các thuận lợi của các vùng cố kết trong ảnh bằng cách xác định các vùng quan sát này để tách chúng làm nhiều phần nhỏ, mỗi phần được xem như một mặt đơn lẻ. Chúng ta áp dụng phương pháp này bằng cách phân chia thành công toàn bộ vùng quan sát thành các hình chữ nhật càng lúc càng nhỏ cho đến khi mỗi vùng nhỏ là hình chiếu của một phần của một mặt đơn lẻ nhìn thấy được, hoặc cho đến khi không thể tiếp tục phân chia.
Hình 7-16
Các phần chia được thực hiện thành công với phép chia 2.
Để thực hiện phương pháp này, ta cần xây dựng các phép kiểm tra để xác định nhanh chóng vùng là một phần của một mặt đơn lẻ hoặc cho ta biết vùng thì quá phức tạp để

phân tích bình thường. Bắt đầu với cái nhìn tổng thể, ta áp dụng các phép kiểm tra để xác định xem có nên phân chia toàn bộ vùng thành các hình chữ nhật nhỏ hơn không. Nếu các phép kiểm tra chỉ ra rằng mặt quan sát đủ phức tạp, ta phân chia nó. Kế tiếp, chúng ta áp dụng các phép kiểm tra đến mỗi vùng nhỏ hơn, chia nhỏ những vùng này nếu các phép kiểm tra xác định rằng tính nhìn thấy được của một mặt đơn là vẫn chưa chắc chắn. Chúng ta tiếp tục quá trình này đến khi các phần phân chia là dễ dàng được phân tích như là một mặt đơn lẻ hoặc đến khi chúng được thu giảm kích thước thành một pixel.

Trang 147 Chương 7: Khử các mặt kuất và đường khuất

Một cách để phân chia một vùng thành công là chia kích thước nó ra làm 2, như trong hình 7-16. Một vùng quan sát với độ phân giải 1024x1024 có thể được chia 10 lần trước khi một phần chia giảm thành 1 điểm.

Các phép kiểm tra để xác định tính nhìn thấy được của một mặt đơn trong phạm vi vùng chỉ định được thực hiện bằng cách so sánh các mặt với biên của vùng. Có bốn khả năng có thể xảy ra khi xem xét mối quan hệ giữa một mặt với biên vùng chỉ định. Ta có thể mô tả đặc điểm của các quan hệ này theo các cách sau (xem hình 7-17):

Mặt bao quanh (surrounding surface) là mặt hoàn toàn bao quanh một vùng.

Mặt nằm chồng (overlapping surface) là mặt có một phần nằm trong và một phần nằm ngoài vùng.

Mặt bên trong (inside surface) là mặt hoàn toàn nằm bên trong vùng.

Mặt bên ngoài (outside surface) là mặt hoàn toàn nằm bên ngoài vùng.

Hình 7-17
Các quan hệ
có thể xảy ra
giữa các mặt
đa giác và một
vùng chữ
nhật.

Mặt bao quanh Mặt nằm chồng Mặt bên trong Mặt bên ngoài

Các phép kiểm tra để xác định tính nhìn thấy được của mặt trong phạm vị một vùng có thể được đề cập giới hạn trong bốn loại này. Không có sự phân chia nào thêm nữa cho một vùng nếu một trong các điều kiện sau là đúng (true):

1. Tất cả các mặt nằm bên ngoài vùng.
2. Chỉ một mặt bên trong, mặt nằm chồng hoặc mặt bao quanh ở trong vùng.
3. Một mặt bao quanh che khuất tất cả các mặt khác trong phạm vi các biên của vùng.

Kiểm tra 1 có thể được thực hiện bằng cách kiểm tra các biên chữ nhật bao quanh các mặt với biên của vùng. Kiểm tra 2 cũng có thể dùng các biên chữ nhật trong mặt

Trang 148 Chương 7: Khử các mặt kuất và đường khuất

xy để xác định mặt nằm trong. Với những kiểu mặt khác, các biên chữ nhật có thể được dùng như một bước kiểm tra ban đầu. Nếu một biên chữ nhật cắt vùng theo cách nào đó, các kiểm tra tiếp theo mới được thực hiện để xác định xem mặt là: mặt bao quanh, mặt nằm chồng, hay mặt bên ngoài. Nếu được xác định là mặt bên trong, mặt nằm chồng, hay mặt bao quanh, các giá trị độ sáng pixel của nó được chuyển đến vùng

thích hợp trong vùng đệm khung.
Một phương pháp để thực hiện bước 3 là sắp xếp các mặt dựa theo độ sâu nhỏ nhất của chúng. Sau đó, với mỗi mặt bao quanh, ta đi tính giá trị z lớn nhất trong vùng được xem xét. Nếu giá trị lớn nhất z của một mặt nào (trong số các mặt mặt bao quanh) nhỏ hơn giá trị z nhỏ nhất của các mặt còn lại trong vùng, kiểm tra 3 thỏa. Hình 7-18 trình bày một ví dụ chứa các điều kiện của phương pháp này.

Hình 7-18
Một mặt bao quanh với độ sâu lớn nhất của zmax (xét tron
vùng quan sát) che khuất tất cả các mặt mà độ sâu nhỏ nh
xmin của chúng lớn hơn zmax.
g
ất
xv
zv
Vùng được xem xét
z’’min
z’min
zmax

Một phương pháp khác để thực hiện kiểm tra 3 mà không cần đến sắp xếp độ sâu là dùng các phương trình mặt phẳng để tính các giá trị z ở bốn đỉnh của vùng cho tất cả các mặt bao quanh, mặt nằm chồng, hay mặt bên trong. Nếu các giá trị z của một trong số các mặt bao quanh mà nhỏ hơn các giá trị z của các mặt còn lại, kiểm tra 3 đúng. Sau đó vùng có thể được tô với các giá trị độ sáng của mặt bao quanh.
Trong vài trường hợp, cả hai phương pháp cho kiểm tra 3 trên sẽ thất bại để xác định đúng một mặt bao quanh che khuất tất cả các mặt còn lại khác. Việc kiểm tra thêm nữa sẽ được thực hiện để xác định mặt đơn che phủ vùng, tuy nhiên, thuật toán sẽ

nhanh hơn nếu ta phân chia vùng hơn là tiếp tục làm các kiểm tra phức tạp. Khi các mặt bên ngoài và mặt bao quanh vừa được xác định cho một vùng, chúng nó sẽ còn lại các mặt bên ngoài và bao quanh cho tất cả các phần phân chia của vùng. Hơn nữa, vài mặt bên trong và mặt nằm chồng có thể đang chờ để bị loại bỏ khi quá trình phân chia tiếp tục, để các vùng trở nên dễ dàng hơn cho phân tích. Trong trường hợp đã đi đến giới hạn, kích thước vùng chia chỉ còn là 1 pixel, ta đơn giản đi tính độ sâu của mỗi mặt có liên quan ở điểm đó và chuyển giá trị độ sáng của mặt gần nhất vào vùng đệm khung.

Hình 7-19
Vùng A được phân c
thành A1 và A2 bằn

hia
g

cách
t S trên dùng biên của mặ
mặt phẳng chiếu.
zv
xv
Vùng A
A2
yv
S
A1
•
•
•

Như một thay đổi lên quá trình phân chia cơ bản, ta có thể phân chia các vùng dọc theo biên của mặt thay vì chia chúng làm 2. Nếu các mặt vừa được sắp theo độ sâu nhỏ nhất, ta có thể dùng mặt có giá trị z nhỏ nhất để phân chia một vùng được cho. Hình 7-19 minh họa phương pháp này để phân chia các vùng. Hình chiếu của biên mặt S được dùng để phân chia vùng ban đầu thành các phần A1 và A2. Mặt S sau đó trở thành mặt bao quanh của A1 và các phép kiểm tra 2 và 3 có thể được áp dụng để xác định xem việc phân chia thêm nữa có cần thiết không. Trong trường hợp tổng quát, sự phân chia ít hơn được cần dùng tiếp cận này, tuy nhiên nhiều xử lý thêm nữa sẽ được cần để chia vùng và phân tích mối liên hệ giữa các mặt với các biên vùng chia.

7.7. Các phương pháp Octree (Octree Methods)

Khi biểu diễn octree được dùng cho các không gian quan sát, việc khử các mặt khuất được thực hiện bằng cách chiếu các nút octree lên mặt quan sát theo thứ tự từ

trước ra sau. Trong hình 7-20, mặt phía trước của vùng không gian (mặt hướng về phía

người quan sát) được hình thành với các phần tám (octant) 0, 1, 2, 3. Mặt trước của các octant này được nhìn thấy bởi người quan sát. Bất kỳ mặt nào hướng về phía sau của các octant phía trước này hoặc các octant ở đằng sau (4, 5, 6, và 7) có thể bị che khuất bởi các mặt phía trước.

Các mặt phía sau bị loại bỏ, với hướng quan sát như trong hình 7-20, bằng cách xử lý các phần tử dữ liệu tại các nút octree theo thứ tự 0, 1, 2, 3, 4, 5, 6, 7. Điều này tạo ra kết quả du hành theo độ sâu của octree, để các octant 0, 1, 2, và 3 của toàn vùng được viếng thăm trước các octant 4, 5, 6, và 7. Tương tự, bốn octant con trước của octant 0 sẽ được viếng thăm trước bốn octant con phía sau. Cuộc du hành của octree sẽ tiếp tục theo thứ tự này cho mỗi phần chia octant.

0
1
2
3
4
5
7
6

Các Octant được đánh số của một vùng
Hướng qua
Hình 7-20
Các đối tượng trong các octant 0, 1, 2, và 3 che khuất các octant phía sau (4, 5, 6, 7) khi hướ quan sát như trong hình.
n sát

ng

Khi giá trị màu được gặp tại một nút của octree, vùng pixel trong vùng đệm khung tương ứng với nút này được gán giá trị màu đó chỉ nếu không giá trị nào được lưu trước đó trong vùng này. Không gì được nạp nếu một vùng trống rỗng. Bất kỳ nút nào được phát hiện là bị che khuất hoàn toàn thì sẽ bị loại bỏ khỏi các xử lý trong tương lai, để các các cây con của nó không được

truy cập vào.
1
2
3
4
5
7
6
0
Các octant trong không gian
Hình 7-21
Sự phân chia octant cho một vùng
không gian và mặt các phần tư tương
ứng.
1
2 3
Các quang cảnh khác nhau của đối tượng
được biểu diễn như octree có thể đạt được bằng
cách áp dụng các phép biến đổi đến sự biểu diễn octree để làm thay đổi đối tượng theo hướng quan sát. Ta giả sử rằng biểu diễn octree luôn được xây dựng sao cho các octant 0, 1, 2, và 3 của một vùng hình thành nên mặt phía trước (xem hình 7-20).
0
Các quadrant (góc 1/4) trong
mặt phẳng quan sát
Trang 151 Chương 7: Khử các mặt kuất và đường khuất
Một phương pháp để hiển thị một octree từ trước ra sau là đầu tiên ánh xạ octree vào một quadtree của các vùng nhìn thấy được bằng cách duyệt qua các nút của octree từ trước ra sau trong một quá trình đệ quy. Sau đó biểu diễn quadtree của các mặt nhìn thấy được được nạp vào trong vùng đệm khung. Hình 7-21 mô tả các octant trong một vùng không gian và các quadtree tương ứng trên mặt phẳng quan sát. Các phần tạo thành quadtree 0 lấy từ octant 0 và 4. Các giá trị màu trong góc phần tư 1 (quadrant 1) có được từ các mặt trong octant 1 và 5, và các giá trị trong mỗi của hai quadrant còn lại được sinh ra từ cặp octant thẳng hàng với mỗi quadrant này.
Việc xử lý đệ quy của các nút octree được trình bày trong thủ tục convert_oct_to_quad, nơi nhận vào một mô tả octree và tạo ra các biểu diễn quadtree cho các mặt nhìn thấy được trong vùng. Trong hầu hết các trường hợp, cả octant phía trước và phía sau phải được xem xét để xác định màu đúng cho một quadrant. Tuy nhiên, ta có thể bỏ qua quá trình xử lý octant phía sau nếu octant phía trước được tô đồng nhất với vài màu. Đối với các vùng không đồng nhất, thủ tục được gọi đệ quy, với các đối số mới – con của octant không đồng nhất và nút quadtree được tạo mới. Nếu octant phía trước rỗng, chỉ cần xử lý con của octant phía sau. Ngược lại, hai lời gọi đệ quy được làm, một cho octant phía sau và một cho octant phía trước.

```
type
oct_node_ptr =^ oct_node;
oct_entry  =  record
case homogeneous: boolean of
```

```
true : (color : integer);
false : (child : oct_node_ptr)
end; {record}
oct_node = array [0..7] of oct_entry;

quad_node_ptr = ^ quad_node;
quad_entry = record
case homogeneous: boolean of
true : (color : integer);
false : (child : oct_node_ptr)
end; {record}
```

```
quad_node = array[0..3] of quad_entry;

var
newquadtree : quad_node_ptr;
backcolor:   integer;

{Giả sử quang cảnh phía trước của một octree (với các octant 0, 1, 2,
3 ở phía trước) và, khi biểu diễn này được hiển thị, biến đổi nó thành
một quatree. Nhận một octree như input, nơi mà mỗi phần tử của
octree là một giá trị màu (homogeneous = true và octant được tô với
màu này) hoặc là con trỏ  đến một nút octant con  (homogeneous =
false).}
procedure convert_oct_to_quad(octree: oct_node;
var quadtree: quad_node);
var  k: integer;
begin
for k:=0 to 3 do begin
quadtree[k].homogeneous:=true;
if (octree[k].color>-1) then     {octant trước đầy}
quadtree[k].color:= octree[k].color
else        {octant trước rỗng}
if octree[k+4].homogeneous then
if (octree[k+4].color > -1) then    {trước rỗng, sau đầy}
quadtree[k].color:=octree[k+4].color
else        {trước và sau rỗng}
quadtree[k].color:=backcolor
else begin      {trước rỗng, sau không  đồng
nhất}
quadtree[k].homogeneous:=flase;
new(newquadtree);
quadtree[k].child: = newquadtree;
convert_oct_to_quad(octree[k+4].child^, newquadtree^);
```

```
end
```

```
else begin   {trước không  đồng nhất, sau không  được
biết}
quadtree[k].homogeneous:=false;
new(newquadtree);
quadtree[k].child:= newquadtree;
convert_oct_to_quad(octree[k+4].child^, newquadtree^);
convert_oct_to_quad(octree[k].child^, newquadtree^);
end;
end; {for}
end;
```

7.8. Loại bỏ các đường bị che khuất

Khi chỉ các phác họa của một đối tượng được hiển thị, các phương pháp khử đường khuất được dùng đến để loại bỏ các viền của đối tượng, cái bị che khuất bởi các mặt ở gần mặt phẳng quan sát hơn. Các phương pháp để loại bỏ các đường khuất có thể được phát triển bằng cách xem xét các viền của đối tượng một cách trực tiếp hay bằng cách chỉnh sửa lại các phương pháp khử mặt khuất.

Một tiếp cận trực tiếp để loại bỏ các đường khuất là so sánh mỗi đường với mỗi mặt trong ảnh. Quá trình này tương tự như clipping các đường bởi một cửa sổ có hình dạng bất kỳ, chỉ khác ở chỗ là bây giờ chúng ta muốn cắt bỏ các phần bị che khuất bởi các mặt. Đối với mỗi đường, các giá trị độ sâu được so sánh với các mặt để xác định xem phần đoạn thẳng nào không nhìn thấy được. Chúng ta có thể dùng các phương pháp cố kết để xác định các phần bị che khuất mà không cần kiểm tra toàn bộ các vị trí tọa độ. Nếu cả hai giao điểm của đường thẳng với hình chiếu của một biên bề mặt có độ sâu lớn hơn độ sâu của mặt ở các điểm này, đoạn thẳng giữa các giao điểm sẽ hoàn

Hình 7-22
Phần đoạn thẳng bị che khuất (nét đứt) của các đường thẳng: (a) đi qua phía sau một mặt và (b) đâm xuyên qua một mặt.

toàn bị che khuất, như hình 7-22 (a). Khi đường thẳng có độ sâu lớn hơn độ sâu ở một giao điểm với biên và có độ sâu nhỏ hơn độ sâu của mặt ở các giao điểm với biên còn lại, đường thẳng phải đi xuyên qua mặt như hình 7-22 (b). Trong trường hợp này, chúng ta tính tọa độ giao điểm của đường với mặt bằng cách dùng phương trình mặt và chỉ hiển thị các phần được nhìn thấy của đường thẳng.

Vài phương pháp khử mặt khuất dễ dàng được áp dụng để khử các đường khuất.

Dùng phương pháp mặt sau (back-face), chúng ta có thể nhận biết được các mặt sau của một đối tượng và chỉ hiển thị các biên của các mặt nhìn thấy được. Với phương pháp sắp xếp theo độ sâu, các mặt được vẽ vào trong vùng đệm làm tươi để phần bên trong của mặt có độ sáng nền, trong khi đó các biên có độ sáng là độ sáng vẽ. Bằng cách xử lý các mặt từ sau đến trước, các đường khuất bị xóa bởi các mặt ở gần hơn. Phương pháp chia vùng có thể được áp dụng để khử các đường khuất bằng cách chỉ hiển thị các biên của các mặt nhìn thấy được. Các phương pháp scan-line có thể được dùng để hiển thị các đường nhìn thấy được bằng cách bố trí các điểm dọc theo các đường quét, các điểm này trùng với các biên của các mặt nhìn thấy được. Bất kỳ phương pháp khử mặt khuất nào dùng các đường quét đều có thể được thay đổi thành phương pháp khử đường khuất theo cách tương tự (xem hình 7-23).

V1
V2
V3
V4
•
•
•
S1
•
• V5
S2
E1
E2
E2
E6
E6

VERTEX TABLE
V1: x1, y1, z1
V2: x2, y2, z2
V3: x3, y3, z3
V4: x4, y4, z4
V5: x5, y5, z5

EDGE TABLE
E1: V1, V2, S1
E2: V2, V3, S1,
S2
E3: V3, V1, S1
E4: V3, V4, S2
E5: V4, V5, S2
E6: V5, V2, S2

POLYGON TABLE
S1: E1, E2, E3
S2: E2, E4, E5, E6

Hình 7-23
Các bảng dữ liệu hình học cho một đối tượng ba chiều được biểu diễn bởi hai mặt phẳng, được hình thành với sáu cạnh và năm đỉnh.

7.9. Tổng kết chương 7
So sánh các phương pháp khử mặt khuất
Hiệu quả của các phương pháp khử mặt khuất phụ thuộc vào đặc tính của từng ứng dụng cụ thể. Nếu một mặt trong ảnh nằm trải ra trên hướng z để có rất ít sự nằm chồng theo độ sâu, phương pháp sắp xếp theo độ sâu có thể tốt nhất. Với các ảnh có những mặt nằm tách biệt theo chiều ngang, phương pháp scan-line hoặc phân chia vùng có thể là một lựa chọn tốt. Trong các phương pháp được chọn này, kỹ thuật sắp xếp và cố kết đem đến những thuận lợi do các thuộc tính tự nhiên của ảnh.
Vì sắp xếp và cố kết là quan trọng đến hiệu quả toàn diện của một phương pháp khử mặt khuất, các kỹ thuật để thực hiện các thao tác này cần được chọn lựa cẩn thận. Khi nào các đối tượng được biết theo thứ tự chính xác, như danh sách động chứa các cạnh trong bảng các cạnh được dùng trong phương pháp scan-line, một sắp xếp bubble sort sẽ hiệu quả để thực hiện việc đổi chỗ. Tương tự, kỹ thuật cố kết được áp dụng để

quét đường, vùng, hay các khung (frame) có thể là công cụ hữu hiệu làm tăng hiệu quả các phương pháp khử mặt khuất.
Như một quy tắc tổng quát, phương pháp sắp xếp theo độ sâu là một tiếp cận có hiệu quả cao cho các ảnh chỉ có vài mặt. Điều này do các ảnh này thường có vài mặt nằm chồng theo độ sâu. Phương pháp scan-line cũng thực hiện tốt khi ảnh chứa ít mặt. Dù vậy phương pháp scan-line hay sắp xếp theo độ sâu có thể được dùng hiệu quả cho các ảnh có đến vài ngàn mặt. Với các ảnh có hơn vài ngàn mặt, tiếp cận vùng đệm độ sâu hoặc octree thực hiện tốt nhất. Phương pháp vùng đệm độ sâu có một thời gian xử lý hằng, độc lập với số lượng mặt trong ảnh. Điều này bởi vì kích thước của các vùng mặt giảm khi số lượng mặt trong ảnh tăng. Do đó, một cách tương đối, phương pháp sắp xếp theo độ sâu thể hiện sự thực hiện kém khi ảnh đơn giản và thực hiện hiệu quả khi ảnh phức tạp. Tiếp cận này thì đơn giản để cài đặt, tuy nhiên, nó cần nhiều bộ nhớ hơn tất cả các phương pháp khác. Vì lý do này, một phương pháp khác, như octree hoặc phân chia vùng có thể được dùng cho các ảnh có nhiều mặt.
Khi phương pháp octree được dùng trong hệ thống, việc xử lý loại bỏ các mặt khuất sẽ nhanh và đơn giản. Chỉ cần dùng các phép cộng và trừ, không cần sắp xếp hoặc tìm các giao điểm. Một thuận lợi khác của octree là chúng lưu nhiều mặt hơn.
Trang 156 Chương 7: Khử các mặt kuất và đường khuất
Toàn bộ hình thể ba chiều của đối tượng có thể được hiển thị, điều này làm cho phương pháp octree hữu ích để thu được các lát cắt của các hình thể ba chiều.
Ta có thể kết hợp và cài đặt các phương pháp khử mặt khuất khác nhau theo các cách khác nhau. Hơn nữa, các thuật toán được cài đặt trong phần cứng, và các hệ thống xử lý song song đặc biệt được tận dụng để làm tăng hiệu quả của các phương pháp này. Các hệ thống phần cứng đặt biệt thường được dùng khi tốc độ xử lý được xem là quan trọng, ví dụ, trong việc tạo ra các hình ảnh động của các mô phỏng bay.

7.10. Bài tập chương 7

1. Phát triển một thủ tục, dựa trên kỹ thuật khử mặt sau, để xác định tất cả các mặt trước của một khối đa diện lồi với các mặt có màu khác nhau liên hệ đến mặt quan sát. Giả sử rằng đối tượng được định nghĩa trong hệ quan sát bàn tay trái với mặt xy dùng làm mặt quan sát.
2. Cài đặt thủ tục trong bài 1 vào một chương trình để chiếu trực giao các mặt nhìn thấy được của đối tượng lên một cửa sổ trong mặt phẳng quan sát. Để đơn giản, giả sử rằng tất cả các phần của đối tượng nằm ở phía trước mặt phẳng quan sát. Ánh xạ cửa sổ lên một vùng quan sát màn hình để hiển thị.
3. Cài đặt thủ tục trong bài 1 vào một chương trình để tạo ra một hình chiếu phối cảnh của các mặt nhìn thấy được của đối tượng lên một cửa sổ trong mặt phẳng quan sát. Để đơn giản, giả sử rằng đối tượng nằm phía trước mặt phẳng quan sát. Ánh xạ cửa sổ lên một vùng quan sát màn hình để hiển thị.
4. Viết một chương trình để cài đặt thủ tục của bài 1 cho một ứng dụng động, quay đối tượng một cách tăng dần xung quanh một trục, cái đâm xuyên qua đối tượng và song song với với mặt phẳng quan sát. Giả sử rằng đối tượng nằm hoàn toàn phía trước mặt phẳng quan sát. Dùng một phép chiếu song song trực giao để ánh xạ thành công các ảnh lên màn hình.
5. Dùng phương pháp vùng đệm độ sâu để hiển thị các mặt nhìn thấy được của một đối tượng bất kỳ, cái được định nghĩa trong hệ tọa độ chuẩn ở phía trước vùng quan sát. Các phương trình (7-4) và (7-5) sẽ được dùng để thu được các

giá trị độ sâu cho tất cả các điểm trên mặt mỗi khi một độ sâu khởi tạo vừa

được xác định. Sự đòi hỏi không gian lưu trữ cho vùng đệm độ sâu có thể được xác định như thế nào từ định nghĩa các đối tượng để được hiển thị?
6. Phát triển một chương trình cài đặt thuật toán scan-line để hiển thị các mặt nhìn thấy được của một đối tượng được định nghĩa bất kỳ nằm trước vùng quan sát. Dùng các bảng đa giác và bảng cạnh (polygon table, edge table) để lưu trữ sự định nghĩa của đối tượng, và dùng kỹ thuật cố kết để tính các điểm dọc theo và giữa các đường quét.
7. Cài đặt một chương trình để hiển thị các mặt nhìn thấy được của một khối đa diện lồi, dùng các thuật toán của họa sĩ (painter's algorithm). Tức là, các bề mặt phải được sắp theo độ sâu và được vẽ lên màn hình từ sau đến trước.
8. Mở rộng chương trình của bài 7 để hiện thị một đối tượng được định nghĩa bất kỳ với các mặt phẳng, dùng các kiểm tra sắp xếp độ sâu (depth-sorting checks) để có các mặt theo thứ tự sắp hợp lý.
9. Cho các ví dụ về các trường hợp mà tại đó hai phương pháp đã được thảo luận về kiểm tra 3 trong các thuật toán phân chia vùng sẽ thất bại để từ đó chỉ ra một cách đúng đắn một mặt bao quanh có thể che khuất tất cả các mặt.
10. Phát triển một thuật toán có thể kiểm tra một mặt được cho tương tác với một vùng chữ nhật để quyết định xem nó là một mặt bao quanh, nằm chồng, bên trong, hay nằm ngoài.
11. Mở rộng các phương pháp trong bài tập 10 thành một thuật toán để sinh ra một biểu diễn quadtree cho các mặt nhìn thấy được của đối tượng bằng cách áp dụng các kiểm tra vùng con (area-subdivision tests) để xác định các giá trị của các phần tử quadtree.
12. Cài đặt một thuật toán để nạp biểu diễn quadtree của bài tập 11 thành đường quét (raster) để hiển thị.
13. Viết một chương trình lên hệ thống của bạn để hiển thị một biểu diễn octree cho một đối tượng để các mặt khuất bị loại bỏ.
14. Phát triển một thuật toán để loại bỏ các đường khuất bằng cách so sánh mỗi đường trong ảnh với mỗi mặt.

15. Thảo luận làm thế nào việc tháo bỏ các đường khuất có thể được thực hiện với các phương pháp khử mặt khuất khác nhau.
16. Cài đặt một thủ tục để hiển thị các cạnh bị che khuất của một đối tượng chứa các mặt phẳng thành những đường nét đứt.
HẾT

TRƯỜNG ĐẠI HỌC KINH TẾ KỸ THUẬT CÔNG NGHIỆP

KHOA CÔNG NGHỆ THÔNG TIN

BÀI GIẢNG MÔN

ĐỒ HỌA MÁY TÍNH

Số ĐVHT: 2

NỘI DUNG MÔN HỌC

- Chương 1: Tổng quan về đồ họa máy tính
- Chương 2: Các phép biến đổi cơ sở
- Chương 3: Hiển thị các đối tượng đồ họa
- Chương 4: Một số kỹ thuật đồ họa nâng cao

CHƯƠNG 2: CÁC PHÉP BIẾN ĐỔI CƠ SỞ

I. Giới thiệu

1.1 Giới thiệu

1.2 Các hệ tọa độ

II. Các thuật toán vẽ đường thẳng, đường tròn

III. Các phép biến đổi của đối tượng hai chiều

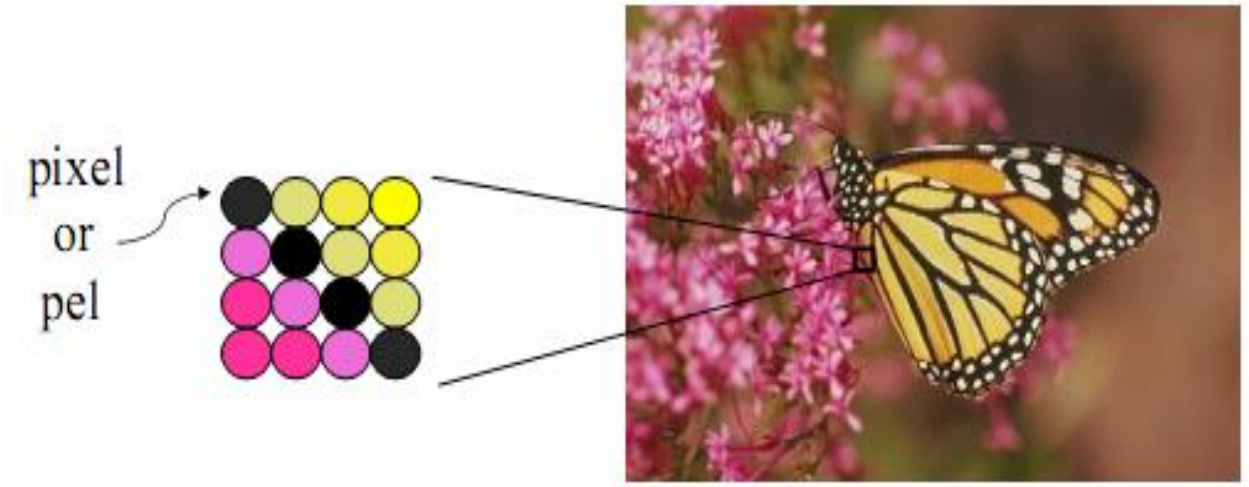
IV. Kết hợp các phép biến đổi

V. Một số tính chất của phép biến đổi affine

VI. Một số phép biến đổi khác

GIỚI THIỆU

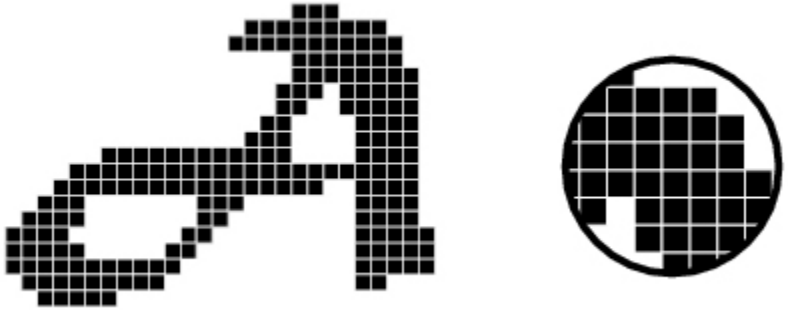
Bất kỳ ảnh nào cũng được cấu trúc từ tập các đối tượng đơn giản hơn.



GIỚI THIỆU

Với các ảnh phát sinh từ đồ họa máy tính, hình dạng và màu sắc của mỗi đối tượng có thể được mô tả riêng biệt bằng 2 mô hình:

+ Dãy các pixel (mô hình dựa trên mẫu số hóa), đối tượng được xây dựng từ tập các pixel.



+ Tập các đối tượng hình học cơ sở (mô hình dựa trên đặc trưng hình học).

Mỗi đối tượng đồ họa cơ sở được mô tả thông qua dữ liệu về tọa độ và các thuộc tính của nó.

GIỚI THIỆU

- Đối tượng đồ họa cơ sở đơn giản nhất là điểm và đoạn thẳng, đường tròn, các đường conics, mặt bậc hai, các vùng tô đa giác, chuỗi kí tự, …

- Các công cụ lập trình đồ họa cung cấp:

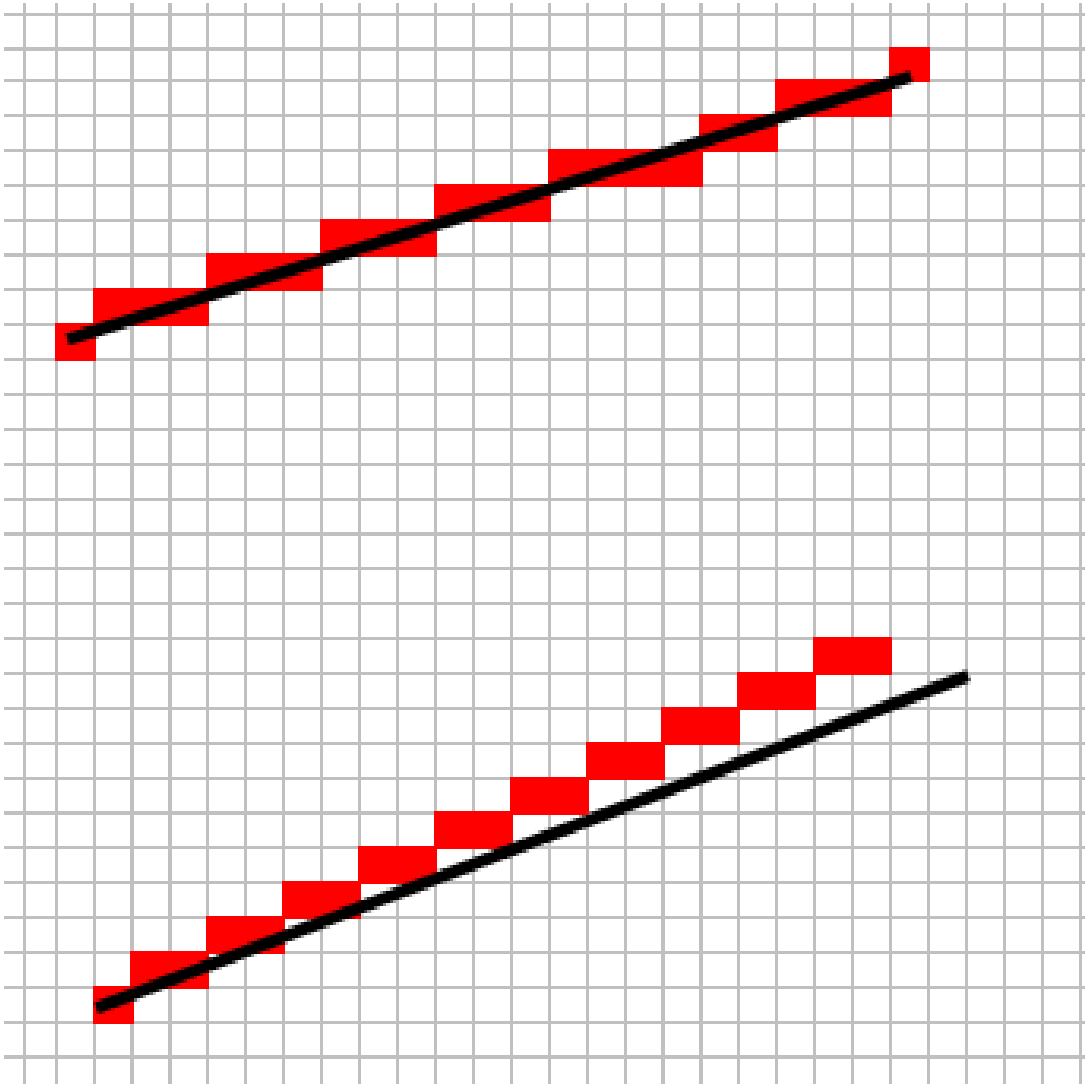
  - Các hàm mô tả một ảnh dưới dạng các đối tượng đồ họa cơ sở

  - Các hàm kết hợp tập các đối tượng cơ sở để tạo thành đối tượng có cấu trúc phức tạp hơn.

Chương này sẽ khảo sát các thuật toán hiển thị các đối tượng đồ họa cơ sở cho các thiết bị hiển thị dạng điểm.

GIỚI THIỆU

Các thuật toán thực hiện quá trình chuyển đổi các đối tượng đồ họa cơ sở được mô tả trong hệ tọa độ thực về dãy các pixel có tọa độ nguyên của thiết bị hiển thị.



GIỚI THIỆU

Có hai yêu cầu đặt ra cho các thuật toán là :

1, Điểm nguyên xấp xỉ đối tượng tốt nhất:

Do: Đối tượng được mô tả trong hệ tọa độ thực là *liên tục*, còn đối tượng trong hệ tọa độ thiết bị là *rời rạc*

Nên: Đối tượng được hiển thị trên hệ tọa độ thiết bị phải "giống nhất" với đối tượng trong hệ tọa độ thế giới thực.

2, Tối ưu hóa về mặt tốc độ:

Do: Các đối tượng đồ họa cơ sở là thành phần chính cấu trúc nên các đối tượng phức tạp,

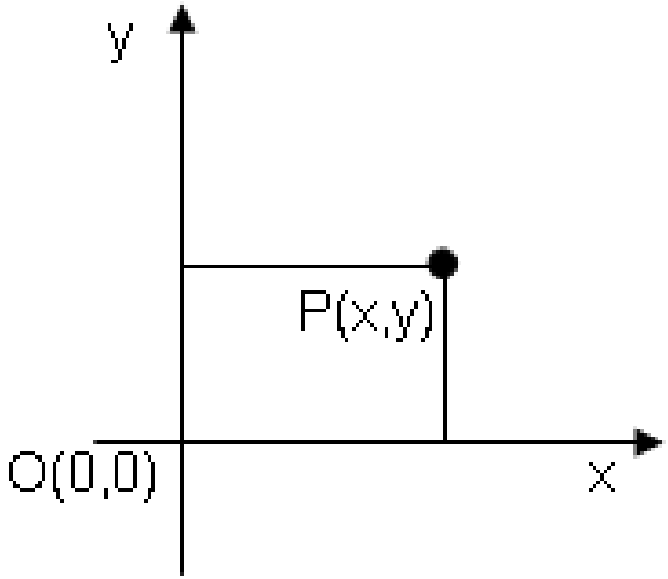
Nên: Các thuật toán hiển thị chúng cần phải được tối ưu hóa về mặt tốc độ.

CÁC HỆ TỌA ĐỘ

- Hệ tọa độ thế giới thực:

Hệ tọa độ thế giới thực mô tả các đối tượng thế giới thực.

Thường dùng hệ tọa độ Descartes.

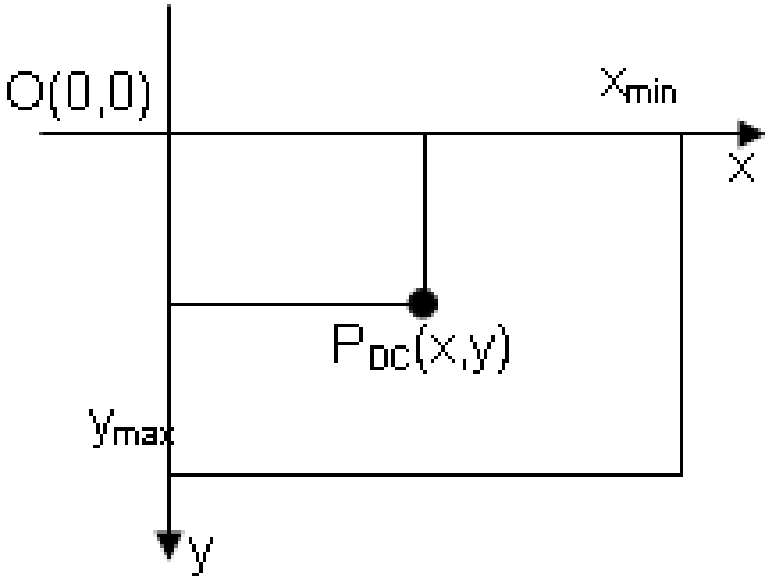


Các điểm trong hệ được mô tả bởi một cặp tọa độ (x, y), trong đó x, y [redacted]**R.**

Do vậy, *các điểm trong hệ tọa độ thực được định nghĩa* ***liên tục.***

CÁC HỆ TỌA ĐỘ

- Hệ tọa độ thiết bị:



Hệ tọa độ thiết bị mô tả các đối tượng trong một thiết bị xuất cụ thể nào đó: máy in, màn hình, ...

Đặc điểm của hệ tọa độ thiết bị:

+ Mỗi điểm được mô tả bởi một cặp tọa độ (x, y), trong đó x, y [redacted]**N.**

Do vậy *các điểm* ***rời rạc*** *nhau* .

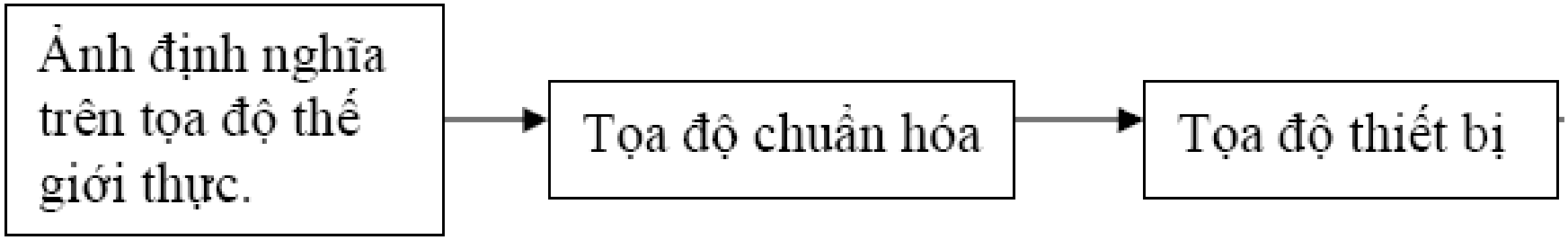
+ x, y bị giới hạn trong một khoảng nào đó và tùy theo từng loại thiết bị.

CÁC HỆ TỌA ĐỘ

- **Hệ tọa độ thiết bị chuẩn:**

✓ Hệ tọa độ trên các thiết bị khác nhau là khác nhau nên một hình ảnh hiển thị trên thiết bị này là chính xác nhưng có thể là không chính xác khi hiển thị trên thiết bị khác.

✓ Hệ tọa độ thiết bị chuẩn đại diện chung cho tất cả các thiết bị để mô tả các hình ảnh mà không phụ thuộc vào thiết bị hiển thị nào



CH□ƠNG 2: CÁC PHÉP BIẾN ĐỔI CƠ SỞ

I. Giới thiệu

II. Các thuật toán vẽ đường thẳng, đường tròn

2.1 Các đối tượng đồ họa cơ sở

2.2 Các thuật toán vẽ đường thẳng

2.3 Thuật toán Midpoint vẽ đường tròn

III. Các phép biến đổi của đối tượng hai chiều

IV. Kết hợp các phép biến đổi

V. Một số tính chất của phép biến đổi affine

VI. Một số phép biến đổi khác

CÁC ĐỐI T□ỢNG ĐỒ HỌA CƠ SỞ

+ Điểm:

Điểm là thành phần cơ sở được định nghĩa trong một hệ tọa độ.

Đối với hệ tọa độ hai chiều mỗi điểm được xác định bởi cặp tọa độ (x, y).

Ngoài thông tin về tọa độ, điểm còn có thuộc tính là màu sắc.

Thủ tục vẽ một điểm (x,y) với mầu c: Putpixel(x,y,c)

CÁC ĐỐI T□ỢNG ĐỒ HỌA CƠ SỞ

+ Đoạn thẳng:

Được xác định bởi: Màu sắc, độ rộng nét vẽ, kiểu nét vẽ của đoạn thẳng (nét liền, nét đứt, chấm gạch…)

Một đường thẳng được xác định nếu biết hai điểm thuộc nó (x_1, y_1) và (x_2, y_2):

$$\frac{x - x_1}{y - y_1} = \frac{x_2 - x_1}{y_2 - y_1}$$

Ta có: **y = mx + b**, trong đó:

$$m = \frac{Dy}{Dx}, \quad Dy = y_2 - y_1, \quad Dx = x_2 - x_1$$

CÁC ĐỐI T□ỢNG ĐỒ HỌA CƠ SỞ

+ Đường gấp khúc:

Tập các đoạn thẳng nối với nhau một cách tuần tự, điểm giao của hai đoạn thẳng được gọi là đỉnh.

Các đường gấp khúc được xác định qua danh sách các đỉnh, mỗi đỉnh được cho bởi các cặp tọa độ.

+ Đa giác:

Là một đường gấp khúc có điểm đầu và điểm cuối trùng nhau.

CÁC ĐỐI T□ỢNG ĐỒ HỌA CƠ SỞ

+ Vùng tô:

Gồm đường biên (là một đường khép kín) và vùng bên trong.

Xác định bởi:

Thuộc tính của đường biên: là các thuộc tính như thuộc tính của đoạn thẳng.

Thuộc tính của vùng bên trong: bao gồm màu tô và mẫu tô.

+ Kí tự, chuỗi kí tự:

xác định bởi:

Màu sắc của các kí tự, Font chữ, kích thước, khoảng cách giữa các kí tự, sự canh chỉnh (gióng lề), hướng hiển thị tuần tự của các kí tự.

II CÁC TT VẼ Đ□ỜNG THẲNG, Đ□ỜNG TRÒN

2.1 Các đối tượng đồ họa cơ sở

2.2 Các thuật toán vẽ đường thẳng

2.2.1 Mở đầu

2.2.2 Thuật toán DDA

2.2.3 Thuật toán Bresenham

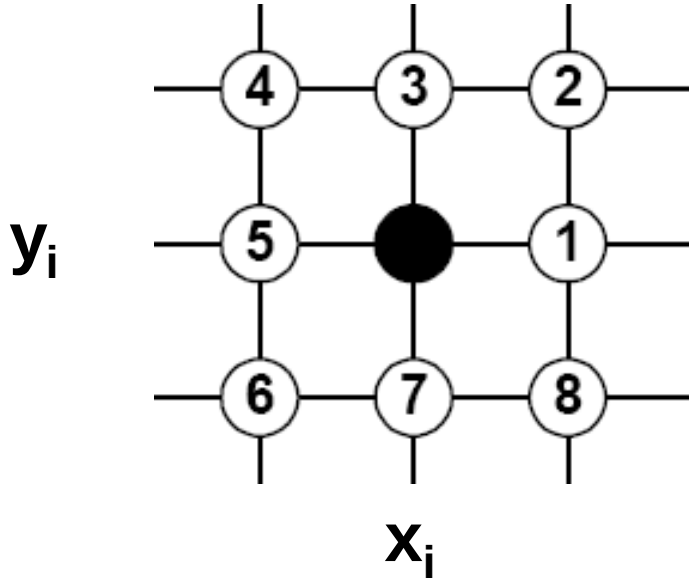
2.2.4 Thuật toán Midpoint

2.3 Thuật toán Midpoint vẽ đường tròn

MỞ ĐẦU

Gọi (x_i,y_i), i= 0,1,… là tọa độ các điểm nguyên sau khi xấp xỉ các điểm thực, và là các điểm sẽ được hiển thị trên màn hình.

(x_i,y_i) là điểm đen trên hình vẽ



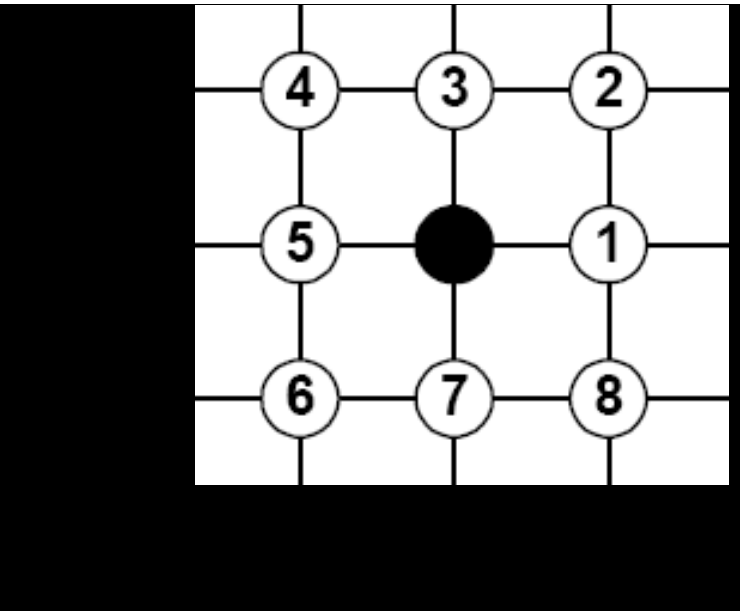
Nếu bước ***i*** xác định được (x_i,y_i)

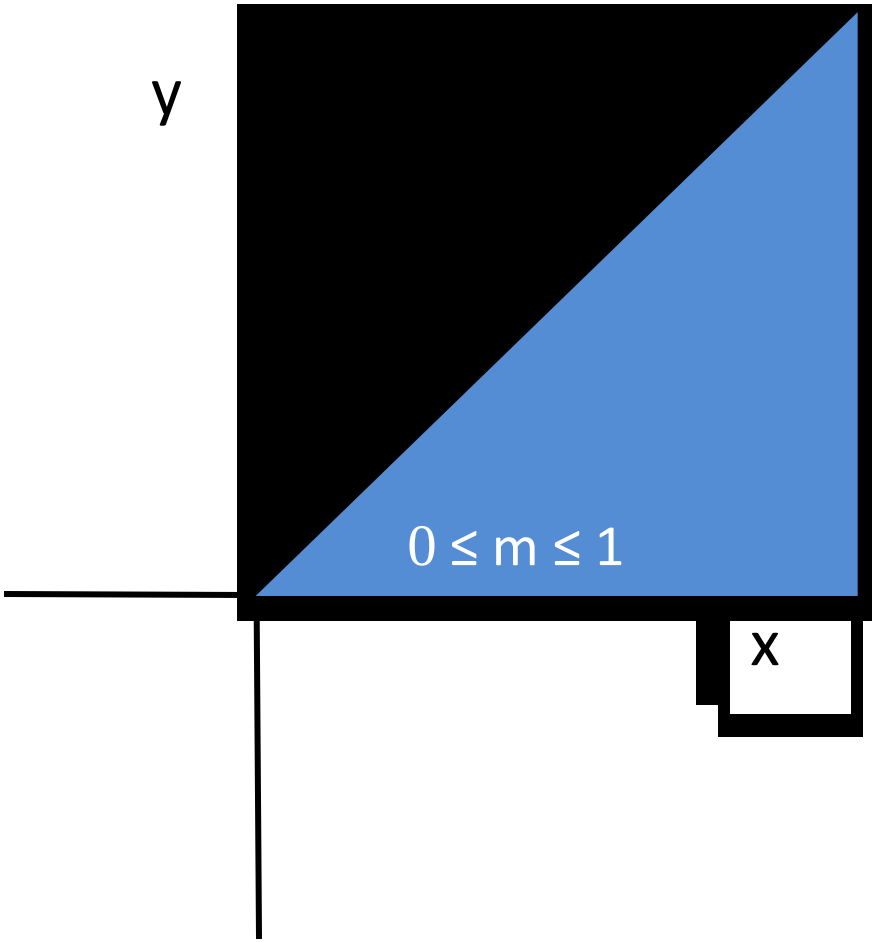
Thì bước thứ ***i+1*** xác định được (x_{i+1},y_{i+1})

(x_{i+1},y_{i+1}) là một trong tám điểm được đánh số từ 1 đến 8.

MỞ ĐẦU

Xét đoạn thẳng có hệ số góc $0 \leq m \leq 1$ và $D_x > 0$.



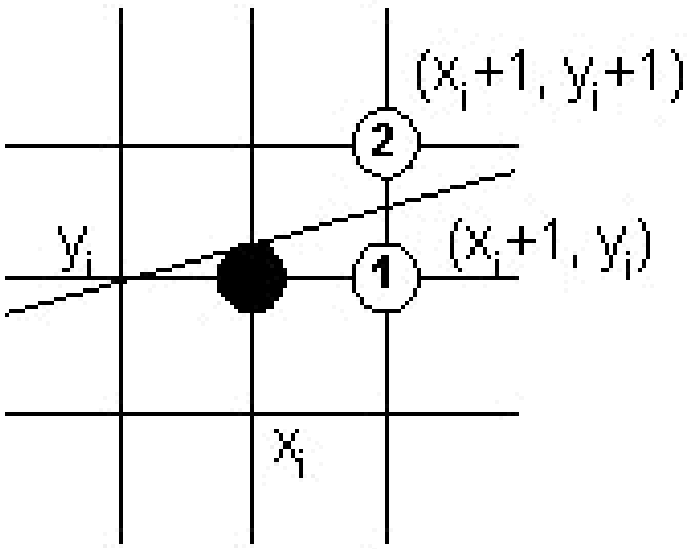


MỞ ĐẦU

Xét đoạn thẳng có hệ số góc $0 \le m \le 1$ và $D_x > 0$.

Nếu (x_i, y_i) là điểm đã xác định được ở bước thứ i

Thì điểm cần chọn (x_{i+1}, y_{i+1}) ở bước thứ (i+1) sẽ là một trong hai trường hợp như hình vẽ.



Như vậy : $x_{i+1} = x_i + 1,$

$y_{i+1} \in \{ y_i, y_i + 1 \}$

Thuật toán DDA (Digital Differential Analyzer)

Ý tưởng:

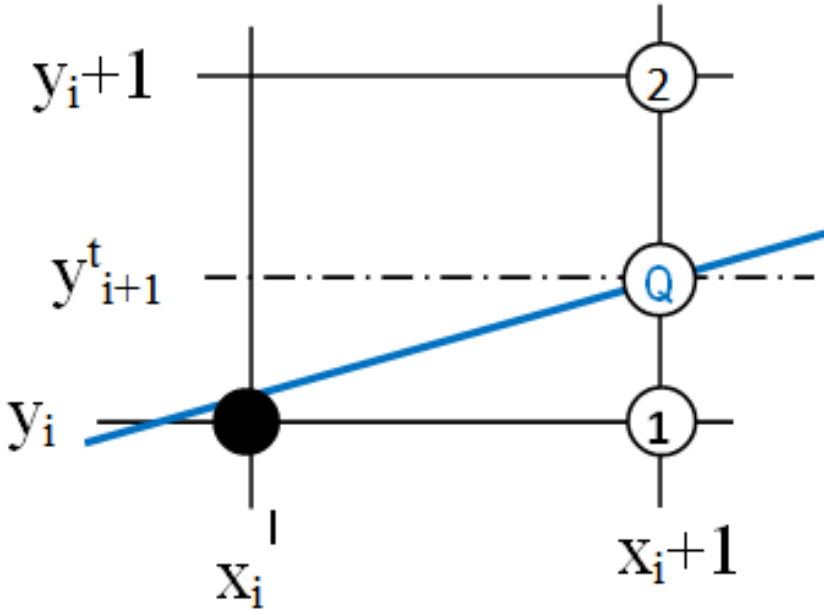
Dựa vào phương trình y = mx + b để chọn y_{i+1} là y_i hay y_i+1.

Tính tọa độ điểm $Q(x_{i+1}, y^t_{i+1})$ thuộc đoạn thẳng thực:

Từ pt của đoạn thẳng ta có:

$$y^t_{i+1} = mx_{i+1} + b$$

➡ $y^t_{i+1} = y_i + m$



y_{i+1} là giá trị sau khi làm tròn y^t_{i+1} :

$$y_{i+1} = \text{round}(y^t_{i+1})$$

Do vậy: $y_{i+1} = \text{round}(y_i + m)$

Thuật toán DDA (Digital Differential Analyzer)

Như vậy:

Nếu bước thứ i ta vẽ được điểm (x_i,y_i)

Thì bước thứ i+1 ta vẽ được điểm (x_{i+1},y_{i+1})

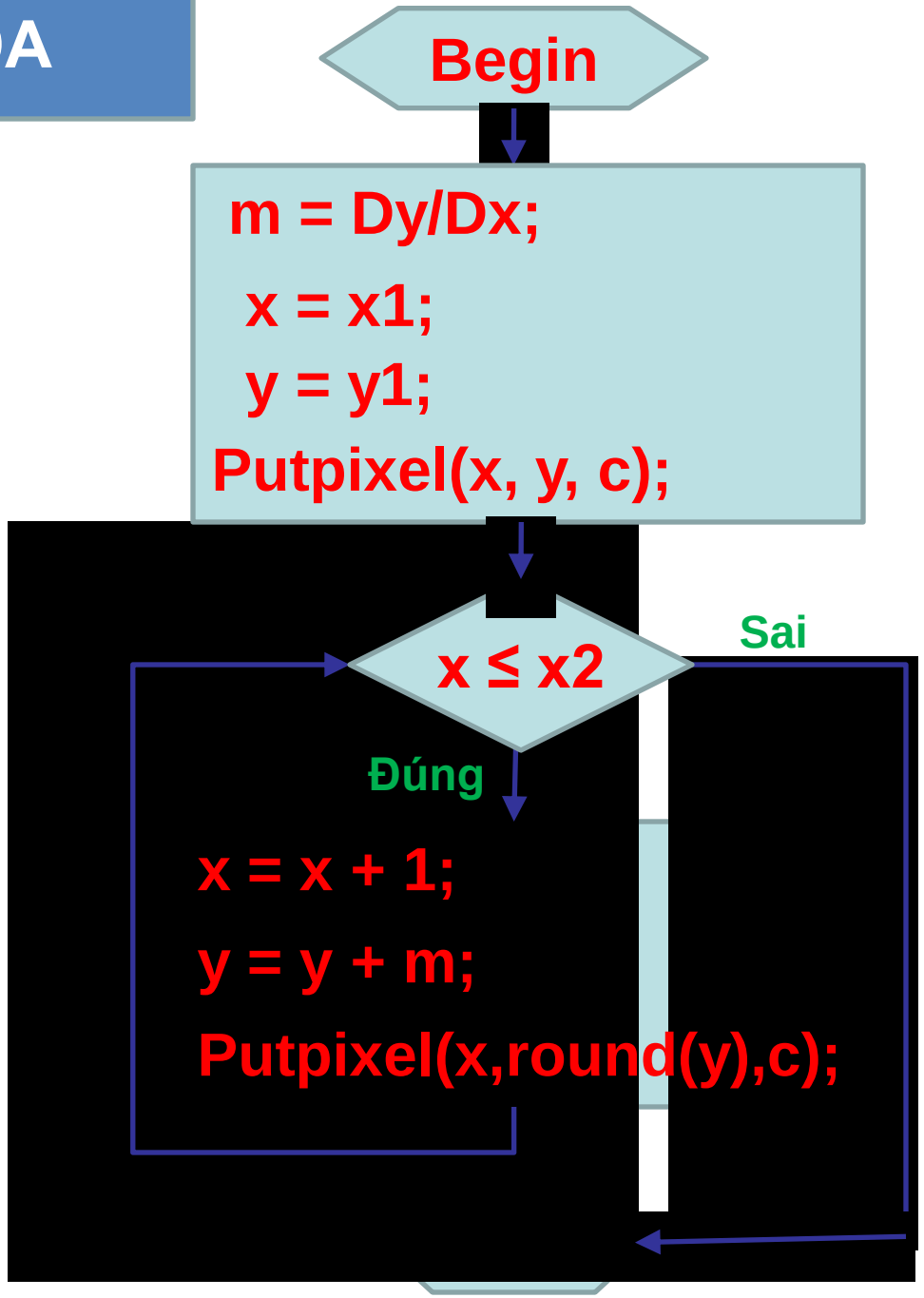
Trong đó:

$$x_{i+1} = x_i + 1,$$

$$y_{i+1} = \text{round}(y_i + m)$$

Thuật toán DDA

Giải thuật:



Thuật toán DDA (Digital Differential Analyzer)

Nhận xét:

- Việc sử dụng công thức $y^t_{i+1} = y_i + m$ đã khử được phép nhân trên số thực.

- Còn bị hạn chế về mặt tốc độ do vẫn còn phép toán cộng số thực và làm tròn.

Thuật toán Bresenham

Ý tưởng:

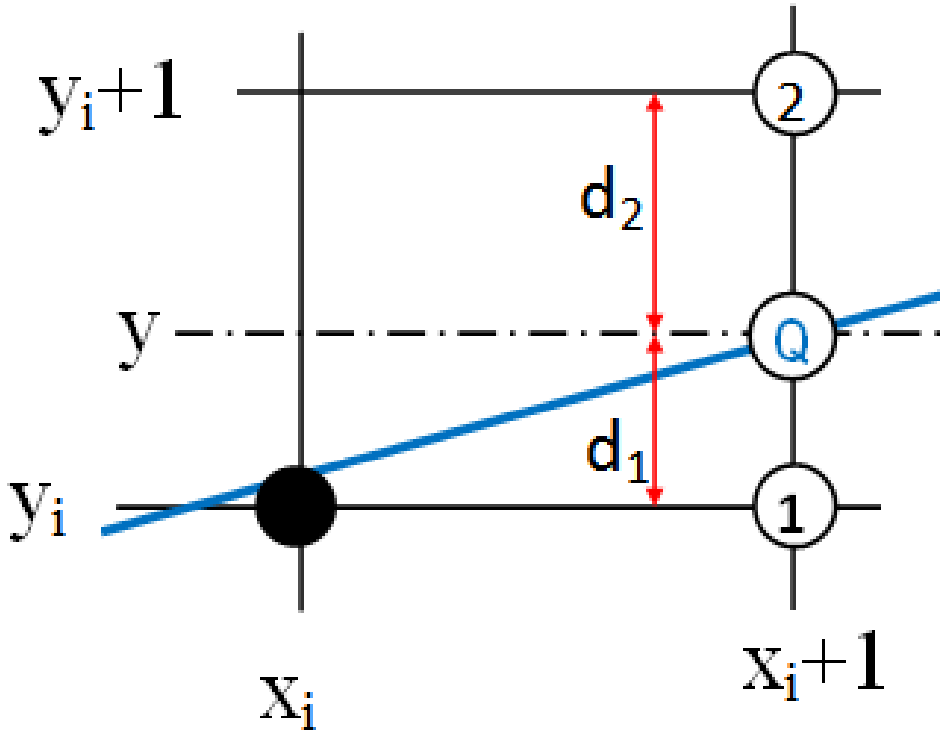
Thuật toán đưa ra cách chọn y_{i+1} là y_i hay y_i+1 theo hướng hạn chế tối đa các phép toán trên số thực.

Gọi $Q(x_{i+1},y) \in$ đường thẳng thực, ta có:

$$y = mx_{i+1} + b .$$
$$= m(x_i+1) + b .$$

Đặt:

$$d_1 = y - y_i ,$$
$$d_2 = (y_i + 1) - y$$



Thuật toán Bresenham

Xét $p_i = Dx\,(d_1 - d_2)$

$= Dx\,(2y - 2y_i - 1)$

$\rightarrow p_i = Dx\,(2\,(m\,(x_i + 1) + b) - 2y_i - 1)$

do $m = Dy/Dx$

$\rightarrow P_i = 2x_iDy - 2y_iDx + c$

với $c = 2Dy + (2b - 1)Dx$

Ta tính:

$$p_{i+1} - p_i = (2Dyx_{i+1} - 2Dxy_{i+1} + c) - (2Dyx_i - 2Dxy_i + c)$$

$$\Leftrightarrow p_{i+1} - p_i = 2Dy(x_{i+1} - x_i) - 2Dx(y_{i+1} - y_i)$$

$$\Leftrightarrow p_{i+1} - p_i = 2Dy - 2Dx(y_{i+1} - y_i), \text{ do } x_{i+1} = x_i + 1$$

Thuật toán Bresenham

Việc chọn điểm (x_{i+1},y_{i+1}) là ① hay ② phụ thuộc vào dấu của biểu thức $d_1 - d_2$ hay dấu của p_i, ta có:

Nếu $d_1 - d_2 < 0$ hay $p_i < 0$ ⇨ chọn ① ⇨ $y_{i+1} = y_i$

⇨ $p_{i+1} = p_i + 2Dy$

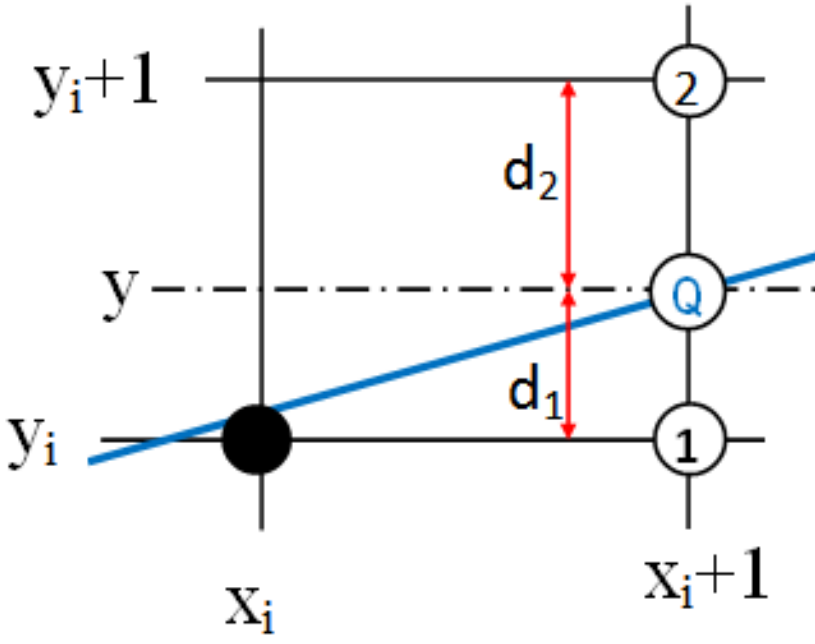
Nếu $d_1 - d_2 \geq 0$ hay $p_i \geq 0$ ⇨ chọn ② ⇨ $y_{i+1} = y_i + 1$

⇨ $p_{i+1} = p_i + 2Dy - 2Dx$

Tính: $P_0 = 2x_0Dy - 2y_0Dx + c$

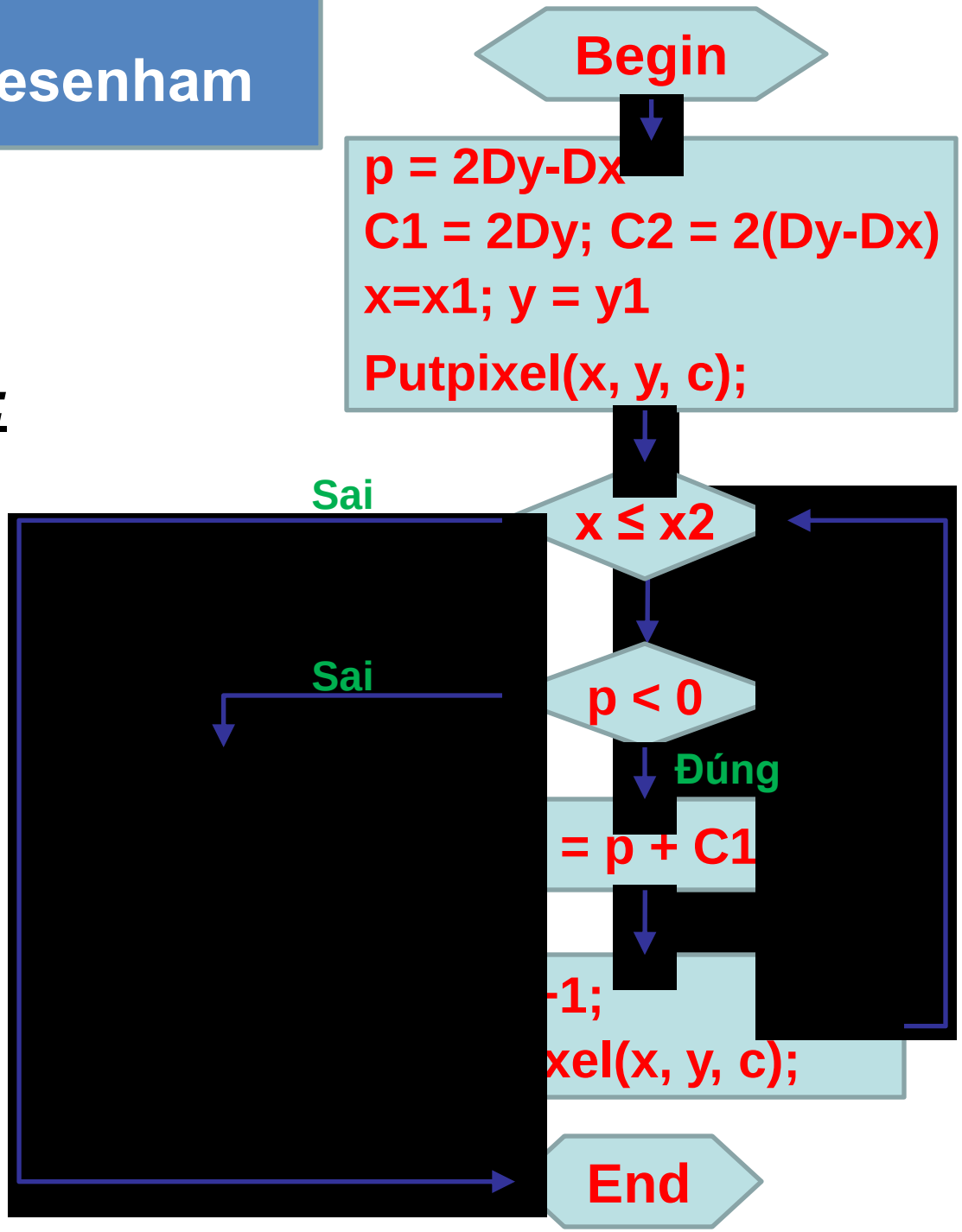
Do $y_0 = mx_0 + b$

⇨ $p_0 = 2Dy - Dx$



Thuật toán Bresenham

Giải thuật:



Thuật toán Bresenham

Nhận xét:

- Thuật toán Bresenham chỉ làm việc trên số nguyên

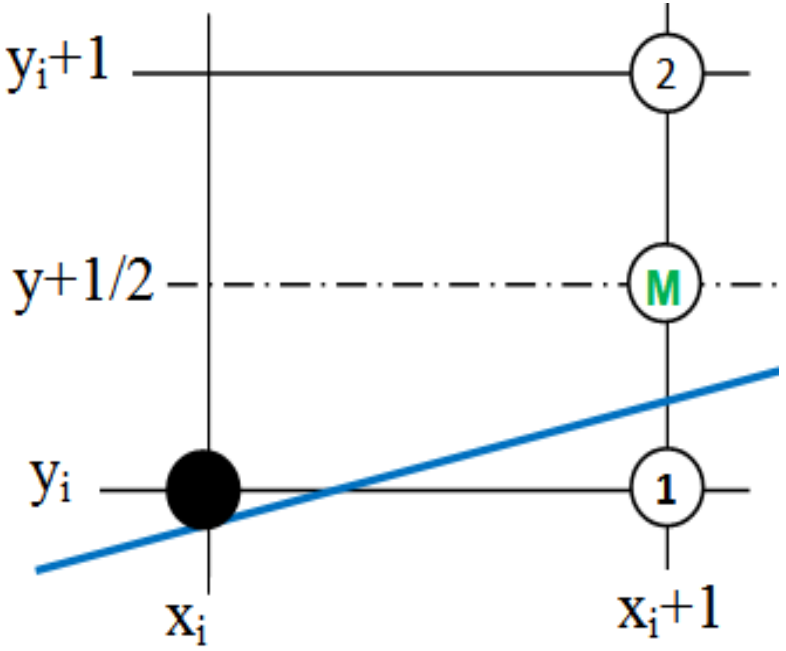
- Các thao tác trên số nguyên chỉ là phép cộng

⇨ Làm tăng tốc độ đáng kể so với thuật toán DDA.

Thuật toán Midpoint

Ý tưởng:

TT chọn y_{i+1} là y_i+1 hay y_i bằng cách so sánh vị trí tương đối giữa trung điểm $M(x_i+1, y_i + ½)$ của ① và ② với đường thẳng thực.



Từ pt đoạn thẳng y = mx + b

⇨ xDy – yDx + c = 0

Đặt F(x,y) = xDy – yDx +c

Đặt $p_i = 2F(M)=2F(x_i +1, y_i + ½)$

⇨ $p_i = 2(x_i+1)Dy - 2(y_i+1/2)Dx + 2c$

Ta tính: $p_{i+1} – p_i = 2 (x_{i+1} – x_i)Dy – 2 (y_{i+1} – y_i)Dx$

Thuật toán Midpoint

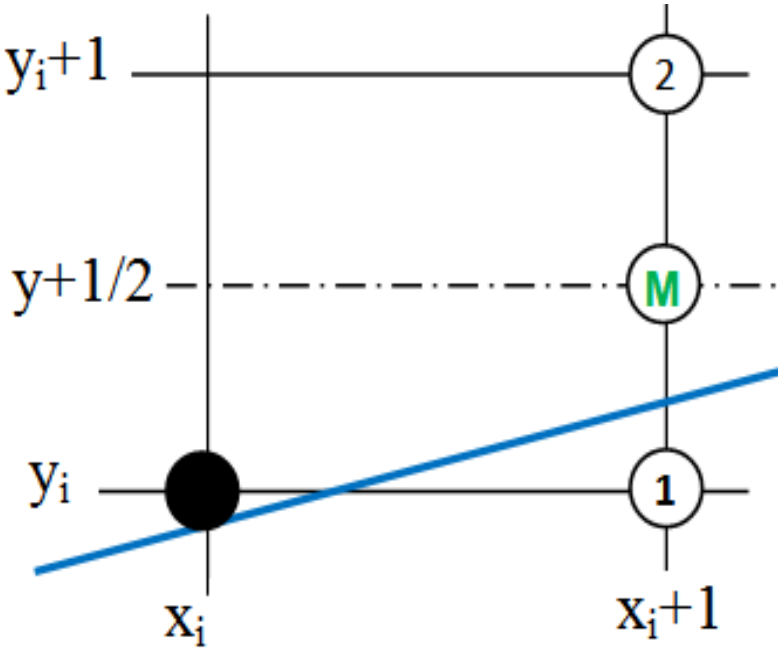
Nhận xét rằng:

- Nếu F(M) < 0 ⇨ M nằm phía trên của đoạn thẳng

Vậy điểm cần vẽ tại bước thứ i+1 là ① → $y_{i+1} = y_i$

- Ngược lại, ⇨ chọn ② → $y_{i+1} = y_i + 1$

Vậy, nếu biết được dấu của p_i (là dấu của F(M) tại bước thứ i thì ta có thể xác định được điểm cần vẽ tại bước thứ i+1.



Thuật toán Midpoint

Ta có:

$p_{i+1} - p_i = 2Dy - 2(y_{i+1} - y_i)Dx$

Như vậy:

Nếu $p_i < 0$ ⇨ $p_{i+1} = p_i + 2Dy$

Nếu $p_i \geq 0$ ⇨ $p_{i+1} = p_i + 2Dy - 2Dx$

Tính: $p_0 = 2F(x_0 + 1, y_0 + 1/2) = 2(x_0+1)Dy - 2(y_0+1/2)Dx + 2c$

$= 2(x_0Dy - y_0Dx + c) + 2Dy - Dx$

$p_0 = 2Dy - Dx$

Nhận xét:

Thuật toán Midpoint cho kết quả tương tự thuật toán Bresenham.

II. CÁC TT VẼ Đ□ỜNG THẲNG, Đ□ỜNG TRÒN

2.1 Các đối tượng đồ họa cơ sở

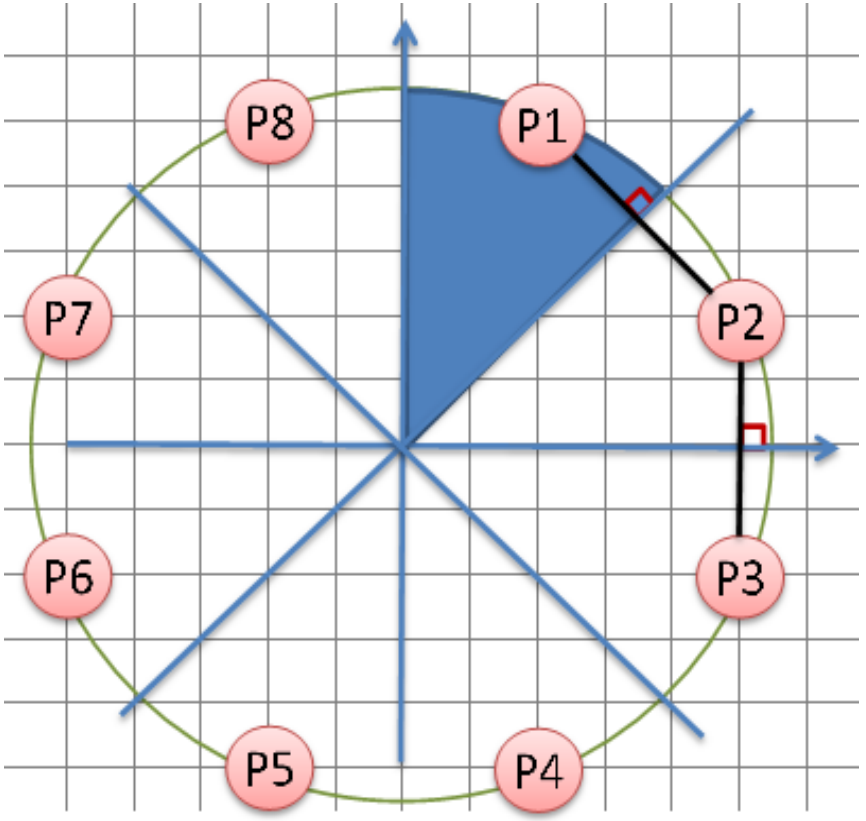
2.2 Các thuật toán vẽ đường thẳng

2.3 Thuật toán Midpoint vẽ đường tròn

Thuật toán Midpoint vẽ đường tròn

Nhận xét:

Do tính đối xứng của đường tròn nên ta chỉ cần khảo sát trên 1/8 đường tròn, sau đó lấy đối xứng.



Giả sử P(x,y) là một điểm bất kỳ trên cung được khảo sát (cung tô màu xám). Các vị trí đối xứng với P qua các trục tọa độ và các đường phân giác là (x, y) và (y, x).

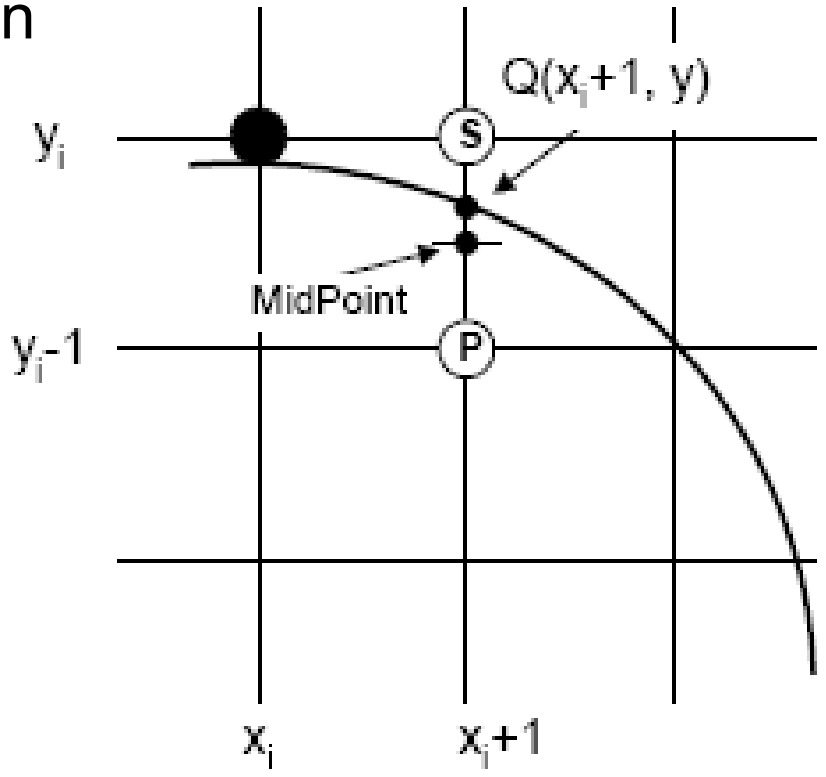
Thuật toán Midpoint vẽ đường tròn

Chọn điểm bắt đầu vẽ là điểm (0,R)

Nếu (x_i,y_i) là điểm nguyên được vẽ ở bước thứ i

Thì ở bước thứ (i+1) có thể chọn

$S(x_i+1,y_i)$ hoặc **$P(x_i+1, y_i - 1)$**.



Vậy:

$$x_{i+1} = x_i+1$$

$$y_{i+1} \quad y_i-1,\ y_i\}$$

Đặt $F(x,y) = x^2+ y^2 - R^2$, ta có :

Thuật toán Midpoint vẽ đường tròn

Đặt $F(x,y) = x^2 + y^2 - R^2$, ta có :

$$F(x,y)\begin{cases} < 0, \text{nếu } (x,y) \text{ nằm trong đường tròn} \\ = 0, \text{nếu } (x,y) \text{ nằm trên đường tròn} \\ > 0, \text{nếu } (x,y) \text{ nằm ngoài đường tròn.} \end{cases}$$

Xét $p_i = F(\text{Midpoint}) = F(x_i + 1, y_i - 1/2)$. Ta có:

- Nếu $p_i < 0$, điểm Midpoint nằm trong đường tròn

  vậy điểm cần vẽ tại bước thứ i+1 là S $\rightarrow$ $y_{i+1} = y_i$

- Nếu $p_i \geq 0$, chọn P $\rightarrow$ $y_{i+1} = y_i - 1$

Thuật toán Midpoint vẽ đường tròn

Vì pi= $F(x_i + 1, y_i - 1/2)$

Ta có:

$p_{i+1} - p_i = F(x_{i+1} + 1, y_{i+1} - 1/2) - F(x_i + 1, y_i - 1/2)$

$= [(x_{i+1}+1)^2 + (y_{i+1} - 1/2)^2 - R^2] - [(x_{i+1})^2 + (y_i - 1/2)^2 - R^2]$

$= 2x_i + 3 + (y^2_{i+1} - y^2_i) - (y_{i+1} - y_i)$

Nếu $p_i < 0$ thì $p_{i+1} = p_i + 2x_i + 3$

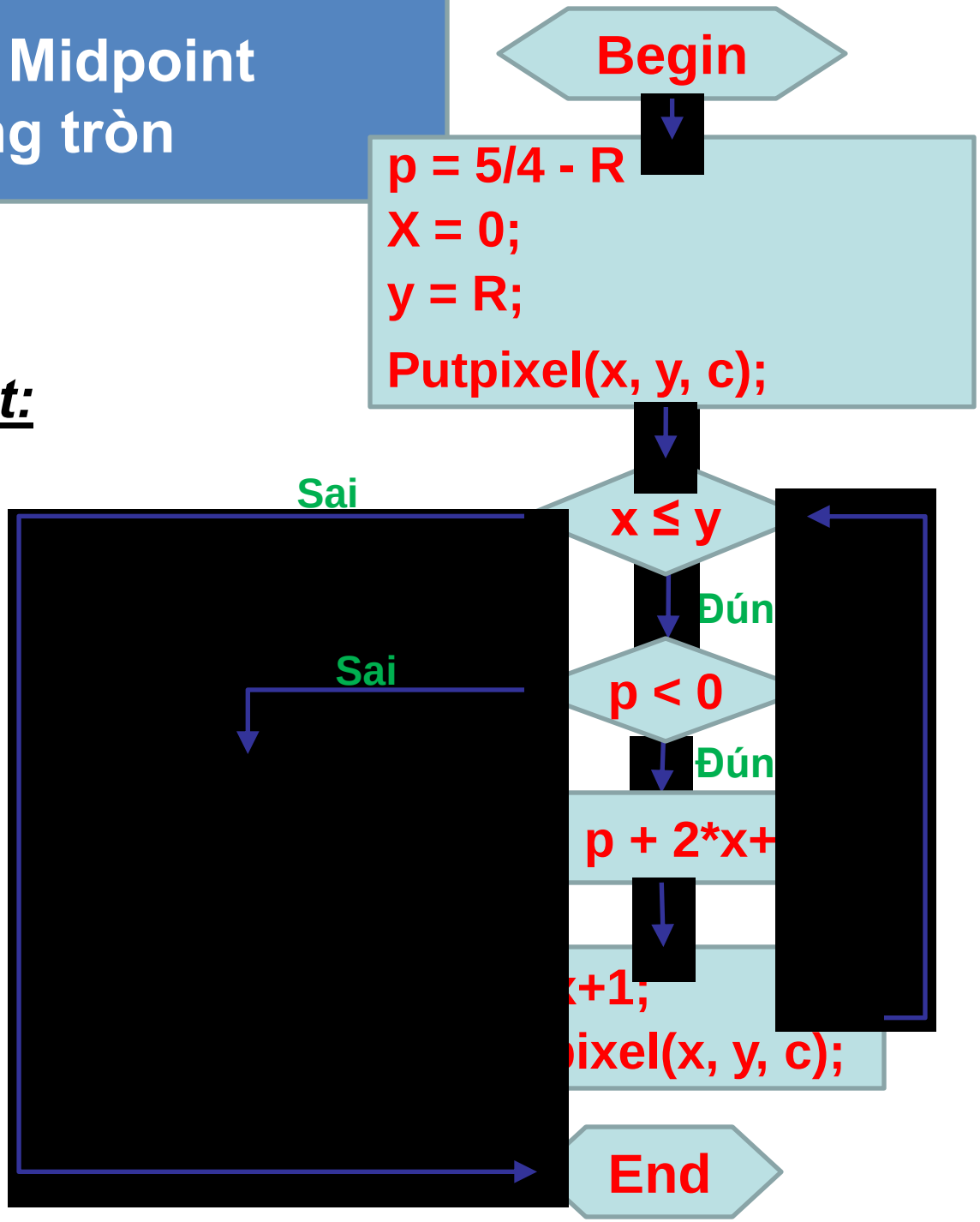
Nếu $p_i \geq 0$ thì $p_{i+1} = p_i + 2x_i - 2y_i + 5$

Tính p_0 ứng với điểm ban đầu (0, R), ta có:

$p_0 = F(0 + 1, R - 1/2) = 5/4 - R$

Thuật toán Midpoint vẽ đường tròn

Giải thuật:



II. CÁC TT VẼ Đ□ỜNG THẲNG, Đ□ỜNG TRÒN

Bài tập:

1. Trình bày các thuật toán vẽ đường thẳng trong trường hợp m>1

2. Tìm các điểm phát sinh khi vẽ đường thẳng bằng thuật toán DDA trong trường hợp 0<m<1 và m>1

3. Tìm các điểm phát sinh khi vẽ đường thẳng bằng thuật toán Bresenham và Midpoint trong trường hợp 0<m<1 và m>1

4. Tìm các điểm phát sinh khi vẽ đường tròn bằng thuật toán Midpoint

TRƯỜNG ĐẠI HỌC ĐÀ LẠT
KHOA CÔNG NGHỆ THÔNG TIN

ThS. VÕ PHƯƠNG BÌNH

GIÁO TRÌNH

ĐỒ HỌA MÁY TÍNH

Dành cho sinh viên ngành: Công nghệ phần mềm, Mạng và truyền thông

Đà Lạt, 2010

MỤC LỤC

MỞ ĐẦU

Đồ họa máy tính là một trong những lĩnh vực hấp dẫn và phát triển nhanh của Công nghệ Thông tin. Nó được ra đời bởi sự kết hợp của 2 lĩnh vực thông tin và truyền hình, và được sử dụng rộng rãi trong hầu hết các ứng dụng như khoa học và công nghệ, y học, giáo dục, kiến trúc, và kể cả giải trí. Ngày nay, nhờ vào sự tiến bộ của khoa học kỹ thuật nên phần cứng và giá thành của máy tính càng lúc càng phù hợp, các kỹ thuật đồ họa được ứng dụng trong thực tế nhiều nên ngày càng có nhiều người quan tâm nghiên cứu đến lĩnh vực này.

Tuy nhiên, việc dạy và học kỹ thuật đồ họa mày tính thì không đơn giản vì chủ đề này có nhiều vần đề phức tạp, liên quan đến tin học và cả toán học. Hầu hết các giải thuật vẽ, tô màu cùng các phép biến hình đều được xây dựng dựa trên nền tảng của hình học không gian hai chiều và ba chiều.

Giáo trình Đồ họa máy tính này được xây dựng dựa trên kinh nghiệm giảng dạy đã qua và dựa trên tài liệu tham khảo chính là : "Donald Hearn, M. Pauline Baker**;** *Computer Graphics*; Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986".

Giáo trình Đồ họa máy tính là một môn học được giảng dạy cho sinh viên chuyên ngành Công nghệ Thông tin với 45 tiết lý thuyết và 30 tiết thực tập. Nội dung của giáo trình này gồm có 3 vấn đề chính như sau :

- Trình bày các thuật toán vẽ và tô các đường cơ bản như đường thẳng, đa giác, đường tròn, ellipse và các đường Bezier, B-Spline. Các thuật toán này giúp cho sinh viên có thể tự thiết kế để vẽ và tô màu một mô hình đồ họa .
- Nội dung thứ hai đề cập đến các phép biến đổi Affine, tìm giao các đối tượng, tô màu của đồ họa hai chiều.
- Nội dung thứ ba trình bày về quan sát, hiển thị và biến đổi Affine trên không gian ba chiều.

Trong quá trình biên soạn chắc không tránh khỏi thiếu sót, tôi xin trân trọng nhận được sự góp ý của các quý đồng nghiệp và sinh viên để giáo trình ngày càng được hoàn thiện hơn.

Chương 1

GIỚI THIỆU VỀ ĐỒ HỌA MÁY TÍNH

Nội dung chính

- Tổng quan về đồ họa máy tính.
- Các ứng dụng của đồ họa máy tính.
- Các thành phần cơ bản của hệ đồ họa máy tính.
- Hệ tọa độ thực và hệ tọa độ đồ họa.

1.1 Tổng quan đồ họa máy tính

Đồ họa máy tính bao gồm tất cả những gì liên quan đến việc sử dụng máy tính để phát sinh ra hình ảnh. Các vấn đề liên quan đến công việc này bao gồm: tạo, lưu trữ, thao tác trên các mô hình và các ảnh.

Ngày nay, hầu hết các chương trình soạn thảo, bảng tính sử dụng đồ họa trong giao diện với người dùng. Sự phát triển của đồ họa máy tính ngày càng rộng rãi với các chế độ đồ họa hai chiều (2D) và 3 chiều (3D), và cao hơn, nó phục vụ trong các lĩnh vực xã hội học khác nhau như khoa học, giáo dục, y học, kỹ thuật, thương mại và giải trí. Tính hấp dẫn và đa dạng của đồ họa máy tính có thể được minh họa rất trực quan thông qua việc khảo sát các ứng dụng của nó.

Đồ họa máy tính được sử dụng rất rộng rãi vì có đến 80% các ứng dụng liên quan đến hình ảnh và được ứng dụng trong nhiều lĩnh vực khác nhau như công nghiệp, thương mại, quản lý, giáo dục, giải trí, …v.v. Số lượng các chương trình đồ họa ứng dụng rất lớn và phát triển liên tục. Sau đây là một số ứng dụng tiêu biểu của đồ họa trong thực tế:

- *Hỗ trợ thiết kế - CAD/CAM (Computer-Aided Design/ Computer-Aided Manufacturing)*: Các hệ thống thiết kế và chế tạo với sự trợ giúp của máy tính được ứng dụng trong các lĩnh vực như phân tích thiết kế kết cấu xây dựng, công nghiệp điện tử, công nghiệp thời trang, các ngành công nghiệp chế tạo ôtô, máy bay, xe máy....

- *Đồ thị và bản đồ (Graphs and Charts)*: Đây là ứng dụng chủ yếu trong lĩnh vực đồ họa minh họa, ứng dụng này cho phép hiển thị các biểu đồ dữ liệu cũng như trong lĩnh vực biểu diễn và xử lý đồ họa. Một trong số những ứng dụng hiện nay là hệ thống thông tin địa lí GIS (Geographical Information System).:
- *Giải trí:* Với sự hỗ trợ đồ họa hiện nay chúng ta có thể sản xuất nhiều sản phẩm phục vụ cho lĩnh vực giải trí đặc biệt là phim hoạt hình và các trò chơi trên máy tính. Nhiều phần mềm và ngôn ngữ lập trình hỗ trợ ra đời cho phép ta tạo ra các hình ảnh động gần với với cuộc sống thực. Trong giáo trình này chúng ta sẽ làm quen với công cụ OpenGL.
- *Ứng dụng mô phỏng và thực tại ảo (Simulation and Virtual Reality)*: Bên cạnh việc hỗ trợ thiết kế kiến trúc và trong sản xuất công nghiệp, đồ họa máy tính còn có ứng dụng rất quan trọng trong mô phỏng các công trình kiến trúc, các di sản văn hóa, trong giảng dạy các môn học. Ứng dụng thực tại ảo là mức cao hơn của mô phỏng. Thực tại ảo áp dụng các kỹ thuật đồ họa kết hợp với các thiết bị 3D tạo ra các ứng dụng mô phỏng giống như thực nhưng được thực hiện trên máy tính như lái máy bay, bắn súng trong quân sự, giải phẫu trong y khoa, ….
- *Xử lý ảnh (Image Processing)*: Các kĩ thuật xử lý và thay đổi một bức ảnh có sẵn và được áp dụng trong nhiều lĩnh vực của đời sống. Ví dụ ta có thể sử dụng phần mềm để khôi phục một bức ảnh, phân tích các bức ảnh được chụp từ vệ tinh...
- *Kĩ thuật nhận dạng (Pattern Recognition)*: Đây là một lĩnh vực của kĩ thuật xử lí ảnh, các chuyên gia sẽ xây dựng một thư viện ảnh gốc bằng cách áp dụng các thuật toán phân tích và chọn lọc từ những ảnh mẫu có sẵn. Dựa trên thư viện đó các chuyên gia có thể phân tích và tổ hợp ảnh
- *Giao diện đồ họa người dùng (Graphical User Interface-GUI)*: Rất nhiều phần mềm ứng dụng ngày nay cung cấp GUI cho người dùng. Thành phần chính của một giao diện đồ họa đó là chương trình quản lí cửa sổ cho phép người sử dụng hiển thị nhiều cửa sổ người ta gọi đó là các cửa sổ hiển thị. Nhờ có GUI mà người sử dụng có thể dễ dàng thiết kế giao diện cho các chương trình ứng dụng.

1.2 Các thành phần cơ bản của hệ đồ họa máy tính

Để phát triển hệ thống đồ họa máy tính ta cần phải trang bị cả phần cứng lẫn phần mềm cũng như các ứng dụng khác. Trong đó, các thiết bị phần cứng là tùy thuộc vào từng ứng dụng đồ họa cụ thể mà có thể cần thiết hoặc không cần thiết.

Phần cứng

- Thiết bị thu nhận: lấy dữ liệu đầu vào cho ứng dụng đồ họa như bàn phím, chuột, máy quét, camera, ...
- Thiết bị hiển thị: hiển thị hình ảnh của ứng dụng đồ họa như các loại màn hình CRT, LCD, …
- Thiết bị tương tác: làm giao tiếp trung gian giữa người dùng và các ứng dụng đồ họa thực tại ảo, tạo cảm giác người dùng giống như thao tác trực tiếp trong môi trường thế giới thực như găng tay, kính 3D, …

Phần mềm

Phần mềm đồ họa có thể phân thành 2 loại: các công cụ lập trình và các trình ứng dụng đồ họa phục vụ cho một mục đích nào đó. Các công cụ lập trình cung cấp một tập các thư viện đồ họa có thể được dùng trong các ngôn ngữ lập trình cấp cao như Pascal, C/C++/C#, Java, … hay thậm trí có cả một thư viên đồ họa có thể nhúng vào các ngôn ngữ lập trình cấp bất kỳ như OpenGL, DirectX. Các hàm cơ sở của nó bao gồm việc tạo các đối tượng cơ sở của hình ảnh như đoạn thẳng, đa giác, đường tròn, … thay đổi màu sắc, chọn khung nhìn, biến đổi affine, …

Để phát triển các ứng dụng đồ họa máy tính cần có các loại phần mềm sau:

- Tạo mô hình: 3DS Max, Maya, …
- Lập trình, phát triển ứng dụng: OpenGL, DirectX, …

1.3 Hệ tọa độ thế giới thực, hệ tọa độ thiết bị và hệ tọa độ chuẩn

Một hệ đồ họa bao gồm 3 miền như sau:

- Miền điều khiển : bao bọc toàn bộ hệ thống.

- Miền thực : nằm trong miền điều khiển. Khi một giá trị nằm trong miền thực, nó sẽ được chuyển thành số thực dấu phẩy động, và khi có một số rời khỏi miền này thì nó sẽ được chuyển thành số nguyên.
- Miền hiển thị : nằm trong miền điều khiển nhưng phân biệt với miền thực. Chỉ có giá trị số nguyên mới nằm trong miền hiển thị.

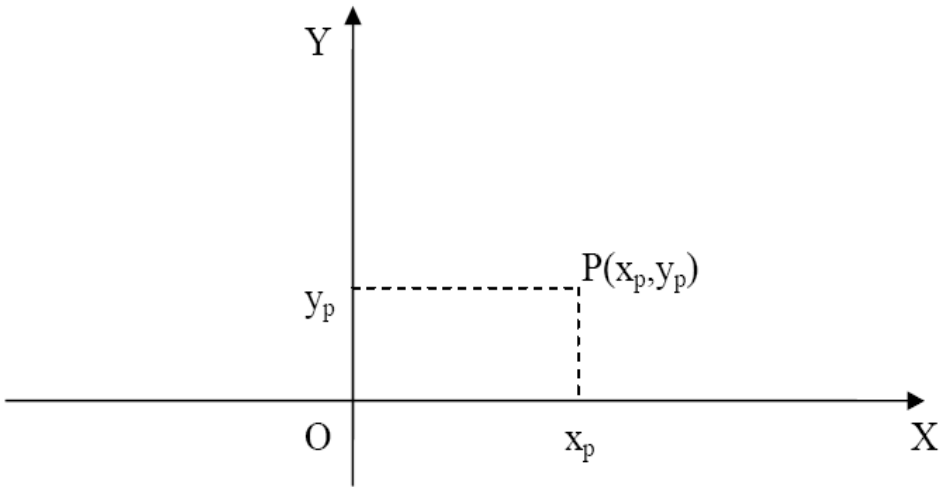
Trong lĩnh vực kỹ thuật đồ họa, chúng ta phải hiểu được rằng thực chất của đồ họa là làm thế nào để có thể mô tả và biến đổi được các đối tượng trong thế giới thực trên máy tính. Các đối tượng trong thế giới thực được mô tả bằng tọa độ trong miền thực. Trong khi đó, hệ tọa độ thiết bị lại sử dụng hệ tọa độ nguyên để hiển thị các hình ảnh. Đây chính là vấn đề cơ bản cần giải quyết. Ngoài ra, còn có một khó khăn khác nữa là với các thiết bị khác nhau thì có các đặc trưng về thông số kỹ thuật khác nhau. Do đó, cần có một phương pháp chuyển đổi tương ứng giữa các hệ tọa độ và đối tượng để có thể mô tả gần đúng với hình ảnh thực bên ngoài.

Hai mô hình cơ bản của ứng dụng đồ họa là dựa trên mẫu số hóa và dựa trên đặc trưng hình học. Trong ứng dụng đồ họa dựa trên mẫu số hóa thì các đối tượng đồ họa được tạo ra bởi lưới các pixel rời rạc. Các pixel này có thể được tạo ra bằng các chương trình vẽ, máy quét, ... Các pixel này mô tả tọa độ xác định vị trí và giá trị mẫu. Thuận lợi của ứng dụng này là dễ dàng thay đổi hình ảnh bằng cách thay đổi màu sắc hay vị trí của các pixel, hoặc di chuyển vùng ảnh từ nơi này sang nơi khác. Tuy nhiên, điều bất lợi là không thể xem xét đối tượng từ các góc nhìn khác nhau.

Ứng dụng đồ họa dựa trên đặc trưng hình học bao gồm các đối tượng đồ họa cơ sở như đoạn thẳng, đa giác, ...v.v. Chúng được lưu trữ bằng các mô hình và các thuộc tính. Chẳng hạn, đoạn thẳng được mô hình bằng hai điểm đầu và cuối, có thuộc tính như màu sắc, độ dày. Người sử dụng không thao tác trực tiếp trên các pixel mà thao tác trên các thành phần hình học của đối tượng.

Hệ tọa độ thế giới thực

Hệ tọa độ thực thường được dùng để mô tả các đối tượng trong thế giới thực là hệ tọa độ Descartes. Trong hệ tọa độ này, mỗi điểm P được biểu diễn bởi một cặp tọa độ (x_p, y_p) với $x_p, y_p \in R$ (xem hình 1.1).



Hình 1.1 Hệ tọa độ thực

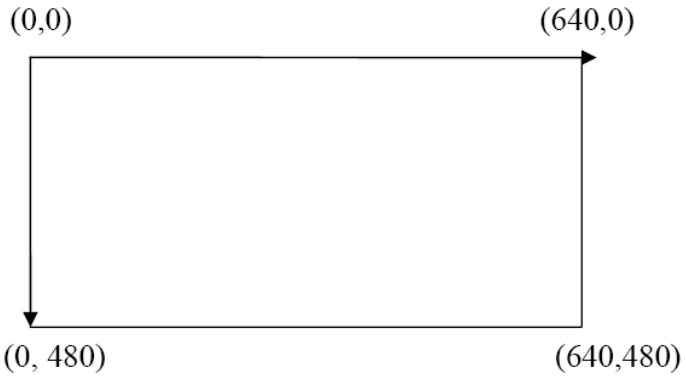
Trong đó :

- O*x* : trục hoành.
- O*y* : trục tung.
- x_p : hoành độ điểm *P*.
- y_p : tung độ điểm *P*.

Hệ tọa độ thiết bị

Hệ tọa độ thiết bị được dùng cho một thiết bị xuất cụ thể nào đó, ví dụ như máy in, màn hình, ...v.v. Trong hệ tọa độ thiết bị thì các điểm cũng được mô tả bởi cặp tọa độ (*x,y*). Tuy nhiên, khác với hệ tọa độ thực là $x, y \in \mathbb{N}$. Điều này có nghĩa là các điểm trong hệ tọa độ thực được định nghĩa liên tục, còn các điểm trong hệ tọa độ thiết bị là rời rạc. Ngoài ra, các tọa độ *x, y* của hệ tọa độ thiết bị chỉ biểu diễn được trong một giới hạn nào đó của $\mathbb{N}$.

Ví dụ : Độ phân giải của màn hình trong chế độ đồ họa là 640x480. Khi đó, $x \in (0,640)$ và $y \in (0, 480)$ (xem hình 1.2).



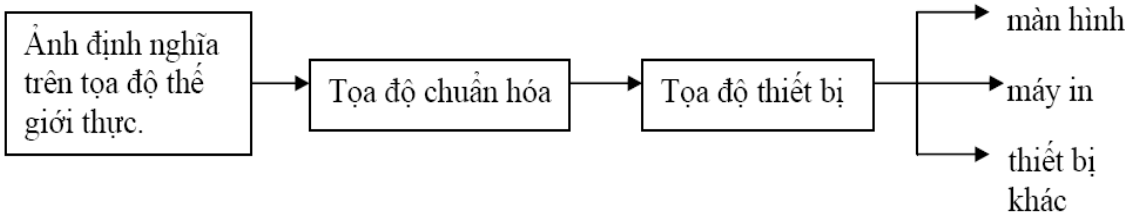
Hệ tọa độ màn hình

Hệ tọa độ thiết bị chuẩn

Do cách định nghĩa các hệ tọa độ thiết bị khác nhau nên hình ảnh hiển thị chính xác trên thiết bị này thì chưa chắc hiển thị chính xác trên thiết bị khác. Người ta xây dựng một hệ tọa độ thiết bị chuẩn đại diện chung cho tất cả các thiết bị để có thể mô tả các hình ảnh mà không phụ thuộc vào bất kỳ thiết bị nào.

Trong hệ tọa độ chuẩn, các tọa độ *x, y* sẽ được gán các giá trị trong đoạn từ [0,1]. Như vậy, vùng không gian của hệ tọa độ chuẩn chính là hình vuông đơn vị có góc trái dưới (0, 0) và góc phải trên là (1, 1).

Quy trình hiển thị các đối tượng thực như sau (xem hình 1.3):



Hình 1.3 Hệ tọa độ thiết bị

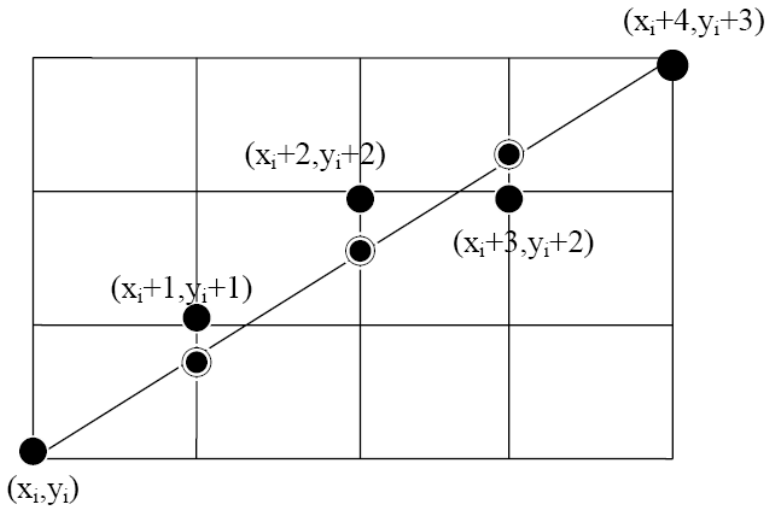
Chương 2

CÁC THUẬT TOÁN VẼ ĐỐI TƯỢNG ĐỒ HỌA CƠ BẢN

Nội dung chính

- Các thuật toán vẽ đoạn thẳng: DDA, Bresenham, MidPoint.
- Thuật toán MidPoint vẽ đường tròn, ellipse.
- Vẽ đường cong tham số Bezier, B-Spline.

2.1 Thuật toán vẽ đoạn thẳng



Hình 2.1: Các điểm gần đoạn thẳng thực

Xét đoạn thẳng có hệ số góc $m \in (0, 1]$ và $\Delta x > 0$. Với các đoạn thẳng dạng này, nếu (x_i, y_i) là điểm đã được xác định ở bước thứ i thì điểm kế tiếp (x_{i+1}, y_{i+1}) ở bước thứ $i+1$ sẽ là một trong hai điểm sau:

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i + 1 \\ y_i \end{cases} \end{cases}$$

Vấn đề đặt ra là chọn điểm vẽ như thế nào để đoạn thẳng được vẽ gần với đoạn

thẳng thực nhất và tối ưu hóa về mặt tốc độ, thời gian thực.

2.1.1 Thuật toán DDA (Digital DifferentialAnalyzer)

DDA (hay còn gọi là thuật toán số gia) là thuật toán vẽ đoạn thẳng xác định các điểm dựa vào hệ số góc của phương trình đường thẳng $y = m.x + b$. Trong đó, $m = \Delta y/\Delta x$, $\Delta y = y_{i+1} - y_i$, $\Delta x = x_{i+1} - x_i$. Nhận thấy trong hình vẽ 2.1 thì tọa độ của điểm x sẽ tăng 1 đơn vị trên mỗi điểm vẽ, còn việc quyết định chọn y_i là $y_i + 1$ hay y_i sẽ phụ thuộc vào giá trị sau khi làm tròn của tung độ y. Tuy nhiên, nếu tính trực tiếp giá trị thực của y ở mỗi bước từ phương trình $y = m.x + b$ thì cần một phép toán nhân và một phép toán cộng số thực:

$$y_{i+1} = m.x_{i+1} + b = m(x_i + 1) + b = m.x_i + b + m$$

Để tối ưu tốc độ, người ta khử phép nhân trên số thực.

Ta có : $y_i = m.x_i + b$

$\Rightarrow y_{i+1} = y_i + m$

- Tóm lại, khi $0 < m \leq 1$thì:

$x_{i+1} = x_i + 1$

$y_{i+1} = y_i + m$

- Trường hợp $m > 1$: chọn bước tăng trên trục y một đơn vị.

$x_{i+1} = x_i + \frac{1}{m}$

$y_{i+1} = y_i + 1$

Hai trường hợp này dùng để vẽ một điểm bắt đầu từ bên trái đến điểm cuối cùng bên phải của đường thẳng (xem hình 2.2). Nếu điểm bắt đầu từ bên phải đến điểm cuối cùng bên trái thì xét ngược lại :

- $0 < m \leq 1$: $x_{i+1} = x_i - 1$

$y_{i+1} := y_i - m$

- $m > 1$: $x_{i+1} = x_i - \frac{1}{m}$

  $y_{i+1} = y_i - 1$

Hình 2.2 : Hai trường hợp m >1 và 0 < m < 1

Cài đặt minh họa thuật toán DDA

```
void DDALine(int x0, int y0, int x1, int y1)
{
        int x;
        float dx, dy, y, m;
        dx:= x1 – x0;
        dy:= y1 – y0;
        m:= dy/dx;
        y = y0;
        for (x=x0; x <= x1; x++)
        {
                glVertex2i(x, Round(y));
```

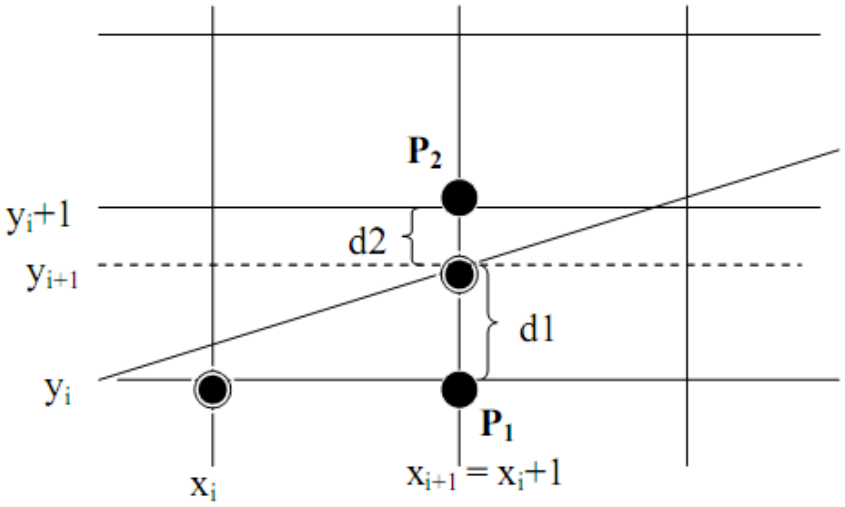
$y = y+m$

}

}

Tương tự, ta có thể tính toán các điểm vẽ cho trường hợp $m < 0$, $|m| \leq 1$ hoặc $|m| > 1$.

2.1.2 Thuật toán Bresenham



Hình 2.3 : Thuật toán Bresenham vẽ đoạn thẳng có $0 \leq m \leq 1$.

Gọi $(x_i + 1, y_{i+1})$ là điểm thuộc đoạn thẳng (xem hình 2.3). Ta có $y = m(x_i + 1) + b$.

Đặt $d_1 = y_{i+1} - y_i$; $d_2 = (y_i + 1) - y_{i+1}$

Việc chọn điểm (x_{i+1}, y_{i+1}) là P_1 hay P_2 phụ thuộc vào việc so sánh d_1 và d_2 hay dấu của $d_1 - d_2$:

- Nếu $d_1 - d_2 < 0$: chọn điểm P_1, tức là $y_{i+1} = y_i$
- Nếu $d_1 - d_2 \geq 0$: chọn điểm P_2, tức là $y_{i+1} = y_i + 1$

Xét $P_i = \Delta x(d_1 - d_2)$

Ta có : $d_1 - d_2 = 2y_{i+1} - 2y_i - 1$

$= 2m(x_i+1) + 2b - 2y_i - 1$

$$\Rightarrow \quad P_i = \Delta x(d_1 - d_2) = \Delta x[2m(x_i+1) + 2b - 2y_i - 1]$$

$$= \Delta x[2(\Delta y/\Delta x)(x_i+1) + 2b - 2y_i - 1]$$

$$= 2\Delta y(x_i+1) - 2\Delta x.y_i + \Delta x(2b - 1)$$

$$= 2\Delta y.x_i - 2\Delta x.y_i + 2\Delta y + \Delta x(2.b - 1)$$

Vậy $C = 2\Delta y + \Delta x(2b - 1) = \text{Const}$ (hằng số)

$$\Rightarrow P_i = 2\Delta y.x_i - 2\Delta x.y_i + C$$

Nhận xét rằng nếu tại bước thứ i ta xác định được dấu của P_i thì xem như ta xác định được điểm cần chọn ở bước $(i + 1)$. Ta có :

$$P_{i+1} - P_i = (2\Delta y.x_{i+1} - 2\Delta x.y_{i+1} + C) - (2\Delta y.x_i - 2\Delta x.y_i + C)$$

$$\Leftrightarrow \quad P_{i+1} = P_i + 2\Delta y - 2\Delta x(y_{i+1} - y_i)$$

- Nếu $P_i < 0$: chọn điểm P_1, tức là $y_{i+1} = y_i$ và $P_{i+1} = P_i + 2\Delta y$.

- Nếu $P_i \geq 0$: chọn điểm P_2, tức là $y_{i+1} = y_i + 1$ và $P_{i+1} = P_i + 2\Delta y - 2\Delta x$

- Giá trị P_0 được tính từ điểm vẽ đầu tiên (x_0, y_0) theo công thức :

$$P_0 = 2\Delta y.x_0 - 2\Delta x.y_0 + C$$

Do (x_0, y_0) là điểm nguyên thuộc về đoạn thẳng nên ta có :

$$y_0 = 2m.x_0 + b = \frac{\Delta y}{\Delta x} x_0 + b$$

Thế vào phương trình trên ta được :

$$P_0 = 2\Delta y - \Delta x$$

Cài đặt minh họa thuật toán Bresenham

void Bresenham_Line (int x1,int y1,int x2,int y2)

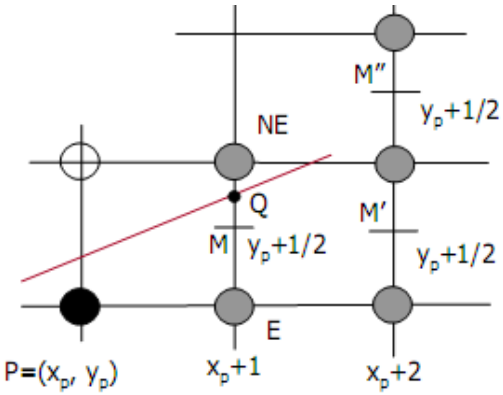
```
{
    int dx, dy, x, y, P, incre1, incre2;
    dx  = x2 - x1;  dy  = y2 - y1;
    P = 2*dy - dx;
    incre1 = 2*dy ; incre2  = 2*(dy - dx) ;
    x= x1; y=y1;
    glVertex2i(x, y);
    while (x < x2 )
    {
        x  = x +1 ;
        if (P < 0)      P = P + incre1
        else
        {
            y  = y+1 ;
            P  = P + incre2
        }
        glVertex2i(x, y);
    }
}
```

Nhận xét

- Thuật toán Bresenham chỉ thao tác trên số nguyên và chỉ tính toán trên phép cộng và phép nhân 2. Điều này là một cải tiến làm tăng tốc độ đáng kể so với thuật toán DDA.
- Ý tưởng chính của thuật toán này là ở chỗ xét dấu Pi để quyết định điểm kế tiếp, và sử dụng công thức truy hồi $P_{i+1} - P_i$ để tính P_i bằng các phép toán đơn giản trên số nguyên.
- Tuy nhiên, việc xây dựng trường hợp tổng quát cho thuật toán Bresenham có phức tạp hơn thuật toán DDA.

2.1.3 Thuật toán MidPoint

Pitteway công bố thuật toán MidPoint vào 1967, Van Aken cải tiến 1984. Xét hệ số góc thuộc [0, 1]. Giả thiết rằng đã chọn *P* để vẽ, xác định pixel tiếp theo sẽ là tại *N* hay *NE* (xem hình 2.4). Giao của đường thẳng với X_p+1 tại *Q*, *M* là trung điểm của *NE* và *E*.



Hình 2.4: Thuật toán MidPoint vẽ đoạn thẳng

Ý tưởng của thuật toán MidPoint là xét điểm *M* xem nằm phía nào của đường thẳng, nếu *M* nằm phía trên đường thẳng thì chọn *E* (tức là đường thẳng gần với E hơn *NE*), ngược lại chọn *NE*. Vì vậy, ta cần xác định vị trí tương đối của *M* so với đường thẳng chứa đoạn thẳng cần vẽ.

- Phân tích thuật toán vẽ đoạn thẳng dựa trên phương trình dạng tổng quát của

đường thẳng chứa đoạn thẳng: $F(x, y)= a.x + b.y + c$

Ta có: $y - \frac{dy}{dx}.x + D$

Suy ra dạng tổng quát: $F(x,y) = \frac{dy}{dx}.x + B - y = 0$. Hay tương đương: $F(x,y) = dy.x - dx.y + B.dx = 0$.

Từ đó, ta có các hệ số của phương trình dạng tổng quát là :

$$a = dy, \quad b = -dx, \quad c = B.dx$$

- Giá trị hàm tại *M*: $F(M)=F(x_p + 1, y_p + \frac{1}{2}) = d$
  - Nếu $d > 0$, *M* nằm dưới đường thẳng thì chọn *NE*.
  - Nếu $d < 0$, *M* nằm phía trên thì chọn *E*.
  - Nếu $d = 0$, chọn *E* hay *NE* tùy ý.
- Giá trị của hàm tại *M* của của điểm tiếp theo sẽ vẽ
  - Gọi giá trị *d* vừa tính là:

$$d_{old} = a(x_p + 1) + b\left(y_p + \frac{1}{2}\right) + c$$

  - Giả sử vừa chọn E:

$$d_{new} - F\left(x_p + 2, y_p + \frac{1}{2}\right) - a(x_p + 2) + b\left(y_p + \frac{1}{2}\right) + c$$

$d_{new} = d_{old} + a = d_{old} + dy \rightarrow dy$ là số gia của điểm tiếp theo.

  - Giả sử vừa chọn NE:

$$d_{new} = F\left(x_p + 2, y_p + \frac{3}{2}\right) = a(x_p + 2) + b\left(y_p + \frac{3}{2}\right) + c$$

$d_{new} = d_{old} + a + b = d_{old} + (dy - dx) \rightarrow (dy - dx)$ là số gia của điểm tiếp theo.

Tính giá trị khởi đầu của *d* tại các trung điểm

- Giả sử vẽ đoạn thẳng từ (x_0, y_0) đến (x_1, y_1), từ đó trung điểm thứ nhất có tọa độ $(x_0 + 1, y_0 + \frac{1}{2})$. Suy ra:

$$F\left(x_0 + 1, y_0 + \frac{1}{2}\right) = a(x_0 + 1) + b\left(y_0 + \frac{1}{2}\right) + c$$

$$= a.x_0 + b.y_0 + c + a + \frac{b}{2} = F(x_0, y_0) + a + \frac{b}{2}$$

- $F(x_0, y_0) = 0 \rightarrow d_{start} = a + \frac{b}{2} = dy - \frac{dx}{2}$
- Tránh số thập phân của d_{start}, định nghĩa lại hàm như sau:

$$F(x, y)=2(a.x + b.y + c)$$

- Do vậy, ta có:

$$d_{start} = 2dy - dx; \qquad \Delta E = 2dy; \qquad \Delta NE = 2(dy - dx)$$

Cài đặt minh họa thuật toán MidPoint

```
void MidPoint_Line(int x0, int y0, int x1, int y1, int color)
{
    int dx, dy, x, y, d, incrE, incrNE ;
    dx = x1 – x0;
    dy = y1 – y0;
    d = 2*dy - dx;
    incrE = 2*dy;
    incrNE = 2*(dy - dx);
    x = x0;
    y =y0;
```

```
        glVertex2i(x, y);
        while (x<x1)
        {
            if (d<=0)
            {   //chọn E
                d:= d+incrE;
                x = x+1
            }
            else
            {   //chọn NE
                d = d+incrNE;
                x =x+1;
                y =y+1
            }
            glVertex2i(x, y);
        }
    }
```

Nhận xét

- Các thuật DDA, MidPoint trình bày xây dựng thuật toán vẽ đoạn thẳng trong trường hợp hệ số góc thuộc đoạn [0, 1]. Các trường hợp còn lại phân tích tương tự đối với từng thuật toán.

- Có một tính chất đối xứng có thể áp dụng để vẽ đoạn thẳng trong các trường hợp hệ số góc không thuộc [0, 1] mà không phụ thuộc vào thuật toán. Điều này có nghĩa là ta sẽ lấy đối xứng các đoạn thẳng này về trường hợp thuộc đoạn [0,1], tính toán xong mỗi tọa độ (*x, y*) ta lại lấy đối xứng trở lại rồi vẽ.
- Sau đây là chương trình cài đặt thuật toán DDA tổng quát cho tất cả các trường hợp theo phương pháp lấy đối xứng :

```
void LineDDA_DX(int x1, int y1, int x2, int y2)
{
        if (x2 < x1)
        {
           int t = x2;
           x2 = x1;
           x1 = t;

           t = y2;
           y2 = y1;
           y1 = t;
        }

        double m;
        int dx = x2 - x1;
        int dy = y2 - y1;
        m = (double)dy / (double)dx;

        int d;
        if (m > 1)
        {
           d = 1;

           int temp = x1;
           x1 = y1;
           y1 = temp;

           temp = x2;
           x2 = y2;
           y2 = temp;

           dy = y2 - y1;
           dx = x2 - x1;
           m = (double)dy / (double)dx;
        }
        else if (m > 0)
        {
           d = 2;
        }
```

```
else if (m > -1)
{
   d = 3;
   y1 = -y1;
   y2 = -y2;

   dy = y2 - y1;
   dx = x2 - x1;
   m = (double)dy / (double)dx;
}
else
{
   d = 4;
   int temp2 = x1;
   x1 = -y1;
   y1 = temp2;

   temp2 = x2;
   x2 = -y2;
   y2 = temp2;

   dy = y2 - y1;
   dx = x2 - x1;
   m = (double)dy / (double)dx;
}

int x;
double y;
y = y1;
for (x = x1; x <= x2; x++)
{
   if (d == 1)
   {
      glVertex2i(Round(y), x);
   }
   else if (d == 2)
   {
      glVertex2i(x, Round(y));
   }
   else if (d == 3)
   {
      glVertex2i(x, -Round(y));
   }
   else // d==4
   {
      glVertex2i(Round(y), -x);
   }
   y += m;
```

}

}

2.2 Thuật toán MidPoint vẽ đường tròn

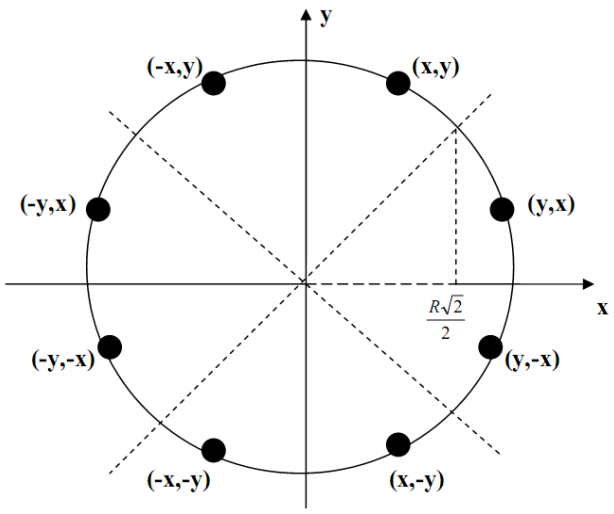
Trong hệ tọa độ Descartes, phương trình đường tròn bán kính R có dạng:

- Với tâm $O(0,0)$: $x^2 + y^2 = R^2$
- Với tâm $C(x_c, y_c)$: $(x - x_c)^2 + (y - y_c)^2 = R^2$

Trong hệ tọa độ cực :

- $x = x_c + R.\cos\theta$
- $y = y_c + Y.\sin\theta$

với $\theta \in [0, 2\pi]$.



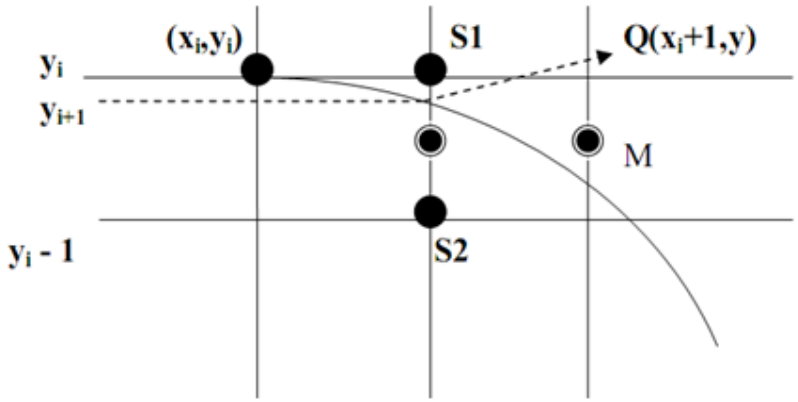
Hình 2.5: Đối xứng 8 điểm trong đường tròn

Do tính đối xứng của đường tròn C (xem hình 2.5) nên ta chỉ cần vẽ 1/8 cung tròn, sau đó lấy đối xứng qua 2 trục tọa độ và 2 đường phân giác thì ta vẽ được cả đường tròn.

Thuật toán MidPoint đưa ra cách chọn y_{i+1} là y_i hay y_{i-1} bằng cách so sánh điểm

thực Q(x_{i+1}, y) với điểm giữa M là trung điểm của $S1$ và $S2$. Chọn điểm bắt đầu để vẽ là (0, R). Giả sử (x_i, y_i) là điểm nguyên đã tìm được ở bước thứ i (xem hình 2.6), thì điểm (x_{i+1}, y_{i+1}) ở bước i+1 là sự lựa chọn giữa $S1$ và $S2$.

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} - \begin{cases} y_i - 1 \\ y_i \end{cases} \end{cases}$$



Hình 2.6 : Đường tròn với điểm Q(x +1, y) và điểm MidPoint.

Đặt $F(x, y) = x^2 + y^2 - R^2$, ta có :

- $F(x, y) < 0$, nếu điểm (x, y) nằm trong đường tròn.
- $F(x, y) = 0$, nếu điểm (x, y) nằm trên đường tròn.
- $F(x, y) > 0$, nếu điểm (x, y) nằm ngoài đường tròn.

Xét $P_i = F(M) = F(x_i + 1, y - \frac{1}{2})$. Ta có :

- Nếu $P_i < 0$: điểm M nằm trong đường tròn. Khi đó, điểm thực Q gần với điểm $S1$ hơn nên ta chọn $y_{i+1} = y_i$.
- Nếu $P_i >= 0$: điểm M nằm ngoài đường tròn. Khi đó, điểm thực Q gần với điểm $S2$ hơn nên ta chọn $y_{i+1} = y_i - 1$.

Mặt khác :

$$P_{i+1} - P_i = F(x_{i+1}+1, y_{i+1} - \frac{1}{2}) - F(x_i + 1, y_i - \frac{1}{2})$$

$= [(x_{i+1}+1)^2 + (y_{i+1} - \frac{1}{2})^2 - R^2] - [(x_i+1)^2 + (y_i - \frac{1}{2})^2 - R^2]$

$= 2x_i + 3 + ((y_i+1)^2 + (y_i)^2) - (y_{i+1} - y_i)$

Vậy :

- Nếu $P_i < 0$: chọn $y_{i+1} = y_i$. Khi đó, $P_{i+1} = P_i + 2x_i + 3$
- Nếu $P_i >= 0$: chọn $y_{i+1} = y_i - 1$. Khi đó, $P_{i+1} = P_i + 2x_i - 2y_i + 5$.
- P_i ứng với điểm ban đầu $(x_0, y_0) = (0, R)$ là:

$$P_0 = F(x_0 + 1, y_0 - \frac{1}{2}) = F(1, R - \frac{1}{2}) = \frac{5}{4} - R$$

- Để rút gọn biểu thức trên và tránh việc tính toán số thực, ta đặt $P'_0 = P_0 - \frac{1}{4} = 1 - R$. Ta có nhận xét rằng dấu của P'_0 không thay đổi trong thuật toán MidPoint.

Cài đặt minh họa thuật toán MidPoint vẽ dường tròn

```
void Ve_doi_xung_8diem(int xc, int yc, int x, int y)
{
        glVertex2i(x + xc, y + yc);
        glVertex2i(y + xc, x + yc);
        glVertex2i(-x + xc, -y + yc);
        glVertex2i(-y + xc, -x + yc);
        glVertex2i(-x + xc, y+ yc);
        glVertex2i(-y + xc, x + yc);
        glVertex2i(x + xc, -y + yc);
        glVertex2i(y + xc, -x + yc);
```

```
 }
void MidPoint_Circle(int xc, int yc, int r);
{
      int  x, y, p;
      x=0 ;
     y=r;
      p=1 - r;
      while ( y > x)
     {
                Ve_doi_xung_8diem(xc, yc, x, y);
                if (p < 0)       p=p + 2*x + 3
                else
                {
                      p = p + 2*(x - y) + 5 ;
                      y =y - 1;
                 }
                x = x + 1;
      }
}
```

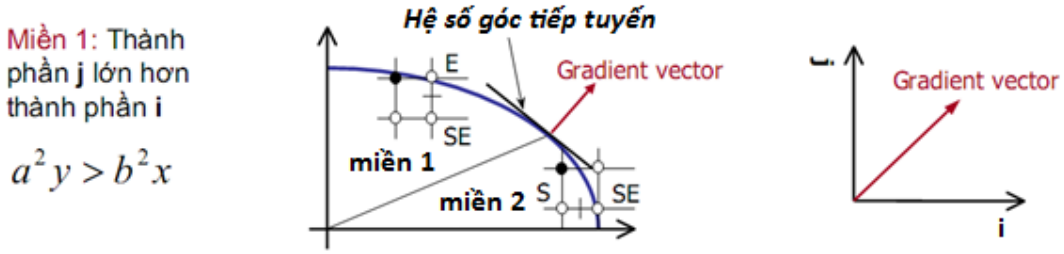
2.3 Thuật toán MidPoint vẽ Ellipse

Xét phương trình elíp có tâm tại gốc tọa độ

$$F(x,y) = b^2x^2 + a^2y^2 - a^2b^2 = 0$$

Áp dụng thuật toán MidPoint vẽ đường tròn để vẽ elíp. Tính đối xứng của elíp là khi biết tọa độ một điểm có thể dễ dàng suy ra tọa độ ba điểm đối xứng với nó qua các trục Ox, Oy và góc tọa độ.

Vì elíp chỉ có tính chất đối xứng bốn điểm nên ta phải vẽ một phần tư của elíp, sau đó mới lấy đối xững để vẽ các phần còn lại của elíp. Đầu tiên ta tìm ranh giới hai miền trong góc phần tư thứ nhất của elíp



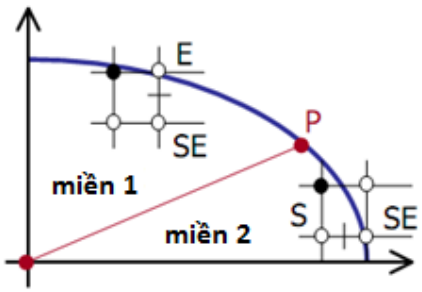
Hình 2.7: Phân chia hai miền của elíp

- **Vị trí:** Điểm *P* là tiếp điểm của tiếp tuyến có hệ số góc –1
- **Xác định:** Véc tơ vuông góc với tiếp tuyến tại tiếp điểm, giá trị gradient:

$$gradF(x,y) = \frac{\partial F}{\partial x}i + \frac{\partial F}{\partial y}j = 2b^2.x.i + 2a^2y.j = 0$$

- Tại *P* các thành phần i và j của véc tơ gradient có cùng độ lớn.

Ý tưởng của thuật toán là đánh giá hàm tại trung điểm hai tọa độ pixel để chọn vị trí tiếp theo để vẽ. Dấu của nó cho biết trung điểm nằm trong hay ngoài elíp.



Hình 2.8: Phân tích vẽ hai miền của ellipse

Xét miền 1

- Tính biến quyết định tại trung điểm đầu tiên nếu điểm đang xét là x_p, y_p:

$$d = F(x, y) = F(x_p + 1, y_p - \frac{1}{2})$$

- Nếu $d < 0$: chọn E, x tăng 1, y không thay đổi.

$$d_{old} - F\left(x_p + 1, y_p - \frac{1}{2}\right) - b^2(x_p + 1)^2 + a^2\left(y_p - \frac{1}{2}\right)^2 - a^2b^2$$

$$d_{new} = F\left(x_p + 2, y_p - \frac{1}{2}\right) = b^2(x_p + 2)^2 + a^2\left(y_p - \frac{1}{2}\right)^2 - a^2b^2$$

$$d_{new} = d_{old} + b^2(2x_p + 3) = d_{old} + \Delta_E$$

- Nếu $d \geq 0$: chọn SE, x tăng 1, y giảm 1:

$$d_{new} = F\left(x_p + 2, y_p - \frac{3}{2}\right) = b^2(x_p + 2)^2 + a^2\left(y_p - \frac{3}{2}\right)^2 - a^2b^2$$

$$d_{new} = d_{old} + b^2(2x_p + 3) + a^2(-2y_p + 2) = d_{old} + \Delta_{SE}$$

Xét miền 2:

- Tương tự, tính biến quyết định $d = F(x, y) = F(x_p + \frac{1}{2}, y_p - 1)$
  - Nếu $d < 0$: chọn SE, x tăng 1, y giảm 1.
  - Nếu $d \geq 0$: chọn S, x không tăng, y giảm 1.
- Tìm số gia như miền 1, ta được:
  - $\Delta S = a^2(-2y_p + 3)$
  - $\Delta SE = b^2(2x_p + 2) + a^2(-2y + 3)$

Tìm giá trị khởi đầu của số gia d:

- **Miền 1:**
  - Giả sử a, b nguyên, điểm bắt đầu vẽ là $(0, b)$
  - Trung điểm thứ nhất: $(1, b - 1/2)$

$$F\left(1, b-\frac{1}{2}\right) = b^2 + a^2\left(b-\frac{1}{2}\right)^2 - a^2b^2 = b^2 + a^2\left(-b+\frac{1}{4}\right)$$
$$= b^2 - a^2b + \frac{a^2}{4}$$

- **Miền 2:** Phụ thuộc vào trung điểm ($x_p + 1$, $y_p - \frac{1}{2}$) của điểm tiếp theo điểm cuối cùng của miền 1.

$$F\left(x_p + \frac{1}{2}, y_p - 1\right) = b^2(x+\frac{1}{2})^2 + a^2(y-1)^2 - a^2b^2$$
$$= b^2x^2 + b^2x + \frac{b^2}{4} + a^2(y-1)^2 - a^2b^2$$

Cài đặt minh họa thuật toán MidPoint vẽ Elíp

```
void Ve_doi_xung_4diem(int xc, int yc, int x, int y)
{
        glVertex2i(x + xc, y + yc);
        glVertex2i(-x + xc, y + yc);
        glVertex2i(-x + xc, -y + yc);
        glVertex2i(-y + xc, x + yc);
 }
void ellipse(int xc, int yc, int a, int  b)
{
        int x, y;
        float d1, d2;
        x=0;   //{Khởi động}
        y=b;
        d1=b^2-a^2b+a^2/4;
        Ve_doixung_4diem(x, y);
        while (a^2(y-1/2)>b^2(x+1)) {Vùng 1}
        {
```

```
        If (d1<0)       {Chọn E}
        {
                d1=d1+b^2(2*x+3);
                x=x+1
        }
        else {Chọn SE}
        {
                d1= d1+b^2(2*x+3)+a^2(-2*y+2);
                x = x+1;
                y = y - 1 ;
        {
        Ve_doixung_4diem(x, y);
    }
    d2=b^2(x+1/2)^2+a^2(y-1)^2 –a^2b^2;
    while (y>0)   // {Vùng 2}
    {
        if (d2<0)       //{ Chon SE }
        {
                d2=d2+b^2(2*x+2)+a^2(-2*y+3);
                x=x+1;
                y=y-1
        }
        else
        {
                d2=d2+a^2(-2*y+3);
                y=y-1
        }
        Ve_doixung_4diem(x, y);
    }
}
```

2.4. Đường cong tham số

2.4.1. Đường cong Bezier

2.4.1.1. Thuật toán de Casteljau

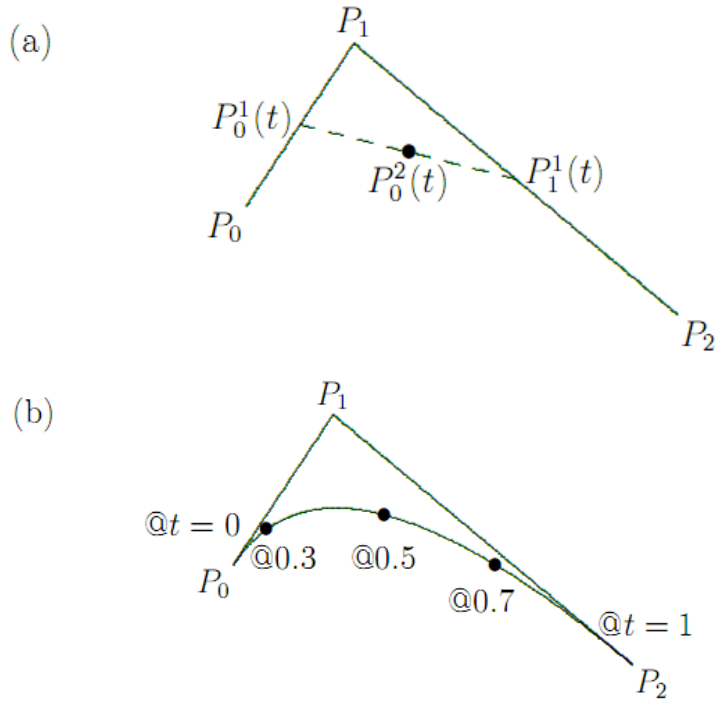
Thuật toán de Casteljau dựa trên dãy các điểm điều khiển để xây dựng với giá trị t trong đoạn [0, 1] tương ứng với một điểm $P(t)$. Do đó, thuật toán sinh ra một dãy các điểm từ các điểm điều khiển cho trước. Khi các điểm điều khiển thay đổi, đường cong sẽ thay đổi theo. Cách xây dựng đường cong dựa trên phép nội suy tuyến tính và do đó rất dễ dàng giao tiếp. Ngoài ra, phương pháp này cũng đưa ra nhiều tính chất quan trọng của đường cong.

Parabol dựa trên ba điểm

Trong mặt phẳng $\boldsymbol{R^2}$ xét ba điểm P_0, P_1, P_2. Đặt

$$P_0^1(t) := (1-t)P_0 + tP_1$$
$$P_1^1(t) := (1-t)P_1 + tP_2$$

Trong đó, $t \in [0, 1]$. Nói cách khác, với mỗi $t \in [0, 1]$, các điểm $P_0^1(t)$, $P_1^1(t)$ nằm trên các đoạn thẳng P_0P_1 và P_1P_2 tương ứng.



Hình 2.9: Đường cong Bezier xác định bởi ba điểm điều khiển

Lặp lại phép nội suy tuyến tính trên các điểm mới $P_0^1(t)$ và $P_1^1(t)$ ta được:

$$P_0^2(t) = (1-t)P_0^1(t) + tP_1^1(t)$$

Quỹ tích của $P(t) := P_0^2(t)$ khi t thay đổi trong đoạn [0, 1] sẽ cho ta đường cong như trên hình (b).

Dễ dàng suy ra

$$P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

Suy ra $P(t)$ là đường cong parabol theo biến t.

Ví dụ: Phương trình đường cong Bezier $P(t)$ tương ứng ba điểm điều khiển $P_0(1, 0)$, $P_1(2, 2)$, $P_2(6, 0)$ là:

$$P(t) = (1-t)^2 P_0 + 2t(1-t)P_1 + t^2 P_2$$

$$= (1-t)^2(1,0) + 2t(1-t)(2,2) + t^2(6,0)$$

$$= (3t^2 + 2t + 1, -4t^2 + 4t)$$

Tổng quát cho trường hợp số điểm điều khiển ≥ 3 ta có:

Thuật toán de Casteljau cho L + 1 điểm điều khiển

Trong mặt phẳng $\mathbf{R}^2$ xét $L+1$ điểm P_0, P_1,..., P_L. Với mỗi giá trị t cho trước, ta xây dựng theo quy nạp đường cong $P_0^L(t)$ như sau:

Bước 1: [Khởi tạo] Đặt $r = 0$ và $P_i^r(t) := P_i$ với mọi $i=0, 1, \ldots, L-r$.

Bước 2: [Kết thúc?] Nếu $r = L$ dừng; ngược lại đặt

$$P_i^{r+1}(t) = (1-t)P_i^r(t) + tP_{i+1}^r(t)$$

Bước 3: Thay r bởi $r+1$ và chuyển sang bước 2.

Cài đặt minh họa thuật toán Casteljau

Point Casteljau(float t)

```
{
    Point Q[Max];
    int i, r;
    for (i = 0; i <= L; i++)
    {
            Q[i].x = P[i].x;
            Q[i].y = P[i].y;
    }
    for (r = 1 ; r <= L; r++)
    {
            for (i = 0; i <= L - r; i++)
            {
                    Q[i].x = (1 - t)*Q[i].x + t*Q[i + 1].x;
                    Q[i].y = (1 - t)*Q[i].y + t*Q[i + 1].y;
            }
    }
    return(Q[0]);
}
```

Để vẽ đường cong Bezier ta chỉ cần áp dụng gọi hàm **Casteljau** trong thủ tục **DrawCurve** sau:

```
void DrawCurve(float a, float b, int NumPoints)
{
    float Delta = (b – a)/(float)NumPoints;
    float t = a;
    int i;
    moveto(Casteljau(t).x, Casteljau(t).y) ;
    for (i = 1; i <= NumPoints; i++)
    {
            t += Delta ;
            lineto(Casteljau(t).x, Casteljau(t).y) ;
    }
}
```

2.4.1.2. Thuật toán Horner

Đa thức Bernstein và đường cong Bezier

Cách tiếp cận trong phần trước cho ta thuật toán hình học vẽ đường cong Bezier. Phần này trình bày cách biểu diễn giải tích của đường cong Bezier.

Thật vậy, dễ dàng chứng minh rằng đường cong Bezier *P(t)* tương ứng các điểm điều khiển P_0, P_1,…, P_L, xác dịnh bởi:

$$P(t) = \sum_{k=0}^{L} P_k B_k^L(t)$$

trong đó

$$B_k^L(t) := \binom{L}{k} (1-t)^{L-k} . t^k$$

là đa thức Bernstein, và $\binom{L}{k}$ là tổ hợp chập *k* của *L* phần tử.

Ví dụ, từ định nghĩa trên, ta có các đa thức Bernstein bậc ba:

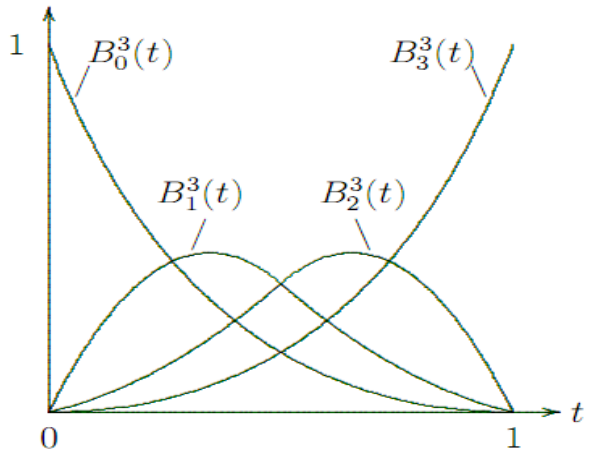
$$B_0^3(t) = \binom{3}{0} (1-t)^3 . t^0 = (1-t)^3$$

$$B_1^3(t) = \binom{3}{1} (1-t)^2 . t^1 = (1-t)^2 . t$$

$$B_2^3(t) = \binom{3}{2} (1-t)^1 . t^2 = (1-t) . t^2$$

$$B_3^3(t) = \binom{3}{3} (1-t)^0 . t^3 = t^3$$

Đồ thị minh họa của bốn đa thức này khi t ∈ [0, 1]:



Hình 2.10 . Các đa thức Bernstein bậc ba

Ví dụ, phương trình tham số của đường cong Bezier tương ứng bốn điểm điều khiển $P_0(1, 0)$, $P_1(2, 3)$, $P_2(6, 0)$, $P_3(9, 2)$ có dạng:

$$P(t) = (1-t)^3P_0 + 3(1-t)^2tP_1 + 3(1-t)t^2P_2 + t^3P_3$$

$$= (1-t)^3(1,0) + 3(1-t)^2t(2,3) + 3(1-t)t^2(6,0) + t^3(9,2)$$

$$= [(1-t)^3 + 6(1-t)^2t + 18(1-t)t^2 + 9t^3, 9(1-t)^2t + 2t^3]$$

$$= [1 + 3t^2 - 3t - t^3 + 6t - 12t^2 + 6t^3 + 18t^2 - 18t^3 + 9t^3, 9t - 18t^2 + 9t^3 + 2t^3]$$

$$= (-4t^3 + 9t^2 + 3t + 1, 11t^3 - 18t^2 + 9t)$$

Vẽ đường cong Bezier qua đa thức Bernstein

Dựa vào lược đồ Horner để tính giá trị đa thức Bernstein, ta xây dựng thủ tục xác định đường cong Bezier hiệu quả hơn Casteljau. Một ví dụ nhân lòng nhau của lược đồ Horner trong trường hợp đa thức bậc ba:

$$c_0 + tc_1 + t^2c_2 + t^3c_3 = c_0 + t(c_1 + t(c_2 + tc_3))$$

Tương tự với đường cong Bezier bậc ba:

$$P^3(t) = \left(\left(\binom{3}{0}sP_0 + \binom{3}{1}tP_1\right)s + \binom{3}{2}t^2P_2\right)s + \binom{3}{3}t^3P_2$$

trong đó, $s = 1 - t$. Nhận xét rằng:

$$\binom{L}{i} = \frac{L-i+1}{i}\binom{L}{i-1}; i > 0$$

Do đó, ta có chương trình tính giá trị hàm Bezier $P(t)$ trong trường hợp tổng quát, với *NumVertices* chính là số điểm điều khiển $L+1$.

Cài đặt minh họa thuật toán Horner

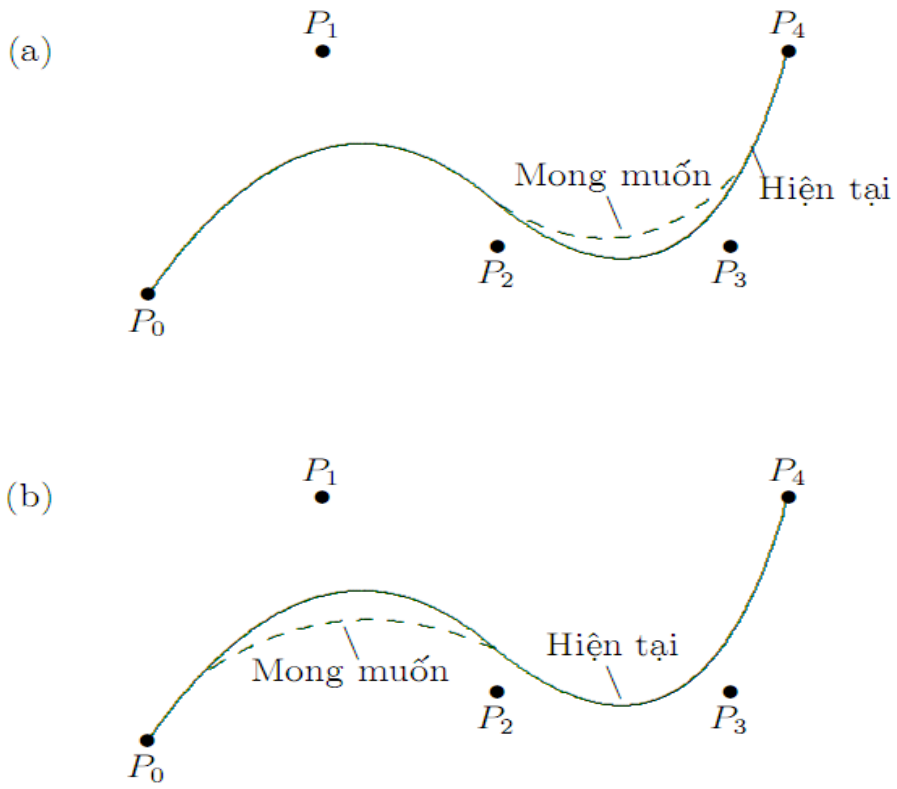
```
Point Horner_Bezier(float t)
{
    int i, L_choose_i;
    float Fact, s;
    Point Q;
    s = 1.0 - t;
    Fact = 1.0;
    L_choose_i = 1;
    Q.x = P[0].x*s;
    Q.y = P[0].y*s;
    for(i = 1; i < L; i++)
    {
        Fact *= t;
        L_choose_i *= (L - i + 1)/i;
        Q.x = (Q.x + Fact*L_choose_i*P[i].x)*s;
        Q.y = (Q.y + Fact*L_choose_i*P[i].y)*s;
    }
    Q.x += Fact*t*P[L].x;
    Q.y += Fact*t*P[L].y;
    return(Q);
}
```

2.4.2. Đường cong B-Spline

Đường cong Bezier được điều khiển một cách "toàn cục", có nghĩa là khi một điểm điều khiển thay đổi thì toàn bộ đường cong cũng thay đổi theo. Trong thực tế, ta mong muốn thay đổi một đoạn trên đường cong như hình 2.11, tức là điều khiển một cách địa phương. Điều này đường cong Bezier không thực hiện được. Do đó, ta cần tìm các đa thức trộn lại (hàm trộn) mà vẫn giữ tính chất tốt của đa thức Bernstein và các đa thức này có giá trị chứa trong đoạn [0, 1] để người thiết kế điều khiển đường cong theo mong muốn một cách địa phương.

Để có thể điều khiển hình dạng các hàm trộn, ta cần xây dựng các hàm liên tục $R_k(t)$ là những đa thức từng khúc. Do đó, $R_k(t)$ trên mỗi khoảng $(t_i, t_{i+1}]$ là một đa thức. Suy ra, đường cong $P(t)$ là tổng các đa thức từng khúc với trọng lượng là các điểm điều khiển. Chẳng hạn, trong khoảng nào đó thì đường cong có dạng:

$$P(t) = P_0(3t^2 - 4t + 2) + P_1(8t^2 - 7.3t - 5.9) + \cdots$$



Hình 2.11: Thay đổi đường cong mong muốn

Trong khoảng kế tiếp, đường cong được cho bởi một các đa thức khác, và tất cả các đoạn cong này tạo thành một đường cong liên tục. Đường cong này được gọi là

đường cong *Spline*. Trên một họ các hàm trộn, ta chọn xây dựng các hàm trộn có giá trị nhỏ nhất và do đó điều khiển địa phương tốt nhất. Khi đó, ta gọi đường cong này là *B-Spline*. Mỗi hàm *B-Spline* phục thuộc vào *m* và có bậc *m*-1, chúng ta ký hiệu $N_{k,m}$ thay cho $R_k(t)$. Do đó, phương trình đường cong *B-Spline* có dạng:

$$P(t) := \sum_{k=0}^{L} P_k N_{k,m}(t)$$

Như vậy, để xác định đường cong *B-Spline*, ta cần:

- Vector knot $T = (t_0, t_1, ...,)$.
- $L+1$ điểm điều khiển $P_0, P_1, ..., P_L$.
- Bậc *m* của các hàm *B-spline*.

Công thức xác định hàm đệ quy *B-spline* $N_{k,m}$

$$N_{k,m}(t) = \left(\frac{t - t_k}{t_{k+m-1} - t_k}\right) N_{k,m-1}(t) + \left(\frac{t_{k+m} - t}{t_{k+m} - t_{k+1}}\right) N_{k+1,m-1}(t)$$

với $k = 0, 1, \ldots, L,$ và

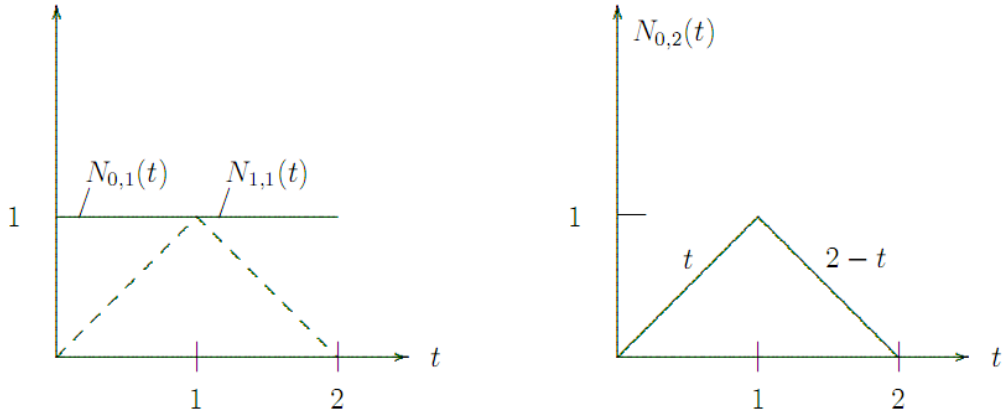
$$N_{k,1}(t) = \begin{cases} 1 \text{ nếu } t_k < t < t_{k+1} \\ 0 \text{ nếu ngược lại} \end{cases}$$

Ví dụ, xét vector Knot $T = (t_0 = 0, t_1 = 1, t_2 = 2,...)$ có khoảng cách giữa các Knot là 1. Khi đó:

$$N_{0,2}(t) = \frac{t}{1} N_{0,1}(t) + \frac{2 - t}{1} N_{1,1}(t)$$

$$= \begin{cases} t \text{ nếu } 0 \le t \le 1 \\ 2 - t \text{ nếu } 1 \le t \le 2 \\ 0 \text{ nếu ngược lại} \end{cases}$$

Đồ thị của hàm $N_{0,2}(t)$ trên đoạn [0, 2] là các đa thức bậc 1 và là một tam giác với các đỉnh (0, 0), (1, 1) và (2, 0).



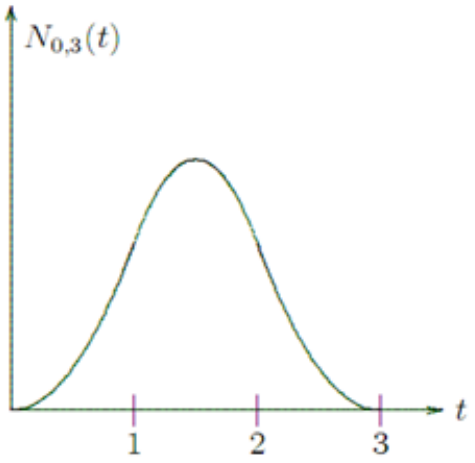
Hình 2.12: Đồ thị các hàm B-spline tuyến tính.

Trong thực tế, *m = 3*, và *m = 4* thường được sử dụng tương ứng với đường cong *B-Sline* bậc 2 và bậc 3.

- **m = 3**

$$N_{0,3}(t) = \frac{t}{2}N_{0,2}(t) + \frac{3-t}{2}N_{1,2}(t)$$

$$= \begin{cases} \frac{1}{2}t^2 \text{ nếu } 0 \le t \le 1 \\ \frac{3}{4} - (t - \frac{3}{2})^2 \text{ nếu } 1 \le t \le 2 \\ \frac{1}{2}(3-t)^2 \text{ nếu } 2 \le t \le 3 \\ 0 \text{ nếu ngược lại} \end{cases}$$

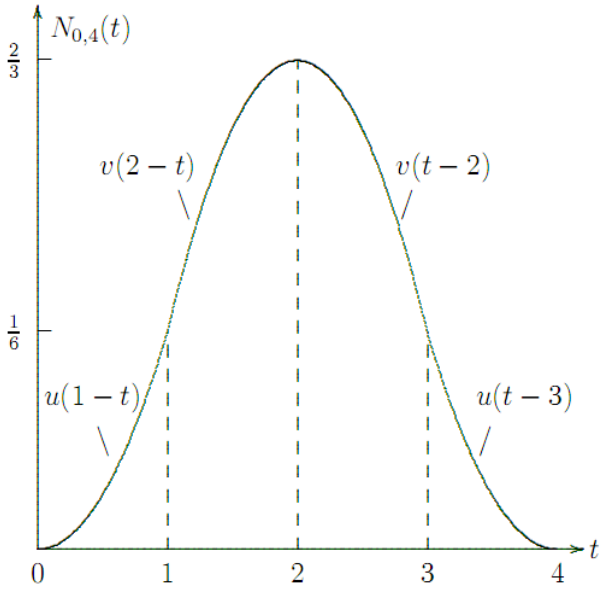


Hình 2.13: Đồ thị hàm B-Spline bậc 2(m=2)

$$N_{0,4}(t) = \begin{cases} u(1-t) \text{ nếu } 0 \le t \le 1 \\ v(2-t) \text{ nếu } 1 \le t \le 2 \\ v(t-2) \text{ nếu } 2 \le t \le 3 \\ u(t-3) \text{ nếu } 3 \le t \le 4 \\ 0 \text{ nếu ngược lại} \end{cases}$$

$$u(t) = \frac{1}{6}(1-t)^3$$

$$v(t) = \frac{1}{6}(3t^3 - 6t^2 + 4)^3$$



Hình 2.14: Đồ thị hàm B-Spline bậc 3 (m=4)

Cài đặt minh họa thuật toán vẽ đường cong B-Spline

```
void Create_Knot(int m)
{
        if (L < m || L + m > Max)
           return;
        int i;
        for (i = 0; i < m; i++) Knot[i] = 0;
        for (; i <= L; i++) Knot[i] = i - m + 1;
        for (; i < L + m; i++) Knot[i] = L - m + 2;
```

```
}

int  N(int k, int m, float t)
{
        if (m == 1)
        {
           if (t < Knot[k] || t > Knot[k + 1]) return 0;
           return 1;
        }
        else
        {
           float Sum, Demo1, Demo2;
           Demo1 = Knot[k + m - 1] - Knot[k];
           if (Demo1 != 0)
              Sum = (t - Knot[k]) * N(k, m - 1, t) / Demo1;
           else
              Sum = 0;
           Demo2 = Knot[k + m] - Knot[k + 1];
           if (Demo2 != 0)
                Sum += (Knot[k + m] - t) * N(k + 1, m - 1, t) / Demo2;
          return Sum;
        }
}

Point Brestern_Spline(float  t)
{
   Create_Knot(M);
   Point Q = new Point();
   Q.X = 0;
   Q.Y = 0;
   float x = 0, y = 0;
   for (int i = 0; i <= NumVertices; i++)
```

```
        {
            x += N(i, M, t) * P[i].X;
            y += N(i, M, t) * P[i].Y;
        }
        Q.X = (int)x;
        Q.Y = (int)y;

        return Q;
}
```

Bài tập chương 2

1. Viết chương trình vẽ bầu trời có 1.000 điểm sao, mỗi điểm sao xuất hiện với một màu ngẫu nhiên. Những điểm sao này hiện lên rồi từ từ tắt cũng rất ngẫu nhiên.
2. Viết chương trình vẽ đoạn thẳng *AB* với theo giải thuật DDA.
3. Viết chương trình vẽ đoạn thẳng *AB* với theo giải thuật Bresenham.
4. Tương tự như bài tập 3 nhưng sử dụng giải thuật MidPoint trong trường hợp hệ số góc thuộc [0, 1].
5. Cải tiến bài tập 3, viết chương trình vẽ đường thẳng bằng giải thuật MidPoint cho tất cả các trường hợp của hệ số góc. Lưu ý xét trường hợp đặc biệt khi đường thẳng song song với trục tung hay với trục hoành.
6. Viết chương trình vẽ đoạn thẳng trong trường hợp hệ số góc tổng quát bằng phương pháp lấy đối xứng với các thuật toán DDA, Bresenham, MidPoint.
7. Viết chương trình vẽ một đường tròn tâm O bán kính R theo thuật toán MidPoint.
8. Cải tiến bài tập 7 để vẽ đường tròn đặc (tô màu đường tròn).
9. Viết chương trình vẽ Elippse theo thuật toán MidPoint.
10. Cải tiến bài tập 9 để vẽ Elippse đặc (tô màu Elippse).
11. Viết chương trình vẽ một hình chữ nhật, một hình vuông và một hình

bình hành.

12. Viết chương trình vẽ một tam giác. Tọa độ các đỉnh được nhập từ bàn phím, mỗi cạnh có một màu khác nhau.
13. Viết chương trình vẽ một đa giác có *n* đỉnh.
14. Viết chương trình vẽ đường cong Bezier với *n* điểm điều khiển: P_1, P_2, …, P_n nhập từ file text.
15. Viết chương trình vẽ đường cong B-Spline với *n* điểm điều khiển: P_1, P_2, …, P_n nhập từ file text.

Chương 3

TÔ MÀU

Nội dung chính

- Cơ sở về màu sắc.
- Các thuật toán tô màu đơn giản
- Thuật toán tô màu bằng dòng quét (ScanConvert).
- Thuật toán tô màu theo biên (FloodFill).

3.1 Giới thiệu về màu sắc

Tô màu một vùng là thay đổi màu sắc của các điểm vẽ nằm trong vùng cần tô. Một vùng tô thường đựoc xác định bởi một đường khép kín nào đó gọi là đường biên. Dạng đường biên đơn giản thường gặp là đa giác. Việc tô màu thường chia làm 2 công đoạn :

- Xác định vị trí các điểm cần tô màu.
- Quyết định tô các điểm trên bằng màu nào. Công đoạn này sẽ trở nên phức tạp khi ta cần tô theo một mẫu tô nào đó chứ không phải tô thuần một màu.

Giáo trình giới thiệu 3 cách tiếp cận chính để tô màu:

- Tô màu theo từng điểm (có thể gọi là tô màu đơn giản).
- Tô màu theo dòng quét (ScanConvert).
- Tô màu dựa theo vết dầu loang (FloodFill).

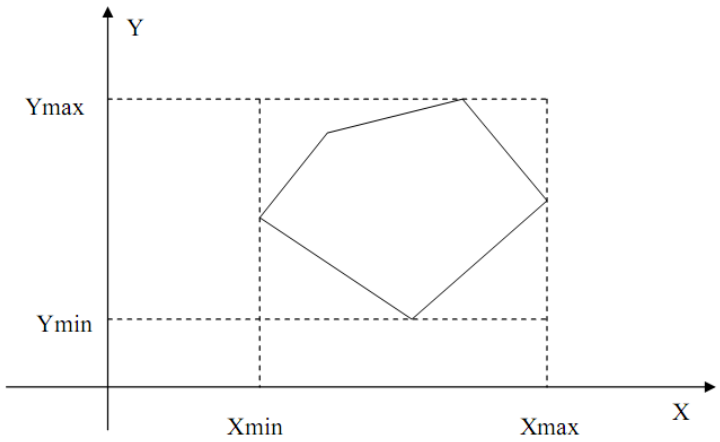
3.2 Tô màu đơn giản

Thuật toán này bắt đầu từ việc xác định một điểm có thuộc vùng cần tô hay không , nếu đúng thì sẽ tô với màu muốn tô.

Không mất tính tổng quát, ta xét tô màu một đa giác bất kỳ. Đầu tiên ta tìm hình chữ nhật nhỏ nhất có các cạnh song song với hai trục tọa độ chứa đa giác cần tô dưa

vào hai tọa độ (x_{min}, y_{min}), (x_{max}, y_{max}). Trong đó, x_{min}, y_{min} là hoành độ và tung độ nhỏ nhất, x_{max}, y_{max} là hoành độ và tung độ lớn nhất của các đỉnh của đa giác.

Cho x đi từ x_{min} đến x_{max}, y đi từ y_{min} đến y_{max}. Xét điểm $P(x, y)$ có thuộc đa giác không, nếu có thì tô với màu cần tô (xem hình 3.1).

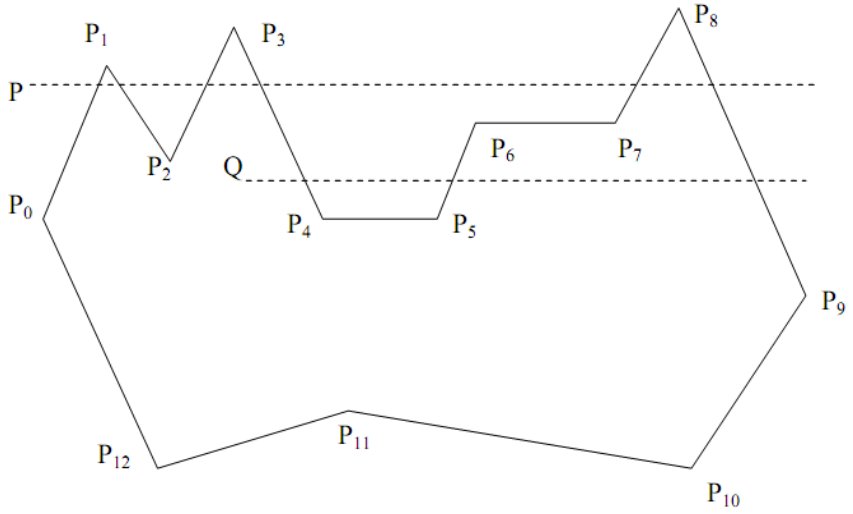


Hình 3.1: Đa giác nội tiếp hình chữ nhật.

Nguyên tắc xác định một điểm nằm trong đa giác

Một điểm nằm trong đa giác thì số giao điểm từ một tia bất kỳ xuất phát từ điểm đó cắt biên của đa giác phải là một số lẻ. Tia xuất phát có thể sang phải hay sang trái. Đặc biệt, tại các đỉnh cực trị thì một giao điểm phải được tính 2 lần (xem hình 3.2).

Ví dụ : Xét đa giác gồm 13 đỉnh là P_0, P_1, ..., P_{12}



Hình 3.2: Đa giác có 13 đỉnh.

Gọi tung độ của đỉnh P_i là $P_i.y$. Nếu :

- $P_i.y < Min(P_{i+1}.y, P_{i-1}.y)$ hay $P_i.y > Max(P_{i+1}.y, P_{i-1}.y)$ thì P_i là đỉnh cực trị.
- $P_{i-1}.y < P_i.y < P_{i+1}.y$ hay $P_{i-1} > P_i.y > P_{i+1}.y$ thì P_i là đỉnh đơn điệu.
- $P_i = P_{i+1}$ và $P_i.y < Min(P_{i+2}.y, P_{i-1}.y)$ hay $P_i > Max(P_{i+2}.y, P_{i-1}.y)$ thì đoạn $[P_i, P_{i+1}]$ là đoạn cực trị.
- $P_i = P_{i+1}$ và $P_{i-1}.y < P_i.y < P_{i+2}.y$ hay $P_{i-1} > P_i.y > P_{i+2}.y$ thì đoạn $[P_i,P_{i+1}]$ là đoạn đơn điệu.

Thuật toán xác định điểm nằm trong đa giác

- Với mỗi đỉnh của đa giác ta đánh dấu là 0 hay 1 theo qui ước như sau: nếu là đỉnh cực trị hay đoạn cực trị thì đánh số 0. Nếu là đỉnh đơn điệu hay đoạn đơn điệu thì đánh dấu 1.
- Xét số giao điểm của tia nửa đường thẳng từ P là điểm cần xét với biên của đa giác. Nếu số giao điểm là chẵn thì kết luận điểm không thụôc đa giác. Ngược lại, số giao điểm là lẻ thì điểm thuộc đa giác.

Minh họa thuật toán xét điểm thuộc đa giác

```
int  next(int i)
{
  return ( (i + n + 1) mod n );
}
 int  prev(int i)
 {
   return ((i + n - 1) mod n );
 }
bool  PointInpoly(Point[] d; Point P;int  n)
{
        int  count, I, x_cut;
         count = 0;
         for (i = 0; i< n; i++)
           if   (d[i].y == P.y )
           {
```

```
                    if    d[i].x > P.x
                    {
                        if ((d[prev(i)].y < P.y) && (P.y < d[next(i)].y)) ||
                         ((d[prev(i)].y > P.y) && (P.y > d[next(i)].y))
                          count = count + 1;
                        if  (d[next(i)].y == P.y)
                            if ((d[prev(i)].y < P.y) && (P.y < d[next(next(i))].y)) ||
                                ((d[prev(i)].y > P.y  && (P.y > d[next(next(i))].y))
                           count = count + 1;
                    }
                } else   //d[i].y = P.y
                if ((d[i].y < P.y) && (P.y < d[next(i)].y)) ||
                 ((d[i].y > P.y) && (P.y > d[next(i)].y))
                {
                  x_cut := d[i].x + Round((d[next(i)].x - d[i].x)
                            / (d[next(i)].y - d[i].y) * (P.y - d[i].y));
                  if x_cut >= P.x            count := count + 1;
                }
    if  (count mod 2 = 0)       return  false;
    return true;
}
```

Minh họa thuật toán tô đa giác

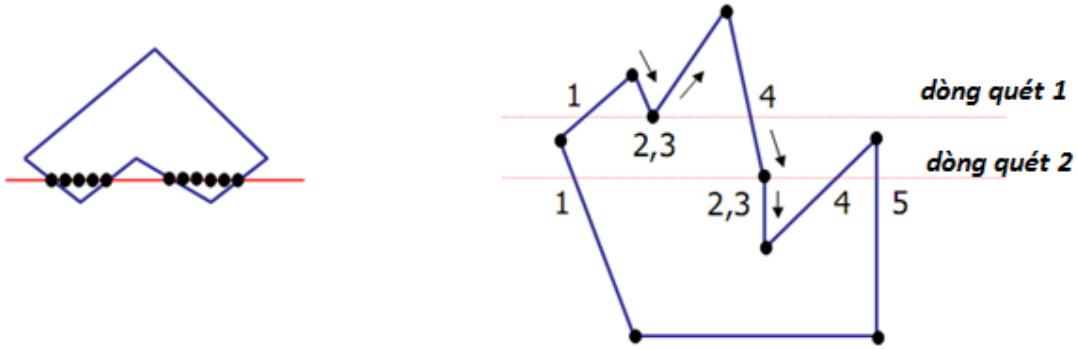
```
void TomauDagiac ( Point[] d; int n, int maubien)
{
        int  x, y;
        Point  P;
        for (x=xmin; x<= xmax; x++)
        for ( y= ymin; y<= ymax; y++)
        {
                P.x= x;   P.y= y;
                if  pointInpoly (d, P, n)
```

```
                    if getpixel(x,y) != maubien          putpixel(x,y,color);
            }
    }
```

Nhận xét: Thuật toán tô đơn giản có ưu điểm là tô rất mịn và có thể sử dụng được cho đa giác lồi hay đa giác lõm, hoặc đa giác tự cắt, đường tròn, ellipse. Tuy nhiên, giải thuật này sẽ trở nên chậm khi ta phải gọi hàm PointInpoly nhiều lần. Để khắc phục nhược điểm này người ta đưa ra thuật toán tô màu theo dòng quét.

3.3 Tô màu theo dòng quét (ScanConvert)

Ý tưởng: Sử dụng giao điểm giữa các biên đa giác và đường quét để xác định các điểm nằm trong đa giác.



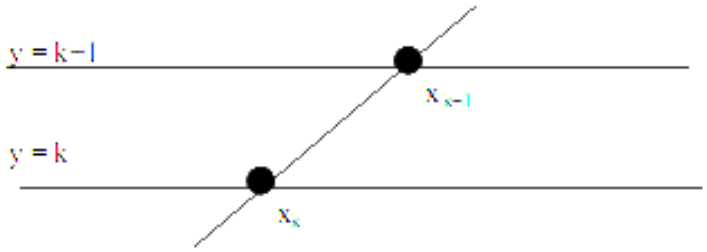
Hình 3.3: Tô màu theo dòng quét.

Các bước thuật toán:

- Tìm y_{min}, y_{max} lần lượt là giá trị nhỏ nhất, lớn nhất của tập các tung độ của các đỉnh của đa giác đã cho.
- Ứng với mỗi dòng quét $y = k$ với k thay đổi từ y_{min} đến y_{max}, lặp :
- Tìm tất cả các hoành độ giao điểm của dòng quét $y = k$ với các cạnh của đa giác.
- Sắp xếp các hoành độ giao điểm theo thứ tự tăng dần : $x_0, x_1,..., x_n$.
- Vẽ các đoạn thẳng trên đường thẳng $y = k$ lần lượt được giới hạn bởi các cặp cách quãng nhau: (x_0, x_1), (x_1, x_2), ...

Vấn đề cần chú ý

- Hạn chế được số cạnh cần tìm giao điểm ứng với mỗi dòng quét vì ứng với mỗi dòng quét không phải lúc nào cũng giao với các cạnh của đa giác.
- Để tìm giao điểm giữa cạnh đa giác và dòng quét, ta có nhận xét sau:



$$x_{k+1} - x_k = \frac{1}{m}((k+1) - k) = \frac{1}{m} \; hay \; x_{k+1} = x_k + \frac{1}{m}$$

Trong đó, *m* là hệ số góc của cạnh.

- Giải quyết trường hợp số giao điểm đi qua đỉnh đơn điệu thì tính số giao điểm là 1 hay đi qua đỉnh cực trị thì tính số giao điểm là 0 (hoặc 2).

Thuật toán ScanConvert

```
void ScanFill(Point[] Poly)
    {
      int min_y = Poly[0].Y, max_y = Poly[0].Y;
      int min_x = Poly[0].X, max_x = Poly[0].X;
      for (int i = 1; i < Poly.Length; i++)
      {
        if (Poly[i].Y > max_y) max_y = Poly[i].Y;
        if (Poly[i].Y < min_y) min_y = Poly[i].Y;

        if (Poly[i].X > max_x) max_x = Poly[i].X;
        if (Poly[i].X < min_x) min_x = Poly[i].X;
      }
      Point G = new Point();
      Point A, B;
      for (int y = min_y; y < max_y; y++)
```

```
{
    List<Point> list = new List<Point>();
    for (int i = 0; i < Poly.Length - 1; i++)
    {
        A = Poly[i];
        B = Poly[i + 1];
if (Giao(A, B, new Point(min_x, y), new Point(max_x, y), ref G) == true)
        {
          if (G == Poly[j])
         {
            if (j == 0)
            {
                    if ((G.Y > Poly[Poly.Length - 1].Y && G.Y > Poly[1].Y) ||
(G.Y < Poly[Poly.Length - 1].Y && G.Y < Poly[1].Y))
                {
                    list.Add(G);
                }
            }
            else
            {
                  if ((G.Y > Poly[j - 1].Y && G.Y > Poly[j + 1].Y) || (G.Y <
Poly[j - 1].Y && G.Y < Poly[j + 1].Y))
                {
                    list.Add(G);
                }
            }
           else
          {
                   list.Add(G);
          }
        }
    }
```

```
        A = Poly[Poly.Length - 1];
        B = Poly[0];
      if (Giao(A, B, new Point(min_x, y), new Point(max_x, y), ref G) == true)
            {
             if (G == Poly[j])
             {
               if (j == 0)
               {
                    if ((G.Y > Poly[Poly.Length - 1].Y && G.Y > Poly[1].Y) ||
      (G.Y < Poly[Poly.Length - 1].Y && G.Y < Poly[1].Y))
                   {
                      list.Add(G);
                   }
               }
               else
               {
                    if ((G.Y > Poly[j - 1].Y && G.Y > Poly[j + 1].Y) || (G.Y <
      Poly[j - 1].Y && G.Y < Poly[j + 1].Y))
                   {
                      list.Add(G);
                   }
               }
              else
              {
                     list.Add(G);
              }
        SortList(list);
        DrawList(list);
      }
    }
    void SortList(List<Point> list)
    {
```

```
    for (int i = 0; i < list.Count - 1; i++)
    {
        for (int j = i + 1; j < list.Count; j++)
        {
            if (list[i].X > list[j].X)
            {
                Point t = list[i];
                list[i] = list[j];
                list[j] = t;
            }
        }
    }
}
void DrawList(List<Point> list)
{
    for (int i = 0; i < list.Count - 1; i += 2)
    {
        Line (list[i], list[i + 1]);
    }
}
```

3.4 Tô màu theo vết dầu loang (FloodFill)

Ý tưởng

- Thuật toán nhằm tô màu vùng kín, giới hạn bởi màu Bcolor, màu sử dụng để tô là Fcolor với điểm (*x, y*) nằm trong vùng tô màu.
- Thuật sử dụng phép gọi đệ quy, ban đầu (*x, y*) được kiểm tra màu, nếu màu của nó là Fcolor hoặc Bcolor thì tiến trình kết thúc. Trong trường hợp ngược lại, điểm (*x,y*) được tô với màu Fcolor và quá trình gọi đệ quy với các điểm láng giềng của (*x, y*). Các điểm láng giềng được sử dụng là bốn láng giềng.

	X	
X	**(*x, y*)**	**X**
	X	

Suy ra, bốn láng giềng của (*x, y*) là: (*x* + 1, *y*), (*x* - 1, *y*), (*x, y* + 1), (*x, y* - 1)

Chương trình minh họa thuật toán FloodFill

```
byte[] getPixel(int x, int y)
{
        int[] Pos = new int[2];
         glRasterPos2i(x, y);
         glGetIntegerv(GL_CURRENT_RASTER_POSITION, Pos);

         int width = 1, height = 1;
         byte[] pixels = new byte[3 * width * height];
         glReadPixels(Pos[0], Pos[1], width, height, GL_RGB,
         GL_UNSIGNED_BYTE, pixels);

         return pixels;
}

void FloodFill(int x, int y, int Bcolor, byte[] Fcolor)
{
        glColor3ubv(Fcolor);
        if(getPixel(x, y) != Bcolor && getPixel(x, y) != Fcolor)
        {
                glVertex2i(x, y);
                Boundary(x+1,y,Bcolor,Fcolor);
                Boundary(x-1,y,Bcolor,Fcolor);
                Boundary(x,y+1,Bcolor,Fcolor);
                Boundary(x,y-1,Bcolor,Fcolor);
        }
}
```

Thuật toán đệ quy trên sẽ không khả thi nếu như vùng cần tô màu lớn. Khi đó, thời gian thực thi và vùng nhớ lưu trữ trong quá trình đệ quy là rất lớn. Để khắc phục điều này, ta sẽ xét thuật toán khử đệ quy bằng cách dùng mảng lưu trữ các pixel (hàng đợi) thay vì gọi hàm đệ quy:

Chương trình khử đệ quy

```
void FloodFill(int x, int y, int Bcolor, byte[] Fcolor)
{
    glColor3ubv(Fcolor);
    byte[] color;
    int count=0;
    Point mPT[MaxPT];
    mPT[count].x=x; mPT[count].y=y;
    while(count>0)
    {
        count--;
        color = getPixel(mPT[count].x, mPT[count].y);
        if(color != Bcolor && color != Fcolor)
        {
            glVertext2i(x,y);
            mPT[count].x=x+1; mPT[count++].y=y;
            mPT[count].x=x-1; mPT[count++].y=y;
            mPT[count].x=x; mPT[count++].y=y+1;
            mPT[count].x=x; mPT[count++].y=y-1;
        }
    }
}
```

Bài tập chương 3

1. Viết chương trình vẽ một đa giác *n* đỉnh, xét xem một điểm *P* nào đó có thuộc đa giác không ?
2. Viết chương trình vẽ một đa giác *n* đỉnh. Tô đa giác bằng giải thuật tô đơn giản.
3. Viết chương trình vẽ một đa giác *n* đỉnh. Tô đa giác bằng giải thuật FloodFill. Lưu ý cho các trường hợp của đa giác : hình chữ nhật, đa giác lồi, đa giác lõm.
4. Viết chương trình vẽ một đa giác *n* đỉnh. Tô đa giác bằng giải thuật ScanConvert.

Chương 4

PHÉP BIẾN ĐỔI HAI CHIỀU

Nội dung chính

- Các phép biến đổi ma trận.
- Các phép biến đổi Affine 2D cơ sở.
- Các phép biến đổi 2D gộp.

4.1 Nhắc lại các phép toán cơ sở với ma ma trận.

Cộng, trừ ma trận: Chỉ thực hiện cho hai ma trận cùng bậc

$$[A(m, n)] + [B(m, n)] = [C(m, n)]$$

$$\lfloor c_{ij} \rfloor = \lfloor a_{ij} \rfloor + \lfloor b_{ij} \rfloor$$

Nhân hai ma trận: Ma trận bậc $n_1 \times m_1$ và ma trận bậc $n_2 \times m_2$ nhân được với nhau nếu $m_1 = n_2$

$$[A(m,n)][B(n,p)] = [C(m,p)]$$

$$c_{jk} = \sum_{i=1}^{n} a_{ji} b_{ik} \ , \ j = 1, \dots, m \text{ và } k = 1, \dots, p$$

Đảo ma trận vuông: Không có phép chia ma trận

- Nếu $[A][X]=[Y]$ thì $[X]=[A]^{-1}[Y]$, trong đó $[A]^{-1}$ là ma trận đảo của ma trận vuông $[A]$.
- $[A][A]^{-1} = [I]$, trong đó $[I]$ là ma trận đơn vị.

$$[A]^{-1} = \frac{1}{det[A]} ||A||^{T}$$

  - ✓ Tính $||A||$: Thay các phần tử của $[A]$ bằng các phần phụ đại số của nó.
  - ✓ Phần phụ đại số của phần tử (a_{ij}) là:

$$(-1)^{i+j}[M_{ij}]$$

✓ $[M_{ij}]$ được tạo ra nhờ xóa hàng i, cột j của $[A]$.

4.2 Phép tịnh tiến

Có hai quan điểm về phép biến đổi hình học, đó là :

- Biến đổi đối tượng : thay đổi tọa độ của các điểm mô tả đối tượng theo một qui tắc nào đó.
- Biến đổi hệ tọa độ : Tạo ra một hệ tọa độ mới và tất cả các điểm mô tả đối tượng sẽ được chuyển về hệ tọa độ mới.

Các phép biến đổi hình học cơ sở là : tịnh tiến, quay, biến đổi tỉ lệ. Phép biến đổi Affine hai chiều (gọi tắc là phép biến đổi) là một ánh xạ T biến đổi điểm $P(P_x, P_y)$ thành điểm $Q(Q_x, Q_y)$ theo hệ phương trình sau:

$$\begin{cases} Q_x = a.P_x + c.P_y + tr_x \\ Q_y = b.P_x + d.P_y + tr_y \end{cases}$$

$$(Q_x, Q_y) = (P_x, P_y).\begin{pmatrix} a & b \\ c & d \end{pmatrix} + (tr_x, tr_y)$$

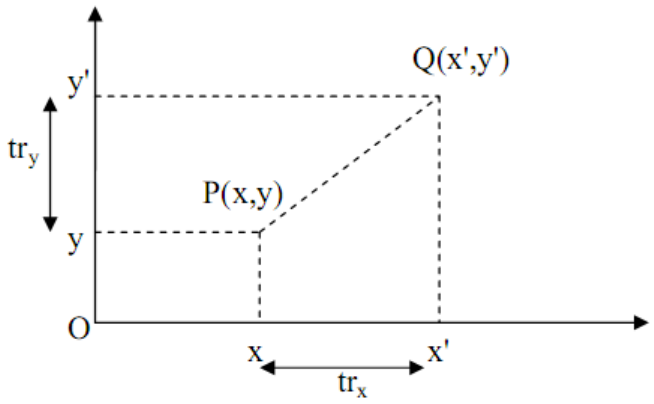
$$hay\ Q = P.M + tr$$

Nếu gọi tr_x và tr_y lần lượt là độ dời theo trục hoành và trục tung thì tọa độ điểm mới $Q(x', y')$ sau khi tịnh tiến điểm $P(x, y)$ sẽ là :

$$\begin{cases} x' = x + tr_x \\ y' = y + tr_y \end{cases}$$

Trong đó, (tr_x, tr_y) được gọi là vector tịnh tiến hay vector độ dời (xem hình 4.1).

Hay $Q = P.M + tr$, $M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}, tr = (tr_x, tr_y)$



Hình 4.1 : Phép biến đổi tịnh tiến điểm P thành Q

4.3 Phép biến đổi tỷ lệ

Phép biến đổi tỉ lệ làm thay đổi kích thước đối tượng. Để co hay giãn tọa độ của một điểm $P(x, y)$ theo trục hoành và trục tung lần lượt là S_x và S_y (gọi là các hệ số tỉ lệ), ta nhân S_x và S_y lần lượt cho các tọa độ của P.

$$\begin{cases} x' = x.S_x \\ y' = y.S_y \end{cases}$$

- Khi các giá trị S_x, S_y nhỏ hơn 1, phép biển đổi sẽ thu nhỏ đối tượng. Ngược lại, khi các giá trị này lớn hơn 1, phép biến đổi sẽ phóng lớn đối tượng.
- Khi $S_x = S_y$, người ta gọi đó là phép đồng dạng. Đây là phép biến đổi bảo toàn tính cân xứng của đối tượng. Ta gọi là phép phóng đại nếu $|S|>1$ và là phép thu nhỏ nếu $|S|<1$.

Nếu hai hệ số tỉ lệ khác nhau thì ta gọi là phép không đồng dạng. Trong trường hợp hoặc S_x hoặc S_y có giá trị 1, ta gọi đó là phép căng.

4.4 Phép quay

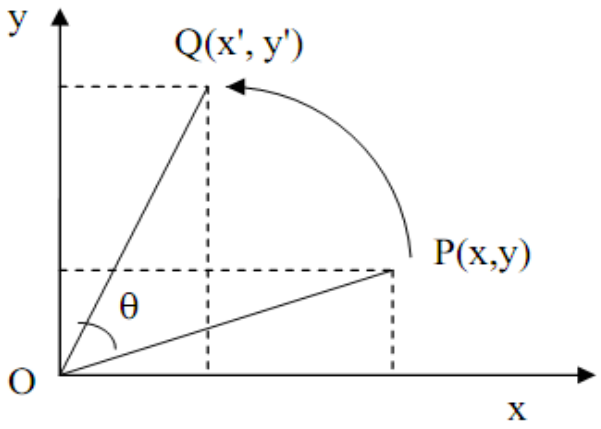
Phép quay làm thay đổi hướng của đối tượng. Một phép quay đòi hỏi phải có tâm quay, góc quay. Góc quay dương thường được qui ước là chiếu ngược chiều kim đồng hồ.

- **Phép quay quanh gốc tọa độ**

Ta có công thức biến đổi của phép quay điểm *P*(*x, y*) quanh gốc tọa độ góc θ (xem hình 4.2):

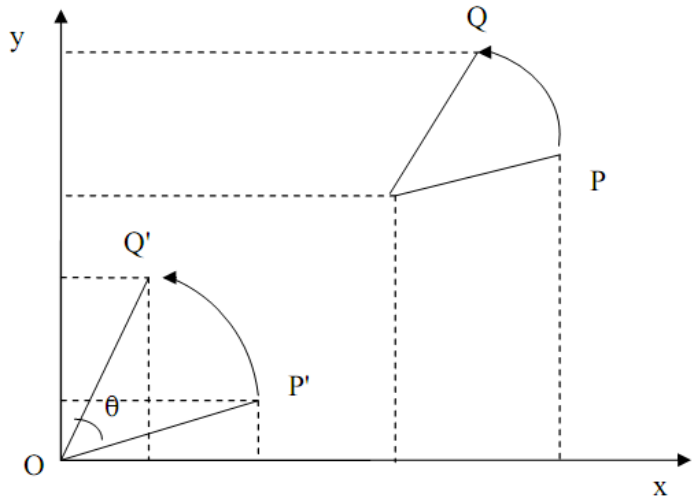
$$\begin{cases} x' = x.cos\theta - y.sin\theta \\ y' = x.sin\theta + y.cos\theta \end{cases}$$

Hay $Q = P.M$, trong đó: $M = \begin{pmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{pmatrix}$



Hình 4.2 : Phép quay quanh gốc tọa độ

- **Phép quay quanh một điểm bất kỳ**



Hình 4.3 : Phép quay quanh một điểm bất kỳ.

Xét điểm $P(P.x, P.y)$ quay quanh điểm $V(V.x, V.y)$ một góc θ đến điểm $Q(Q.x,Q.y)$. Ta có thể xem phép quay quanh tâm V được kết hợp từ phép các biến cơ bản sau:

- Phép tịnh tiến $(-V.x, -V.y)$ để dịch chuyển tâm quay về gốc tọa độ.
- Quay quanh gốc tọa độ O một góc θ.
- Phép tịnh tiến $(V.x, V.y)$ để đưa tâm quay về vị trí ban đầu.

Ta cần xác định tọa độ của điểm Q (xem hình 4.3).

- Từ phép tịnh tiến $(-V.x,-V.y)$ biến đổi điểm P thành P' ta được:

$$P' = P + V$$

Hay

$$P'.x = P.x - V.x$$

$$P'.y = P.y - V.y$$

- Phép quay quanh gốc tọa độ biến đổi điểm P' thành Q'

$$Q' = P'.M$$

Hay

$$Q'.x = P'.x.\cos\theta - P'.y.\sin\theta$$

$$Q'.y = P'.x.\sin\theta + P'.y.\cos\theta$$

- Phép tịnh tiến $(V.x, V.y)$ biến đổi điểm Q' thành Q ta được

$$Q = Q' + V$$

Hay

$$Q.x = Q'.x + V.x$$

$$Q.y = Q'.y + V.y$$

$$Q.x = (P.x - V.x)cos\theta - (P.y - V.y)\ sin\theta\ + V.x$$

$$Q.y = (P.x - V.x)sin\theta + (P.y - V.y)\ cos\theta + V.y$$

$$Q.x = P.x.cos\theta - P.y.sin\theta\ + V.x\ (1- cos\theta) + V.y.sin\theta$$

$$Q.y = P.x.sin\theta + P.y.cos\theta - V.x.sin\theta + V.y\ (1- cos\theta)$$

Vậy, $Q = P.M + tr$, trong đó:

$$\begin{cases} M = \begin{pmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{pmatrix} \\ tr = (V.x(1 - cos\theta) + V.y.sin\theta, -V.x.sin\theta + V.y(1 - cos\vartheta)) \end{cases}$$

4.5 Phép đối xứng

Phép đối xứng trục có thể xem là phép quay quanh trục đối xứng mõt góc 180^0. Phương trình ban đầu :

$$Q.x = a.P.x + c.P.y + tr_x$$

$$Q.y = b.P.x + d.P.y + tr_y$$

Hay $(Q.x, Q.y) = (P.x, P.y)\begin{pmatrix} a & b \\ c & d \end{pmatrix} + (tr_x, tr_y)$

- Trục đối xứng là trục hoành : $M = \begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$

  Ta có : $\begin{cases} Q.x = P.x \\ Q.y = -P.y \end{cases}$

- Tương tự trục đối xứng là trục tung :

  Ta có : $\begin{cases} Q.x = -P.x \\ Q.y = P.y \end{cases}$ $\quad M = \begin{pmatrix} -1 & 0 \\ 0 & 1 \end{pmatrix}$

4.6 Phép biến dạng

Phép biến dạng làm thay đổi, méo mó hình dạng của các đối tượng. Biến dạng theo phương trục x sẽ làm thay đổi hoành độ còn tung độ giữ nguyên.

Ví dụ, biến đổi điểm $P(P.x, P.y)$ thành điểm $Q(Q.x, Q.y)$ theo phương trục x là phép biến đổi được biểu diễn bởi phương trình sau:

$$Q.x = P.x + h.P.y$$

$$Q.y = P.y$$

$$M = \begin{pmatrix} 1 & 0 \\ h & 1 \end{pmatrix}$$

- Biến dạng theo phương trục y sẽ làm thay đổi tung độ còn hoành độ giữ nguyên.

$$Q.x = P.x$$

$$Q.y = g.P.x + P.y$$

$$M = \begin{pmatrix} 1 & g \\ 0 & 1 \end{pmatrix}$$

4.7 Phép biến đổi Affine ngược

Phép biến đổi ngược dùng để khôi phục một phép biến đổi đã thực hiện. Gọi Q là ảnh của P qua phép biến đổi T có ma trận biến đổi M là $P.M$

Phép biến đổi ngược T^{-1} sẽ có ma trận biến đổi là M^{-1} là ma trận nghịch đảo của ma trận M. Với ma trận biến đổi Affine dạng:

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$

thì ma trận nghịch đảo là:

$$M^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

- **Phép tịnh tiến**

$$M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$M^{-1} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

- **Phép quay**

$$M = \begin{pmatrix} cos\theta & sin\theta \\ -sin\theta & cos\theta \end{pmatrix}$$

$$M = \begin{pmatrix} cos0 & -sin0 \\ sin\theta & cos\theta \end{pmatrix}$$

- **Phép biến đổi tỉ lệ**

$$M = \begin{pmatrix} S_x & 0 \\ 0 & S_y \end{pmatrix}$$

$$M^{-1} = \begin{pmatrix} \frac{1}{S_x} & 0 \\ 0 & \frac{1}{S_y} \end{pmatrix}$$

- **Phép biến dạng**

$$M = \begin{pmatrix} 1 & g \\ h & 1 \end{pmatrix}$$

$$M^{-1} = \frac{1}{1 - g.h} \begin{pmatrix} 1 & -g \\ -h & 1 \end{pmatrix}$$

4.8 Hệ tọa độ thuần nhất

Tọa độ thuần nhất của một điểm trên mặt phẳng được biểu diễn bằng bộ ba số tỉ lệ (x_h, y_h, h) không đồng thời bằng 0 và liên hệ với các tọa độ (x, y) của điểm đó bởi công thức :

$$x = \frac{x_h}{h} \qquad y = \frac{y_h}{h}$$

Nếu một điểm có tọa độ thuần nhất là (x, y, z) thì nó cũng có tọa độ thuần nhất là ($h.x$, $h.y$, $h.z$), trong đó h là số thực khác 0 bất kỳ. Một điểm $P(x, y)$ sẽ được biểu diễn dưới dạng tọa độ thuần nhất là (x, y, 1). Trong hệ tọa độ thuần nhất các ma trận của

phép biến đổi được biểu diễn như sau :

- **Phép tịnh tiến**

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tr_x & tr_y & 1 \end{pmatrix}$$

- **Phép quay**

$$M = \begin{pmatrix} cos\theta & sin\theta & 0 \\ -sin\theta & cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- **Phép biến đổi tỉ lệ**

$$M = \begin{pmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Thuận lợi của hệ tọa độ thuần nhất là khi ta kết hợp hai hay nhiều phép biến đổi affine thi ma trận hợp của nhiều phép biến đổi được tính bằng cách nhân các ma trận của các phép biến đổi thành phần.

4.9 Kết hợp các phép biến đổi

Quá trình áp dụng các phép biến đổi liên tiếp để tạo nên một phép biến đổi tổng thể được gọi là sự kết hợp các phép biến đổi.

- **Kết hợp phép tịnh tiến**

Nếu ta thực hiện phép tịnh tiến lên điểm P được điểm P', rồi lại thực hiện tiếp một phép tịnh tiến khác lên P' được điểm Q. Như vậy, điểm Q là ảnh của phép biến đổi kết hợp hai phép tịnh tiến liên tiếp.

$$\begin{cases} Q.x = P.x + (tr_{x1} + tr_{x2}) \\ Q.y = P.y + (tr_{y1} + tr_{y2}) \end{cases}$$

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tr_{x1} & tr_{y1} & 1 \end{pmatrix}\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tr_{x2} & tr_{y2} & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tr_{x1} + tr_{x2} & tr_{y1} + tr_{y2} & 1 \end{pmatrix}$$

Vậy kết hợp hai phép tịnh tiến là một phép tịnh tiến. Từ đó, ta có kết hợp của nhiều phép tịnh tiến là một phép tịnh tiến.

- **Kết hợp phép quay**

Tương tự, ta có tọa độ điểm Q là điểm kết quả sau khi kết hợp hai phép quay quanh gốc tọa độ $M_{R1}(\theta_1)$ và $M_{R2}(\theta_2)$ là :

$$\begin{cases} Q.x = P.x.\cos(\theta_1 + \theta_2) + P.y.\sin(\theta_1 + \theta_2) \\ Q.y = P.x.\sin(\theta_1 + \theta_2) + P.y.\cos(\theta_1 + \theta_2) \end{cases}$$

$$M = \begin{pmatrix} cos\theta_1 & sin\theta_1 & 0 \\ -sin\theta_1 & cos\theta_1 & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} cos\theta_2 & sin\theta_2 & 0 \\ -sin\theta_2 & cos\theta_2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$
$$- \begin{pmatrix} \cos(\theta_1 + \theta_2) & \sin(\theta_1 + \theta_2) & 0 \\ \sin(\theta_1 + \theta_2) & \cos(\theta_1 + \theta_2) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- **Kết hợp phép biến đổi tỉ lệ**

Tương tự như phép tịnh tiến, ta có tọa độ điểm Q là điểm có được sau hai phép tịnh tiến $M_1(S_{x1}, S_{y1})$, $M_2(S_{x2}, S_{y2})$ là:

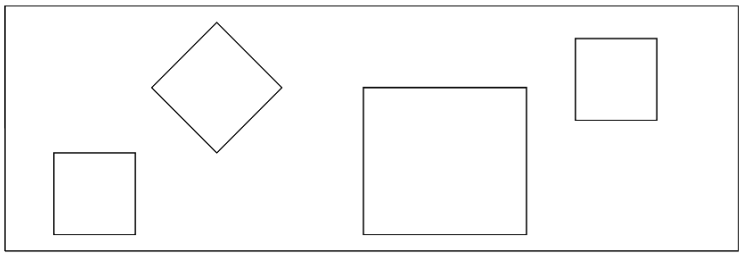
$$\begin{cases} Q_x = P_x.S_{x1}.S_{x2} \\ Q_y = P_y.S_{y1}.S_{y2} \end{cases}$$

$$M = \begin{pmatrix} S_{x1} & 0 & 0 \\ 0 & S_{y1} & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} S_{x2} & 0 & 0 \\ 0 & S_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} S_{x1}.S_{x2} & 0 & 0 \\ 0 & S_{y1}.S_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Bài tập chương 4

1. Vẽ một hình bình hành bằng cách sử dụng phép tịnh tiến. (Vẽ đoạn thẳng AB, sau đó tịnh tiến AB thành đoạn thẳng CD//AB; vẽ AD, tịnh tiến AD thành BC.
2. Viết chương trình vẽ một hình vuông ABCD (xem hình vẽ).

- Tịnh tiến hình vuông đó đến vị trí khác.
- Phóng to hình vuông ABCD.
- Biến dạng hình vuông thành hình thoi.



3. Vẽ một elip, sau đó vẽ thêm 3 elip khác có cùng tâm với elip đã cho, có độ dãn ở trục O*x* là *k* và O*y* là 1.
4. Viết chương trình mô phỏng sự chuyển động của trái đất quay quanh mặt trời.
5. Viết chương trình vẽ một đường tròn tâm O bán kính *R*. Vẽ một đường kính *AB*, và quay đường kính này quanh tâm đường tròn.
6. Tìm vị trí mới của tam giác *A*(1, 1), *B*(3, 2), *C*(2, 4) qua phép quay góc 30^o qua điểm (5, 5).

Chương 5

GIAO CÁC ĐỐI TƯỢNG ĐỒ HỌA

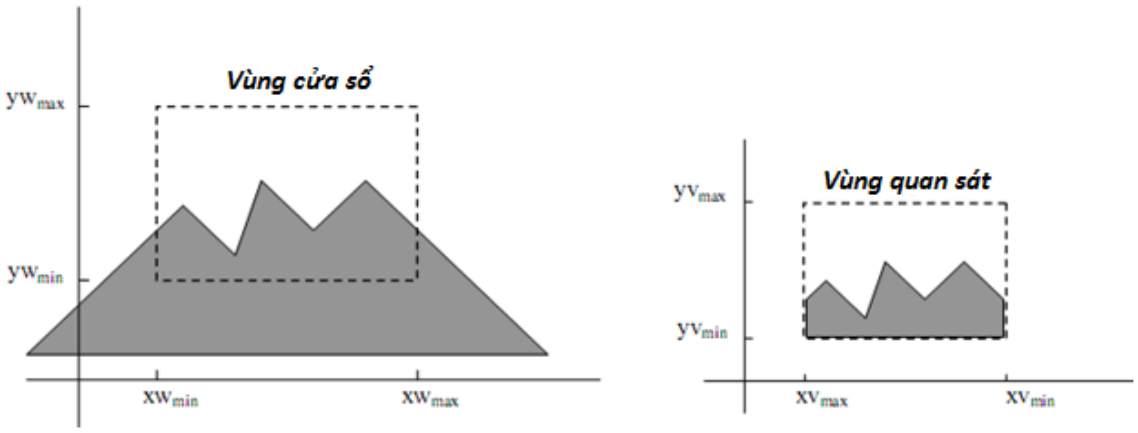
Nội dung chính

- Khái niệm cửa sổ.
- Thiết kế và cài đặt được các thuật toán tìm giao các đối tượng đồ họa: đường thẳng, hình chữ nhật, đa giác.
- Kỹ thuật Ray tracing.

5.1. Mở đầu

Hình ảnh trong hệ tọa độ thế giới thực được hiển thị trên các hệ tọa độ thiết bị dựa trên hệ tọa độ đồ họa. Chẳng hạn, trong một vùng đồ họa ta cần xác định vùng nào của hình ảnh sẽ được hiển thị và muốn đặt nó ở đâu trên hệ tọa độ thiết bị. Một vùng đơn lẻ hoặc vài vùng của hình ảnh có thể được chọn. Những vùng này có thể được đặt ở những vị trí tách biệt, hoặc một vùng có thể được chèn vào một vùng lớn hơn. Quá trình biến đổi này liên quan đến những thao tác như tịnh tiến, biến đổi tỷ lệ vùng được chọn và xóa bỏ những phần bên ngoài vùng được chọn.

Vùng có dạng hình chữ nhật được xác định trong hệ tọa độ thế giới thực được gọi là một **cửa sổ** (***window***). Còn vùng hình chữ nhật trên thiết bị hiển thị để cửa sổ đó ánh xạ đến được gọi là một **vùng quan sát** (***viewport***).



Hình 5.1: Cửa sổ và vùng quan sát

Ánh xạ một vùng cửa sổ vào trong một vùng quan sát, kết quả là chỉ hiển thị những phần trong phạm vi cửa sổ. Mọi thứ bên ngoài cửa sổ sẽ bị loại bỏ. Các thủ tục để loại bỏ các phần hình ảnh nằm bên ngoài biên cửa sổ được gọi là các thuật toán tìm giao hoặc cắt xén (***clipping***).

Vấn đề đặt ra trên đây cũng là một trong những vấn đề cơ sở, quan trọng của đồ họa máy tính như xác định phần giao của các đối tượng đồ họa: giao của hai đoạn thẳng, đoạn thẳng và hình chữa nhật, đa giác và hình chữ nhật, … Các thuật toán cần thực hiện nhanh nhất có thể để minh họa cập nhật các kết quả thay đổi trong ứng dụng đồ họa. Phương pháp giải tích được dùng để giải quyết các bài toán trong chương này.

5.2. Giao của hai đoạn thẳng

Ta xét giao của hai đường thẳng đi qua hai điểm được minh họa thông qua thí dụ đơn giản sau: đường thẳng đi qua tọa độ (4,2) và tọa độ (2,0) có giao với đoạn thẳng đi qua (0,4) và (4,0)?

Giải pháp

- Xác định phương trình đường thẳng qua 2 điểm $y = ax + b$, trong đó $a = (y_2 - y_1)/(x_2 - x_1)$
- Từ thí dụ trên, ta có: $y = x - 2$ và $y = - x + 4$ giao điểm tại (3, 1)

Tổng quát: nếu ta có $y = a_1 + b_1x$ và $y = a_2 + b_2x$ thì giao điểm sẽ ở tại:

$$x_i = -(a_1 - a_2)/(b_1 - b_2)$$
$$y_i = a_1 + b_1x_i$$

Trường hợp đặc biệt: đường thẳng song song với trục x hay trục y, hay song song với nhau. Nếu sử dụng phương pháp tìm giao đường thẳng: đòi hỏi kiểm tra tọa độ của giao đường thẳng có nằm trong các đoạn thẳng.

Phương pháp biểu diễn đoạn thẳng bằng tham số sẽ hiệu quả hơn, đoạn thẳng thứ nhất từ (x_A, y_A) đến (x_B, y_B); đoạn thẳng thứ hai từ (x_C, y_C) đến (x_D, y_D); tính toán giao của 2 đoạn thẳng tại tọa độ có t, s:

$$x = x_A + t(x_B - x_A)$$

$$y = y_A + t(y_D - y_A)$$

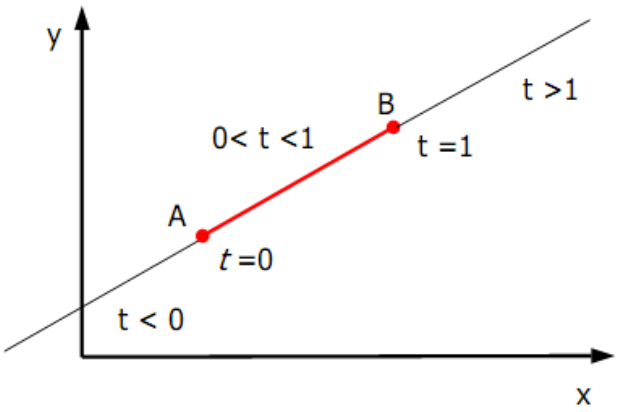
$$x = x_C + t(x_D - x_C)$$

$$y = y_C + t(y_D - y_C)$$

$$0 \leq t \leq 1 \text{ và } 0 \leq s \leq 1$$

$$t = \frac{(x_C - x_A)(y_C - y_D) - (x_C - x_D)(y_C - y_A)}{(x_C - x_A)(y_C - y_D) - (x_C - x_A)(y_C - y_A)}$$

$$s = \frac{(x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A)}{(x_B - x_A)(y_C - y_D) - (x_C - x_D)(y_B - y_A)}$$



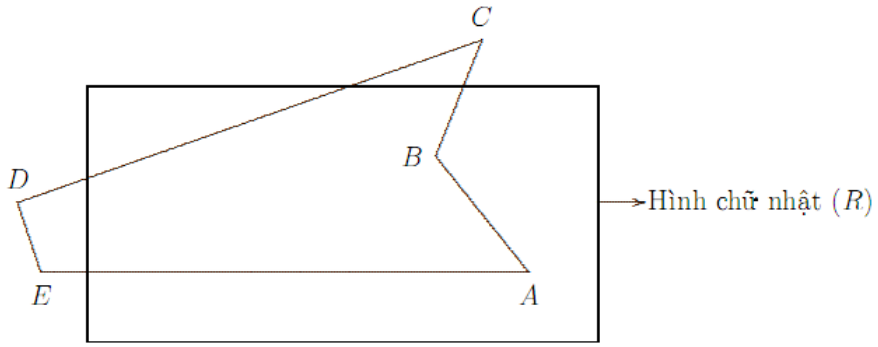
Hình 5.2: Biểu diễn tham số của đoạn thẳng

5.3. Đoạn thẳng và hình chữ nhật

Vị trí tương đối của đoạn thẳng và hình chữ nhật (*R*) có bốn trường hợp sau:

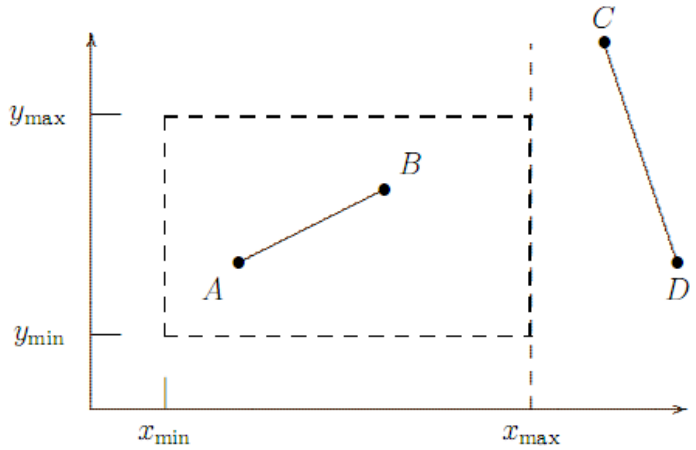
1. Cả hai đầu mút của đoạn thẳng nằm trong hình chữ nhật, chẳng hạn AB.
2. Một trong hai đầu mút của đoạn thẳng nằm trong hình chữ nhật, chẳng hạn BC.
3. Cả hai đầu mút của đoạn thẳng nằm ngoài hình chữ nhật nhưng có giao điểm, chẳng hạn CD.

4. Cả hai đầu mút của đoạn thẳng nằm ngoài hình chữ nhật và không có giao điểm, chẳng hạn DE.



Hình 5.3: Các trường hợp giao của đoạn thẳng và hình chữ nhật

Hai trường hợp 1 và 4 gọi là các trường hợp tầm thường, tức là xác định được ngay có tồn tại giao điểm hay không. Các trường hợp còn lại ta phải tiến hành thuật toán xác định giao điểm sẽ được trình bày trong phần tiếp theo sau.



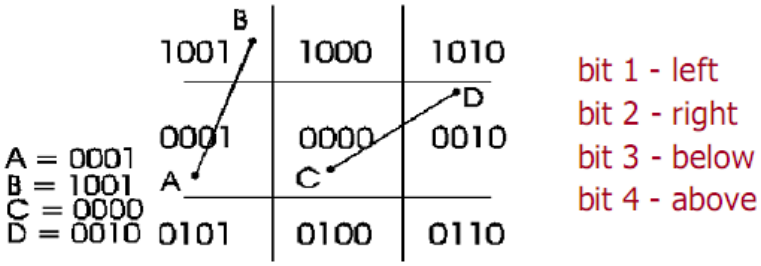
Hình 5.4: Hai trường hợp tầm thường của giao đoạn thẳng và hình chữ nhật

5.3.1 Tìm giao bằng cách giải hệ phương trình

Đưa bài toán về xác định giao điểm của hai đoạn thẳng được trình bày trong phần 3.2. Theo phương pháp này, chúng ta cần tính toán và kiểm tra nhiều khả năng; do đó không hiệu quả.

5.3.2 Thuật toán chia nhị phân

Một tiếp cận là dựa trên cơ chế đánh mã được phát triển bởi Cohen và Sutherland. Mọi điểm ở hai đầu mút đoạn thẳng trong hình ảnh sẽ được gán một mã nhị phân 4 bit, được gọi là mã vùng, giúp nhận ra vùng tọa độ của một điểm. Các vùng này được xây dựng dựa trên sự xem xét với biên cửa sổ, như ở hình 6-8. Mỗi vị trí bit trong mã vùng được dùng để chỉ ra một trong bốn vị trí tọa độ tương ứng của điểm so với cửa sổ: bên trái (left), phải (right), trên đỉnh (top), dưới đáy (bottom). Việc đánh số theo vị trí bit trong mã vùng từ 1 đến 4 cho từ phải sang trái, các vùng tọa độ có thể liên quan với vị trí bit như sau:



Hình 5.5: Mã hóa các đầu mút của đoạn thẳng

Giá trị 1 ở bất kỳ vị trí nào chỉ ra rằng điểm ở vị trí tương ứng, ngược lại bit ở vị trí đó là 0. Nếu một điểm nằm trong cửa sổ, mã vị trí là 0000. Một điểm bên dưới và bên trái cửa sổ có mã vùng là 0101.

Cài đặt minh họa thuật toán mã hóa

```
byte EncodePoint(Point LeftTop, Point RightBottom, Point P)
{
        byte code = 0;
        if (P.X < LeftTop.X)
        {
           code |= 8;
        }
        if (P.X > RightBottom.X)
        {
           code |= 4;
        }
        if (P.Y < RightBottom.Y)
```

```
        {
            code |= 2;
        }
        if (P.Y > LeftTop.Y)
        {
            code |= 1;
        }

        return code;

}
```

Thuật toán chia nhị phân

1. Nếu $E(A) = 0$ và $E(B) = 0$ kết luận $AB \cap (R) = AB$; thuật toán dừng.
2. Nếu [$E(A)$ AND $E(B)$] != 0 kết luận $AB \cap (R) = \emptyset$; kết thúc thuật toán.
3. Nếu $E(A) = 0$ và $E(B) \neq 0$(tức $A \in (R)$ và $B \notin (R)$) thực hiện:
    a. Đặt $C = A$, $D = B$.
    b. Trong khi độ dài $\|CD\|$ lớn hơn ε

       Đặt M là trung điểm của đoạn CD.

       Nếu $E(\mathrm{M}) = 0$ thì cập nhật $C = M$ ngược lại $D = M$.
    c. Kết luận $AB \cap (R) = AM$; kết thúc thuật toán.
4. Nếu $E(A) = 0$ và $E(B) = 0$, hoán đổi vai trò của A và B; lặp lại bước 3.
5. Ngược lại, thực hiện:
    a. Đặt $C = A$, $D = B$.
    b. Trong khi độ dài $\|CD\|$ lớn hơn ε

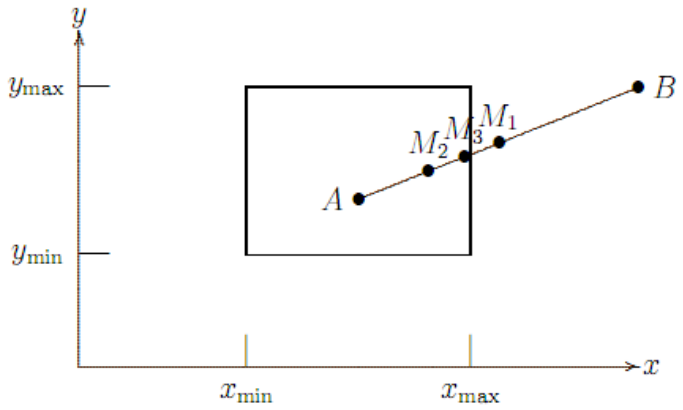
       Đặt M là trung điểm của đoạn CD.

       Nếu $E(\mathrm{M}) = 0$ áp dụng Bước 3 cho hai đoạn MC và MD. Kết luận $AB \cap (R) = CD$;kết thúc thuật toán.

       Nếu [$E(\mathrm{M})$ AND $E(R)$] != 0 đặt $C = M$.

Nếu [E(M) AND E(D)] != 0 đặt $D = M$.

Nếu [$E(R)$ AND E(D)] != 0 kết luận $AB \cap (R) = \emptyset$; kết thúc thuật toán.

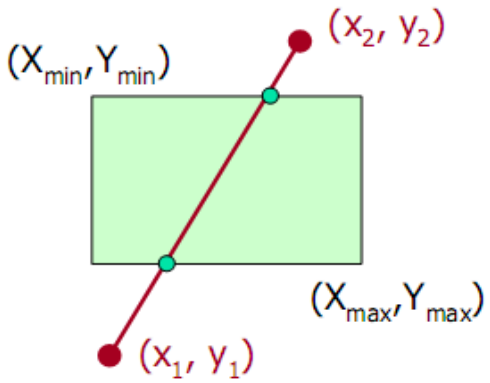


Hình 5.6: Minh họa của thuật toán chia nhị phân.

5.3.3 Thuật toán Cohen-Sutherland

Xác định nhanh đoạn thẳng có cần cắt xén hay không nhờ các phép toán logíc AND và OR:

- Kết quả phép OR hai mã đầu mút đoạn thẳng cho kết quả 0: cả hai điểm nằm trong chữ nhật.
- Kết quả phép AND hai mã đầu mút đoạn thẳng cho kết quả khác 0: cả hai điểm nằm ngoài chữ nhật.



Hình 5.7: Đoạn thẳng giao với hình chữ nhật

Giao của đoạn thẳng với các cạnh chữ nhật song song trục tung:

- x có giá trị X_{min}, X_{max} và hệ số góc $a = (y_2 - y_1)/(x_2 - x_1)$

- $y = y_1 + a(x - x_1)$

Giao đoạn thẳng với các cạnh song song trục hoành:

- y có giá trị Y_{min}, Y_{max} và hệ số góc $a = (y_2 - y_1)/(x_2 - x_1)$
- $x = x_1 + (y - y_1)/a$

Cài đặt thuật toán Cohen-Shutherland

```
void InterLineRectangle(Point LeftTop, Point RightBottom, Point A, Point B)
{
   byte codeA, codeB, codeOut;
   float x = 0, y = 0;
   codeA = EncodePoint(LeftTop, RightBottom, A);
   codeB = EncodePoint(LeftTop, RightBottom, B);
   while (true)
   {
      if (codeA == 0 && codeB == 0)
      {
         return true;
      }
      if ((codeA & codeB) != 0)
      {
         return false;
      }

      if (codeA != 0) codeOut = codeA;
      else codeOut = codeB;

      if ((codeOut & 8) != 0)//L
      {
         x = LeftTop.X;
         y = A.Y + (float)((x - A.X) * (B.Y - A.Y)) / (float)(B.X - A.X);
      }
```

```
        else if ((codeOut & 4) != 0) //R
        {
            x = RightBottom.X;
            y = A.Y + (float)((x - A.X) * (B.Y - A.Y)) / (float)(B.X - A.X);
        }
        else if ((codeOut & 2) != 0)//B
        {
            y = RightBottom.Y;
            x = A.X + (float)((y - A.Y) * (B.X - A.X)) / (float)(B.Y - A.Y);
        }
        else if ((codeOut & 1) != 0)//T
        {
            y = LeftTop.Y;
            x = A.X + (float)((y - A.Y) * (B.X - A.X)) / (float)(B.Y - A.Y);
        }

        if (codeOut == codeA)
        {
            A.X = (int)x;
            A.Y = (int)y;
            codeA = EncodePoint(LeftTop, RightBottom, A);
        }
        else
        {
            B.X = (int)x;
            B.Y = (int)y;
            codeB = EncodePoint(LeftTop, RightBottom, B);
        }
    }
}
```

5.3.4 Thuật toán Liang-Barsky

Một thuật toán tìm giao đoạn thẳng và hình chữ nhật hiệu quả dùng phương trình tham số đã được phát triển bởi Liang và Barsky. Nhận xét rằng hình chữ nhật (*R*) gồm tập các điểm nằm trong mặt phẳng giới hạn bởi các đường thẳng qua các cạnh giới hạn của nó; tức là:

$$(R) = \{(x, y) \in R^2 | x_{min} \leq x \leq x_{max}, y_{min} \leq y \leq y_{max}\}$$

Phương trình tham số của đoạn *AB*: $P(t) := A + t\ (B - A)$ với $t \in [0, 1]$. Như vậy bài toán đưa về xác định các giá trị tham số t thỏa hệ các bất phương trình sau:

$$\begin{cases} x_{min} \leq x_A + t(x_B - x_A) \leq x_{max} \\ y_{min} \leq y_A + t(y_B - y_A) \leq y_{max} \\ 0 \leq t \leq 1 \end{cases}$$

Hay tương đương : $q_i \leq p_i t$, $i = 0,1, 2, 3$. Trong đó :

$p_0 = dx,$ $q_0 = x_{min} - x_A$

$p_1 = -dx,$ $q_1 = x_A - x_{max}$

$p_2 = dy,$ $q_2 = y_{min} - y_A$

$p_3 = -dy,$ $q_3 = y_A - y_{Amax}$

với $dx = x_B - x_A$; $dy = y_B - y_A$.

Nếu tồn tại chỉ số $i \in \{0, 1, 2, 3\}$ sao cho $p_i = 0$ và $q_i > 0$ thì hệ bất phương trình trên vô nghiệm ; tức là đoạn thẳng *AB* giao với hình nhật bằng rỗng. Ngược lại, ta đặt :

$$t_0 := max\{0, max\{\frac{q_i}{p_i} | p_i > 0, i = 0, 1, 2, 3\}\}$$

$$t_1 := min\{1, min\{\frac{q_i}{p_i} | p_i < 0, i = 0, 1, 2, 3\}\}.$$

Khi đó hệ bất phương trình tương đương với : $t_0 \leq t \leq t_1$. Điều này chỉ ra rằng, điều kiện để đoạn thẳng *AB* có giao điểm với hình chữ nhật (*R*) là $t_0 \leq t_1$:

$$AB \cap (R) = \{P(t) | t \in [t_0, t_1]\}$$

Từ đó, ta có thuật toán sau :

Bước 1: Tính p_i, q_i với $i = 0, 1, 2, 3$.

Bước 2: Khởi tạo $i = 0$ và $t_0 = 0$, $t_1 = 1$.

Bước 3: Nếu $p_i = 0$ và $q_i > 0$ thì kết luận $AB \cap (R) = \emptyset$; kết thúc thuật toán.

Bước 4: Nếu $p_i > 0$ đặt $t_0 := max\{t_0, \frac{q_i}{p_i}\}$. Chuyển sang bước 7.

Bước 5: Nếu $p_i < 0$ đặt $t_1 := min\{t_1, \frac{q_i}{p_i}\}$. Chuyển sang bước 7.

Bước 6: Nếu $t_0 > t_1$ thì kết luận $AB \cap (R) = \emptyset$; kết thúc thuật toán.

Bước 7: Nếu $i < 3$ thay $i = i + 1$ và lặp lại bước 3; Ngược lại, kết luận $AB \cap (R) = CD$; trong đó :

$$C = A + t_0(B - A)$$

$$D = A + t_1(B - A)$$

Ví dụ, xét hình chữ nhật :

$$(R) = \{(x, y) \in R^2 | 0 \le x \le 8, 0 \le y \le 4\}$$

và hai điểm $A(1, -2)$ và $B(10, 9)$. Ta có phương trình tham số của đoạn thẳng AB là: $A + t(B - A)$. Ta cần giải hệ phương trình:

$$\begin{cases} 0 \le 1 + t(10 + 1) \le 8 \\ 0 \le -2 + t(9 + 2) \le 4 \\ 0 \le t \le 1 \end{cases}$$

Hay tương đương :

$$\begin{cases} 1 \le 11t \le 8 \\ 2 \le 11t \le 6 \\ 0 \le t \le 1 \end{cases}$$

Vậy

$$\frac{2}{11} \le t \le \frac{6}{11}$$

Suy ra, giao điểm của AB và hình chữ nhật là đoạn thẳng CD, trong đó :

$$C = A + \frac{2}{11}(B - A) = (1,-2) + \frac{2}{11}[(10,9) - (1,-2)] = (2\frac{7}{11},0)$$

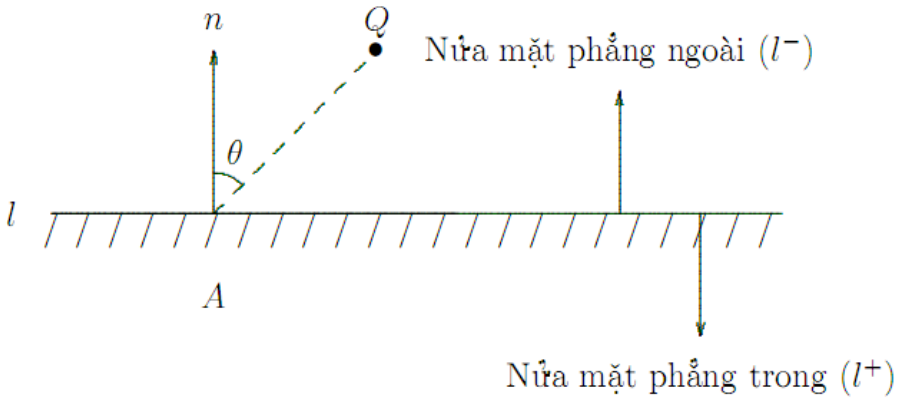
$$D = A + \frac{6}{11}(B - A) = (1,-2) + \frac{6}{11}[(10,9) - (1,-2)] = (5\frac{10}{11},4)$$

Thuật toán Liang và Barsky giảm bớt các tính toán cần thiết để cắt các đoạn. Mỗi lần cập nhật t_0 và t_1 chỉ cần một phép chia, và các giao điểm với cửa sổ được tính chỉ một lần, khi mà các giá trị t_0 và t_1 vừa hoàn thành. Ngược lại, thuật toán của Cohen-Sutherland lặp lại việc tính giao điểm của đoạn với các biên cửa sổ, và mỗi phép tính giao điểm cần cả hai phép chia và nhân.

5.4. Giao của đoạn thẳng và đa giác lồi

Ví trị tương đối của một điểm với đoạn thẳng

Trong nhiều ứng dụng, ta quan tâm đến khái niệm nửa mặt phẳng trong và nửa mặt phẳng ngoài xác định bởi một đoạn thẳng. Khái niệm này liên quan mật thiết đến pháp vector của đoạn thẳng.



Hình 5.8: Ví trị tương đối của điểm Q với đoạn thẳng l.

Phương trình tổng quát của đoạn thẳng *l* có dạng *a.x + b.y + c = 0*. Ký hiệu:

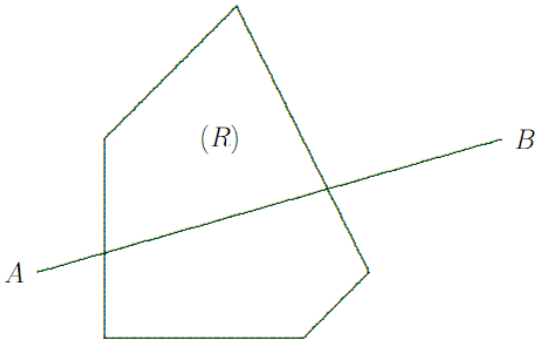
$$(l^+) := \{P \in R^2 | \langle \overrightarrow{OP}, n \rangle \geq D\}$$

$$(l^-) := \{P \in R^2 | \langle \overrightarrow{OP}, n \rangle < D\}$$

là các nửa mặt phẳng ngoài và nửa mặt phẳng trong xác định bởi *l*, trong đó *D = -c*, *n = (a, b)*^{t}. Tiêu chuẩn để kiểm tra điểm Q thuộc một nửa phẳng nào của đoạn thẳng *l:*

1. $Q \in (l^+)$ *nếu* $\langle n, \overrightarrow{OQ} \rangle > D$
2. $Q \in (l^-)$ *nếu* $\langle n, \overrightarrow{OQ} \rangle < D$
3. $Q \in l$ *nếu* $\langle n, \overrightarrow{OQ} \rangle = D$

Thuật toán xác định giao điểm đoạn thẳng và đa giác lồi



Hình 5.9: Giao của đoạn thẳng và đa giác lồi

Thuật toán *Cyrus-Beck* dựa trên tiêu chuẩn loại bỏ đơn giản bằng cách xác định vị trí tương đối của một điểm với một đoạn thẳng. Giả sử đa giác lồi *(R)* được định nghĩa như một dãy các đỉnh $P_i = (x_i, y_i)$, i=0, 1,..., L, trong hệ tọa độ thực với $P_0 = P_L$. Mục đích của phần này là loại bỏ những phần của đoạn AB không nằm trong cửa sổ (*R) (Hình 5.9).*

Ký hiệu l_i, i = 0, 1,..., L, là đoạn thẳng đi qua hai đỉnh liên tiếp P_i, P_{i+1}. Đặt:

$$(l_i^+) := \{P \in R^2 | \langle \overrightarrow{OP}, n_i \rangle > D_i\}$$

$$(l_i^-) := \{P \in R^2 | \langle \overrightarrow{OP}, n_i \rangle < D_i\}$$

là các nửa mặt phẳng ngoài và mặt phẳng trong xác định bởi l_i, trong đó n_i là pháp vector của l_i được chọn hướng ra nửa mặt phẳng ngoài và D_i là hằng số nào đó. Vì (*R*) là tập lồi nên được xác định bởi:

$$(R) = \bigcap_{i=0}^{L} (l_i^-)$$

Ý tưởng của thuật toán như sau:

- Với mỗi đoạn thẳng l_i, $i = 0, 1, ..., L$, chúng ta loại bỏ phần của đoạn thẳng AB thuộc nửa mặt phẳng ngoài xác định bởi l_i và cập nhật AB=AB∩(l_i^-).
- Nếu tại bước nào đó, AB ⊂ (l_i^+) thì kết luận giao của đoạn thẳng và đa giác lồi bằng trống; ngược lại, nếu ở bước cuối cùng phần đoạn thẳng AB còn lại nằm trong (*R*) chính là phần giao cần tìm.

Thuật toán

```
void  Cyrus_Beck(Point *A, Point *B, VertPtr Poly)
{
        float t_in = 0.0, t_out = 1.0, t_hit, Denom, D;
        Point F, S;
        Vector c, n, a;
        VertPtr Tempt = Poly;
        if (Tempt == NULL)
                return False;
        F = Tempt->Vertex;
        c.dx = (*B).x - (*A).x;
        c.dy = (*B).y - (*A).y;
        a.dx = (*A).x;
        a.dy = (*A).y;
        while ((Tempt = Tempt->Next) != NULL)
        {
                S = Tempt->Vertex;
                n.dx = (S.y - F.y);
                n.dy = -(S.x - F.x);
                D = n.dx*F.x + n.dy*F.y;
                if ((Denom = Dot2D(n, c)) == 0.0)
                        if (Dot2D(n, a) > D)          return false;
                else
                {
                        t_hit = (D - Dot2D(n, a)) / Denom;
                        if (Denom > 0.0)
```

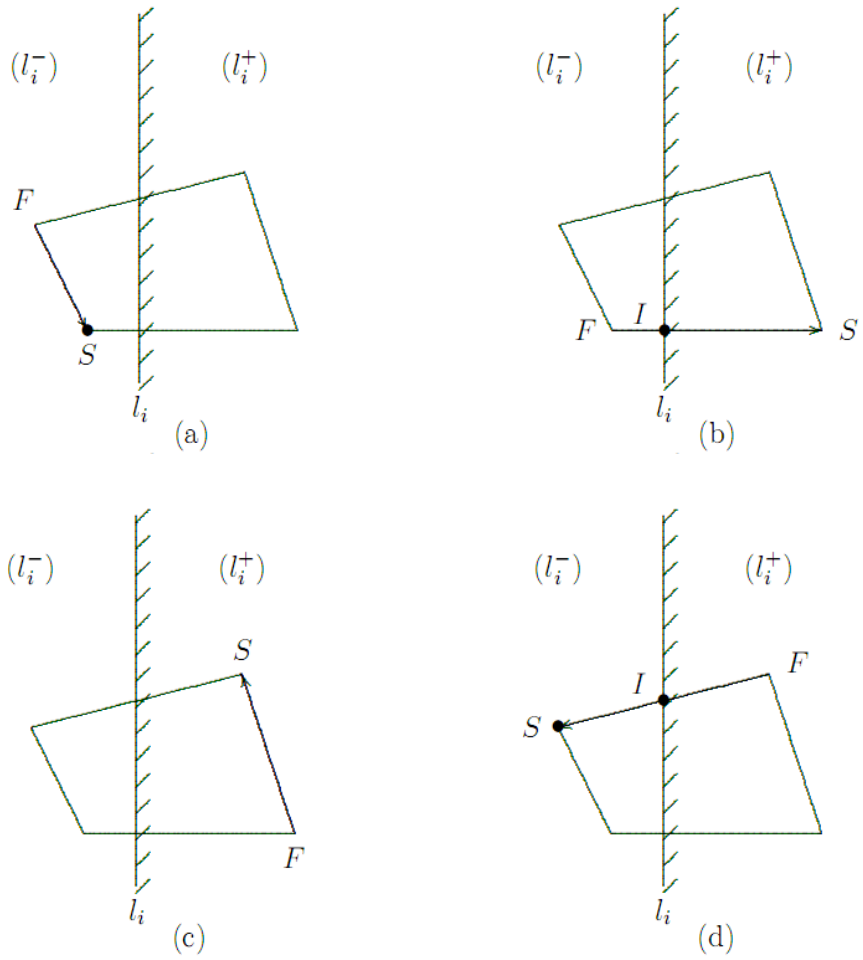
```
                    if (t_out > t_hit) t_out = t_hit;
                else
                    if (t_in < t_hit) t_in = t_hit;
                if (t_in > t_out) return false;
            }
            F=S;
        }
        F.x = (1 - t_in)*(*A).x + t_in*(*B).x;
        F.y = (1 - t_in)*(*A).y + t_in*(*B).y;
        S.x = (1 - t_out)*(*A).x + t_out*(*B).x;
        S.y = (1 - t_out)*(*A).y + t_out*(*B).y;
        *A = F;
        *B = S;
        return true;
}
```

5.5. Giao hai đa giác

Thuật toán Sutherland-Hodgman

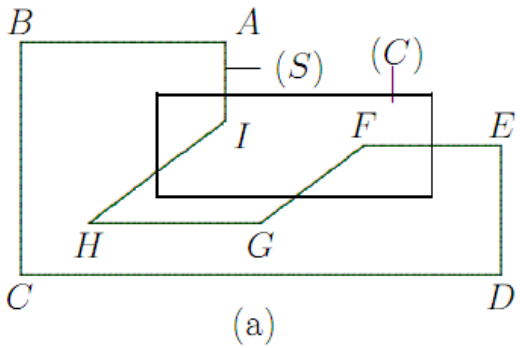
Ký hiêu Subj và Clip là danh sách các đỉnh của hai đa giác *(S)* và *(C)* tương ứng. Có bốn khả năng xảy ra giữa mỗi cạnh của *(S)* và của *(C):*

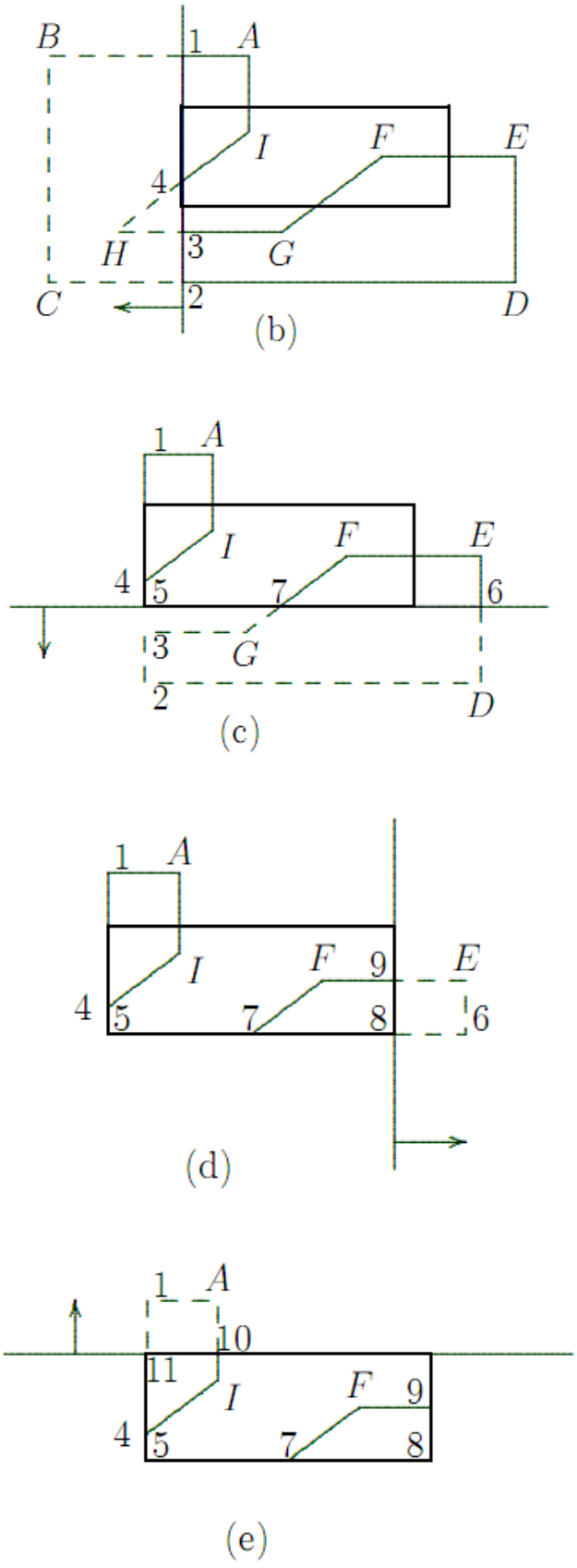
1. Hai đỉnh F và S nằm trong: xuất S.
2. Đỉnh F nằm trong và S nằm ngoài: tìm giao điểm I và xuất nó.
3. Hai đỉnh F và S nằm ngoài: không xuất.
4. Đỉnh F nằm ngoài và S nằm trong: tìm giao điểm I; xuất I và S.



Hình 5.10: Bốn trường hợp với mỗi cạnh của (S)

Xét ví dụ hình *(a):*





Hình 5.11: Ví dụ thuật toán Sutherland-Hodgman

1. Danh sách *Subj* sau khi cắt *(S)* với cạnh bên trái của *(C)*:

   (1, 2, D, E, F, G, 3, 4, I, A, 1).

2. Danh sách *Subj* sau khi cắt *(S)* với cạnh bên dưới của *(C)*:

   (5, 6, E, F, 7, 5, 4, I, A, 1, 5).

3. Danh sách *Subj* sau khi cắt *(S)* với cạnh bên phải của *(C)*:

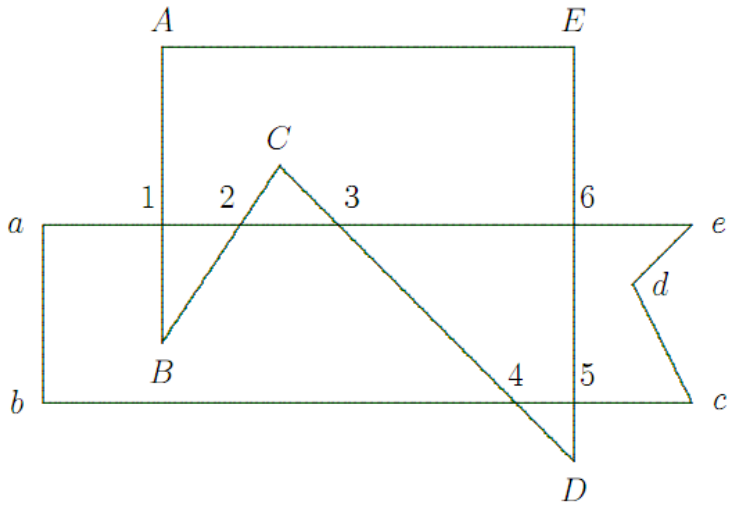
   (8, 9, F, 7, 5, 4, I, A, 1, 5, 8).

4. Danh sách *Subj* sau khi cắt *(S)* với cạnh bên trên của *(C)*:

   (9, F, 7, 5, 4, I, 10, 11, 5, 8, 9).

Thuật toán Weiler-Atherton

Cách tiếp cận của Weiler-Atherton nhằm tìm ra giao của hai đa giác bất kỳ, thậm chí có lỗ hổng trong các đa giác. Ngoài ra có thể tìm phần hợp và hiệu hai đa giác nữa. Xét ví dụ hình 5.12 sau:



Hình 5.12: Ví dụ thuật toán Weiler-Atherton

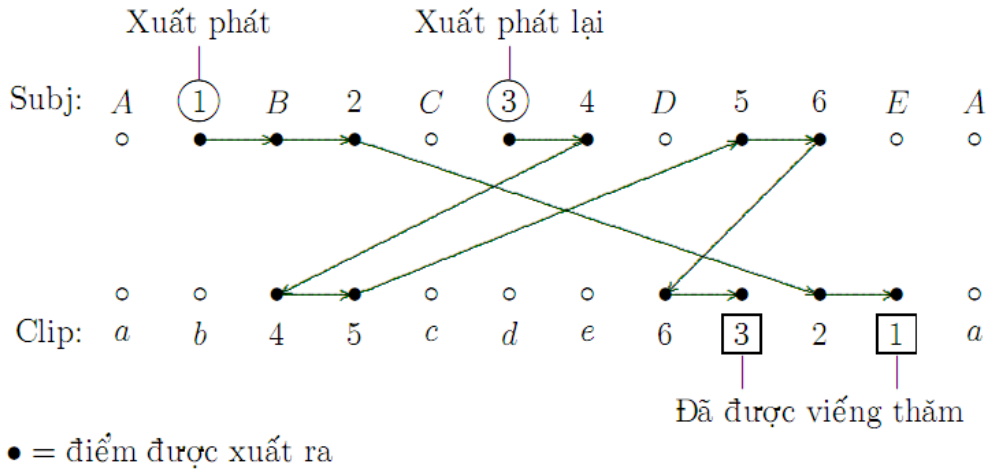
Hai đa giác *(S)* và *(C)* được biểu diễn bởi danh sách các đỉnh, ký hiệu *Subj = (A, B, C, D, E, A)* và *Clip = (a, b, c, d, e, a)* tương ứng.

- Tất cả các giao điểm của hai đa giác được xác định và lưu vào một danh sách. Trong ví dụ trên có tất cả sáu giao điểm: *1, 2, 3, 4, 5, 6.*
- Thực hiện tiến trình: "lần theo hướng thuận và nhảy" là xây dựng hai danh sách:

  Subj: (*A*, 1, *B*, 2, *C*, 3, 4, *D*, 5, 6, *E, A*)

  Clip: (*a, b*, 4, 5, *c, d, e*, 6, 3, 2, 1, *a*)

Xuất phát từ giao điểm *"đi vào"* là điểm đi từ ngoài vào trong của đa giác (*C*), duyệt trên (*S*) đến khi gặp giao điểm thì chuyển sang duyệt trên (*C*), và lặp lại. Quá kết thúc khi gặp điểm xuất phát ban đầu. Tiếp tục kiểm tra giao điểm trên (*S*) chưa được đi qua và lặp lại tiến trình trên. Ta có hai đa giác sinh ra là (1, *B*, 2, 1) và (3, 4, 5, 6, 3).



Hợp hai đa giác (*S*) ∪ (*C*)

Đi trên (*S*) theo hướng thuận cho đến khi gặp "*điểm ra*" là điểm đi từ trong ra ngoài của đa giác (*C*) duyệt cho đến khi gặp giao điểm khác với (*C*) thì duyệt sang (*C*) cho đến khi gặp giao điểm kế tiếp rồi chuyển sang (*S*). Quá trình kết thúc khi gặp điểm xuất phát ban đầu. Kết quả (*S*) ∪ (*C*) gồm hai đa giác:

- (2, *C*, 3, 2) (lỗ hỗng).
- (4, *D*, 5, *c, d, e*, 6, *E, A*, 1, *a, b*, 4).

Hiệu hai đa giác (*C*) \ (*S*)

Đi trên (*S*) theo hướng thuận cho đến khi gặp "*điểm vào"* duyệt cho đến khi gặp giao điểm khác với (*C*) thì duyệt sang (*C*) theo hướng ngược cho đến khi gặp giao điểm kế tiếp rồi chuyển sang (*S*). Quá kết thúc khi gặp điểm xuất phát ban đầu.

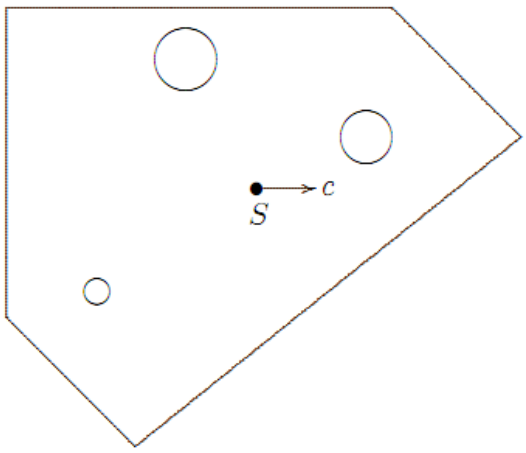
- (*C*) \ (*S*)*:* (1, *B,* 2, 3, 4, *b, a,* 1)*; và* (5, 6, *e, d, c*, 5).
- (*S*) \ (*C*)*:* (4, 5, *D,* 4)*; và* (6, 3*, C,* 2, 1, *A, E*, 6).

5.6. Kỹ thuật Ray tracing

Kỹ thuật Ray tracing là phương pháp áp dụng một số khái niệm hình học để tạo ứng dụng đồ họa mô phỏng quá trình chuyển động của tia sáng trong buồng kín. Kỹ thuật Ray tracing là một công cụ quan trọng trong đồ họa máy tính để tổng hợp hình ảnh. Trong tổng hợp hình ảnh, các tia sáng nhân tạo "lần theo" trong thế giới thực ba chiều chứa nhiều đối tượng. Đường đi của mỗi tia sáng xuyên qua các đối tượng hoặc phản xạ lại tùy theo mức độ phản xạ của đối tượng cho đến khi nó dừng lại ở đối tượng nào đó. Màu của đối tượng này sẽ được đặt cho pixel tương ứng trên thiết bị hiển thị. Mô phỏng quá trình Ray tracing rất dễ dàng trong không gian hai chiều.

Ta xét quỹ đạo của trái *pinpall* nhỏ khi va chạm vào các đối tượng trong *buồng kín.* Hình bên dưới minh họa nhát cắt ngang của một buồng kín có năm bức tường và chứa ba "trụ tròn". Trái *pinpall* bắt đầu tại vị trí *S* va di chuyển theo hướng vector *c* cho đến gặp vật cản sẽ bị phản xạ và di chuyển theo hướng mới. Quá trình được lặp lại nhiều lần. Quỹ đạo của trái *pinpall* là đường gấp khúc mà ta có thể hình dung đường đi của tia sáng di chuyển trong buồng kín.

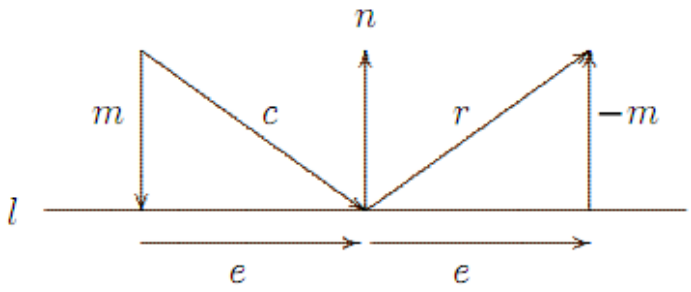
Bây giờ ta sẽ xây dựng thuật toán xác định đối tượng giao với tia sáng sẽ trước và vị trí va trạm tại đó. Ví trí va chạm là điểm khởi đầu cho cho đường đi kế tiếp với hướng di chuyển mới.



Hình 5.13: Ví dụ về Ray tracing.

Vector phản xạ

Hình 5.14 phân giải vector *c* gồm thành phần *m* dọc theo *n* và thành phần *e* vuông gốc với *n*. Ta có, $r = e - m$. Nhưng $e = c - m$ nên $r = c - 2m$.



Hình 5.14: Phản xạ tia sáng

Vì vector *m* là vector phân giải của vector *c* theo vector *n* nên

$$m = \frac{\langle c,n \rangle}{||n||^2} n \quad , \quad r = c - 2\frac{\langle c,n \rangle}{||n||^2} n$$

Thủ tục xác định vector phản xạ r từ vector c và pháp vector n

```
void Reflection(Vector c, Vector n, Vector *r)
{
```

float Coeff = 2.Dot2D(c, n)/Dot2D(n, n);

*(*r).dx = c.dx – Coeff.n.dx;*

*(*r).dy = c.dy – Coeff.n.dy;*

}

Giao của tia sáng và đường thẳng

Phương trình tham số của tia sáng xuất phát từ S chuyển động theo vector chỉ phương c cho bởi

$$R(t) := S + c.t,\ t \geq 0.$$

Bức tường tương ứng đường thẳng:

$$(l) := \{P \in R^2 | \langle n, \overrightarrow{OP} \rangle = D\}$$

trong đó, pháp vector n hướng ra ngoài buồng kín. Nếu $\langle n, c \rangle \neq 0$ thì tia sáng cắt đường thẳng tại $P_h = S + ct_h$, trong đó:

$$t_h = \frac{D - \langle n, \overrightarrow{OS} \rangle}{\langle n, c \rangle}$$

Ví dụ, cho đường thẳng $6x - 8y + 10 = 0$ và tia sáng xuất phát từ $S(2, 4)$ di chuyển theo vector chỉ phương $c = (-2, 1)^t$. Đường thẳng có $n = (6,-8)^t$ và $D = -10$. Ta có:

$$t_h = \frac{D - \langle n, \overrightarrow{OS} \rangle}{\langle n, c \rangle} = \frac{-10 - (12 - 32)}{(-12 - 8)} = \frac{1}{2}$$

Suy ra giao điểm I:

$$I = S + t.c = (2, 4) + (-2, 1) \times \frac{1}{2} = (1, \frac{9}{2}).$$

Vector phản xạ:

$$r = c - 2\frac{\langle c,n\rangle}{||n||^2}n = \begin{pmatrix}-2\\1\end{pmatrix} - 2\frac{-20}{100}\begin{pmatrix}6\\8\end{pmatrix} = \frac{1}{5}\begin{pmatrix}2\\11\end{pmatrix}$$

Thủ tục xác định thời điểm giao t_h

```
void  Ray_With_Line(Point S, Vector c, Vector normal, float D, float *t_hit)
{
    float Denom = Dot2D(normal, c);
    Vector2D s;
    PointToVector2D(S, &s);
    if (Denom == 0.0)
        *t_hit = -1.0;
    else
        *t_hit = (D - Dot2D(normal, s))/Denom;
}
```

Giao của tia sáng và hình tròn

Hình trụ tương ứng đường tròn *(C)* bán kính *R* tâm *I.* Xét sự tương giao của tia sáng và đường tròn. Ta có:

$$||\overrightarrow{IS} + c.t||^2 = R^2$$

Suy ra:

$$||\overrightarrow{IS}||^2 + 2\langle\overrightarrow{IS}, c\rangle t + ||c||^2 t^2 = R^2$$

Hay tương đương:

$$A.t^2 \mid 2B.t \mid C = 0$$

trong đó, $A = ||c||^2, B = \langle \overrightarrow{IS}, c \rangle, C = ||\overrightarrow{IS}||^2 - R^2$.

Nghiệm của phương trình (có thể là nghiệm ảo)

$$t_h = -\frac{B}{A} \pm \frac{\sqrt{B^2 - A.C}}{A}$$

Ví dụ, cho đường tròn $(x - 1)^2 + (y - 4)^2 = 4$ và tia sáng xuất phát từ $S(8, 9)$ di chuyển theo vector chỉ phương $c = (-1, -1)^t$.

Phương trình giao điểm:

$$A.t^2 + 2B.t + C = 0,$$

trong đó, $A = ||c||^2 = 2, B = \langle \overrightarrow{IS}, c \rangle = -12, C = ||\overrightarrow{IS}||^2 - R^2 = 70$.

Suy ra:

$$t_h = -\frac{B}{A} \perp \frac{\sqrt{B^2 - A.C}}{A} - 5$$

Vậy tọa độ giao điểm là $P_h = S + t_h.c = (8 - 5, 9 - 5) = (3, 4)$.

Vector phản xạ:

$$r = c - 2\frac{\langle c, n \rangle}{||n||^2} n = \begin{pmatrix} -1 \\ -1 \end{pmatrix} - 2\frac{-2}{4}\begin{pmatrix} 2 \\ 0 \end{pmatrix} = \begin{pmatrix} 1 \\ -1 \end{pmatrix}$$

Thủ tục xác định thời điểm giao

```
void Ray_With_Circle(Point S, Vector c, Point Center, float Rad,float *t_hit)
{
    float A, B, C, delta;
    Vector tempt;
```

```
        tempt.dx = S.x - Center.x;
        tempt.dy = S.y - Center.y;
        A = Dot2D(c, c);
        B = Dot2D(tempt, c);
        C = Dot2D(tempt, tempt) - Rad*Rad;
        delta = B*B - A*C;
        if (delta < 0.0)
                *t_hit = -1.0;
        else
                *t_hit = (-B - sqrt(delta))/A;
}
```

Chương 6

ĐỒ HỌA BA CHIỀU

Nội dung chính

- Giới thiệu đồ họa 3 chiều (3D).
- Hiển thị đối tượng 3D.
- Các phép biến đổi Affine 3D cơ sở.

6.1. Giới thiệu đồ họa 3 chiều

Các đối tượng trong thế giới thực phần lớn là các đối tượng 3 chiều còn thiết bị hiển thị chỉ 2 chiều. Do vậy, muốn có hình ảnh 3 chiều ta cần phải giả lập. Chiến lược cơ bản là chuyển đổi từng bước. Hình ảnh sẽ được hình thành từ từ, ngày càng chi tiết hơn.

Qui trình hiển thị ảnh 3 chiều như sau

- Biến đổi từ hệ tọa độ đối tượng sang hệ tọa độ thế giới thực**.** Mỗi đối tượng được mô tả trong một hệ tọa độ riêng được gọi là hệ tọa độ đối tượng.

  Có 2 cách mô hình hóa đối tượng:

  - Solid modeling: mô tả các vật thể (kể cả bên trong).
  - Boudary representation: chỉ quan tâm đến bề mặt đối tượng.

Các đối tượng có thể được biểu diễn bằng mô hình Wire-Frame. Nhận thấy rằng khi biểu diễn đối tượng, ta có thể chọn gốc tọa độ và đơn vị đo lường sao cho việc biểu diễn là thuận lợi nhất. Thường thì người ta chuẩn hóa kích thước của đối tượng khi biểu diễn. Biểu diễn biên cho phép xử lý nhanh còn silid modeling cho hình ảnh đầy đủ và xác thực hơn.

- Loại bỏ các đối tượng không nhìn thấy được: Loại bỏ các đối tượng hoàn toàn không thể nhìn thấy trong cảnh. Thao tác này giúp ta lược bỏ bớt các đối tượng không cần thiết do đó giảm chi phí xử lý.
- Chiếu sáng các đối tượng: Gán cho các đối tượng màu sắc dựa trên các đặc tính của các chất tạo nên chúng và các nguồn sáng tồn tại trong cảnh. Có nhiều mô hình chiếu sáng và tạo bóng : constant-intensity, Interpolate,...
- Chuyển từ word *space* sang *eye space*. Thực hiện một phép biến đổi hệ tọa độ để đặt vị trí quan sát về gốc tọa độ và mặt phẳng quan sát về một vị trí mong ước.

  Hình ảnh hiển thị phụ thuộc vào vị trí quan sát và góc nhìn.
  Hệ qui chiếu có gốc đặt tại vị trí quan sát và phù hợp với hướng nhìn sẽ thuận lợi cho các xử lý thật.

- Loại bỏ phần nằm ngoài: Thực hiện việc xén đối tượng trong cảnh để cảnh nằm gọn trong một phần không gian hình chóp cụt giới hạn vùng quan sát mà ta gọi là **viewing frustum**. Viewung frustum có trục trùng với tia nhìn, kích thước giới hạn bởi vùng ta muốn quan sát.
- Chiếu từ không gian nhìn xuống không gian màn hình: Thực hiện việc chiếu cảnh 3 chiều từ không gian quan sát xuống không gian màn hình. Có 2 phương pháp chiếu là phép chiếu song song và phép chiếu phối cảnh. Khi chiếu ta phải tiến hành việc khử mặt khuất để có thể nhận được hình ảnh trung thực. Khử mặt khuất cho phép xác định vị trí (x, y) trên màn hình thuộc về đối tượng nào trong cảnh.
- Chuyển đối tượng sang dạng pixel.
- Hiển thị đối tượng.

6.2. Biểu diễn đối tượng 3 chiều

Trong đồ họa máy tính, các đối tượng lập thể có thể được mô tả bằng các bề mặt của chúng. Ví dụ : một hình lập phương được xây dựng từ sáu mặt phẳng, một hình trụ được xây dựng từ sự kết hợp của một mặt cong và hai mặt phẳng và hình cầu được xây dựng từ chỉ một mặt cong. Thông thường để biểu diễn một đối tượng bất kỳ, người ta dùng phương pháp xấp xỉ để đưa các mặt về dạng các mặt đa giác.

• **Điểm** trong không gian 3 chiều có tọa độ (x,y,z) mô tả một vị trí trong không gian.

```
typedef struct {
        int x;
        int y;
        int z;
} Point _3D ;
```

• **Vectơ** : xác định bởi 3 tọa độ dx, dy, dz mô tả một hướng và độ dài của véc tơ.

Véc tơ không có vị trí trong không gian.

$$|\vec{V}| = \sqrt{dx^2 + dy^2 + dz^2}$$

Tích vô hướng của hai véc tơ

$$V_1 * V_2 = dx_1 dx_2 + dy_1 dy_2 + dz_1 dz_2$$

Hay $V_1 * V_2 = |V_1||V_2| \cos\theta$

```
typedef struct {
        int dx;
        int dy;
        int dz;
} Vector ;
```

• **Đoạn thẳng** trong không gian 3 chiều: biểu diễn tổ hợp tuyến tính của 2 điểm

Để biểu diễn dạng tham số của đoạn thẳng, ta có :

$$P = P_1 + t(P_2 - P_1),\ (0 \le t \le 1)$$

```
typedef struct {
        Point P1;
        Point P2;
} Segment ;
```

• **Tia (Ray)** : là một đoạn thẳng với một đầu nằm ở vô cực.

Biểu diễn dạng tham số của tia :

$P = P_1 + t.V$, ($0 \leq t < \infty$)

```
typedef struct {
    Point P1;
    Vector V;
} Ray;
```

• **Đường thẳng (Line)**: là một đoạn thẳng với cả hai đầu nằm ở vô cực

Biểu diễn dạng tham số của đường thẳng

$P = P_1 + t.V$, ($\infty \leq t < \infty$)

```
typedef struct {
    Point P1;
    Vector V;
} Line;
```

• **Đa giác (Polygon)** : là một vùng giới hạn bởi hạn dãy các điểm đồng phẳng .

```
typedef struct {
    Point *Points;
    int nPoints;
} Polygon;
```

Có thể biểu diễn một mặt đa giác bằng một tập họp các đỉnh và các thuộc tính kèm theo. Khi thông tin của mỗi mặt đa giác được nhập, dữ liệu sẽ được điền vào các bảng sẽ được dùng cho các xử lý tiếp theo, hiển thị và biến đổi.

Các bảng dữ liệu mô tả mặt đa giác có thể tổ chức thành hai nhóm : bảng hình học và bảng thuộc tính. Các bảng lưu trữ dữ liệu hình học chứa tọa độ các đỉnh và các tham số cho biết về định hướng trong không gian của mặt đa giác. Thông tin về thuộc

tính của các đối tượng chứa các tham số mô tả độ trong suốt, tính phản xạ và các thuộc tính kết cấu của đối tượng. Một cách tổ chức thuận tiện để lưu trữ các dữ liệu hình học là tạo ra 3 danh sách : một bảng lưu đỉnh, một bảng lưu cạnh và một bảng lưu đa giác. Trong đó:

- Các giá trị tọa độ cho mỗi đỉnh trong đối tượng được chứa trong bảng lưu đỉnh.
- Bảng cạnh chứa các con trỏ trỏ đến bảng đỉnh cho biết đỉnh nào được nối với một cạnh của đa giác.
- Cuối cùng là bảng lưu đa giác chứa các con trỏ trỏ đến bảng lưu cạnh cho biết những cạnh nào tạo nên đa giác.

• Mặt phẳng (Plane) :

```
typedef struct {
        Vector N;
        int d;
} Plane;
```

Phương trình biểu diễn mặt phẳng có dạng : $A.x + B.y + C.z + D = 0$. Trong đó (x, y, z) là một điểm bất kỳ của mặt phẳng và A, B, C, D là các hằng số diễn tả thông tin không gian của mặt phẳng.

Để xác định phương trình mặt phẳng, ta chỉ cần xác định 3 điểm không thẳng hàng của mặt phẳng này. Như vậy, để xác định phương trình mặt phẳng qua một đa giác, ta sẽ sử dụng tọa độ của 3 đỉnh đầu tiên (x_1,y_1), (x_2,y_2), (x_2,y_3) trong đa giác này.

Từ phương trình mặt phẳng trên, ta có:

$$A.x_k + B.y_k + C.z_k + D = 0 \text{ , } k = 0, 1, 2, 3.$$

Trong đó :

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_2 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad D = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_2 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

Khai triển các định thức trên ta có :

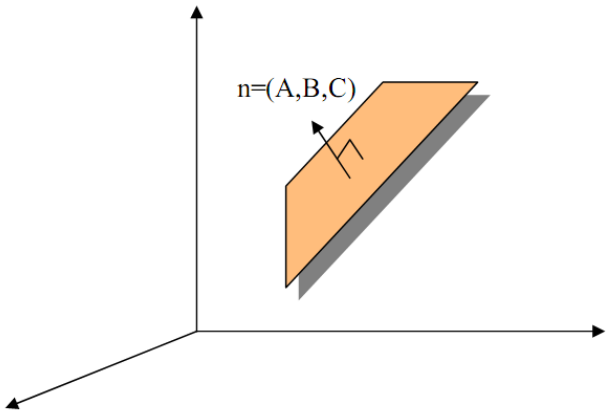
$$A = y_1 (z_2 - z_3) + y_2 (z_3 - z_1) + y_3 (z_1 - z_2)$$

$$B = z_1 (x_2 - x_3) + z_2 (x_3 - x_1) + z_3 (x_1 - x_2)$$

$$C = x_1 (y_2 - y_3) + x_2 (y_3 - y_1) + x_3 (y_1 - y_2)$$

$$A = - x_1 (y_2 z_3 - y_3 z_2) - x_2 (y_3 z_1 - y_1 z_3) - x_3 (y_1 z_2 - y_2 z_1)$$

Hướng của mặt phẳng thường được xác định thông qua véc tơ pháp tuyến của nó. Véc tơ pháp tuyến n = (A,B,C).

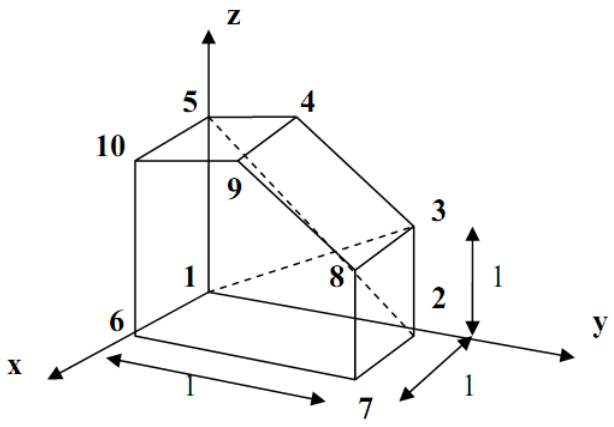


Hình 5.15: Mặt phẳng trong không gian

• Mô hình khung nối kết

Một phương pháp thông dụng và đơn giản để mô hình hóa đối tượng là mô hình khung nối kết. Một mô hình khung nối kết gồm có một tập các đỉnh và tập các cạnh nối các đỉnh đó. Khi thể hiện bằng mô hình này, các đối tượng 3 chiều có vẻ rỗng và không giống thực tế lắm. Tuy nhiên, vẽ bằng mô hình này thì nhanh nên người ta thường dùng nó trong việc xem phác thảo các đối tượng. Để hoàn thiện hơn, người ta dùng các kỹ thuật tạo bóng và loại bỏ các đường khuất, mặt khuất.

Với mô hình khung nối kết, hình dạng của đối tượng 3 chiều được biểu diễn bằng hai danh sách: danh sách các đỉnh và danh sách các cạnh nối các đỉnh đó. Danh sách các đỉnh cho biết thông tin hình học, còn danh sách các cạnh xác định thông tin về sự kết nối. Chúng ta hãy quan sát một vật thể ba chiều được biểu diễn bằng mô hình khung nối kết như sau:



Hình 5.16: Mô hình khung kết nối

Bảng danh sách các cạnh và đỉnh biểu diễn vật thể

Vertex List				
Vertex	**x**	**y**	**z**	
1	0	0	0	back side
2	0	1	0	
3	0	1	1	
4	0	0.5	1.5	
5	0	0	1	
6	1	0	0	front side
7	1	1	0	
8	1	1	1	
9	1	0.5	1.5	
10	1	0	1	

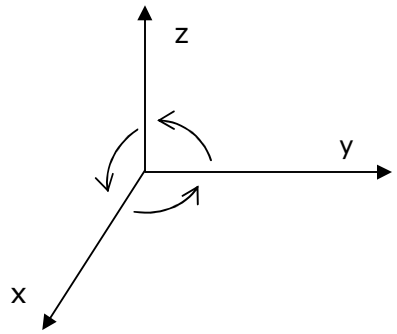
Edge List		
Edge	**Vertex1**	**Vertex2**
1	1	2
2	2	3
3	3	4
4	4	5
5	5	1
6	6	7
7	7	8
8	8	9
9	9	10
10	10	6
11	1	6
12	2	7
13	3	8
14	4	9
15	5	10
16	2	5
17	1	3

Người ta có thể vẽ các đối tương theo mô hình khung nối kết bằng cách sử dụng các phép chiếu song song hay phép chiếu phối cảnh sẽ được giới thiệu ở chương 6.

6.3. Các phép biến đổi 3 chiều

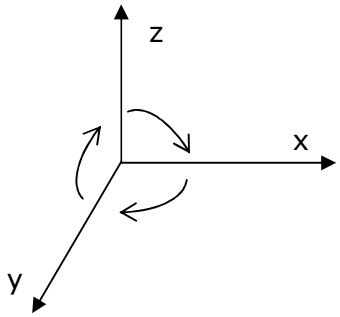
6.3.1. Hệ tọa độ bàn tay phải - bàn tay trái

• Hệ tọa độ theo qui ước bàn tay phải : để bàn tay phải sao cho ngón cái hướng theo trục z, khi nắm tay lại, các tay chuyển động theo hướng từ trục x đến trục y.



Hình 5.17: Hệ tọa độ bàn tay phải

• Hệ tọa tọa độ theo qui ước bàn tay trái : để bàn tay phải sao cho ngón cái hướng theo trục z, khi nắm tay lại, các ngón tay chuyển động theo hướng từ trục x đến trục y.



Hình 5.18: Hệ tọa độ bàn tay trái

• Hệ tọa độ thuần nhất: Mỗi điểm (x, y, z) trong không gian Đề-các được biểu diễn bởi một bộ bốn tọa độ trong không gian 4 chiều thu gọn (hx, hy, hz, h). Người ta thường chọn h = 1.

• Các phép biến đổi tuyến tính là tổ hợp của các phép biến đổi: tỉ lệ, quay, biến dạng và đối xứng. Các phép biến đổi tuyến tính có các tính chất sau:

- Gốc tọa độ là điểm bất động.
- Ảnh của đường thẳng là đường thẳng.
- Ảnh của các đường thẳng song song là các đường thẳng song song.
- Bảo toàn tỉ lệ khoảng cách.
- Tổ hợp các phép biến đổi có tính phân phối

6.3.2. Các phép biến đổi Affine cơ sở

• Phép tịnh tiến

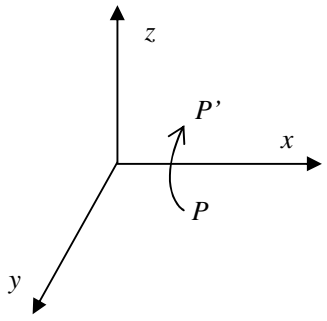
$$Tr(tr_x, tr_y, tr_z) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tr_x & tr_y & tr_z & 1 \end{pmatrix}$$

• Phép biến đổi tỉ lệ

$$S(S_x, S_y, S_z) = \begin{pmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Khi $S_x = S_y = S_z$ ta có phép biến đổi đồng dạng.

6.3.2.1 Phép quay quanh trục *x*



Hình 5.19 : Phép quay quanh trục x

Phép quay quanh trục là phép biến đổi $P(x, y, z) \rightarrow P'(x', y', z')$ qua phép quay góc α quanh trục *x*. Ta có :

$$\begin{cases} x' = x \\ y' = y\cos\alpha - z\cos\alpha \\ z' = y\sin\alpha + z\cos\alpha \end{cases}$$

Các tọa độ *y', z'* biến thiên tương tự phép quay góc α quanh góc tọa độ trong mặt phẳng *y*O*z* (*y* đóng vai trò *x*, *z* đóng vai trò *y*).

Do đó,

$$T_{x\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

6.3.2.2 Phép quay quanh trục *y*

Phép quay quanh trục y là phép biến đổi P(x,y,z) → P'(x',y',z') qua phép quay góc β quanh trục y. Ta có :

$$\begin{cases} y' = y \\ z' = z\cos\beta - x\sin\beta \\ x' = z\sin\beta + x\cos\beta \end{cases}$$

Do đó,

$$T_{y\beta} = \begin{pmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

6.3.2.3 Phép quay quanh trục z

Phép quay quanh trục z là phép biến đổi $P(x, y, z) \rightarrow P'(x', y', z')$ qua phép quay góc γ quanh trục *y*. Ta có :

$$\begin{cases} z' = z \\ x' = x\cos\gamma - y\sin\gamma \\ y' = x\sin\gamma + y\cos\gamma \end{cases}$$

Do đó,

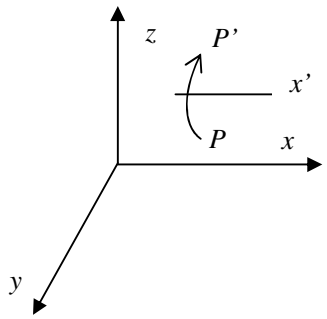
$$T_{z\gamma} = \begin{pmatrix} \cos\gamma & \gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

6.3.2.4 Phép quay quanh trục song song với trục tọa độ

Phép quay quanh trục song song với trục tọa độ hiệu quả với nhiều phép biến đổi, trong thực tế đối tượng thường quay quanh trục của nó. Ta xét trường hợp trục đối tượng song song với 1 trong các trục tọa độ. Để đơn giản ta phân tích chuyển động quay của đối tượng song song với trục cho trước theo các bước :

- ✓ **Bước 1 :** Tịnh tiến trục đối tượng trùng với trục tọa độ mà nó song song.
- ✓ **Bước 2 :** Quay đối tượng quanh trục của nó tương đương quay quanh trục tọa độ.
- ✓ **Bước 3 :** Tịnh tiến trả lại.

VD : Xét phép quay góc α quanh trục *x'* song song với *x* đi qua *(m, n, l)*



Hình 5.20 : Phép quay quanh trục x' song song với x

Bước 1 : Tịnh tiến *x'* trùng với *x*

$$T[-m,-n,-l] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -m & -n & -l & 1 \end{pmatrix}$$

Bước 2 : Quay quanh trục *x* với góc α

$$T_{x\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Bước 3 : Tịnh tiến trả lại

$$T^{-1}[-m,-n,-l] = T[m,n,l] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ m & n & l & 1 \end{pmatrix}$$

Do đó, ma trận biểu diễn phép quay góc α quanh trục *x'*song song *x* đi qua *(m,n,l)* là :

$$T = T[-m,-n,-l] \times T_{x\alpha} \times T[m,n,l]$$

Ví dụ: Tìm ảnh của hình chữ nhật *A*(1, 2, 1), *B*(3, 2, 1), *C*(3, 4, 3), *D*(1, 4, 3) sau phép quay góc α = 30^o quanh trục *x'* song song *x* đi qua (1, 1, 1).

Hướng dẫn giải:

Tính *:* $T = T[-1, -1, -1] \times T_{x(30)} \times T[1, 1, 1]$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$A(1, 2, 1) \rightarrow A' = (1, 2, 1, 1) \times T$

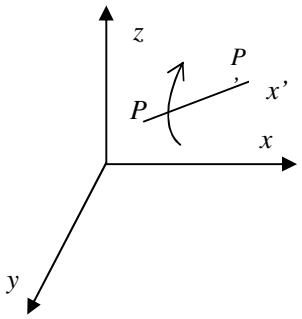
$B(3, 2, 1) \rightarrow B' = (3, 2, 1, 1) \times T$

$C(3, 4, 3) \rightarrow C' = (3, 4, 3, 1) \times T$

$D(1, 4, 3) \rightarrow D' = (1, 4, 3, 1) \times T$

6.3.2.5 Phép quay quanh trục bất kỳ

Xét phép quay góc α quanh trục bất kỳ, ta thực hiện qua các bước sau :



Hình 5.21: Phép quay quanh trục bất kỳ

Bước 1 : Tịnh tiến trùng gốc tọa độ

$$T[-P.x,-P.y,-P.z] = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -P.x & -P.y & -P.z & 1 \end{pmatrix}$$

Bước 2 : Quay quanh trục z góc α sao cho P,P' thuộc (xOz)

$$T_{z\gamma} = \begin{pmatrix} \cos\gamma & \gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Bước 3 : Quay quanh trục y góc β sao cho P, P' thuộc Ox

$$T_{y\beta} = \begin{pmatrix} \cos\beta & 0 & -\sin\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\beta & 0 & \cos\beta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Bước 4 : Quay quanh trục *x* góc α:

$$T_{x\alpha} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha & \sin\alpha & 0 \\ 0 & -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Bước 5 : Ngược bước 3

Bước 6: Ngược bước 2

Bước 7 : Ngược bước 1

Cách xác định chiều dương trong các phép quay

Định nghĩa về chiều quay được dùng chung cho cả hệ tọa độ theo qui ước bàn tay phải và bàn tay trái. Cụ thể chiều dương được định nghĩa như sau :

- Quay quanh truc *x* : từ trục dương *y* đến trục dương *x*
- Quay quanh trục *y* : từ trục dương *z* đến trục dương *x*
- Quay quanh trục *z* : từ trục dương *x* đến trục dương *y*

Ngoài các phép biến đổi trên, ta xét thêm một số phép biến đổi affine khác sau đây:

• Phép đối xứng qua mặt phẳng tọa độ

$$(yOz): \qquad Mr(x) = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$(zOx): \qquad Mr(y) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$(xOy): \qquad Mr(z) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

• Phép đối xứng qua trục *x*, *y* và *z*

$$M(x) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M(y) = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$M(z) = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

• Phép biến dạng

$$S(h) = \begin{pmatrix} 1 & h_{yx} & h_{zx} & 0 \\ h_{xy} & 1 & h_{zy} & 0 \\ h_{xz} & h_{yz} & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Bài tập chương 6

1. Tìm vị trí mới của hình chữ nhật *ABCD* với *A*(1, 2, 1), *B*(3, 2, 1), *C*(3, 4, 3), *D*(1, 4, 3) sau phép quay góc α=45^o quanh góc tọa độ.
2. Tìm vị trí mới của hình chữ nhật *ABCD* với *A*(1, 2, 1), *B*(3, 2, 1), *C*(3, 4, 3), *D*(1, 4, 3) sau phép quay góc α=30^o quanh điểm *M*(1, 1, 1).
3. Tìm vị trí mới của hình chữ nhật *A*(1, 2, 1), *B*(3, 2, 1), *C*(3, 4, 3), *D*(1, 4, 3) sau phép quay góc α=45^o quanh trục *x'* song song *x* đi qua (1, 1, 1).

PHỤ LỤC

THƯ VIỆN ĐỒ HỌA OpenGL

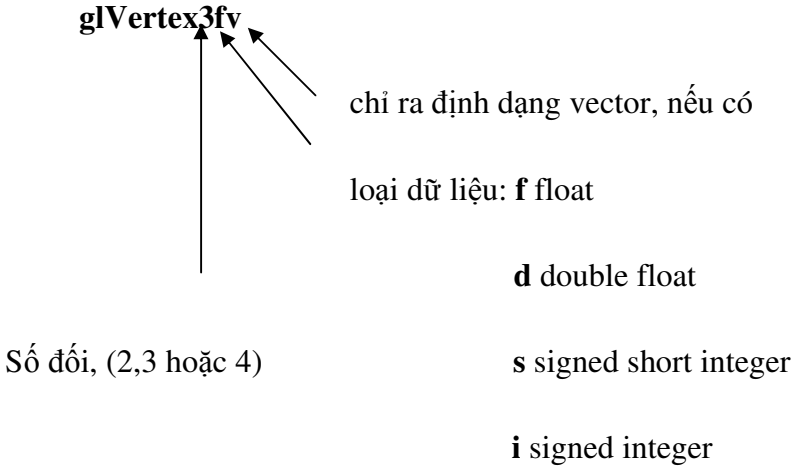
OpenGL là gì?

OpenGL (Open Graphics Library) là phần mềm giao diện với các phần cứng đồ hoạ. OpenGL được phát triển bởi Silicon Graphic Inc. OpenGL cũng là một giao diện lập trình ứng dụng (Application Program Interface – API). Nó bao gồm khoảng 150 câu lệnh hỗ trợ nhiều ngôn ngữ như C, C++, Java, C#...Cho phép người lập trình sử dụng để tạo ra ứng dụng tương tác đồ hoạ 3D.

OpenGL được thiết kế không phụ thuộc nền tảng phần cứng cũng như hệ điều hành máy tính. Như một chương trình trung gian giữa người dùng và phần cứng máy tính. Với OpenGL chúng ta sẽ tạo ra các mô hình phức tạp từ những đối tượng hình học cơ bản. Đó là các điểm (Point), đoạn thẳng (Line) và đa giác (Polygon).

Cú pháp của OpenGL

Các câu lệnh của OpenGL đều sử dụng tiền tố gl và các từ tiếp theo được bắt đầu bằng kí tự hoa, ví dụ glClearColor(). Tương tự như vậy, các hằng được định nghĩa bằng tiền tố GL_ tiếp theo là các từ viết hoa được ngăn cách bởi kí tự gạch dưới, ví dụ GL_COLOR_BUFFER_BIT.



Loại dữ liệu khác trong lệnh OpenGL :

- **b** character
- **ub** unsigned character
- **us** unsigned short integer
- **ui** unsinged integer.

Dữ liệu vô hướng và định dạng vector.

Câu lệnh OpenGL cho ta thấy được ý nghĩa chức năng của hàm.Tham số và loại tham số xuất hiện tuỳ thuộc các hàm khác nhau.

Đôi khi trong câu lệnh có thêm dấu * để chỉ rằng cú pháp này có thể có nhiều lệnh. Ví dụ, glColor*() có giá trị cho các lệnh khác nhau để bạn thiết lập màu hiện hành. Hoặc **glClear*()** có các lệnh sau: **glClearColor(), glClearDepth(), glClearAccum(), glClearStencil()**.

OpenGL là một máy trạng thái

OpenGL là một máy trạng thái. Chúng ta có thể đặt nó các trạng thái khác nhau. Chúng giữ nguyên tác dụng cho đến khi ta thay đổi trạng thái khác. Chẳng hạn đặt màu hiện hành là một biến trạng thái. Chúng ta có thể đặt màu hiện tại bởi màu trắng, màu đỏ hoặc màu nào khác, và sau đó mỗi đối tượng được vẽ bởi màu đó cho tới khi bạn đặt màu hiện tại bằng màu khác. Màu hiện tại chỉ là một trong nhiều biến trạng thái mà OpenGL lưu giữ. Còn nhiều trạng thái khác như điểm nhìn hiện hành, vị trí và đặc tính ánh sáng, thuộc tính chất liệu, …

Biến trạng thái là nơi lưu giữ các trạng thái. Mỗi biến trạng thái hoặc chế độ có một giá trị mặc định ban đầu. Ta có thể xem giá trị của chúng thông qua 6 hàm sau:

- glGetBooleanv()
- glGetDoublev()
- glGetFloatv()
- glGetIntergerv()

- glGetPointerv()
- glIsEnabled()

Một vài biến trạng thái có nhiều hơn chỉ định lệnh yêu cầu (chẳng hạn **glGetLight***(), **glGetError**(), hoặc **glGetPolygonStipple**()). Hơn nữa ta có thể lưu và lấy ra các giá trị của tập trạng thái biến trên thuộc tính stack với lệnh **glPushAttrib()** hoặc **glPushClientAttrib()** và **glPopAttrib()** hoặc **glPopClientAttrib().**

Các thư viện liên quan

Mặc dù OpenGL là công cụ mạnh song các đối tượng vẽ đều là những đối tượng hình học cơ bản. Để đơn giản một số thủ tục, chúng ta được cung cấp một số thư viện để có thể điều khiển việc vẽ đối tượng ở mức cao hơn.

♦ **OpenGL Utility Library (GLU):** Bao gồm một số thủ tục thiết lập ma trận xác định hướng nhìn, ma trận các phép chiếu, và biểu diễn các mặt trong không gian 3 chiều.

♦ **OpenGL Utility Toolkit (GLUT):** bao gồm các thủ tục nhằm đơn giản hoá việc tạo các đối tượng hình học. Đặc biệt hình trong không gian 3 chiều (solid hình đặc, wire hình khung).

♦ Khi lập trình OpenGL trong C# ta sử dụng một số thư viện sau: csgl.dll, csgl.native.dll, CsGL.Basecode.

Hiển thị các đối tượng hình học cơ bản: điểm, đoạn thẳng, đa giác

Để tạo một đối tượng hình học từ các đỉnh, ta đặt các đỉnh giữa hai hàm glBegin(*param*) và glEnd(). Tham số *param* đưa vào cho hàm glBegin() sẽ quyết định đối tượng OpenGL vẽ ra từ các đỉnh khai báo bên trong.

Ví dụ:

```
glBegin(GL_POLYGON);
    glVertex2f(0.0, 0.0);
```

glVertex2f(0.0, 1.0);

glVertex2f(0.5, 1.0);

glVertex2f(1.0, 0.5);

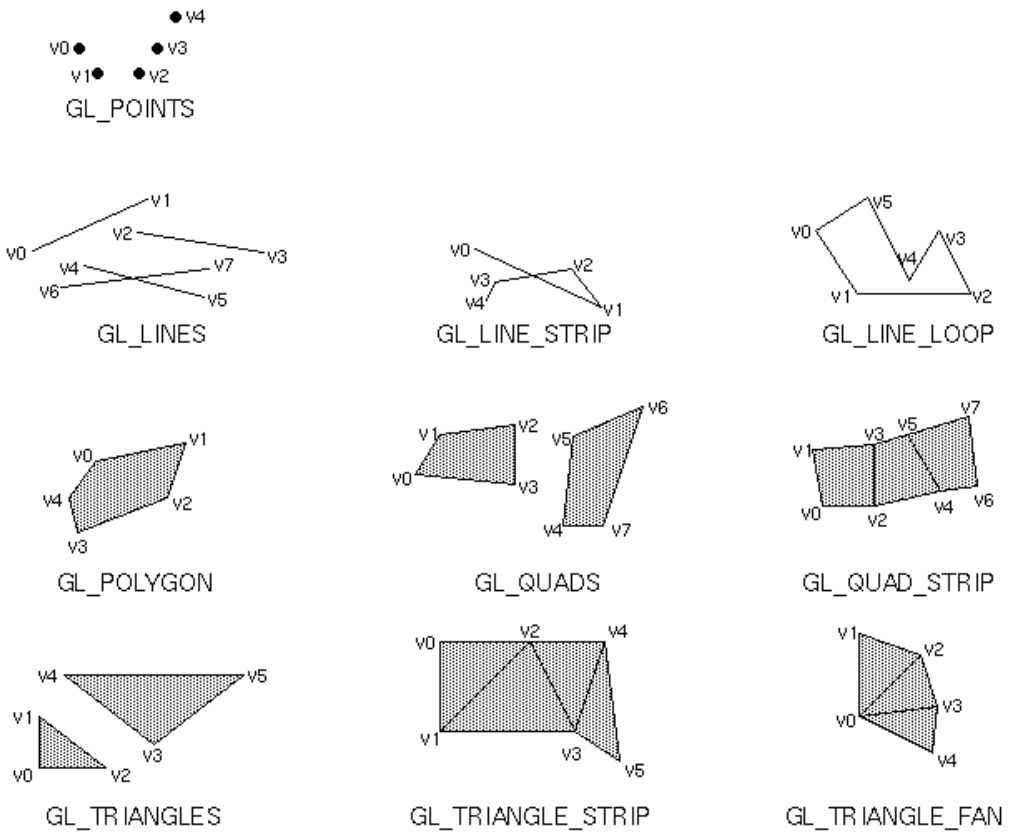
glVertex2f(0.5, 0.0);

glEnd();

Hàm glBegin(*Glenum mode*). Biến *mode* chỉ ra đối tượng được vẽ, nhận một trong các giá trị sau:

GIÁ TRỊ	Ý NGHĨA
GL_POINTS	Vẽ các điểm
GL_LINES	Vẽ các đoạn thẳng
GL_POLYGON	Vẽ đa giác lồi
GL_TRIANGLES	Vẽ tam giác
GL_QUADS	Vẽ tứ giác
GL_LINE_STRIP	Vẽ đường gấp khúc không khép kín
GL_LINE_LOOP	Vẽ đường gấp khúc khép kín
GL_TRIANGLE_STRIP	Một dải các tam giác liên kết với nhau
GL_TRIANGLE_FAN	Một dải các tam giác liên kết theo hình quạt
GL_QUAD_STRIP	Một dải các tứ giác liên kết với nhau

Danh sách các hình thể hiện kết quả tương ứng của biến mode



Hình phụ lục 1: Các đối tượng hình học cơ bản

Quy luật hiển thị rõ ràng trên hình vẽ, riêng với GL_QUAD_STRIP được vẽ với quy luật nối 4 điểm có vị trí $2n$, $2n + 1$, $2n + 3$, $2n + 2$ với $2n$ là điểm khởi đầu của hình tứ giác. Để chỉ định 1 đỉnh ta dùng lệnh sau: *glVertex*{2,3,4}*{sifd}[v](toạ độ)***.** Trong đó:

- {2,3,4} chỉ định số chiều của không gian.
- [v] nếu toạ độ điểm được truyền từ một mảng cho trước.
- {sifd} chỉ định kiểu dữ liệu của toạ độ, ý nghĩa được chỉ định trong bảng sau:

Kí hiệu	**Kiểu dữ liệu**	**Tên kiểu của OpenGL**
s	16 bit - integer	GLshort
i	32 bit - integer	GLint
f	32 bit - float	GLfloat

d	64 bit - float	GLdouble

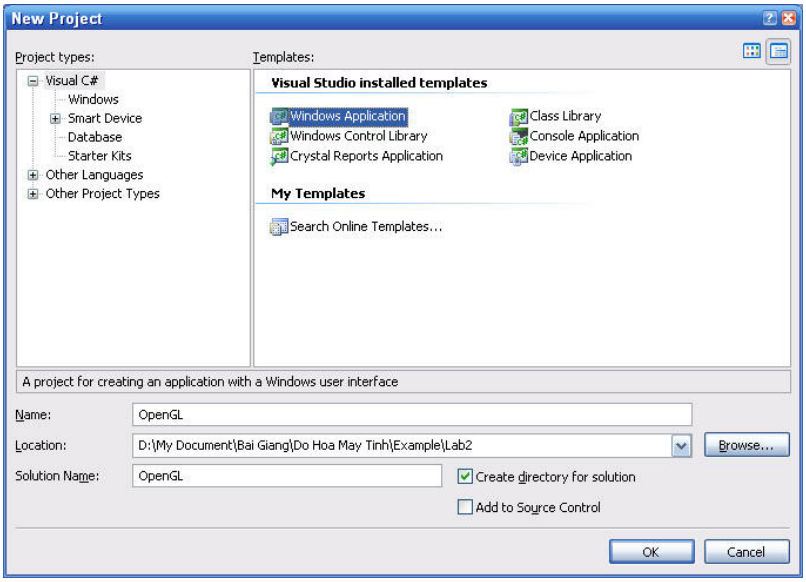
OpenGL chỉ cho phép một số lệnh nằm bên trong glBegin() và glEnd()

glVertex*()	Khai báo vertex
glColor*()	Thiết lập màu
glIndex*()	Thiết lập chỉ mục màu
glNormal*()	Thiết lập tọa độ vector chỉ phương
glEvalCoord*()	Sinh tọa độ
glCallList(), glCallLists()	Thực thi Display List
glTexCoord*()	Thiết lập tọa độ texture
glEdgeFlag*()	Điều khiển việc vẽ cạnh
glMaterial*()	Thiết lập thuộc tính chất liệu

Mọi hàm OpenGL ngoài các hàm trên đều không được nằm giữa glBegin() và glEnd(). Tuy nhiên ta vẫn có thể dùng các cấu trúc điều khiển khác (ví dụ 1 vòng lặp for chẳng hạn).

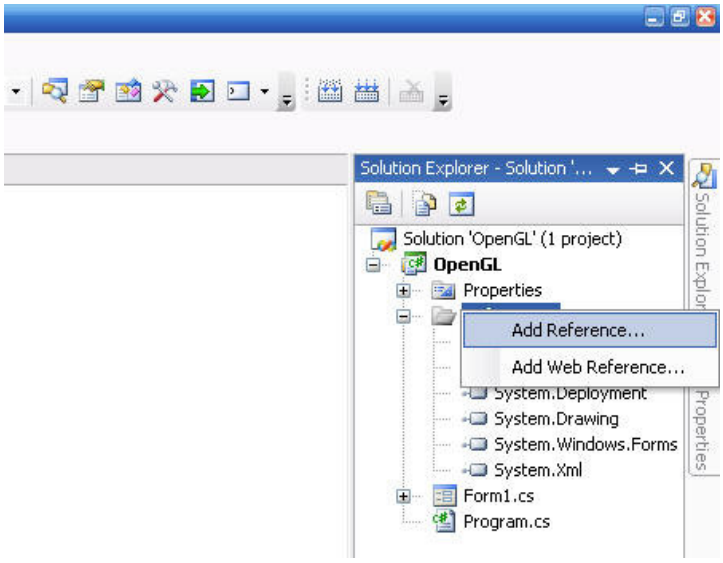
Bắt đầu làm quen OpenGL bằng ngôn ngữ C#

- Mở chương trình Visual Studio .NET (2005) và tạo mới ứng dụng C# trong Windows Application.



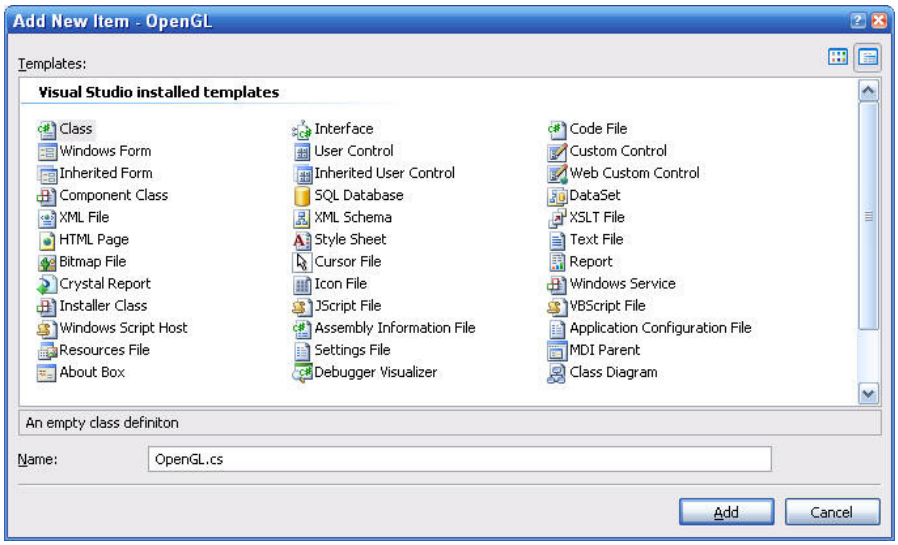
Hình phụ lục 2:Tạo project đồ họa mới

- Nhấp phải chuột vào References, chọn Add References ... chọn thẻ **Browse** để thêm thư viện CsGL. (File **csgl.dll**)



Hình phụ lục 3: Thêm thư viện OpenGL vào project

- Nhấp phải chuột vào project và chọn Add\Class... (Ví dụ, đặt tên lớp là OpenGL.cs)



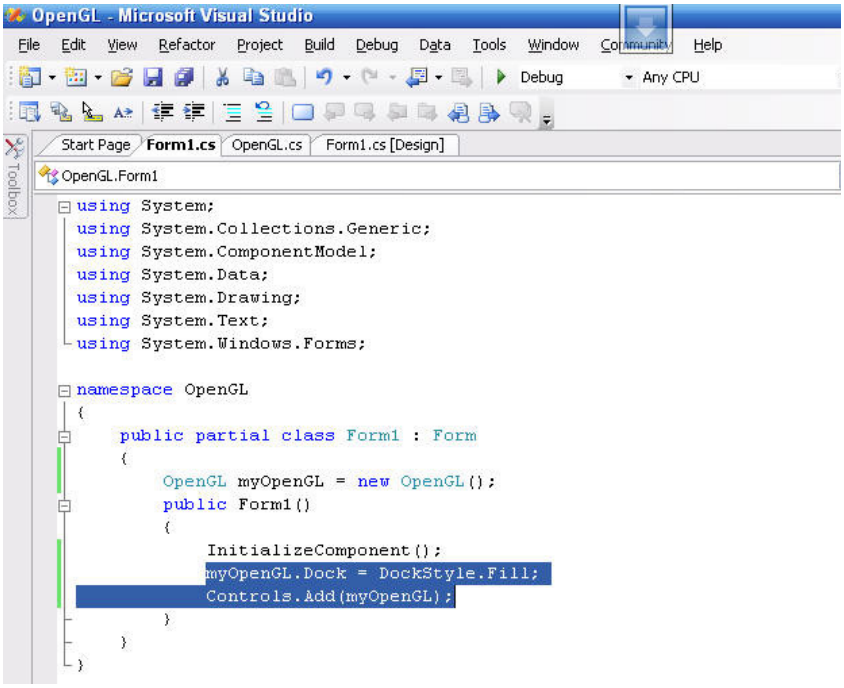
Hình phụ lục 4: Thêm lớp mới vào project

- Khai báo lớp OpenGL:
  - Khai báo thư viện CsGL.OpenGL;
  - Thừa kế lớp OpenGLControl.
  - Khai báo quá tải phương thức glDraw(), InitGLContext() và OnSizeChanged(EventArgs e).

```
using CsGL.OpenGL; //Khai báo thư viện OpenGL

namespace OpenGL
{
    class OpenGL:OpenGLControl
    {
        public override void glDraw()//Phương thức vẽ đồ họa
        {
            GL.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT); // Xóa sạch màn hình và vùng đệm
            GL.glViewport(0, 0, Size.Width, Size.Height); //Khai báo khung nhìn đồ họa để vẽ
            GL.glBegin(GL.GL_POINTS); //Khởi tạo chế độ đồ họa vẽ các pixel

            //Thuật hiện các thuật toán vẽ đồ họa ở đây
            GL.glVertex2i(-100, 100); //Vẽ một pixel tại tọa độ (x=-100, y=100)

            GL.glEnd();//Đóng chế độ đồ họa vẽ các pixel
            GL.glFlush();//Dọn sạch rác trong vùng nhớ
        }
        protected override void InitGLContext()//Phương thức khởi tạo các thông số đồ họa
        {
            GL.glClearColor(0.0f, 0.0f, 0.0f, 0.0f); //Khởi tạo màu nền (R,G,B,hệ số alpha), f chỉ số thực
            GL.glColor3f(1.0f, 0.0f, 0.0f); //Khởi tạo màu vẽ đối tượng (R,G,B)
            GL.glPointSize(1.0f);//Khởi tạo kích thước vẽ đối tượng, chuẩn là 1.0
        }
        protected override void OnSizeChanged(EventArgs e) //Phương thức bắt sự kiện khi kích thước giao diện thay đổi
        {
            base.OnSizeChanged(e);
            GL.glMatrixMode(GL.GL_PROJECTION);
            GL.glLoadIdentity();
            GL.glOrtho(-Size.Width / 2, Size.Width / 2, -Size.Height / 2, Size.Height / 2, -Size.Height / 2, Size.Height / 2);
        }
    }
}
```

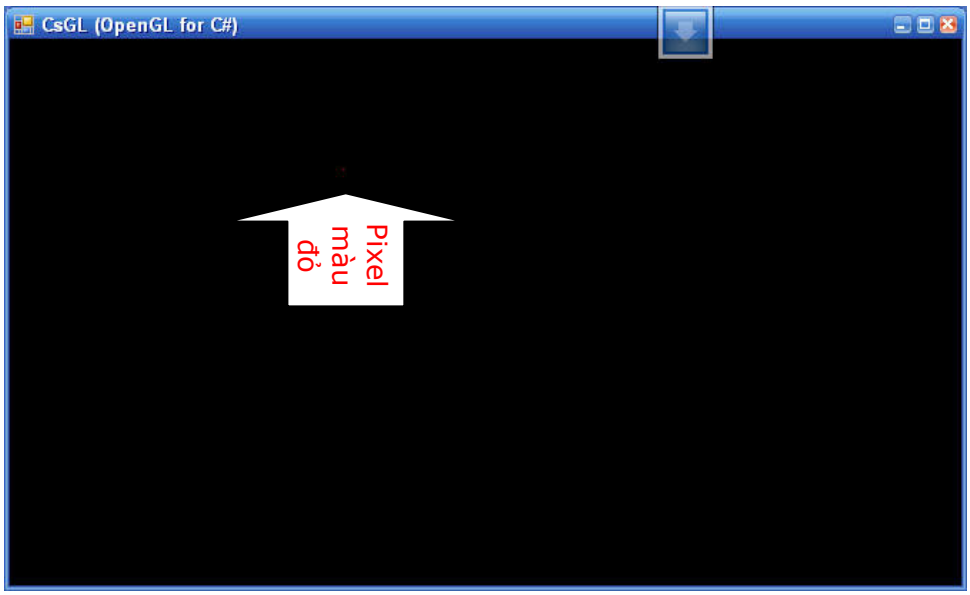
Hình phụ lục 5: Viết code khởi tạo chế độ đồ họa OpenGL

- Lớp OpenGL khai báo trên như một User Control. Bây giờ ta thêm User Control trên vào Form1 như sau: (hoặc có thể kéo User Control qua Form)



Hình phụ lục 6:Thêm control OpenGL vào form

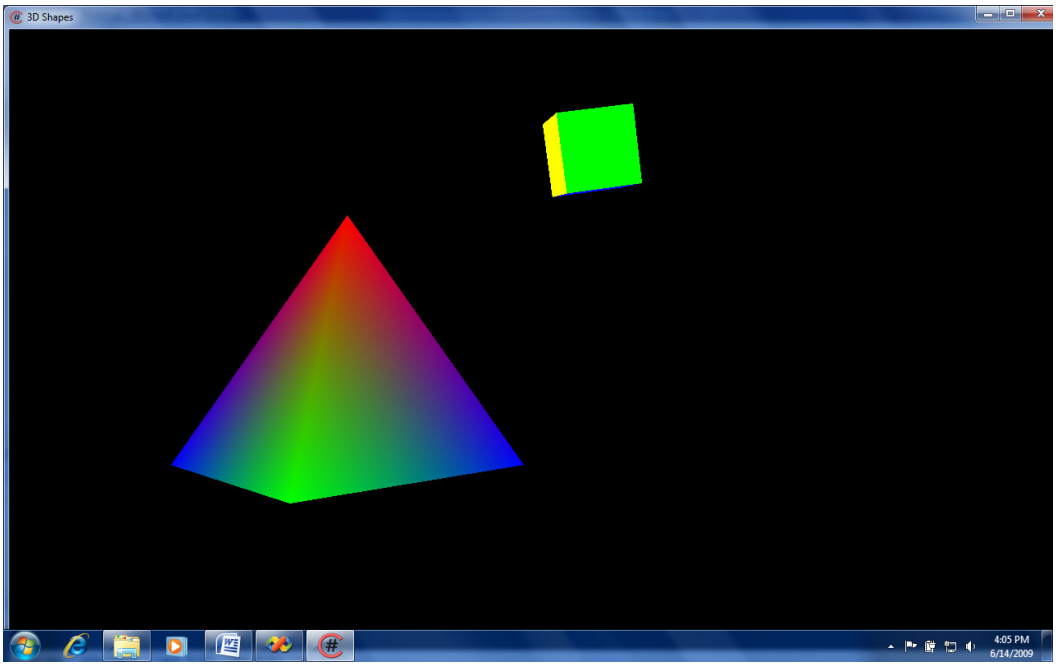
- Chạy chương trình thử (nhớ chép file ***csgl.native.dll*** vào thư mục ***Debug***), nếu thấy xuất hiện một pixel màu đỏ trên nền màu đen là thành công.



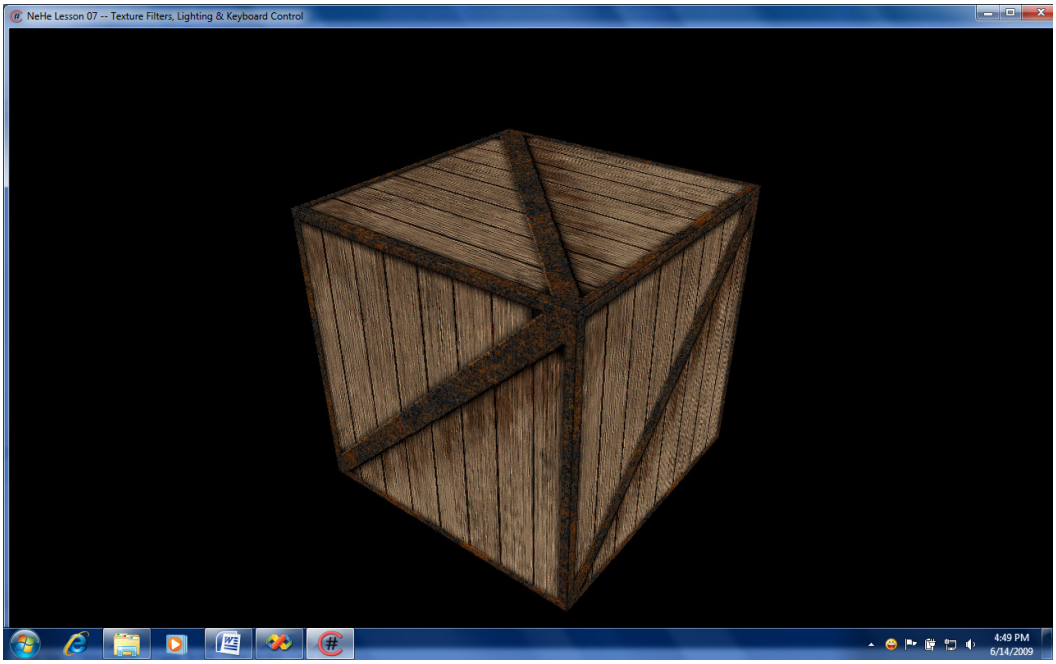
Hình phụ lục 7: Kết quả khởi tạo chế độ đồ họa OpenGL

Vẽ đối tượng 3D trong OpenGL

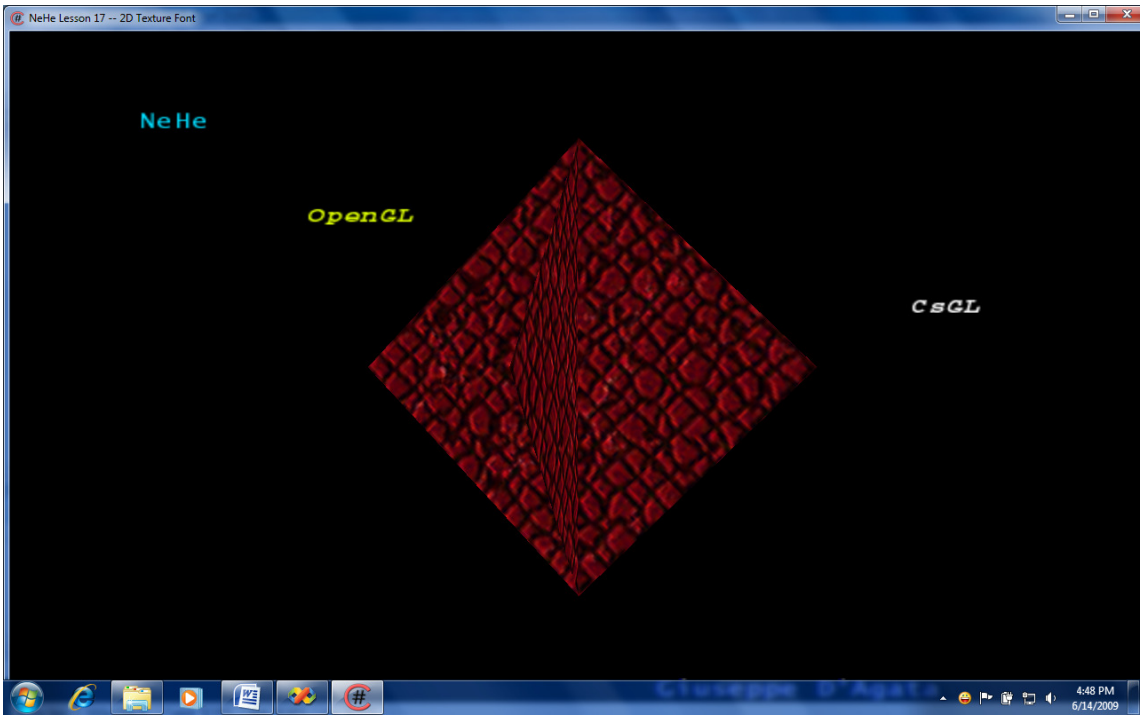
OpenGL cho phép vẽ các đối tượng 3D dễ dàng và tạo các hiệu ứng màu, ánh sáng, biến đổi trong không gian 3D rất chính xác. Sau đây là một số hình ảnh 3D được lập trình từ thư viện OpenGL.



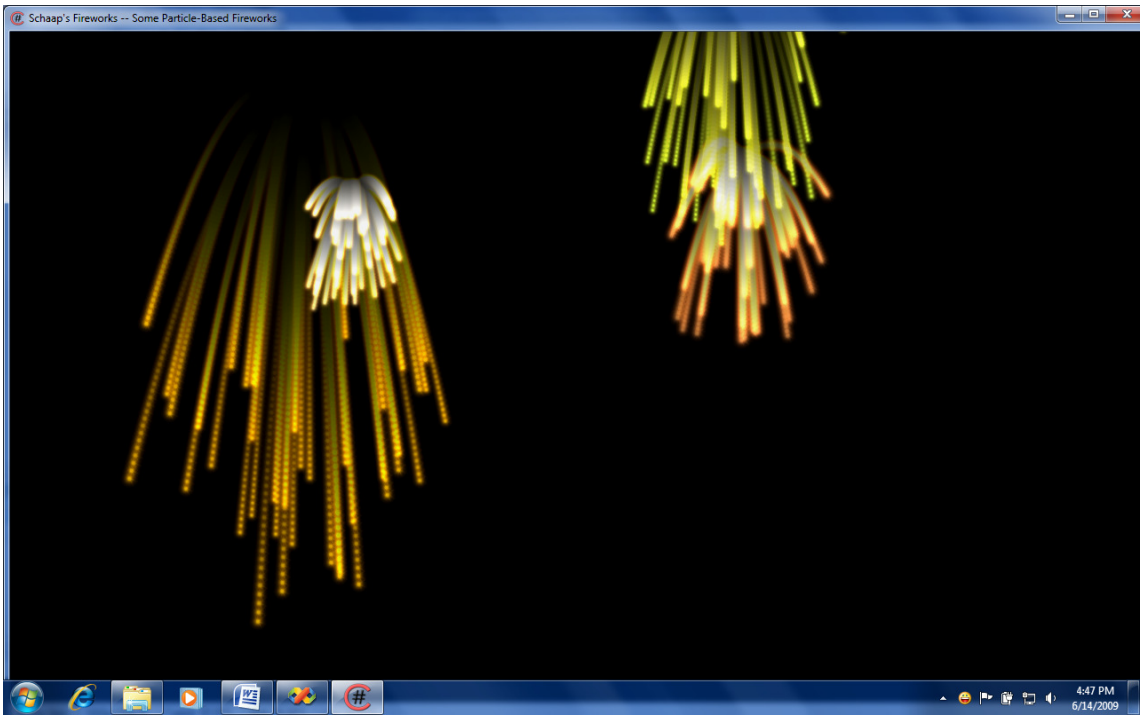
Hình phụ lục 8: Các đối tượng hình học 3D cơ bản



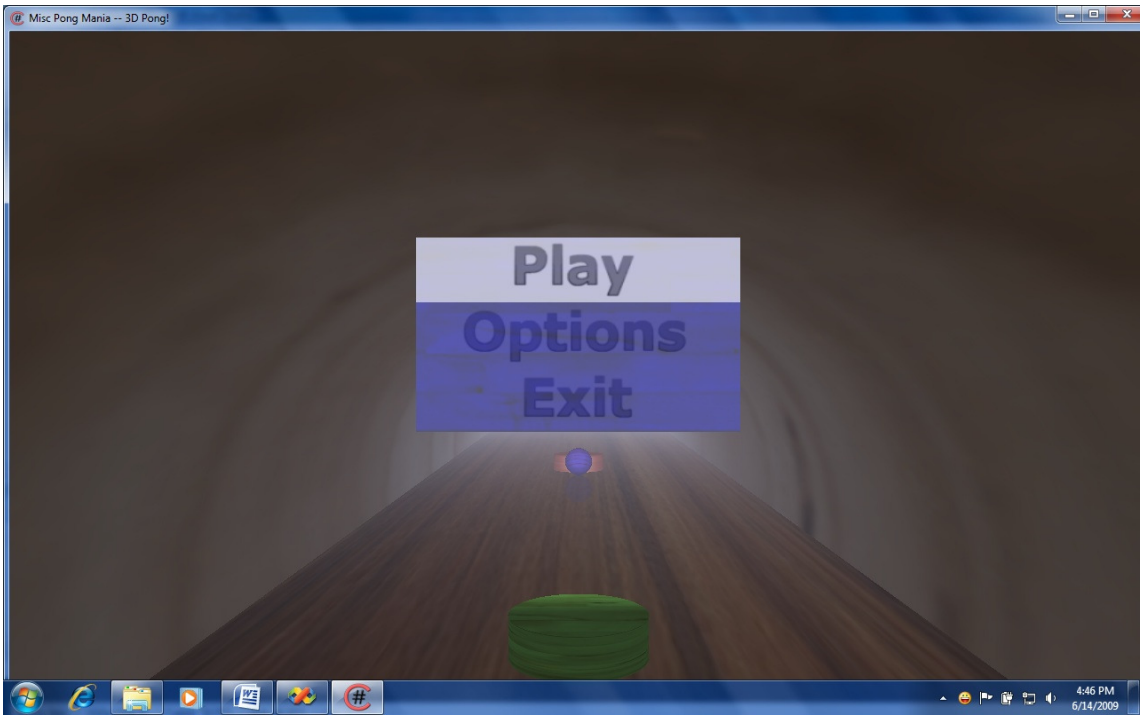
Hình phụ lục 9:Hình khối lập phương hiển thị bằng texture



Hình phụ lục 10: Hình 3D và phông chữ trong đồ họa OpenGL



Hình phụ lục 11: Pháo hoa được biểu diễn bằng OpenGL

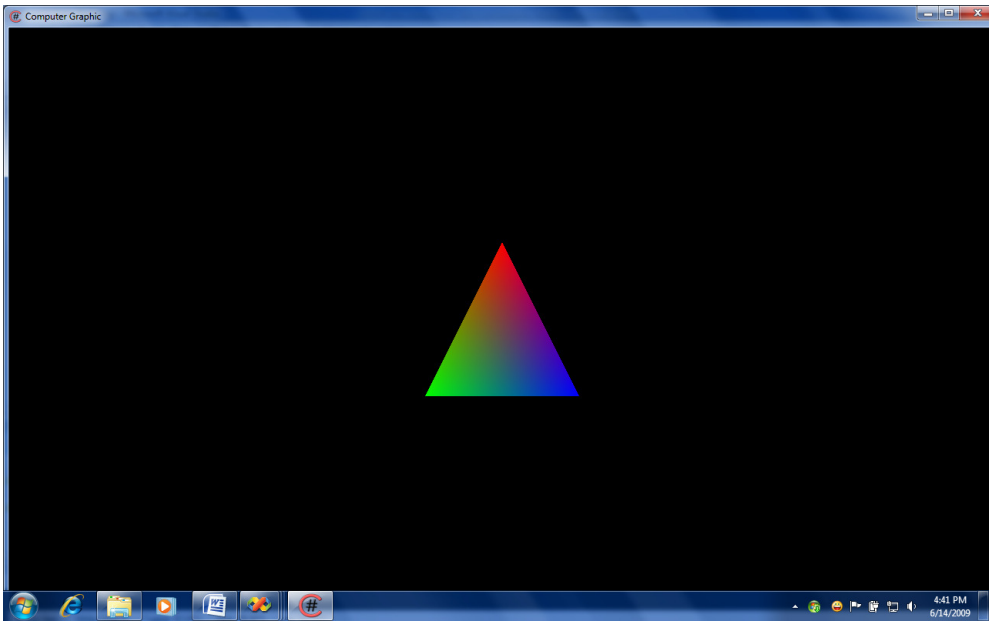


Hình phụ lục 12: Game đơn giản viết bằng OpenGL

Đoạn chương sau minh họa vẽ đối tượng hình chóp tam giác. Ta thực hiện vẽ 4 mặt của hình chóp, mỗi mặt của nó là một tam giác gồm 3 đỉnh, có màu được pha trộn từ màu của các đỉnh.

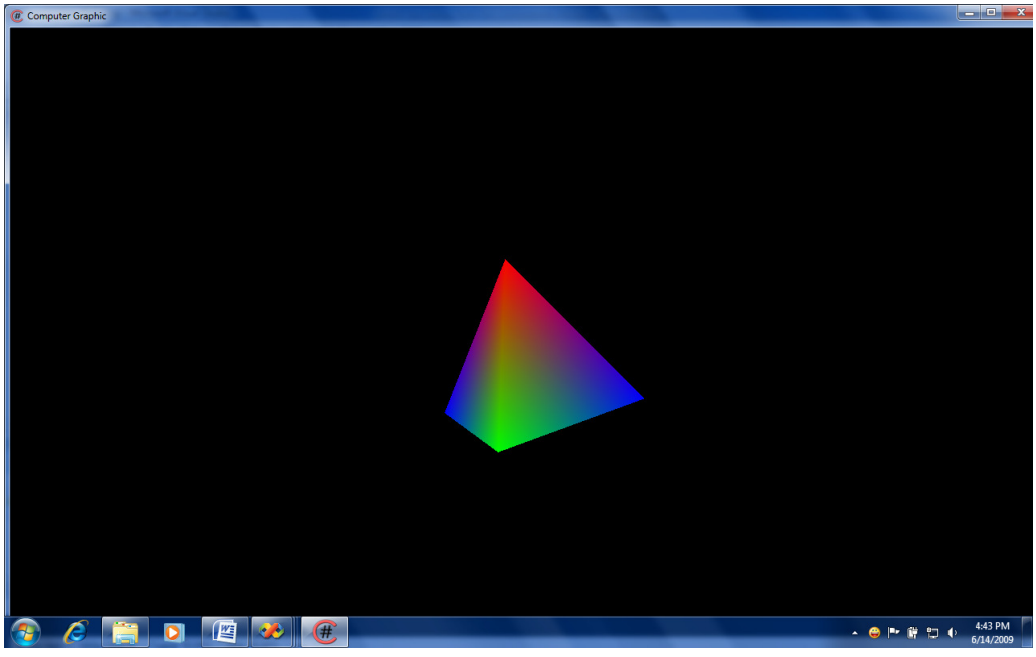
```
glBegin(GL_TRIANGLES);
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 100.0f, 0.0f);
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3f(-100.0f, -100.0f, 100.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(100.0f, -100.0f, 100.0f);

glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 100.0f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(100.0f, -100.0f, 100.0f);
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3f(100.0f, -100.0f, -100.0f);

glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 100.0f, 0.0f);
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3f(100.0f, -100.0f, -100.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(-100.0f, -100.0f, -100.0f);

glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 100.0f, 0.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(-100.0f, -100.0f, -100.0f);
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3f(-100.0f, -100.0f, 100.0f);
glEnd();
```

Vì 3 mặt sau của đối tượng trên bị khuất nên khi hiển thị chỉ thấy một mặt của hình chóp tam giác.



Hình phụ lục 13: Hình chóp tam giác ban đầu trong OpenGL

Để thấy được các mặt còn lại ta phải quay hình chóp tam giác quanh các trục một góc nào đó. Chẳng hạn, ta quay hình chóp một góc 30^o quanh các trục x, y, và z, ta được kết quả sau: *glRotatef(30, 1, 1, 1);*



Hình phụ lục 14: Hình chóp tam giác sau khi quay trong OpenGL

TÀI LIỆU THAM KHẢO

[1] Donald Hearn, M. Pauline Baker**;** *Computer Graphics*; Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986

[2]. F.S. Hill Jr. *Computer Graphics*, Macmillan Publishing Company, New York, 1990.

[3]. J. D. Foley, A. Van Dam, S. K. Feiner, J. F. Hughes, *Computer graphics: principles and practice*, Addison-Wesley 1991.

[4]. Phạm Tiến Sơn, *Đồ họa máy tính I*, Lưu hành nội bộ, Đại học Đà Lạt, 2005.

Hết

Lời nói đầu

Đồ họa máy tính được ra đời bởi sự kết hợp của 2 lĩnh vực thông tin và truyền hình. Đầu tiên kỹ thuật đồ họa được phát triển bởi các nhóm kỹ sư sử dụng máy tính lớn. Trong giai đoạn đầu của sự phát triển người ta phải tốn nhiều tiền cho việc trang bị các thiết bị phần cứng. Ngày nay, nhờ vào sự tiến bộ của vi xử lý, giá thành của máy tính càng lúc càng phù hợp với túi tiền của người sử dụng trong khi các kỹ thuật ứng dụng đồ họa của nó ngày càng cao hơn nên có nhiều người quan tâm nghiên cứu đến lĩnh vực này. Chúng ta có thể vẽ ra những hình ảnh không chỉ là ảnh tĩnh mà còn có thể biến đổi thành những hình ảnh sinh động qua các phép quay, tịnh tiến... Do vậy, đồ họa máy tính trở thành một lĩnh vực lý thú và có nhiều ứng dụng trong thực tế.

Tuy nhiên, việc dạy và học kỹ thuật đồ họa thì không là đơn giản do chủ đề này có nhiều phức tạp. Kỹ thuật đồ họa liên quan đến tin học và toán học bởi vì hầu hết các giải thuật vẽ, tô cùng các phép biến hình đều được xây dựng dựa trên nền tảng của hình học không gian hai chiều và ba chiều.

Hiện nay, Kỹ thuật đồ họa là một môn học được giảng dạy cho sinh viên chuyên ngành Tin học với 45 tiết lý thuyết và 15 tiết thực tập. Nội dung của giáo trình kỹ thuật đồ họa này tập trung vào 2 vấn đề chính như sau :

- Trình bày các thuật toán vẽ và tô các đường cơ bản như đường thẳng, đa giác, đường tròn, ellipse và các đường conic. Các thuật toán này giúp cho sinh viên có thể tự mình thiết kế để vẽ và tô một hình nào đó (chương 1 và 2).

- Nội dung thứ hai đề cập đến đồ họa hai chiều và đồ họa ba chiều bao gồm các phép biến đổi Affine, windowing và clipping, quan sát ảnh ba chiều qua các phép chiếu, khử các mặt khuất và đường khuất, thiết kế đường cong và mặt cong (từ chương 3 đến chương 7).

Giáo trình kỹ thuật đồ họa này được sửa đổi và cập nhật dựa trên kinh nghiệm giảng dạy đã qua và được xây dựng dựa trên tài liệu tham khảo chính là :

Donald Hearn, M. Pauline Baker; Computer Graphics; Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986.

Sau cùng, chúng tôi hy vọng rằng giáo trình này sẽ đóng góp tích cực trong việc cải tiến sự hiểu biết của sinh viên về lĩnh vực đồ họa và mong nhận được sự góp ý của các đồng nghiệp và sinh viên để công việc biên soạn ngày càng được tốt hơn.

Mục lục

PHẦN TỔNG QUAN

1. Mục đích yêu cầu

Sau khi học xong môn này, sinh viên cần đạt được các yêu cầu sau:

- Hiểu thế nào là đồ họa trên máy tính.
- Thiết kế và cài đặt được các thuật toán vẽ các đường cơ bản như đường thẳng, đường tròn,...
- Thiết kế và cài đặt được các thuật toán tô một hình.
- Sử dụng được các phép biến hình trong không gian 2 chiều, 3 chiều để làm thay đổi một hình ảnh đã có sẵn.
- Có thể tạo một cửa sổ để cắt - dán một hình.
- Hiểu khái niệm về các tiếp cận để mô phỏng được một hình ảnh trong không gian 3 chiều trên máy tính.

2. Đối tượng sử dụng

Môn kỹ thuật đồ họa được giảng dạy cho sinh viên năm thứ tư của các khoa sau:

- Chuyên ngành công nghệ thông tin.
- Chuyên ngành điện tử (viễn thông, tự động hóa,...)
- Chuyên ngành sư phạm (Toán tin, Lý tin)

3. Nội dung cốt lõi

Giáo trình Kỹ thuật đồ họa bao gồm 7 chương.

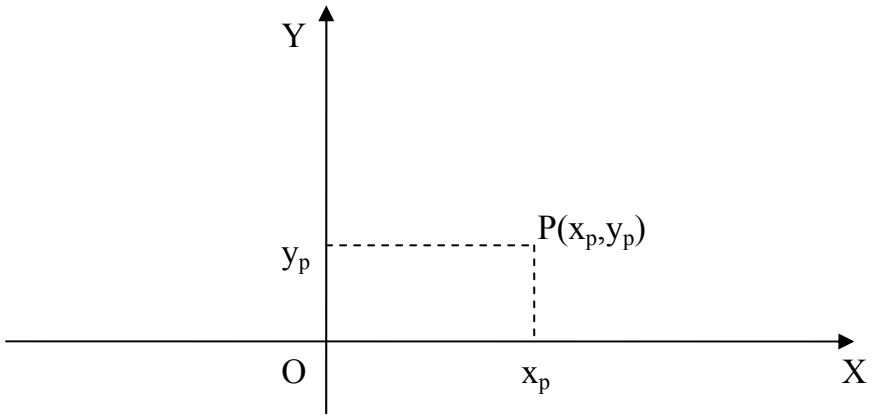
- Chương 1: Giới thiệu thuật toán vẽ và tô các đường cơ bản
- Chương 2: Các thuật toán tô màu
- Chương 3: Phép biến đổi trong đồ họa 2 chiều
- Chương 4: Tạo cửa sổ và cắt hình
- Chương 5: Đồ họa 3 chiều
- Chương 6: Quan sát ảnh 3 chiều
- Chương 7: Khử các mặt khuất và đường khuất

4. Kiến thức tiên quyết

- Kiến thức về hình học không gian và hình giải tích
- Kiến thức lập trình căn bản, lập trình đồ họa
- Kiến thức về cấu trúc dữ liệu, lập trình đệ qui

5. Danh mục tài liệu tham khảo

- Donald Hearn, M. Pauline Baker; Computer Graphics; Prentice-Hall, Inc., Englewood Cliffs, New Jersey , 1986.
- F.S.Hill; Computer graphics ; 1990
- Vũ Mạnh Tường, Dương Anh Đức, Trần Đan Thư, Lý Quốc Ngọc. Giáo trình Nhập môn đồ họa & xử lý ảnh.1995.
- VERA B.ANAND, người dịch TS Nguyễn Hữu Lộc. Đồ họa máy tính và Mô hình hóa hình học. Nhà xuất bản Thành Phố Hồ Chí Minh - 2000.
- Foley, Van Darn, Feiner, Hughes, Phillips. Introduction à L'Infographie. 1995.
- Lê Tấn Hùng, Huỳnh Quyết Thắng. Kỹ thuật đồ họa. Nhà xuất bản khoa học và kỹ thuật, Hà nội - 2000.



Hình 1.1 : Hệ tọa độ thực.

. Ox : gọi là trục hoành.

. Oy : gọi là trục tung.

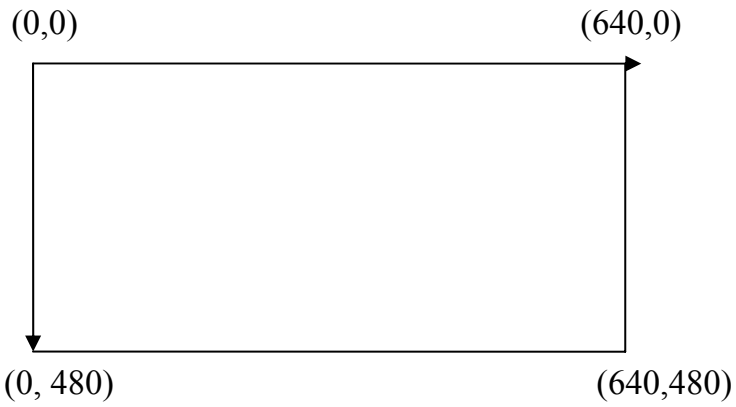
. x_p : hoành độ điểm P.

. y_p : tung độ điểm P.

b. Hệ tọa độ thiết bị

Hệ tọa độ thiết bị (device coordinates) được dùng cho một thiết bị xuất cụ thể nào đó, ví dụ như máy in, màn hình,..

Trong hệ tọa độ thiết bị thì các điểm cũng được mô tả bởi cặp tọa độ (x,y). Tuy nhiên, khác với hệ tọa độ thực là x, y $\in$ N. Điều này có nghĩa là các điểm trong hệ tọa độ thực được định nghĩa liên tục, còn các điểm trong hệ tọa độ thiết bị là rời rạc. Ngoài ra, các tọa độ x, y của hệ tọa độ thiết bị chỉ biểu diễn được trong một giới hạn nào đó của N.
Ví dụ : Độ phân giải của màn hình trong chế độ đồ họa là 640x480. Khi đó, x$\in$(0,640) và y$\in$(0,480) (xem hình 1.2).



Hình 1.2 : Hệ tọa độ trên màn hình.

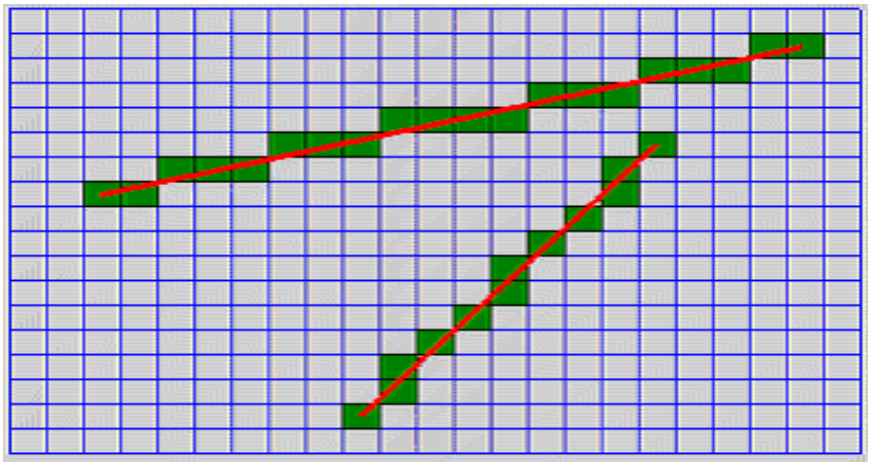
Hai trường hợp này dùng để vẽ một điểm bắt đầu từ bên trái đến điểm cuối cùng bên phải của đường thẳng (xem hình 1.5). Nếu điểm bắt đầu từ bên phải đến điểm cuối cùng bên trái thì xét ngược lại :

- 0<m<=1: $x_{i+1} := xi - 1$

  $y_{i+1} := yi - m \rightarrow int(yi+1)$

- m>1: $x_{i+1} := xi - 1/m \rightarrow int(xi+1)$

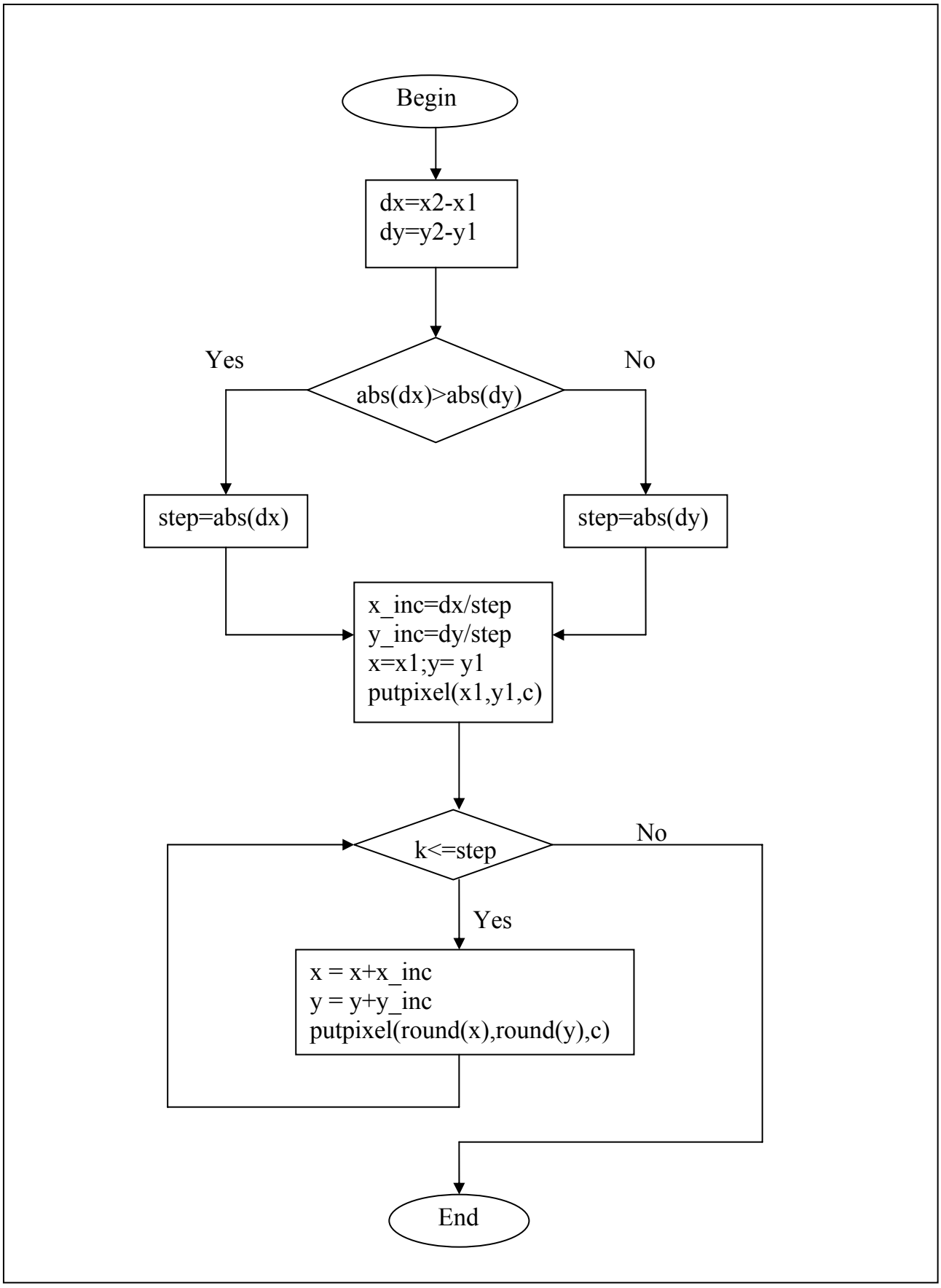
  $y_{i+1} := yi - 1$



Hình 1.5 : Hai dạng đường thẳng có 0<m<1 và m>1.

Tương tự, có thể tính toán các điểm vẽ cho trường hợp m<0: khi |m|<=1 hoặc |m|>1 (sinh viên tự tìm hiểu thêm).

Lưu đồ thuật toán DDA



Gọi (x_i+1, y_{i+1}) là điểm thuộc đoạn thẳng (xem hình 1.6). Ta có y:= $m(x_i+1)+b$.

Đặt $d_1 = y_{i+1} - y_i$

$d_2 = (y_i+1) - y_{i+1}$

Việc chọn điểm (x_{i+1}, y_{i+1}) là P1 hay P2 phụ thuộc vào việc so sánh d1 và d2 hay dấu của d1-d2.

- Nếu d1-d2<0 : chọn điểm P1, tức là $y_{i+1} = y_i$
- Nếu d1-d2 ≥0 : chọn điểm P2, tức là $y_{i+1} = y_i + 1$

Xét $P_i = \Delta x (d_1 - d_2)$

Ta có : $d_1 - d_2 = 2y_{i+1} - 2y_i - 1$

$= 2m(x_i+1) + 2b - 2y_i - 1$

$\Rightarrow \quad P_i = \Delta x (d_1 - d_2) = \Delta x[2m(x_i+1) + 2b - 2y_i - 1]$

$= \Delta x[2\frac{\Delta y}{\Delta x}(x_i+1) + 2b - 2y_i - 1]$

$= 2\Delta y(x_i+1) - 2\Delta x.y_i + \Delta x(2b - 1)$

$= 2\Delta y.x_i - 2\Delta x.y_i + 2\Delta y + \Delta x(2b - 1)$

Vậy $C = 2\Delta y + \Delta x(2b - 1) = Const$

$\Rightarrow \quad P_i = 2\Delta y.x_i - 2\Delta x.y_i + C$

Nhận xét rằng nếu tại bước thứ i ta xác định được dấu của P_i thì xem như ta xác định được điểm cần chọn ở bước (i+1). Ta có :

$P_{i+1} - P_i = (2\Delta y.x_{i+1} - 2\Delta x.y_{i+1} + C) - (2\Delta y.x_i - 2\Delta x.y_i + C)$

$\Leftrightarrow P_{i+1} = P_i + 2\Delta y - 2\Delta x (y_{i+1} - .y_i)$

- Nếu $P_i < 0$: chọn điểm P1, tức là $y_{i+1} = y_i$ và $P_{i+1} = P_i + 2\Delta y$.
- Nếu $P_i \geq 0$: chọn điểm P2, tức là $y_{i+1} = y_i + 1$ và $P_{i+1} = P_i + 2\Delta y - 2\Delta x$
- Giá trị P_0 được tính từ điểm vẽ đầu tiên (x_0, y_0) theo công thức :

$P_0 = 2\Delta y.x_0 - 2\Delta x.y_0 + C$

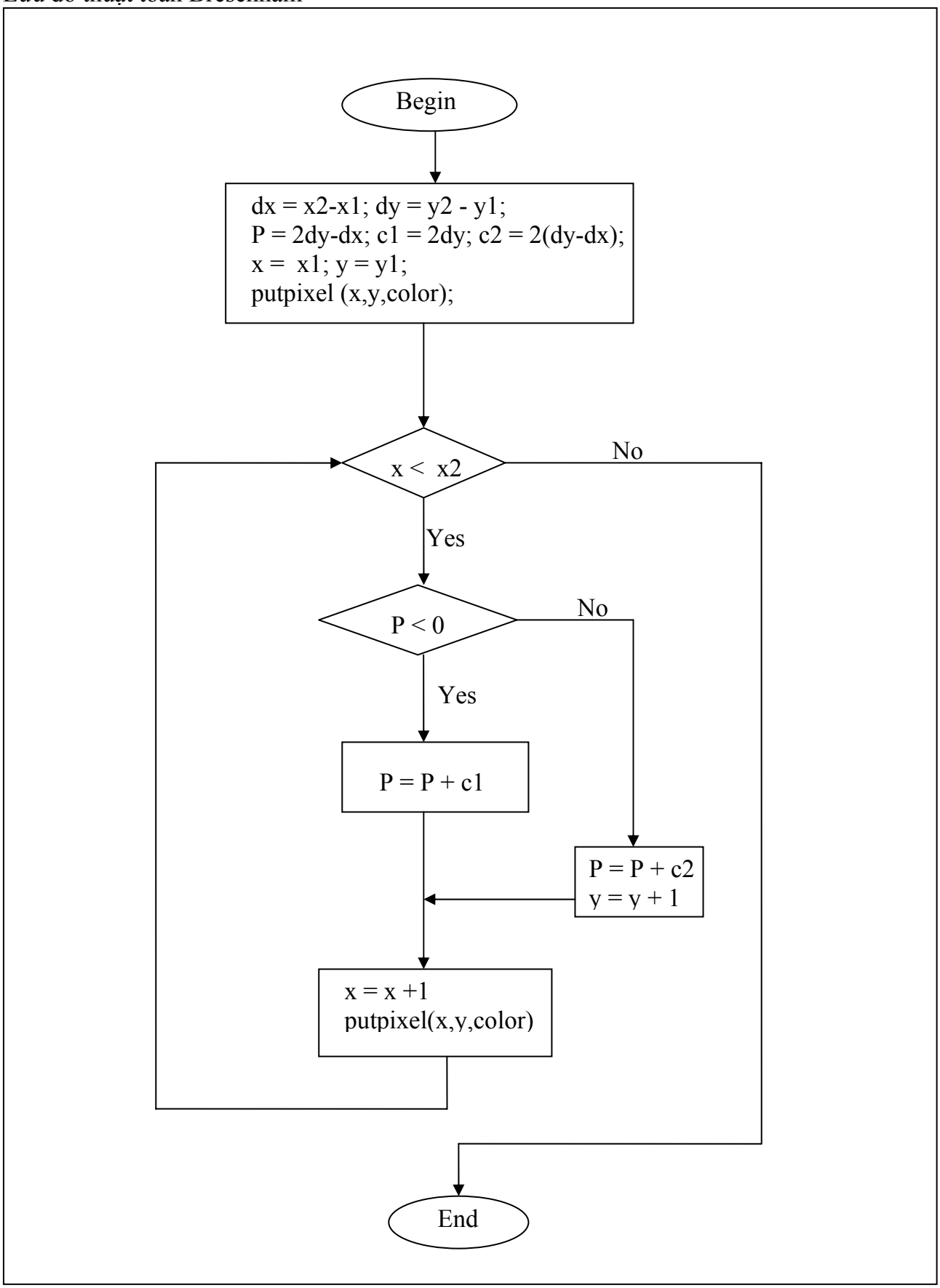
Do (x_0, y_0) là điểm nguyên thuộc về đoạn thẳng nên ta có :

$y_0 = m.x_0 + b = \frac{\Delta y}{\Delta x}.x_0 + b$

Thế vào phương trình trên ta được :

$P_0 = 2\Delta y - \Delta x$

Lưu đồ thuật toán Bresenham



Cài đặt minh họa thuật toán Bresenham

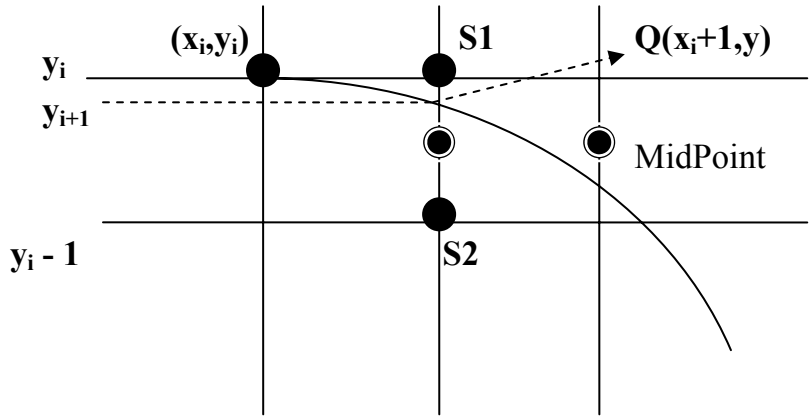
```
Procedure Bres_Line (x1,y1,x2,y2 : integer);
    Var dx, dy, x, y, P, const1, const2 : integer;
    Begin
        dx : = x2 - x1;  dy : = y2 - y1;
        P : = 2*dy - dx;
        Const1 : = 2*dy ; const2 : = 2*(dy - dx) ;
        x:= x1; y:=y1;
        Putpixel ( x, y, Color);
        while (x < x-2 ) do
            begin
                x : = x +1 ;
                if (P < 0) then P : = P + const1
        else
            begin
                    y : = y+1 ;
                    P : = P + const2
             end ;
                putpixel (x, y, color) ;
          end ;
End ;
```

Nhận xét :

Thuật toán Bresenham chỉ thao tác trên số nguyên và chỉ tính toán trên phép cộng và phép nhân 2 (phép dịch bit). Điều này là một cải tiến làm tăng tốc độ đáng kể so với thuật toán DDA.

Ý tưởng chính của thuật toán này là ở chổ xét dấu P_i để quyết định điểm kế tiếp, và sử dụng công thức truy hồi $P_{i+1} - P_i$ để tính P_i bằng các phép toán đơn giản trên số nguyên.

Tuy nhiên, việc xây dựng trường hợp tổng quát cho thuật toán Bresenham có phức tạp hơn thuật toán DDA.



Hình 1.8 : Đường tròn với điểm Q(x_i+1, y) và điểm MidPoint.

Đặt $F(x,y) = x^2 + y^2 - R^2$, ta có :

. $F(x,y) < 0$, nếu điểm (x,y) nằm trong đường tròn.

. $F(x,y) = 0$, nếu điểm (x,y) nằm trên đường tròn.

. $F(x,y) > 0$, nếu điểm (x,y) nằm ngoài đường tròn.

Xét $P_i = F(MidPoint) = F(x_i +1, y_i - 1/2)$. Ta có :

- Nếu $P_i < 0$: điểm MidPoint nằm trong đường tròn. Khi đó, điểm thực Q gần với điểm S1 hơn nên ta chọn $y_{i+1} = y_i$.

- Nếu $P_i >= 0$: điểm MidPoint nằm ngòai đường tròn. Khi đó, điểm thực Q gần với điểm S2 hơn nên ta chọn $y_{i+1} = y_i - 1$.

Mặt khác :

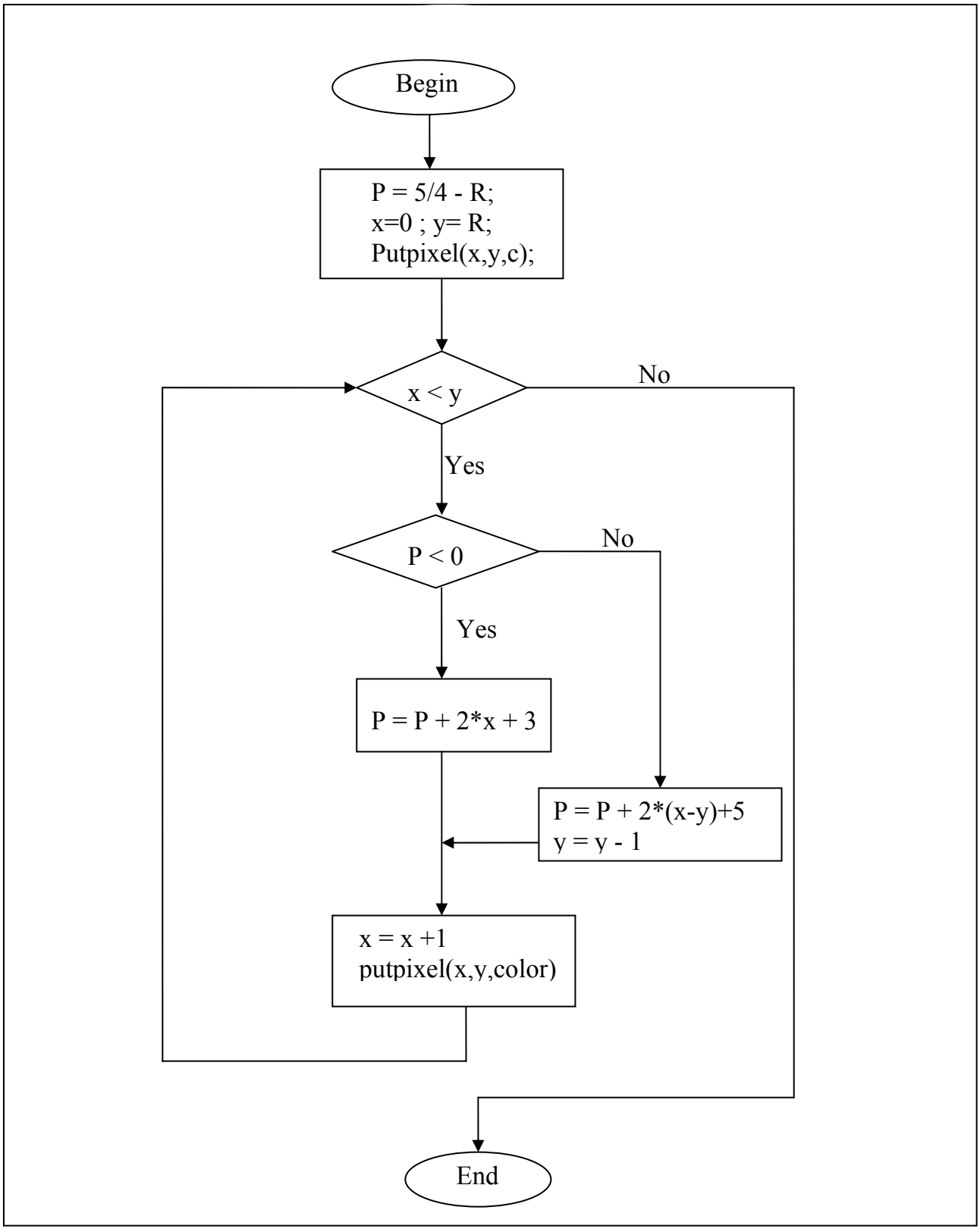
$$P_{i+1} - P_i = F(x_{i+1} +1, y_{i+1} - 1/2) - F(x_i + 1, y_i - 1/2)$$
$$= [(x_{i+1} +1)^2 + (y_{i+1} - 1/2)^2 - R^2] - [(x_i +1)^2 + (y_i - 1/2)^2 - R^2]$$
$$= 2x_i + 3 + ((y_{i+1})^2 + (y_i)^2) - (y_{i+1} - y_i)$$

Vậy :

- Nếu $P_i < 0$: chọn $y_{i+1} = y_i$. Khi đó $P_{i+1} = P_i + 2x_i +3$
- Nếu $P_i >= 0$: chọn $y_{i+1} = y_i - 1$. Khi đó $P_{i+1} = P_i + 2x_i - 2y_i +5$.
- P_i ứng với điểm ban đầu (x_0 , y_0) = (0,R) là:

$$P_0 = F(x_0 + 1, y_0 - 1/2) = F(1, R - 1/2) = \frac{5}{4} - R$$

Lưu đồ thuật toán MidPoint vẽ đường tròn



Chọn tọa độ pixel đầu tiên cần hiển thị là $(x_i, y_i) = (0,b)$. Cần xác định pixel tiếp theo là (x_{i+1}, y_{i+1}). Ta có :

$$\begin{cases} x_{i+1} = x_i + 1 \\ y_{i+1} = \begin{cases} y_i - 1 \\ y_i \end{cases} \end{cases}$$

$d1 = (y_i)^2 - y^2$

$d2 = y^2 - (y_i - 1)^2$

$P_i = d1 - d2$

Tính $P_{i+1} - P_i$

$$\Rightarrow \quad P_{i+1} = P_i + 2((y_{i+1})^2 - (y_i)^2) - 2(y_{i+1} - y_i) + \frac{2b^2}{a^2}(2x_i + 3)$$

- Nếu $P_i < 0$: chọn $y_{i+1} = y_i$. Khi đó $P_{i+1} = P_i + \frac{2b^2}{a^2}(2x_i + 3)$

- Nếu $P_i >= 0$: chọn $y_{i+1} = y_i - 1$. Khi đó $P_{i+1} = P_i + \frac{2b^2}{a^2}(2x_i + 3) + 4(1 - y_i)$

- P_i ứng với điểm ban đầu $(x_0, y_0) = (0,b)$ là: $P_0 = \frac{2b^2}{a^2} - 2b + 1$

Minh họa thuật toán vẽ Ellipse

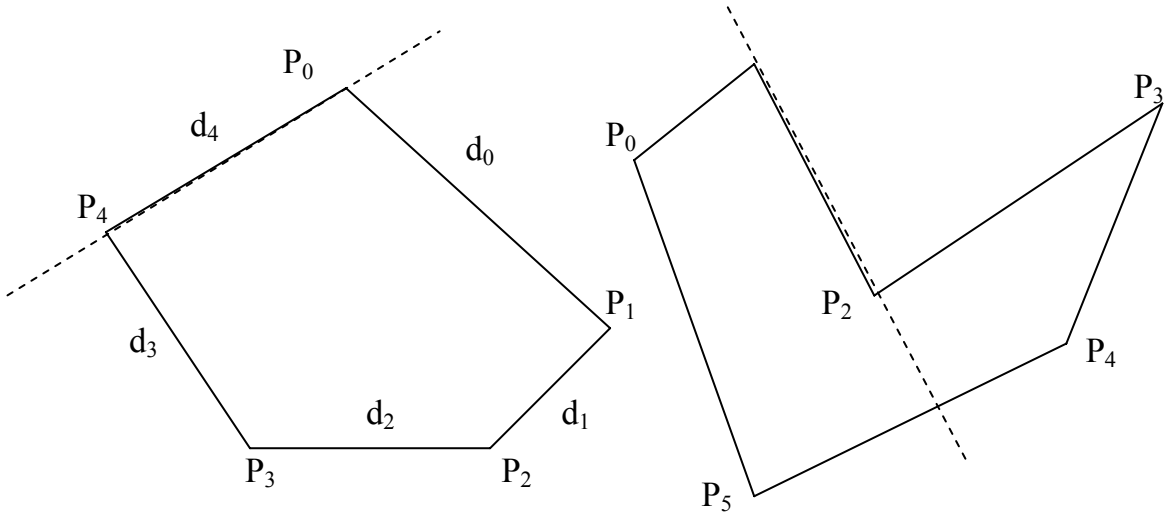
```
Procedure Ellipse(xc,yc,a,b : integer);
var     x,y : integer;
        z1, z2, P : real;
procedure dx;
begin
            putpixel (xc + x , yc +y, color) ;
            putpixel (xc - x , yc + y, color) ;
            putpixel (xc + x , yc - y, color) ;
            putpixel (xc -  x , yc- y, color) ;
end;
begin
        x:=0 y:=b;
```

- **Định nghĩa đa giác (Polygone)**: Đa giác là một đường gấp khúc kín có đỉnh đầu và đỉnh cuối trùng nhau (xem hình 1.10)
- **Xây dựng cấu trúc dữ liệu để vẽ đa giác**

```
Type
        d_dinh = record
                x,y: longint;
        end;
        dinh = array[0..10] of d_dinh;
var
        d: dinh;
```

Với cách xây dựng cấu trúc dữ liệu như thế này thì chúng ta chỉ cần nhập vào tọa độ các đỉnh và sau đó gọi thủ tục vẽ đường thẳng lần lượt qua 2 đỉnh như (0, 1), (1,2), ..., (n-1, n), trong đó đỉnh n trùng với đỉnh 0 thì ta sẽ vẽ được toàn bộ đa giác.

- **Đa giác được gọi là lồi** nếu bất kỳ đường thẳng nào đi qua một cạnh của đa giác thì toàn bộ đa giác nằm về một phía của đường thẳng đó. Ngược lại, nếu tồn tại ít nhất một cạnh của đa giác chia đa giác làm 2 phần thì gọi là **đa giác lõm** (xem hình 1.11).



Hình 1.11 : Đa giác lồi và đa giác lõm

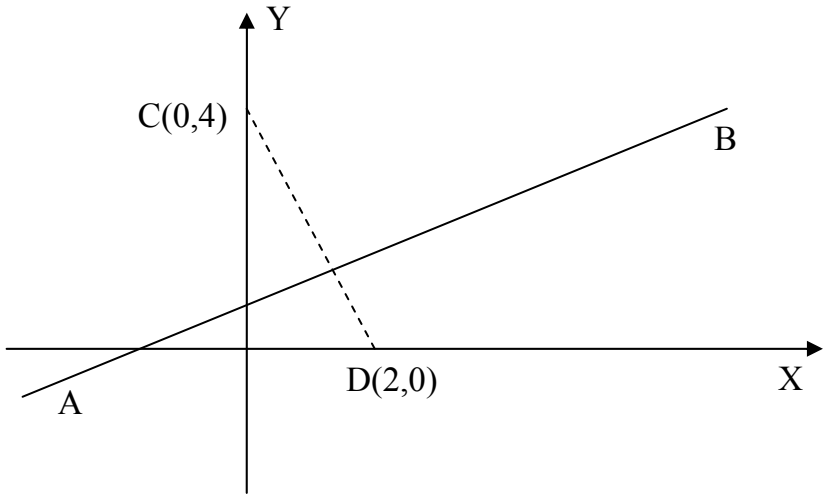
- **Thuật toán kiểm tra một đa giác là lồi hay lõm**

Thuật toán 1: Lần lượt thiết lập phương trình đường thẳng đi qua các cạnh của đa giác. Ứng với từng phương trình đường thẳng, xét xem các đỉnh còn lại có nằm về một

phía đối với đường thẳng đó hay không ? Nếu đúng thì kết luận đa giác lồi, ngược lại là đa giác lõm.

Nhận xét : Phương trình đường thẳng y = ax + b chia mặt phẳng ra làm 2 phần. Các điểm nằm C(xc,yc) trên đường thẳng sẽ có yc > axc + b và các điểm D(xd,yd) nằm phía dưới đường thẳng sẽ có yd < axd + b.

Ví dụ : Cho đường thẳng AB có phương trình y = $\frac{1}{2}$x + 1 và hai điểm C, D có tọa độ là C(0,4), D(2,0) (xem hình 1.12).



Hình 1.12 : Đường thẳng AB và 2 điểm C, D.

Ta có : $Y_c = 4 > ax_c + b = \frac{1}{2}.0 + 1$

và $Y_d = 0 < ax_d + b = \frac{1}{2}.2 + 1$

Vậy hai điểm C, D nằm về hai phía đối với đường thẳng AB.

Thuật toán 2 :

Nhận xét :

Trong mặt phẳng Oxy, cho 2 véc tơ $\vec{a}$ và $\vec{b}$, Tích vô hướng của 2 véc tơ là :

$$T(\vec{a}, \vec{b}) = \begin{vmatrix} a_x & a_y \\ b_x & b_y \end{vmatrix} = a_x * b_y - a_y * b_x$$

Khi đó :

$\vec{a}$ quẹo trái sang $\vec{b}$ nếu T ≥0

$\vec{a}$ quẹo phải sang $\vec{b}$ nếu T < 0

2. Viết chương trình thực hiện 2 thao tác sau :

   - Khởi tạo chế độ đồ họa, đặt màu nền, đặt màu chữ, định dạng chữ (settextstyle(f,d,s)), xuất một chuỗi ký tự ra màn hình. Đổi font, hướng, kích thước.

   - Xuất một chuỗi ra màn hình, chuỗi này có tô bóng.

(lưu ý rằng nội dung chuỗi ký tự, màu tô, màu bóng là được nhập từ bàn phím).

3. Viết chương trình vẽ đoạn thẳng AB với màu color theo giải thuật DDA. Biết rằng tọa độ A,B, color được nhập từ bàn phím. Trang trí màu nền, ghi chú các tọa độ A, B ở hai đầu đoạn thẳng.
4. Tương tự như bài tập 3 nhưng sử dụng giải thuật Bresenham. Lưu ý các trường hợp đặc biệt của hệ số góc.
5. Tổng hợp bài tập 4, viết chương trình vẽ đường thẳng bằng giải thuật Bresenham cho tất cả các trường hợp của hệ số góc. Lưu ý xét trường hợp đặc biệt khi đường thẳng song song với trục tung hay với trục hoành.
6. Viết chương trình nhập tọa độ 3 điểm A, B, C từ bàn phím. Tìm tọa độ điểm D thuộc AB sao cho CD vuộng góc AB. Vẽ đoạn thẳng AB và CD.
7. Viết chương trình xét vị trí tương đối của 2 đoạn thẳng AB và CD. Biết rằng trong màn hình đồ họa đoạn thẳng AB và CD được gọi là cắt nhau khi hai điểm A, B ở về hai phía của CD và ngược lại.
8. Viết chương trình vẽ đường tròn theo giải thuật đơn giản (đối xứng).
9. Viết chương trình vẽ đường tròn theo giải thuật Bresenham.
10. Viết chương trình vẽ đường tròn theo giải thuật MidPoint.
11. Viết chương trình vẽ một đường tròn tâm O bán kính R. Vẽ các đường tròn đồng tâm với O, có bán kính chạy từ 1 đến R. Sau đó xoá các đường tròn đồng tâm này và vẽ các đường tròn đồng tâm khác đi từ R đến 1.
12. Viết chương trình vẽ một đường tròn tâm O bán kính R. Hãy vẽ một đoạn thẳng từ tâm O độ dài R. Hãy quay đoạn thẳng này quanh đường tròn.
13. Viết chương trình vẽ Elippse.
14. Viết chương trình vẽ Elippse có bán kính lớn là a, bán kính nhỏ là b và một đường tròn nội tiếp Elippse. Tô đường tròn bằng các đường tròn đồng tâm. Sau đó tô elippse bằng các elippse đồng tâm có bán kính lớn chạy từ b đến a, bán kính nhỏ là b.

15. Viết chương trình vẽ một hình chữ nhật, một hình vuông và một hình bình hành. Yêu cầu chú thích tọa độ các đỉnh.
16. Viết chương trình vẽ một tam giác. Tọa độ các đỉnh được nhập từ bàn phím, mỗi cạnh có một màu khác nhau.
17. Viết chương trình vẽ một đa giác có n đỉnh.
18. Viết chương trình xét tính lồi lõm của một đa giác bằng cách thiết lập phương trình đường thẳng đi qua các cạnh của đa giác.
19. Viết chương trình xét tính lồi lõm của một đa giác bằng cách thiết lập các véc tơ chỉ phương của các cạnh.

- **Tô đường tròn**

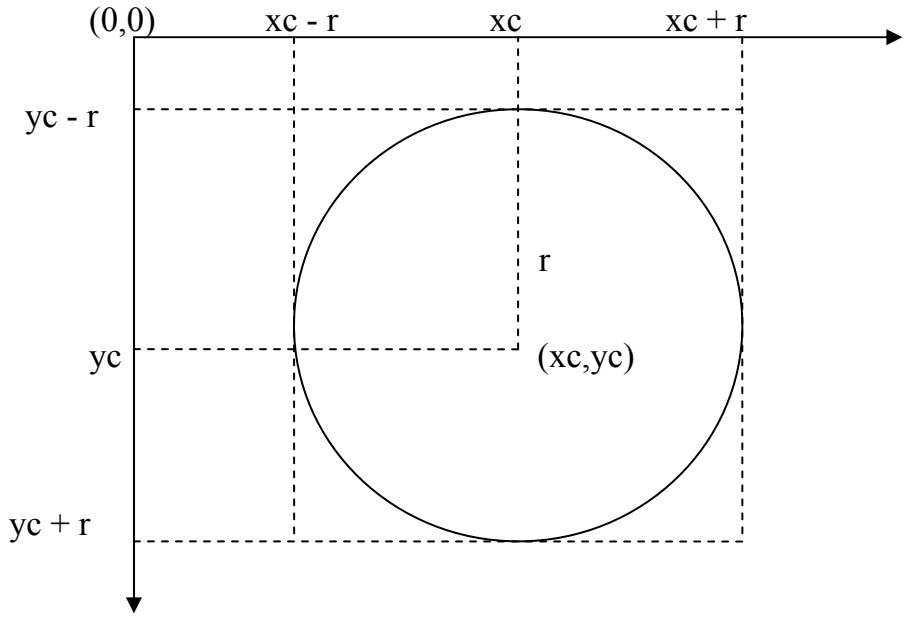
- Để tô đường tròn thì ta tìm hình vuông nhỏ nhất ngoại tiếp đường tròn bằng cách xác định điểm trên bên trái (xc-r, yc-r) và điểm dưới bên phải (xc+r, yc+r) của hình vuông (xem hình 2.2).

- Cho i đi từ xc-r đến xc+r

Cho j đi từ yc-r đến yc+r

Tính khoảng cách d giữa hai điểm (i,j) và tâm (xc,yc)

Nếu d<r thì tô điểm (i,j) với màu muốn tô

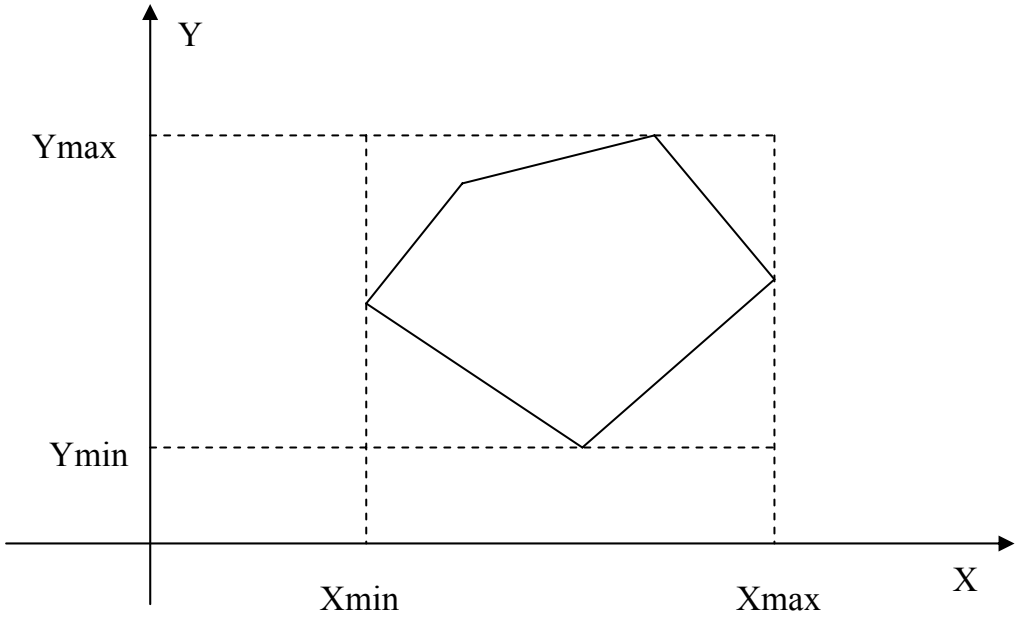


Hình 2.3 : đường tròn nội tiếp hình vuông.

- **Tô đa giác**

- Tìm hình chữ nhật nhỏ nhất có các cạnh song song với hai trục tọa độ chứa đa giác cần tô dưa vào hai tọa độ (xmin, ymin), (xmax, ymax). Trong đó, xmin, ymin là hoành độ và tung độ nhỏ nhất, xmax, ymax là hoành độ và tung độ lớn nhất của các đỉnh của đa giác.

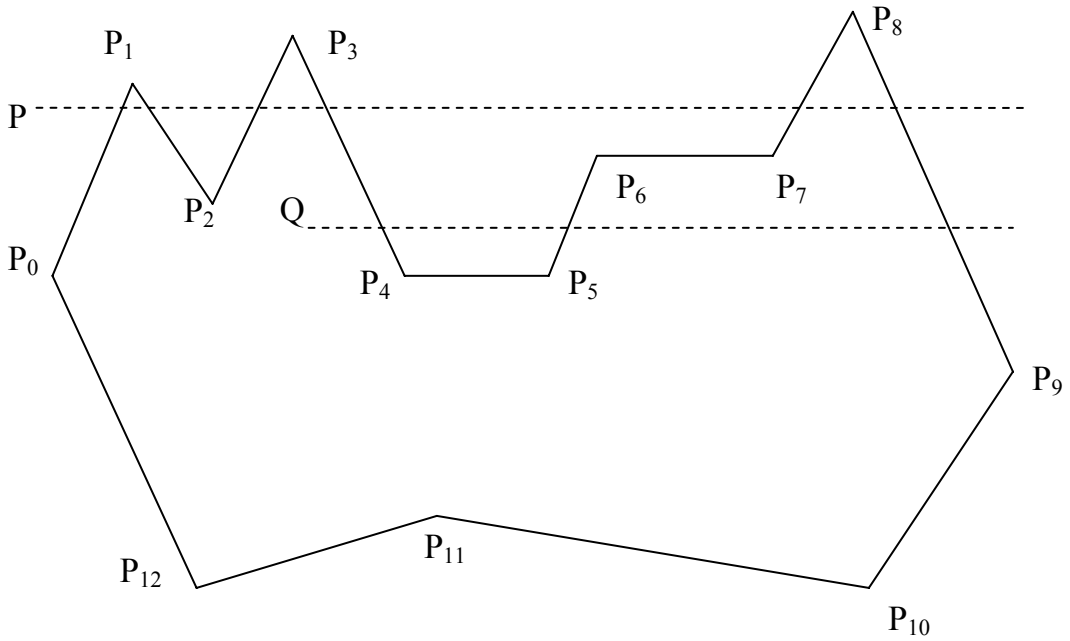
- Cho x đi từ xmin đến xmax, y đi từ ymin đến ymax (hoặc ngược lai). Xét điểm P(x,y) có thuộc đa giác không ? Nếu có thì tô với màu cần tô (xem hình 2.4).



Hình 2.4 : đa giác nội tiếp hình chữ nhật.

Thông thường một điểm nằm trong đa giác thì số giao điểm từ một tia bất kỳ xuất phát từ điểm đó cắt biên của đa giác phải là một số lẻ lần. Đặc biệt, tại các đỉnh cực trị (cực đại hay cực tiểu) thì một giao điểm phải được tính 2 lần (xem hình 2.5). Tia có thể qua phải hay qua trái. Thông thường ta chọn tia qua phải.

Ví dụ : Xét đa giác gồm 13 đỉnh là P_0 , P_1 , ..., $P_{12} = P_0$
(xem hình 2.5).



Hình 2.5 : Đa giác có 13 đỉnh.

Lưu ý :

Gọi tung độ của đỉnh P_i là $P_i.y$. Nếu :

- $P_i.y < Min (P_{i+1}.y, P_{i-1}.y)$ hay $P_i.y > Max (P_{i+1}.y, P_{i-1}.y)$ thì P_i là đỉnh cực trị (cực tiểu hay cực đại).
- $P_{i-1}.y < P_i.y < P_{i+1}.y$ hay $P_{i-1} > P_i.y > P_{i+1}.y$ thì P_i là đỉnh đơn điệu.
- $P_i = P_{i+1}$ và $P_i.y < Min (P_{i+2}.y, P_{i-1}.y)$ hay $P_i > Max (P_{i+2}.y, P_{i-1}.y)$ thì đoạn $[P_i,P_{i+1}]$ là đoạn cực trị (cực tiểu hay cực đại).
- $P_i = P_{i+1}$ và $P_{i-1}.y < P_i.y < P_{i+2}.y$ hay $P_{i-1} > P_i.y > P_{i+2}.y$ thì đoạn $[P_i,P_{i+1}]$ là đoạn đơn điệu.

- **Thuật toán kiểm tra điểm có nằm trong đa giác**

- Với mỗi đỉnh của đa giác ta đánh dấu là 0 hay 1 theo qui ước như sau: nếu là đỉnh cực trị hay đoạn cực trị thì đánh số 0. Nếu là đỉnh đơn điệu hay đoạn đơn điệu thì đánh dấu 1.
- Xét số giao điểm của tia nữa đường thẳng từ P là điểm cần xét với biên của đa giác. Nếu số giao điểm là chẵn thì kết luận điểm không thụôc đa giác. Ngược lại, số giao điểm là lẻ thì điểm thuộc đa giác.

- **Minh họa thuật toán xét điểm thuộc đa giác**

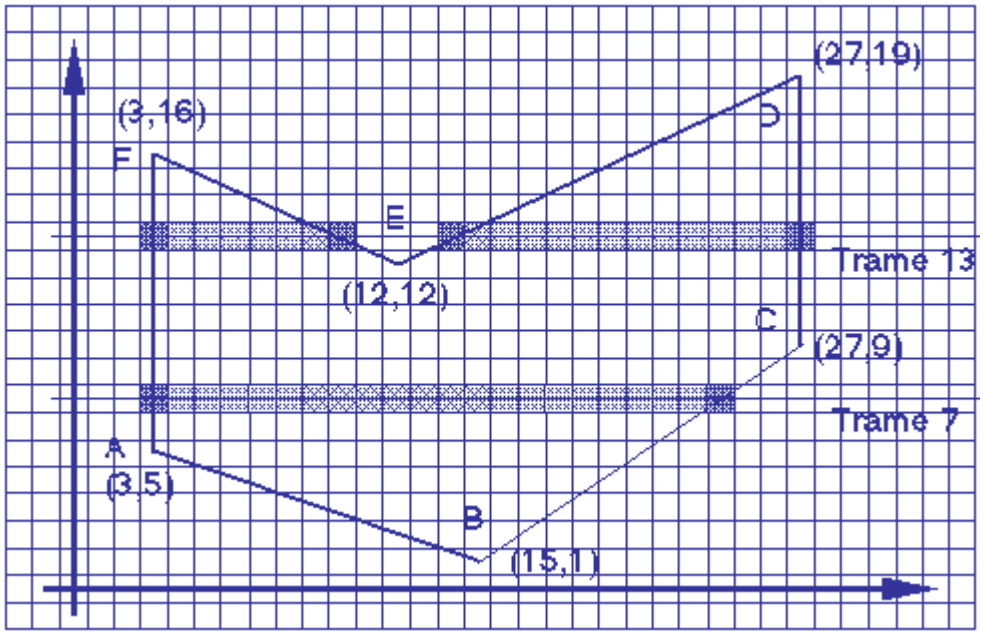
```
function  PointInpoly(d: dinh;  P: d_dinh; n: integer): boolean;
var       count, i: integer;
          x_cut: longint;
function  next(i: integer): integer;
begin
          next := (i + n + 1) mod n
end;
function  prev(i: integer): integer;
 begin
          prev := (i + n - 1) mod n
end;
begin
          count := 0;
for i := 0 to n-1 do
```

```
            if   d[i].y = P.y   then
            begin
                  if    d[i].x > P.x    then
                  begin
                        if ((d[prev(i)].y < P.y) and (P.y < d[next(i)].y)) or
                        ((d[prev(i)].y > P.y) and (P.y > d[next(i)].y)) then
                        count := count + 1;
                        if  d[next(i)].y = P.y then
                        if ((d[prev(i)].y < P.y) and (P.y < d[next(next(i))].y)) or
                         ((d[prev(i)].y > P.y  and (P.y > d[next(next(i))].y)) then
                         count := count + 1;
                  end;
            end   else   {d[i].y = P.y}
            if ((d[i].y < P.y) and (P.y < d[next(i)].y)) or
             ((d[i].y > P.y) and (P.y > d[next(i)].y)) then
            begin
                  x_cut := d[i].x + Round((d[next(i)].x - d[i].x)
                   / (d[next(i)].y - d[i].y) * (P.y - d[i].y));
                  if x_cut >= P.x then count := count + 1;
            end;
      if  (count mod 2 = 0) then   PointInPoly :=  false
      else PointInpoly := true;
end;
```

- **Minh họa thuật toán tô đa giác**

(xmin, ymin, xmax, ymax: đã khai báo trong chương trình chính.)

```
Procedure Todg ( d:dinh; n,maubien : integer ; d: dinh; n:integer ) ;
var     x, y:integer;
        P: d_dinh;
begin
        for x:=xmin  to  xmax do
                for   y:= ymin to ymax  do
```



Hình 2.6 : Tô đa giác bằng giải thuật scan -line.

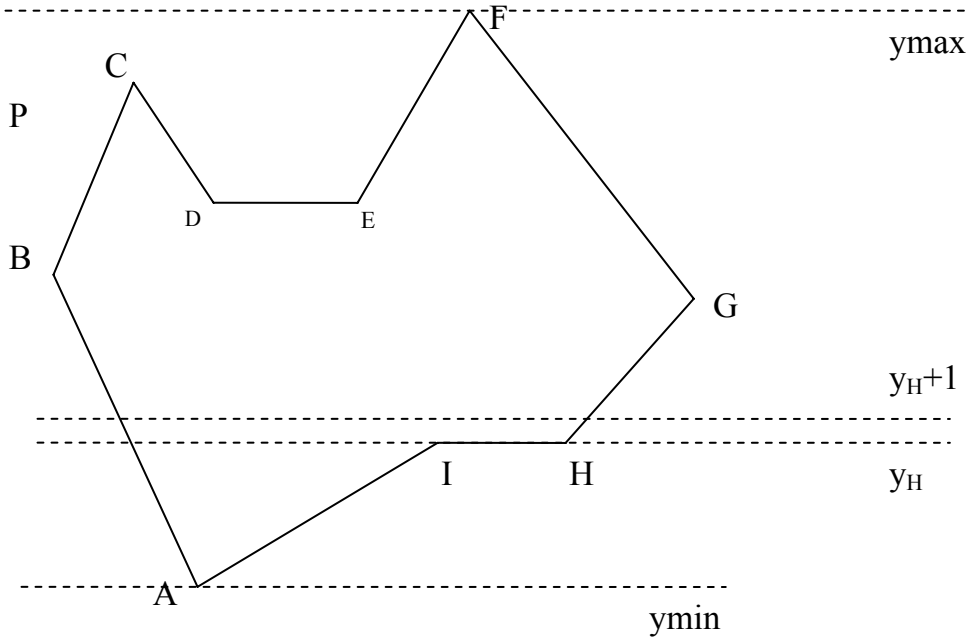
- **Các vấn đề cần lưu ý:**

- Hạn chế đuợc số cạnh cần tìm giao điểm ứng với mỗi dòng quét vì ứng với mỗi dòng quét không phải lúc nào cũng giao điểm với các cạnh của đa giác.
- Xác định nhanh hoàn độ giao điểm vì nếu lặp lại thao tác tìm giao điểm của cạnh đa giác với mỗi dòng quét sẽ tốn rất nhiều thời gian.
- Giải quyết trường hợp số giao điểm đi qua đỉnh đơn điệu thì tính số giao điểm là 1 hay đi qua đỉnh cực trị.thì tính số giao điểm là 0 (hoặc 2).

- **Tổ chức cấu trúc dữ liệu và thuật toán**

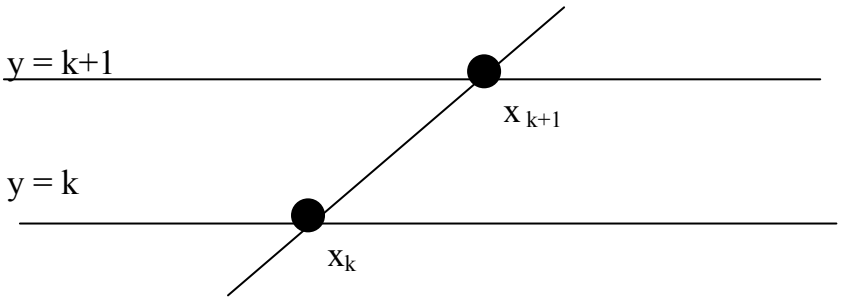
- Danh sách các cạnh (Edge Table - ET) : chứa toàn bộ các cạnh của đa giác (loại các cạnh song song với trục Ox) được sắp theo thứ tự tăng dần của trục y. Xem hình 2.5 ta có thể sắp xếp các cạnh trong ET là : AB, AI, HG, BC, GF, DC, EF (loại IH và DE)
- Danh sách các cạnh đang kích hoạt (Active Edge Table - AET) : chứa các cạnh của đa giác có thể cắt ứng với dòng quét hiện hành, các cạnh này được sắp theo thứ tự tăng dần của hoành độ giao điểm của hoành độ giao điểm giữa cạnh và dòng quét.
- Khi dòng quét đi từ ymin đến ymax, các cạnh thoả điều kiện sẽ được chuyển từ ET sang AET. Nghĩa là, khi dòng quét y=k bắt đầu cắt một cạnh, khi đó k ≥ ymin, cạnh này sẽ được chuyển từ ET sang AET. Khi dòng quét không còn cắt cạnh này nữa, khi

đó, k > ymax, cạnh này sẽ bị loại khỏi AET. Khi không còn cạnh nào trong ET hay AET thì quá trình tô màu kết thúc (xem hình 2.5).



Hình 2.7 : Tô đa giác bằng giải thuật scan -line.

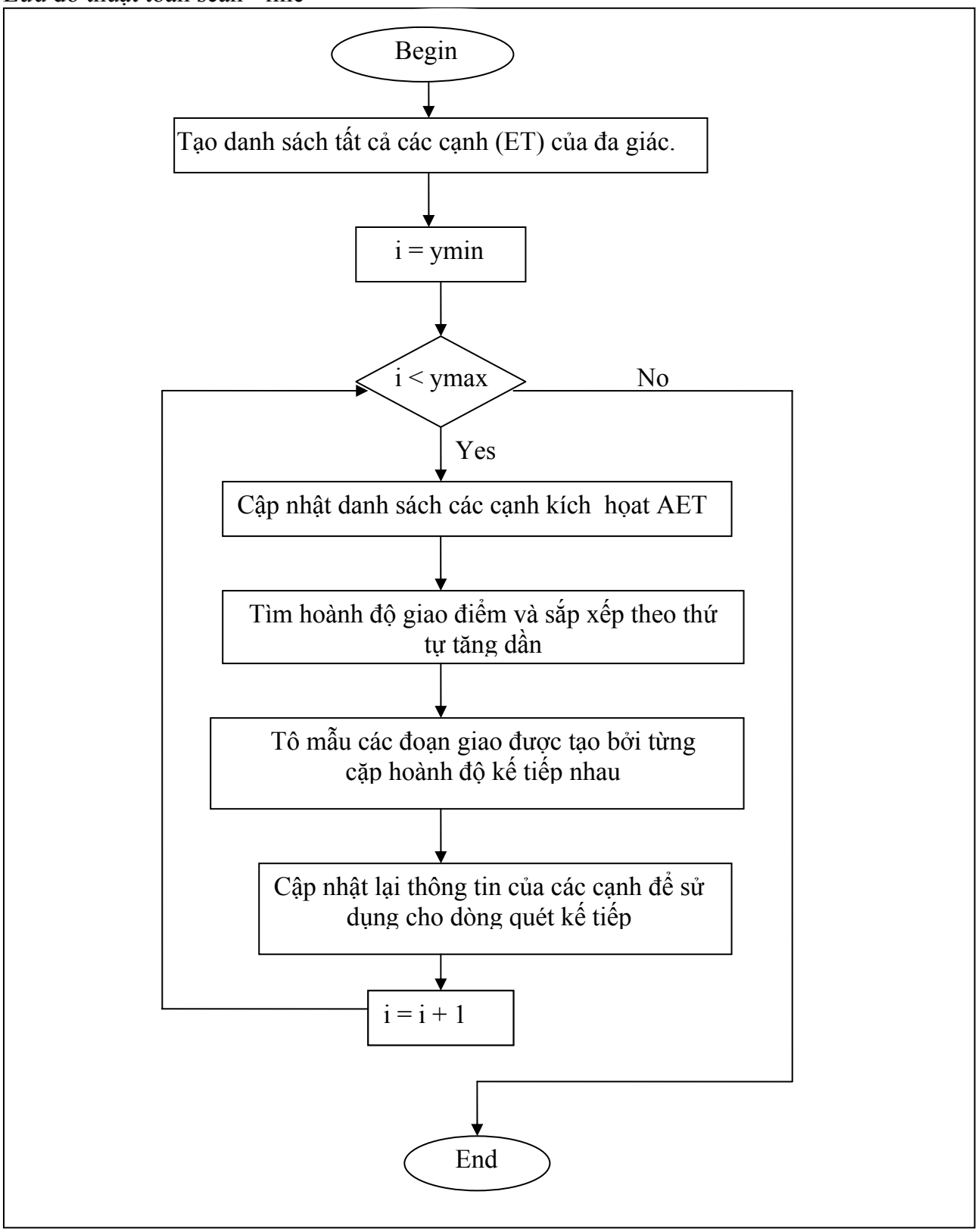
- Để tìm giao điểm giữa cạnh đa giác và dòng quét, ta có nhận xét sau :



$$x_{k+1} - x_k = \frac{1}{m}((k+1) - k) = \frac{1}{m} \quad \text{hay} \quad x_{k+1} = x_k + \frac{1}{m}$$

Trong đó m là hệ số góc của cạnh.

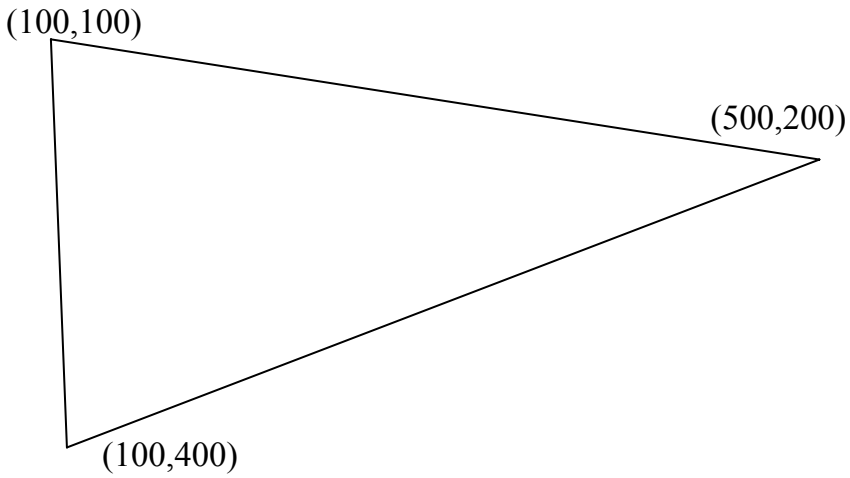
Lưu đồ thuật toán scan - line



```
            putpixel(x,y,color);
            Boundary_fill ( x+1,y, mauto, maubien );
            Boundary_fill ( x-1,y, mauto, maubien );
            Boundary_fill ( x,y+1, mauto, maubien );
            Boundary_fill ( x,y-1, mauto, maubien );
        end;
end;
```

Nhận xét :

- Thuật toán có thể không chính xác khi có một số điểm nằm trong vùng tô có màu là màu cần tô của vùng.
- Việc thực hiện gọi đệ qui làm thuật toán không thể sử dụng cho vùng tô lớn (tràn stack).
- Có thể khắc phục việc tràn stack bằng cách giảm số lần gọi đệ qui. Khởi đầu điểm (x,y) là điểm có vị trí đặc biệt trong vùng tô, sau đó, gọi đệ qui các điểm lân cận của (x,y) (xem hình 2.8).



Hình 2.10: Tam giác với 3 tọa độ đỉnh.

Ví dụ 1: Trong hình 2.10, ta có thể xét điểm (x,y) có tọa độ là (498, 200). Với điểm khởi đầu này thì chỉ cần xét 3 điểm lân cận là (x-1,y), (x,y-1), (x,y+1). Khi đó thủ tục tô màu theo đường biên được viết lại như sau :

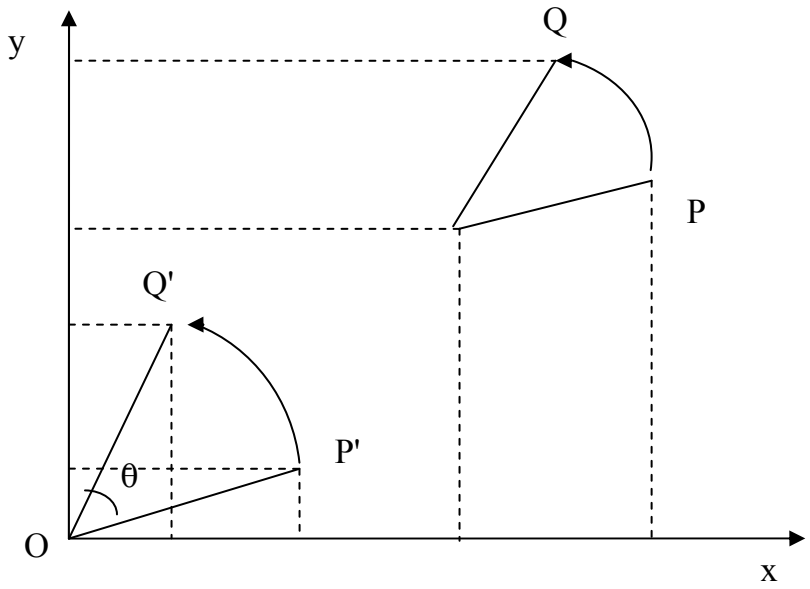
```
 Procedure Boundary_fill ( x,y,mauto, maubien :integer);
var mau_ht : integer;
```

```
begin
        mau_ht:= getpixel(x,y);
        if  (mau_ht <> mauto) and (mau_ht <> maubien) then
        begin
                putpixel(x,y,color);
                Boundary_fill ( x-1,y, mauto, maubien );
                Boundary_fill ( x,y+1, mauto, maubien );
                Boundary_fill ( x,y-1, mauto, maubien );
        end;
end;
```

Ví dụ 2: Trong hình 2.10, ta có thể xét điểm (x,y) có tọa độ là (102, 102). Với điểm khởi đầu này thì chỉ cần xét 2 điểm lân cận là (x+1,y), (x,y+1). Khi đó thủ tục tô màu theo đường biên được viết lại như sau :

```
 Procedure Boundary_fill ( x,y,mauto, maubien :integer);
var mau_ht : integer;
begin
        mau_ht:= getpixel(x,y);
        if  (mau_ht <> mauto) and (mau_ht <> maubien) then
        begin
                putpixel(x,y,color);
                Boundary_fill ( x+1,y, mauto, maubien );
                Boundary_fill ( x,y+1, mauto, maubien );
        end;
end;
```

- **Phép quay quanh một điểm bất kỳ**



Hình 3.3 : Phép quay quanh một điểm bất kỳ.

Xét điểm P(P.x,P.y) quay quanh điểm V(V.x, V.y) một góc θ đến điểm Q(Q.x,Q.y). Ta có thể xem phép quay quanh tâm V được kết hợp từ phép các biến đổi cơ bản sau:

- Phép tịnh tiến (-V.x, -V.y) để dịch chuyển tâm quay về gốc tọa độ
- Quay quanh gốc tọa độ O một góc θ
- Phép tịnh tiến (+V.x, +V.y) để đưa tâm quay về vị trí ban đầu

Ta cần xác định tọa độ của điểm Q (xem hình 3.3).

- Từ phép tịnh tiến (-V.x,-V.y) biến đổi điểm P thành P' ta được:

$$P' = P + V$$

hay
$$\begin{cases} P'.x = P.x - V.x \\ P'.y = P.y - V.y \end{cases}$$

- Phép quay quanh gốc tọa độ biến đổi điểm P' thành Q'

$$Q' = P'.M$$

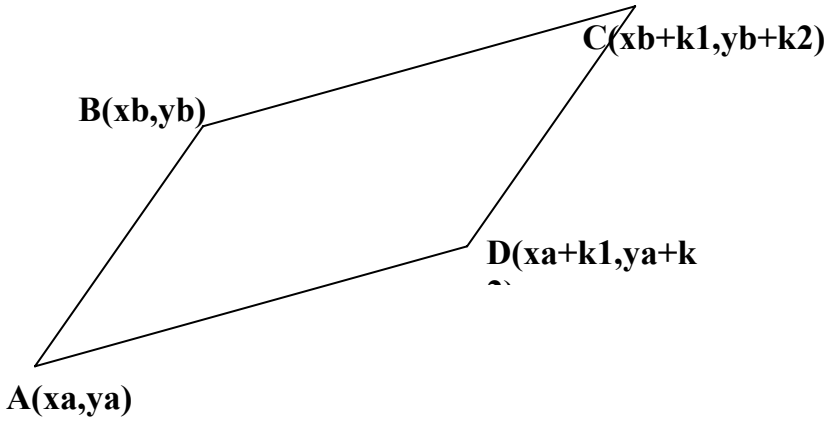
$$\begin{cases} Q'.x = P'.x*\cos\theta - P'.y*\sin\theta \\ Q'.y = P'.x*\sin\theta + P'.y*\cos\theta \end{cases}$$

- Phép tịnh tiến (+V.x, +V.y) biến đổi điểm Q' thành Q ta được

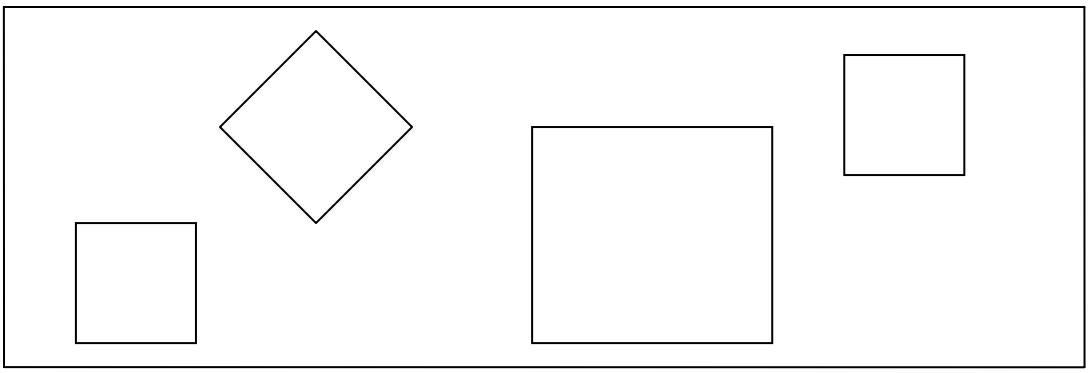
$$Q = Q' + V$$

hay
$$\begin{cases} Q.x = Q'.x + V.x \\ Q.y = Q'.y + V.y \end{cases}$$

$$\begin{cases} Q.x = (P.x - V.x)*\cos\theta - (P.y - V.y)*\sin\theta + V.x \end{cases}$$



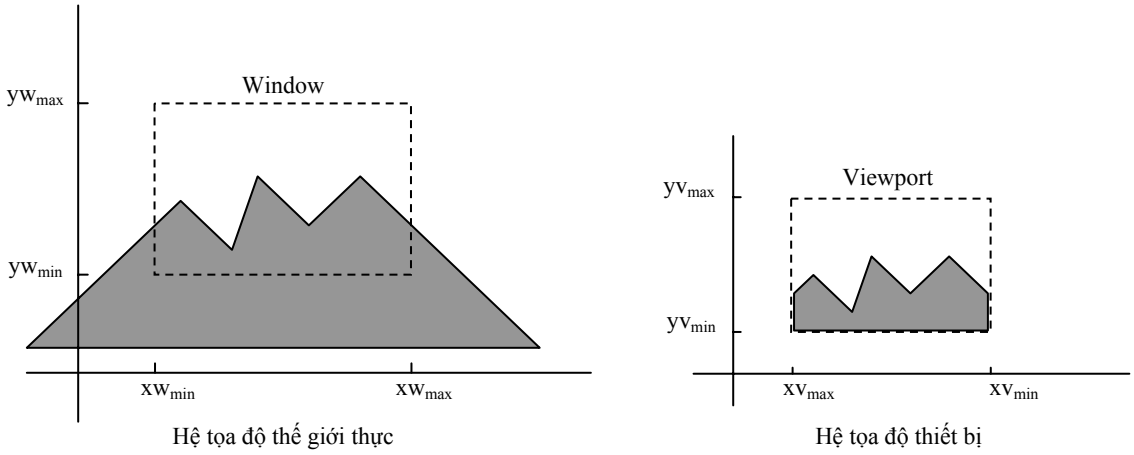
2. Viết chương trình vẽ một hình vuông ABCD (xem hình vẽ).
   - Tịnh tiến hình vuông đó đến vị trí khác.
   - Phóng to hình vuông ABCD.
   - Biến dạng hình vuông thành hình thoi.



3. Vẽ một elip, sau đó vẽ thêm 3 elip khác có cùng tâm với elip đã cho, có độ dãn ở trục Ox là K và Oy là 1.
4. Vẽ một elip nghiêng một góc G độ có các trục không song song với các trục tọa độ.
5. Vẽ một bông hoa bằng cách vẽ các elip nghiêng một góc G độ với các màu khác nhau. Vẽ đến khi nào ấn phím bất kỳ thì ngưng.
6. Viết chương trình mô phỏng sự chuyển động của elip bằng cách cho elip này quay quanh tâm của nó.
7. Viết chương trình mô phỏng sự chuyển động của trái đất quay quanh mặt trời.
8. Viết chương trình vẽ một đường tròn tâm O bán kính R. Vẽ một đường kính AB. Quay đường kính này quanh tâm đường tròn.

9. Viết chương trình vẽ đoạn thẳng AB.

tịnh tiến, biến đổi tỷ lệ vùng được chọn và xóa bỏ những phần bên ngoài vùng được chọn. Những thao tác này được gọi là **windowing** và **clipping** (xem hình 4.1).



Hình 4.1 : **Một ánh xạ cửa sổ - đến – vùng quan sát**

Một vùng có dạng hình chữ nhật được xác định trong hệ tọa độ thế giới thực được gọi là một **cửa sổ (window)**. Còn vùng hình chữ nhật trên thiết bị hiển thị để cửa sổ đó ánh xạ đến được gọi là một **vùng quan sát (viewport)**. Hình 4.1 minh họa việc ánh xạ một phần hình ảnh vào trong một viewport. Việc ánh xạ này gọi là **một phép biến đổi hệ quan sát (viewing transformation), biến đổi cửa sổ (windowing tranformation), biến đổi chuẩn hóa (normalization transformation).**

Các lệnh để xây dựng một cửa sổ và vùng quan sát từ một chương trình ứng dụng có thể được định nghĩa như sau:

```
set_window(xw_min, xw_max, yw_min, yw_max)
set_viewport(xv_min, xv_max, yv_min, yv_max)
```

Các tham số trong mỗi hàm được dùng để định nghĩa các giới hạn tọa độ của các vùng chữ nhật. Các giới hạn của *cửa sổ* được xác định trong hệ tọa độ thế giới thực. Hệ tọa độ thiết bị chuẩn thường được dùng nhất cho việc xác định *vùng quan sát*, dù rằng hệ tọa độ thiết bị có thể được dùng nếu chỉ có một thiết bị xuất (output device) duy nhất trong hệ thống. Khi hệ tọa độ thiết bị chuẩn được dùng, lập trình viên xem thiết bị xuất có giá trị tọa độ trong khoảng 0..1. Một sự xác định vùng quan sát được cho với các giá trị trong khoảng này. Các việc xác định sau đây, đặt một phần

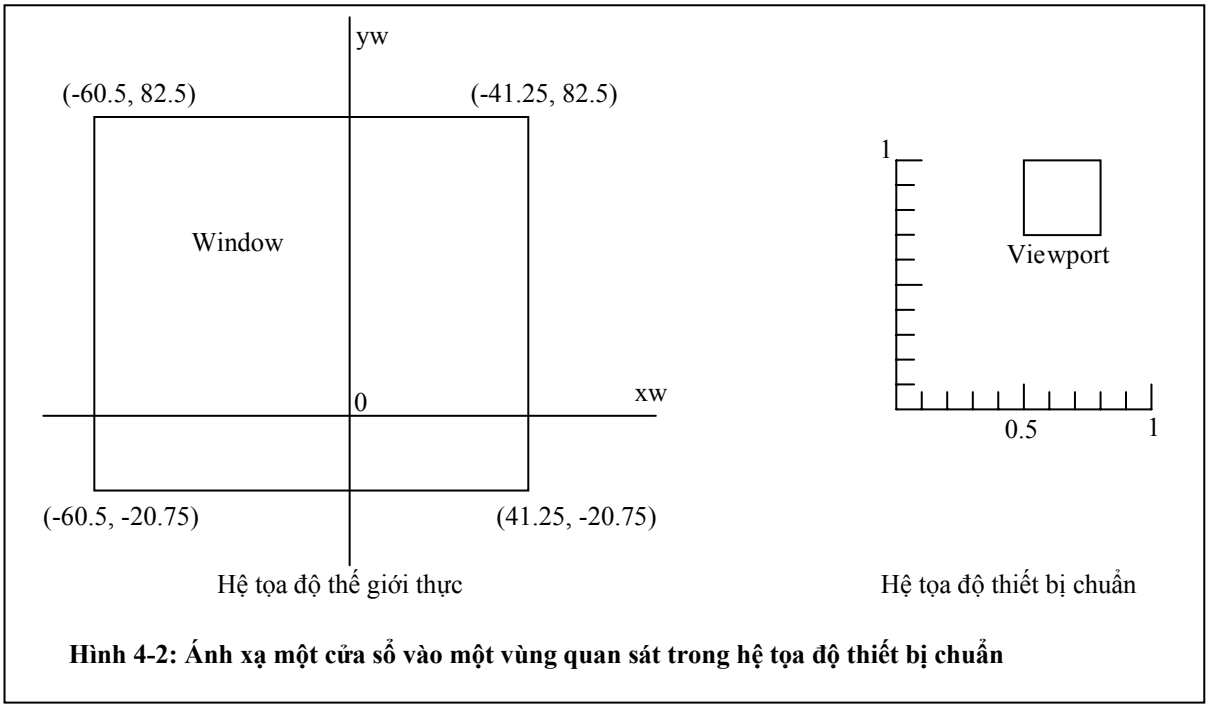
của sự định nghĩa hệ tọa độ thế giới thực vào trong góc trên bên phải của vùng hiển thị, như được minh họa trong hình 4-2:

```
set_window(-60.5, 41.25, -20.75, 82.5);
set_viewport(0.5, 0.8, 0.7, 1.0);
```

Nếu một cửa sổ buộc phải được ánh xạ lấp đầy vùng hiển thị, sự xác định viewport được cho là:

```
Set_viewport(0,1, 0, 1)
```

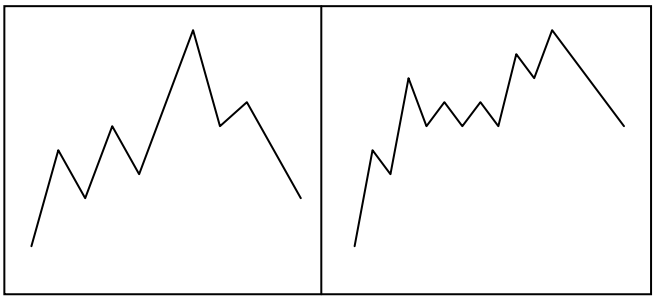
Các vị trí được biểu diễn trên hệ tọa độ thiết bị chuẩn phải được biến đổi sang hệ tọa độ thiết bị trước khi được hiển thị bởi một thiết bị xuất cụ thể. Thông thường một thiết bị xác định được chứa trong các gói đồ họa cho mục đích này. Thuận lợi của việc dùng hệ tọa độ thiết bị chuẩn là để các gói đồ họa độc lập với thiết bị. Các thiết bị xuất khác nhau có thể được dùng nhờ việc cung cấp các trình điều khiển thiết bị thích hợp. Mọi điểm được tham khảo đến trong các gói đồ họa phải được xác định tương ứng trong hệ tọa độ Descartes. Bất kỳ sự định nghĩa hình ảnh nào dùng trong một hệ tọa độ khác, như hệ tọa độ cực, người sử dụng trước tiên phải biến đổi nó sang hệ tọa độ thế giới thực. Những hệ tọa độ Descart này sau đó được dùng trong các lệnh cửa sổ để xác định phần nào của hình ảnh muốn được hiển thị (xem hình 4.2).



Hình 4-2: Ánh xạ một cửa sổ vào một vùng quan sát trong hệ tọa độ thiết bị chuẩn

Các lệnh về cửa sổ và vùng quan sát được phát biểu trước khi gọi các thủ tục vẽ ảnh. Các sự xác lập cho cửa sổ và vùng quan sát sẽ ảnh hưởng đến bất kỳ lệnh xuất theo sau nào cho đến khi có một sự xác lập mới.

Bằng việc thay đổi vị trí vùng quan sát, các đối tượng có thể được hiển thị ở bất kỳ vị trí nào trên thiết bị xuất. Cũng như vậy, bằng việc thay đổi kích thước vùng quan sát, kích thước các phần của đối tượng có thể bị thay đổi. Khi các cửa sổ được đặt lại các kích thước khác được ánh xạ thành công vào một vùng quan sát, các hiệu ứng về **phóng to (zooming)** có thể thực hiện được.



Hình 4-3: Hiển thị đồng thời hai biểu đồ, dùng đa cửa sổ và sự xác định vùng quan sát.

Khi các cửa sổ được làm nhỏ hơn, người dùng có thể phóng to vài nơi trên ảnh để xem chi tiết hơn mà không cần phóng to toàn bộ cửa sổ. Các hiệu ứng **panning** có thể được tạo ra bằng cách di chuyển một cửa sổ có kích thước xác định ngang qua một hình ảnh lớn.

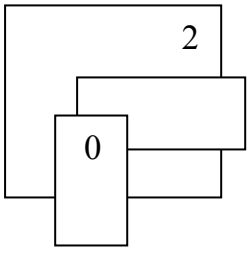
Một ví dụ của việc dùng đa cửa sổ và các lệnh về vùng quan sát được cho trong các thủ tục sau đây. Hai biểu đồ được hiển thị trên hai phần đều nhau của một thiết bị hiển thị (xem hình 4-3).

```
type
   points = array[1..max_points] of real;
   procedure two_graphs;
      var x,y : points; k: integer;
      begin
        set_window(0, 1, 0, 1);     {vẽ đường chia ở trung tâm}
        set_viewport(0, 1, 0, 1);
        x[1]:=0.5; y[1]:=0; x[2]:=0.5; y[2]:=1;
        polyline(2, x, y);
```

```
for k:=1 to 9 do begin      {đọc dữ liệu cho đồ thị thứ nhất}
    x[k]:=k;            {các giá trị dữ liệu từ 300 đến 700}
    readln(y[k]);
    end; {for k}
set_window(1, 9, 300, 700);
set_viewport(0.1, 0.4, 0.2, 0.8);{đặt vào phần bên trái màn hình}
polyline(9, x, y);

for k:=1 to 13 do begin    {đọc dữ liệu cho đồ thị thứ hai}
    x[k]:=k;
    readln(y[k]);
    end;
set_window(1, 13, 10, 100);      {các giá trị dữ liệu từ 10 đến 100}
set_viewport(0.6, 0.9, 0.2, 0.8);{đặt dữ liệu vào phần bên phải màn hình}
polyline(13, x, y);
end;{two graph}
```

Một phương pháp khác để xây dựng các vùng đa cửa sổ và vùng quan sát trong gói đồ họa là gán nhãn đến mỗi sự xác định. Điều này có thể được làm bằng việc thêm đối số thứ năm vào các lệnh về cửa sổ và vùng quan sát để xác định vùng chỉ định. Các tham số có thể là một chỉ số nguyên (0, 1, 2, 3, …). Các lệnh xuất sau đó dùng các chỉ số này để chỉ định sự chuyển đổi từ cửa sổ đến vùng quan sát nào. Cơ chế đánh số này cũng có thể được dùng để gắn kết một độ ưu tiên với mỗi vùng quan sát, đây là cơ sở để cài đặt tính chất nhìn thấy được của các cửa sổ nằm đè lên nhau. Các vùng quan sát được hiển thị theo độ ưu tiên được trình bày ở hình 4-4:

Hình 4-4: **Hiển thị các vùng quan sát theo thứ tự ưu tiên. Các vùng quan sát có số thứ tự nhỏ hơn sẽ có quyền ưu tiên cao hơn.**

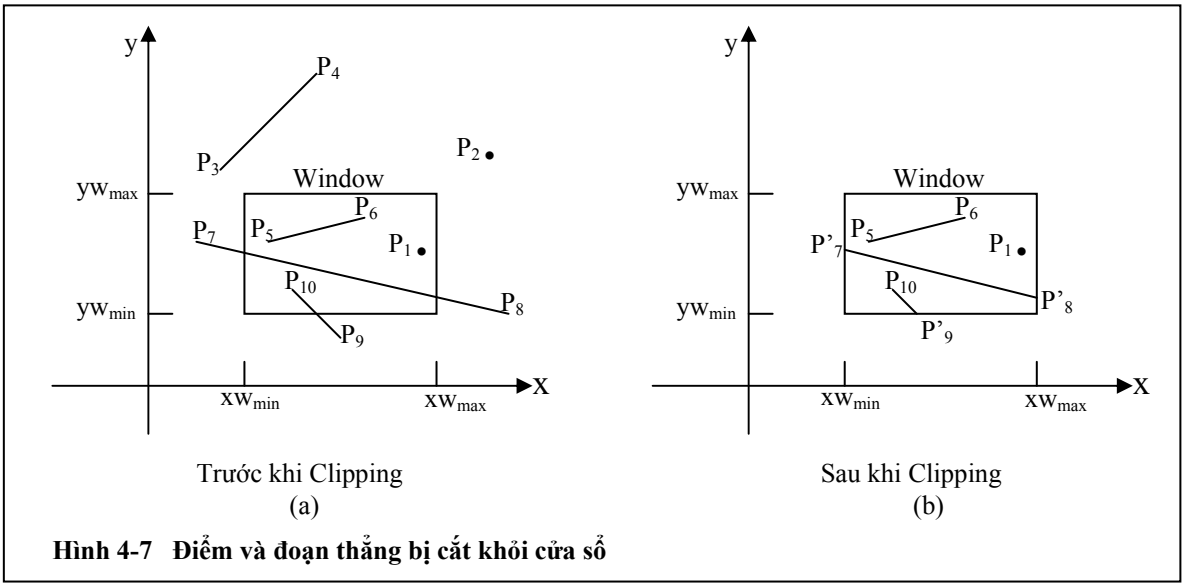


Việc cài đặt phép biến đổi cửa sổ thường được thực hiện bằng việc cắt (clipping) khỏi cửa sổ, sau đó ánh xạ phần bên trong cửa sổ vào một vùng quan sát (hình 6-6). Như một lựa chọn, một vài gói đồ họa đầu tiên ánh xạ sự định nghĩa trong hệ tọa độ thế giới thực vào trong hệ tọa độ thiết bị chuẩn và sau đó cắt khỏi biên vùng quan sát. Trong các các phần thảo luận sau, chúng ta giả thiết rằng việc cắt được thực hiện dựa vào đường biên cửa sổ trong hệ tọa độ thế giới thực. Sau khi cắt xong, các điểm bên trong cửa sổ mới được ánh xạ đến vùng quan sát.

Việc cắt các điểm khỏi cửa sổ được hiểu đơn giản là chúng ta kiểm tra các giá trị tọa độ để xác định xem chúng có nằm bên trong biên không. Một điểm ở vị trí (x,y) được giữ lại để chuyển đổi sang vùng quan sát nếu nó thỏa các bất phương trình sau:

$$xw_{min} \le x \le xw_{max}, yw_{min} \le y \le yw_{max} \qquad (4\text{-}1)$$

Nếu điểm nào không thỏa một trong bốn bất phương trình trên, nó bị cắt bỏ. Trong hình 4-7, điểm P_1 được giữ lại, trong khi điểm P_2 bị cắt bỏ.



Hình 4-7 Điểm và đoạn thẳng bị cắt khỏi cửa sổ

Hình 4-7 minh họa các quan hệ có thể có giữa các vị trí đoạn thẳng với biên cửa sổ. Chúng ta kiểm tra một đoạn thẳng xem có bị cắt hay không bằng việc xác định xem hai điểm đầu mút đoạn thẳng là nằm trong hay nằm ngoài cửa sổ. Một đoạn thẳng với cả hai đầu nằm trong cửa sổ thì được giữ lại hết, như đoạn từ P_5 đến P_6. Một đoạn với một đầu nằm ngoài (P_9) và một đầu nằm trong (P_{10}) sẽ bị cắt bớt tại giao điểm với biên cửa sổ (P'_9). Các đoạn thẳng có cả hai đầu đều nằm ngoài cửa sổ, có thể rơi vào hai trường hợp: toàn bộ đoạn thẳng đều nằm ngoài hoặc đoạn thẳng cắt hai cạnh cửa sổ.

Đoạn từ P_3 đến P_4 bị cắt bỏ hoàn toàn. Nhưng đoạn từ P_7 đến P_8 sẽ được giữ lại phần từ P'_7 đến P'_8.

Thuật toán clipping đường (line-clipping) xác định xem đoạn nào toàn bộ nằm trong, đoạn nào bị cắt bỏ hoàn toàn hay bị cắt một phần. Đối với các đoạn bị cắt bỏ một phần, các giao điểm với biên cửa sổ phải được tính. Vì một hình ảnh có thể chứa hàng ngàn đoạn thẳng, việc xử lý clipping nên được thực hiện sao cho có hiệu quả nhất. Trước khi đi tính các giao điểm, một thuật toán nên xác định rõ tất cả các đoạn thẳng được giữ lại hoàn toàn hoặc bị cắt bỏ hoàn toàn. Với những đoạn được xem xét là bị cắt bỏ, việc xác định các giao điểm cho phần được giữ lại nên được thực hiện với sự tính toán ít nhất.

Một tiếp cận để cắt các đoạn là dựa trên cơ chế đánh mã được phát triển bởi Cohen và Sutherland. Mọi điểm ở hai đầu mút đoạn thẳng trong hình ảnh sẽ được gán một mã nhị phân 4 bit, được gọi là **mã vùng (region code)**, giúp nhận ra vùng tọa độ của một điểm. Các vùng này được xây dựng dựa trên sự xem xét với biên cửa sổ, như ở hình 6-8. Mỗi vị trí bit trong mã vùng được dùng để chỉ ra một trong bốn vị trí tọa độ tương ứng của điểm so với cửa sổ: bên trái (left), phải (right), trên đỉnh (top), dưới đáy (bottom). Việc đánh số theo vị trí bit trong mã vùng từ 1 đến 4 cho từ phải sang trái, các vùng tọa độ có thể liên quan với vị trí bit như sau:

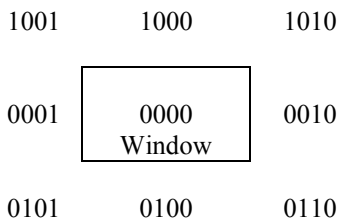
Bit 1 – left

Bit 2 – right

Bit 3 – below

Bit 4 – above

Giá trị 1 ở bất kỳ vị trí nào chỉ ra rằng điểm ở vị trí tương ứng, ngược lại bit ở vị trí đó là 0. Nếu một điểm nằm trong cửa sổ, mã vị trí là 0000. Một điểm bên dưới và bên trái cửa sổ có mã vùng là 0101 (xem hình 4-8).

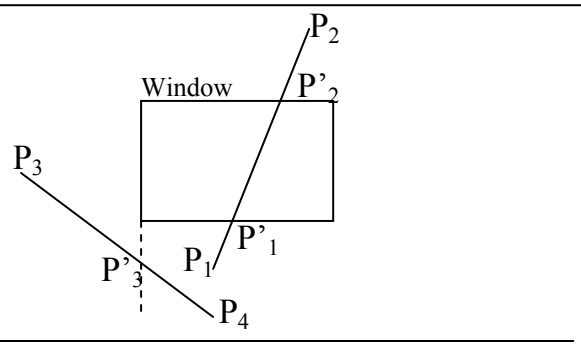


Hình 4-8
Các mã vùng nhị phân cho các điểm đầu mút đoạn thẳng, được dùng để định nghĩa các vùng tọa độ liên hệ với một cửa sổ.

Các giá trị bit trong mã vùng được xác định bằng cách so sánh giá trị tọa độ (x,y) của điểm đầu mút với biên cửa sổ. Bit 1 đặt lên 1 nếu $x < xw_{min}$. Các giá trị của ba bit còn lại được xác định bằng cách so sánh tương tự. Trong các ngôn ngữ lập trình, làm việc trên bit như thế này có thể thực hiện được, các giá trị bit mã vùng có thể được xác định theo các bước sau: (1) Tìm hiệu giữa tọa độ các điểm đầu mút với biên cửa sổ. (2) Dùng bit dấu (kết quả của mỗi hiệu) để đặt giá trị tương ứng trong mã vùng. Bit 1 là bit dấu của $x - xw_{min;}$ bit 2 là bit dấu của $xw_{max} – x$; bit 2 là bit dấu của $y - yw_{min}$; và bit 4 là bit dấu của $yw_{max} – y$.

Khi chúng ta xây dựng xong các mã vùng cho tất cả các điểm đầu mút, chúng ta có thể xác định nhanh chóng đoạn thẳng nào là hoàn toàn nằm trong cửa sổ, đoạn nào là hoàn toàn nằm ngoài. Bất kỳ đoạn nào có mã vùng của cả 2 đầu mút là 0000 thì nằm trong cửa sổ và chúng ta chấp nhận các đường này. Bất kỳ đường nào mà trong hai mã vùng của hai đầu mút có một số 1 ở cùng vị trí bit thì đoạn hoàn toàn nằm ngoài cửa sổ, và chúng ta loại bỏ các đoạn này. Ví dụ, chúng ta vứt bỏ đoạn có mã vùng ở một đầu là 1001, còn đầu kia là 0101 (có cùng bit 1 ở vị trí 1 nên cả hai đầu mút của đoạn này nằm ở phía bên trái cửa sổ). Một phương pháp có thể được dùng để kiểm tra các đoạn cho việc cắt toàn bộ là thực hiện phép logic *and* với cả hai mã vùng. Nếu kết quả không phải là 0000 thì đoạn nằm bên ngoài cửa sổ (xem hình 4-9).

Hình 4-9
Các đoạn từ một điểm này đến một điểm khác có thể cắt cửa sổ hoặc giao điểm với các biên nằm ngoài cửa sổ.



Các đường không được nhận dạng là hoàn toàn nằm trong hay hoàn toàn nằm ngoài một cửa sổ thông qua các phép kiểm tra trên sẽ được tìm giao điểm với biên cửa sổ. Như được chỉ ra ở hình 4-9, các đường thuộc nhóm này có thể cắt hoặc không cắt cửa sổ. Chúng ta có thể xử lý các đoạn này bằng cách so sánh một điểm đầu mút (cái đang nằm ngoài cửa sổ) với một biên cửa sổ để xác định phần nào của đường sẽ bị bỏ. Sau đó, phần đường được giữ lại sẽ được kiểm tra với các biên khác, và chúng ta tiếp tục cho đến khi toàn bộ đường bị bỏ đi hay đến khi một phần đường được xác định là

nằm trong cửa sổ. Chúng ta xây dựng thuật toán để kiểm tra các điểm đầu mút tương tác với biên cửa sổ là ở bên trái, bên phải, bên dưới hay trên đỉnh.

Để minh họa các bước xác định trong việc cắt các đoạn khỏi biên cửa sổ dùng thuật toán của Cohen-Sutherland, chúng ta xem các đoạn trong hình 4-9 được xử lý như thế nào. Bắt đầu ở điểm đầu mút bên dưới từ P_1 đến P_2, ta kiểm tra P_1 với biên trái, phải và đáy cửa sổ và thấy rằng điểm này nằm phía dưới cửa sổ. Ta tìm giao điểm P'_1 với biên dưới. Sau khi tìm giao điểm P'_1, chúng ta vứt bỏ đoạn từ P_1 đến P'_1. Tương tự, vì P_2 bên ngoài cửa sổ, chúng ta kiểm tra và thấy rằng điểm này nằm phía trên cửa sổ. Giao điểm P'_2 được tính, và đoạn từ P'_1 đến P'_2 được giữ lại. Kết thúc quá trình xử lý đoạn P_1P_2. Bây giờ xét đoạn kế tiếp, P_3P_4. Điểm P_3 nằm bên trái cửa sổ, vì vậy ta xác định giao điểm P'_3 và loại bỏ đoạn từ P'_3 đến P_3. Bằng cách kiểm tra mã vùng phần đoạn thẳng từ P'_3 đến P4, chúng ta thấy rằng phần còn lại này nằm phía dưới cửa sổ và cũng bị vứt bỏ luôn.

Các giao điểm với biên cửa sổ có thể được tính bằng cách dùng các tham số của phương trình đường thẳng. Với một đường thẳng đi qua hai điểm (x_1, y_1) và (x_2, y_2), tung độ y của giao điểm với một biên dọc cửa sổ có thể tính được theo phép tính:

$$y = y_1 + m\,(x - x_1) \qquad (4\text{-}2)$$

Ở đây giá trị x được đặt là xw_{min} hoặc xw_{max}, và độ dốc m được tính bằng là

$$m = (y_2 - y_1)/(x_2 - x_1)$$

Tương tự, nếu ta tìm giao điểm với biên ngang, hoành độ x có thể được tính như sau:

$$x = x_1 + (y - y_1)/m \qquad (4\text{-}3)$$

với y là yw_{min} hoặc yw_{max}.

Thủ tục sau đây minh họa thuật toán clipping đường (line-clipping) của Cohen-Sutherland. Các mã cho mỗi điểm đầu mút được chứa trong các mảng Boolean bốn phần tử.

```
var
   xw_min, xw_max, yw_min, yw_max: real;
procedure clip_a_line (x1, y1, x2, y2: real);
   type
```

```
        boundaries = (left, right, bottom, top);
        code = array [boundaries] of boolean;
    var
        code1, code2 : code;
        done, display: boolean;
        m: real;
procedure encode (x, y : real; var c: code);
    begin
        if x < xw_min then c[left]:= true
          else c[left]:= false;
        if x > xw_max then c[right]:= true
          else c[right]:= false;
        if y < yw_min then c[bottom]:= true
          else c[bottom]:= false;
        if y > yw_max then c[top]:= true
          else c[top]:= false
    end; {encode}

function accept (c1, c2 : code) : boolean;
    var k : boundaries;
    begin
        {nếu điểm có trị "true" ở bất kỳ vị trí nào trong mã của nó,
          một chấp nhận bình thường là không thể}
        accept :=true;
        for k:= left to top do
                if c1[k] or c2[k] then accept :=false
    end; {accept}
function reject (c1, c2 : code) : boolean;
    var k : boundaries;
    begin
        {nếu hai điểm đầu mút có trị 'true' ở cùng vị trí tương ứng,
        đoạn thẳng bị xóa bỏ}
```

```
        reject:=false;
        for  k:= left to top do
            if c1[k] and c2[k] then reject :=true
    end; {reject}


procedure swap_if_needed (var x1, y1, x2, y2: real;
                                  var c1, c2: code);
    begin
        {đảm bảo rằng x1, y1 là điểm nằm ngoài cửa sổ và c1 chứa mã đó}
    end; {swap_if_needed}


begin
        done :=false;
        display :=false;
        while not done do begin
                encode (x1, y1, code1);
                encode (x2, y2, code2);
                if accept (code1, code2) then begin
                        done :=true;
                        display :=true;
                end {if accept}
                else
                  if reject (code1, code2) then done :=true
                  else begin {tìm giao điểm}
                        {bảo đảm rằng  x1, y1 nằm ngoài cửa sổ}
                        swap_if_needed (x1, y1, x2, y2, code1, code2);
                        m := (y2-y1) / (x2-x1);
                        if code1[left] then begin
                                y1 := y1 + (xw_min – x1) * m;
                                x1 :=xw_min
                          end {cắt biên phải}
                        else
                           if code1[right] then begin
```

```
                    y1 := y1 + (xw_max – x1)*m;
                    x1 := xw_max
                  end {cắt biên trái}
                else
                  if code1[bottom] then begin
                    x1 := x1 + (yw_min – y1) / m;
                    y1 := yw_min
                  end {cắt biên dưới đáy}
                else
                  if code1[top] then begin
                    x1 := x1 + (yw_max – y1) / m;
                    y1 := yw_max
                  end {cắt biên đỉnh}
                end {ngược lại tìm giao điểm}
        end; {while not done}
        if display then {draw x1, y1, to x2, y2}
end; {clip_a_line}
```

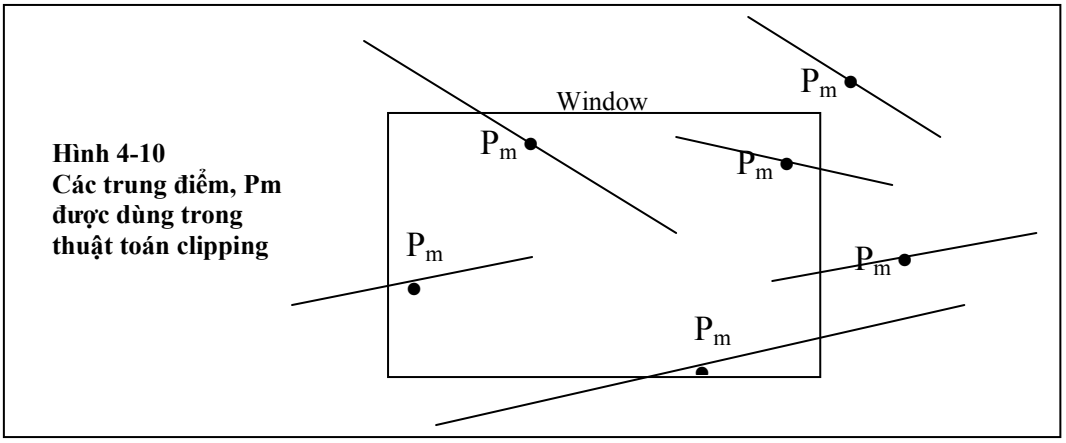
Một kỹ thuật để xác định giao điểm với biên cửa sổ mà không dùng đến phương trình đường thẳng là dùng thủ tục tìm kiếm nhị phân, được gọi là sự phân chia tại trung điểm. Đầu tiên, việc kiểm tra các đoạn một lần nữa được thực hiện bằng cách dùng mã vùng. Bất kỳ đoạn nào không được chấp nhận hoàn toàn hoặc không bị huỷ bỏ hoàn toàn (nhờ vào kiểm tra mã vùng) thì sẽ được đi tìm giao điểm bằng cách kiểm tra tọa độ trung điểm.

Tiếp cận này được minh họa trong hình 4-10 (xem hình 4-10). Mọi đoạn thẳng với hai điểm đầu mút (x_1,y_1) và (x_2, y_2), trung điểm được tính như sau:

$$x_m = (x_1 + x_2) / 2; \quad y_m = (y_1 + y_2) / 2 \qquad (4\text{-}4)$$

Mỗi kết quả tính toán cho tọa độ giao điểm liên quan đến một phép cộng và một phép chia 2. Khi tọa độ giao điểm được xác định, mỗi nữa đoạn thẳng được kiểm tra để chấp nhận hay huỷ bỏ toàn bộ. Nếu một nữa đoạn được chấp nhận hoặc bị huỷ bỏ, một nữa kia sau đó sẽ được xử lý theo cách tương tự. Điều này tiếp tục cho đến khi gặp một giao điểm. Nếu một nữa được chấp nhận hoặc bị huỷ bỏ toàn bộ, nữa kia tiếp

tục được xử lý cho đến khi toàn bộ nó là bị huỷ bỏ hoặc được giữ lại. Cài đặt phần cứng theo phương pháp này có thể giúp ta clipping khỏi biên vùng quan sát nhanh chóng sau khi các đối tượng vừa được chuyển sang hệ tọa độ thiết bị.



Hình 4-10
Các trung điểm, Pm được dùng trong thuật toán clipping

Các kỹ thuật khác cho việc clipping đoạn dùng phương trình tham số của đường thẳng. Chúng ta có thể viết phương trình đường thẳng qua 2 điểm (x_1, y_1) và (x_2, y_2) theo hình thức tham số:

$$x = x_1 + (x_2 - x_1)u = x_1 + \Delta x\, u \qquad (4\text{-}5)$$
$$y = y_1 + (y_2 - y_1)u = y_1 + \Delta y\, u$$

Với $\Delta x = x_2 - x_1$ và $\Delta y = y_2 - y_1$. Tham số u được gán các giá trị từ 0 đến 1, và các tọa độ (x,y) là tọa độ các điểm trên đường ứng với các giá trị cụ thể của u trong đoạn [0,1]. Khi u = 0, $(x, y) = (x_1, y_1)$. Ở đầu kia của đoạn, u = 1 và $(x, y) = (x_2, y_2)$.

Một thuật toán clipping đường hiệu quả dùng phương trình tham số đã được phát triển bởi Liang và Barsky. Họ ghi chú rằng nếu một điểm (x, y) dọc theo đường mà nằm trong cửa sổ được định nghĩa bởi các tọa độ (xw_{min}, yw_{min}) và (xw_{max}, yw_{max}), thì các điều kiện sau đây phải được thỏa:

$$xw_{min} \le x_1 + \Delta x\, u \le xw_{max} \qquad (4\text{-}6)$$
$$yw_{min} \le y_1 + \Delta y\, u \le yw_{max}$$

Bốn bất phương trình trên có thể được viết lại theo hình thức sau:

$$p_k u \le q_k,\ k = 1, 2, 3, 4 \qquad (4\text{-}7)$$

ở đây p và q được định nghĩa như sau:

$$p_1 = -\Delta x, \quad q_1 = x_1 - xw_{min}$$
$$p_2 = -\Delta x, \quad q_2 = xw_{max} - x_1 \qquad (4\text{-}8)$$

$p_3 = -\Delta y, \quad q_3 = y_1 - yw_{min}$

$p_4 = \Delta y, \quad q_4 = yw_{max} - y_1$

Bất kỳ đoạn thẳng nào song song với một trong các biên cửa sổ sẽ có $p_k = 0$, giá trị k phụ thuộc vào biên cửa sổ (k = 1, 2, 3, và 4 tương ứng với biên trái, phải, dưới, trên). Nếu với các giá trị đó của k, chúng ta có thể gặp $q_k < 0$, khi đó đoạn thẳng sẽ hoàn toàn nằm ngoài biên và có thể bị loại bỏ khi xét sau này. Nếu $q_k \geq 0$, đường thẳng tương ứng nằm trong biên.

Khi $p_k < 0$, sự kéo dài không giới hạn của đoạn thẳng từ bên ngoài vào bên trong của biên cửa sổ kéo dài. Nếu $p_k > 0$, đoạn thẳng tiến từ bên trong ra bên ngoài. Với p_k khác 0, chúng ta có thể tính giá trị của u tương ứng với điểm mà tại đó đoạn thẳng kéo dài cắt biên k kéo dài của cửa sổ:

$$u = q_k/p_k \qquad (4\text{-}9)$$

Đối với mỗi đoạn thẳng, chúng ta có thể tính các giá trị cho các tham số u_1 và u_2 để xác định phần nào của đoạn nằm bên trong cửa sổ. Giá trị của u_1 được xác định bằng cách nhìn ở các cạnh của cửa sổ xem đoạn kéo dài nào từ ngoài vào trong ($p<0$). Đối với các cạnh cửa sổ, chúng ta tính $r_k = q_k/ p_k$. Giá trị của u_1 là lớn nhất trong tập chứa 0 và các giá trị khác của r. Ngược lại, giá trị của u_2 được xác định bằng cách kiểm tra các biên xem đoạn nào kéo dài nào từ bên trong ra bên ngoài ($p>0$). Một giá trị của r_k được tính cho mỗi biên cửa sổ, và giá trị của u_2 là nhỏ nhất trong tập chứa 1 và các giá trị đã được tính của r.
Nếu $u_1 > u_2$, đoạn hoàn toàn nằm ngoài cửa sổ và có thể bị vứt bỏ. Ngược lại, các điểm đầu mút của đoạn bị cắt được tính từ hai giá trị của tham số u.

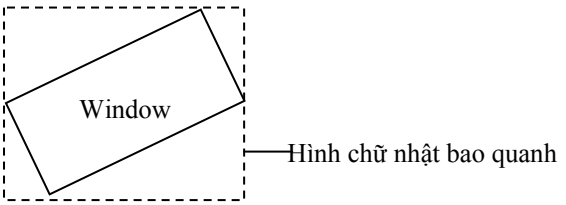
Thuật toán này được trình bày trong thủ tục sau đây. Các tham số giao điểm của đoạn được khởi tạo các giá trị $u_1 = 0$ và $u_2 = 1$. Đối với mỗi biên cửa sổ, các giá trị thích hợp cho p và q được tính và được dùng bởi hàm *cliptest* để xác định xem đoạn nào có thể bị loại bỏ hoặc xem các tham số giao điểm sắp sửa bị thay đổi không. Khi $p < 0$, tham số r được dùng để cập nhật u_1; khi $p>0$, tham số r được dùng để cập nhật u_2. Nếu việc cập nhật u_1 hoặc u_2 đưa đến kết quả $u_1 > u_2$, chúng ta loại bỏ đoạn thẳng. Ngược lại, chúng ta cập nhật tham số u thích hợp chỉ nếu giá trị mới đưa đến kết quả làm ngắn đoạn thẳng. Khi $p=0$ và $q<0$, chúng ta vứt bỏ đoạn thẳng bởi vì nó song song và ở bên ngoài biên. Nếu đoạn thẳng vẫn chưa bị loại bỏ sau tất cả bốn giá trị của p và

q vừa được kiểm tra xong, các điểm đầu mút của đoạn bị cắt được xác định từ các giá trị của u_1 và u_2.

```
var
    xwmin, xwmax, ywmin, ywmax : real;
procedure clipper (var x1, y1, x2, y2 : real);
    var
        u1, u2, dx, dy : real;
    function cliptest (p, q : real; var u1, u2 : real);
        var
            r : real;
            result : boolean;
        begin
            result := true;
            if  p < 0 then begin              {đoạn từ bên ngoài vào bên trong biên }
                r := q / p;
                if  r > u2 then  result := false
                {huỷ bỏ đoạn hoặc cập nhật u1 nếu thích hợp}
                else if  r > u1 then u1 :=r
              end {if p < 0}
            else
                if p > 0 then begin   {đoạn từ bên trong ra bên ngoài của biên}
                    r := q / p;
                    if  r < u1 then  result := false
                    else if  r < u2  then u2 := r
                  end {if p > 0}
                else
                    if  q < 0 then result := fasle;
            cliptest := result
        end; {cliptest}
        begin {clipper}
            u1 := 0;
            u2 := 1;
```

```
dx := x2 – x1;
if  cliptest (-dx, x1 – xwmin, u1, u2) then
  if  cliptest (dx, xwmax – x1, u1, u2) then begin
    dy := y2 - y1;
    if cliptest (-dy, y1 – ywmin, u1, u2) then
       if cliptest(dy, ywmax – y1, u1, u2) then begin
         {nếu u1 và u2 nằm  trong đoạn [0,1],
          dùng để tính các điểm đầu mút mới}
         if u2 < 1 then begin
           x2 := x1 + u2 * dx;
           y2 := y1 + u2 * dy
         end; {if u2 < 1}
         if u1 > 0 then begin
           x1 := x1 + u1 * dx;
           y1 := y1 + u1 * dy
         end; {if u1 >  0}
       end {if cliptest}
  end {if cliptest}
end; {clipper}
```

Thuật toán clipping đường của Liang và Barsky giảm bớt các tính toán cần thiết để cắt các đoạn. Mỗi lần cập nhật u1 và u2 cần chỉ một phép chia, và các giao điểm với cửa sổ được tính chỉ một lần, khi mà các giá trị u1 và u2 vừa hoàn thành. Trái lại, thuật toán của Cohen và Sutherland lặp lại việc tính giao điểm của đoạn với các biên cửa sổ, và mỗi phép tính giao điểm cần cả hai phép chia và nhân (xem hình 4-11).



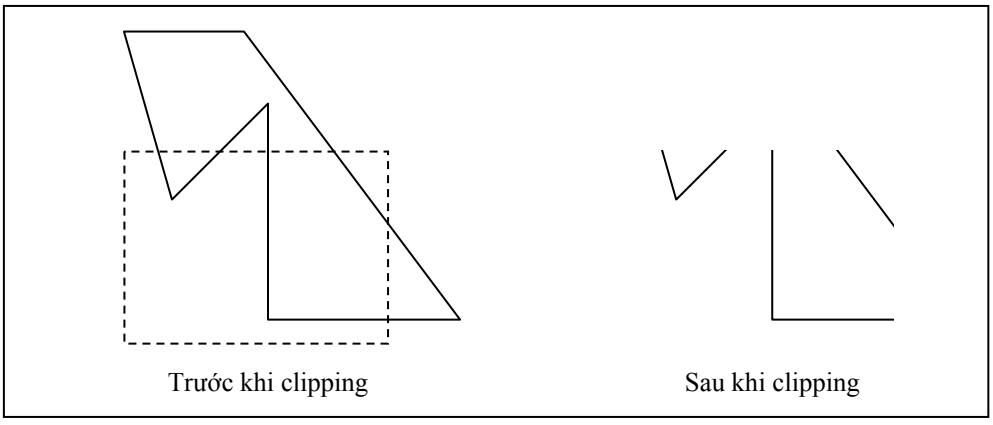
Hình 4-11
Cửa sổ bị quay được bao quanh bởi một biên chữ nhật lớn hơn (có các cạnh song song với hệ trục tọa độ)

Khi các cửa sổ bị quay hay các đa giác có hình dạng bất kỳ (được dùng làm cửa sổ và vùng quan sát), các thuật toán clipping đã được thảo luận sẽ cần vài sự thay đổi. Nó vẫn có thể được dùng để che chắn các đoạn thẳng. Một cửa sổ bị quay, hoặc một đa giác bất kỳ nào khác, có thể bị bao quanh trong một hình chữ nhật lớn hơn (hình chữ nhật này có các trục song song với các trục tọa độ) (hình 4 -11). Bất kỳ đoạn thẳng nào nằm bên ngoài hình chữ nhật bao quanh lớn hơn (bounding rectangle) thì cũng nằm bên ngoài cửa sổ (window). Các kiểm tra nằm trong cũng không dễ dàng, và các giao điểm phải được tính dùng phương trình đường thẳng của các biên cửa sổ và của các đoạn thẳng bị cắt.

Clipping một vùng (Area clipping)

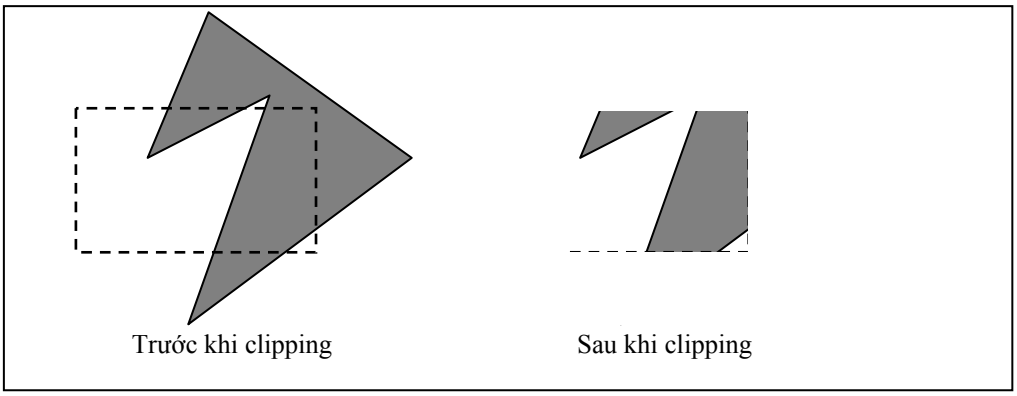
Làm thế nào các đa giác được dùng trong các ứng dụng vẽ đường (line-drawing application) có thể bị cắt bằng cách xử lý các đoạn thẳng thành phần thông qua các thuật toán clipping đường đã được thảo luận. Một đa giác được xử lý theo cách này sẽ được thu giảm một loạt các đoạn sẽ bị cắt (xem hình 4-12).

Hình 4-12: Đa giác bị cắt bởi một thuật toán clipping đường.



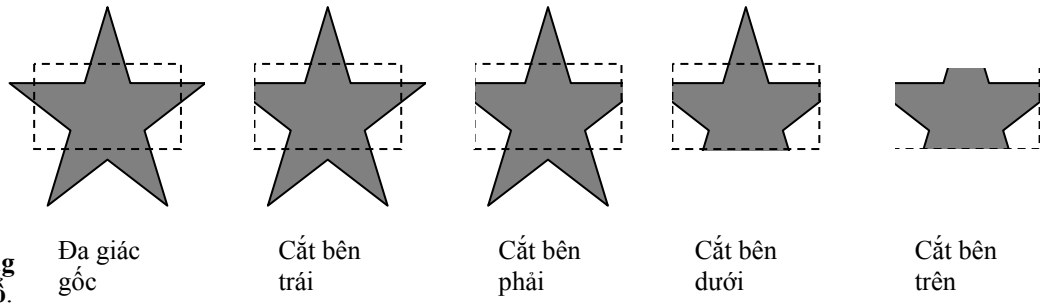
Khi một một biên đa giác định nghĩa một vùng tô, như ở hình 4-13. Một version thay đổi của thuật toán clipping đường được cần đến. Trong trường hợp này, một hoặc nhiều vùng kép kín phải được tạo ra để định nghĩa các biên cho vùng tô (xem hình 4-13).

Hình 4 –13: Một vùng có hình dạng, trước và sau khi clipping.

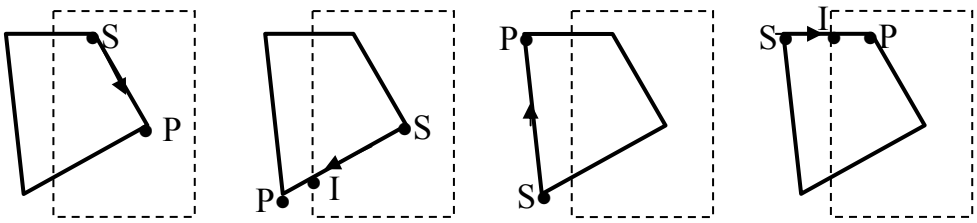


Một kỹ thuật cho việc clipping đa giác, được phát triển bởi Sutherland và Hodgman, thực hiện việc clipping bằng cách so sánh một đa giác với lần lượt mỗi biên cửa sổ. Kết quả trả về của thuật toán là một tập các đỉnh định nghĩa vùng bị cắt (vùng này được tô với một màu hay một mẫu tô nào đó). Phương pháp căn bản được thể hiện trong hình 4-14.

Các vùng đa giác được định nghĩa bằng việc xác định một dãy có thứ tự các đỉnh. Để cắt một đa giác, chúng ta so sánh lần lượt mỗi đỉnh với biên một cửa sổ. Các đỉnh nằm bên trong cạnh cửa sổ này được giữ lại cho việc clipping với biên kế tiếp của cửa sổ (xem hình 4-15).

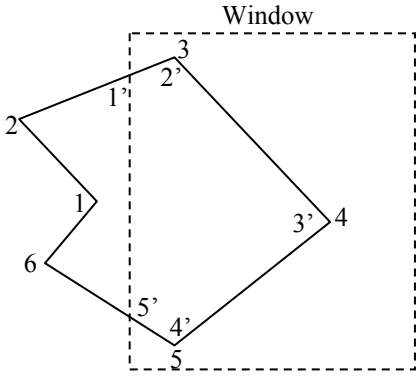


Hình 4-14
Clipping một vùng đa giác bằng cách dùng các biên cửa sổ.



Hình 4-15 clipping.

Hình 4-16
Clipping một đa giác khỏi cạnh bên trái cửa sổ, bắt đầu với đỉnh 1. Các số có phẩy được dùng để đánh nhãn các điểm được lưu bởi thuật toán clipping.



Quá trình xử lí các đỉnh của một dâ giác liên quan đến biên của cửa sổ. Từ đỉnh S, đỉnh kế tiếp được xét (P) có thể sinh ra một điểm, không điểm nào, hoặc hai điểm sẽ được lưu bởi thuật toán các đỉnh bên ngoài cạnh cửa sổ bị vứt bỏ. Nếu chúng ta khởi hành từ một điểm bên trong cạnh cửa sổ đi đến một điểm bên ngoài, chúng ta lưu lại giao điểm của đoạn thẳng với biên cửa sổ. Cả hai giao điểm và đỉnh đa giác được lưu lại nếu chúng ta đi từ ngoài cạnh cửa sổ vào bên trong. Khả năng thứ tư có thể xảy ra khi chúng ta xử lí một điểm (P) và điểm trước đó (S) với biên cửa sổ được minh họa trong hình 4-15. Một điểm bên trong biên cửa sổ được lưu lại (trường hợp a), trong khi một điểm bên ngoài thì không (trường hợp c). Nếu một điểm P và điểm trước đó S nằm trên các phía đối diện nhau qua một biên (P ở trong, S ở ngoài và ngược lại), giao điểm I được tính và được lưu (trường hợp b và d). Trong trường hợp d, điểm P nằm trong và điểm trước đó S nằm ngoài, vì vậy cả hai giao điểm I và P được lưu. Khi tất cả các đỉnh vừa được xử lí với biên trái cửa sổ, tập các điểm được lưu sẽ tiếp tục bị cắt khi xem xét với biên kế tiếp của cửa sổ.

Chúng ta minh họa phương pháp này bằng việc xử lí vùng trong hình 4-16 khi xem xét với biên bên trái của cửa sổ. Đỉnh 1 và 2 được xác định là nằm bên ngoài của biên. Đi qua đến đỉnh 3, đang nằm bên trong, chúng ta tính giao điểm và lưu lại cả hai giao điểm và đỉnh 3. Đỉnh 4 và 5 được xác định là nằm trong, và chúng nó cũng được lưu lại. Đỉnh thứ sáu và đỉnh cuối cùng thì nằm ngoài, vì vậy chúng ta tính và lưu giao điểm. Dùng năm điểm vừa được lưu, chúng ta lặp lại quá trình này khi xem xét với biên kế tiếp của cửa sổ.

Cài đặt các thuật toán vừa được mô tả đòi hỏi phải dùng không gian lưu trữ ngoài để lưu các điểm. Điều có thể tránh được nếu chúng ta quản lý được mỗi điểm (điểm sắp sửa được lưu và đi nhanh qua nó để kiểm tra tiếp), cùng với các lệnh (instructions) để cắt nó khỏi biên kế tiếp của cửa sổ. Chúng ta lưu một điểm (dù là một đỉnh nguyên thuỷ của đa giác hay một đỉnh có được khi tính giao điểm) chỉ sau khi nó được xử lí khi xem xét với tất cả các biên. Như thể chúng ta có một đường ống chứa

một chuỗi các động tác clipping. Một điểm nằm bên trong hay nằm trên biên cửa sổ ở một giai đoạn sẽ được đi qua để đến giai đoạn kế tiếp.

Thủ tục sau đây thể hiện tiếp cận này . Một mảng s, lưu những điểm mới nhất vừa bị cắt cho với mỗi biên của cửa sổ. Quá trình chính đi qua mỗi đỉnh p đi vào quá trình *clip_this* để xem xét việc cắt với cạnh đầu tiên của cửa sổ . Nếu đoạn thẳng được định nghĩa bởi điểm đầu mút *p* và *s[edge]* cắt cạnh cửa sổ này, giao điểm được xác định và được đi qua để đến giai đoạn kế tiếp. Nếu p nằm bên trong cửa sổ, nó bị bỏ qua để đến giai đoạn clipping kế tiếp. Bất kì điểm nào còn được giữ lại sau khi xem xét với tất cả các cạnh của cửa sổ thì sau đó được gia nhập vào mảng kết quả kết xuất *x_out* và *y_out*. Mảng *first_point* lưu giữ cho mỗi cạnh cửa sổ điểm đầu tiên bị cắt bởi cạnh đó. Sau khi tất cả các đỉnh của đa giác vừa được xem xét xong, một quá trình kết thúc cắt các đoạn (đoạn đã được định nghĩa bởi các điểm đầu và cuối (các điểm bị cắt khỏi mỗi mỗi cạnh)).

```
type
    point = array [1..max_points] of real;
procedure polygon_clip (n : integer; x, y : points; var m : integer;
                                var x_out, y_out : points);
    const
        boundary_count = 4;
    type
        vertex  =  array [1..2] of real;
        boundary_range  = 1..boundary_count;
     var
        k :  integer;
        p :  vertex;
        s, first_point : array [1..boudary_count] of vertex;
        new_edge  :  array [1..boundary_count] of boolean;

    function inside (p : vertex; edge : boundary_range) : boolean;
        begin
{trả về true nếu đỉnh p nằm trong cạnh edge cửa sổ}
        end; { inside}
```

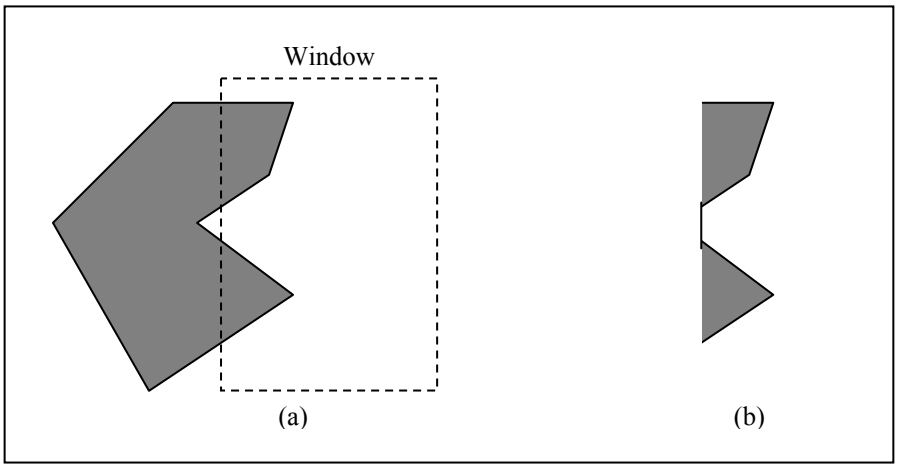
```
    function cross (p, s : vertex;  edge : integer) : boolean;
        begin
{trả về true nếu cạnh đa giác ps cắt biên cửa sổ}
        end; {cross}

    procedure output_vertex (p : vertex);
        begin
            m := m +1;
            x_out[m] := p[1]; y_out[m] := p[2];
        end; { output_vertex }

    procedure find_intersection (p, s : vertex;
                                 edge : boundary_range; var i;  vertex);
        begin
{trả về trong tham số i giao điểm của ps với biên edge cửa sổ }
        end; { intersection }

procedure clip_this (p : vertex; edge : boundary_range);
        var i : vertex;
        begin{ clip_this }
            {lưu điểm đầu tiên cắt biên cửa sổ}
            if new_edge[edge] then begin
                first_point[edge] := p;
                new_edge[edge] := false
            end  {new_edge}
            else
                {nếu ps cắt biên cửa sổ, tìm giao điểm,
                 cắt giao điểm khỏi cạnh kế tiếp của cửa sổ}
                if cross (p, s[edge], edge) then begin
                    find_intersection (p, s[edge], edge , i);
                    if edge < boundary_count then clip_this (i, edge +1)
                    else output_vertex (i)
```

```
                end; {nếu ps cắt cạnh}
            {cập nhật các đỉnh đã được lưu}
            s[edge] := p;
            {nếu p nằm bên trong cạnh cửa sổ này,
              cắt nó khỏi cạnh kế tiếp của cửa sổ}
            if inside (p, edge) then
                if edge  < boundary_count  then clip_this (p, edge +1)
                else output_vertex (p)
    end; {clip_this}
procedure clip_closer;
  {đóng quá trình. Đối với mỗi cạnh của cửa sổ,
    cắt đường (đang nối với đỉnh được lưu sau cùng và điểm first_point
    bị xử lý khỏi cạnh)}
var
    i : vertex;
    edge : integer;
begin
    for edge := 1 to boundary_count do
        if cross (s[edge], first_point[edge], edge) then begin
            find_intersection (s[edge], first_point[edge], edge, i);
            if edge < boundary_count then clip_this (i, edge +1)
            else output_vertex (i)
            end {nếu s và first_point cắt cạnh}
end; {clip_closer}

begin {polygon_clip}
    m :=0;                          {số các đỉnh kết xuất}
    for k := 1 to boundary_count do
        new_edge[k] := true;
    for  k:= 1 to  n do begin       {đặt mỗi đỉnh vào đường ống (pipeline)}
        p[1] := x[k]; p[2] := y[k];
        clip_this (p, 1)            {cắt khỏi cạnh đầu tiên của cửa sổ}
    end; {for k}
```

```
    clip_closer                    {đóng đa giác}
end; { polygon_clip }
```

Khi một đa giác lõm bị cắt bởi một cửa sổ hình chữ nhật, vùng bị cắt sau cùng có thể hình thành hai đa giác riêng biệt thật sự. Vì thuật toán cắt vùng này chỉ tạo ra một danh sách các đỉnh, các vùng riêng biệt này sẽ được nối lại bằng các đoạn thẳng nối. Một ví dụ của hiệu ứng này được thể hiện trong hình 4-17. Sự xem xét đặt biệt có thể được thực hiện đối với trường hợp như thế để gỡ bỏ các đoạn nối dư thừa, hoặc các thuật toán clipping tổng quát hơn sẽ được phát triển (xem hình 4-17).

Hình 4-17: Clipping đa giác lõm trong hình (a) bởi một cửa sổ tạo ra hai vùng nối nhau trong hình (b)



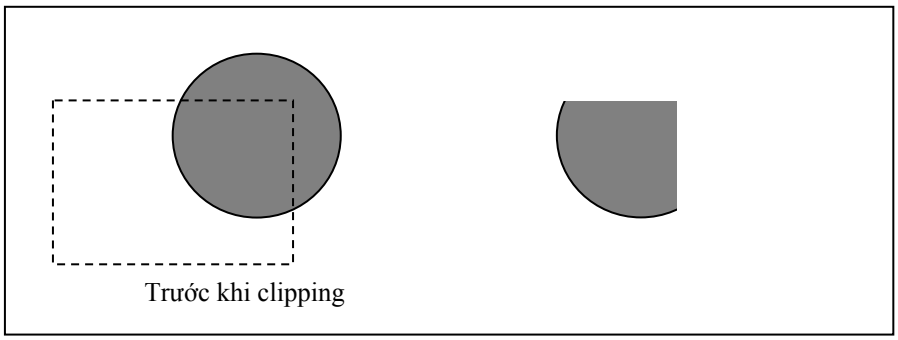
Dù chúng ta đã và đang giới hạn việc thảo luận của chúng ta đối với các cửa sổ chữ nhật có các cạnh song song với trục x và trục y., chúng ta có thể cài đặt thuật toán này với cửa sổ có hình đa giác bất kì. Chúng ta có thể cần lưu trữ thông tin về mỗi biên cửa sổ, và chúng ta có thể cần thay đổi thủ tục ***inside*** và ***find_intersection*** để quản lý thuộc tính của các biên tuỳ ý.

Một tiếp cận khác để clipping các vùng đa giác là dùng các phương pháp phương trình tham số. Các cửa sổ hình dạng tuỳ ý sau đó có thể được xử lí bằng cách dùng phương trình tham số của đường thẳng để mô tả cả hai: biên cửa sổ và các biên của vùng bị cắt.

Các vùng bị clipping hình dạng khác đa giác cần thực hiện nhiều công việc hơn một chút, vì biên của các vùng này không được định nghĩa bằng các phương trình

đường thẳng. Ví dụ, trong hình 4-18, phương trình đường tròn được cần để tìm hai giao điểm trên biên cửa sổ.

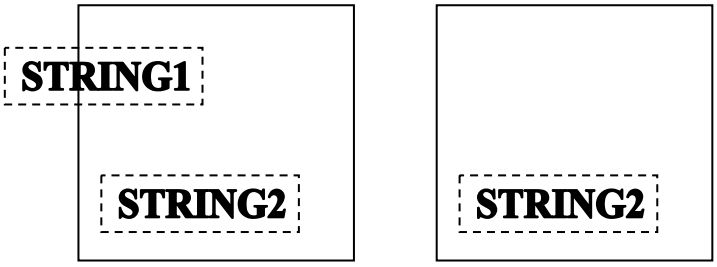
Hình 4-18: Clipping một vùng có hình dạng tròn.



Clipping văn bản (Text Clipping)

Có vài kỹ thuật có thể được dùng để clipping văn bản trong gói đồ họa. Việc chọn lựa phương pháp cụ thể để cài đặt phụ thuộc vào các phương pháp đã được dùng để sinh ra các kí tự và mức độ tinh vi được đòi hỏi bởi người dùng trong việc xử lí văn bản (xem hình 4-19).

Hình 4-19
Clipping văn bản dùng các biên chữ nhật. Bất kỳ hình chữ nhật nào mà nằm đè lên biên cửa sổ đều bị vứt bỏ hoàn toàn.

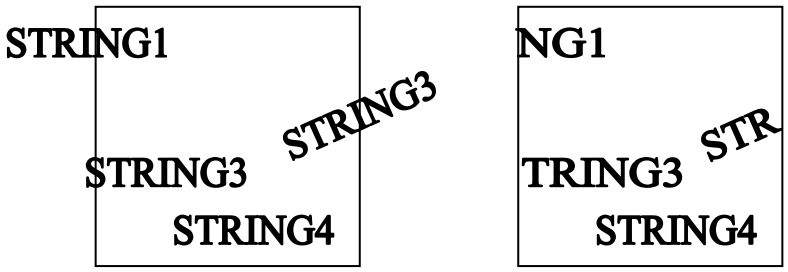


Phương pháp đơn giản nhất để xử lí các chuỗi kí tự có liên quan đến một biên cửa sổ là dùng chiến lượt "clipping tất cả văn bản hoặc không clipping gì cả" (all-or-none text-clipping), được trình bày trong hình 6-19. Nếu tất cả chuỗi kí tự nằm bên trong một cửa sổ, chúng ta giữ lại nó. Ngược lại, chuỗi vứt bỏ. Thủ tục này có thể được cài đặt bằng việc xem xét một hình chữ nhật bao quanh mẫu văn bản. Các vị trí biên của hình chữ nhật sau đó được so sánh với các biên cửa sổ, và chuỗi bị huỷ bỏ nếu có bất kì sự nằm đè nào. Phương pháp này cho ta clipping nhanh nhất.

Một sự chọn lựa để loại bỏ toàn bộ chuỗi kí tự nếu nó nằm đè lên biên một cửa sổ là dùng chiến lược "**clipping kí tự toàn bộ hoặc không**" (all-or-none character-clipping). Ở đây chúng ta vứt bỏ chỉ những kí tự nào không hoàn toàn nằm trong cửa

sổ (xem hình 4-20). Trong trường hợp này, các giới hạn biên của các kí tự đơn lẻ được so sánh với cửa sổ. Bất kì kí tự nào hoặc nằm đè lên hoặc nằm bên ngoài biên cửa sổ đều bị cắt bỏ.

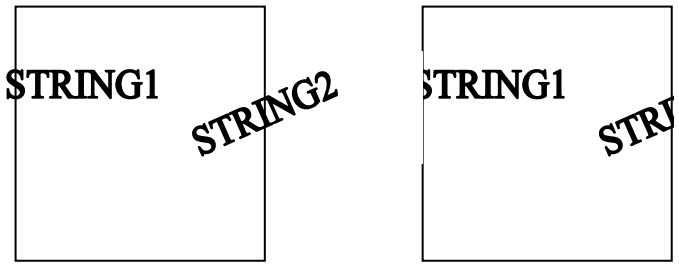
Hình 4-20
Các chuỗi kí tự có thể hoàn toàn bị cắt để mà chỉ những kí tự hoàn nằm bên trong cửa sổ mới được giữ lại.



Trước khi clipping Sau khi clipping

Phương pháp sau cùng cho việc quản lí việc cắt văn bản là cắt các kí tự riêng lẻ. Bây giờ chúng ta xem các kí tự cũng tương tự như các đoạn thẳng. Nếu một kí tự riêng lẻ nằm đè lên biên cửa sổ, chúng ta cắt bỏ phần nằm ngoài cửa sổ (xem hình 4-21). Các kí tự được hình thành với các đoạn thẳng có thể được xử lí theo cách này, bằng cách dùng thuật toán clipping đường. Việc xử lí các kí tự được hình thành bởi các bản đồ bit cần clipping những pixel đơn lẻ bằng cách so sánh các vị trí liên hệ của các mẫu lưới (patern grid) với các biên cửa sổ.

Hình 4-21
Clipping các kí tự đơn lẻ.



Trước khi clipping Sau khi clipping

Tẩy xoá (banking)

Thay vì lưu giữ lại thông tin trong một vùng được định nghĩa,, một vùng cửa sổ có thể được dùng để xóa bỏ bất kì thứ gì bên trong biên của nó. Những gì nằm bên ngoài được giữ lại.

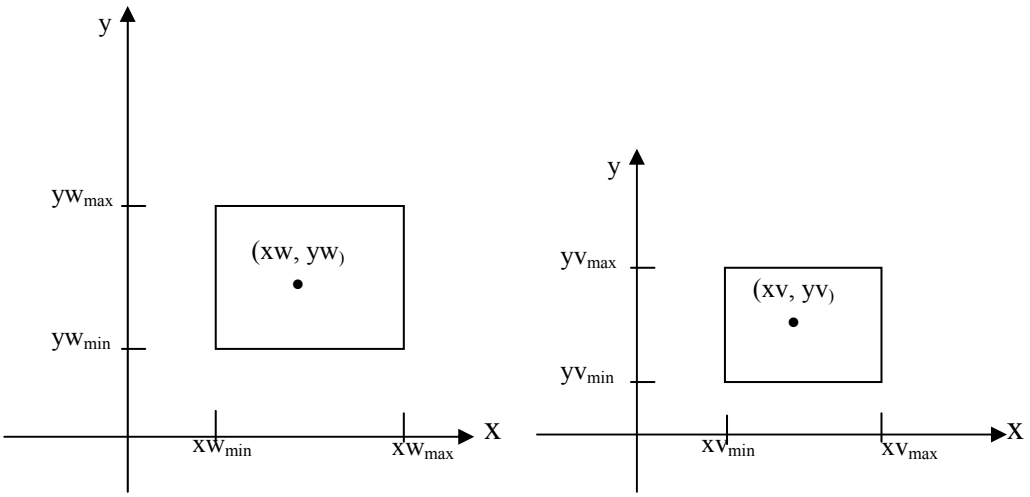
Việc xoá bỏ tất cả các màu kết xuất trong một vùng chỉ định có ý nghĩa thuận lợi cho việc nạp chồng các hình ảnh khác. Các kỹ thuật này thường được dùng để thiết kế các trang trình bày (layout) trong quảng cáo hoặc trong các ứng dụng xuất bản (publishing) hoặc cho việc thêm các nhãn hoặc mẫu thiết kế đến một hình ảnh. Kỹ

Các phép tính biến đổi từ cửa sổ - đến - vùng quan sát có thể được viết chặt chẽ hơn như sau:

$$xv = sx(xw - xw_{min}) + xv_{min} \qquad (4\text{-}13)$$
$$yv = sy(yw - yw_{min}) + yv_{min}$$

Phép biến đổi này bao gồm cả hai phép biến đổi tỉ lệ và tịnh tiến. Các hệ số tỉ lệ sx và sy phụ thuộc vào kích thước liên hệ của cửa sổ và vùng quan sát. Các hệ số tỉ lệ này phải bằng nhau nếu các đối tượng muốn được bảo tồn sự cân đối (đồng dạng) khi chúng được ánh xạ đến vùng quan sát. Khi cửa sổ và vùng quan sát có kích thước bằng nhau (sx = sy = 1), không có sự thay đổi nào về kích thước của các đối tượng được biến đổi. Giá trị của xv_{min} và yv_{min} cho biết các hệ số tịnh tiến để di chuyển các đối tượng vào vùng quan sát.

Các chuỗi kí tự có thể được quản lí theo hai cách khi chúng được ánh xạ đến vùng quan sát. Việc ánh xạ đơn giản nhất bảo tồn kích thước kí tự, thậm chí khi vùng quan sát được mở rộng hay thu nhỏ lại so với cửa sổ. Phương pháp này có thể được dùng đến khi văn bản được tạo ra với các font chuẩn – không thể bị thay đổi. Trong các hệ thống khi mà có sự cho phép thay đổi kích thước kí tự chuẩn, sự định nghĩa chuỗi có thể được đặt trong cửa sổ tương tự như các từ gốc. Đối với các kí tự được hình thành bởi các đoạn thẳng, việc ánh xạ đến vùng quan sát có thể được thực hiện như một dãy tuần tự các phép biến đổi đường (xem hình 4-23).



Hình 4-23: **Một điểm ở vị trí (xw, yw) trong cửa sổ được ánh xạ đến điểm (xv, yv) trong vùng quan sát. Việc ánh xạ được thực hiện sao cho tỷ lệ tương quan trong hai vùng tương tự nhau.**

6. So sánh số lượng các phép tính toán học được thực hiện trong các thuật toán clipping đường Cohen-Sutherland và Liang-Barsky đối với vài hướng đoạn thẳng khác nhau liên quan đến cửa sổ clipping.

7. Cài đặt thuật toán thuật toán clipping đường Liang-Barsky lên hệ thống của bạn.

8. Hãy nghĩ ra một thuật toán để thực hiện việc clipping đường bằng cách dùng phương pháp phân chia điểm ở giữa. Sự cài đặt phần mềm của thuật toán này có thuận lợi hơn hai thuật toán clipping đường đã được thảo luận trong chương không?

9. Cài đặt một thuật toán cắt các đoạn thẳng bằng cách dùng một cửa sổ bị quay, được định nghĩa bởi các giá trị tọa độ nhỏ nhất và lớn nhất và bị quay một góc như trong hình 6-5.

10. Thay đổi thuật toán clipping đa giác để cắt các vùng đa giác lõm một cách hợp lý. (Một phương pháp để thực hiện điều này là chia đa giác lõm ra làm các đa giác lồi.)

11. Sửa lại cho hợp lí thuật toán clipping đường Liang-Barsky để clipping đa giác.

12. Viết thủ tục để cắt một ellipse bằng cách dùng cửa sổ chữ nhật.

13. Giả sử rằng các kí tự được định nghĩa trong một lưới điểm (pixel grid), hãy phát triển một thuật toán clipping văn bản để cắt các kí tự đơn lẻ theo chiến lược "tất cả - hoặc - không".

14. Hãy phát triển một thuật toán clipping văn bản để cắt các kí tự đơn lẻ, giả sử rằng các kí tự được định nghĩa trong một lưới điểm (pixel grid).

15. Viết một thủ tục thực hiện xóa một phần bất kì của hình ảnh đã được định nghĩa, dùng kích thước cửa sổ xóa được xác định bất kỳ.

16. Viết các thủ tục để cài đặt các lệnh của cửa sổ và vùng quan sát. Tức là, các thủ tục có chứa tham số về hệ tọa độ trong các lệnh để thực hiện biến đổi sang vùng quan sát cho các cảnh cụ thể: clipping trong hệ tọa độ thế giới thực, chuyển đổi sang hệ tọa độ chuẩn hóa, sau cùng biến đổi đến hệ tọa độ thiết bị.

Các đối tượng có thể được biểu diễn bằng mô hình Wire-Frame.

Nhận thấy rằng khi biểu diễn đối tượng, ta có thể chọn gốc tọa độ và đơn vị đo lường sao cho việc biểu diễn là thuận lợi nhất. Thường thì người ta chuẩn hóa kích thước của đối tượng khi biểu diễn.

Boudary representation cho phép xử lý nhanh còn silid modeling cho hình ảnh đầy đủ và xác thực hơn.

- Loại bỏ các đối tượng không nhìn thấy được (**Trivial Rejection**).

  Loại bỏ các đối tượng hoàn toàn không thể nhìn thấy trong cảnh.

  Thao tác này giúp ta lược bỏ bớt các đối tượng không cần thiết do đó giảm chi phí xử lý.

- Chiếu sáng các đối tượng (**Illumination**).

Gán cho các đối tượng màu sắc dựa trên các đặc tính của các chất tạo nên chúng và các nguồn sáng tồn tại trong cảnh.

Có nhiều mô hình chiếu sáng và tạo bóng : constant-intensity, Interpolate,...

- Chuyển từ word space sang eye space (**Viewing Transformation**).

Thực hiện một phép biến đổi hệ tọa độ để đặt vị trí quan sát (viewing position) về gốc tọa độ và mặt phẳng quan sát (viewing plane) về một vị trí mong ước.

Hình ảnh hiển thị phụ thuộc vào vị trí quan sát và góc nhìn.

Hệ qui chiếu có gốc đặt tại vị trí quan sát và phù hợp với hướng nhìn sẽ thuận lợi cho các xử lý thật.

- Loại bỏ phần nằm ngoài viewing frusturn (**Clipping**).

Thực hiện việc xén đối tượng trong cảnh để cảnh nằm gọn trong một phần không gian hình chóp cụt giới hạn vùng quan sát mà ta gọi là **viewing frustum**. Viewung frustum có trục trùng với tia nhìn, kích thước giới hạn bởi vùng ta muốn quan sát.

- Chiếu từ eye space xuống screen space (**Projection**).

Thực hiện việc chiếu cảnh 3 chiều từ không gian quan sát xuống không gian màn hình.

Có 2 phương pháp chiếu:

- Chiếu song song
- Chiếu phối cảnh

```
        int     dx;
        int     dy;
        int     dz;
} Vector ;
```

- **Đoạn thẳng** trong không gian 3 chiều: biểu diễn tổ hợp tuyến tính của 2 điểm

  Để biểu diễn dạng tham số của đoạn thẳng, ta có :

  $P = P_1 + t*(P_2 - P_1)$, $(0 \le t \le 1)$

```
typedef  struct  {
        Point  P1;
        Point  P2;
} Segment ;
```

- **Tia (Ray)** : là một đoạn thẳng với một đầu nằm ở vô cực.

  Biểu diễn dạng tham số của tia :

  $P = P_1 + t*V$, $(0 \le t < \infty)$

```
typedef  struct  {
        Point  P1;
        VectorV;
} Ray;
```

- **Đường thẳng (Line)**: là một đoạn thẳng với cả hai đầu nằm ở vô cực

  Biểu diễn dạng tham số của đường thẳng

  $P = P_1 + t*V$, $(\infty \le t < \infty)$

```
typedef  struct  {
        Point  P1;
        VectorV;
} Line;
```

- **Đa giác (Polygon)** : là một vùng giới hạn bởi hạn dãy các điểm đồng phẳng .

(Các điểm được cho theo thứ tự ngược chiều kim đồng hồ)

```
typedef  struct  {
```

```
        Point   *Points;
        int     nPoints;
} Polygon;
```

Có thể biểu diễn một mặt đa giác bằng một tập hợp các đỉnh và các thuộc tính kèm theo. Khi thông tin của mỗi mặt đa giác được nhập, dữ liệu sẽ được điền vào các bảng (mãng dữ liệu) sẽ được dùng cho các xử lý tiếp theo, hiển thị và biến đổi.

Các bảng dữ liệu mô tả mặt đa giác có thể tổ chức thành hai nhóm : bảng hình học và bảng thuộc tính. Các bảng lưu trữ dữ liệu hình học chứa tọa độ các đỉnh và các tham số cho biết về định hướng trong không gian của mặt đa giác. Thông tin về thuộc tính của các đối tượng chứa các tham số mô tả độ trong suốt, tính phản xạ và các thuộc tính kết cấu của đối tượng. Một cách tổ chức thuận tiện để lưu trữ các dữ liệu hình học là tạo ra 3 danh sách : một bảng lưu đỉnh, một bảng lưu cạnh và một bảng lưu đa giác. Trong đó:

- Các giá trị tọa độ cho mỗi đỉnh trong đối tượng được chứa trong bảng lưu đỉnh.

- Bảng cạnh chứa các con trỏ trỏ đến bảng đỉnh cho biết đỉnh nào được nối với một cạnh của đa giác .

- Cuối cùng là bảng lưu đa giác chứa các con trỏ trỏ đến bảng lưu cạnh cho biết những cạnh nào tạo nên đa giác.

- **Mặt phẳng (Plane)** :

```
typedef   struct  {
        Vector      N;
        int         d;
} Plane;
```

Phương trình biểu diễn mặt phẳng có dạng : $Ax + By + Cz + D = 0$ (5-1)

Trong đó (x,y,z) là một điểm bất kỳ của mặt phẳng và A, B, C, D là các hằng số diễn tả thông tin không gian của mặt phẳng.

Để xác định phương trình mặt phẳng, ta chỉ cần xác định 3 điểm không thẳng hàng của mặt phẳng này. Như vậy, để xác định phương trình mặt phẳng qua một đa giác, ta sẽ sử dụng tọa độ của 3 đỉnh đầu tiên (x_1,y_1), (x_2,y_2), (x_3,y_3) trong đa giác này.

Từ phương trình (5-1) ta có :

$$Ax_k + By_k + Cz_k + D = 0 \qquad , k= 0,1,2,3. \qquad (5\text{-}2)$$

Trong đó :

$$A = \begin{vmatrix} 1 & y_1 & z_1 \\ 1 & y_3 & z_2 \\ 1 & y_3 & z_3 \end{vmatrix} \qquad B = \begin{vmatrix} x_1 & 1 & z_1 \\ x_2 & 1 & z_2 \\ x_3 & 1 & z_3 \end{vmatrix}$$

$$C = \begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_3 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} \qquad C = \begin{vmatrix} x_1 & y_1 & z_1 \\ x_2 & y_3 & z_2 \\ x_3 & y_3 & z_3 \end{vmatrix}$$

Khai triển các định thức trên ta có :

$A = y_1(z_2 - z_3) + y_2(z_3 - z_1) + y_3(z_1 - z_2)$

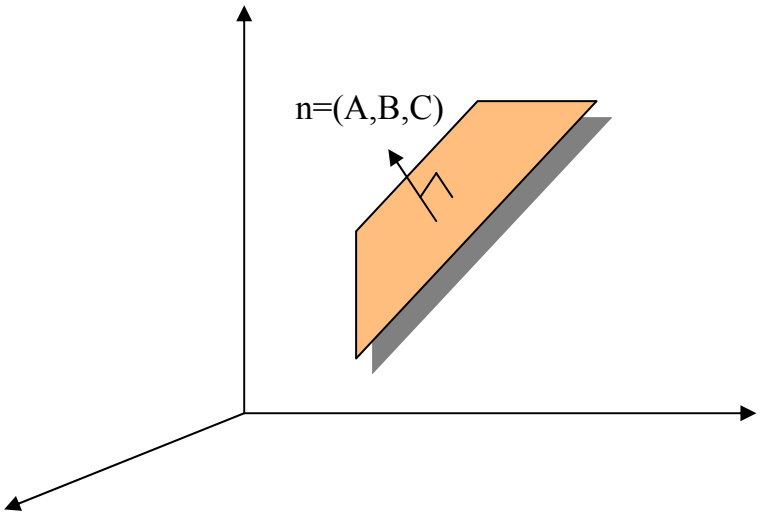
$B = z_1(x_2 - x_3) + z_2(x_3 - x_1) + z_3(x_1 - x_2)$

$C = x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2)$

$A = - x_1(y_2z_3 - y_3z_2) - x_2(y_3z_1 - y_1z_3) - x_3(y_1z_2 - y_2z_1)$

Hướng của mặt phẳng thường được xác định thông qua véc tơ pháp tuyến của nó. Véc tơ pháp tuyến $\vec{n}$ = (A,B,C) (xem hình 5-1)

.



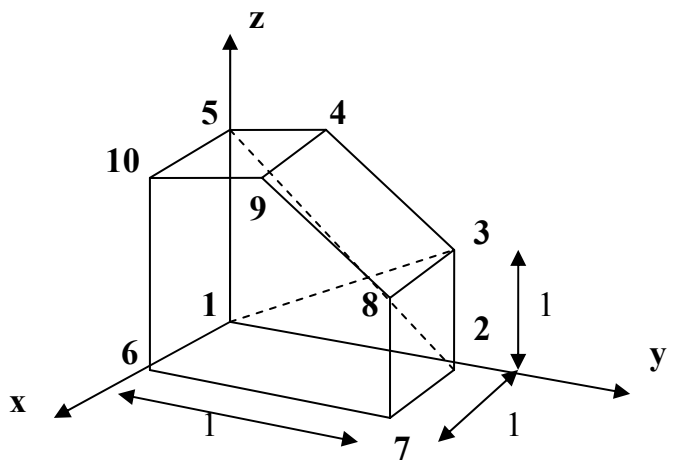
Hình 5.1 : Vec tơ pháp tuyến của mặt phẳng.

- **Mô hình khung nối kết (Wireframe-Model)**

Một phương pháp thông dụng và đơn giản để mô hình hóa đối tượng là mô hình khung nối kết. Một mô hình khung nối kết gồm có một tập các đỉnh và tập các cạnh nối các đỉnh đó. Khi thể hiện bằng mô hình này, các đối tượng 3 chiếu có vẻ rỗng và không giống thực tế lắm. Tuy nhiên, vẽ bằng mô hình này thì nhanh nên người ta

thường dùng nó trong việc xem phác thảo các đối tượng. Để hoàn thiện hơn, người ta dùng các kỹ thuật tạo bóng và loại bỏ các đường khuất, mặt khuất.

Với mô hình khung nối kết, hình dạng của đối tượng 3 chiều được biểu diễn bằng hai danh sách (list) : danh sách các đỉnh (vertices) và danh sách các cạnh (edges) nối các đỉnh đó. Danh sách các đỉnh cho biết thông tin hình học (đó là vị trí các đỉnh), còn danh sách các cạnh xác định thông tin về sự kết nối (cho biết cặp các đỉnh tạo ra cạnh). Chúng ta hãy quan sát một vật thể ba chiều (xem hình 5-2) được biểu diễn bằng mô hình khung nối kết như sau:



Hình 5.2 : Vật thể 3 chiều được biểu diễn bằng khung nối kết.

Bảng danh sách các cạnh và đỉnh biểu diễn vật thể

Vertex List				
Vertex	**x**	**y**	**z**	
1	0	0	0	back side
2	0	1	0	
3	0	1	1	
4	0	0.5	1.5	
5	0	0	1	
6	1	0	0	front side
7	1	1	0	
8	1	1	1	
9	1	0.5	1.5	
10	1	0	1	

Edge List		
Edge	**Vertex1**	**Vertex2**
1	1	2
2	2	3
3	3	4
4	4	5
5	5	1
6	6	7
7	7	8
8	8	9
9	9	10
10	10	6
11	1	6
12	2	7
13	3	8
14	4	9
15	5	10
16	2	5
17	1	3

- Phép quay quanh trục Z

$$R(z,\theta) = \begin{pmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Phép quay quanh trục X

$$R(x,\theta) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Phép quay quanh trục Y

$$R(y,\theta) = \begin{pmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

- Cách xác định chiều dương trong các phép quay

Định nghĩa về chiều quay được dùng chung cho cả hệ tọa độ theo qui ước bàn tay phải và bàn tay trái. Cụ thể chiều dương được định nghĩa như sau :

- Quay quanh truc x : từ trục dương y đến trục dương x
- Quay quanh trục y : từ trục dương z đến trục dương x
- Quay quanh trục x : từ trục dương x đến trục dương y

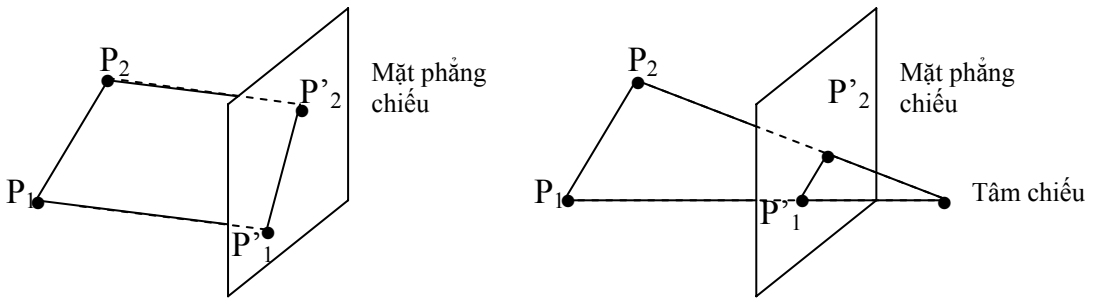
- Phép đối xứng qua mặt phẳng tọa độ

$$(yOx): \quad Mr(x) = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

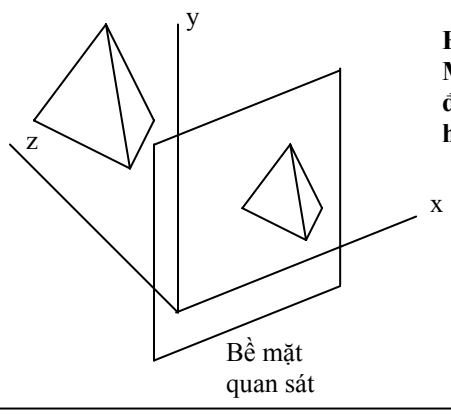
$$(zOx): \quad Mr(y) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

một nhóm các đối tượng. Ngoài ra, sự mô tả các đối tượng ba chiều phải được chiếu lên bề mặt quan sát của thiết bị xuất. Trong chương này, trước hết chúng ta sẽ thảo luận các cơ chế của phép chiếu. Sau đó, các thao tác liên quan đến phép biến đổi cách quan sát, và đầy đủ các kỹ thuật quan sát ảnh ba chiều sẽ được phát triển.

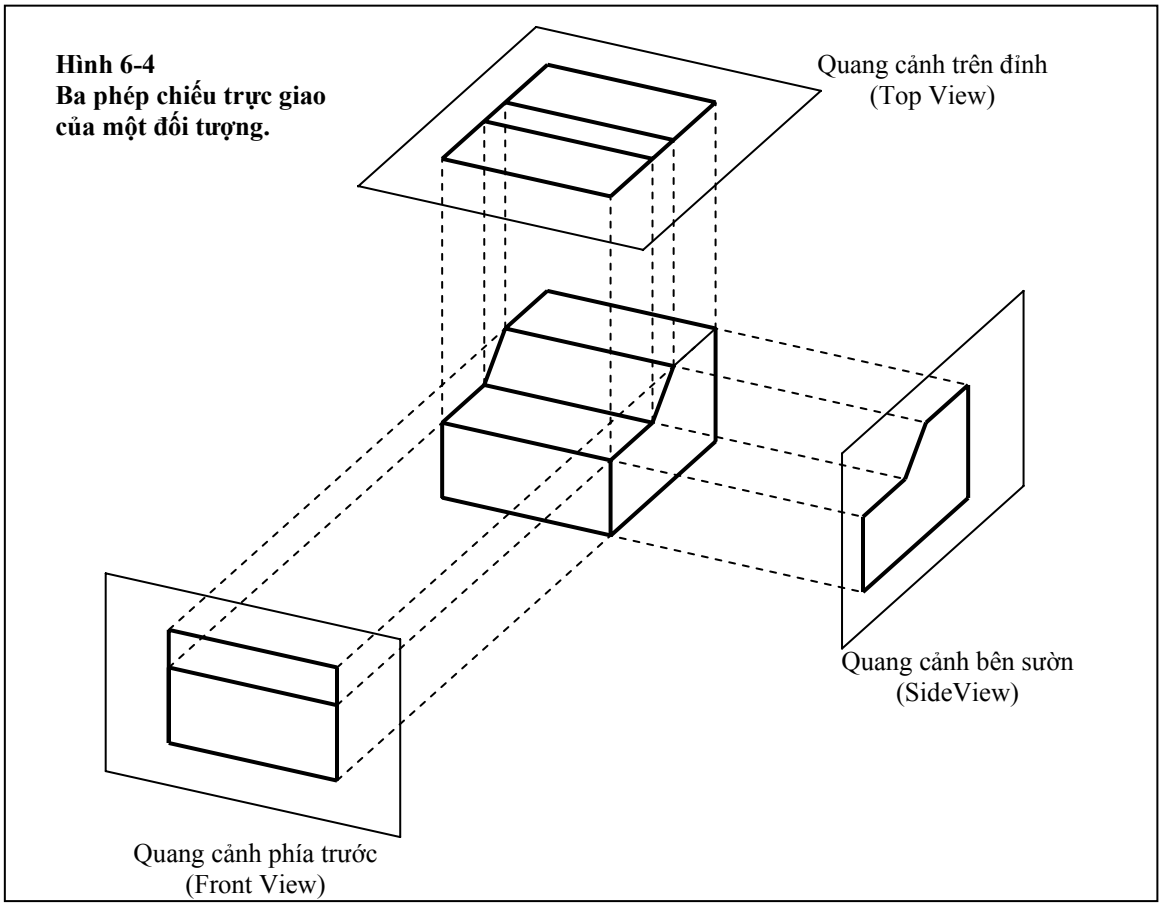
Có hai phương pháp cơ bản để chiếu các đối tượng ba chiều lên bề mặt quan sát hai chiều. Tất cả các điểm của đối tượng có thể được chiếu lên bề mặt theo các đường thẳng song song, hoặc các điểm có thể được chiếu theo các đường hội tụ về một điểm được gọi là **tâm chiếu (the center of projection)**. Hai phương pháp này được gọi là **phép chiếu song song (parallel projection)** và **phép chiếu phối cảnh (perspective projection)** (xem hình 6-1). Trong cả hai trường hợp, giao điểm của đường chiếu với bề mặt quan sát xác định các tọa điểm của điểm được chiếu lên mặt phẳng chiếu này. Chúng ta giả sử rằng mặt phẳng chiếu là mặt z = 0 của hệ tọa độ bàn tay trái (left-handed coordinate system) (xem hình 6-2).



Hình 6-1 Hai phương pháp chiếu một đoạn thẳng lên bề mặt của mặt phẳng chiếu



Hình 6-2
Một bề mặt quan sát được định nghĩa trong mặt z=0 của hệ tọa độ bàn tay trái.

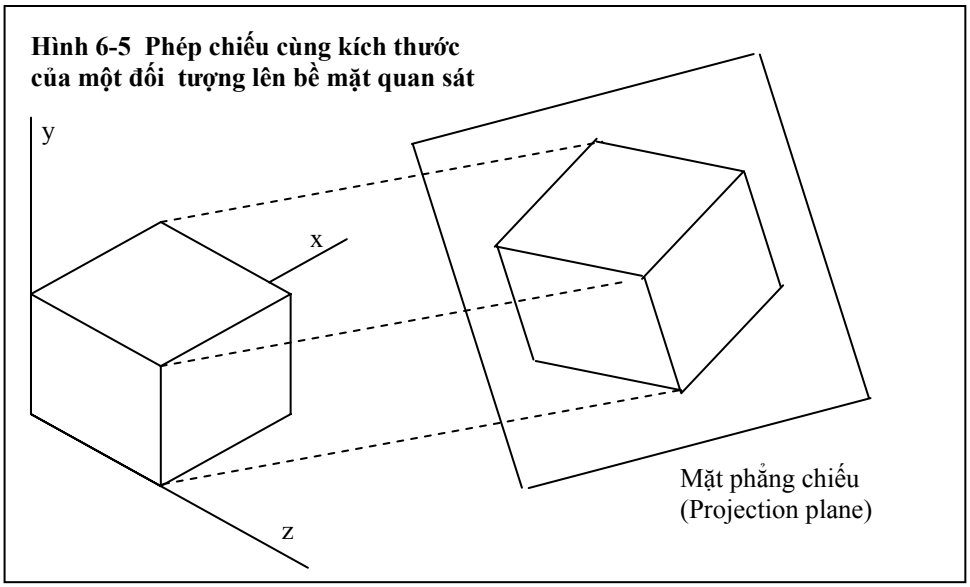


Hình 6-4
Ba phép chiếu trực giao của một đối tượng.

Chúng ta cũng có thể xây dựng các phép chiếu trực giao để có thể quan sát nhiều hơn một mặt của một đối tượng. Các quang cảnh như thế được gọi là các phép chiếu trực giao **trục lượng học (axonometric orthographic projection)**. Hầu hết phép chiếu trục lượng học được dùng là **phép chiếu cùng kích thước (isometric projection).** Một phép chiếu cùng kích thước được thực hiện bằng việc sắp xếp song song mặt phẳng chiếu mà nó cắt mỗi trục tọa độ ở nơi đối tượng được định nghĩa (được gọi là các trục chính) ở các khoảng cách như nhau từ ảnh gốc. Hình 6-5 trình bày phép chiếu cùng kích thước. Có tám vị trí, một trong tám mặt, đều có kích thước bằng nhau. Tất cả ba trục chính được vẽ thu gọn bằng nhau trong phép chiếu cùng kích thước để kích thước liên hệ của các đối tượng được bảo tồn. Đây không là trường hợp phép chiếu trực giao trục lượng học tổng quát, khi mà các hệ số tỷ lệ theo ba trục chính có thể khác nhau.

Các phương trình biến đổi để thực hiện một phép chiếu song song trực giao thì dễ hiểu. Đối với điểm bất kỳ (x, y, z), điểm chiếu (x_p, y_p, x_p) trên bề mặt chiếu được tính như sau:

$$x_p = x, \qquad y_p = y, \qquad z_p = 0 \qquad (6\text{-}1)$$

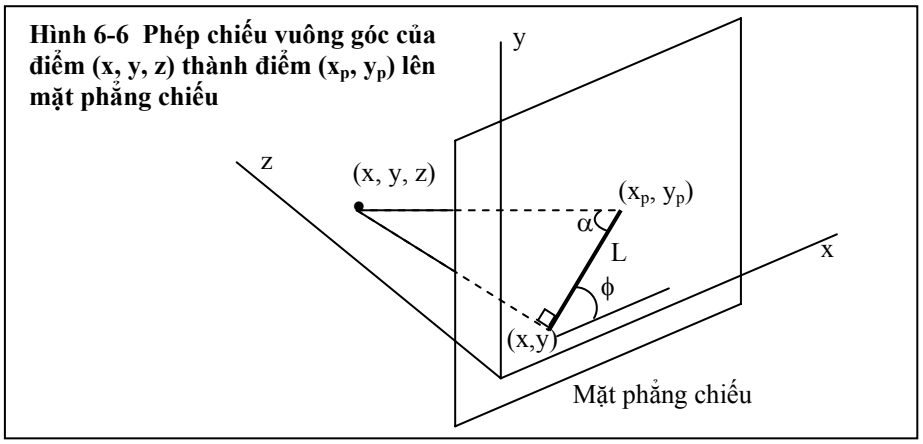


Hình 6-5 Phép chiếu cùng kích thước của một đối tượng lên bề mặt quan sát

Một phép chiếu xiên đạt được bằng việc chiếu các điểm theo các đường thẳng song song, các đường thẳng này không vuông góc với mặt phẳng chiếu. Hình 6-6 trình bày hình chiếu xiên của điểm (x, y, z) theo một đường thẳng chiếu đến vị trí (x_p, y_p). Các tọa độ chiếu trực giao trên mặt phẳng chiếu là (x, y). Đường thẳng của phép chiếu xiên tạo một góc α với đường thẳng trên mặt phẳng chiếu (đây là đường nối điểm (x_p, y_p) với điểm (x, y)). Đường này, có chiều dài L, hợp một góc ϕ với phương ngang trên mặt phẳng chiếu. Chúng ta có thể diễn tả các tọa độ chiếu qua các số hạng x, y, L, và ϕ:

$$x_p = x + L\cos\phi \qquad (6\text{-}2)$$

$$y_p = y + L\sin\phi$$



Hình 6-6 Phép chiếu vuông góc của điểm (x, y, z) thành điểm (x_p, y_p) lên mặt phẳng chiếu

Phương chiếu có thể định nghĩa bằng việc chọn các giá trị cho góc α và ϕ. Các chọn lựa thông thường cho góc ϕ là 30^o và 45^o, là các góc hiển thị một quang cảnh của mặt trước, bên sườn, và trên đỉnh (hoặc mặt trước, bên sườn, và dưới đáy) của một đối

tượng. Chiều dài L là một hàm của tọa độ z, và chúng ta có thể tính tham số này từ các thành phần liên quan.

$$\tan \alpha = \frac{z}{L} = \frac{1}{L_1} \qquad (6\text{-}3)$$

ở đây L_1 là chiều dài của các đường chiếu từ (x, y) đến (x_p, y_p) khi z = 1.

Từ phương trình 6-3, chúng ta có

$$L = z\, L_1 \qquad (6\text{-}4)$$

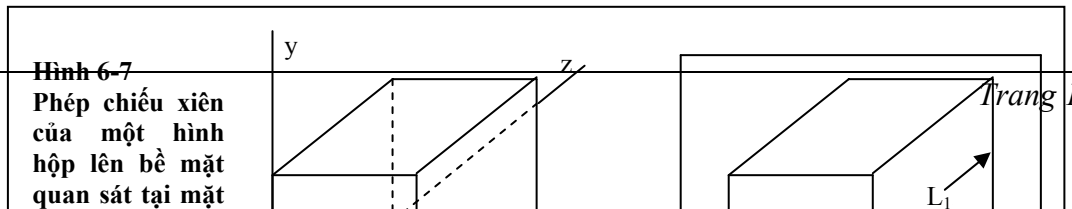
và các phương trình của phép chiếu xiên 6-2 có thể được viết lại như sau

$$x_p = x + z(L_1 \cos\phi) \qquad (6\text{-}5)$$
$$y_p = y + z(L_1 \sin\phi)$$

Ma trận biến đổi để tạo ra bất kỳ việc chiếu song song có thể được viết như sau

$$P_{parallel} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ L_1 \cos\varphi & L_1 \sin\varphi & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (6\text{-}6)$$

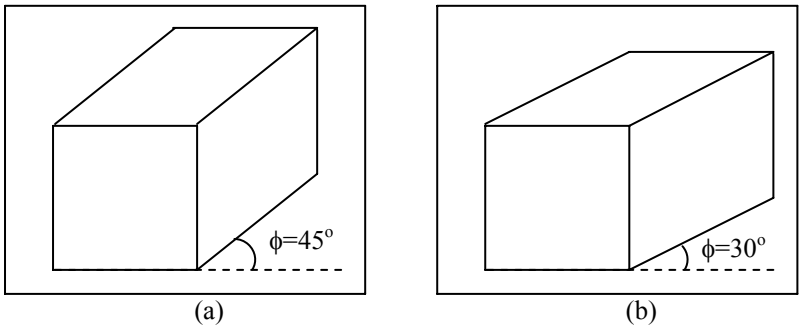
Một phép chiếu trực giao có thể đạt được khi $L_1 = 0$ (xảy ra ở góc chiếu $\alpha=90^o$). Các phép chiếu xiên được sinh ra với giá trị L_1 khác không. Ma trận chiếu 6-6 có cấu trúc tương tự ma trận của phép làm biến dạng theo trục z. Thực tế, kết quả của ma trận chiếu này là làm biến dạng mặt phẳng của hằng z và chiếu chúng lên mặt phẳng quan sát. Các giá trị tọa độ x và y trong mỗi mặt của hằng z bị thay đổi bởi một hệ số tỷ lệ đến giá trị z của mặt phẳng để các góc, các khoảng cách, và các đường song song trong mặt phẳng được chiếu chính xác. Hiệu quả này được thể hiện trong hình 6-7, ở đây mặt sau của hình hộp bị biến dạng và bị nằm đè bởi mặt trước trong phép chiếu đến bề mặt quan sát. Một cạnh của hình hộp, cái nối mặt trước với mặt sau, được chiếu thành đoạn chiều dài L_1, cái hợp thành một góc ϕ với đường ngang trong mặt phẳng chiếu.



Hình 6-7
Phép chiếu xiên của một hình hộp lên bề mặt quan sát tại mặt

Hai góc được dùng phổ biến trong phép chiếu xiên là các góc có tgϕ =1 và tgϕ=2. Trường hợp đầu, ϕ = 45^o và quang cảnh đạt được được gọi là phép chiếu **cavalier**. Tất cả các đường vuông góc v ới mặt phẳng chiếu được chiếu với chiều dài không thay đổi. Các ví dụ của phép chiếu cavalier đối với một hình lập phương được cho trong hình 6-8.

Hình 6-8
Phép chiếu cavalier của một hình lập phương lên bề mặt chiếu với hai giá trị góc ϕ. Độ sâu của phép chiếu bằng với chiều rộng và chiều cao.



Khi góc chiếu đuợc chọn để tgϕ = 2, kết quả quang cảnh được gọi là phép chiếu **cabinet**. Góc phép chiếu này xấp xỉ 63.4^o làm cho các đường chiếu vuông góc với bề mặt chiếu được chiếu ở một nữa chiều dài của chúng.

Các phép chiếu cabinet cho hình ảnh thực hơn phép chiếu cavalier vì sự thu giảm chiều dài của các đường song song. Hình 6-9 trình bày phép chiếu cabinet cho hình lập phương.

(0, 0, d). Để thu được các tọa độ trên mặt phẳng chiếu, chúng ta đặt z' = 0 và tìm ra tham số u:

$$u = \frac{z}{z+d} \qquad (6\text{-}8)$$

Giá trị của tham số u tạo ra giao điểm của đường chiếu với mặt phẳng chiếu tại (x_p, y_p, 0). Thế phương trình 6-8 vào phương trình 6-7, ta thu được các phương trình biến đổi của phép chiếu phối cảnh.

$$x_p = x\left(\frac{d}{z+d}\right) = x\left(\frac{1}{z/d+1}\right)$$

$$y_p = y\left(\frac{d}{z+d}\right) = y\left(\frac{1}{z/d+1}\right) \qquad (6\text{-}9)$$

$$z_p = 0$$

Dùng biểu diễn hệ tọa độ thuần nhất ba chiều (three-dimentional homogeneous coordinate representation), chúng ta có thể viết phép biến đổi phối cảnh theo hình thức ma trận:

$$[x_h\ y_h\ x_h\ w] = [x\ y\ z\ 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1/d \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad (6\text{-}10)$$

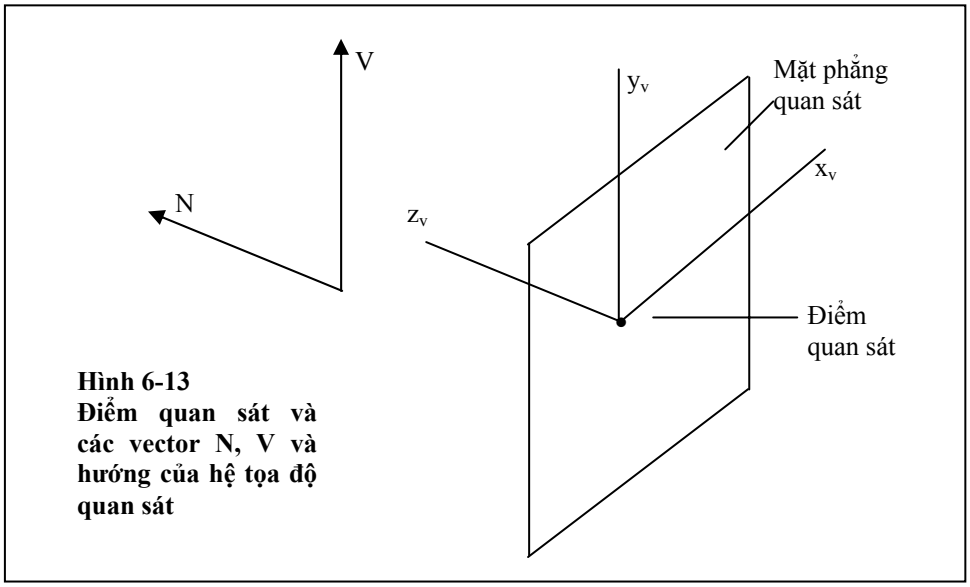
Trong biểu diễn này,

$$w = \frac{z}{d} + 1 \qquad (6\text{-}11)$$

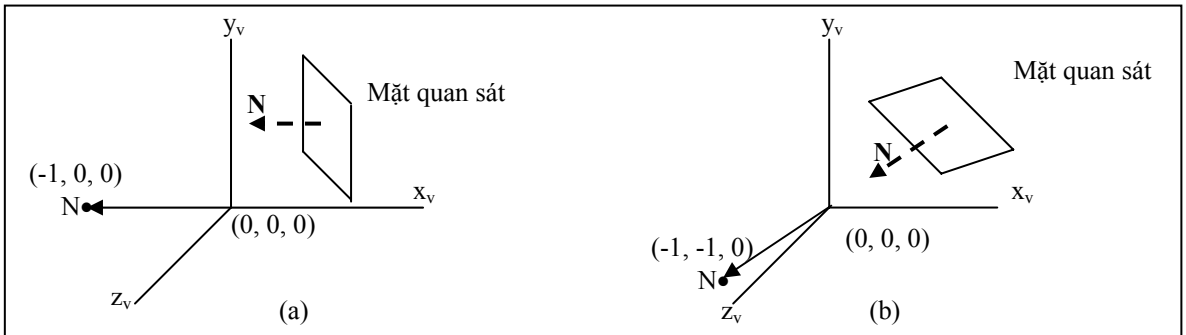
và các tọa độ chiếu trên mặt phẳng chiếu được tính từ các tọa độ thuần nhất như sau

$$[x_p\ y_p\ z_p\ 1] = [x_h/w\ y_h/w\ z_h/w\ 1] \qquad (6\text{-}12)$$

Khi các đối tượng ba chiều được chiếu lên một mặt phẳng dùng các phương trình biến đổi phối cảnh, bất kỳ tập hợp các đường thẳng song song nào của đối tượng mà không song song với mặt phẳng chiếu được chiếu thành các đường hội tụ (đồng quy). Các đường thẳng song song với mặt phẳng khi chiếu sẽ tạo ra các đường song song. Điểm mà tại đó tập hợp các đường thẳng song song được chiếu xuất hiện hội tụ về đó được gọi là **điểm ảo** (vanishing point). Mỗi tập hợp các đường thẳng song song được chiếu như thế sẽ có một điểm ảo riêng (xem hình 6.11).



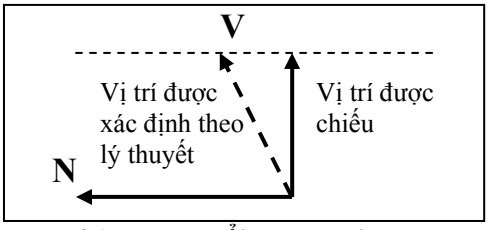
Hình 6-13
Điểm quan sát và các vector N, V và hướng của hệ tọa độ quan sát



Hình 6-14
Hướng của mặt phẳng quan sát để xác định các tọa độ vector pháp tuyến. Vị trí (-1, 0, 0) định hướng mặt phẳng quan sát trong (a), trong khi đó vị trí (-1, -1, 0) cho hướng trong (b).

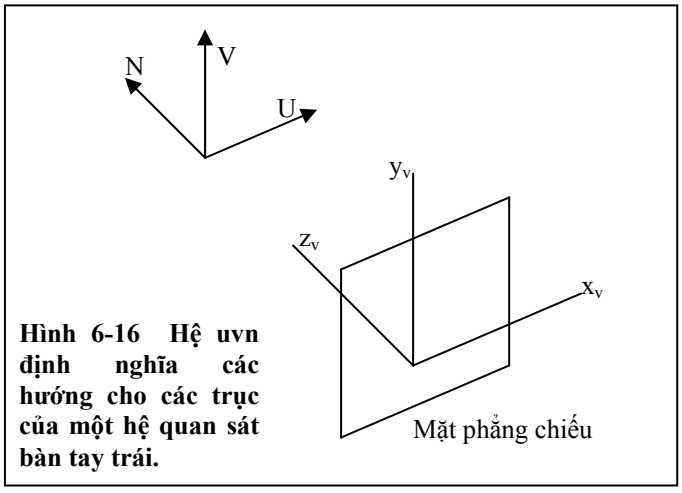
Vector pháp tuyến của mặt phẳng quan sát N có thể được xây dựng bằng việc xác định một vị trí tọa độ liên hệ với gốc tọa độ thế giới thực. Việc làm này định nghĩa hướng của vector pháp tuyến như đường thẳng từ gốc (của tọa độ thế giới thực) đến vị trí tọa độ

được chỉ định (gốc hệ quan sát). Hình 6-14 cho hai hướng của mặt phẳng quan sát để các tọa độ vector pháp tuyến được xác định. Vector V có thể được xác định theo cách tương tự. Người sử dụng thường khó khăn để xác định chính xác hai vector vuông góc này, vì vậy một vài gói đồ họa thay đổi cách xác định vector V của người dùng. Như được thể hiện trong hình 6-15, V được chiếu đến vị trí để vuông gốc với pháp vector.



Hình 6-15 Thay đổi sự xác định theo lý thuyết của vector V đến vị trí vuông góc với vector N.

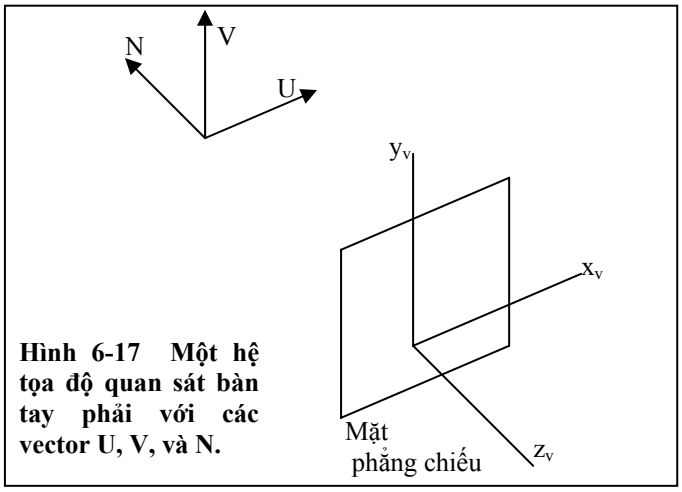
Đôi khi vector thứ ba U, được dùng để chỉ rõ hướng x của hệ quan sát. Hệ quan sát sau đó có thể được mô tả như hệ *uvn*, và mặt phẳng quan sát được gọi là mặt *uv*. Chúng ta giả thuyết rằng vị trị x theo hướng như ở hình 6-16. Hướng của U và V trong bức ảnh này thì không đổi so với hướng chuẩn của trục x và y trên thiết bị hiển thị. Chúng ta có thể nghĩ về mặt phẳng quan sát trong hệ quan sát này như một thiết bị logic (logical device) làm cơ sở cho việc hiển thị ảnh.



Hình 6-16 Hệ uvn định nghĩa các hướng cho các trục của một hệ quan sát bàn tay trái.

Dù là hệ tọa độ bàn tay trái (xem hình 6-16) hay hệ tọa độ bàn tay phải (xem hình 6-17) đều có thể được dùng làm hệ quan sát. Trong các thảo luận sau này, chúng ta sẽ dùng hệ tọa độ bàn tay trái, vì nó trực quan hơn một chút. Các đối tượng xa hơn từ người quan sát có các giá trị theo trục z lớn. Tuy nhiên, hệ tọa độ bàn tay phải thường được

dùng, vì nó có hướng tương tụ như hệ tọa độ thế giới thực. Do đó, sự biến đổi giữa hai hệ này được làm đơn giản.

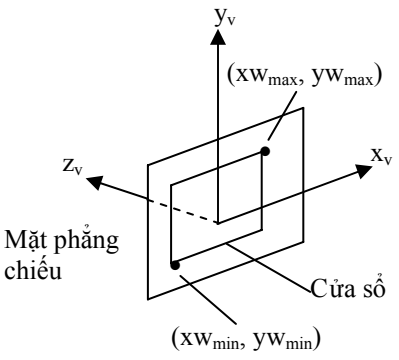


Hình 6-17 Một hệ tọa độ quan sát bàn tay phải với các vector U, V, và N.

Trong việc xây dựng mặt phẳng quan sát, vài vùng đồ họa sử dụng các tham số bổ sung được gọi là khoảng cách quan sát. Mặt phẳng quan sát được định nghĩa như mặt phẳng song song với mặt phẳng xy, cái nằm ở một khoảng cách xác định từ điểm quan sát. Đối với thảo luận của ta, chúng ta giả thuyết rằng mặt phẳng quan sát là mặt xy ở gốc tọa độ của hệ quan sát. Điều này cho phép chúng ta chiếu lên mặt $z = 0$.

Để tạo ra một quang cảnh từ một điểm quan sát thuận lợi do người dùng chọn, các vị trí được định nghĩa liên hệ với gốc của hệ tọa độ thế giới thực phải được định nghĩa lại liên hệ với gốc của hệ quan sát. Tức là, chúng ta phải biến đổi các tọa độ từ hệ tọa độ thế giới thực sang hệ tọa độ quan sát. Sự biến đổi này được thực hiện bằng một dãy biến đổi tuần tự của phép tịnh tiến và phép quay để ánh xạ các trục của hệ tọa độ quan sát lên trên các trục của hệ tọa độ thế giới thực. Khi được áp dụng đến định nghĩa hệ tọa độ thế giới thực của các đối tượng trong ảnh, dãy biến đổi tuần tự này biến đổi chúng đến vị trí mới trong hệ tọa độ quan sát. Ma trận biểu diễn dãy biến đổi tuần tự này có thể được thu được bằng việc kết hợp các ma trận biến đổi như sau (xem hình 6-18):

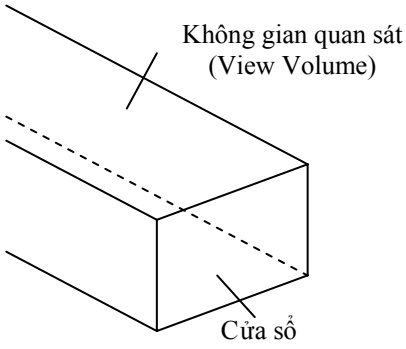
1. Phản chiếu liên hệ đến mặt xy, đảo ngược dấu mỗi tọa độ z. Điều này thay đổi hệ quan sát bàn tay trái thành hệ quan sát bàn tay phải.
2. Tịnh tiến điểm quán sát đến gốc của hệ tọa độ thế giới thực.
3. Quay quanh trục tọa độ thế giới thực x để mang trục tọa độ quan sát z vào mặt phẳng xz của hệ tọa độ thế giới thực.



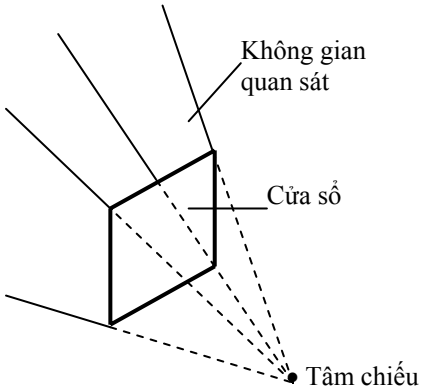
Hình 6-19
Sự xác định cửa sổ trên mặt phẳng chiếu, tọa độ điểm thấp nhất và cao nhất được cho trong hệ quan sát

Cửa sổ chiếu được dùng để định nghĩa một **không quan sát (view volume)**. Chỉ những đối tượng nằm trong không quan sát mới được chiếu và hiển thị lên mặt phẳng chiếu. Hình dạng chính xác của không quan sát dựa vào kiểu phép chiếu được yêu cầu bởi người dùng. Trong bất kỳ trường hợp nào, bốn mặt của không gian quan sát đi xuyên qua các cạnh của cửa sổ. Với phép chiếu song song, bốn mặt của không gian quan sát này hình thành một hình hộp không giới hạn (xem hình 6-20). Một hình chóp bị cắt cụt (hình kim tự tháp), với đỉnh nằm ở tâm chiếu (xem hình 6-21), được dùng như không gian quan sát cho phép chiếu phối cảnh. Hình chóp bị cắt cụt này được gọi là một **hình cụt (frustum)**.

Hình 6-20 Không gian quan sát cho phép chiếu song song.



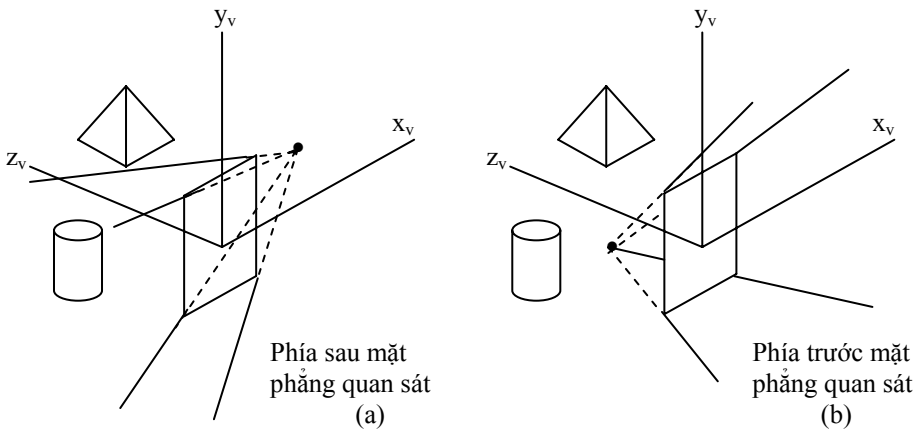
Hình 6-21 Không gian quan sát cho phép chiếu phối cảnh.



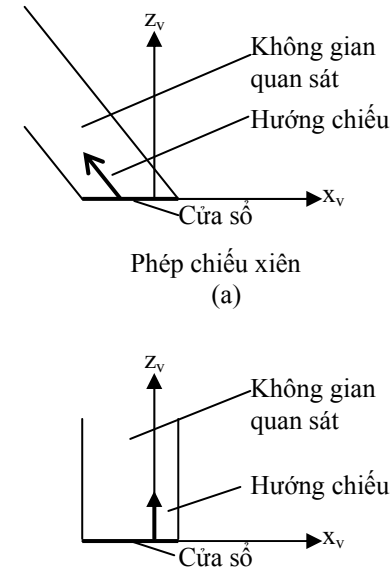
Vài vùng đồ họa giới hạn tọa độ của tâm chiếu là các vị trí dọc theo trục z của hệ quan sát. Chúng ta cần một tiếp cận tổng quát hơn là cho phép tâm chiếu được đặt ở bất

kỳ vị trí nào trong hệ quan sát. Hình 6-22 trình bày hai hướng của không gian quan sát hình chóp liên hệ với các trục quan sát. Trong hình 6-22 (b), không điểm nào chiếu đến mặt phẳng quan sát, vì tâm chiếu và các đối tượng được quan sát thì ở cùng phía với mặt quan sát. Trong trường hợp này, không có cái gì nào được hiển thị.

Hình 6-22
Hai vị trí tâm chiếu của không gian quan sát trong phép chiếu phối cảnh.



Trong các phép chiếu song song, hướng của phép chiếu định nghĩa hướng của không gian quan sát. Bằng cách cho một vị trí liên hệ đến gốc hệ quan sát, người dùng định nghĩa được một vector xác định hướng của không gian quan sát liên hệ với mặt phẳng quan sát. Hình 6-23 trình bày hình dạng của các không gian quan sát cho cả hai: phép chiếu song song trực giao và phép chiếu song song xiên.



Hình 6-23 Các không gian quan sát cho các phép chiếu song song xiên và trực giao, được quan sát tại mặt xz.

Thông thường, một hoặc hai mặt phẳng bổ sung được dùng để định nghĩa rõ hơn không gian quan sát. Gồm một **mặt gần (near plane)** và một **mặt xa (far plane)** tạo ra không gian quan sát có giới hạn, được bao quanh bởi sáu mặt phẳng, (xem hình 6-24). Các mặt gần và xa thì luôn song song với mặt phẳng quan sát, và chúng được xác định bởi các khoảng cách với mặt phẳng quan sát trong hệ quan sát. Các tên lần lượt cho các mặt gần và mặt xa là các mặt ở đây, ở đằng kia hay các mặt ở phía trước, phía sau.

cuối cùng là phải biến đổi mô tả trong hệ tọa độ thiết bị chuẩn vào trong các hệ tọa độ thiết bị và hiển thị quang cảnh lên một thiết bị xuất.

Mô hình được trình bày ở hình 6-25 thì hữu ích như mô hình cho lập trình viên hoặc cho việc khái niệm hóa các thao tác quan sát ba chiều. Tuy nhiên, để hiệu quả, việc cài đặt thật sự của việc quan sát ba chiều trong một gói đồ họa cần một hình thức khác hơn nhiều. Trong phần này, chúng ta nhìn vào những vấn đề, nơi mà các lo lắng về cài đặt làm cho chúng ra xa rời với mô hình cơ bản của việc quan sát ba chiều

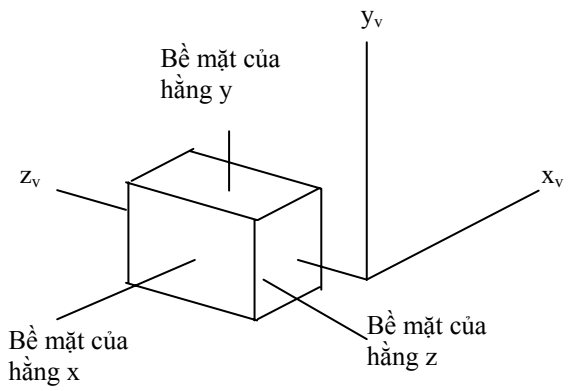
Các không gian quan sát được chuẩn hóa (Normalized View Volumes)

Clipping trong không gian hai chiều được thực hiện một cách tổng quát bởi một hình chữ nhật có các cạnh song song với trục x và y. Điều này làm đơn giản rất nhiều các tính toán clipping, vì mỗi biên cửa sổ được xác định bởi chỉ một giá trị tọa độ. Ví dụ, các giao điểm của các đoạn cắt biên trái cửa sổ đều có giá trị tọa độ x bằng với biên trái.

Trong mô hình của lập trình viên ba chiều, việc clipping được thực hiện bởi một không gian quan sát được xác định bởi cửa sổ chiếu, kiểu chiếu, và các mặt gần, xa. Bởi vì các mặt gần và xa song song với mặt phẳng chiếu, mỗi mặt có giá trị tọa độ z không đổi. Tọa độ z của các giao điểm của các đoạn với các mặt này thì đơn giản là tọa độ z của các mặt phẳng tương ứng. Nhưng bốn mặt còn lại của không gian quan sát có thể có hướng không gian tùy ý. Để tìm giao điểm của một đoạn với một trong các mặt đó ta cần tìm phương trình cho mặt phẳng chứa các mặt của không gian quan sát. Tuy nhiên, điều này trở nên không cần thiết nếu chúng ta biến đổi không gian quan sát trước khi clipping đến một hình hộp thông thường.

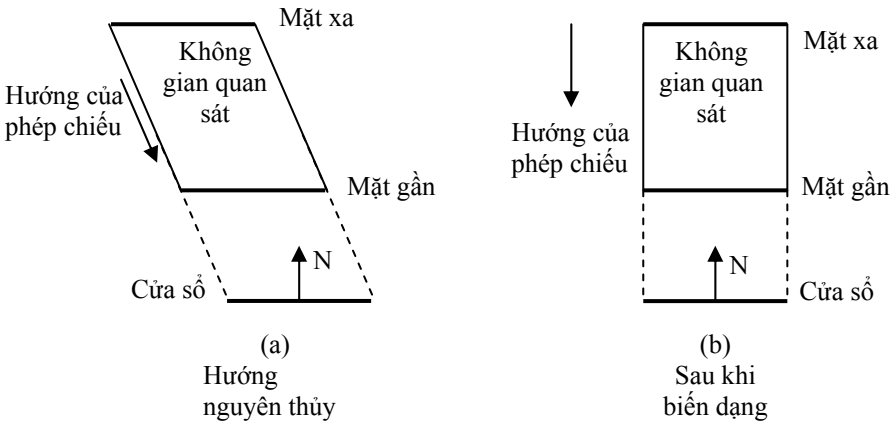
Clipping bởi một hình hộp thông thường thì đơn giản hơn bởi vì mỗi mặt bây giờ vuông góc với một trong các trục tọa độ. Như được trình bày trong hình 6-26, đỉnh và đáy của một không gian quan sát như thế là các mặt phẳng của hằng y, các mặt là các mặt phẳng của hằng x, và các mặt gần và xa có một giá trị z được xác định trước. Ví dụ, tất cả các đoạn thẳng cắt mặt trên đỉnh của hình hộp, bây giờ sẽ có giá trị tọa độ y của mặt đó. Thêm vào đó, để làm đơn giản hóa các thao tác clipping, việc biến đổi thành một hình hộp thông thường làm rút ngắn quá trình xử lý chiếu thành một phép chiếu trực giao đơn giản. Chúng ta đầu tiên xem xét làm thế nào để biến đổi một không gian quan sát thành một hình hộp thông thường, sau đó thảo luận về thao tác chiếu.

Hình 6-26
Một không gian quan sát hình hộp thông thường.



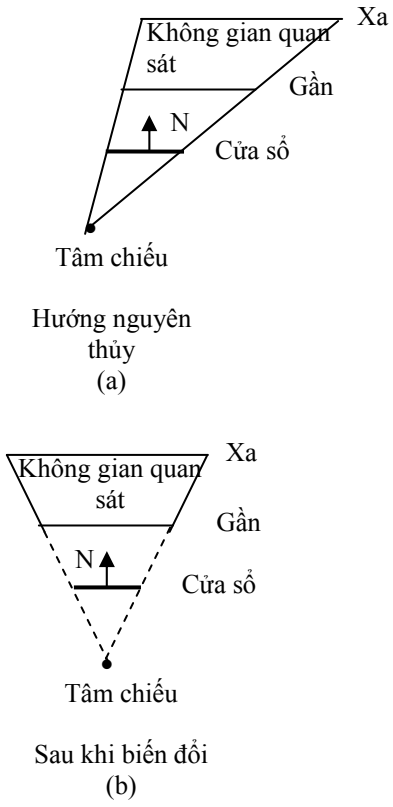
Trong trường hợp của một phép chiếu song song trực giao, không gian quan sát đã là một hình hộp chữ nhật ngay từ đầu. Đối với phép chiếu song song xiên, chúng ta làm biến dạng không gian quan sát để làm cho cùng phương hướng chiếu với hướng vector pháp tuyến của mặt phẳng quan sát, N. Phép biến dạng này mang các mặt của không gian quan sát hình hộp thành mặt quan sát, như trong hình 6-27.

Hình 6-27
Làm biến dạng một không gian quan sát chiếu song song xiên thành một hình hộp thông thường (quang cảnh nhìn từ trên đỉnh xuống).



Đối với không gian quan sát trong một phép chiếu phối cảnh, phép biến dạng và biến đổi tỷ lệ được cần để tạo ra hình hộp chữ nhật. Chúng ta đầu tiên làm biến dạng theo phương x và y để mang tâm chiếu đặt lên đường thẳng vuông góc tại tâm cửa sổ (Hình 6-28). Với đỉnh tại điểm này, các mặt đối diện của hình chóp cụt (trừ hai mặt gần, xa, ta có trái đối với phải, đỉnh đối với đáy) có cùng kích thước. Chúng ta sau đó áp dụng phép biến đổi tỷ lệ để biến đổi các mặt của hình chóp cụt hành các mặt chữ nhật của một hình hộp thông thường.

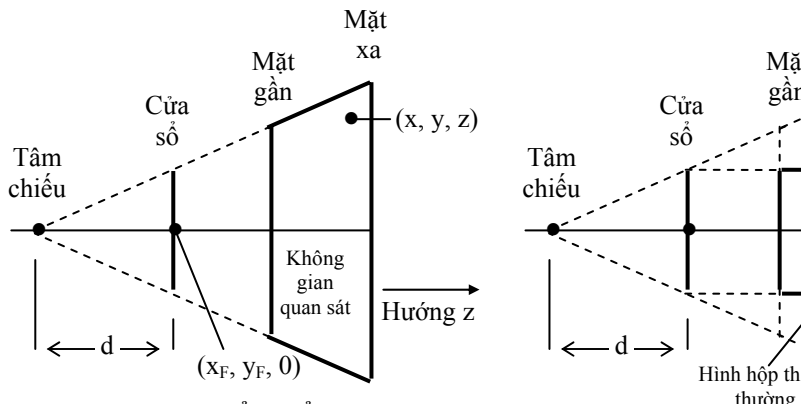
Hình 6-28
Biến dạng một không gian quan sát chiếu phối cảnh để mang tâm chiếu lên đường vuông góc với tâm cửa sổ (quang cảnh trên đỉnh)



Hình 6-29 trình bày một quang cảnh bên sườn của hình chóp cụt đối với phép chiếu song song. Để biến đổi hình chóp cụt này thành hình chữ nhật thông thường với chiều cao bằng với chiều cao cửa sổ, chúng ta áp dụng một phép biến đổi tỷ lệ liên hệ với điểm cố định (x_F, y_F, 0) ở tâm cửa sổ. Phép biến đổi này phải biến đổi theo tỷ lệ giữa các điểm trong hình chóp ở xa cửa sổ hơn so với các điểm ở gần cửa sổ hơn để mang chúng vào trong vùng hình hộp. Thực tế, hệ số tỷ lệ được cần sẽ tỷ lệ nghịch với khoảng cách đến cửa sổ. Đối với một tâm chiếu ở khoảng cách d phía sau cửa sổ, hệ số tỷ lệ được cần là d/(z+d), với z được xem như khoảng cách từ điểm đến cửa sổ. Ma trận cho phép biến đổi tỷ lệ này là

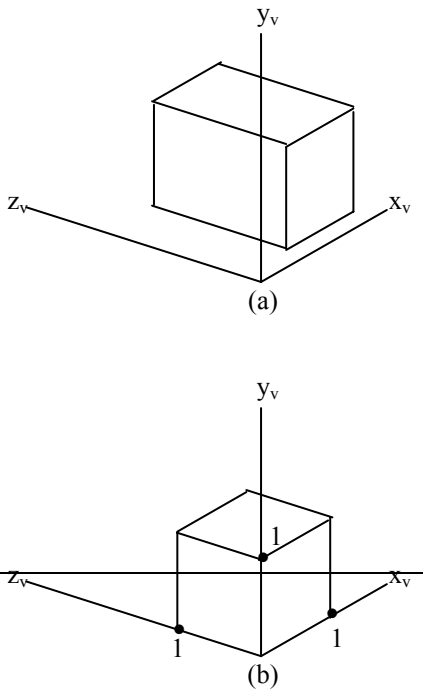
$$\begin{bmatrix} S & 0 & 0 & 0 \\ 0 & S & 0 & 0 \\ 0 & 0 & 1 & 0 \\ (1-S)x_F & (1-S)y_F & 0 & 1 \end{bmatrix} \qquad (6\text{-}13)$$

Ở đây hệ số tỷ lệ là S = d(z + d). Tất cả các giá trị tọa độ x và y trong cảnh được biến đổi tỷ lệ bởi phép biến đổi này. Các điểm trong không gian này được ánh xạ thành các điểm trong hình hộp mà không có sự thay đổi giá trị của z (hình 6-29)



Việc chuyển đổi thành một hình hộp còn có một lợi ích quan trọng khác. Các phép biến đổi

Hình 6-30
Biến đổi một hình hộp thông thường (a) thành một hình lập phương đơn vị (b).



được áp dụng thực hiện một số lượng lớn công việc cần thiết để chiếu các điểm nguyên thủy lên mặt phẳng chiếu. Ví dụ, phép biến đổi để chuyển hình chóp thành một hình hộp nhất thiết phải thực hiện phép chiếu phối cảnh. Các vị trí tọa độ được biến đổi sang các giá trị chiếu x và y, tuy nhiên ta giữ nguyên các giá trị khác không z. Các điểm nằm trong phạm vi các giá trị z của hai mặt gần và xa có thể được chiếu bằng cách đơn giản vứt bỏ tọa độ z. Vì thế bằng việc biến đổi không gian quan sát thành hình hộp thông thường, thao tác chiếu được rút gọn thành một phép chiếu trực giao đơn giản.

Chúng ta có thể làm các thao tác quan sát hiệu quả hơn. Việc biến đổi không gian quan sát thành hình hộp thông thường (cái được làm tương tự như thao tác chiếu) xảy ra liền sau việc ánh xạ từ tọa độ thế giới thực sang hệ tọa độ quan sát. Nếu chúng ta kết hợp các ma trận lại để làm một lúc một dãy các thao tác này, mỗi vị trí tọa độ có thể được chuyển từ vị trí của nó trong tọa độ thế giới thực sang vị trí tương ứng trong hình hộp chỉ là một bước thực hiện.

Vài gói đồ họa thực hiện việc clipping bằng cách dùng hình hộp thông thường như vừa được trình bày. Các phần của đối tượng trong phạm vi hình hộp được chiếu đến mặt trước (front plane) và sau đó được ánh xạ đến vùng quan sát hai chiều.

Các gói đồ họa khác thì ánh xạ hình hộp này vào một **hình lập phương đơn vị (unit cube)** (hình 6-30) trước khi clipping và chiếu.

Hình lập phương đơn vị là một không gian được xác định bởi các mặt sau:

x =0, x= 1, y = 0, y = 1, z = 0, z = 1 (6-14)

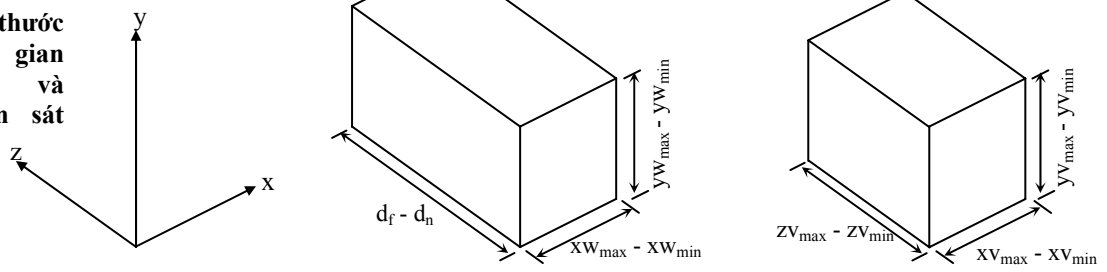
Vì hình lập phương đơn vị được định nghĩa bởi các giá trị trong đoạn [0..1], nó có thể được xem như một **không gian quan sát chuẩn hóa (normalized view volume).** Cũng như với hình hộp, khi các thành phần nằm trong không gian vừa được ánh xạ đến mặt trước (front plane), các điểm đó sẽ được ánh xạ đến một vùng quan sát hai chiều.

Như một chọn lựa khác, hình hộp thông thường, được xác định bởi cửa sổ mặt quan sát, có thể được ánh xạ đến một **vùng quan sát ba chiều (three-dimensional viewport)** trước khi clipping. Vùng quan sát này là một hình hộp thông thường được định nghĩa trong hệ tọa độ chuẩn hóa. Việc ánh xạ từ cửa sổ-đến–vùng quan sát trong không gian ba chiều cần được thực hiện với một phép biến đổi kết hợp tỷ lệ và tịnh tiến tương tự như với việc ánh xạ từ cửa sổ-đến–vùng quan sát trong không gian hai chiều. Chúng ta có thể biểu diễn ma trận biến đổi ba chiều của tập các thao tác này như sau:

$$\begin{bmatrix} D_x & 0 & 0 & 0 \\ 0 & D_y & 0 & 0 \\ 0 & 0 & D_z & 0 \end{bmatrix} \quad (6\text{-}15)$$

Các tham số D_x, D_y, và D_z là các tỷ lệ về kích thước của vùng quan sát so với không gian quan sát hình hộp theo các hướng x, y, và z (xem hình 6-31):

Hình 6-31
Các kích thước của không gian quan sát và vùng quan sát



Hướng các trục tọa độ

Không gian quan sát được xác định bởi các tọa độ cửa sổ và các mặt gần và xa

Vùng quan sát ba chiều

$$D_x = \frac{xv_{max} - xv_{min}}{xw_{max} - xw_{min}}$$
$$D_y = \frac{yv_{max} - yv_{min}}{yw_{max} - yw_{min}} \qquad (6\text{-}16)$$
$$D_z = \frac{zv_{max} - zv_{min}}{d_f - d_n}$$

ở đây các biên của không gian quan sát được xây dựng bởi các giới hạn cửa sổ (xw_{min}, xw_{max}, yw_{min}, yw_{max}), và các vị trí d_n và d_f của các mặt gần và xa. Các biên của vùng quan sát được thiết đặt với các giá trị tọa độ xv_{min}, xv_{max}, yv_{min}, yv_{max}, zv_{min}, và zv_{max}. Các tham số bổ sung K_x, K_y, và K_z trong phép biến đổi là:

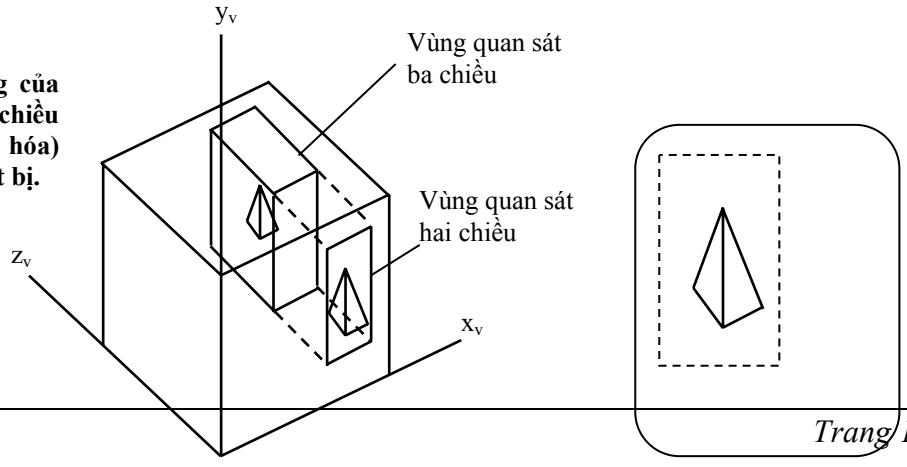
$$K_x = xv_{min} - xw_{min} * D_x$$
$$K_y = yv_{min} - yw_{min} * D_y \qquad (6\text{-}17)$$
$$K_z = zv_{min} - d_n * D_z$$

Việc ánh xạ từ cửa sổ-đến-vùng quan sát được thực hiện trước khi clipping như trong hình 6-32.

Hình 6-32 Thực hiện các phép biến đổi hệ quan sát để các thao tác có thể được nối kết vào một ma trận biến đổi đơn, được áp dụng trước khi clipping.

Thuận lợi của cách làm này là ma trận biến đổi chuẩn hóa (từ không gian quan sát-đến-ánh xạ vào vùng quan sát) có thể được kết hợp với ma trận biến đổi các tọa độ trong hệ thế giới thực sang các vị trí trong hình hộp. Ma trận kết quả biến đổi các vị trí trong phạm vi hệ tọa độ thế thực thành các điểm chiếu x và y trong vùng quan sát. Mỗi tọa độ của cảnh gốc cần được tịnh tiến chỉ một lần. Các điểm được tịnh tiến này bị clipping bởi vùng quan sát. Các giá trị x và y của các điểm trong không gian quan sát sau đó được biến đổi đến các hệ tọa độ thiết bị để hiển thị (xem hình 6-33).

Hình 6-33
Ánh xạ phần bên trong của một vùng quan sát ba chiều (trong hệ tọa độ chuẩn hóa) đến các tọa độ trên thiết bị.



Clipping dựa vào một không gian quan sát được chuẩn hóa

Các bề mặt có thể bị cắt khỏi các biên vùng quan sát bằng các thủ đơn giản hơn trong đồ họa hai chiều. Dù là các thủ tục clipping đường hay cliping đa giác đều có thể được sửa lại cho thích hợp với clipping một vùng quan sát trong ba chiều. Các mặt cong được xử lý bằng cách dùng các phương trình mặt biên kết hợp với việc xác định đường cắt với các mặt của hình hộp. Bây giờ chúng ta xem các thủ tục clipping hai chiều được thay đổi thế nào để dùng cho ba chiều.

Các khái niệm trong hai chiều về các mã vùng có thể được mở rộng cho ba chiều bằng việc xem xét các vị trí phía trước và phía sau vùng quan sát ba chiều, cũng như các vị trị bên trái, bên phải, phía dưới, hoặc phía trên không gian. Đối với clipping hai chiều, chúng ta đã dùng mã vùng nhị phân bốn bit để xác định vị trí của các điểm đầu mút đoạn thẳng có quan hệ với các biên cửa sổ thế nào. Đối với các điểm ba chiều, chúng ta cần mở rộng mã vùng thành sáu bit. Mỗi điểm trong cảnh khi đó được gán một mã vùng sáu bit để xác định mối quan hệ với các mặt biên của vùng quan sát. Với một điểm đầu mút đoạn thẳng ở vị trí (x, y, z), ta gán các vị trí bit trong mã vùng từ phải sang trái như sau:

bit 1 =1 nếu $x < xv_{min}$ (left)

bit 2 =1 nếu $x > xv_{max}$ (right)

bit 3 =1 nếu $y < yv_{min}$ (below)

bit 4 =1 nếu $y > yv_{max}$ (above)

bit 5 =1 nếu $z < zv_{min}$ (front)

bit 6 =1 nếu $z > zv_{max}$ (back)

Ví dụ, một mã vùng 101000 chỉ ra rằng một điểm thì ở trên và phía sau vùng quan sát, trong khi đó mã vùng 000000 chỉ ra rằng một điểm nằm trong không gian quan sát.

Một đoạn thẳng có thể được xác định ngay là hoàn toàn nằm trong vùng quan sát nếu cả hai điểm đầu mút của nó đều có mã vùng là 000000. Nếu điểm đầu mút nào không có mã vùng 000000, chúng ta thực hiện phép logic *and* lên hai mã đầu mút. Kết quả phép toán *and* sẽ khác 0 đối với các đoạn thẳng hoàn toàn nằm ngoài không gian quan sát. Nếu

chúng ta không thể xác định được một đoạn thẳng là hoàn toàn nằm trong hay hoàn toàn nằm ngoài không gian, ta sẽ đi tìm giao điểm với các mặt biên của không gian.

Như trong clipping đường hai chiều, chúng ta dùng các giao điểm được tính của đường với các mặt của vùng quan sát để xác định xem phần nào của đoạn thẳng bị vứt bỏ. Phần được giữ lại của đoạn sẽ được kiểm tra với các mặt khác, và chúng ta tiếp tục đến khi xác định được là đoạn bị vứt bỏ hoàn toàn hay đến khi thấy nó nằm bên trong không gian.

Việc xác định các giao điểm trong clipping đường, cũng như trong các thủ tục clipping đa giác, nên được làm sao cho hiệu quả. Các phương trình của các đoạn ba chiều được biểu diễn thuận tiện theo dạng tham số. Với một đoạn có hai điểm đầu mút $P_1 = (x_1, y_1, z_1)$ và $P_2 = (x_2, y_2, z_2)$, chúng ta có thể viết phương trình tham số là

$$\begin{aligned} x &= x_1 + (x_2 - x_1)u \\ y &= y_1 + (y_2 - y_1)u \\ z &= z_1 + (z_2 - z_1)u \end{aligned} \qquad (6\text{-}18)$$

Tọa độ (x, y, z) biểu diễn cho một điểm bất kỳ trên đoạn thẳng giữa hai điểm đầu mút, và các tham số u thay đổi từ 0 đến 1. Giá trị u =0 tạo ra điểm P_1, u=1 cho điểm P_2.

Để tìm giao điểm của một đường với một mặt của vùng quan sát, chúng ta thay thế giá trị tọa độ, cái là giá trị hằng của mặt đó, vào phương trình tham số 12-18 và giải tìm u. Cho trường hợp này, giả sử chúng ta đang xét một đường với mặt trước (front plane) của vùng quan sát. Khi đó $z = zv_{min}$, và

$$u = \frac{zv_{\min} - z_1}{z_2 - z_1} \qquad (6\text{-}19)$$

Khi giá trị u được tính bởi phương trình 12-19 không nằm trong đoạn [0..1], đều này có nghĩa là đoạn thẳng không cắt mặt trước ở bất kỳ điểm nào nằm giữa hai đầu mút P_1 và P_2 (đường A trong hình 6-34). Nếu giá trị u được tính nằm trong đoạn [0..1], chúng ta tính tọa độ giao điểm x và y như sau

$$\begin{aligned} x_1 &= x_1 + (x_2 - x_1)\left(\frac{zv_{\min} - z_1}{z_2 - z_1}\right) \\ y_1 &= y_1 + (y_2 - y_1)\left(\frac{zv_{\min} - z_1}{z_2 - z_1}\right) \end{aligned} \qquad (6\text{-}20)$$

```
     St[2,1] :=  2;    St[2,2] :=  2.7;    St[2,3] :=   0;
     St[3,1] :=  2;    St[3,2] := -2.7;    St[3,3] :=   0;
     St[4,1] :=  2;    St[4,2] := -2.7;    St[4,3] :=  -2;
     St[5,1] := -2;    St[5,2] := -2.7;    St[5,3] :=  -2;
     St[6,1] := -2;    St[6,2] :=  2.7;    St[6,3] :=  -2;
     St[7,1] := -2;    St[7,2] :=  2.7;    St[7,3] :=   0;
     St[8,1] :=  0;    St[8,2] :=  1.7;    St[8,3] :=   2;
     St[9,1] :=  0;    St[9,2] := -1.7;    St[9,3] :=   2;
     St[10,1] := -2;    St[10,2] := -2.7;    St[10,3] :=   0;
  End;

Procedure LectureFaces;
  Begin
   NF := 9;
   FC[1,0] := 4; FC[1,1] := 1; FC[1,2] := 2; FC[1,3] := 3; FC[1,4] := 4;
   FC[2,0] := 4; FC[2,1] := 1; FC[2,2] := 6; FC[2,3] := 7; FC[2,4] := 2;
   FC[3,0] := 3; FC[3,1] := 2; FC[3,2] := 7; FC[3,3] := 8;
   FC[4,0] := 4; FC[4,1] := 2; FC[4,2] := 8; FC[4,3] := 9; FC[4,4] := 3;
   FC[5,0] := 4; FC[5,1] := 1; FC[5,2] := 4; FC[5,3] := 5; FC[5,4] := 6;
   FC[6,0] := 4; FC[6,1] := 7; FC[6,2] :=10; FC[6,3] := 9; FC[6,4] := 8;
   FC[7,0] := 3; FC[7,1] := 3; FC[7,2] := 9; FC[7,3] :=10;
   FC[8,0] := 4; FC[8,1] :=10; FC[8,2] := 5; FC[8,3] := 4; FC[8,4] := 3;
   FC[9,0] := 4; FC[9,1] := 5; FC[9,2] :=10; FC[9,3] := 7; FC[9,4] := 6;
 End;

ProCedure VecteurVision(St1, St2, St3:integer;  Var V1, V2, V3 : rEal);
 Begin
   V1 := O1 - St[St1,1]; V2 := O2 - St[St1,2]; V3 := O3 - St[St1,3];
 End;

Procedure VecteurNormal(St1, ST2, St3:integer; Var N1, N2, N3 : Real);
 Var P1, P2, P3, Q1, Q2, Q3 : Real;
 Begin
     P1 := ST[St2,1] - ST[St1,1];
     P2 := ST[St2,2] - ST[St1,2];
     P3 := ST[St2,3] - ST[St1,3];
     Q1 := ST[St3,1] - ST[St1,1];
     Q2 := ST[St3,2] - ST[St1,2];
     Q3 := ST[St3,3] - ST[St1,3];
   N1 := P2*Q3 - Q2*P3;
   N2 := P3*Q1 - P1*Q3;
   N3 := P1*Q2 - Q1*P2;
 End;

Function ProDuitScalaire(V1, V2, V3, N1, N2, N3: Real):Real;
 Begin
    ProDuitScalaire := V1*N1 + V2*N2 + V3*N3
 End;

Procedure DessineObject;
   Var F, St1, St2, St3, NS, No : Integer;
       V1, V2, V3, N1, N2, N3 : Real;
       X, Y, Z,  XO, YO, ZO : Real;

  Procedure DessineFace;
   Var S : Integer;
   Begin
```

```
      NS := FC[f,0];
      For S := 1 To NS Do
        Begin
          No := FC[F,S]; X := ST[No,1]; Y := ST[No,2]; Z := ST[No,3];
          If S = 1  Then Begin
                           DePlaceEn(X, Y, Z);
                XO := X; YO := Y; ZO := Z;
               End
                     Else Tracevers(X, Y, Z);
        End;
      TraceVers(XO, YO, ZO);
    End;

  Begin
    FillChar(FFF, Sizeof(fff), #0);
    SetLineStyle(DottedLn, 0, NormWidth);
    SetColor(LightRed);
    For F := 1 to NF Do
      Begin
       St1 := Fc[F,1]; St2 := Fc[F,2]; St3 := Fc[F,3];
       VecteurVision(St1, St2, St3, V1, V2, V3);
       VecteurNormal(St1, St2, St3, N1, N2, N3);
       If ProDuitScalaire(V1, V2, V3, N1, N2,  N3) <= 0 then
          Begin
            If Pointille Then DessineFace;
            FFF[f] := True;
          End;
      End;
    SetLineStyle(SolidLn, 0, NormWidth);
    SetColor(White);
    For F := 1 to Nf Do
      If Not FFF[F] Then DessineFace;
  End;

Procedure CoordonneeOeil;
  Begin
    InitialiseProjection;
    O1 := Rho * Aux7; O2 := Rho * Aux8; O3 := Rho * Aux2;
  End;

Procedure Affichage;
  Var S1, S2, S3, S4, S5 : String;
  Begin
    Cloture(0, MaxX, 0, 23);  ClearViewPort;
    SetTextStyle(SmallFont, HorizDir, 4);
    SetColor(14);
    Str(Theta:3:1, S2); Str(Phi:3:1, S3); Str(DE:3:1, S4);
    IF Projection = Perspective
    Then Begin
           Str(Rho:3:1, S1);
           OutTextXY(80, 0,'Chieu Phoi Canh: Rho = '+S1+'   Theta = '+S2+
               +'   Phi = '+S3+'   Ecran = '+S4);
         End
    Else OutTextXY(80,0, 'Chieu Song Song: Rho = infini  Theta = '+S2+
               +'   Phi = '+S3+'   Ecran = '+S4);
    Str(MaxX, S1); Str(MaxY, S2);
    OutTextXY(5,  0, S1+' x '+S2);
    OutTextXY(5, 12, 'Control: ArrowKey, E, A, +, -, T, C, F-Fine');
    Cloture(0, MaxX, 24, MaxY);
  End;

Procedure Commandes;
```

```
Const RhoPara = 1e20;
Var RhoPersp, DEPersp, DEPara : Real;
    Ch : Char;
 Begin
   VueDeDepart;
   DEPara := 30;
   CoorDonneeOeil;
   DessineObject;
   Affichage;
   RePeat
     Ch := UpCase(Readkey);
     IF Ch = #0 Then Ch := UpCase(Readkey);
     If Ord(Ch)  In [72,80,75,77,69,65,43,45,84,67,61,95]
     Then
       Begin
         ClearDevice;
         Case Ord(Ch) Of
           72 : Phi := Phi + IncAng;
           80 : Phi := Phi - IncAng;
           75 : Theta := Theta + IncAng;
           77 : Theta := Theta - IncAng;
           69 : Rho := Rho + IncRho;
           65 : Rho := Rho - IncRho;
           43,61 : DE :=  DE + IncEcran;
           45,95 : DE :=  DE - IncEcran;
           84 : Pointille := not (Pointille);
           67 : If Projection = Perspective
                Then Begin
                       RhoPersp := Rho;
                       DEPersp := DE;
                       Projection := Parallele;
                       Rho := RhoPara;
            DE := DEPara;
                     End
                Else Begin
                       DEPara := DE;
                       Projection := Perspective;
                       Rho := RhoPersp;
            De := DePersp;
                     End;
         End;
         CoordonneeOeil;
         DessineObject;
         Affichage
       End;
   Until (CH = 'F') Or (ch = #13) Or (ch =#27);
   EcranTexte;
 End;

Begin
  EcranGraphique('');
  Cloture(0, MaxX, 0, MaxY);
  LectureSommets;
  LectureFaces;
  VueDeDepart;
  Commandes;
End.
```

(function) *view_matrix*, được cung cấp các tọa độ của điểm quan sát, pháp vector, và vector nhìn lên (view up vector).

9. Mở rộng các thủ tục trong bài tập 8 để thu được một phép chiếu song song (đã được xác định) của đối tượng lên một cửa sổ được định nghĩa trên mặt xy của hệ quan sát. Sau đó biến đổi cửa sổ đến một vùng quan sát trên màn ảnh. Giả sử rằng các đối tượng thì ở phía trước mặt phẳng quan sát và rằng không có việc clipping bởi một không gian quan sát nào được thực hiện.

10. Mở rộng các thủ tục trong bài tập 8 để thu được một phép chiếu phối cảnh (đã được xác định) của đối tượng lên một cửa sổ được định nghĩa trên mặt xy của hệ quan sát. Sau đó biến đổi cửa sổ đến một vùng vùng quan sát trên màn ảnh. Giả sử rằng các đối tượng thì ở phía trước mặt phẳng quan sát và rằng không có việc clipping bởi một không gian quan sát nào được thực hiện.

11. Nghĩ ra một thuật toán để cắt (clip) các đối tượng trong một quang cảnh bởi một hình chóp cụt đã được định nghĩa. So sánh các phép toán được cần trong thuật toán này với các phép toán được cần trong thuật toán cắt quang cảnh bởi một hình hộp thông thường.

12. Viết một chương trình thực hiện chiếu phối cảnh một hình chóp cụt thành một hình hộp thông thường.

13. Thay đổi thuật toán clipping đường Liang-Barsky hai chiều để cắt (clip) các đường ba chiều bởi một bởi một hình hộp (đã được xác định).

14. Mở rộng thuật toán của bài tập 13 để cắt một khối đa diện (đã được xác định) bởi một hình hộp.

15. Đối với cả hai phép chiếu song song và phối cảnh, hãy thảo luận các điều kiện để việc clipping ba chiều được thực hiện trước, phép chiếu lên mặt phẳng chiếu được thực hiện sau có thể tương đương với việc chiếu trước rồi thực hiện clipping sau.

16. Dùng bất kỳ thủ tục clipping nào, viết một chương trình thực hiện một phép biến đổi hệ quan sát hoàn chỉnh từ tọa độ thế giới thực sang vùng quan sát cho một phép chiếu song song trực giao của một đối tượng.

17. Mở rộng thủ tục của bài tập 16 để thực hiện một phép chiếu song song (được xác định bất kỳ) của một đối tượng lên một vùng quan sát đã được định nghĩa.

18. Phát triển một chương trình để cài đặt một hướng quan sát hoàn chỉnh cho một phép chiếu phối cảnh. Chương trình phải biến đổi sự xác định hệ tọa độ thế giới thực của một đối tượng lên một vùng quan sát hai chiều đã được định nghĩa để hiển thị lên một phần của màn hình video.

19. Cài đặt các hàm *set_view_representation* và *set_view_index* để thực hiện một hướng chiếu (được xác định bất kỳ) trên một đối tượng được định nghĩa trong hệ tọa độ thế giới thực để thu được sự hiển thị vùng quan sát trên màn hình.

20. Thay đổi các thủ tục trong bài tập 19 để cho phép clipping bởi mặt phẳng của không gian quan sát bất kỳ. Điều này có thể được thực hiện với các tham số bổ sung đến tập các điều kiện clipping cho mỗi mặt phẳng là cắt (clip) hoặc không cắt (noclip).

thứ tự các đường, mặt, và các đối tượng trong ảnh dựa vào khoảng cách từ chúng đến mặt phẳng quan sát. Phương pháp cố kết được dùng để thu được thuận lợi của sự cân đối trong ảnh. Một đường quét riêng lẻ có thể được dùng để chứa đựng các giá trị về độ sáng của các pixel, và các mẫu đường quét (scan-line patterns) thường thay đổi ít từ đường này đến đường kế tiếp. Các khung nối kết động chứa các thay đổi chỉ trong vùng lân cận của các đối tượng di chuyển. Và các mối quan hệ cố định thường được xây dựng giữa các đối tượng và các mặt trong ảnh.

Một phương pháp không gian đối tượng đơn giản để xác định **các mặt sau (back faces)** đối tượng là dựa vào các phương trình mặt:

$$Ax + By + Cz + D = 0 \qquad (7\text{-}1)$$

Bất kỳ điểm (x', y', z') trên hệ tọa độ bàn tay trái sẽ ở "phía trong" mặt này nếu nó thỏa bất phương trình:

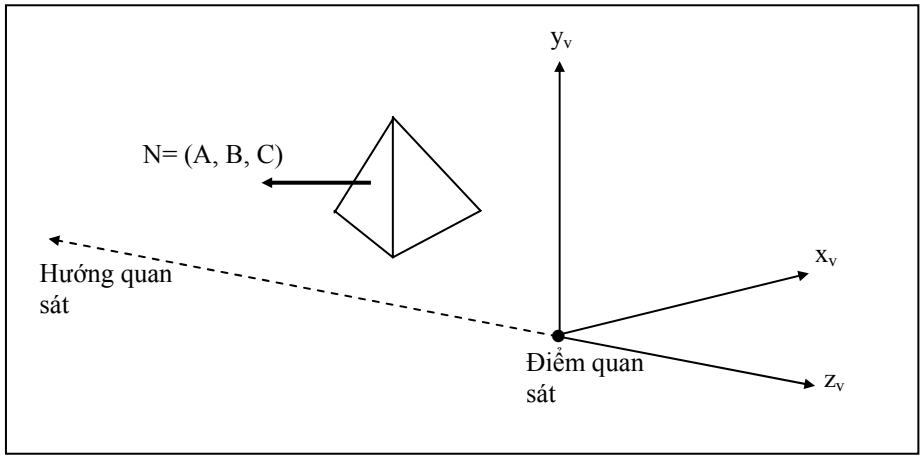
$$Ax' + By' + Cz' + D < 0 \qquad (7\text{-}2)$$

Nếu điểm (x', y', z') là vị trí quan sát (viewing position), khi đó bất kỳ mặt phẳng nào làm cho bất phương trình 7-2 đúng phải là một mặt ở đằng sau. Tức là, nó là mặt ta không thể nhìn thấy từ vị trí quan sát.

Chúng ta có thể thực hiện một cách kiểm tra mặt đằng sau đơn giản hơn bằng cách nhìn ở pháp vector (normal vector) của mặt có phương trình 7-1. Pháp vector này có tọa độ Descartes (A, B, C). Trong hệ tọa độ bàn tay phải với hướng quan sát cùng chiều với trục z_v âm (xem hình 7-1), pháp vector có tham số C song song với hướng quan sát. Nếu C<0, pháp vector chỉ ra xa khỏi vị trí quan sát, và mặt phải là mặt ở đằng sau.

Hình 7-1

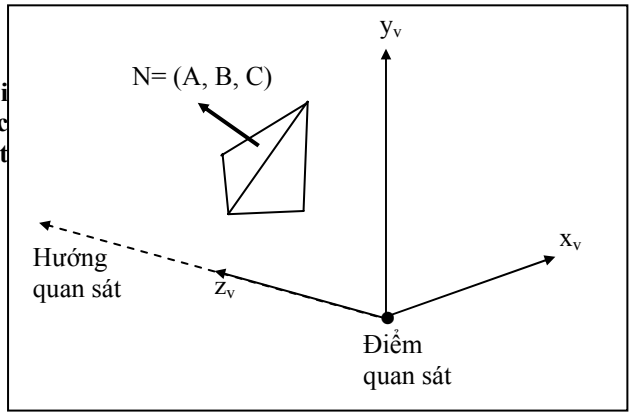
Một mặt phẳng với tham số C < 0 trong hệ quan sát bàn tay phải được xác định như mặt ở đằng sau khi hướng quan sát cùng chiều với trục z_v âm.



Các phương pháp tương tự có thể được dùng trong các gói đồ họa, nơi sử dụng hệ quan sát bày tay trái. Trong các gói đồ họa này, các tham số A, B, C, và D có thể được tính từ tọa độ các đỉnh được xét theo chiều kim đồng hồ (thay vì hướng ngược chiều kim đồng hồ được dùng trong hệ tọa độ bàn tay phải). Bất phương trình 7-2 sau đó cho một kiểm tra hợp lệ đối với các điểm nằm phía trong. Cũng như vậy, các mặt ở đằng sau có các pháp vector chỉ ra xa khỏi vị trí quan sát và được xác định bởi C>0 khi hướng quan sát cùng hướng với trục z_v dương (xem hình 7-2). Trong tất cả các thảo luận sau này trong chương, chúng ta giả sử rằng hệ quan sát bàn tay trái được dùng.

Hình 7-2

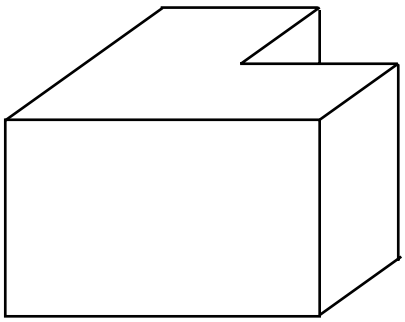
Trong hệ quan sát bàn tay trái, khi hướng quan sát cùng chiều với trục z_v dương, một mặt ở đằng sau là mặt với tham số C>0.



Bằng việc kiểm tra tham số C ở mỗi mặt của đối tượng, ta có thể xác định được ngay tất cả các mặt ở đằng sau. Đối với một khối đa diện lồi đơn lẽ, như hình kim tự tháp trong hình 7-1, việc kiểm tra này xác định tất cả các mặt bị che khuất trên đối tượng, bởi vì mỗi mặt thì là hoàn toàn được nhìn thấy hoặc hoàn toàn bị che khuất. Đối với các đối tượng khác, các kiểm tra phức tạp hơn cần được thực hiện để xác định xem các mặt là bị che khuất hoàn toàn hay chỉ bị che khuất một phần (xem hình 7-3). Tương tự, chúng ta cần xác định xem các đối tượng là có một phần hay toàn bộ bị che khuất bởi các đối tượng khác. Một cách tổng quát, việc khử mặt khuất sẽ loại bỏ khoảng một nữa số mặt trong một ảnh khi thực hiện các phép kiểm tra tính nhìn thấy được sau này.

Hình 7-3

Ảnh một đối tượng với một mặt bị che khuất một phần



Chúng ta có thể tổng kết các bước của thuật toán vùng đệm độ sâu như sau:

1. Khởi tạo vùng đệm độ sâu và vùng đệm làm tươi để với tất cả các vị trí *(x,y)*, *depth(x, y) =1* và *refresh(x, y) = background*.
2. Đối với mỗi vị trí trên mỗi mặt, so sánh các giá trị độ sâu với các giá trị độ sâu được lưu trước đó trong vùng đệm độ sâu để xác định tính chất nhìn thấy được.
    a. Tính giá trị z cho mỗi vị trí (x, y) trên mặt.
    b. Nếu $z < depth(x, y)$ thì đặt lại $depth(x, y) = z$ và $refresh(x, y) = i$, với i là giá trị độ sáng trên mặt ở vị trí (x, y).

Trong bước cuối cùng, nếu z không nhỏ hơn giá trị trong vùng đệm độ sâu ở vị trí đó, điểm không được nhìn thấy. Khi quá trình này được hoàn thành cho tất cả các mặt, vùng đệm độ sâu chứa các giá trị z của các mặt nhìn thấy được và vùng đệm làm tươi chỉ chứa các giá trị độ sáng của các mặt nhìn thấy được đó.

Các giá trị độ sâu cho một vị trí (x, y) được tính từ phương trình của mỗi mặt:

$$z = \frac{-Ax - By - D}{C} \qquad (7\text{-}3)$$

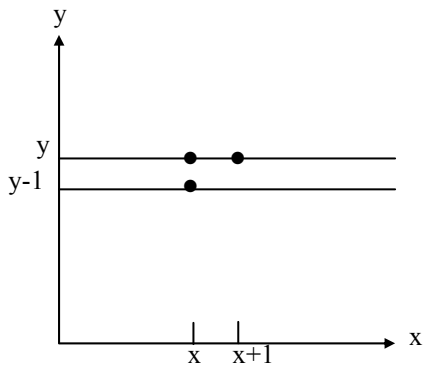
Với mỗi đường quét bất kỳ (xem hình 7-5), các tọa độ x trên cùng đường quét sai khác nhau 1, và các giá trị y giữa hai đường quét cũng sai khác nhau 1. Nếu độ sâu của vị trí (x,y) được xác định là z, khi đó độ sâu z' của vị trí kế tiếp (x+1, y) dọc theo theo đường quét có được từ phương trình 13-3 như sau:

$$z' = \frac{-A(x+1) - By - D}{C}$$

hoặc

$$z' = z - \frac{A}{C} \qquad (7\text{-}4)$$

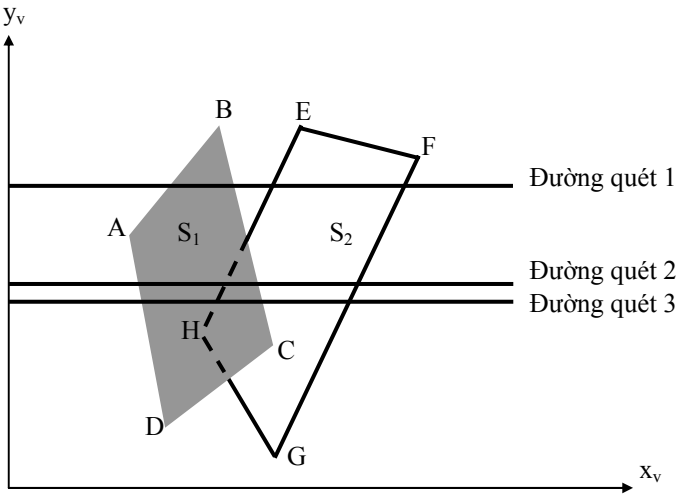
Hình 7-5
Từ vị trí (x, y) trên một đường quét, vị trí kế tiếp qua phải có tọa độ (x+1, y), và vị trí liền ngay bên dưới trên dòng kế tiếp có tọa độ (x, y-1)



bảng đa giác để xác định mặt nào chứa mỗi cạnh. Bảng đa giác chứa các hệ số của phương trình mỗi mặt, thông tin về độ sáng cho các mặt, và có thể chỉ đến bảng các cạnh.

Để dễ dàng nghiên cứu các mặt cắt một đường quét được cho, chúng ta cài đặt một danh sách động chứa các cạnh lấy thông tin trong bảng cạnh. Danh sách động này sẽ chỉ chứa các cạnh cắt đường quét hiện hành, được sắp xếp theo thứ thự x tăng. Và, chúng ta định nghĩa một cờ (flag) cho mỗi mặt, cờ này được đặt là on hay off để chỉ ra mỗi vị trí nằm dọc trên đường quét là nằm trong hay nằm ngoài mặt. Các đường quét được xử lý từ trái sang phải. Ở biên bên trái nhất của một mặt, cờ của mặt là on; và ở biên bên phải nhất cờ là off.

Hình 7-6 minh họa phương pháp scan-line để xác định vị trí các phần nhìn thấy được dọc theo một đường quét. Danh sách động cho đường quét 1 (scan line 1) lấy thông tin từ bảng các cạnh đối với các cạnh AB, BC, HE, và FG. Đối với các vị trí dọc theo đường quét này giữa các cạnh AB và BC, chỉ cờ mặt S_1 là on. Do đó, không phép tính độ sâu nào là cần thiết, và thông tin độ sáng của mặt S_1 được lấy từ bảng đa giác để nhập vào vùng làm tươi. Tương tự, các cạnh HE và FG, chỉ cờ cho mặt S_2 là on. Không vị trí nào khác dọc theo đường quét 1 cắt các mặt, vì vậy các giá trị độ sáng trong các vùng khác được đặt là độ sáng nền.Độ sáng nền có thể được nạp vào trong vùng đệm trong một thủ tục khởi tạo.



Hình 7-6
Các đường quét cắt hình chiếu của hai mặt S_1 và S_2 trên mặt phẳng chiếu. Các đường nét đứt chỉ ra rằng đó là biên của mặt bị che khuất.

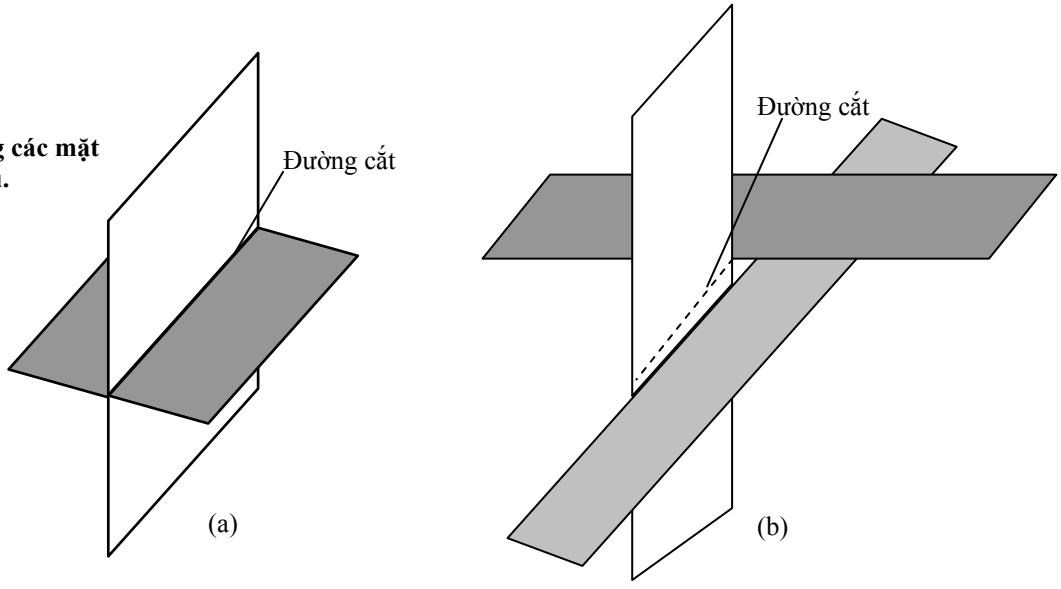
Danh sách động cho đường quét 2 và 3 trong hình 7-6 chứa các cạnh DA, HE, BC, và FG. Dọc theo đường quét 2 từ cạnh DA đến cạnh EH, chỉ cờ của mặt S_1 là on.

Nhưng giữa HE và BC, các cờ cho cả hai mặt là on. Trong đoạn này, các tính toán độ về độ sâu phải được thực hiện bằng cách dùng tham số mặt của các mặt. Trong ví dụ này, độ sâu của mặt S_1 được được giả thiết là nhỏ hơn của mặt S_2, vì vậy độ sáng của mặt S_1 được nạp vào trong vùng đệm làm tươi đến khi biên BC được gặp. Sau đó cờ của mặt S_1 trở thành off, và độ sáng của mặt S_2 được lưu cho đến cạnh FG được đi qua.

Chúng ta có thể tận dụng các thuận lợi có được từ quan hệ cố kết dọc theo các đường quét khi chúng ta đi từ đường này đến đường kế tiếp. Trong hình 7-6, đường quét 3 có danh sách động giống như của dòng 2. Bởi vì không có thay đổi nào xảy ra tại các giao điểm đường, ta không cần tính lại độ sâu giữa các cạnh HE và BC. Hai mặt phải có hướng tương tự như được xác định trên đường quét 2, vì vậy độ sáng cho mặt S_1 không cần nhập lại.

Dù có bao nhiêu mặt được xếp chồng lên nhau, ta cũng có thể dùng phương pháp scan-line này. Các cờ cho các mặt được đặt để chỉ rõ xem một vị trí là bên trong hay bên ngoài, và các tính toán về độ sâu được thực hiện khi các mặt xếp chồng lên nhau. Trong vài trường hợp, các mặt có thể che khuất nhau một cách luân phiên (xem hình 7-7). Khi các phương pháp cố kết được dùng, ta cần cẩn thận để lưu vết phần nào của mặt là thấy được trên mỗi đường quét. Một cách để xử lý trường hợp này là phân chia các mặt. Cụ thể, mặt ABC trong hình 7-8 có thể được chia làm ba mặt ABED, DEGF, và CFG. Mỗi mặt nhỏ có thể được xét như một mặt riêng biệt, để mà không có hai mặt nào là bị che khuất và nhìn thấy một cách luân phiên.

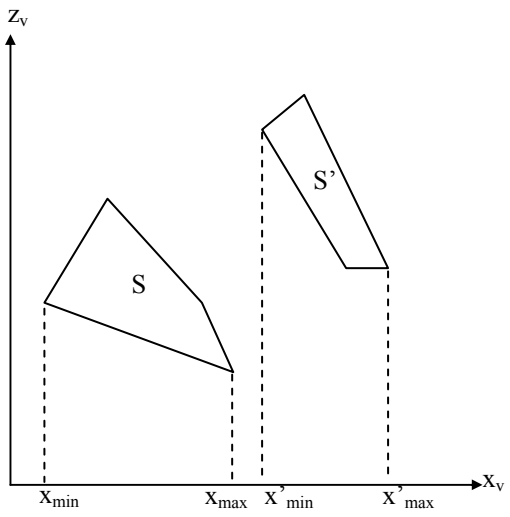
Hình 7-7
Các ví dụ về hướng các mặt che khuất lẫn nhau.



vùng đệm làm tươi. Với mỗi mặt kế tiếp (xét theo thứ tự độ sâu giảm dần), ta "sơn" các độ sáng của mặt lên vùng đệm làm tươi (phủ lên các độ sáng của mặt được xử lý trước đó).

Việc sơn các mặt đa giác lên vùng đệm làm tươi dựa theo độ sâu được thực hiện trong vài bước. Đầu tiên, các mặt được sắp xếp dựa vào giá trị z lớn nhất của mỗi mặt. Mặt với độ sâu lớn nhất (gọi là S) sau đó được so sánh với các mặt còn lại trong danh sách để xác định xem có bất kỳ sự chồng độ sâu nào không (nằm chồng lên nhau). Nếu không có sự chồng độ sâu nào xảy ra, S được vẽ ra (vẽ ra theo từng đường quét). Trong hình 7-9 trình bày hai mặt không có sự chồng độ sâu (hai mặt không nằm chồng nhau), hình chiếu của chúng lên mặt phẳng xz. Xử lý này sau đó được lặp lại cho mặt kế tiếp trong danh sách. Khi không có sự chồng độ sâu nào xảy ra, mỗi mặt sẽ được xử lý theo thứ tự độ sâu đó cho đến khi tất cả đều được quét qua. Nếu có một sự chồng độ sâu được phát hiện ở bất kỳ điểm nào trong danh sách, ta cần làm vài so sánh bổ sung để xác định xem mặt nào nên được sắp xếp lại.

Với mỗi mặt nằm chồng với S, ta thực hiện các phép kiểm tra sau. Chỉ cần một trong số các phép kiểm tra này là đúng (true), ta không cần sắp lại vị trí mặt đó. Các phép kiểm tra được lập danh sách theo mức độ khó tăng dần:



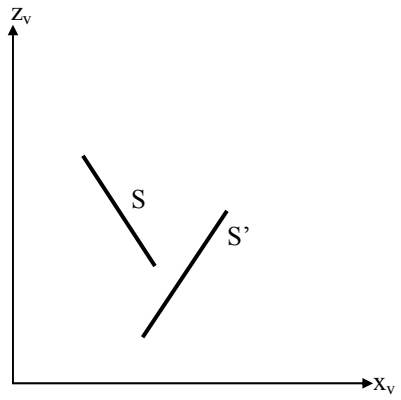
Hình 7-10
Hai mặt không có sự chồng độ sâu theo hướng x.

1. Trên mặt phẳng xy, các hình chữ nhật bao quanh hai mặt không chồng lên nhau.
2. Mặt S thì ở "phía ngoài" mặt nằm chồng, so sánh dựa vào mặt phẳng quan sát.
3. Mặt nằm chồng thì ở "phía trong " mặt S, so sánh dựa vào mặt phẳng quan sát.
4. Các hình chiếu của hai mặt lên mặt phẳng quan sát không nằm chồng lên nhau.

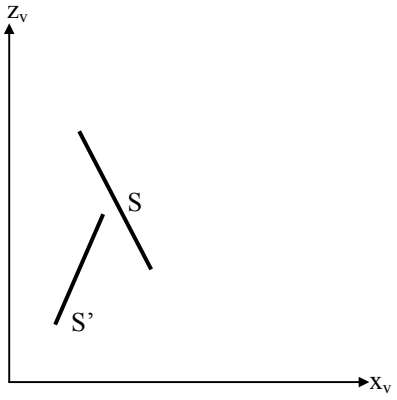
Vừa khi một phép kiểm tra được phát hiện là đúng cho một mặt nằm chồng, ta biết rằng mặt không nằm phía sau S. Vì vậy ta chuyển đến mặt chồng S kế tiếp. Nếu tất cả các mặt mặt nằm chồng vượt qua được ít nhất một trong các phép kiểm tra trên, ta không phải sắp xếp và S có thể được vẽ ra.

Phép kiểm tra 1 được thực hiện trong hai phần: Chúng ta kiểm tra sự nằm chồng theo hướng x, sau đó kiểm tra nằm chồng theo hướng y. Nếu cái nào trong hai hướng này được phát hiện là không có nằm chồng, hai mặt phẳng không che khuất nhau. Một ví dụ về hai mặt có nằm chồng theo hướng z nhưng không chồng theo hướng x được cho trong hình 7-10.

Chúng ta có thể thực hiện phép kiểm tra 2 bằng cách thế tọa độ tất cả các đỉnh của S vào phương trình mặt của mặt nằm chồng và kiểm tra dấu của kết quả. Giả sử rằng mặt nằm chồng có hệ số A', B', C', và D'. Nếu A'x + B'y + C'z + D' > 0 với mỗi đỉnh có tọa độ (x, y, z) của S, mặt S sẽ ở "phía ngoài" mặt nằm chồng S' (xem hình 7-11). Như được đề cập trước đây, các hệ số A', B', C', và D' phải được xác định trước để pháp vector của mặt nằm chồng S' chỉ ra xa khỏi mặt phẳng quan sát.



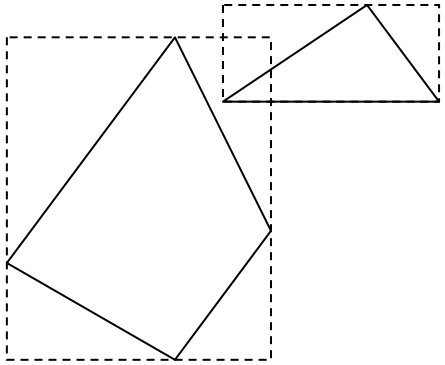
Hình 7-11
Mặt S hoàn toàn ở "phía ngoài" mặt nằm chồng S' khi nhìn từ mặt quan sát xy.



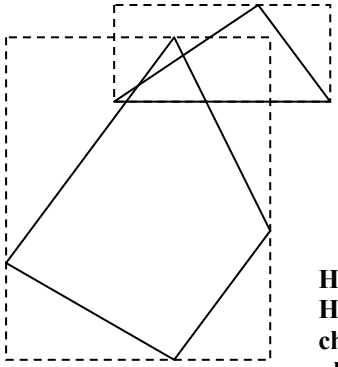
Hình 7-12
Mặt nằm chồng S' hoàn toàn ở "phía trong" mặt S, khi nhìn từ mặt quan sát xy.

Phép kiểm tra 3 được thực hiện dùng các hệ số A, B, C, và D của mặt S. Nếu tọa độ (x, y, z) của tất cả các đỉnh của mặt nằm chồng S' thỏa điều kiện Ax + By + Cz + D < 0, khi đó mặt nằm chồng S' sẽ ở "phía trong" mặt S (cung cấp pháp vector của mặt S hướng ra xa mặt phẳng quang sát). Hình 7-12 trình bày một mặt nằm chồng S' thỏa phép kiểm tra này. Trong ví dụ này, mặt S thì không ở "phía ngoài" S' (phép kiểm tra 2 không đúng).

Nếu tất cả các phép kiểm tra từ 1 đến 3 đều thất bại (sai), chúng ta thử đến phép kiểm tra 4 bằng cách kiểm tra sự cắt nhau giữa các cạnh biên của hai mặt, dùng các phương trình đường thẳng trong mặt xy. Như được minh họa trong hình 7-13, hai mặt có thể cắt hoặc không cắt nhau thậm chí khi các không gian bao quanh chồng nhau theo các hướng x, y, và z (xem hình 7-13).



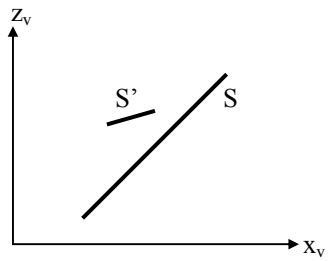
Các mặt không cắt nhau



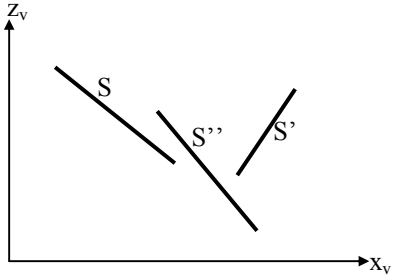
Các mặt cắt nhau

Hình 7-13
Hai mặt với các biên chữ nhật nằm chồng nhau trong mặt xy.

Nếu tất cả bốn phép kiểm tra trên đều thất bại với một mặt nằm chồng cụ thể S', ta đổi chỗ hai mặt S và S' cho nhau trong danh sách đã được sắp. Một ví dụ của hai mặt sẽ được sắp xếp lại với thủ tục này được cho trong hình 7-14. Tuy nhiên, ta vẫn không biết chắc rằng ta đã tìm gặp mặt xa nhất tính từ mặt phẳng quan sát chưa. Hình 7-15 minh họa một trường hợp mà tại đó đầu tiên chúng ta đổi chỗ S và S'' với nhau. Nhưng vì S'' che khuất một phần của S' (nhìn lên từ mặt xy), chúng ta cần đổi chỗ S'' và S' với nhau để có ba mặt được sắp hợp lý theo độ sâu. Do đó, chúng ta cần lặp lại quá trình kiểm tra cho mỗi mặt, cái vừa được sắp lại trong danh sách.



Hình 7-14
Mặt S có độ sâu z lớn hơn



Hình 7-15
Ba mặt ban đầu đã được sắp theo thứ tự độ sâu z : S,

Thuật toán vừa được phác thảo có thể đi vào một vòng lặp vô tận nếu hai hay nhiều mặt che khuất lẫn nhau một cách luân phiên như trong hình 7-7 (xem hình 7-7).

Một cách để phân chia một vùng thành công là chia kích thước nó ra làm 2, như trong hình 7-16. Một vùng quan sát với độ phân giải 1024x1024 có thể được chia 10 lần trước khi một phần chia giảm thành 1 điểm.

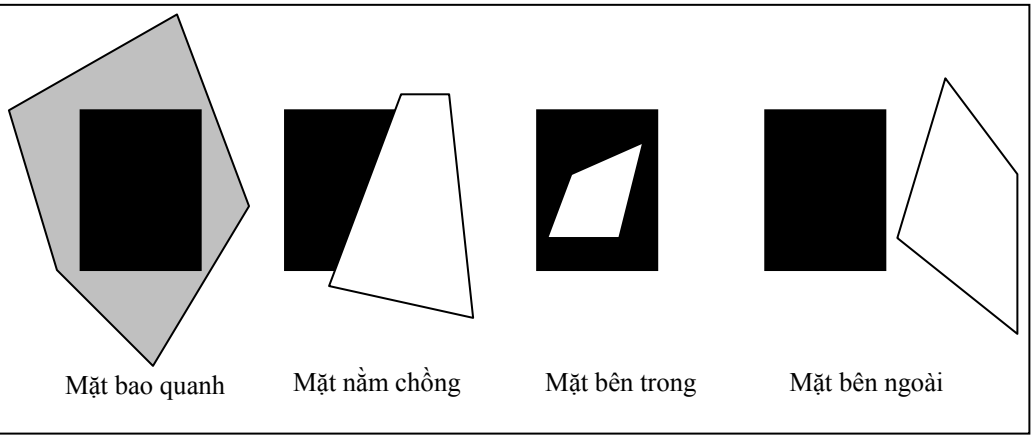
Các phép kiểm tra để xác định tính nhìn thấy được của một mặt đơn trong phạm vi vùng chỉ định được thực hiện bằng cách so sánh các mặt với biên của vùng. Có bốn khả năng có thể xảy ra khi xem xét mối quan hệ giữa một mặt với biên vùng chỉ định. Ta có thể mô tả đặc điểm của các quan hệ này theo các cách sau (xem hình 7-17):

Mặt bao quanh (surrounding surface) là mặt hoàn toàn bao quanh một vùng.

Mặt nằm chồng (overlapping surface) là mặt có một phần nằm trong và một phần nằm ngoài vùng.

Mặt bên trong (inside surface) là mặt hoàn toàn nằm bên trong vùng.

Mặt bên ngoài (outside surface) là mặt hoàn toàn nằm bên ngoài vùng.



Hình 7-17
Các quan hệ có thể xảy ra giữa các mặt đa giác và một vùng chữ nhật.

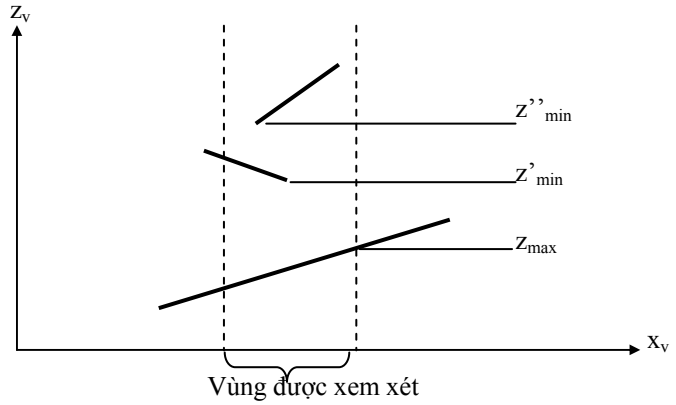
Các phép kiểm tra để xác định tính nhìn thấy được của mặt trong phạm vị một vùng có thể được đề cập giới hạn trong bốn loại này. Không có sự phân chia nào thêm nữa cho một vùng nếu một trong các điều kiện sau là đúng (true):

1. Tất cả các mặt nằm bên ngoài vùng.
2. Chỉ một mặt bên trong, mặt nằm chồng hoặc mặt bao quanh ở trong vùng.
3. Một mặt bao quanh che khuất tất cả các mặt khác trong phạm vi các biên của vùng.

Kiểm tra 1 có thể được thực hiện bằng cách kiểm tra các biên chữ nhật bao quanh các mặt với biên của vùng. Kiểm tra 2 cũng có thể dùng các biên chữ nhật trong mặt

xy để xác định mặt nằm trong. Với những kiểu mặt khác, các biên chữ nhật có thể được dùng như một bước kiểm tra ban đầu. Nếu một biên chữ nhật cắt vùng theo cách nào đó, các kiểm tra tiếp theo mới được thực hiện để xác định xem mặt là: mặt bao quanh, mặt nằm chồng, hay mặt bên ngoài. Nếu được xác định là mặt bên trong, mặt nằm chồng, hay mặt bao quanh, các giá trị độ sáng pixel của nó được chuyển đến vùng thích hợp trong vùng đệm khung.

Một phương pháp để thực hiện bước 3 là sắp xếp các mặt dựa theo độ sâu nhỏ nhất của chúng. Sau đó, với mỗi mặt bao quanh, ta đi tính giá trị z lớn nhất trong vùng được xem xét. Nếu giá trị lớn nhất z của một mặt nào (trong số các mặt mặt bao quanh) nhỏ hơn giá trị z nhỏ nhất của các mặt còn lại trong vùng, kiểm tra 3 thỏa. Hình 7-18 trình bày một ví dụ chứa các điều kiện của phương pháp này.



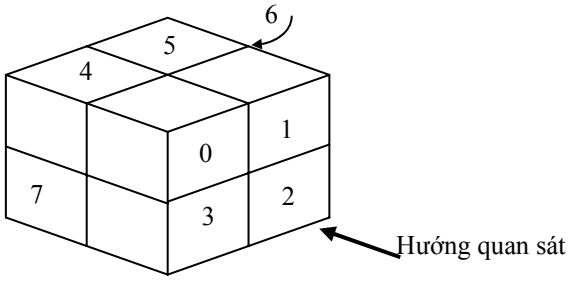
Hình 7-18

Một mặt bao quanh với độ sâu lớn nhất của z_{max} (xét trong vùng quan sát) che khuất tất cả các mặt mà độ sâu nhỏ nhất x_{min} của chúng lớn hơn z_{max}.

Một phương pháp khác để thực hiện kiểm tra 3 mà không cần đến sắp xếp độ sâu là dùng các phương trình mặt phẳng để tính các giá trị z ở bốn đỉnh của vùng cho tất cả các mặt bao quanh, mặt nằm chồng, hay mặt bên trong. Nếu các giá trị z của một trong số các mặt bao quanh mà nhỏ hơn các giá trị z của các mặt còn lại, kiểm tra 3 đúng. Sau đó vùng có thể được tô với các giá trị độ sáng của mặt bao quanh.

Trong vài trường hợp, cả hai phương pháp cho kiểm tra 3 trên sẽ thất bại để xác định đúng một mặt bao quanh che khuất tất cả các mặt còn lại khác. Việc kiểm tra thêm nữa sẽ được thực hiện để xác định mặt đơn che phủ vùng, tuy nhiên, thuật toán sẽ

người quan sát) được hình thành với các phần tám (octant) 0, 1, 2, 3. Mặt trước của các octant này được nhìn thấy bởi người quan sát. Bất kỳ mặt nào hướng về phía sau của các octant phía trước này hoặc các octant ở đằng sau (4, 5, 6, và 7) có thể bị che khuất bởi các mặt phía trước.



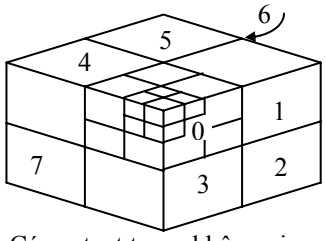
Các Octant được đánh số của một vùng

Hình 7-20

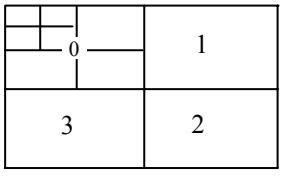
Các đối tượng trong các octant 0, 1, 2, và 3 che khuất các octant phía sau (4, 5, 6, 7) khi hướng quan sát như trong hình.

Các mặt phía sau bị loại bỏ, với hướng quan sát như trong hình 7-20, bằng cách xử lý các phần tử dữ liệu tại các nút octree theo thứ tự 0, 1, 2, 3, 4, 5, 6, 7. Điều này tạo ra kết quả du hành theo độ sâu của octree, để các octant 0, 1, 2, và 3 của toàn vùng được viếng thăm trước các octant 4, 5, 6, và 7. Tương tự, bốn octant con trước của octant 0 sẽ được viếng thăm trước bốn octant con phía sau. Cuộc du hành của octree sẽ tiếp tục theo thứ tự này cho mỗi phần chia octant.

Hình 7-21
Sự phân chia octant cho một vùng không gian và mặt các phần tư tương ứng.



Các octant trong không gian



Các quadrant (góc 1/4) trong mặt phẳng quan sát

Khi giá trị màu được gặp tại một nút của octree, vùng pixel trong vùng đệm khung tương ứng với nút này được gán giá trị màu đó chỉ nếu không giá trị nào được lưu trước đó trong vùng này. Không gì được nạp nếu một vùng trống rỗng. Bất kỳ nút nào được phát hiện là bị che khuất hoàn toàn thì sẽ bị loại bỏ khỏi các xử lý trong tương lai, để các các cây con của nó không được truy cập vào.

Các quang cảnh khác nhau của đối tượng được biểu diễn như octree có thể đạt được bằng cách áp dụng các phép biến đổi đến sự biểu diễn octree để làm thay đổi đối tượng theo hướng quan sát. Ta giả sử rằng biểu diễn octree luôn được xây dựng sao cho các octant 0, 1, 2, và 3 của một vùng hình thành nên mặt phía trước (xem hình 7-20).

Một phương pháp để hiển thị một octree từ trước ra sau là đầu tiên ánh xạ octree vào một quadtree của các vùng nhìn thấy được bằng cách duyệt qua các nút của octree từ trước ra sau trong một quá trình đệ quy. Sau đó biểu diễn quadtree của các mặt nhìn thấy được được nạp vào trong vùng đệm khung. Hình 7-21 mô tả các octant trong một vùng không gian và các quadtree tương ứng trên mặt phẳng quan sát. Các phần tạo thành quadtree 0 lấy từ octant 0 và 4. Các giá trị màu trong góc phần tư 1 (quadrant 1) có được từ các mặt trong octant 1 và 5, và các giá trị trong mỗi của hai quadrant còn lại được sinh ra từ cặp octant thẳng hàng với mỗi quadrant này.

Việc xử lý đệ quy của các nút octree được trình bày trong thủ tục *convert_oct_to_quad*, nơi nhận vào một mô tả octree và tạo ra các biểu diễn quadtree cho các mặt nhìn thấy được trong vùng. Trong hầu hết các trường hợp, cả octant phía trước và phía sau phải được xem xét để xác định màu đúng cho một quadrant. Tuy nhiên, ta có thể bỏ qua quá trình xử lý octant phía sau nếu octant phía trước được tô đồng nhất với vài màu. Đối với các vùng không đồng nhất, thủ tục được gọi đệ quy, với các đối số mới – con của octant không đồng nhất và nút quadtree được tạo mới. Nếu octant phía trước rỗng, chỉ cần xử lý con của octant phía sau. Ngược lại, hai lời gọi đệ quy được làm, một cho octant phía sau và một cho octant phía trước.

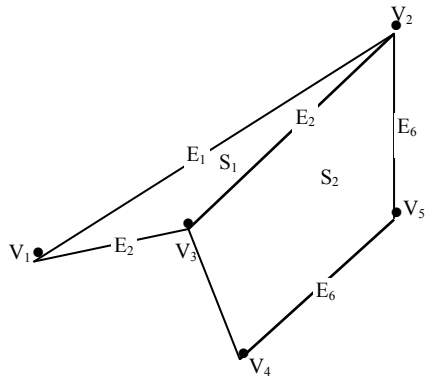
```
type
  oct_node_ptr =^ oct_node;
  oct_entry  =  record
        case homogeneous: boolean of
              true : (color : integer);
              false : (child : oct_node_ptr)
           end; {record}
        oct_node = array [0..7] of oct_entry;

        quad_node_ptr = ^ quad_node;
        quad_entry = record
             case homogeneous: boolean of
              true : (color : integer);
              false : (child : oct_node_ptr)
           end; {record}
```

```
quad_node = array[0..3] of quad_entry;

var
    newquadtree : quad_node_ptr;
    backcolor:   integer;

{Giả sử quang cảnh phía trước của một octree (với các octant 0, 1, 2,
3 ở phía trước) và, khi biểu diễn này được hiển thị, biến đổi nó thành
một quatree. Nhận một octree như input, nơi mà mỗi phần tử của
octree là một giá trị màu (homogeneous = true và octant được tô với
màu này) hoặc là con trỏ đến một nút octant con  (homogeneous =
false).}
procedure convert_oct_to_quad(octree: oct_node;
                              var quadtree: quad_node);
    var  k: integer;
    begin
        for k:=0 to 3 do begin
            quadtree[k].homogeneous:=true;
            if (octree[k].color>-1) then        {octant trước đầy}
              quadtree[k].color:= octree[k].color
            else                                {octant trước rỗng}
              if octree[k+4].homogeneous then
                if (octree[k+4].color > -1) then    {trước rỗng, sau đầy}
                    quadtree[k].color:=octree[k+4].color
                else                                {trước và sau rỗng}
                    quadtree[k].color:=backcolor
            else begin                          {trước rỗng, sau không đồng
nhất}
                    quadtree[k].homogeneous:=flase;
                    new(newquadtree);
                    quadtree[k].child: = newquadtree;
                    convert_oct_to_quad(octree[k+4].child^, newquadtree^);
```

toàn bị che khuất, như hình 7-22 (a). Khi đường thẳng có độ sâu lớn hơn độ sâu ở một giao điểm với biên và có độ sâu nhỏ hơn độ sâu của mặt ở các giao điểm với biên còn lại, đường thẳng phải đi xuyên qua mặt như hình 7-22 (b). Trong trường hợp này, chúng ta tính tọa độ giao điểm của đường với mặt bằng cách dùng phương trình mặt và chỉ hiển thị các phần được nhìn thấy của đường thẳng.

Vài phương pháp khử mặt khuất dễ dàng được áp dụng để khử các đường khuất. Dùng phương pháp mặt sau (back-face), chúng ta có thể nhận biết được các mặt sau của một đối tượng và chỉ hiển thị các biên của các mặt nhìn thấy được. Với phương pháp sắp xếp theo độ sâu, các mặt được vẽ vào trong vùng đệm làm tươi để phần bên trong của mặt có độ sáng nền, trong khi đó các biên có độ sáng là độ sáng vẽ. Bằng cách xử lý các mặt từ sau đến trước, các đường khuất bị xóa bởi các mặt ở gần hơn. Phương pháp chia vùng có thể được áp dụng để khử các đường khuất bằng cách chỉ hiển thị các biên của các mặt nhìn thấy được. Các phương pháp scan-line có thể được dùng để hiển thị các đường nhìn thấy được bằng cách bố trí các điểm dọc theo các đường quét, các điểm này trùng với các biên của các mặt nhìn thấy được. Bất kỳ phương pháp khử mặt khuất nào dùng các đường quét đều có thể được thay đổi thành phương pháp khử đường khuất theo cách tương tự (xem hình 7-23).

Hình 7-23
Các bảng dữ liệu hình học cho một đối tượng ba chiều được biểu diễn bởi hai mặt phẳng, được hình thành với sáu cạnh và năm đỉnh.



VERTEX TABLE
V_1: x_1, y_1, z_1
V_2: x_2, y_2, z_2
V_3: x_3, y_3, z_3
V_4: x_4, y_4, z_4
V_5: x_5, y_5, z_5

EDGE TABLE
E_1: V_1, V_2, S_1
E_2: V_2, V_3, S_1, S_2
E_3: V_3, V_1, S_1
E_4: V_3, V_4, S_2
E_5: V_4, V_5, S_2
E_6: V_5, V_2, S_2

POLYGON TABLE
S_1: E_1, E_2, E_3
S_2: E_2, E_4, E_5, E_6

được xác định. Sự đòi hỏi không gian lưu trữ cho vùng đệm độ sâu có thể được xác định như thế nào từ định nghĩa các đối tượng để được hiển thị?

6. Phát triển một chương trình cài đặt thuật toán scan-line để hiển thị các mặt nhìn thấy được của một đối tượng được định nghĩa bất kỳ nằm trước vùng quan sát. Dùng các bảng đa giác và bảng cạnh (polygon table, edge table) để lưu trữ sự định nghĩa của đối tượng, và dùng kỹ thuật cố kết để tính các điểm dọc theo và giữa các đường quét.

7. Cài đặt một chương trình để hiển thị các mặt nhìn thấy được của một khối đa diện lồi, dùng các thuật toán của họa sĩ (painter's algorithm). Tức là, các bề mặt phải được sắp theo độ sâu và được vẽ lên màn hình từ sau đến trước.

8. Mở rộng chương trình của bài 7 để hiện thị một đối tượng được định nghĩa bất kỳ với các mặt phẳng, dùng các kiểm tra sắp xếp độ sâu (depth-sorting checks) để có các mặt theo thứ tự sắp hợp lý.

9. Cho các ví dụ về các trường hợp mà tại đó hai phương pháp đã được thảo luận về kiểm tra 3 trong các thuật toán phân chia vùng sẽ thất bại để từ đó chỉ ra một cách đúng đắn một mặt bao quanh có thể che khuất tất cả các mặt.

10. Phát triển một thuật toán có thể kiểm tra một mặt được cho tương tác với một vùng chữ nhật để quyết định xem nó là một mặt bao quanh, nằm chồng, bên trong, hay nằm ngoài.

11. Mở rộng các phương pháp trong bài tập 10 thành một thuật toán để sinh ra một biểu diễn quadtree cho các mặt nhìn thấy được của đối tượng bằng cách áp dụng các kiểm tra vùng con (area-subdivision tests) để xác định các giá trị của các phần tử quadtree.

12. Cài đặt một thuật toán để nạp biểu diễn quadtree của bài tập 11 thành đường quét (raster) để hiển thị.

13. Viết một chương trình lên hệ thống của bạn để hiển thị một biểu diễn octree cho một đối tượng để các mặt khuất bị loại bỏ.

14. Phát triển một thuật toán để loại bỏ các đường khuất bằng cách so sánh mỗi đường trong ảnh với mỗi mặt.

15. Thảo luận làm thế nào việc tháo bỏ các đường khuất có thể được thực hiện với các phương pháp khử mặt khuất khác nhau.

16. Cài đặt một thủ tục để hiển thị các cạnh bị che khuất của một đối tượng chứa các mặt phẳng thành những đường nét đứt.

HẾT

Đồ họa máy tính
(Computer Graphics)

Giới thiệu

Giới thiệu môn học

- Giảng viên: Ma Thị Châu – Phòng thí nghiệm HMI, phòng 303, E3
- Email: chaumt@vnu.edu.vn
- Website môn học:

http://www.fotech.vnu.edu.vn/courses

http://bbc.vnu.edu.vn

Chọn môn: Đồ họa máy tính

Đồ họa máy tính

Tạo ra, lưu trữ và thao tác với các bức ảnh nhân tạo dựa trên mô tả hoặc mô hình

Mục tiêu môn học

Kết thúc thành công môn học, sinh viên sẽ:

- hiểu các nguyên lý cơ bản của đồ họa máy tính hiện đại
- hiểu kiến thức hình học bên dưới các mô hình 3 chiều
- hiểu vấn đề hiệu năng khi vẽ các mô hình 3D
- có thể xây dựng một chương trình hiển thị một cảnh 3 chiều sử dụng OpenGL và C/C++
- có thể làm hoạt hình các mô hình 3D và áp dụng ánh sáng và texture để tăng tính hiện thực

Nội dung khóa học

- Giới thiệu các thuật toán cơ bản của ĐHMT
- Mô tả lại qui trình ĐHMT từ khâu mô hình hóa cho đến khi hiển thị
- Giới thiệu một số kiến thức cơ bản để lập trình OpenGL

Nội dung khóa học (…)

- Lập trình và các thuật toán đồ họa
- Các cấu trúc dữ liệu đồ họa
- Màu sắc và thị giác con người
- Các cấu trúc hình học, mô hình hóa và kết xuất đồ họa (rendering)

Nội dung khóa học

Không phải là!

- Các chương trình vẽ (Adobe Photoshop)
- Các chương trình thiết kế hỗ trợ bởi máy tính (AutoCAD)
- Các chương trình tạo mô hình (3D Studio MAX)
- Các chương trình tạo hoạt ảnh (Digimation)

Kiểm tra đánh giá

Chuyên cần

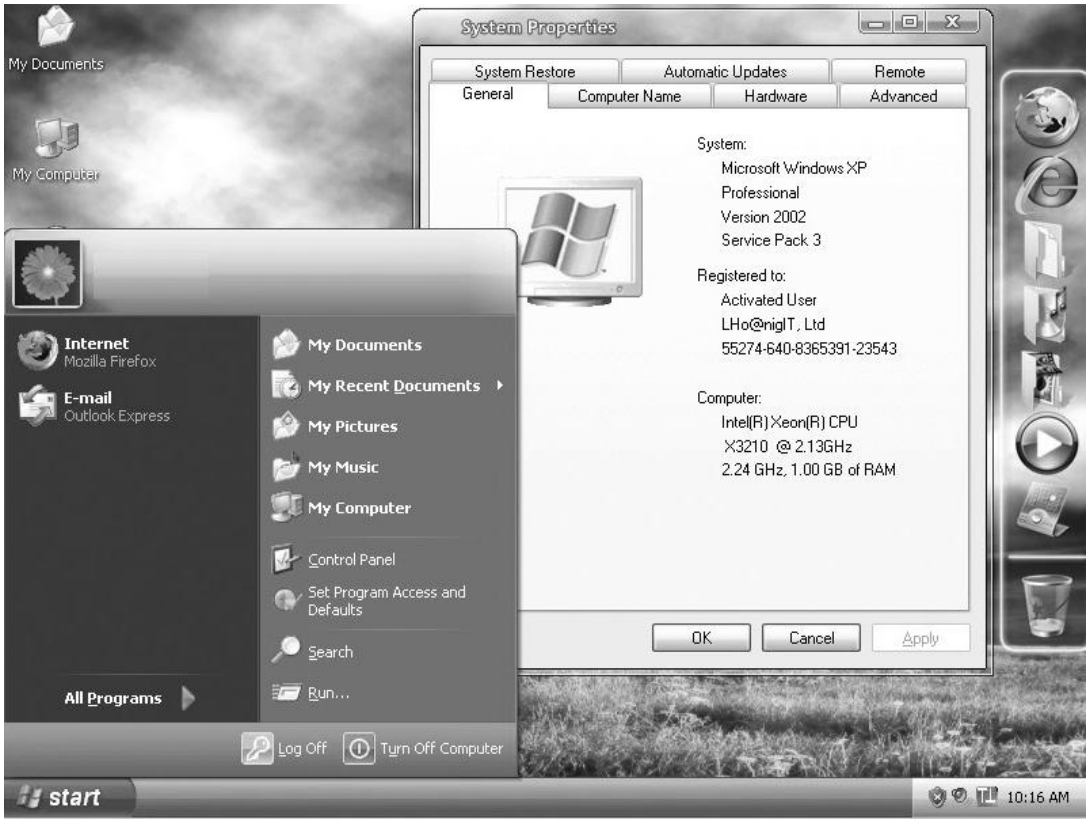
- *Điểm danh*
- *Đăng kí chuẩn bị bài có trình bày*
- *Đặt các câu hỏi phản biện khi có thảo luận*

Thi giữa kỳ

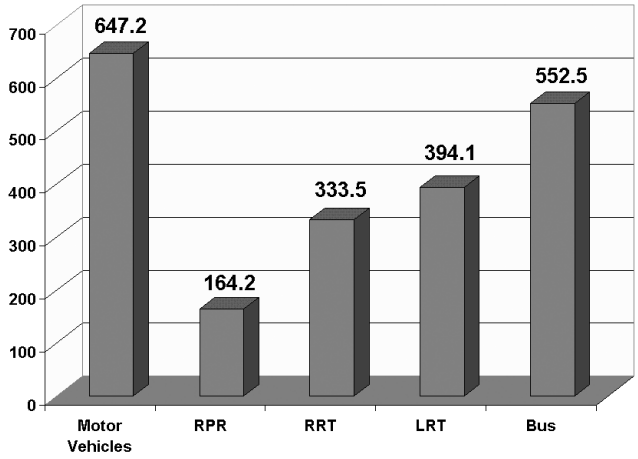
Thi cuối kỳ

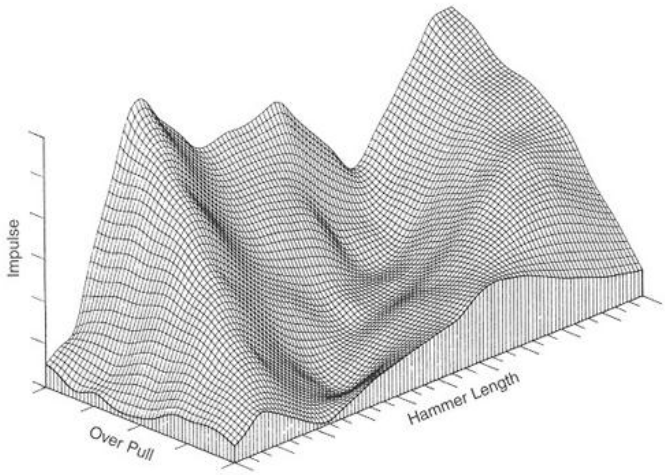
Ứng dụng
Giao diện người dùng

Sử dụng hàng ngày



Ứng dụng
Vẽ biểu đồ





Trong kinh doanh, khoa học, công nghệ…

Ứng dụng
Vẽ bản đồ

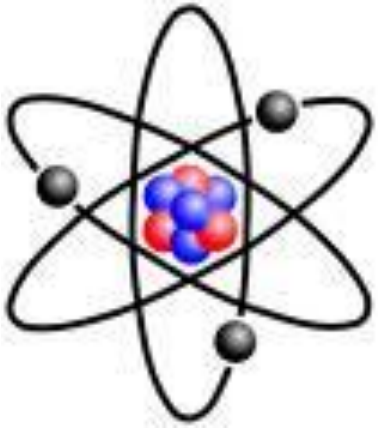
Từ việc thu thập các dữ liệu tự nhiên

Ứng dụng
Dựng ảnh y tế

Nguồn hỗ trợ tài chính dồi dào

Thúc đẩy mối liên hệ giữa đồ họa và video, dữ liệu quét …

Ứng dụng
Trực quan hóa khoa học



Mô phỏng các hệ thống vi mô cũng như vĩ mô

Ứng dụng
Giải trí

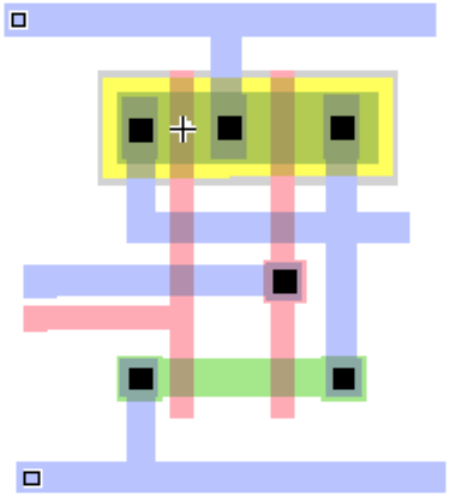
Tạo hoạt ảnh

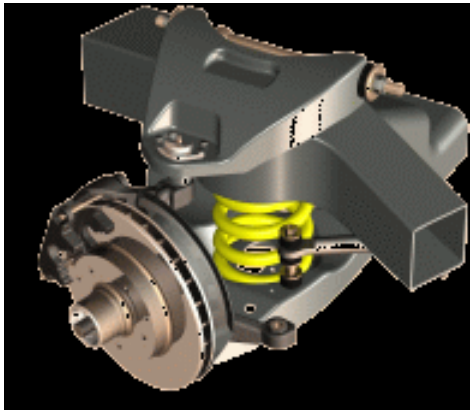
Động lực phát triển chủ yếu

Ứng dụng
Giải trí

ĐHMT rất quan trọng với trò chơi điện tử

Ứng dụng
Với thiết kế hỗ trợ bởi máy tính (CAD)

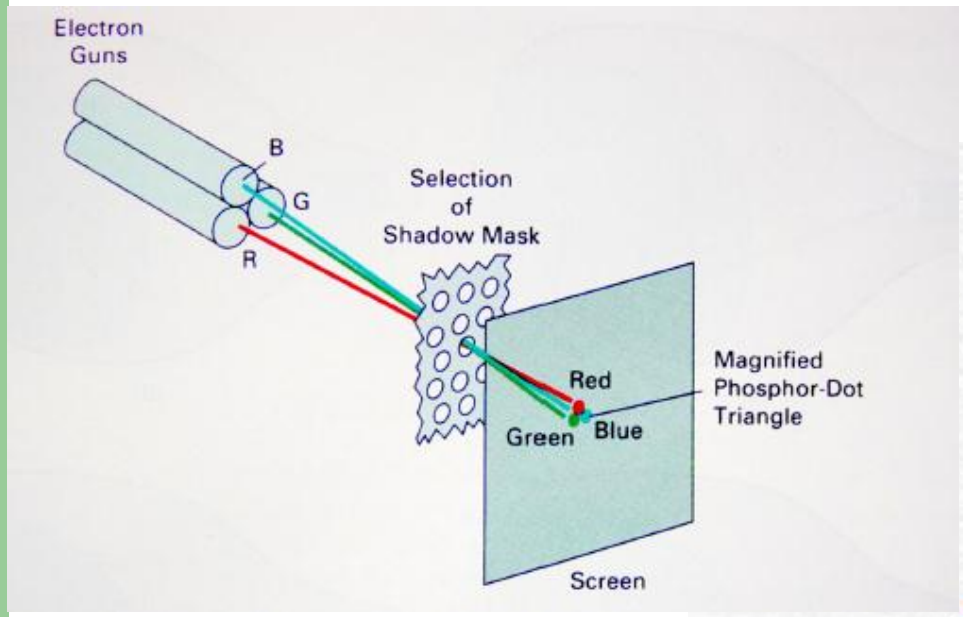


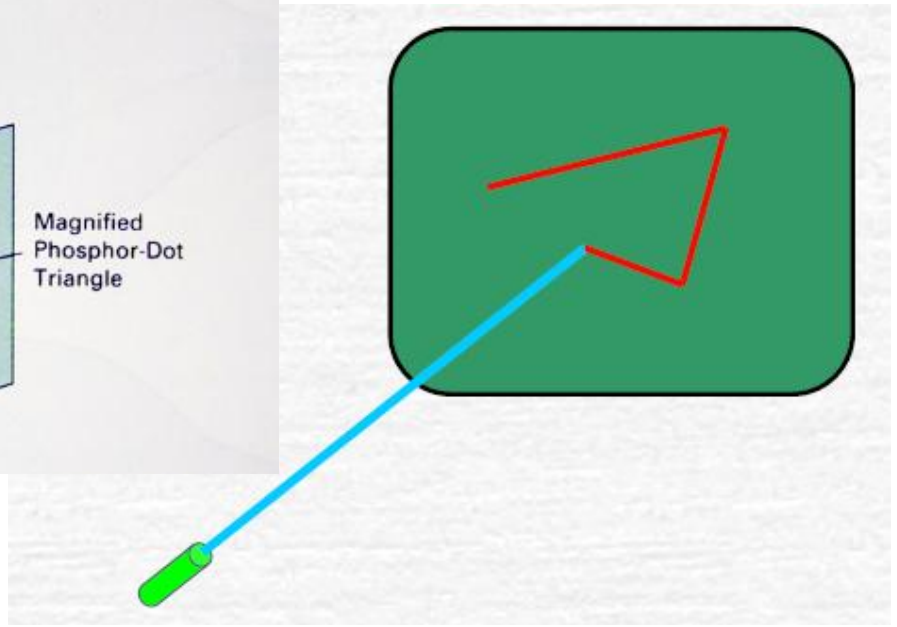


Ứng dụng khác?

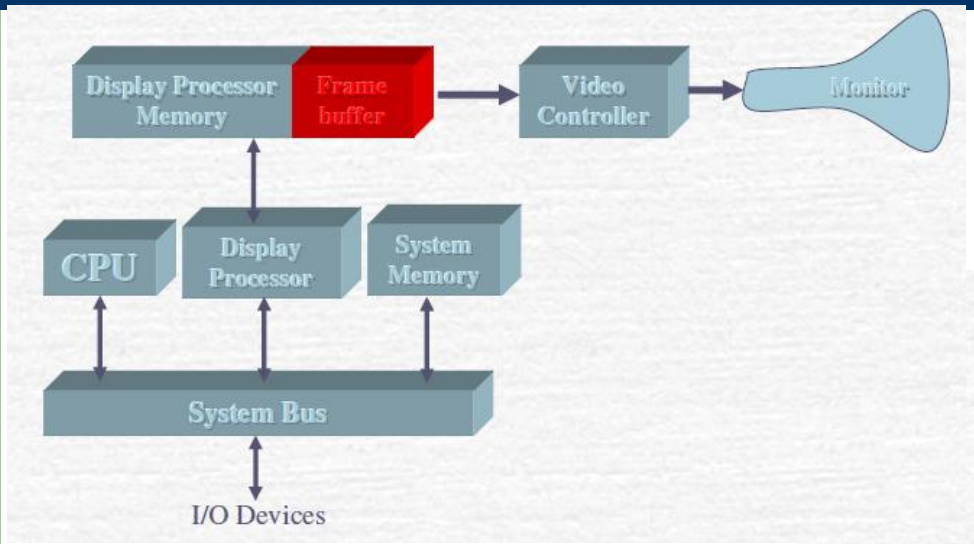
Công nghệ hiển thị

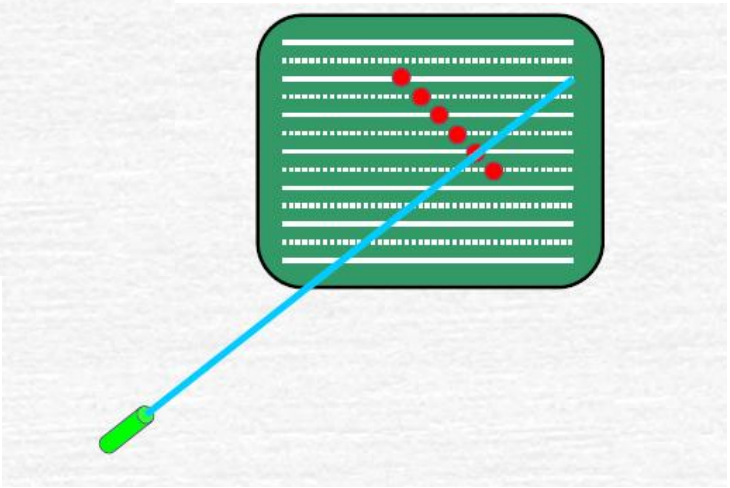
Công nghệ hiển thị
Hệ thống hiển thị vectơ – kỹ thuật quét ngẫu nhiên





Công nghệ hiển thị
Thiết bị đồ họa mành





Thiết bị hiển thị vecto vs. thiết bị hiển thị mành
Liệt kê những loại thiết bị hiển thị mà đã từng biết

Phần thảo luận buổi sau:

1. Lịch sử ĐHMT (01 sv – presentation 15p)
2. Các khái niệm cơ bản của ĐHMT (01 sv - presentation 15p)
3. Luồng xử lý đồ họa (01 sv –presentation 15p)
1 nhóm làm trợ giảng dạy OPENGL

06 người – Dạy thông qua ví dụ
- Ra bài tập tương ứng
- Kiểm tra bài tập
- Bắt đầu từ buổi học thứ 4

http://nehe.gamedev.net/lesson.asp?index=01
http://www.onecore.net/dev-c-opengl.htm

Đồ họa máy tính

Các thuật toán mành hóa

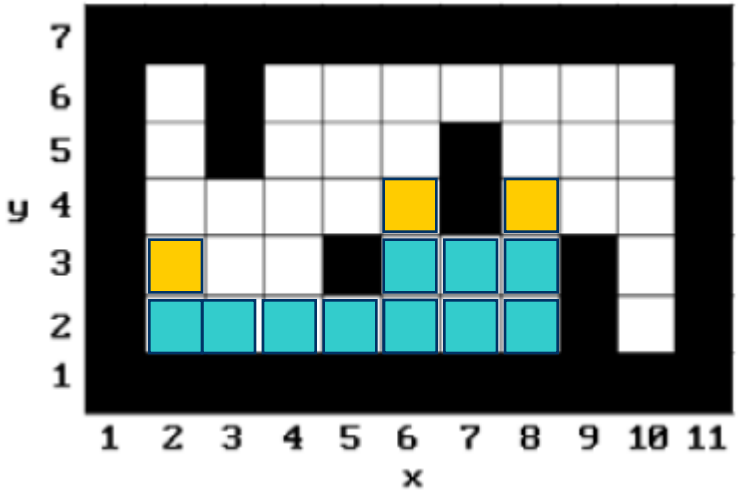
Các thuật toán tô phủ

Bài toán tô phủ loang (*Flood fill problem*):
*Với hai màu khác nhau **c** và **c'**, một tập các điểm **A** có cùng màu **c** được bao quanh bởi các điểm có màu khác với **c** và **c'**, tìm thuật toán thay màu của tất cả các điểm thuộc **A** và chỉ các điểm này thành màu **c'***

Thuật toán tô phủ cơ bản

```
procedure BFA (integer x, y)
begin
      if Inside (x,y) then
                  Begin
                  Set (x,y);
                  BFA (x,y - 1); BFA (x,y + 1);
                  BFA (x - 1,y); BFA (x + 1,y);
                  end
end;
```

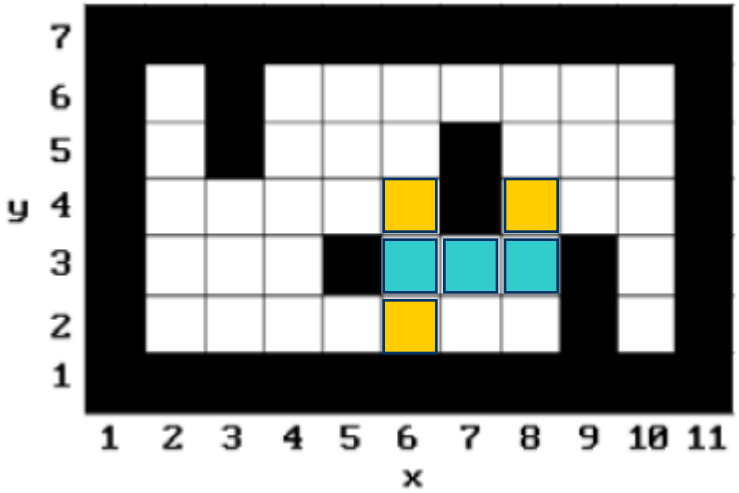
Thuật toán tô phủ của Smith



Bắt đầu: (7,3).
FillRight: đoạn (7,3) đến (8,3) được tô.
FillLeft: (6,3) được tô.
ScanHi: điểm (6,4) và (8,4) vào ngăn xếp.
ScanLo:điểm (6,2) vào ngăn xếp.
Lấy(6,2) ra, và coi đây là điểm bắt đầu.
Lệnh FillRight và FillLeft: tô phủ đoạn từ (2,2) đến (8,2).
ScanHi và ScanLo:cho (2,3) và (6,3) vào ngăn xếp.
Lấy (6,3) ra.
(6,3) đã được tô lấy ra (2,3) và cứ tiếp tục như thế cho đến khi ngăn xếp rỗng

6,3
2,3
8,4
6,4

Thuật toán tô phủ Smith

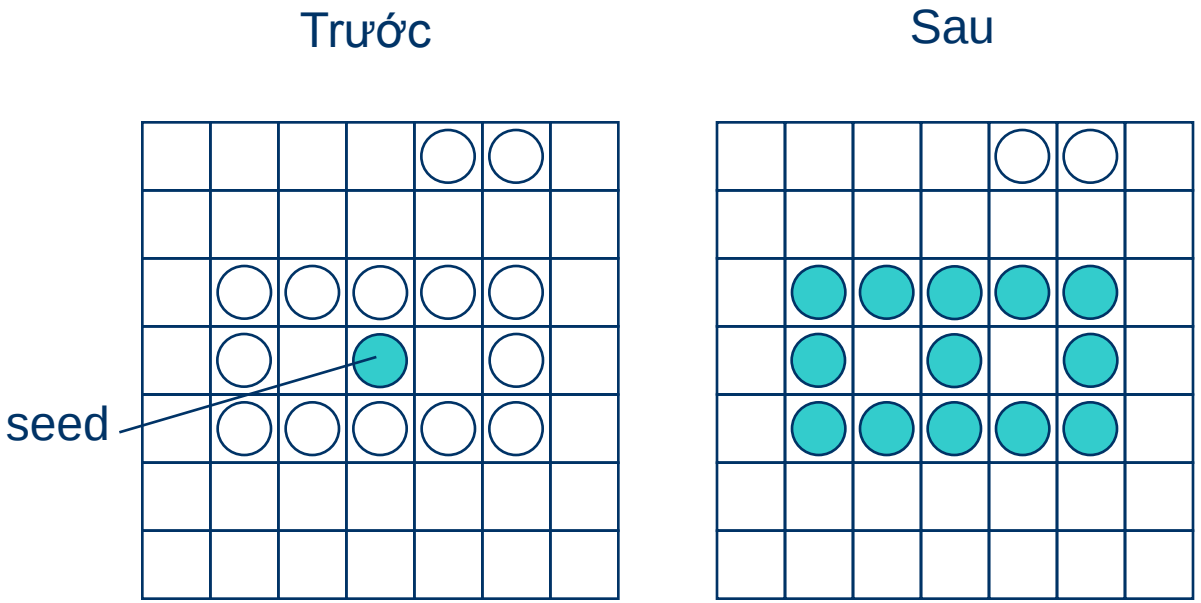


Các đoạn chứa (6,4), (8,4) và (6,2) được gọi là vùng bóng tối

Thuật toán tô phủ của Fishkin

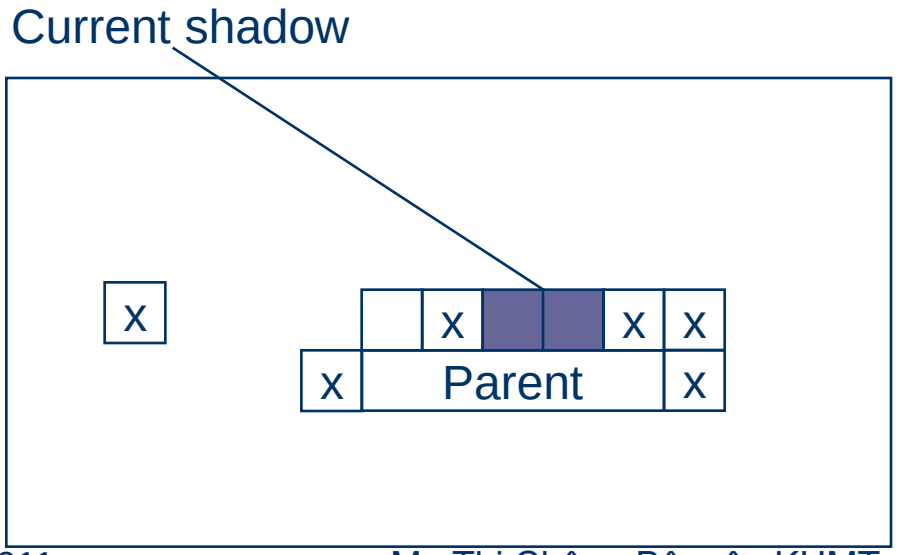
Vùng bóng tối – shadow

Thuật toán tô phủ của Fishkin

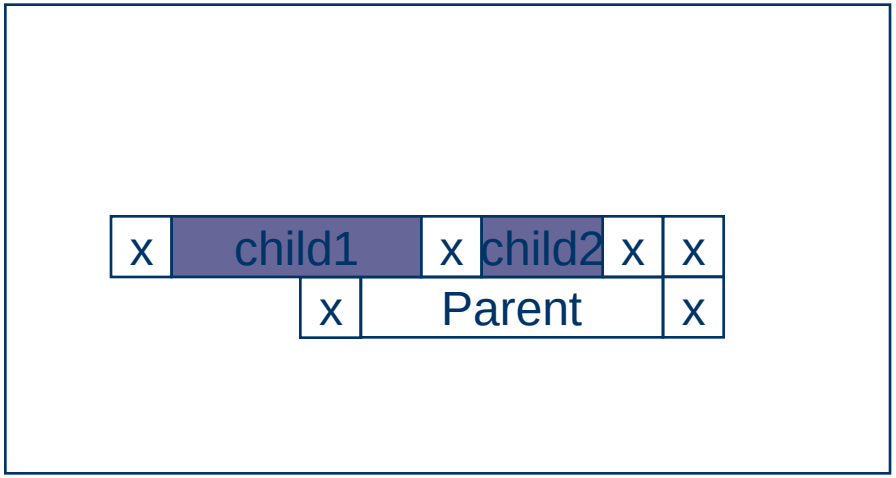


Thuật toán tô phủ của Fishkin

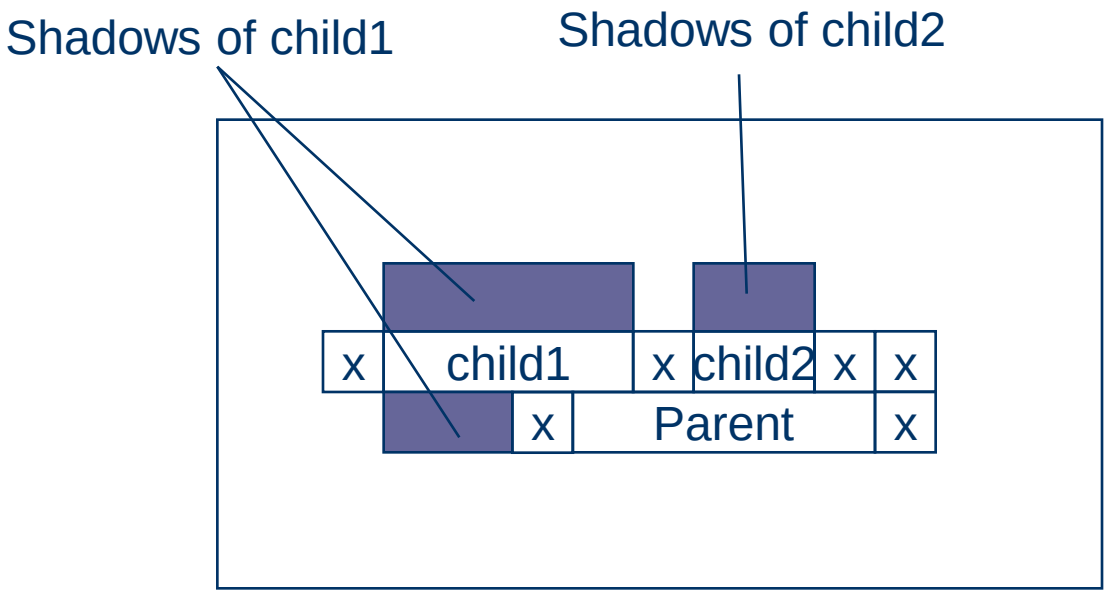
```
stackRec = record // Một bản ghi dữ liệu cho vùng bóng tối
    {       integer myLx, myRx, // điểm kết thúc của vùng bóng tối này
                    dadLx, dadRx, // điểm kết thúc của vùng mẹ
                    myY; // dòng quét của vùng này
            direction myDirection; // -1 ở dưới vùng mẹ,+1 ở trên vùng mẹ
    }
```



Thuật toán tô phủ của Fishkin

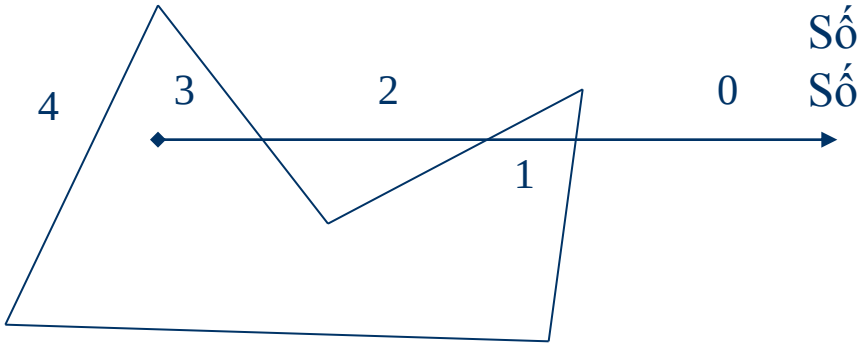


Thuật toán tô phủ của Fishkin

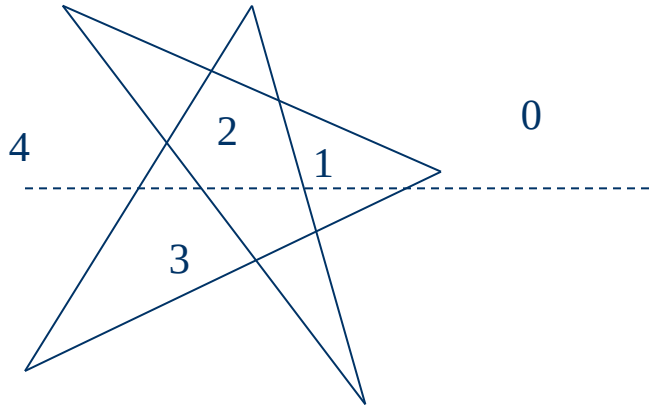


Cài đặt thuật toán tô phủ cơ bản
Cài đặt thuật toán tô phủ Smith
Cài đặt thuật toán tô phủ Fishkin

Định lý Jordan.



Số điểm cắt chẵn: Ngoài đa giác
Số điểm cắt lẻ: Trong đa giác



Không đúng đối với đa giác tự cắt

Định lý Jordan

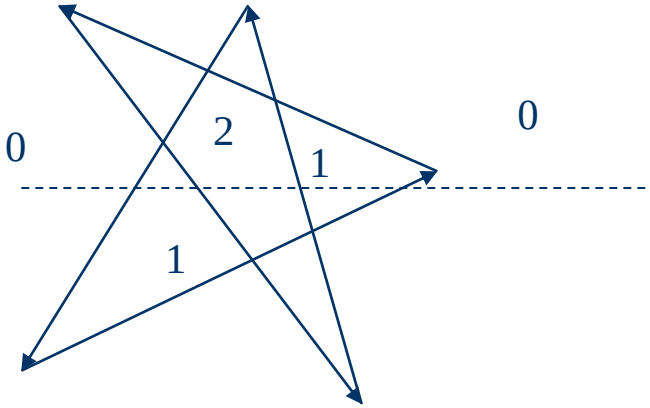
Kiểm tra đại lượng e

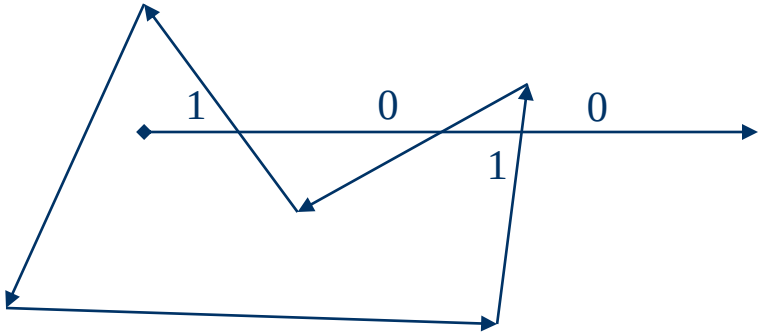
-Sử dụng cả hướng của đường thẳng

-đặt e = 0

-Cắt từ trái qua phải e + +, phải qua trái e - -

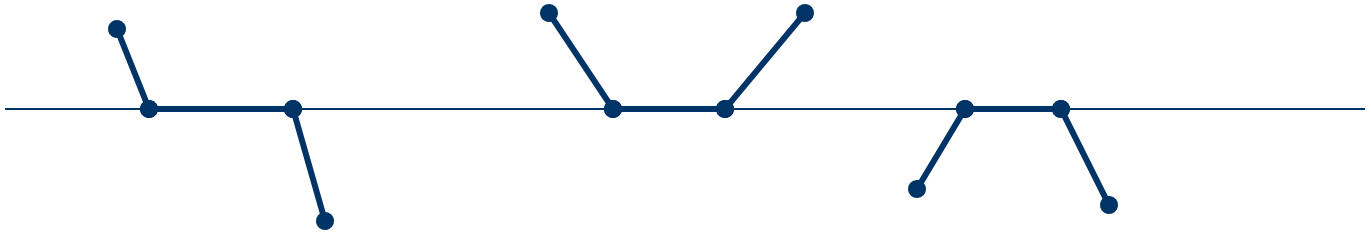
-e != 0, nằm trong



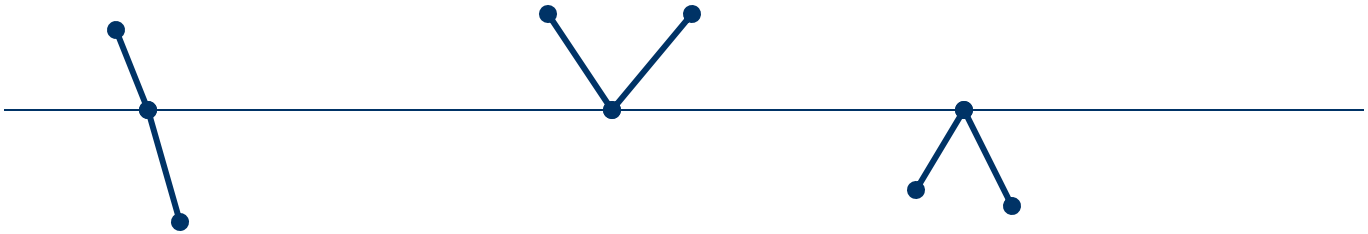


Trường hợp đặc biệt

- Có 2 trường hợp đặc biệt trong thuật toán Jordan :
- Cắt trùng lên cạnh

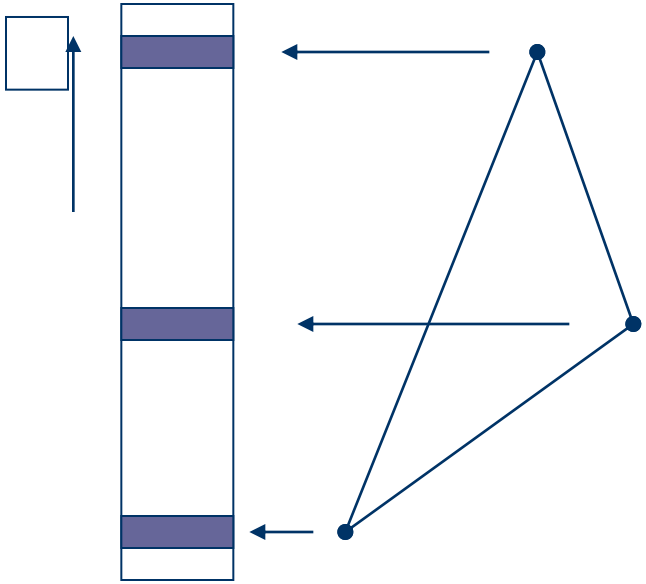


- Cắt trùng lên đỉnh đa giác



Thuật toán đường quét

- Kiểm tra Jordan tăng dần
- Sắp xếp theo giá trị của y



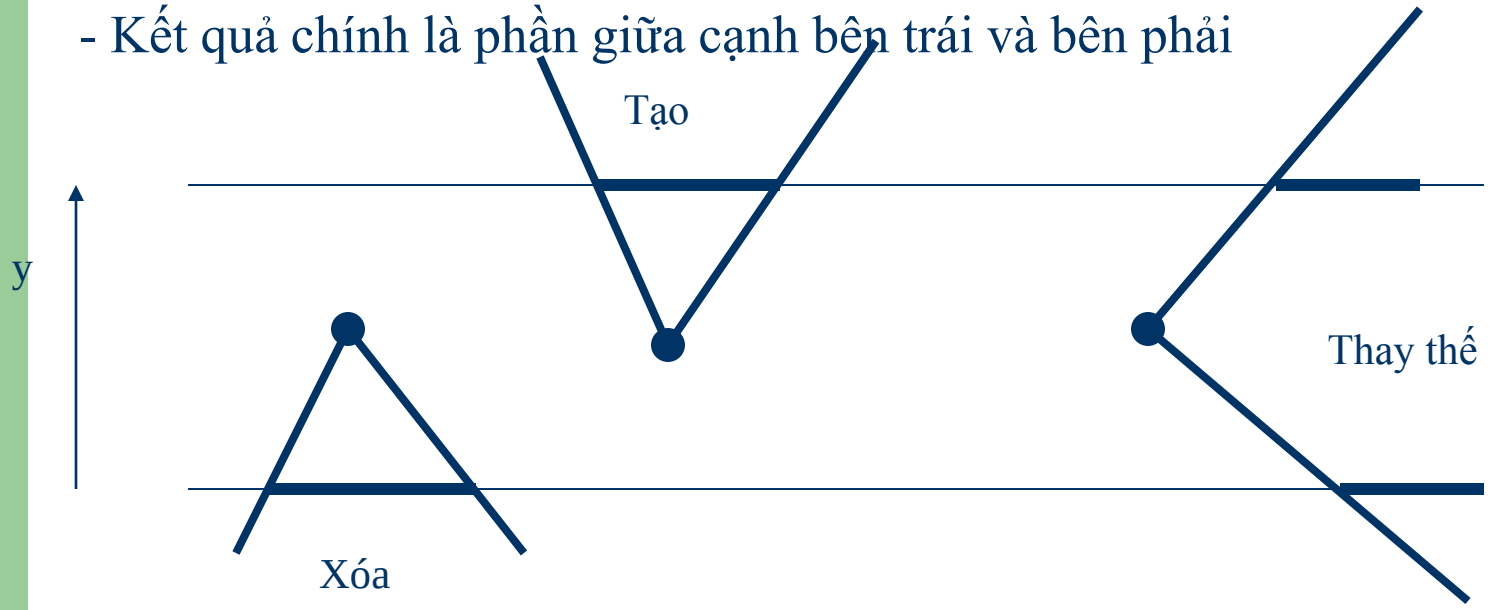
Thuật toán đường quét

- Kiểm tra Jordan tăng dần
- Sắp xếp theo giá trị của y
- Sử dụng sự liên kết giữa các đường quét – giá trị cho đường quét trước gần bằng giá trị cho đường quét sau.
- Lưu trữ danh sách các cạnh đang xét

Danh sách các cạnh đang xét

Các đỉnh là các 'sự kiện' trong danh sách cạnh – các cạnh có thể được xét, không được xét hoặc được thay bằng các cạnh khác

- Sắp xếp các giao điểm theo x
- Kết quả chính là phần giữa cạnh bên trái và bên phải



Danh sách các cạnh đang xét

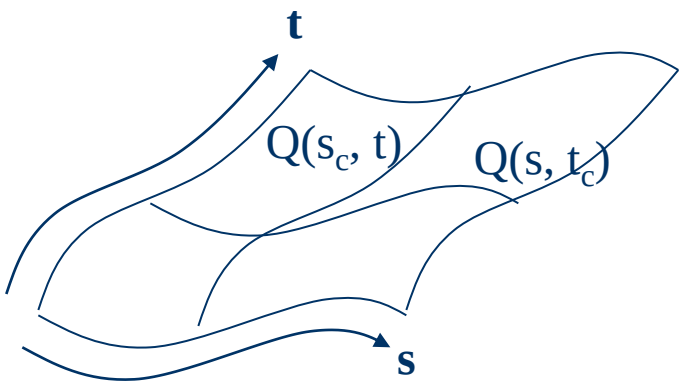
Phần thảo luận buổi sau:

1. Các thuật toán cắt xén 03 sv – Presentation120p

Đồ họa máy tính

Đường cong và bề mặt II

Bề mặt cong

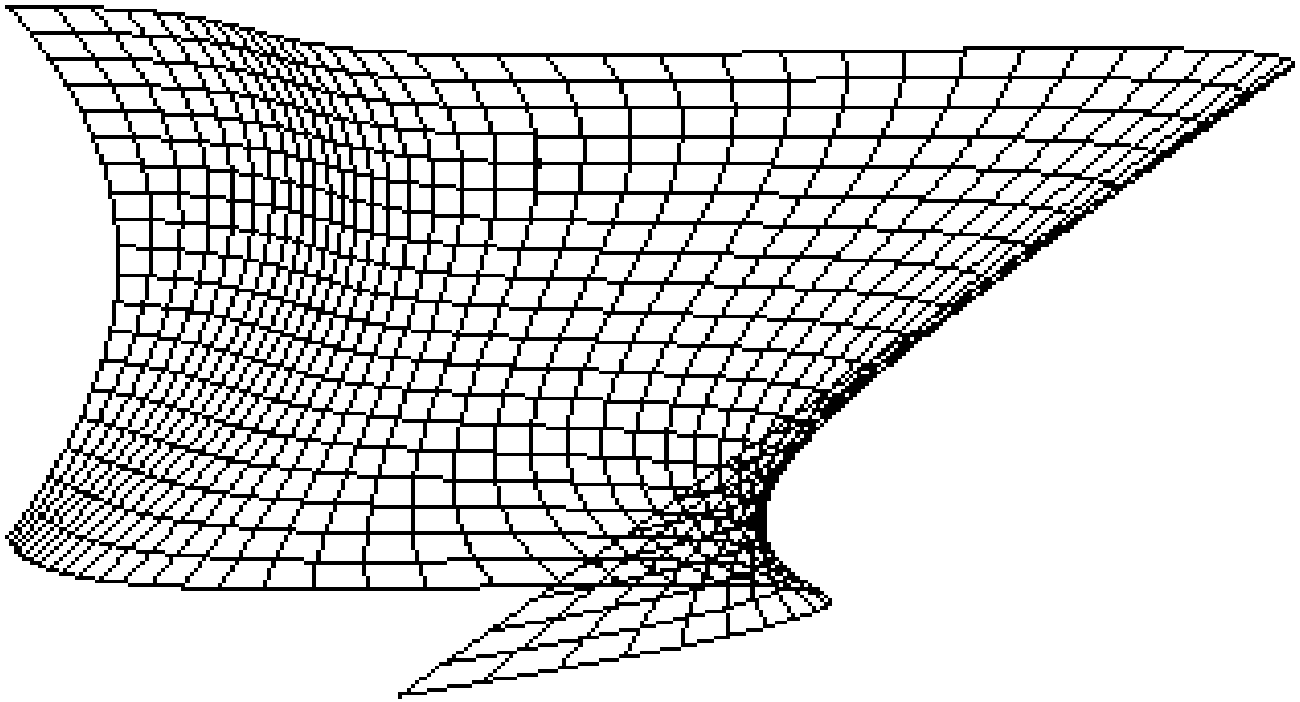


- Có thể mở rộng khái niệm đoạn cong cho các bề mặt cong.
- Các bề mặt cong được xác định bởi công thức tham số của hai biến, *s* và *t*.

$$0 \le s \le 1 \quad and \quad 0 \le t \le 1$$

- Nghĩa là, một bề mặt cong là một tập hợp các đường cong tham số
- Xấp xỉ bằng một lưới đa giác. Khi vẽ, càng giảm nhỏ bước của *s* và *t* càng cho độ chính xác cao.

Bề mặt cong Bézier



Kiểm soát hình dạng của bề mặt

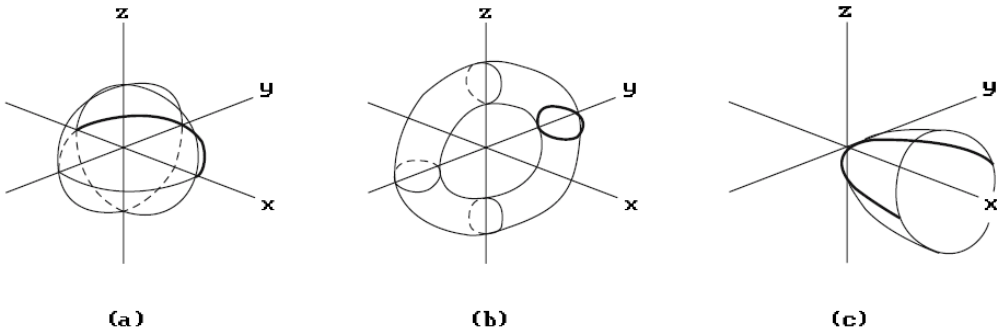
- Điều khiển bởi một lưới 2D các điểm điều khiển.
- Hàm bề mặt hai tham số có dạng:

$$X(s,t) = \sum_{ij} f_i(s)\, f_j(t)\, q_{ij}$$

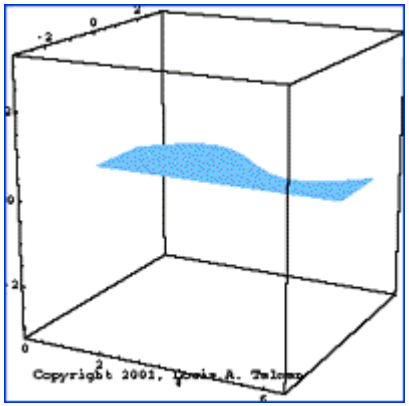
$$similarly \;\; for \; Y(s,t) \; and \; Z(s,t)$$

- Sử dụng các hàm cơ bản phù hợp cho các bề mặt Bézier và B-Spline.

Các bề mặt tròn xoay

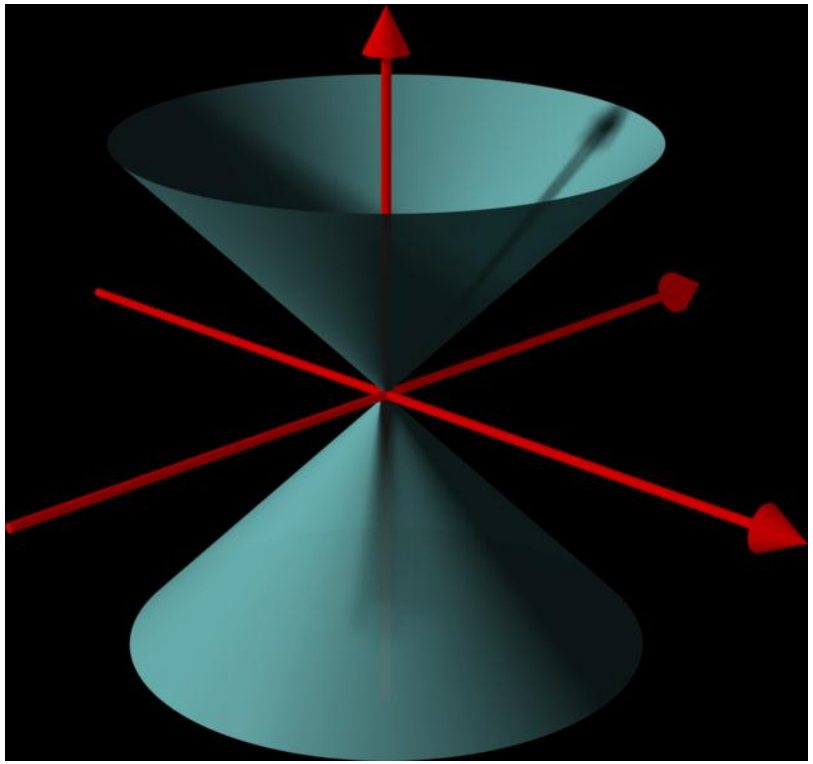


(a) bề mặt cầu, (b) bề mặt xuyến và (c) bề mặt parabol.

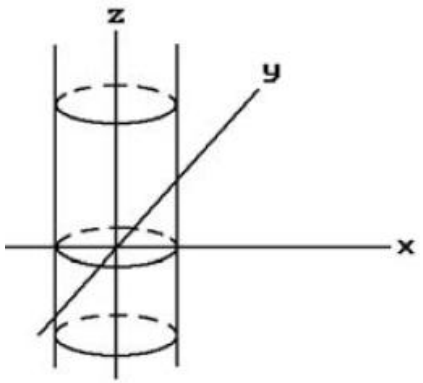


Các bề mặt bậc 2

$$ax^2 + by^2 + cz^2 + dxy + exz + fyz + gx + hy + iz + j = 0$$

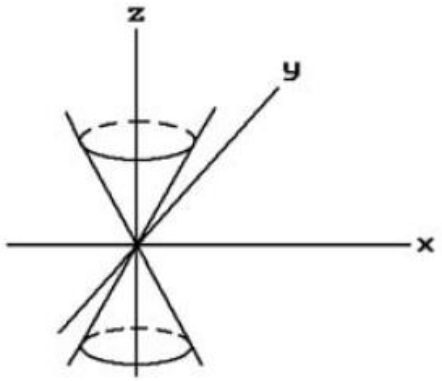


Các bề mặt bậc 2



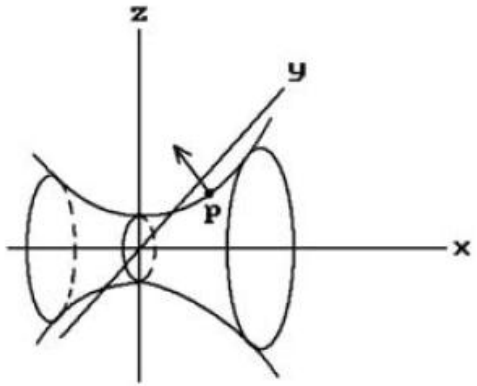
$x^2 + y^2 - 1 = 0$

(a) Cylinder



$x^2 + y^2 - z^2 = 0$

(b) Double cone



$-x^2 + y^2 + z^2 - 1 = 0$

(c) Hyperboloid

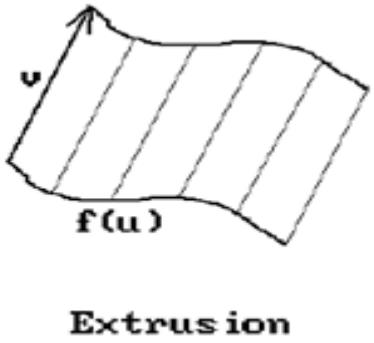
Các bề mặt theo qui tắc

Bề mặt trồi: Cho một đường cong f: [a,b] → **R**3 và vectơ v ∈ **R**3, bề mặt tham số p: [a,b] ×[0,1] → **R**3

được định nghĩa bởi $p(u, t) = f(u) + tv$

được gọi là một bề mặt trồi (extrusion).

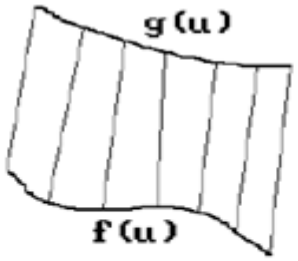
Véc-tơ *v* được gọi là véc-tơ quét của bề mặt trồi.



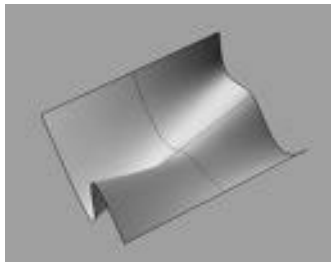
Extrusion

Các bề mặt theo qui tắc

Bề mặt lofted: Cho trước 2 đường cong f và g: [a, b] → R3, bề mặt tham số p: [a,b] ×[0,1] → R3
được xác định bởi p(u, v) = (1 - v)f(u) + vg(u) (8.3)
được gọi là một bề mặt lofted

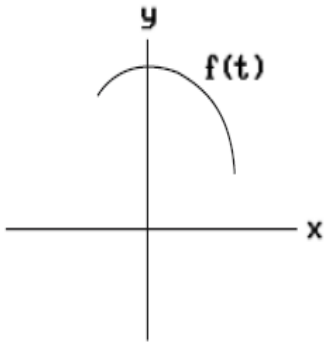


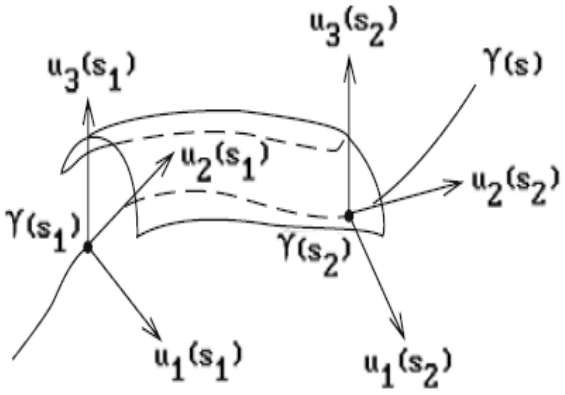
Lofted surface



Các bề mặt quét

Quét một tập (đường cong hoặc khối hình) dọc theo một đường cong





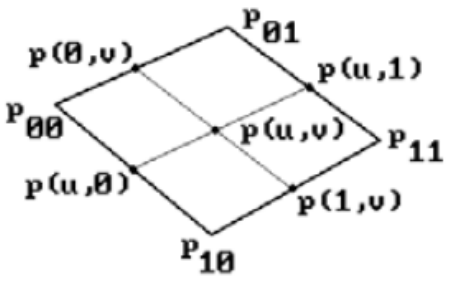
Các bề mặt song tuyến

Cho điểm p00, p01, p10 và p11. Định nghĩa:

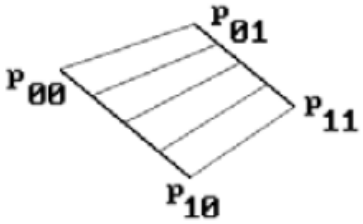
p(u,v) = (1-v)[(1-u)p00 + u.p10] + v[(1-u)p01 + u.p11],
= (1-u)[(1-v)p00 + v.p01] + u[(1-v)p10 + v.p11],
= (1-u)(1-v)p00 + (1-u)v.p01 + u(1-v)p10 + u.vp11

$$p(u,v) = (1-u\ u)\begin{pmatrix} \mathbf{p}_{00} & \mathbf{p}_{01} \\ \mathbf{p}_{10} & \mathbf{p}_{11} \end{pmatrix}\begin{pmatrix} 1-v \\ v \end{pmatrix}.$$

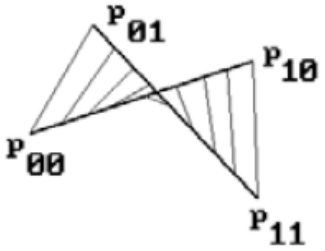
Các bề mặt song tuyến



(a)

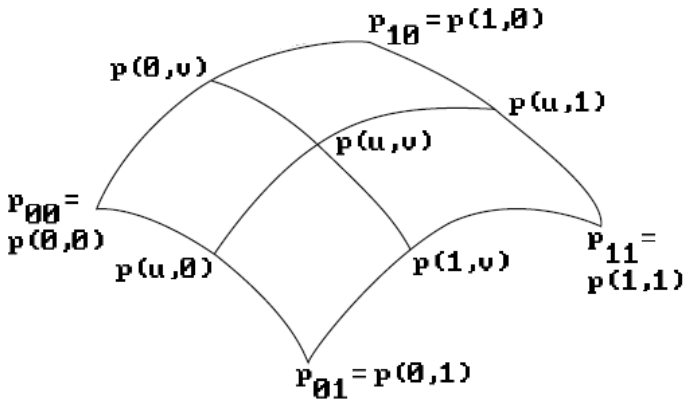


(b)



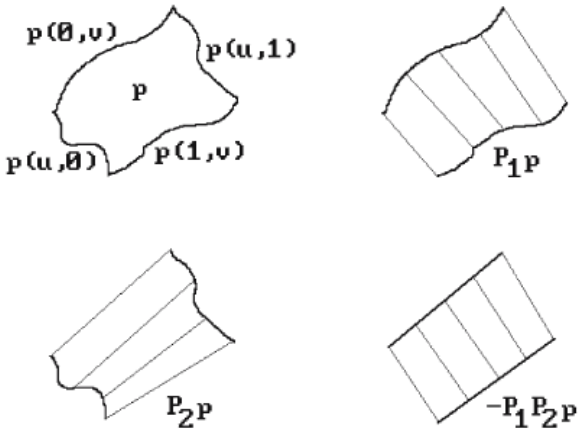
(c)

Các bề mặt Coons



Các bề mặt Coons

$$(P1p)(u,v) = (1 - u)p(0,v) + up(1,v)$$
$$(P2p)(u,v) = (1 - v)p(u,0) + vp(u,1)$$



$$p(u,v) = P1p(u,v) + P2(p - P1p)(u,v)$$
$$= P1p(u,v) + P2p(u,v) - P2P1p(u,v)$$

$$p(u,v) = (1-v)p(u,0) + vp(u,1) + (1-u)p(0,v) + up(1,v)$$
$$- (1-u)(1-v)p(0,0) - (1-u)vp(0,1) - u(1-v)p(1,0) - uvp(1,1).$$

Tổng kết

- Tính liên tục của các đường cong B-spline
- Các bề mặt cong