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.

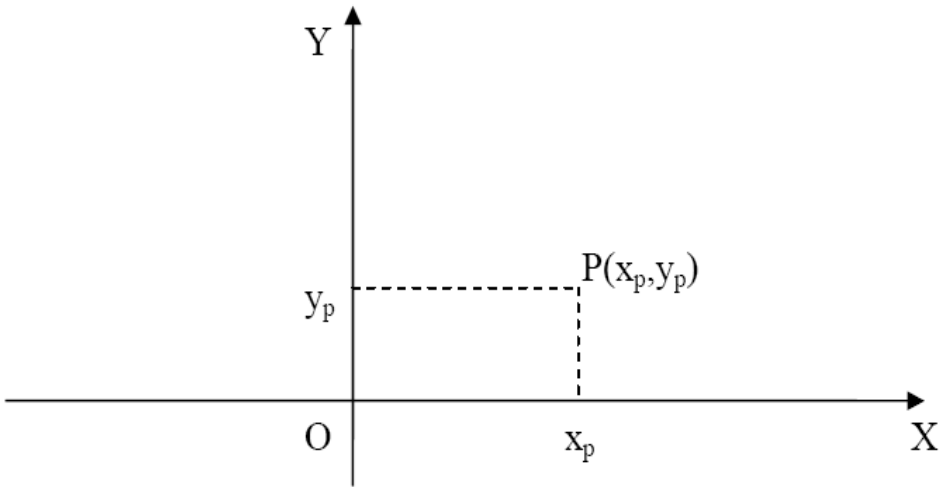
Trong lĩnh vực kỹ thuật đồ họa, chúng ta phải hiểu được rằng thực chất của đồ họa là làm thế nào để có thể mô tả và biến đổi được các đối tượng trong thế giới thực trên máy tính. Các đối tượng trong thế giới thực được mô tả bằng tọa độ trong miền thực. Trong khi đó, hệ tọa độ thiết bị lại sử dụng hệ 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

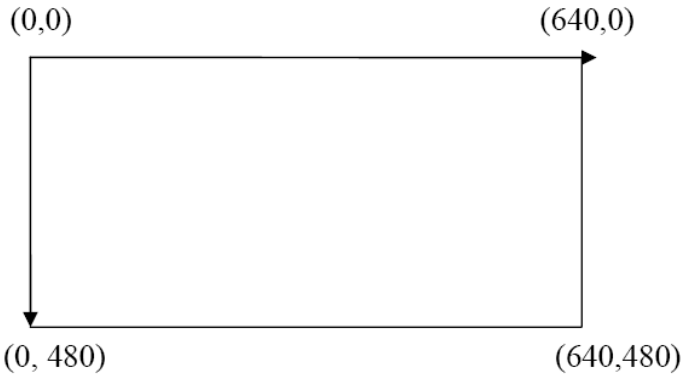
Trong đó :

- 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).



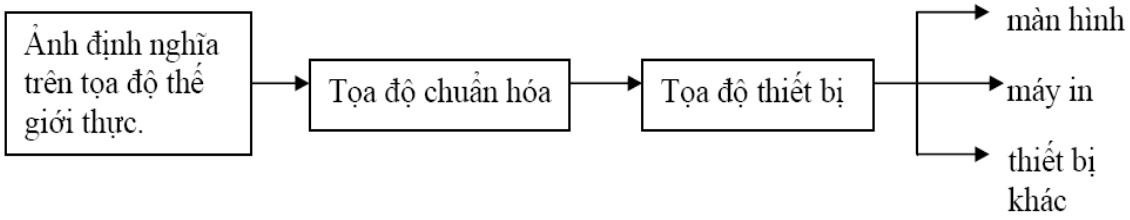
Hệ tọa độ màn hình

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

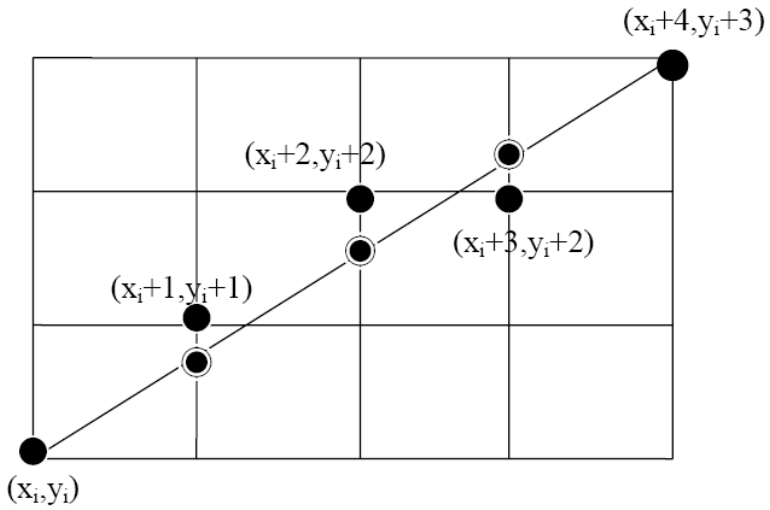
Chương 2

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));
```

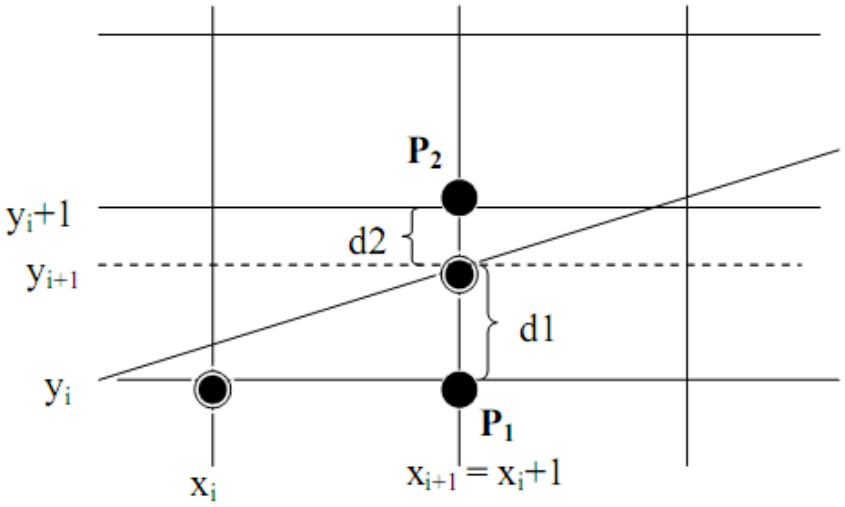
$y = y+m$

}

}

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)

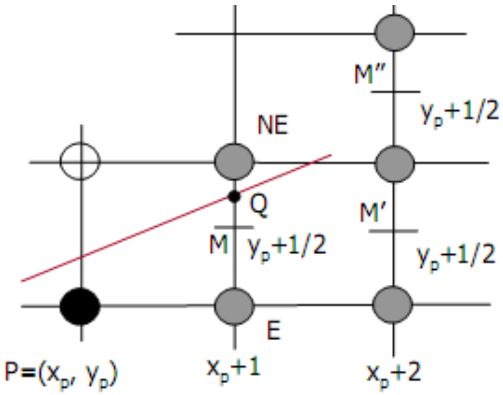
```
{
    int dx, dy, x, y, P, incre1, incre2;
    dx  = x2 - x1;  dy  = y2 - y1;
    P = 2*dy - dx;
    incre1 = 2*dy ; incre2  = 2*(dy - dx) ;
    x= x1; y=y1;
    glVertex2i(x, y);
    while (x < x2 )
    {
        x  = x +1 ;
        if (P < 0)      P = P + incre1
        else
        {
            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

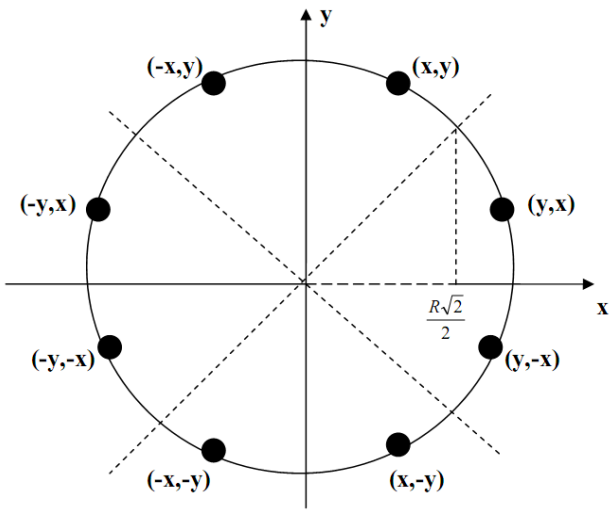
Trong hệ tọa độ Descartes, phương trình đường tròn bán kính R có dạng:

- 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 $\theta \in [0, 2\pi]$.



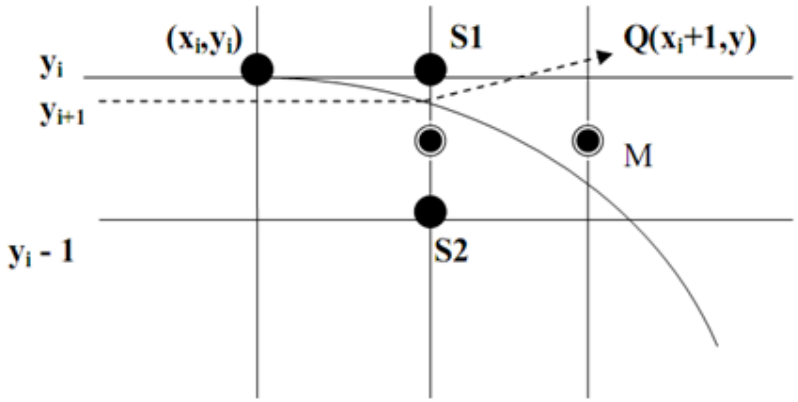
Hình 2.5: Đối xứng 8 điểm trong đường tròn

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;
      }
}
```

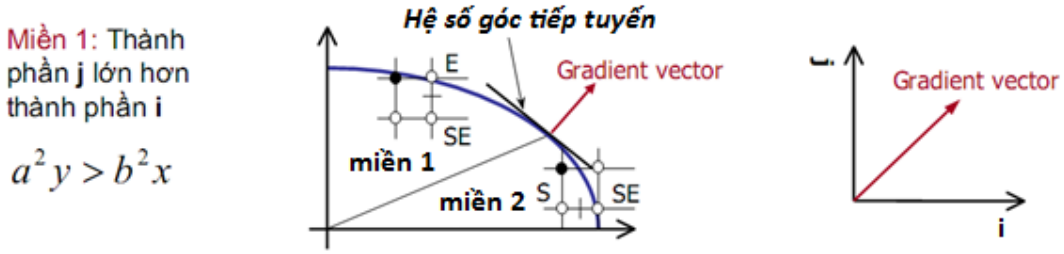
2.3 Thuật toán MidPoint vẽ Ellipse

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



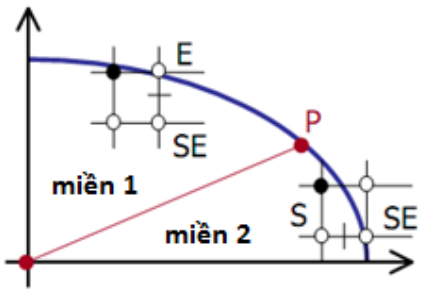
Hình 2.7: Phân chia hai miền của elíp

- **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

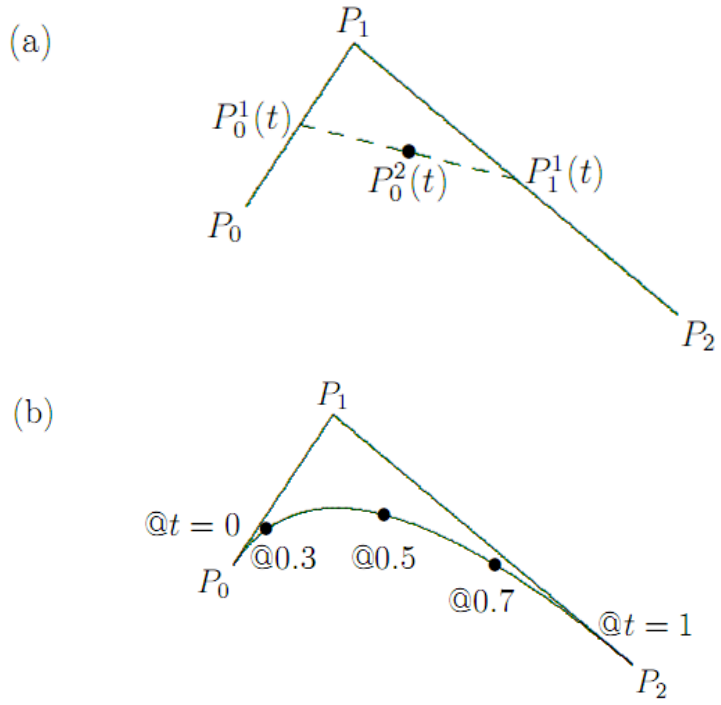
Thuật toán de Casteljau dựa trên dãy các điểm điều khiển để xây dựng với giá trị t trong đoạn [0, 1] tương ứng với một điểm $P(t)$. Do đó, thuật toán sinh ra một dãy các điểm từ các điểm điều khiển cho trước. Khi các điểm điều khiển thay đổi, đường cong sẽ thay đổi theo. Cách xây dựng đường cong dựa 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:

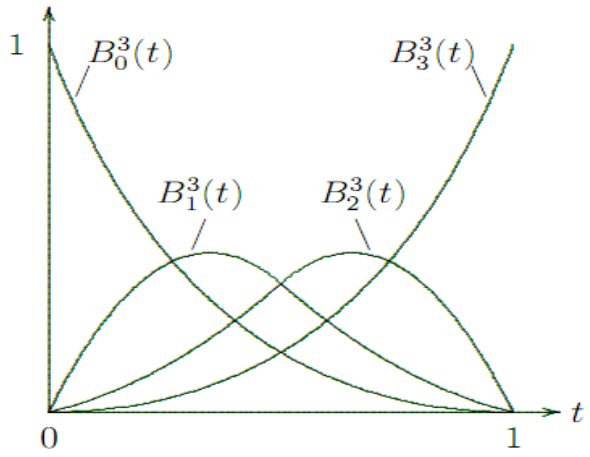
$$B_0^3(t) = \binom{3}{0} (1-t)^3 . t^0 = (1-t)^3$$

$$B_1^3(t) = \binom{3}{1} (1-t)^2 . t^1 = (1-t)^2 . t$$

$$B_2^3(t) = \binom{3}{2} (1-t)^1 . t^2 = (1-t) . t^2$$

$$B_3^3(t) = \binom{3}{3} (1-t)^0 . t^3 = t^3$$

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

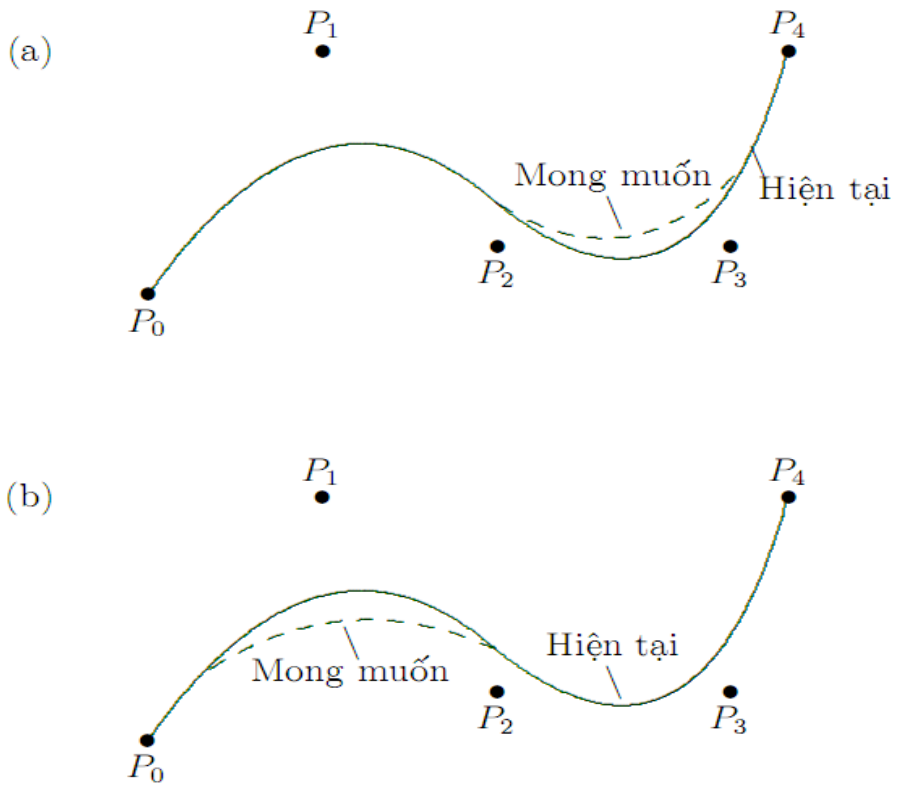
```
Point Horner_Bezier(float t)
{
    int i, L_choose_i;
    float Fact, s;
    Point Q;
    s = 1.0 - t;
    Fact = 1.0;
    L_choose_i = 1;
    Q.x = P[0].x*s;
    Q.y = P[0].y*s;
    for(i = 1; i < L; i++)
    {
        Fact *= t;
        L_choose_i *= (L - i + 1)/i;
        Q.x = (Q.x + Fact*L_choose_i*P[i].x)*s;
        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

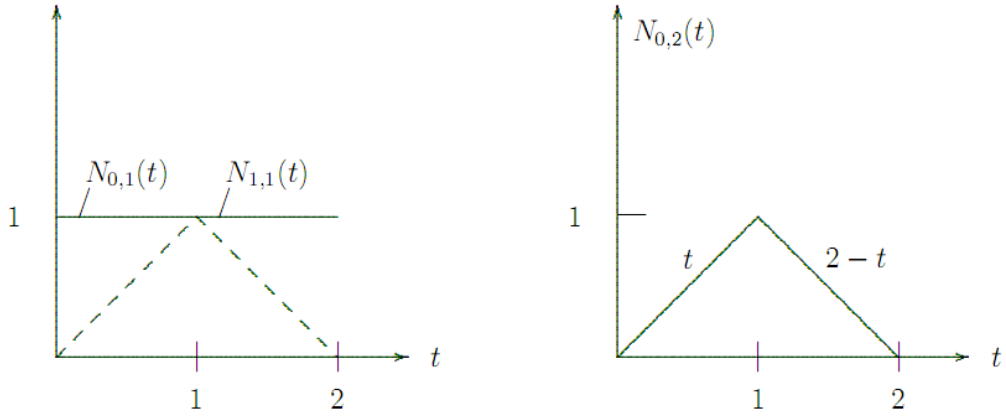
$$N_{k,1}(t) = \begin{cases} 1 \text{ nếu } t_k < t < t_{k+1} \\ 0 \text{ nếu ngược lại} \end{cases}$$

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).



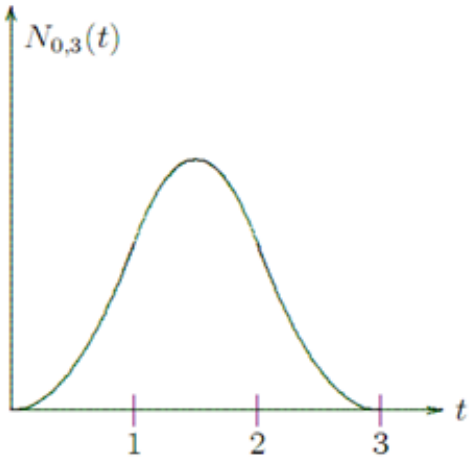
Hình 2.12: Đồ thị các hàm B-spline tuyến tính.

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**

$$N_{0,3}(t) = \frac{t}{2}N_{0,2}(t) + \frac{3-t}{2}N_{1,2}(t)$$

$$= \begin{cases} \frac{1}{2}t^2 \text{ nếu } 0 \le t \le 1 \\ \frac{3}{4} - (t - \frac{3}{2})^2 \text{ nếu } 1 \le t \le 2 \\ \frac{1}{2}(3-t)^2 \text{ nếu } 2 \le t \le 3 \\ 0 \text{ nếu ngược lại} \end{cases}$$

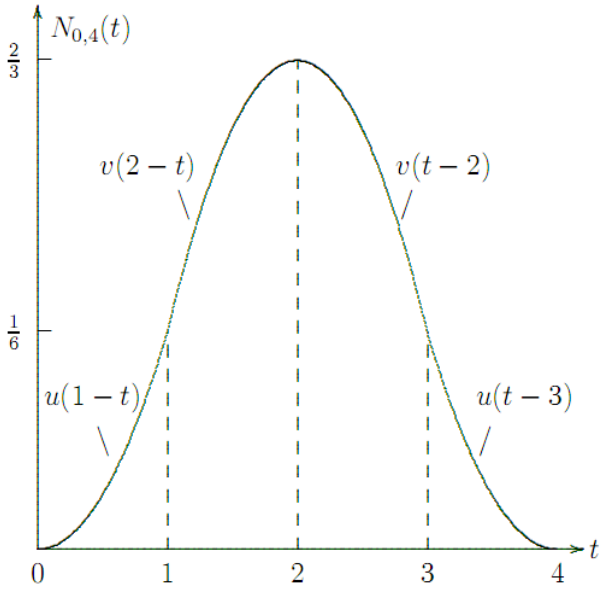


Hình 2.13: Đồ thị hàm B-Spline bậc 2(m=2)

$$N_{0,4}(t) = \begin{cases} u(1-t) \text{ nếu } 0 \le t \le 1 \\ v(2-t) \text{ nếu } 1 \le t \le 2 \\ v(t-2) \text{ nếu } 2 \le t \le 3 \\ u(t-3) \text{ nếu } 3 \le t \le 4 \\ 0 \text{ nếu ngược lại} \end{cases}$$

$$u(t) = \frac{1}{6}(1-t)^3$$

$$v(t) = \frac{1}{6}(3t^3 - 6t^2 + 4)^3$$



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).

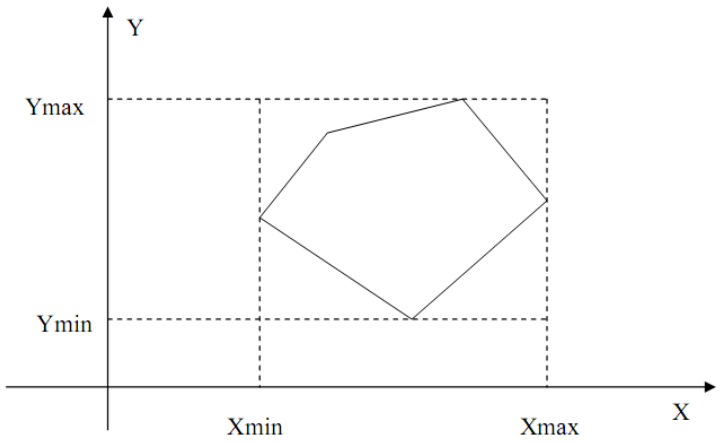
3.2 Tô màu đơn giản

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).

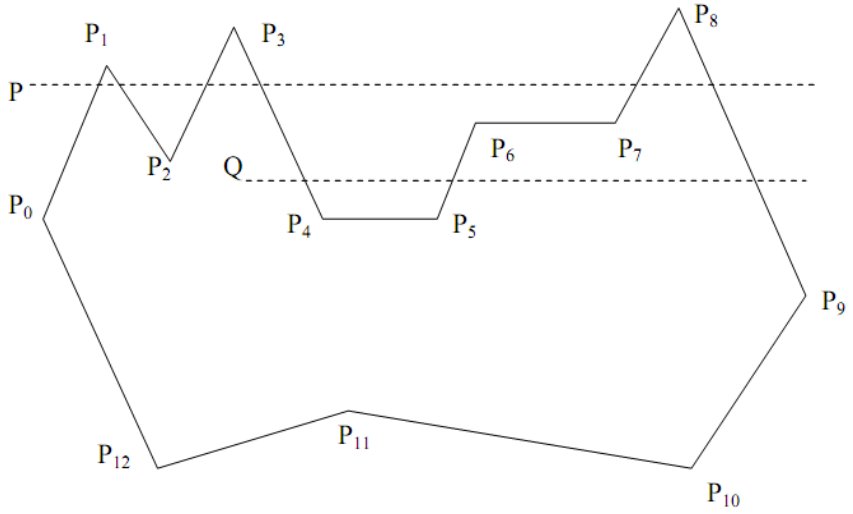


Hình 3.1: Đa giác nội tiếp hình chữ nhật.

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

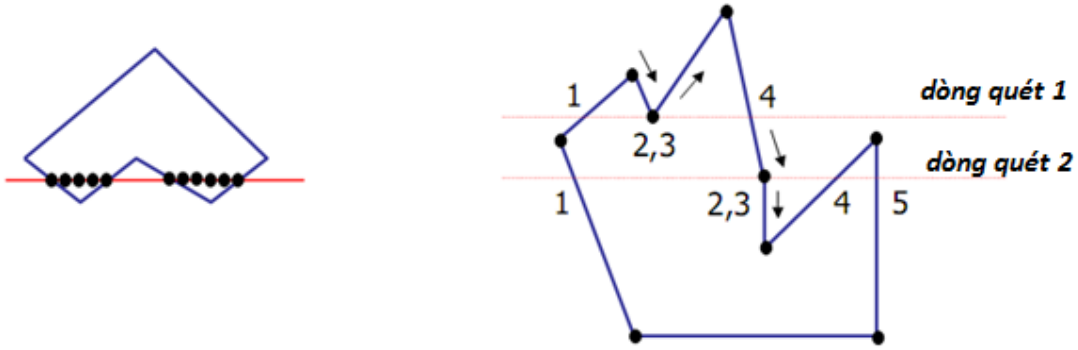
```
void TomauDagiac ( Point[] d; int n, int maubien)
{
        int  x, y;
        Point  P;
        for (x=xmin; x<= xmax; x++)
        for ( y= ymin; y<= ymax; y++)
        {
                P.x= x;   P.y= y;
                if  pointInpoly (d, P, n)
```

```
                    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.



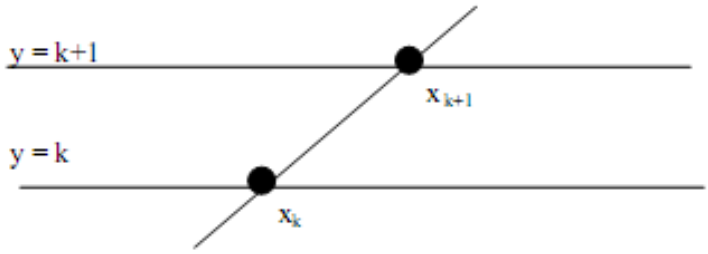
Hình 3.3: Tô màu theo dòng quét.

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:

$$\begin{cases} Q_x = a.P_x + c.P_y + tr_x \\ Q_y = b.P_x + d.P_y + tr_y \end{cases}$$

$$(Q_x, Q_y) = (P_x, P_y).\begin{pmatrix} a & b \\ c & d \end{pmatrix} + (tr_x, tr_y)$$

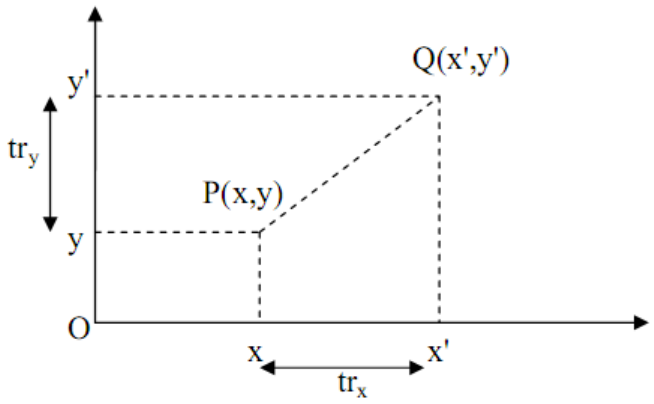
$$hay\ Q = P.M + tr$$

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

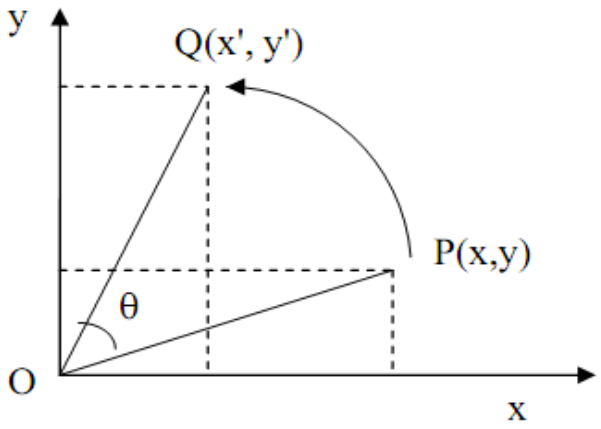
Phép quay làm thay đổi hướng của đối tượng. Một phép quay đòi hỏi phải có tâm quay, góc quay. Góc quay dương thường được qui ước là chiếu ngược chiều kim đồng hồ.

- **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):

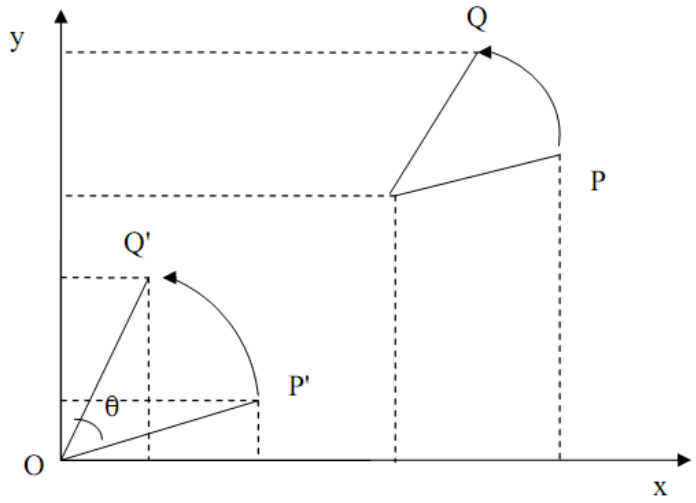
$$\begin{cases} x' = x.cos\theta - y.sin\theta \\ y' = x.sin\theta + y.cos\theta \end{cases}$$

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:

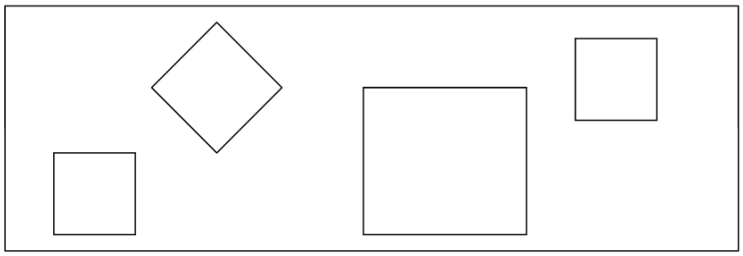
$$\begin{cases} Q_x = P_x.S_{x1}.S_{x2} \\ Q_y = P_y.S_{y1}.S_{y2} \end{cases}$$

$$M = \begin{pmatrix} S_{x1} & 0 & 0 \\ 0 & S_{y1} & 0 \\ 0 & 0 & 1 \end{pmatrix}\begin{pmatrix} S_{x2} & 0 & 0 \\ 0 & S_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} S_{x1}.S_{x2} & 0 & 0 \\ 0 & S_{y1}.S_{y2} & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

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

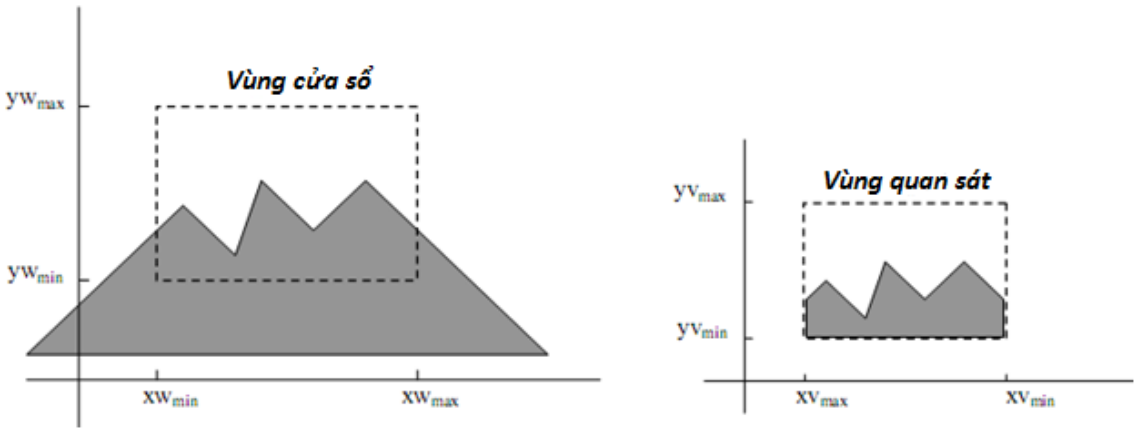
Nội dung chính

- 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:

$$x = x_A + t(x_B - x_A)$$

$$y = y_A + t(y_D - y_A)$$

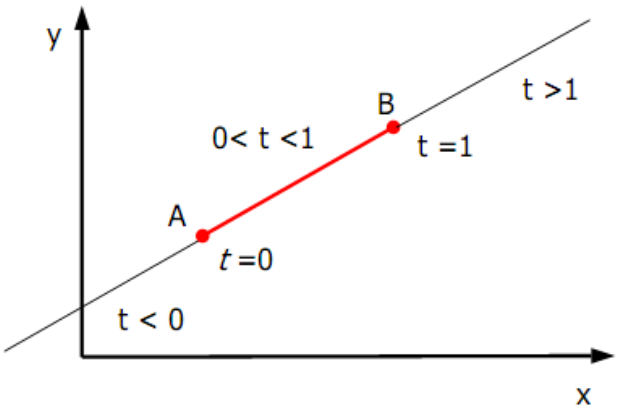
$$x = x_C + t(x_D - x_C)$$

$$y = y_C + t(y_D - y_C)$$

$$0 \leq t \leq 1 \text{ và } 0 \leq s \leq 1$$

$$t = \frac{(x_C - x_A)(y_C - y_D) - (x_C - x_D)(y_C - y_A)}{(x_C - x_A)(y_C - y_D) - (x_C - x_A)(y_C - y_A)}$$

$$s = \frac{(x_B - x_A)(y_C - y_A) - (x_C - x_A)(y_B - y_A)}{(x_B - x_A)(y_C - y_D) - (x_C - x_D)(y_B - y_A)}$$



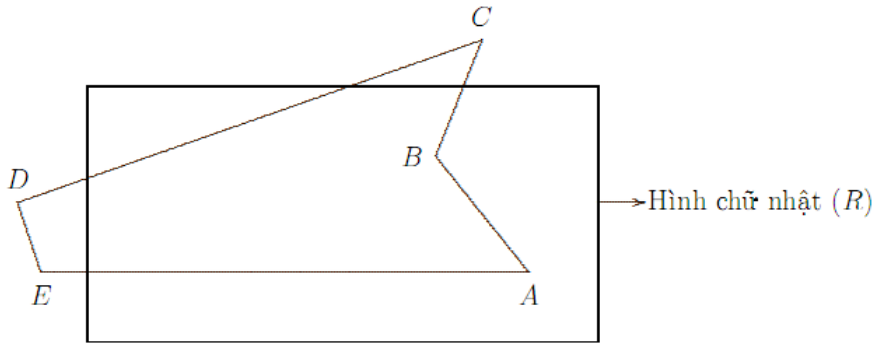
Hình 5.2: Biểu diễn tham số của đoạn thẳng

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:

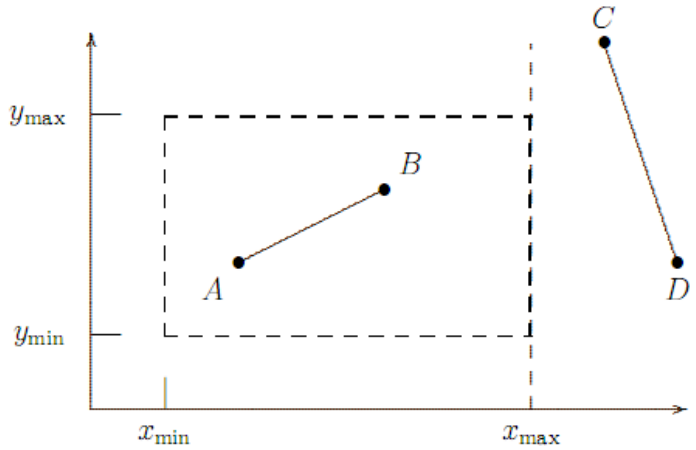
1. Cả hai đầu mút của đoạn thẳng nằm trong hình chữ nhật, chẳng hạn AB.
2. Một trong hai đầu mút của đoạn thẳng nằm trong hình chữ nhật, chẳng hạn BC.
3. Cả hai đầu mút của đoạn thẳng nằm ngoài hình chữ nhật nhưng có giao điểm, chẳng hạn CD.

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.



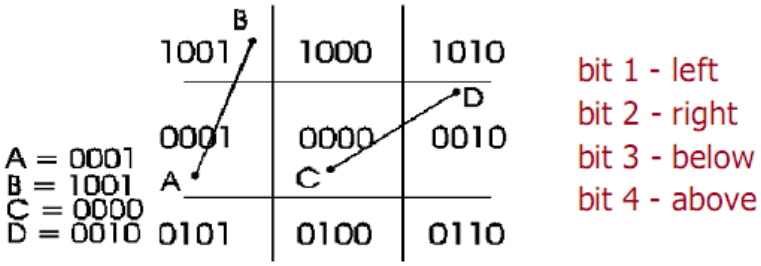
Hình 5.4: Hai trường hợp tầm thường của giao đoạn thẳng và hình chữ nhật

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

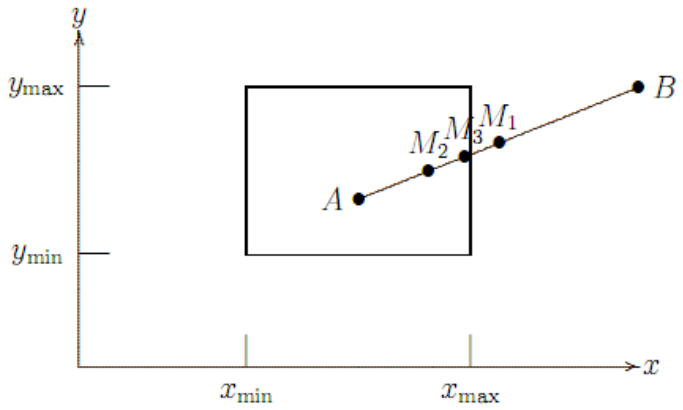
       Đặt M là trung điểm của đoạn CD.

       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.

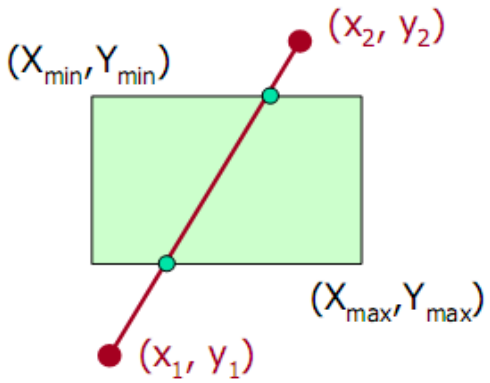


Hình 5.6: Minh họa của thuật toán chia nhị phân.

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 :

$$C = A + \frac{2}{11}(B - A) = (1,-2) + \frac{2}{11}[(10,9) - (1,-2)] = (2\frac{7}{11},0)$$

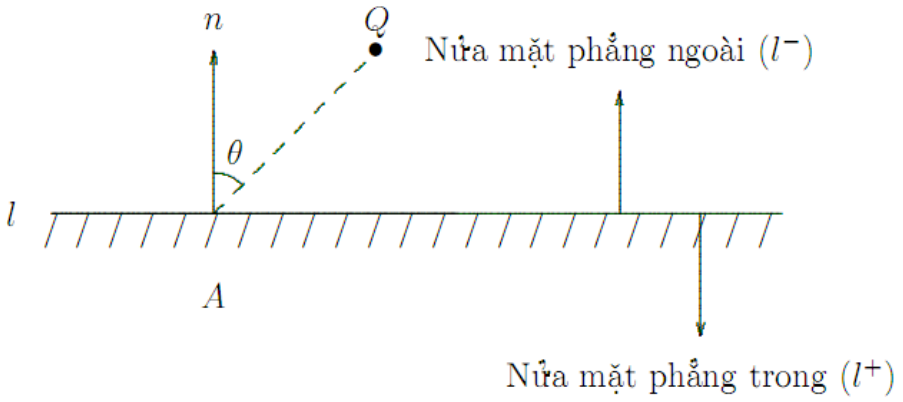
$$D = A + \frac{6}{11}(B - A) = (1,-2) + \frac{6}{11}[(10,9) - (1,-2)] = (5\frac{10}{11},4)$$

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:

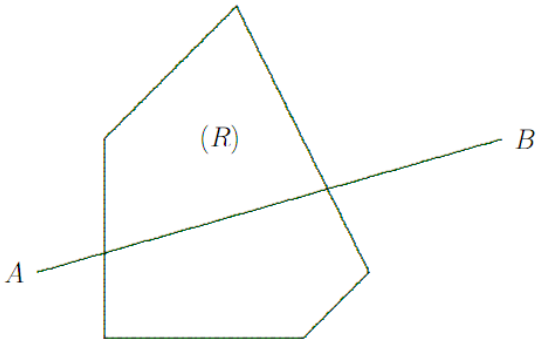
$$(l^+) := \{P \in R^2 | \langle \overrightarrow{OP}, n \rangle \geq D\}$$

$$(l^-) := \{P \in R^2 | \langle \overrightarrow{OP}, n \rangle < D\}$$

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)
```

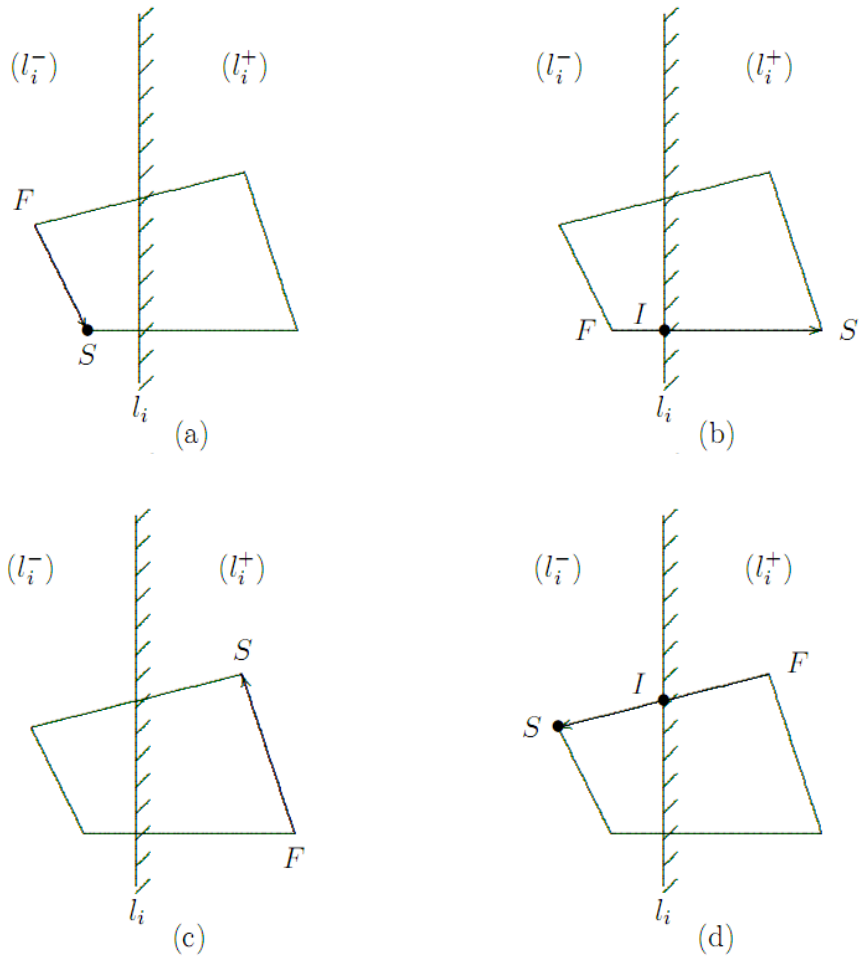
```
                    if (t_out > t_hit) t_out = t_hit;
                else
                    if (t_in < t_hit) t_in = t_hit;
                if (t_in > t_out) return false;
            }
            F=S;
        }
        F.x = (1 - t_in)*(*A).x + t_in*(*B).x;
        F.y = (1 - t_in)*(*A).y + 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

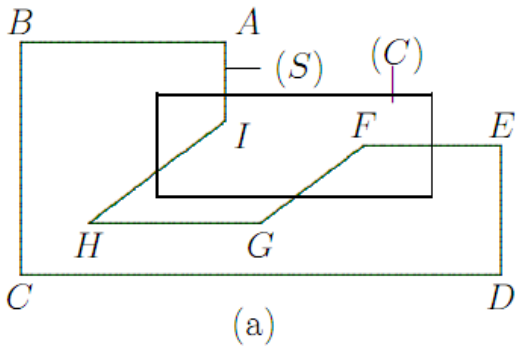
Ký hiêu Subj và Clip là danh sách các đỉnh của hai đa giác *(S)* và *(C)* tương ứng. Có bốn khả năng xảy ra giữa mỗi cạnh của *(S)* và của *(C):*

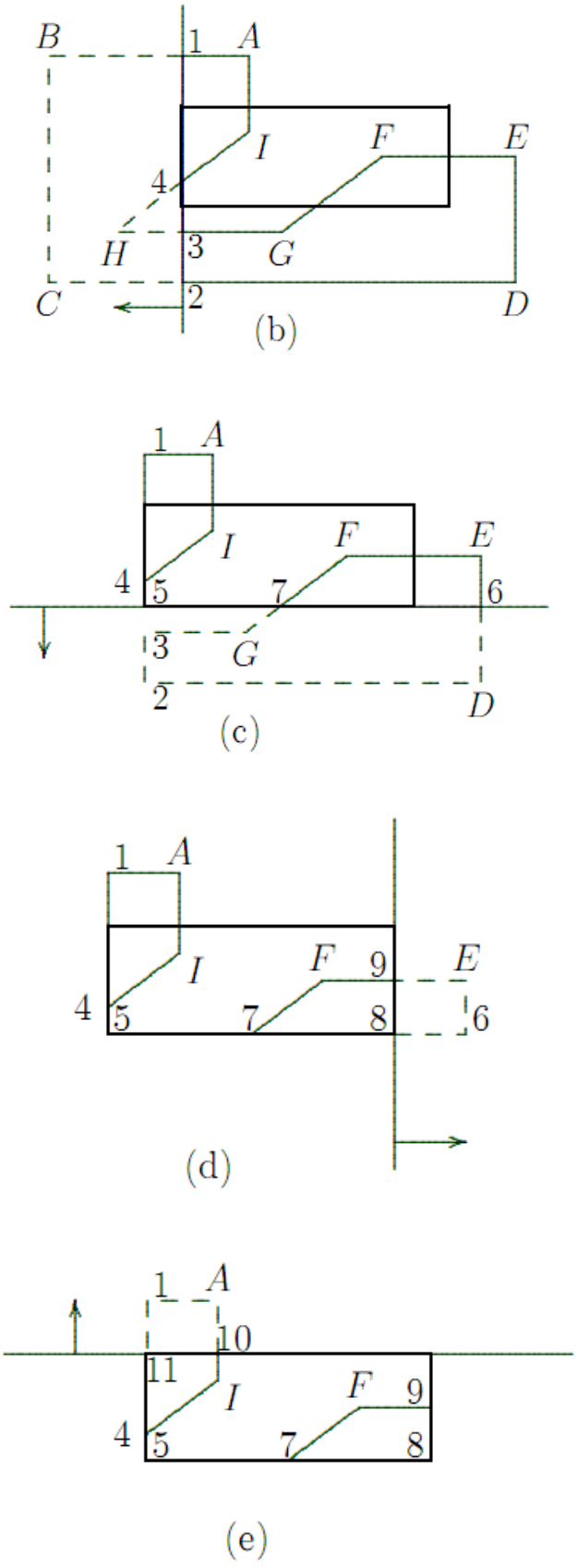
1. Hai đỉnh F và S nằm trong: xuất S.
2. Đỉnh F nằm trong và S nằm ngoài: tìm giao điểm I và xuất nó.
3. Hai đỉnh F và S nằm ngoài: không xuất.
4. Đỉnh F nằm ngoài và S nằm trong: tìm giao điểm I; xuất I và S.



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)*:

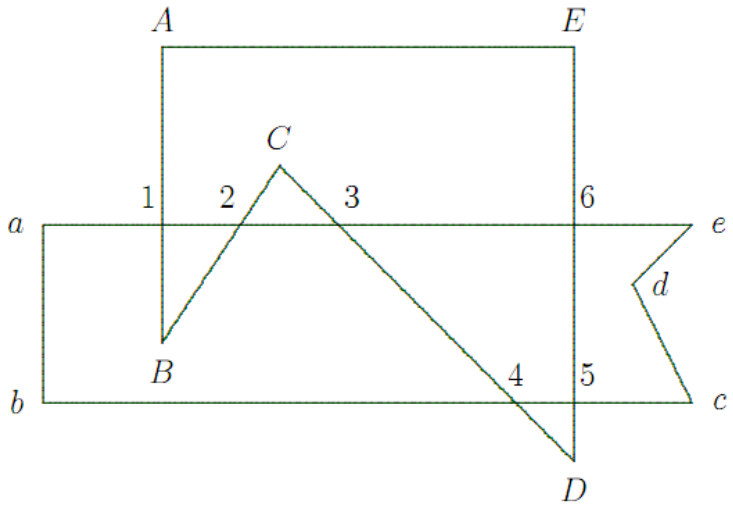
   (8, 9, F, 7, 5, 4, I, A, 1, 5, 8).

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

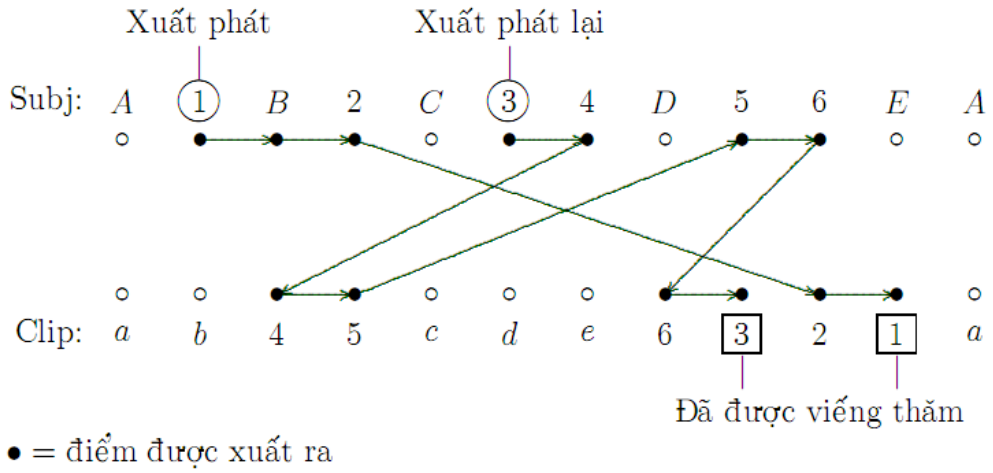
Hai đa giác *(S)* và *(C)* được biểu diễn bởi danh sách các đỉnh, ký hiệu *Subj = (A, B, C, D, E, A)* và *Clip = (a, b, c, d, e, a)* tương ứng.

- 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.

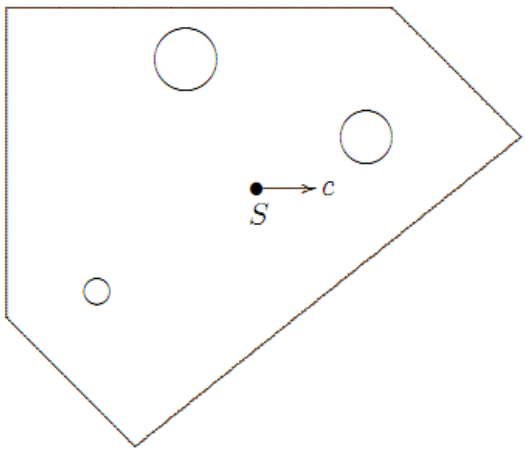
- (*C*) \ (*S*)*:* (1, *B,* 2, 3, 4, *b, a,* 1)*; và* (5, 6, *e, d, c*, 5).
- (*S*) \ (*C*)*:* (4, 5, *D,* 4)*; và* (6, 3*, C,* 2, 1, *A, E*, 6).

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.

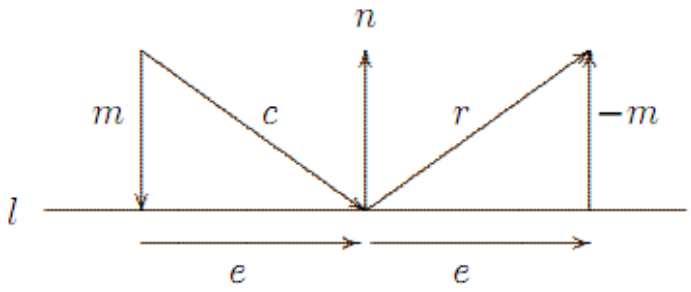
Bây giờ ta sẽ xây dựng thuật toán xác định đối tượng giao với tia sáng sẽ trước và vị trí va trạm tại đó. Ví trí va chạm là điểm khởi đầu cho cho đường đi kế tiếp với hướng di chuyển mới.



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 :

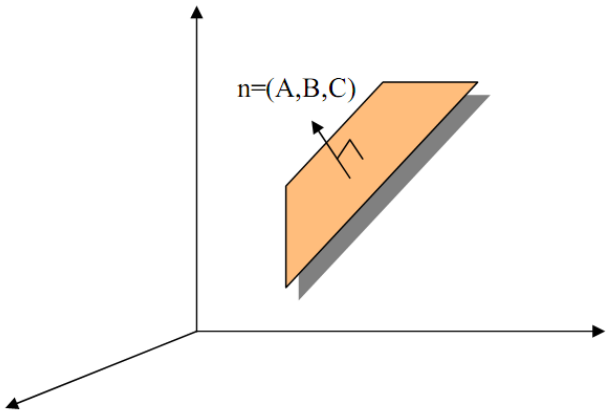
$$A = y_1 (z_2 - z_3) + y_2 (z_3 - z_1) + y_3 (z_1 - z_2)$$

$$B = z_1 (x_2 - x_3) + z_2 (x_3 - x_1) + z_3 (x_1 - x_2)$$

$$C = x_1 (y_2 - y_3) + x_2 (y_3 - y_1) + x_3 (y_1 - y_2)$$

$$A = - x_1 (y_2 z_3 - y_3 z_2) - x_2 (y_3 z_1 - y_1 z_3) - x_3 (y_1 z_2 - y_2 z_1)$$

Hướng của mặt phẳng thường được xác định thông qua véc tơ pháp tuyến của nó. Véc tơ pháp tuyến n = (A,B,C).

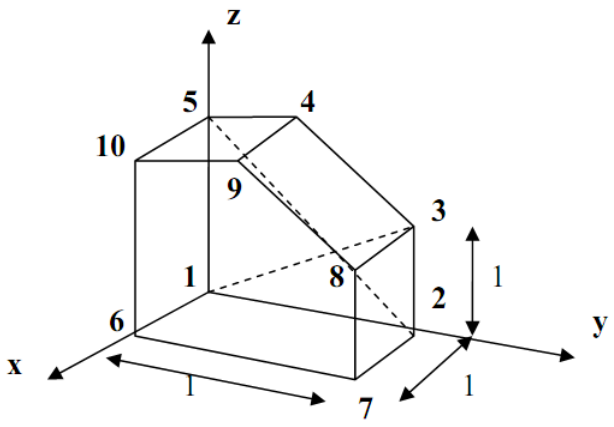


Hình 5.15: Mặt phẳng trong không gian

• 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	

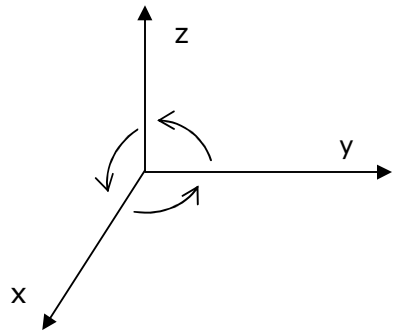
Edge List		
Edge	**Vertex1**	**Vertex2**
1	1	2
2	2	3
3	3	4
4	4	5
5	5	1
6	6	7
7	7	8
8	8	9
9	9	10
10	10	6
11	1	6
12	2	7
13	3	8
14	4	9
15	5	10
16	2	5
17	1	3

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

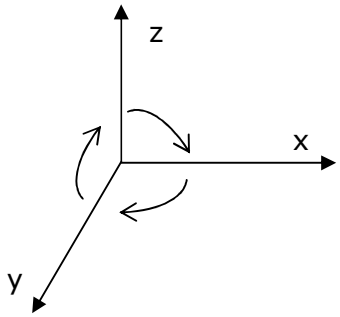
6.3.1. Hệ tọa độ bàn tay phải - bàn tay trái

• 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

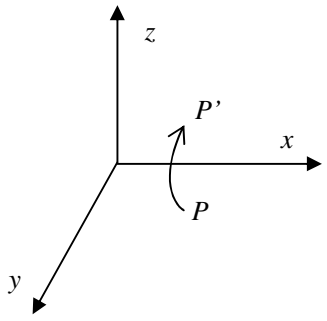
$$Tr(tr_x, tr_y, tr_z) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tr_x & tr_y & tr_z & 1 \end{pmatrix}$$

• 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,

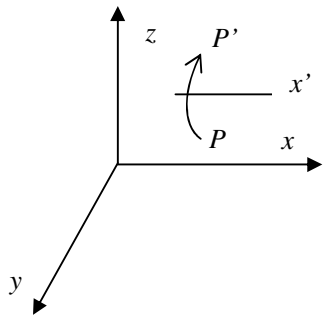
$$T_{z\gamma} = \begin{pmatrix} \cos\gamma & \gamma & 0 & 0 \\ -\sin\gamma & \cos\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

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 *:* $T = T[-1, -1, -1] \times T_{x(30)} \times T[1, 1, 1]$

$$= \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -1 & -1 & -1 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \frac{\sqrt{3}}{2} & \frac{1}{2} & 0 \\ 0 & -\frac{1}{2} & \frac{\sqrt{3}}{2} & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 1 & 1 & 1 & 1 \end{pmatrix}$$

$A(1, 2, 1) \rightarrow A' = (1, 2, 1, 1) \times T$

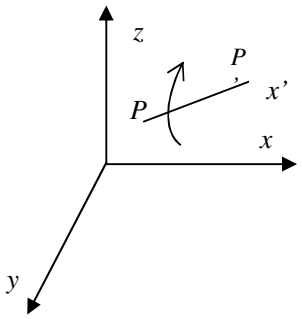
$B(3, 2, 1) \rightarrow B' = (3, 2, 1, 1) \times T$

$C(3, 4, 3) \rightarrow C' = (3, 4, 3, 1) \times T$

$D(1, 4, 3) \rightarrow D' = (1, 4, 3, 1) \times T$

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

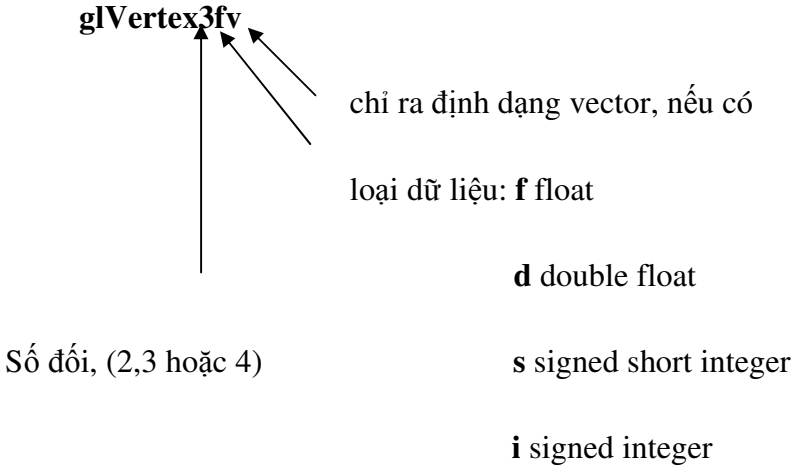
OpenGL là gì?

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);

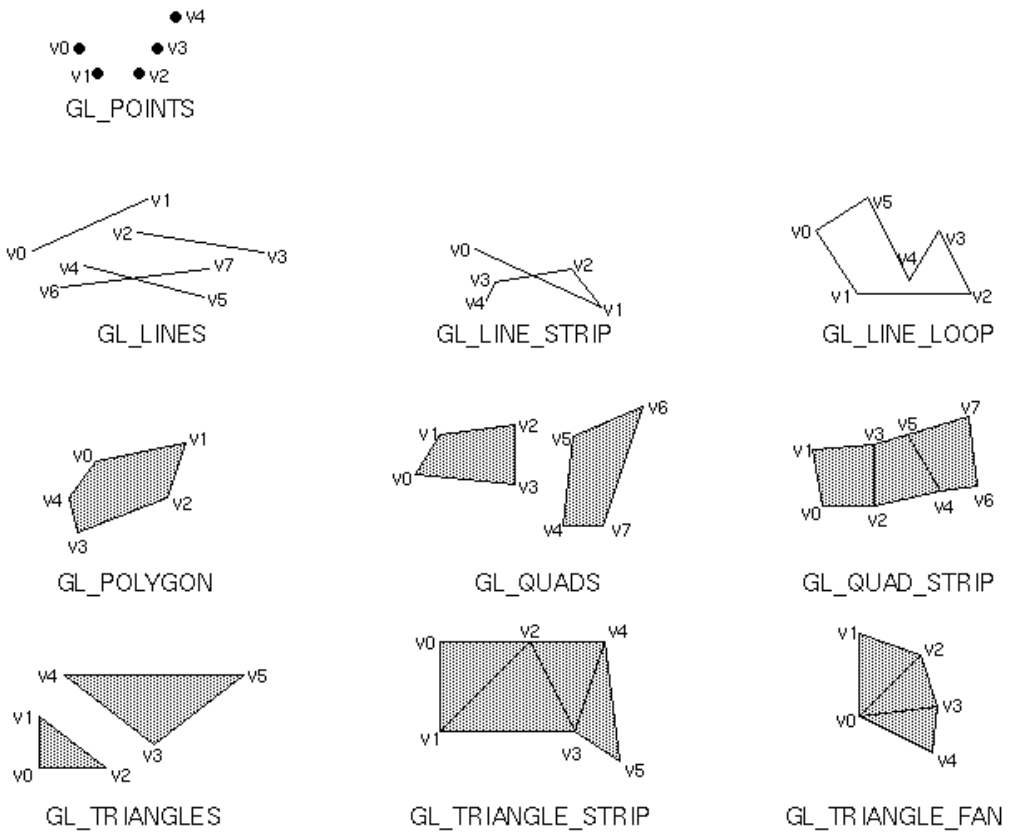
glVertex2f(0.5, 0.0);

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

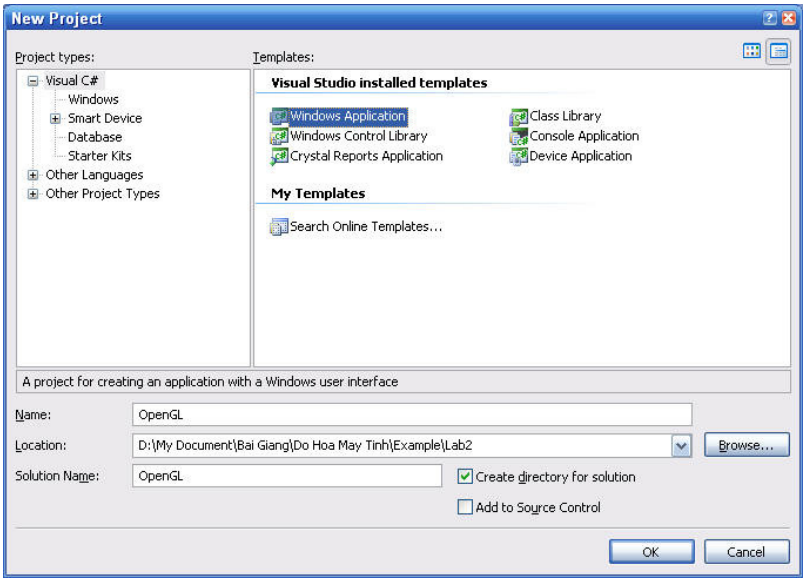
OpenGL chỉ cho phép một số lệnh nằm bên trong glBegin() và glEnd()

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).

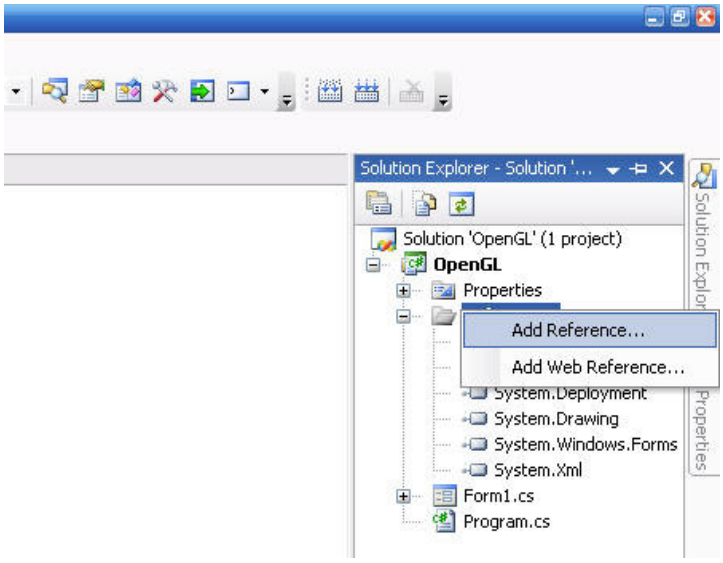
Bắt đầu làm quen OpenGL bằng ngôn ngữ C#

- Mở chương trình Visual Studio .NET (2005) và tạo mới ứng dụng C# trong Windows Application.



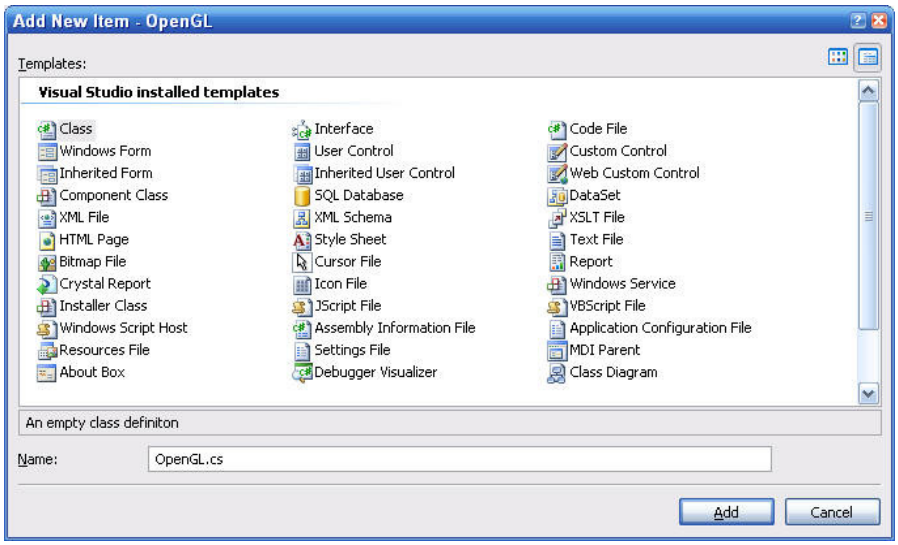
Hình phụ lục 2:Tạo project đồ họa mới

- 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)



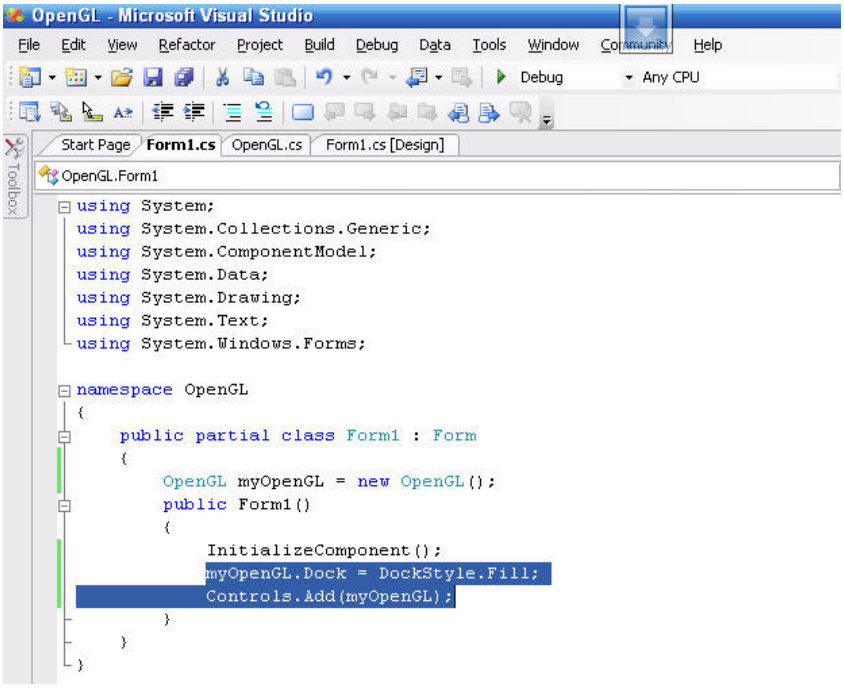
Hình phụ lục 4: Thêm lớp mới vào project

- 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);
        }
    }
}
```

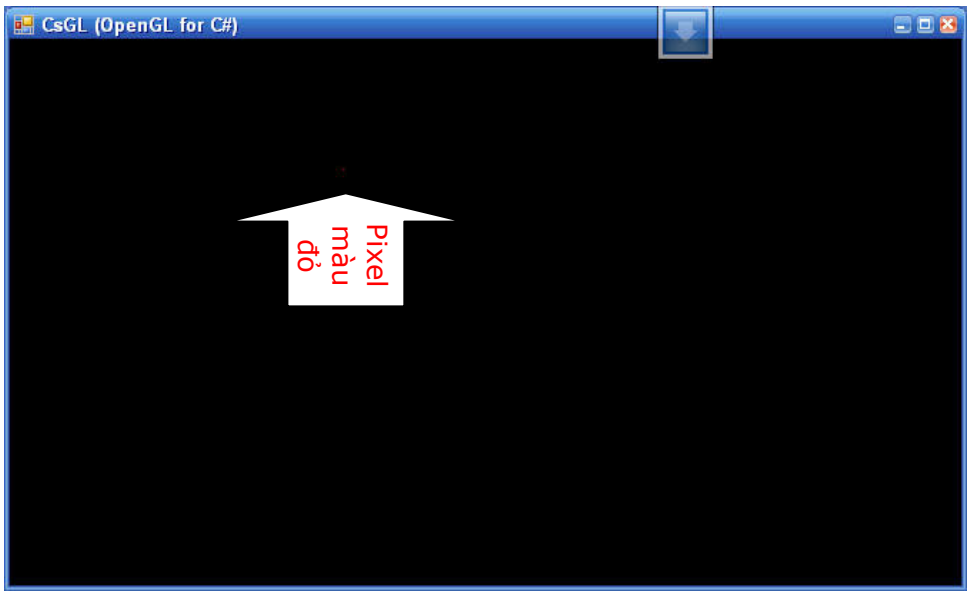
Hình phụ lục 5: Viết code khởi tạo chế độ đồ họa OpenGL

- 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

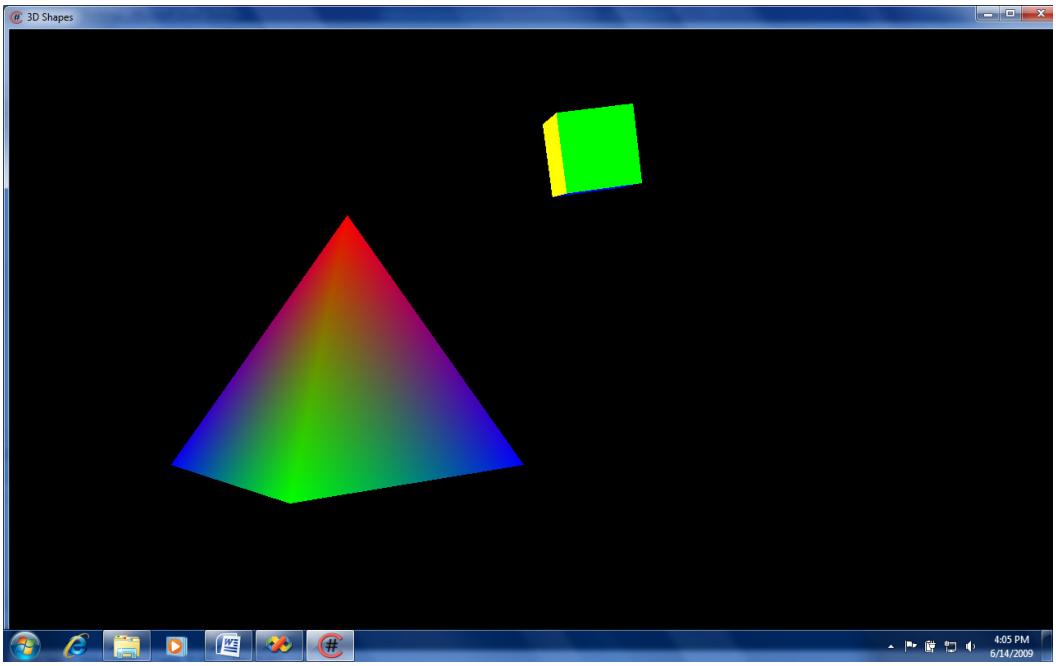
- Chạy chương trình thử (nhớ chép file ***csgl.native.dll*** vào thư mục ***Debug***), nếu thấy xuất hiện một pixel màu đỏ trên nền màu đen là thành công.



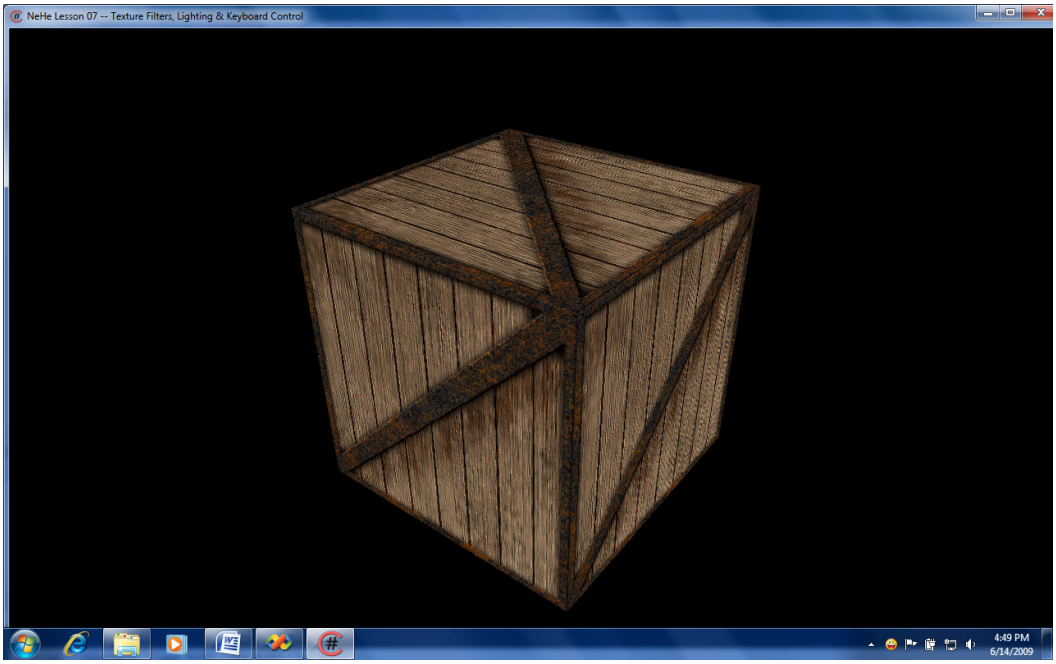
Hình phụ lục 7: Kết quả khởi tạo chế độ đồ họa OpenGL

Vẽ đối tượng 3D trong OpenGL

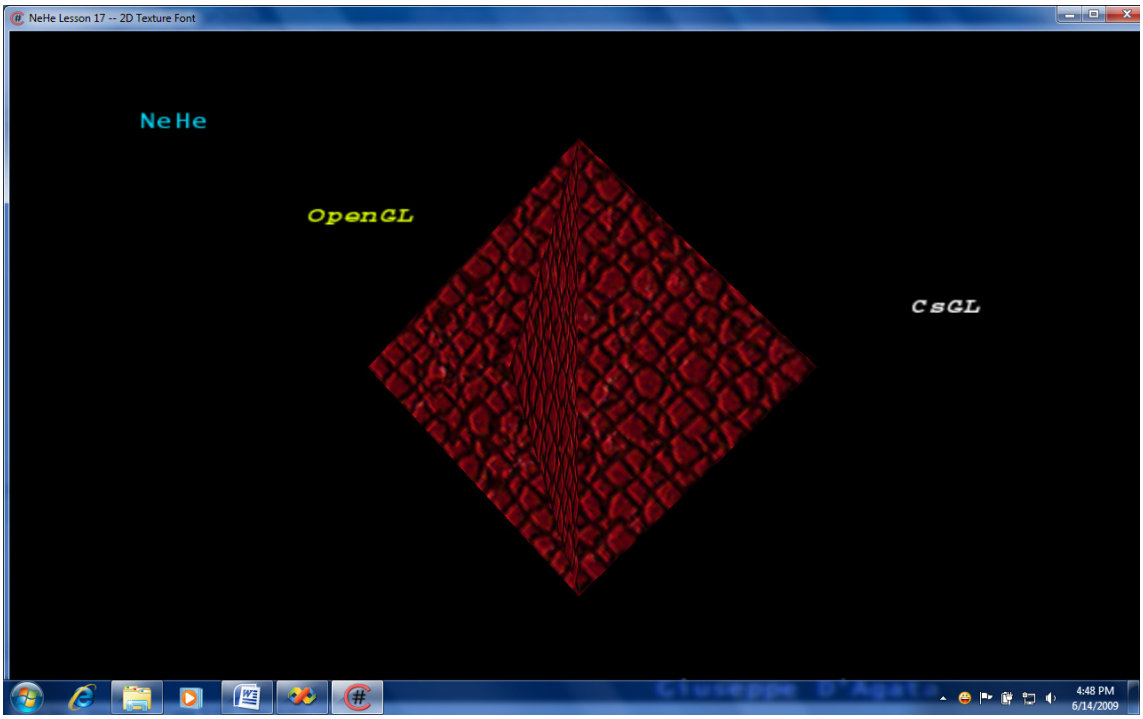
OpenGL cho phép vẽ các đối tượng 3D dễ dàng và tạo các hiệu ứng màu, ánh sáng, biến đổi trong không gian 3D rất chính xác. Sau đây là một số hình ảnh 3D được lập trình từ thư viện OpenGL.



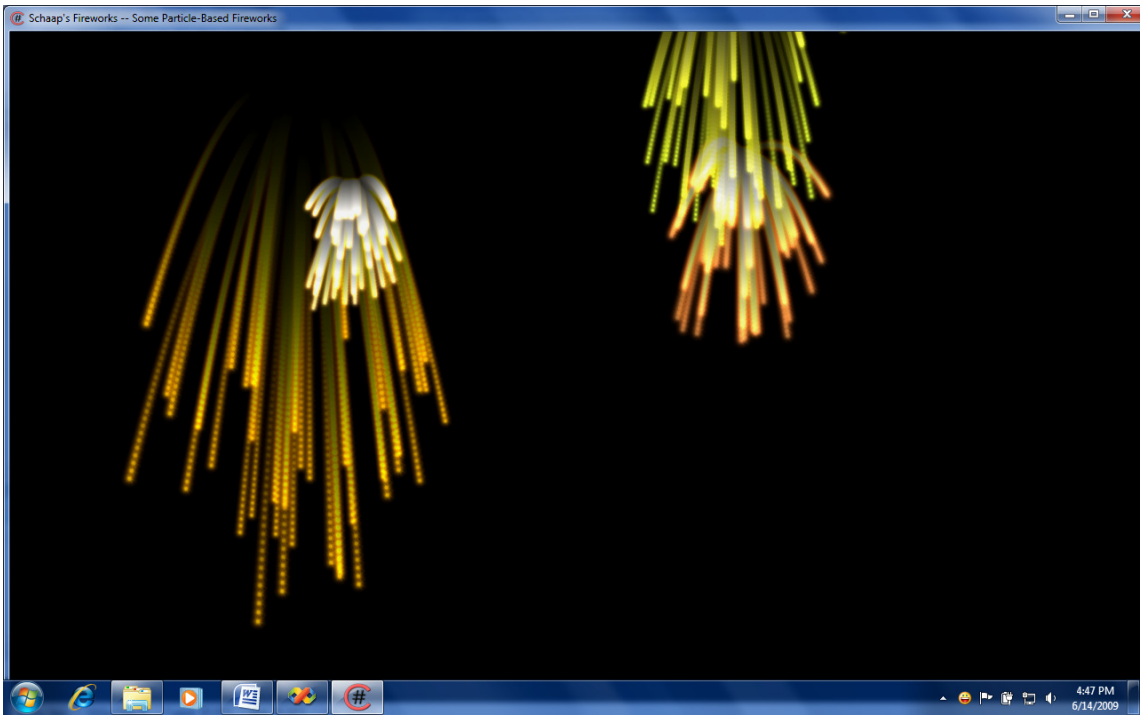
Hình phụ lục 8: Các đối tượng hình học 3D cơ bản



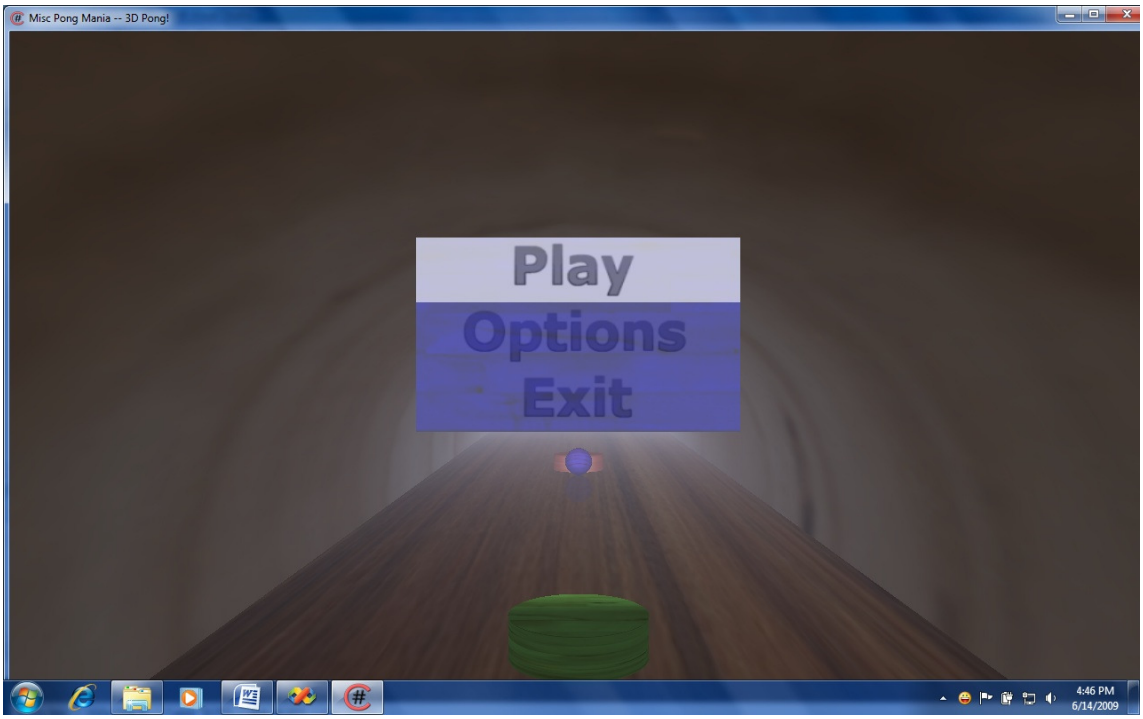
Hình phụ lục 9:Hình khối lập phương hiển thị bằng texture



Hình phụ lục 10: Hình 3D và phông chữ trong đồ họa OpenGL



Hình phụ lục 11: Pháo hoa được biểu diễn bằng OpenGL

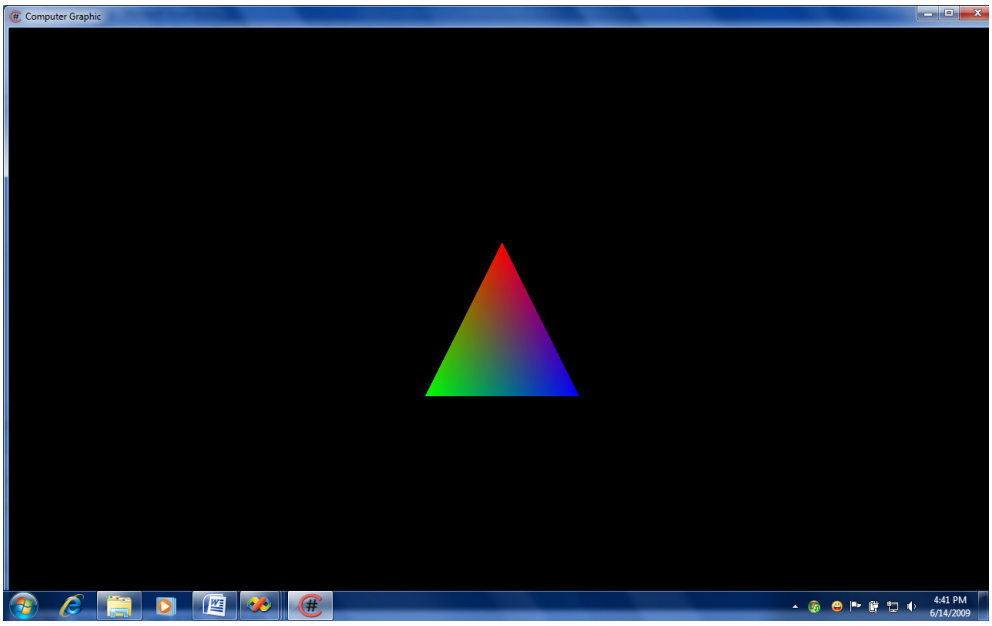


Hình phụ lục 12: Game đơn giản viết bằng OpenGL

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.

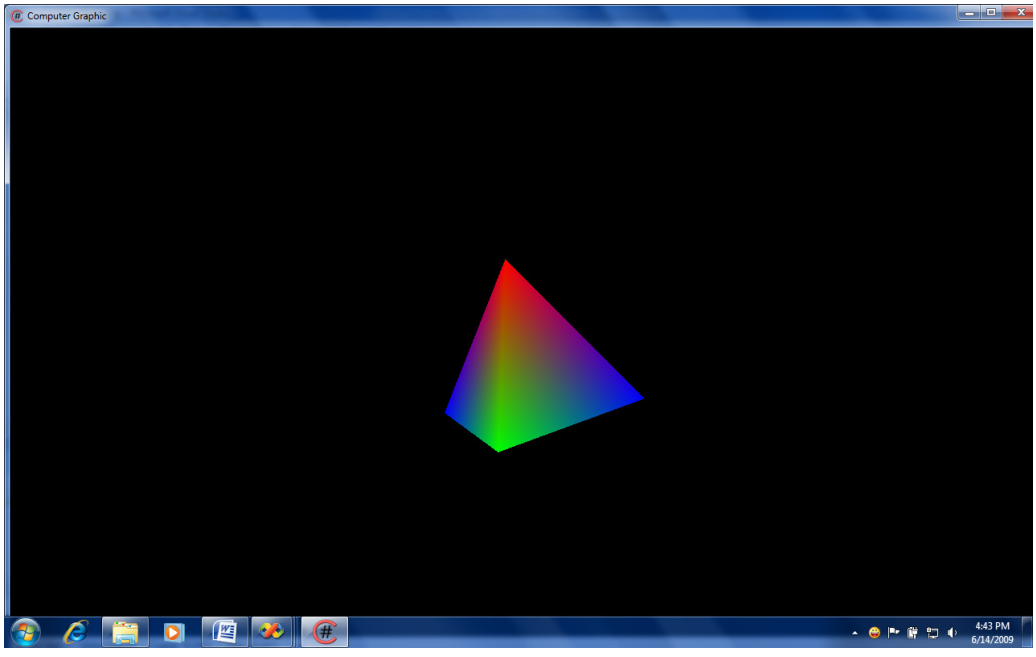
```
glBegin(GL_TRIANGLES);
glColor3f(1.0f, 0.0f, 0.0f);
glVertex3f(0.0f, 100.0f, 0.0f);
glColor3f(0.0f, 1.0f, 0.0f);
glVertex3f(-100.0f, -100.0f, 100.0f);
glColor3f(0.0f, 0.0f, 1.0f);
glVertex3f(100.0f, -100.0f, 100.0f);

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