TRƯỜNG ĐẠI HỌC SƯ PHẠM ĐÀ NẴNG

KHOA TIN HỌC

PHẠM ANH PHƯƠNG

GIÁO TRÌNH

ĐỒ HỌA MÁY TÍNH

LƯU HÀNH NỘI BỘ

Đà Nẵng - 2020

LỜI MỞ ĐẦU

Theo khung chương trình đào tạo ở các hệ cử nhân và kỹ sư của Bộ Giáo Dục và Đào Tạo, **Đồ họa máy tính** là một trong những học phần quan trọng của ngành Công nghệ Thông tin.

Qua nhiều năm nghiên cứu và giảng dạy môn **Đồ họa máy tính** ở các trường Đại học Khoa học Huế, Đại học Sư phạm Đà Nẵng và Huế, Đại học Điện lực Hà Nội và một số trường Đại học khác ở miền Trung và Tây Nguyên, chúng tôi đã cố gắng đúc kết để biên soạn giáo trình **Đồ họa máy tính** nhằm đáp ứng nhu cầu học tập và nghiên cứu của sinh viên chuyên ngành Công nghệ Thông tin, giúp sinh viên có một tài liệu tham khảo tốt khi bước đầu làm quen với các kiến thức cũng như kỹ năng lập trình đồ họa. Đây là môn học khó liên quan đến nhiều kiến thức về toán học như hình học, hình học giải tích, ma trận…, vì vậy để học tốt môn này đòi hỏi sinh viên phải có tư duy toán học và kỹ năng lập trình tốt.

Nội dung của giáo trình được chia thành 7 chương. Các vấn đề trong mỗi chương được trình bày ngắn gọn từ cơ sở lý thuyết đến xây dựng thuật toán và cuối cùng là mã nguồn cài đặt được minh hoạ bằng ngôn ngữ C++. Để thuận tiện cho việc thực hành của sinh viên, tất cả các mã nguồn trong giáo trình đều tương thích với trình biên dịch Dev C++.

Chúng tôi chân thành cảm ơn các đồng nghiệp ở Khoa Công nghệ Thông tin của các trường Đại học Sư phạm Đà Nẵng, Đại học Bách khoa Đà Nẵng, Đại học Khoa học Huế, Đại học Điện lực Hà Nội đã giúp đỡ, đóng góp nhiều ý kiến quý báu để hoàn thiện nội dung giáo trình này.

Chúng tôi cũng hy vọng sớm nhận được các ý kiến đóng góp, phê bình của bạn đọc về nội dung, chất lượng và hình thức trình bày để giáo trình này ngày một hoàn thiện hơn.

Đà Nẵng, Tháng 08 Năm 2020

TÁC GIẢ

Phạm Anh Phương

MỤC LỤC

CHƯƠNG MỞ ĐẦU: TỔNG QUAN VỀ ĐỒ HỌA MÁY TÍNH 5
1. GIỚI THIỆU VỀ ĐỒ HỌA MÁY TÍNH 5
2. CÁC KỸ THUẬT ĐỒ HỌA 5
2.1. Kỹ thuật đồ họa điểm 5
2.2. Kỹ thuật đồ họa vector 5
3. CÁC ỨNG DỤNG CỦA ĐỒ HỌA 5
4. CÁC LĨNH VỰC NGHIÊN CỨU ĐỒ HỌA 6
5. TỔNG QUAN VỀ MỘT HỆ ĐỒ HỌA 7
5.1. Hệ thống đồ họa 7
5.2. Các thành phần của một hệ thống đồ họa 7
CHƯƠNG 1: CÁC YẾU TỐ CƠ SỞ CỦA ĐỒ HỌA 8
1.1. MÀN HÌNH ĐỒ HỌA 8
1.2. BIỂU DIỄN TOẠ ĐỘ 8
1.3. VẼ ĐIỂM 8
1.4. CÁC THUẬT TOÁN VẼ ĐOẠN THẲNG 9
1.4.1. Thuật toán DDA (Digital differential analyzer) 9
1.4.2. Thuật toán Bresenham 10
1.4.3. Thuật toán MidPoint 13
1.5. THUẬT TOÁN VẼ ĐƯỜNG TRÒN 14
1.5.1. Thuật toán Bresenham 15
1.5.2. Thuật toán MidPoint 17
1.6. THUẬT TOÁN VẼ ELLIPSE 18
1.6.1. Thuật toán Bresenham 18
1.6.2. Thuật toán MidPoint 20
1.7. PHƯƠNG PHÁP VẼ ĐỒ THỊ HÀM SỐ 22
BÀI TẬP 24
CHƯƠNG 2: TÔ MÀU 25
2.1. GIỚI THIỆU CÁC HỆ MÀU 25
2.1.1.Hệ RGB (Red, Green, Blue) 25
2.1.2. Hệ CMY (Cyan, Magenta, Yellow) 26

2.1.3. Hệ YIQ 26
2.1.4. Hệ HSV (Hue, Saturation, Value) 26
2.1.5. Hệ HSL (Hue, Saturation, Lightness) 27
2.2. CÁC THUẬT TOÁN TÔ MÀU 27
2.2.1. Bài toán 27
2.2.2. Thuật toán xác định P ∈ S? 27
2.2.3. Thuật toán Scanline 31
2.2.4. Thuật toán tô loang 34
BÀI TẬP 38
CHƯƠNG 3 : XÉN HÌNH 39
3.1. BÀI TOÁN TỔNG QUÁT 39
3.2. XÉN ĐOẠN THẲNG VÀO HÌNH CHỮ NHẬT 39
3.2.1. Thuật toán Cohen - Sutherland 40
3.2.2. Thuật toán chia nhị phân 43
3.2.3. Thuật toán Liang - Barsky 46
3.2.4. Khi cạnh của hình chữ nhật tạo với trục hoành một góc $\alpha \in (0,\pi/2)$ 49
3.5. XÉN ĐA GIÁC VÀO HÌNH CHỮ NHẬT 50
BÀI TẬP 55
CHƯƠNG 4: THIẾT KẾ ĐƯỜNG CONG BEZIER VÀ B-SPLINE 56
4.1. ĐƯỜNG CONG BEZIER VÀ MẶT BEZIER 56
4.1.1. Thuật toán Casteljau 56
4.1.2. Dạng Bernstein của các đường cong Bezier 57
4.1.3. Tạo và vẽ các đường Bezier 58
4.1.4. Các tính chất của đường cong Bezier 60
4.1.5. Đánh giá các đường cong Bezier 61
4.2. ĐƯỜNG CONG SPLINE VÀ B-SPLINE 62
4.2.1. Định nghĩa 62
4.2.2. Các tính chất hữu ích trong việc thiết kế các đường cong B-Spline 65
4.2.3. Thiết kế các mặt Bezier và B-Spline 66
CHƯƠNG 5: CÁC PHÉP BIẾN ĐỔI TRONG MẶT PHẲNG 67
5.1. CƠ SỞ TOÁN HỌC 67
5.2. CÁC PHÉP BIẾN ĐỔI CƠ BẢN 67

5.2.1. Phép tịnh tiến 67
5.2.2. Phép đồng dạng 68
5.2.3. Phép đối xứng 68
5.2.4. Phép quay 68
5.2.5. Phép biến dạng 68
5.2.6. Hợp của các phép biến đổi 68
5.3. CÁC VÍ DỤ MINH HỌA 70
BÀI TẬP 73
CHƯƠNG 6: VẼ CÁC ĐỐI TƯỢNG BA CHIỀU 74
6.1. CÁC PHÉP BIẾN ĐỔI TRONG KHÔNG GIAN 74
6.1.1. Các hệ trục tọa độ 74
6.1.2. Các phép biến đổi cơ bản 75
6.1.3. Ma trận nghịch đảo 76
6.2. PHÉP CHIẾU VẬT THỂ TRONG KHÔNG GIAN LÊN MẶT PHẲNG 76
6.2.1. Phép chiếu phối cảnh 76
6.2.2. Phép chiếu song song 77
6.3. CÔNG THỨC CỦA CÁC PHÉP CHIẾU LÊN MÀN HÌNH 77
6.4. PHỤ LỤC 82
6.5. MÔ HÌNH WIREFRAME 86
6.5.1. Xây dựng cấu trúc dữ liệu 88
6.5.2. Vẽ mô hình WireFrame 89
6.6. VẼ CÁC MẶT TOÁN HỌC 89
BÀI TẬP 93
CHƯƠNG 7: KHỬ ĐƯỜNG VÀ MẶT KHUẤT 94
7.1. MÔ HÌNH CÁC MẶT ĐA GIÁC 94
7.2. CÁC PHƯƠNG PHÁP KHỬ MẶT KHUẤT 95
7.2.1. Giải thuật Depth-sorting 95
7.2.2. Giải thuật BackFace 97
7.2.3. Giải thuật vùng đệm độ sâu (Z-Buffer) 101
BÀI TẬP 102
TÀI LIỆU THAM KHẢO 103

CHƯƠNG MỞ ĐẦU: TỔNG QUAN VỀ ĐỒ HỌA MÁY TÍNH

1. GIỚI THIỆU VỀ ĐỒ HỌA MÁY TÍNH

Đồ họa máy tính là một **ngành khoa học máy tính** chuyên nghiên cứu về các **phương pháp** và **kỹ thuật** để có thể mô tả và thao tác trên các đối tượng của thế giới thực bằng máy tính.

Về bản chất: đó là một **quá trình xây dựng và phát triển** các công cụ trên cả hai lĩnh vực phần cứng và phần mềm hỗ trợ cho các lập trình viên thiết kế các chương trình có khả năng đồ họa cao.

Với việc **mô tả dữ liệu thông qua các hình ảnh và màu sắc** đa dạng của nó, các chương trình đồ họa thường thu hút người sử dụng bởi tính thân thiện, dễ dùng,... kích thích khả năng sáng tạo và nâng cao năng suất làm việc.

Thuật ngữ **đồ họa máy tính (Computer Graphics)** được đề xuất bởi nhà khoa học người Mỹ tên là William Fetter vào năm 1960 khi ông đang nghiên cứu xây dựng mô hình buồng lái máy bay cho hãng Boeing.

Các chương trình đồ họa ứng dụng cho phép chúng ta làm việc với máy tính một cách thoải mái, tự nhiên.

2. CÁC KỸ THUẬT ĐỒ HỌA

Dựa vào các phương pháp xử lý dữ liệu trong hệ thống, có thể phân thành hai kỹ thuật đồ họa:

2.1. Kỹ thuật đồ họa điểm

Nguyên lý của kỹ thuật này như sau: các hình ảnh được hiển thị thông qua từng pixel (từng mẫu rời rạc). Với kỹ thuật này, chúng ta có thể tạo ra, xóa hoặc thay đổi thuộc tính của từng pixel của các đối tượng. Các hình ảnh được hiển thị như một lưới điểm rời rạc (grid), từng điểm đều có vị trí xác định được hiển thị với một giá trị nguyên biểu thị màu sắc hoặc độ sáng của điểm đó. Tập hợp tất cả các pixel của grid tạo nên hình ảnh của đối tượng mà ta muốn biểu diễn.

2.2. Kỹ thuật đồ họa vector

Nguyên lý của kỹ thuật này là xây dựng mô hình hình học (geometrical model) cho hình ảnh đối tượng, xác định các thuộc tính của mô hình hình học, sau đó dựa trên mô hình này để thực hiện quá trình tô trát (rendering) để hiển thị từng điểm của mô hình, hình ảnh của đối tượng. Kỹ thuật này chỉ lưu trữ mô hình toán học của các thành phần trong mô hình hình học cùng với các thuộc tính tương ứng mà không cần lưu lại toàn bộ tất cả các pixel của hình ảnh đối tượng.

3. CÁC ỨNG DỤNG CỦA ĐỒ HỌA

Ngày nay, đồ họa máy tính được sử dụng rộng rãi trong nhiều lĩnh vực khác nhau như: Công nghiệp, thương mại, quản lý, giáo dục, giải trí,... Sau đây là một số ứng dụng tiêu biểu:

3.1. Tạo giao diện (User Interfaces): như các chương trình ứng dụng WINDOWS, WIN-WORD, EXCEL ... đang được đa số người sử dụng ưa chuộng nhờ tính thân thiện, dễ sử dụng.

3.2. Tạo ra các biểu đồ dùng trong thương mại, khoa học và kỹ thuật: Các biểu đồ được tạo ra rất đa dạng, phong phú bao gồm cả hai chiều lẫn ba chiều góp phần thúc đẩy xu hướng phát triển các mô hình dữ liệu hỗ trợ đắc lực cho việc phân tích thông tin và trợ giúp ra quyết định.

3.3. Tự động hóa văn phòng và chế bản điện tử: dùng những ứng dụng của đồ họa để in ấn các tài liệu với nhiều loại dữ liệu khác nhau như: văn bản, biểu đồ, đồ thị và nhiều loại hình ảnh khác ...

3.4. Thiết kế với sự trợ giúp của máy tính (Computer aided design): Một trong những lợi ích lớn nhất của máy tính là trợ giúp con người trong việc thiết kế. Các ứng dụng đồ họa cho phép chúng ta thiết kế các thiết bị cơ khí, điện, điện tử, ô tô, máy bay, ... như phần mềm AUTOCAD ...

3.5. Lĩnh vực giải trí, nghệ thuật: Các phần mềm PAINTBRUSH, CORELDRAW, PHOTOSHOP... cho phép tạo ra các hình ảnh trực quan trên màn hình của máy tính, người họa sĩ có thể tự pha màu, trộn màu, thực hiện một số thao tác: cắt, dán, tẩy, xóa, phóng to, thu nhỏ ... Ngoài ra hiện nay còn có hàng triệu game online cũng như offline phục vụ trong việc giải trí.

3.6. Lĩnh vực bản đồ: xây dựng và in ấn các bản đồ địa lý. Một trong những ứng dụng hiện nay của đồ họa là hệ thống thông tin địa lý (GIS - Geographical Information System).

4. CÁC LĨNH VỰC NGHIÊN CỨU ĐỒ HỌA

4.1. Các hệ CAD/CAM (CAD – Computer Aided Design, CAM – Computer Aided Manufacture)

Các hệ này xây dựng tập hợp các công cụ đồ họa trợ giúp cho việc thiết kế các chi tiết và các hệ thống khác nhau: các thiết bị cơ khí, điện tử... Chẳng hạn như phần mềm Auto Cad của hảng AutoDesk...

4.2. Xử lý ảnh (Image Processing)

Đây là lĩnh vực xử lý các dữ liệu ảnh trong cuộc sống. Sau quá trình xử lý ảnh, dữ liệu đầu ra là ảnh của đối tượng. Trong quá trình xử lý ảnh, chúng ta sẽ sử dụng rất nhiều các kỹ thuật phức tạp: khôi phục ảnh, xác định biên...

Ví dụ: phần mềm PhotoShop, Corel Draw, ...

4.3. Khoa học nhận dạng (Pattern Recognition)

Nhận dạng là một lĩnh vực trong kỹ thuật xử lý ảnh. Từ những mẫu ảnh có sẵn, ta phân loại theo cấu trúc hoặc theo các phương pháp xác định nào đó và bằng các thuật toán chọn lọc để có thể phân tích hay tổng hợp ảnh đã cho thành một tập hợp các ảnh gốc, các ảnh gốc này được lưu trong một thư viện và căn cứ vào thư viện này để nhận dạng các ảnh khác.

Ví dụ: Phần mềm nhận dạng chữ viết (VnDOCR) của viện Công nghệ Thông tin Hà Nội, nhận dạng vân tay, nhận dạng mặt người trong khoa học hình sự...

4.4. Đồ họa minh họa (Presentation Graphics)

Đây là lĩnh vực đồ họa bao gồm các công cụ trợ giúp cho việc hiển thị các số liệu thống kê một cách trực quan thông qua các mẫu đồ thị hoặc biểu đồ có sẵn. Chẳng hạn như các biểu đồ (Chart) trong các phần mềm Word, Excel...

4.5. Hoạt hình và nghệ thuật

Lĩnh vực đồ họa này bao gồm các công cụ giúp cho các họa sĩ, các nhà thiết kế phim ảnh chuyên nghiệp thực hiện các công việc của mình thông qua các kỹ xảo vẽ tranh, hoạt hình hoặc các kỹ xảo điện ảnh khác...

Ví dụ: Phần mềm xử lý các kỹ xảo hoạt hình như 3D Animation, 3D Studio Max..., phần mềm xử lý các kỹ xảo điện ảnh: Adobe Primiere, Cool 3D,...

5. TỔNG QUAN VỀ MỘT HỆ ĐỒ HỌA

5.1. Hệ thống đồ họa

Phần mềm đồ họa: Là tập hợp các câu lệnh đồ họa của hệ thống. Các câu lệnh lập trình dùng cho các thao tác đồ họa không được các ngôn ngữ lập trình thông dụng như PASCAL, C, ... hỗ trợ. Thông thường, nó chỉ cung cấp như là một tập công cụ thêm vào trong ngôn ngữ. Tập các công cụ này dùng để tạo ra các thành phần cơ sở của một hình ảnh đồ họa như: Điểm, đoạn thẳng, đường tròn, màu sắc,... Qua đó, các nhà lập trình phải tạo ra các chương trình đồ họa có khả năng ứng dụng cao hơn.

Phần cứng đồ họa: Là các thiết bị điện tử: CPU, Card, màn hình, chuột, phím... giúp cho việc thực hiện và phát triển các phần mềm đồ họa.

5.2. Các thành phần của một hệ thống đồ họa

Tập hợp các công cụ này được phân loại dựa trên những công việc trong từng hoàn cảnh cụ thể: xuất, nhập, biến đổi ảnh, ... bao gồm:

- **Tập công cụ tạo ra ảnh gốc** (output primitives): cung cấp các công cụ cơ bản nhất cho việc xây dựng các hình ảnh. Các ảnh gốc bao gồm các chuỗi ký tự, các thực thể hình học như điểm, đường thẳng, đa giác, đường tròn,...
- **Tập các công cụ thay đổi thuộc tính** (attributes): dùng để thay đổi thuộc tính của các ảnh gốc. Các thuộc tính của ảnh gốc bao gồm màu sắc (color), kiểu đường thẳng (line style), kiểu văn bản (text style), mẫu tô vùng (area filling pattern),...
- **Tập các công cụ thay đổi hệ quan sát** (viewing transformation): Một khi mà các ảnh gốc và các thuộc tính của nó được xác định trong hệ tọa độ thực, ta cần phải chiếu phần quan sát của ảnh sang một thiết bị xuất cụ thể. Các công cụ này cho phép định nghĩa các vùng quan sát trên hệ tọa độ thực để hiển thị hình ảnh đó.
- **Tập các công cụ phục vụ cho các thao tác nhập dữ liệu** (input operations): Các ứng dụng đồ họa có thể sử dụng nhiều loại thiết bị nhập khác nhau như bút vẽ, bảng, chuột, ... Chính vì vậy, cần xây dựng thêm các công cụ này để điều khiển và xử lý các dữ liệu nhập sao cho có hiệu quả.

Một yêu cầu về phần cứng không thể thiếu đặt ra cho các phần mềm đồ họa là: tính dễ mang chuyển (portability), có nghĩa là chương trình có thể chuyển đổi một cách dễ dàng giữa các kiểu phần cứng khác nhau. Nếu không có sự chuẩn hóa, các chương trình thiết kế thường không thể chuyển đổi đến các hệ thống phần cứng khác mà không viết lại gần như toàn bộ chương trình.

Sau những nổ lực của các tổ chức chuẩn hóa quốc tế, một chuẩn cho việc phát triển các phần mềm đồ họa đã ra đời: đó là **GKS** (**Graphics Kernel System** - Hệ đồ họa cơ sở). Hệ thống này ban đầu được thiết kế như là một tập các công cụ đồ họa hai chiều, sau đó được phát triển để mở rộng trong đồ họa ba chiều.

Ngoài ra, còn có một số chuẩn đồ họa phổ biến như:

- **CGI** (Computer Graphics Interface System): hệ chuẩn cho các phương pháp giao tiếp với các thiết bị ngoại vi.
- **OPENGL**: thư viện đồ họa của hảng Silicon Graphics.
- **DIRECTX**: thư viện đồ họa của hảng Microsoft.

CHƯƠNG 1: CÁC YẾU TỐ CƠ SỞ CỦA ĐỒ HỌA

1.1. MÀN HÌNH ĐỒ HỌA

Mỗi máy tính đều có một CARD dùng để quản lý màn hình, gọi là Video Adapter hay Graphics Adapter. Có nhiều loại adapter như: CGA, MCGA, EGA, VGA, Hercules... Các adapter có thể làm việc ở hai chế độ: văn bản (Text Mode) và đồ họa (Graphics Mode).

Có nhiều cách để khởi tạo các mode đồ họa. Ta có thể sử dụng hàm $00 ngắt $10 của BIOS với các Mode sau:

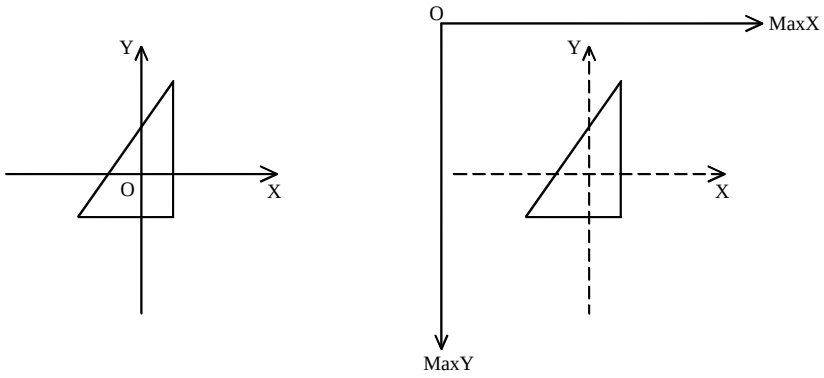
- Mode 12h: chế độ phân giải 640x480x16
- Mode 13h: chế độ phân giải 320x200x256

Chúng ta có thể viết một hàm để khởi tạo các mode đồ họa như sau:

```
void InitGraph(int Mode)
{
 union REGS intregs, outregs;
 intregs.h.ah = 0;
 intregs.h.al = (unsigned char)Mode;
 _int86(0x10,&intregs,&outregs);
}
```

Chú ý: ***Các bạn có thể tham khảo thêm về cách sử dụng các véc tơ ngắt của ROM BIOS ở các tài liệu về lập trình hệ thống.***

1.2. BIỂU DIỄN TOẠ ĐỘ



Tọa độ thế giới thực Tọa độ thiết bị màn hình.

Hình 1.1

Hầu hết các chương trình đồ họa đều dùng hệ tọa độ Decartes (Hình 1.1). Tuy nhiên tùy theo từng ứng dụng cụ thể, có thể sử dụng hệ tọa độ cầu để thuận lợi cho việc tính toán sau đó chuyển qua hệ toạ độ Decartes để vẽ lên màn hình.

1.3. VẼ ĐIỂM

Trong các hệ thống đồ họa, một điểm (**pixel**) được biểu thị bởi các tọa độ bằng số.

Ví dụ: Trong mặt phẳng, một điểm là một cặp (x,y).

Trong không gian ba chiều, một điểm là bộ ba (x,y,z).

Trên màn hình của máy tính, một điểm là một vị trí trong vùng nhớ màn hình dùng để lưu trữ các thông tin về độ sáng của điểm tương ứng trên màn hình.

Số điểm vẽ trên màn hình được gọi là ***độ phân giải*** của màn hình (320x200, 480x640, 1024x1024,...)

Cách hiển thị thông tin lên màn hình đồ họa:

Vùng đệm màn hình hay còn gọi là bộ nhớ hiển thị được bắt đầu từ địa chỉ **A000h:$0000h**. Vì vậy, để hiển thị thông tin ra màn hình thì ta chỉ cần đưa thông tin vào vùng đệm màn hình bắt đầu từ địa chỉ trên là được.

Có nhiều cách để vẽ một điểm ra màn hình: có thể dùng các phục vụ của BIOS hoặc cũng có thể truy xuất trực tiếp vào vùng nhớ màn hình.

- Nếu dùng phục vụ của BIOS, dùng hàm 0Ch ngắt 10h.
- Nếu muốn truy xuất trực tiếp vào vùng đệm màn hình: Giả sử một điểm (x,y) được vẽ trên màn hình với độ phân giải 320x200x256 (mode 13h), điểm đó sẽ được định vị trong vùng đệm màn hình bắt đầu từ địa chỉ segment là A000h và địa chỉ offset được tính theo công thức:

$$\textbf{Offset} = y*320 + x = (y<<8)+(y<<6)+ x$$

Ta có thể viết một hàm để vẽ điểm (x,y) với màu là color như sau:

```
void putPixel(int x,int y,int Color)
{
 unsigned char far *video_buffer;
 video_buffer=(unsigned char far *)0xA0000000L;
 int offset = (y<<8)+(y<<6)+ x;
 video_buffer[offset]=(unsigned char)Color;
}
```

1.4. CÁC THUẬT TOÁN VẼ ĐOẠN THẲNG

Trong các hệ thống đồ họa, các đoạn thẳng được biểu thị bởi việc "tô" đoạn thẳng bắt đầu từ điểm đầu mút này kéo dài cho đến khi gặp điểm đầu mút kia.

Bài toán: Vẽ đoạn thẳng đi qua 2 điểm A(x1,y1) và B(x2,y2).

* Trường hợp x1=x2 hoặc y1=y2: rất đơn giản.

* Trường hợp đường thẳng có hệ số góc m:

Ý tưởng:

Vì các Pixel được vẽ ở các vị trí nguyên nên đường thẳng được vẽ giống như hình bậc thang (do làm tròn).

Vấn đề đặt ra là chọn các tọa độ nguyên gần với đường thẳng nhất.

1.4.1. Thuật toán DDA (Digital differential analyzer)

Xét đường thẳng có hệ số góc $0 < m \le 1$ (giả sử điểm đầu A nằm bên trái và điểm cuối B nằm bên phải). Nếu ta chọn $\Delta x=1$ và tính giá trị y kế tiếp như sau:

$$y_{k+1} = y_k + \Delta y = y_k + m.\Delta x = y_k + m$$

Với hệ số góc m>1: hoán đổi vai trò của x,y cho nhau. Nếu chọn $\Delta y=1$ thì:

$$x_{k+1} = x_k + 1/m$$

Tương tự, nếu điểm B nằm bên trái và A nằm bên phải thì:

$y_{k+1} = y_k - m$ $(0<m\leq 1, \Delta x= -1)$

$x_{k+1} = x_k - 1/m$ $(m>1, \Delta y= -1)$

Tóm lại: Ta có thuật toán vẽ đường thẳng DDA như sau:

- Nhập A(x1,y1) B(x2,y2)
- Tính Δx = x2 - x1, Δy = y2 - y1 và Step = Max(|Δx| , |Δy|)
- Khởi tạo các giá trị:

  IncX = Δx/Step; IncY = Δy/Step; {bước tăng khi vẽ}
  x = x1; y = y1; {Chọn điểm vẽ đầu tiên}
  Vẽ điểm (x,y);
- Cho i chạy từ 1 đến Step:
  - x = x + IncX; y = y + IncY;
  - Vẽ điểm (Round(x),Round(y))

Từ đó ta có hàm vẽ đoạn thẳng theo thuật toán DDA như sau:

```
int round(float x)
{
  if(x>0) return int(x+0.5);
  else return int(x-0.5);
}
void DDALine(int x1,int y1,int x2,int y2)
{
 int step;
 float dx,dy,xInc,yInc,x,y;
 dx=x2-x1; dy=y2-y1;
 if (abs(dx)>abs(dy)) step=abs(dx);
 else step=abs(dy);
 xInc=dx/step;
 yInc=dy/step;
 x=x1; y=y1;
 putpixel(round(x),round(y),15);
 for(int i=1;i<=step;i++)
   putpixel(round(x+xInc),round(y+yInc),color);
}
```

1.4.2. Thuật toán Bresenham

Phương trình đường thẳng có thể phát biểu dưới dạng:

$$y = m.x + b \qquad (1)$$

Phương trình đường thẳng qua 2 điểm $A(x_1,y_1)$, $B(x_2,y_2)$:

$$\frac{x - x_1}{x_2 - x_1} = \frac{y - y_1}{y_2 - y_1} \qquad (*)$$

Đặt Δx = x2 - x1, Δy = y2 - y1

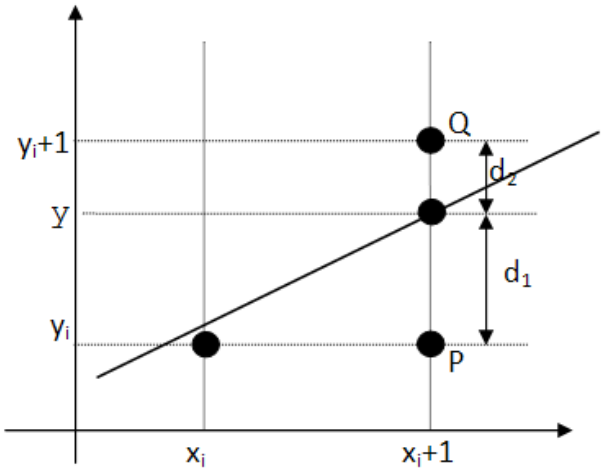
$$(*) \Leftrightarrow y = x.\frac{\Delta y}{\Delta x} + y1 - x1.\frac{\Delta y}{\Delta x}$$

Suy ra

$$m = \frac{\Delta y}{\Delta x} \quad \text{nên } \Delta y = m.\Delta x \text{ và } \quad b = y_1 - m.x_1$$

Trong thuật toán này, ta chỉ xét trường hợp đường thẳng có hệ số góc 0<m<1.

Giả sử ở bước thứ i đã vẽ được điểm (x_i,y_i), cần chọn điểm kế tiếp là P hoặc Q (Hình 1.2).



Hình 1.2

Đặt:

$$d_1 = y - y_i = m.(x_i + 1) + b - y_i$$
$$d_2 = (y_i + 1) - y = y_i + 1 - m.(x_i + 1) - b$$

Suy ra:

$$d_1 - d_2 = 2m.(x_i + 1) - 2y_i + 2b - 1$$
$$= 2.\frac{\Delta y}{\Delta x}.(x_i + 1) - 2y_i + 2b - 1$$
$$\Leftrightarrow \Delta x(d_1 - d_2) = 2\Delta y.x_i - 2\Delta x.y_i + 2\Delta y + \Delta x.(2b - 1)$$

Đặt

$$p_i = \Delta x(d_1 - d_2) \text{ và } C = 2\Delta y + \Delta x.(2b - 1)$$
$$\Rightarrow p_i = 2\Delta y.x_i - 2\Delta x.y_i + C \quad (2)$$

Tương tự, tại bước thứ i+1 ta có:

$$p_{i+1} = 2\Delta y.x_{i+1} - 2\Delta x.y_{i+1} + C \quad (3)$$

Lấy (3) – (2), ta có:

$$p_{i+1} - p_i = 2\Delta y(x_{i+1} - x_i) - 2\Delta x(y_i - y_{i+1})$$
$$= 2\Delta y - 2\Delta x(y_{i+1} - y_i) \quad (\text{vì } x_{i+1} - x_i = 1)$$

Suy ra:

$$\boxed{\mathbf{p_{i+1} = p_i + 2\Delta y - 2\Delta x(y_{i+1} - y_i)}} \quad (4)$$

* ***Nhận xét***:

- Nếu $p_i < 0$ ($d_1<d_2$): Chọn P hay chọn $\mathbf{y_{i+1} = y_i}$,

từ (4) $\Rightarrow$ $\boxed{\mathbf{p_{i+1} = p_i + 2\Delta y}}$.

- Nếu $p_i \geq 0$ ($d_1 \geq d_2$): Chọn Q hay chọn $\mathbf{y_{i+1} = y_i + 1}$,

từ (4) $\Rightarrow$ $\boxed{\mathbf{p_{i+1} = p_i + 2\Delta y - 2\Delta x}}$.

- Với điểm mút đầu tiên, thay (x_1,y_1) vào (2) ta có:

$$\boxed{\mathbf{p_1} = 2\Delta y.x_1 - 2\Delta x.y_1 + 2\Delta y + \Delta x[2.(y_1 - m.x_1) - 1] = \mathbf{2\Delta y - \Delta x}}$$

Thuật toán Bresenham vẽ đoạn thẳng trường cho trường hợp hệ số góc $0<m<1$ có thể mô tả tóm tắt như sau:

- **Bước 1**: Nhập các điểm đầu mút. Điểm đầu mút bên trái chứa tọa độ (x_1,y_1), điểm đầu mút bên phải chứa tọa độ (x_2,y_2).
- **Bước 2**:
  - Tính $\Delta x = |x_2 - x_1|$, $\Delta y = |y_2 - y_1|$ và $p = 2\Delta y - \Delta x$.
  - Chọn điểm xuất phát $(x,y) = (x_1,y_1)$.
  - Vẽ điểm (x,y).
- **Bước 3: Nếu $x<x_2$ thì $x = x+1$.**
  - Nếu $p < 0$: $p = p + 2\Delta y$

    Ngược lại: $p = p + 2(\Delta y - \Delta x)$ và $\mathbf{y = y+1}$.
  - Vẽ điểm (x,y) mới.
- **Bước 4: Lặp lại bước 3 cho đến khi $x = x_2$.**

Sau đây là hàm cài đặt thuật toán:

```
void Line(int x1,int y1,int x2,int y2)
// 0<m<1
{
 int x_max,x,y;
 int dx=abs(x1-x2);
 int dy=abs(y1-y2);
 int c1=2*dy;
 int c2=2*(dy-dx);
 int p=2*dy-dx;
 if (x1>x2)
 {
   x=x2; y=y2; x_max=x1;
 }
 else
  {
            x=x1; y=y1; x_max=x2;
  }
 putpixel(x,y,color);
 while (x < x_max)
```

```
    {
     x=x+1;
     if (p<0) p=p+c1;
     else
           {
           y=y+1;
           p=p+c2;
           }
      putpixel(x,y,5);
    }
  }
```

1.4.3. Thuật toán MidPoint

Ta chỉ xét trường hợp đường thẳng có hệ số góc 0<m<1.

Thuật toán này đưa ra cách chọn điểm $P(x_i+1,y_i)$ hay $Q(x_i+1,y_i+1)$ bằng cách xét vị trí tương đối của điểm M (trung điểm của PQ) so với đường thẳng.

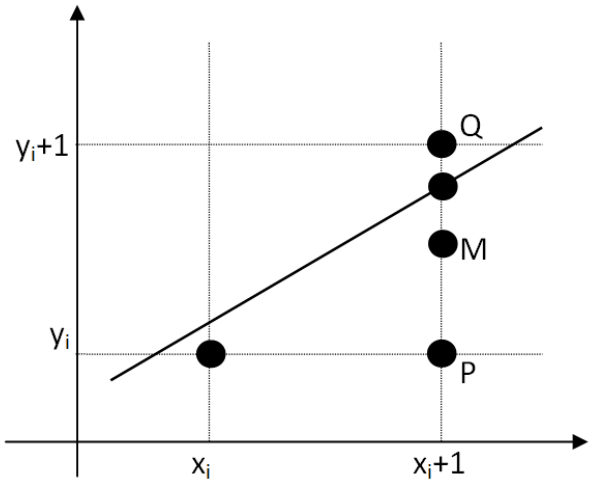
- Nếu điểm M nằm phía dưới đường thẳng ⇔ đường thẳng gần điểm Q hơn ⇒ chọn điểm Q.
- Ngược lại, chọn điểm P.

Phương trình tổng quát của đường thẳng có dạng: $Ax + By + C = 0$, với $A = y_2 - y_1$, $B = -(x_2 - x_1)$ và $C = x_2.y_1 - x_1.y_2$

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

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

Giả sử tại bước thứ i đã vẽ được điểm (x_i,y_i), cần chọn điểm kế tiếp là P hoặc Q (Hình 1.3).



Hình 1.3

Đặt $p_i = F(M) = F(x_i + 1, y_i + \frac{1}{2}) = A(x_i+1) - B(y_i + \frac{1}{2}) + C$ (1)

- Nếu $p_i < 0 \Rightarrow$ M nằm phía trên đường thẳng $\Rightarrow$ đường thẳng gần điểm P hơn $\Rightarrow$ Chọn P.
- Nếu $p_i \geq 0 \Rightarrow$ M nằm phía dưới đường thẳng $\Rightarrow$ Chọn Q.

Tương tự, tại bước thứ i+1, ta có:

$$p_{i+1} = F(x_{i+1} + 1, y_{i+1} + \frac{1}{2}) = A(x_{i+1}+1) + B(y_{i+1} + \frac{1}{2}) + C \qquad (2)$$

Lấy (2) – (1), ta có:

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

$$= A(x_{i+1} - x_i) + B(y_{i+1} - y_i)$$

$$= A + B(y_{i+1} - y_i) \text{ (vì } x_{i+1} - x_i = 1)$$

Suy ra:

$$\boxed{\mathbf{p_{i+1} = p_i + A + B(y_{i+1} - y_i)}} \qquad \mathbf{(3)}$$

Nhận xét:

- Nếu $p_i < 0$: Chọn điểm P hay chọn $\mathbf{y_{i+1} = y_i}$.

Từ (3) suy ra $\boxed{\mathbf{p_{i+1} = p_i + A}}$

- Nếu $p_i \geq 0$: Chọn điểm Q hay chọn $y_{i+1} = y_i + 1$.

Từ (3) suy ra $\boxed{\mathbf{p_{i+1} = p_i + A + B}}$

- Với điểm mút đầu tiên, thay (x_1,y_1) vào (1) ta có:

$$p_1 = F(x_1 + 1, y_1 + \frac{1}{2}) = A(x_1+1) + B(y_1 + \frac{1}{2}) + C$$

$$= Ax_1 + Bx_1 + C + A + \frac{B}{2} = A + \frac{B}{2} \text{ (vì } Ax_1 + Bx_1 + C = 0)$$

$$\boxed{\mathbf{p_1 = A + \frac{B}{2}}}$$

Thuật toán MidPoint có độ phức tạp tính toán tương đương với thuật toán Bresenham.

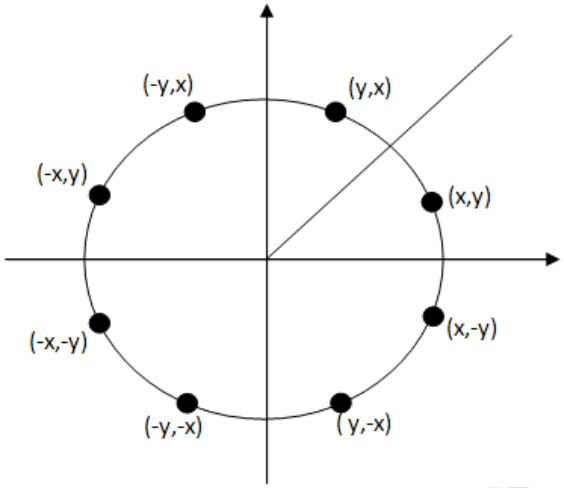
1.5. THUẬT TOÁN VẼ ĐƯỜNG TRÒN

Xét đường tròn (C) tâm $O(x_c,y_c)$ bán kính R. Phương trình tổng quát của đường tròn có dạng:

$$(x - x_c)^2 + (y - y_c)^2 = R^2 \qquad (*)$$

$$\Leftrightarrow y = y_c \pm \sqrt{R^2 - (x - x_C)^2} \qquad (1)$$

Để đơn giản thuật toán, đầu tiên ta xét đường tròn có tâm ở gốc tọa độ ($x_c=0$ và $y_c=0$).



Hình 1.4

*** *Ý tưởng*:**

Do tính đối xứng của đường tròn nên nếu điểm (x,y)∈(C) thì các điểm (y,x), (-y,x), (-x,y), (-x,-y), (-y,-x), (y,-x), (x,-y) cũng ∈ (C).

Vì vậy, chỉ cần vẽ một phần tám cung tròn rồi lấy đối xứng qua 2 trục toạ độ và đường phân giác góc phần tư thứ nhất thì sẽ có được toàn bộ đường tròn (Hình 14).

1.5.1. Thuật toán Bresenham

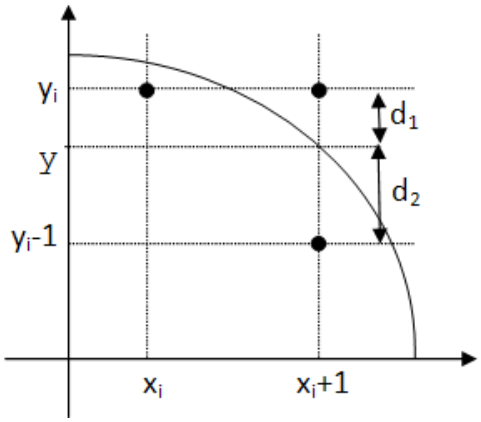
Giả sử đã vẽ được điểm (x_i,y_i), điểm kế tiếp cần vẽ là (x_i+1,y_i) hoặc (x_i+1,y_i-1) (Hình 1.5).

Từ phương trình: $x^2 + y^2 = R^2$, giá trị y thực ứng với $x_i + 1$ sẽ là:

$$y^2 = R^2 - (x_i + 1)^2$$

Đặt: $$d_1 = y_i^2 - y^2 = y_i^2 - R^2 + (x_i + 1)^2$$

$$d_2 = y^2 - (y_i - 1)^2 = R^2 - (x_i + 1)^2 - (y_i - 1)^2$$



Hình 1.5

Suy ra:

$$p_i = d_1 - d_2 = 2.(x_i + 1)^2 + y_i^2 + (y_i - 1)^2 - 2R^2 \quad (2)$$

$$\Rightarrow p_{i+1} = 2.(x_{i+1} + 1)^2 + y^2_{i+1} + (y_{i+1} - 1)^2 - 2R^2 \quad (3)$$

Lấy (3) - (2), ta có:

$$p_{i+1} - p_i = 4x_i + 6 + 2.(y^2_{i+1} - y_i^2) - 2.(y_{i+1} - y_i)$$

$$\Rightarrow \boxed{p_{i+1} = p_i + 4x_i + 6 + 2.(y^2_{i+1} - y_i^2) - 2.(y_{i+1} - y_i)} \quad (4)$$

**** Nhận xét***:

- Nếu $p_i < 0$: chọn $y_{i+1} = y_i$ (4) $\Rightarrow p_{i+1} = p_i + 4x_i + 6$
- Nếu $p_i \geq 0$: chọn $y_{i+1} = y_i - 1$ (4) $\Rightarrow p_{i+1} = p_i + 4.(x_i - y_i) + 10$
- Ta chọn điểm đầu tiên cần vẽ là (0,R), theo (2): $p_1 = 3 - 2R$.

Thuật toán Bresenham để vẽ đường tròn được phát thảo như sau:

- **Bước 1**:
  - Chọn điểm đầu cần vẽ (x,y) = (0,R).
  - Tính p đầu tiên: p = 3 - 2R.
  - Vẽ 8 điểm ứng với (x,y).
- **Bước 2**:
  - Tăng x lên 1 pixel: x = x + 1.
  - Nếu p < 0: p = p + 4x + 6.

  Ngược lại: p = p + 4(x - y) + 10 và y = y – 1.
  - Vẽ điểm 8 điểm ứng với (x,y) mới.
- **Bước 3**: Lặp lại bước 2 cho đến khi x = y.

Sau đây là hàm cài đặt thuật toán:

```
void ve8diem(int x0,int y0,int x, int y, int color)
{
  putpixel( x0 + x , y0 + y ,color);
  putpixel( x0 - x , y0 + y ,color);
  putpixel( x0 + x , y0 - y ,color);
  putpixel( x0 - x , y0 - y ,color);
  putpixel( x0 + y , y0 + x ,color);
  putpixel( x0 - y , y0 + x ,color);
  putpixel( x0 + y , y0 - x ,color);
  putpixel( x0 - y , y0 - x ,color);
}
void circle(int x0,int y0,int r)
{
  int x=0;int y=r;
  int p=3-2*r;
  while (x<=y)
  {
   ve8diem(x0,y0,x,y,15);
   if(p<0) p=p+4*x+6;
   else
       {
            p=p+4*(x-y)+10;
```

```
                y=y-1;
            }
        x=x+1;
       }
    }
```

1.5.2. Thuật toán MidPoint

Từ phương trình đường tròn: $x^2 + y^2 = R^2$

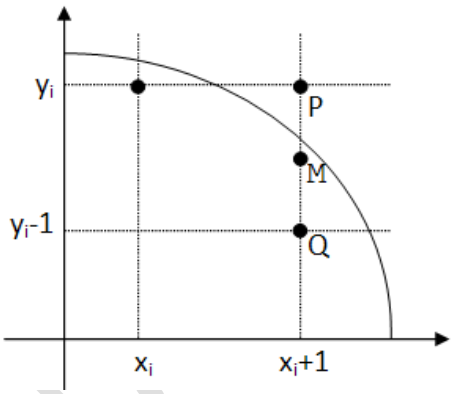
Đặt $F(x,y) = x^2 + y^2 - R^2$,ta có nhận xét:

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

Giả sử đã vẽ được điểm (x_i,y_i), điểm kế tiếp cần vẽ là P hoặc Q (Hình 1.6).

Gọi M là trung điểm của PQ $\Rightarrow$ $M(x_i + 1, y_i - \frac{1}{2})$. Lúc này, việc chọn các điểm P hay Q được đưa về việc xét dấu của:

$$p_i = F(M) = F(x_i + 1, y_i - \frac{1}{2})$$



Hình 1.6

- Nếu $p_i < 0$: M ở trong đường tròn $\Rightarrow$ đường tròn gần P hơn $\Rightarrow$ Chọn P.
- Ngược lại: Chọn Q.

Tương tự:

$$p_{i+1} = F(x_{i+1} + 1, y_{i+1} - \frac{1}{2})$$

Suy ra:

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

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

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

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

hay

$$\boxed{p_{i+1} = p_i + 2x_i + 3 + (y_{i+1}^2 - y_i^2) - (y_{i+1} - y_i)} \quad (*)$$

Nhận xét:

- Nếu $p_i < 0$: Chọn điểm P hay chọn $y_{i+1} = y_i$.
  Từ (*) $\Rightarrow$ $p_{i+1} = p_i + 2x_i + 3$
- Nếu $p_i \geq 0$: Chọn điểm Q hay chọn $y_{i+1} = y_i - 1$.
  Từ (*) $\Rightarrow$ $p_{i+1} = p_i + 2(x_i - y_i) + 5$
- Với điểm đầu tiên (0,R), ta có:

$$p_1 = F(x_1 + 1, y_1 - \frac{1}{2}) = F(1, R - \frac{1}{2}) = 1 + (R - \frac{1}{2})^2 - R^2$$

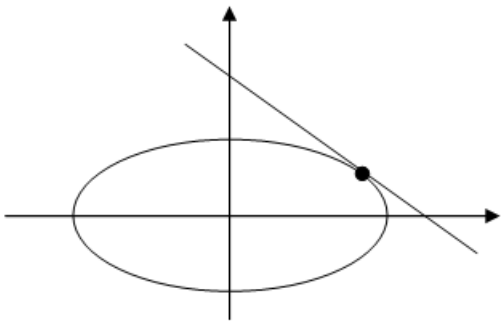
$$p_1 = \frac{5}{4} - R \approx 1 - R$$

1.6. THUẬT TOÁN VẼ ELLIPSE

Để đơn giản, ta chọn Ellipse (E) có tâm ở gốc tọa độ. Phương trình của nó có dạng:

$$\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$$

Ta có thể viết lại: $y^2 = -\frac{b^2}{a^2}.x^2 + b^2$ (*)



Hình 1.7

Ý tưởng: Giống như thuật toán vẽ đường tròn, chỉ có sự khác biệt ở đây là phải vẽ 2 nhánh: Một nhánh từ trên xuống và một nhánh từ dưới lên và 2 nhánh này sẽ gặp nhau tại điểm mà ở đó hệ số góc của tiếp tuyến với Ellipse bằng -1.

Phương trình tiếp tuyến với Ellipse tại điểm $(x_0,y_0) \in (E)$:

$$x_0.\frac{x}{a^2} + y_0.\frac{y}{b^2} = 1$$

Suy ra, hệ số góc của tiếp tuyến tại điểm đó là: $-\frac{x_0.b^2}{y_0 a^2}$.

1.6.1. Thuật toán Bresenham

Xét nhánh vẽ từ trên xuống. Giả sử điểm (x_i,y_i) đã được vẽ. Điểm tiếp theo cần chọn sẽ là (x_i+1,y_i) hoặc (x_i+1,y_i-1)

Thay (x_i+1) vào (*): $y^2 = -\frac{b^2}{a^2}.(x_i+1)^2 + b^2$

Đặt:

$$d_1 = yi^2 - y^2 = y_i^2 + \frac{b^2}{a^2}.(x_i + 1)^2 - b^2$$

$$d_2 = y^2 - (y_i - 1)^2 = -\frac{b^2}{a^2}.(x_i + 1)^2 + b^2 - (y_i - 1)^2$$

$$\Rightarrow \quad p_i = d_1 - d_2 = 2.[\frac{b^2}{a^2}.(x_i + 1)^2 - b^2] + 2.(y_i^2 - y_i) + 1$$

$$p_{i+1} = 2.[\frac{b^2}{a^2}.(x_{i+1} + 1)^2 - b^2] + 2.(y_{i+1}^2 - y_{i+1}) + 1$$

Suy ra:

$$p_{i+1} - p_i = 2.\frac{b^2}{a^2}.[(x_{i+1} + 1)^2 - (x_i + 1)^2] + 2(y_{i+1}^2 - y_i^2 + y_i - y_{i+1}) \quad (**)$$

***** ***Nhận xét***:

- $p_i < 0$: Chọn $y_{i+1} = y_i$, (**) $\Rightarrow p_{i+1} = p_i + 2.\frac{b^2}{a^2}.(2x + 3)$
- $p_i \geq 0$: Chọn $y_{i+1} = y_i - 1$, (**) $\Rightarrow p_{i+1} = p_i + 2.\frac{b^2}{a^2}(2x + 3) - 4y_i + 4$
- Với điểm đầu tiên (0,b), ta có: $p_1 = 2\frac{b^2}{a^2} - 2b + 1$

Từ đó, ta có hàm vẽ Ellipse theo thuật toán Bresenham như sau:

```
void ve4diem(int xc, int yc, int x, int y, int color)
{
  putpixel(xc+x,yc+y,color);
  putpixel(xc-x,yc+y,color);
  putpixel(xc+x,yc-y,color);
  putpixel(xc-x,yc-y,color);
}
void ellipse(int xc, int yc, int a, int b)
{
  float p,a2,b2,t;
  int x,y,x0,y0;
  a2 = pow(a,2);
  b2 = pow(b,2);
  t  = sqrt(a2+b2);
  x0 = a2/t;  //hoanh do cua tiep diem
  y0 = b2/t;  //tung do cua tiep diem
  // Ve nhanh thu 1 (tu tren xuong)
  x = 0; y = b;
  p = 2*(b2/a2) - 2*b + 1;
  while ((x<=x0)||(y>=y0))
```

```
    {
    ve4diem(xc,yc,x,y,color);
    if (p<0) p = p + 2*(b2/a2)*(2*x + 3);
    else
     {
       p = p - 4*y +  2*(b2/a2)*(2*x + 3) + 4;
       y = y - 1;
     }
     x = x + 1;
    }
    // Ve nhanh thu 2 (tu duoi len)
    y = 0; x = a;
    p = 2*(a2/b2) - 2*a + 1;
    while ((x>=x0)||(y<=y0))
    {
     ve4diem(xc,yc,x,y,color);
     if (p<0)  p = p + 2*(a2/b2)*(2*y + 3);
     else
       {
            p = p - 4*x +2*(a2/b2)*(2*y+3) + 4;
            x = x - 1;
       }
     y = y + 1;
    }
}
```

1.6.2. Thuật toán MidPoint

Gợi ý:

Phương trình Ellipse: $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$

Nhánh 1:

$p_1 = b^2 - a^2b + \frac{1}{4}.a^2$

If $p_i < 0$ Then $p_{i+1} = p_i + b^2 + 2b^2x_{i+1}$

else $p_{i+1} = p_i + b^2 + 2b^2x_{i+1} - 2a^2y_{i+1}$

Nhánh 2:

$p_1 = b^2(x_i + \frac{1}{2})^2 + a^2(y_i - 1)^2 - a^2b^2$

If $p_i > 0$ Then $p_{i+1} = p_i + a^2 - 2a^2y_{i+1}$

else $p_{i+1} = p_i + a^2 + 2b^2x_{i+1} - 2a^2y_{i+1}$

Từ đó, ta có hàm vẽ Ellipse theo thuật toán MidPoint như sau:

```
void ve4diem(int xc,int yc,int x,int y,int color)
{
   putpixel(xc+x,yc+y,color);
   putpixel(xc-x,yc+y,color);
   putpixel(xc-x,yc-y,color);
   putpixel(xc+x,yc-y,color);
}
void midEllipse (int xc,int yc,int a,int b)
{
  int x,y;
  x=0;
  y=b;
  float p,a2,b2;
  a2 = a*a;
  b2 = b*b;
// Ve nhanh 1
  ve4diem(xc,yc,x,y,color);
  p=b2-a2*b+a2*0.25;
  while ((b2/a2)*(x/y)<1)
  {
    x=x+1;
    if (p<0)p=p+b2+2*b2*x;
    else
    {
       y = y-1;
       p = p+b2+2*b2*x-2*a2*y;
    }
    ve4diem(xc,yc,x,y,color);
  }
// Ve nhanh 2
  p=b2*(x+1/2)*(x+1/2)+a2*(y-1)*(y-1)-a2*b2;
  while (y>0)
  {
       y=y-1;
       if (p>0) p=p+a2-2*a2*y;
       else
        {
            x=x+1;
            p=p+a2+2*b2*x-2*a2*y;
        }
        ve4diem(xc,yc,x,y,color);
```

```
    }
  }
```

1.7. PHƯƠNG PHÁP VẼ ĐỒ THỊ HÀM SỐ

1.7.1. Bài toán: Vẽ đồ thị của hàm số y = f(x) trên đoạn [Min,Max].

Ý tưởng: Cho x chạy từ Min đến Max để lấy các tọa độ (x,f(x)) sau đó làm tròn thành số nguyên rồi nối các điểm đó lại với nhau.

1.7.2. Giải thuật:

Bước 1: Xác định đoạn cần vẽ [Min,Max].

Bước 2:

- Đặt gốc tọa độ lên màn hình (x0,y0).
- Chia tỷ lệ vẽ trên màn hình theo hệ số k.
- Chọn bước tăng dx của mỗi điểm trên đoạn cần vẽ.

Bước 3:

- Chọn điểm đầu cần vẽ: x = Min, tính f(x)
- Đổi qua tọa độ màn hình và làm tròn:

  x1 = x0 + Round(x.k);

  y1 = y0 - Round(y.k);
- Di chuyển đến (x1,y1): MOVETO(x1,y1);

Bước 4:

- Tăng x lên với số gia dx: x = x + dx;
- Đổi qua tọa độ màn hình và làm tròn:

  x2 = x0 + Round(x.k);

  y2 = y0 - Round(y.k);
- Vẽ đến (x2,y2): LINETO(x2,y2);

Bước 5: Lặp lại bước 4 cho đến khi x ≥ Max thì dừng.

Ví dụ: Vẽ đồ thị hàm bậc ba $f(x) = ax^3 + bx^2 + cx + d$ trên đoạn [-10,10].

```
#include<conio.h>
#include<graphics.h>
float a,b,c,d,min,max;
int round(float x)
{
     if (x>0)  return int (x+0.5);
     else          return int (x-0.5);
}
void khoitaodohoa()
{
     int gd=0,gm=0;
```

```
     initgraph(&gd,&gm,"");
}
float f(float x)
{
  return(a*x*x*x+b*x*x+c*x+d);
}
void vedothi(float min,float max)
{
     int x0,y0,x1,y1,x2,y2;
     float x,dx,k;
     x0=getmaxx()/2;
     y0=getmaxy()/2;
     k=(float)getmaxx()/50;
     dx=0.001;
     setcolor(12);
     line(0,y0,2*x0,y0);
     line(x0,0,x0,2*y0);
     x=min;
     setcolor(14);
     x1=x0+round(x*k);
     y1=y0-round(f(x)*k);
     moveto(x1,y1);
     while (x<max)
     {
          x=x+dx;
          x2=x0+round(x*k);
          y2=y0-round(f(x)*k);
          lineto(x2,y2);
     }
}
int main()
{
     khoitaodohoa();
     min=-10;max=10;
     a=1;b=-1;c=-1;d=2;
     vedothi(min,max);
     getch();
}
```

BÀI TẬP

1. Cho hai điểm A(20,10) và B(25,13). Hãy tính các giá trị P_i, x_i, y_i ở mỗi bước khi vẽ đoạn thẳng AB theo thuật toán Bresenham/MidPoint và kết qủa điền vào bảng sau:

i	1	2	3	4	5	6
P_i	?	?	?	?	?	?
x_i	?	?	?	?	?	?
y_i	?	?	?	?	?	?

2. Cài đặt hàm vẽ đoạn thẳng theo thuật toán Bresenham và MidPoint cho các trường hợp hệ số góc m>1, m<-1, -1<m<0.

3. Cài đặt hàm vẽ đường tròn theo thuật toán MidPoint.

4. Viết hàm **Arc(int x0, int y0, int g1, int g2, int R)** để vẽ cung tròn có tâm (x0,y0) bán kính R với góc bắt đầu là g1 và góc kết thúc là g2.

5. Viết hàm **Sector(int x0, int y0,int g1,int g2, int Rx, int Ry)** để vẽ cung Ellipse có tâm (x0,y0) bán kính theo trục X là Rx, bán kính theo trục Y là Ry với góc bắt đầu là g1 và góc kết thúc là g2.

6. Viết hàm **DrawPoly(ToaDo2D P[ ]; int n, int xc, int yc, int R)** để vẽ một đa giác đều có n đỉnh lưu trong mảng P nội tiếp trong đường tròn tâm (xc,yc) bán kính R.

7. Viết hàm **Circle3P(ToaDo2D A, ToaDo2D B, ToaDo2D C);** để vẽ đường tròn đi qua 3 điểm A,B,C.

8. Viết hàm **Arc3P(ToaDo2D A, ToaDo2D B, ToaDo2D C);** để vẽ cung tròn đi qua 3 điểm A,B,C.

9. Viết chương trình vẽ đồ thị các hàm số:

$y = ax^2 + bx + c$, $y = ax^3 + bx^2 + cx + d$, $y = ax^4 + bx^3 + cx^2 + dx + e$

$y = \frac{ax+b}{cx+d}$, $y = \frac{ax^2+bx+c}{dx+e}$.

10. Viết chương trình vẽ các đường cong sau:

$y^2 = 2px$, $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$, $\frac{x^2}{a^2} - \frac{y^2}{b^2} = \pm 1$

Bài tập lớn: Viết phần mềm khảo sát và vẽ đồ thị các hàm số sơ cấp ở bài tập số 9.

CHƯƠNG 2: TÔ MÀU

2.1. GIỚI THIỆU CÁC HỆ MÀU

Giác quan của con người cảm nhận được các vật thể xung quanh thông qua các tia sáng màu tốt hơn nhiều so với 2 màu trắng đen. Vì vậy, việc xây dựng nên các chuẩn màu là một trong những lý thuyết cơ bản của lý thuyết đồ họa.

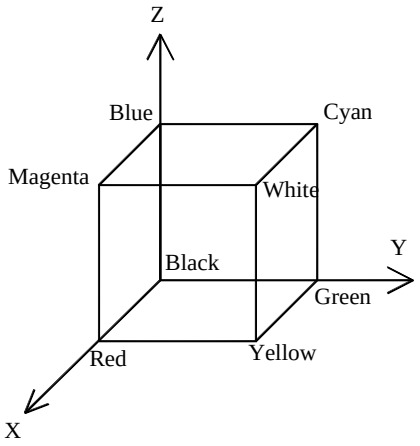
Việc nghiên cứu về màu sắc ngoài các yếu tố về mặt vật lý như bước sóng, cường độ, còn có 3 yếu tố khác liên quan đến cảm nhận sinh lý của mắt người dưới tác động của chùm sáng màu đi đến từ vật thể là: Hue (sắc màu), Saturation (độ bảo hòa), Lightness (độ sáng). Một trong những hệ màu được sử dụng rộng rãi đầu tiên do A.H.Munsell đưa ra vào năm 1976, bao gồm 3 yếu tố: Hue, Lightness và Saturation.

Ba mô hình màu được sử dụng và phát triển nhiều trong các phần cứng là: RGB - dùng với các màn hình CRT (Cathode Ray Bube), YIQ – dùng trong các hệ thống ti vi màu băng tần rộng và CMY - sử dụng trong một số thiết bị in màu.

Ngoài ra, còn có nhiều hệ màu khác như: HSV, HSL, YIQ, HVC, ...

2.1.1.Hệ RGB (Red, Green, Blue)

Mắt của chúng ta cảm nhận ba màu rõ nhất là Red (đỏ), Green (lục), Blue (xanh). Vì vậy, người ta đã xây dựng mô hình màu RGB (Red,Green, Blue) là tập tất cả các màu được xác định thông qua ba màu vừa nêu. Chuẩn này đầu tiên được xây dựng cho các hệ vô tuyến truyền hình và trong các máy vi tính. Tất nhiên, không phải là tất cả các màu đều có thể biểu diễn qua ba màu nói trên nhưng hầu hết các màu đều có thể chuyển về được.



Hình 2.1. Hệ màu RGB

Hệ này được xem như một khối ba chiều với màu Red là trục X, màu Green là trục Y và màu Blue là trục Z. Mỗi màu trong hệ này được xác định theo ba thành phần RGB (Hình 2.1).

Ví dụ:

Màu Red là (1, 0, 0)

Màu Blue là (0, 0, 1)

Red + Green = Yellow

Red + Green + Blue = White

2.1.2. Hệ CMY (Cyan, Magenta, Yellow)

Hệ này cũng được xem như một khối ba chiều như hệ RGB. Nhưng hệ CMY trái ngược với hệ RGB, chẳng hạn:

White có thành phần (0, 0, 0)

Cyan có thành phần (1, 0, 0)

Green có thành phần (1, 0, 1) ...

Sau đây là công thức chuyển đổi từ hệ RGB $\rightarrow$ CMY:

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

2.1.3. Hệ YIQ

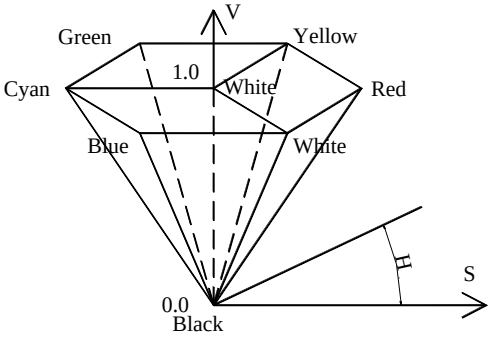
Hệ màu này được ứng dụng trong truyền hình màu băng tần rộng tại Mỹ, do đó nó có mối quan hệ chặt chẽ với màn hình raster. YIQ là sự thay đổi của RGB cho khả năng truyền phát và tính tương thích với ti vi đen trắng thế hệ trước. Tín hiệu truyền sử dụng trong hệ thống NTSC (National Television System Committee).

Sau đây là công thức biến đổi từ hệ RGB thành hệ YIQ:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} * \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Ma trận nghịch đảo của ma trận biến đổi RGB thành hệ YIQ được sử dụng cho phép biến đổi từ hệ YIQ thành RGB.

2.1.4. Hệ HSV (Hue, Saturation, Value)



Hình 2.2. Hệ màu HSV

Mô hình màu này còn được gọi là hệ HSB với B là Brightness (độ sáng) dựa trên cơ sở nền tảng trực giác về tông màu, sắc độ và sắc thái mỹ thuật (Hình 2.2).

Hue có giá trị từ $0^0 \rightarrow 360^0$

S, V có giá trị từ $0 \rightarrow 1$

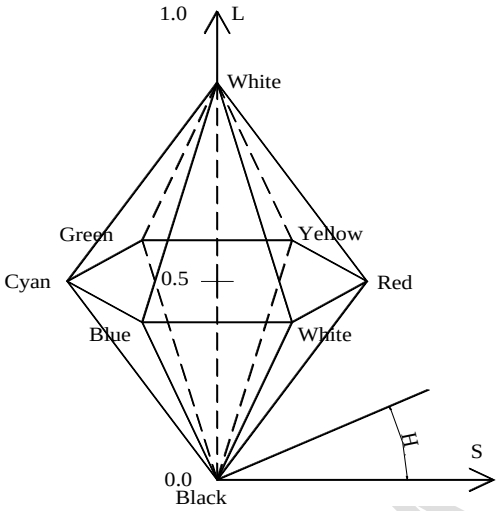
Ví dụ:

Red được biểu diễn (0^0, 1, 1)

Green được biểu diễn (120^0,1,1)

2.1.5. Hệ HSL (Hue, Saturation, Lightness)

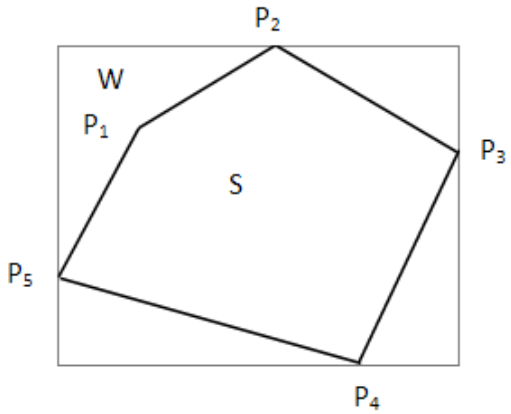
Hệ này được xác định bởi tập hợp hình chóp sáu cạnh đôi của không gian hình trụ (Hình 2.3).



Hình 2.3. Hệ màu HSL

2.2. CÁC THUẬT TOÁN TÔ MÀU

2.2.1. Bài toán



Hình 2.4

Cho đa giác S xác định bởi n đỉnh : $P_1, P_2, ..., P_n$. Hãy tô màu miền S.

* Phương pháp tổng quát :

- Tìm hình chữ nhật W nhỏ nhất chứa S (Hình 2.4).
- Duyệt qua tất cả các điểm P(x, y) ∈ W. Nếu P ∈ S thì tô màu điểm P.

2.2.2. Thuật toán xác định P ∈ S?

2.2.2.1. S là đa giác lồi

- Lấy P ∈ W, nối P với các đỉnh của S thì ta được n tam giác : $S_i = PP_iP_{i+1}$, với $P_{n+1}=P_1$.
- Nếu $\sum_{i=1}^{n} dt(S_i) = dt(S)$ thì P ∈ S.

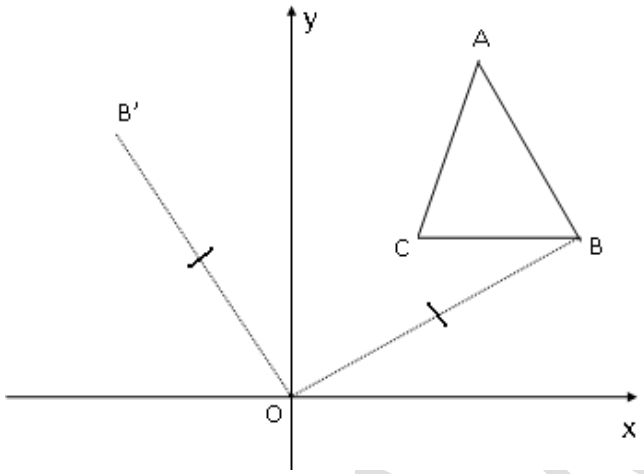
Ta có công thức tính diện tích của S:

$$\boxed{S = \frac{1}{2}\left|\sum_{i=1}^{n}(x_{i+1}y_i - x_i y_{i+1})\right|} \quad (*)$$

Chứng minh: Dùng phương pháp quy nạp.

Đầu tiên ta sẽ chứng minh công thức (*) đúng với n = 3 (tam giác ABC).

Từ O kẻ OB'⊥OB và OB'=OB (Hình 2.5).



Hình 2.5

Theo công thức tính diện tích có hướng:

S_{ABC}=| dt(OAB) + dt(OBC) + dt(OAC) | (1)

Trong đó:

$$dt(OAB) = \frac{1}{2}OA.OB.\sin(AOB)$$

$$= \frac{1}{2}OA.OB.\cos(AOB')$$

$$= \frac{1}{2}OA.OB.\frac{\overrightarrow{OA}.\overrightarrow{OB'}}{\left|\overrightarrow{OA}\right|.\left|\overrightarrow{OB'}\right|}$$

$$= \frac{1}{2}\overrightarrow{OA}.\overrightarrow{OB'} = \frac{1}{2}(x_A x_{B'} + y_A y_{B'})$$

dt(OAB) = $\frac{1}{2}(x_B y_A - x_A y_B)$ (vì $x_{B'}$=-y_B và $y_{B'}$=x_B)

Tương tự, ta cũng tính được:

dt(OBC) = $\frac{1}{2}(x_C y_B - x_B y_C)$

dt(OCA)= $\frac{1}{2}(x_A y_C - x_C y_A)$

Thay vào (1) ta có công thức tính diện tích tam giác ABC:

$$\boxed{S_{ABC} = \frac{1}{2}\left|(x_B y_A - x_A y_B) + (x_C y_B - x_B y_C) + (x_A y_C - x_C y_A)\right|}$$

Giả sử công thức (*) đúng với n = k, ta sẽ chứng minh (*) đúng với n = k+1.

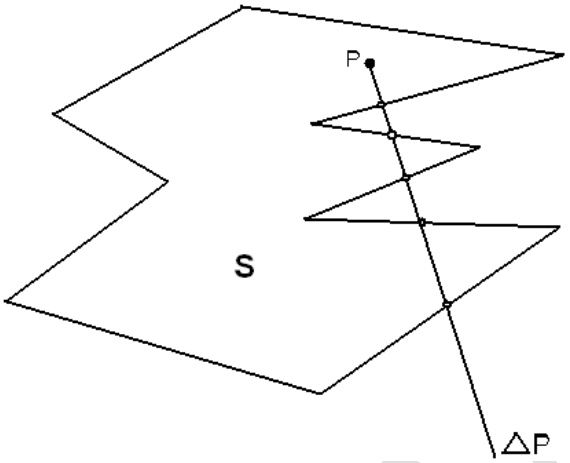
Thật vậy, nếu thêm vào một đỉnh mà vẫn đảm bảo là đa giác lồi thì:

S(đa giác k+1 đỉnh) = S(đa giác k đỉnh) + S(tam giác $P_1P_kP_{k+1}$)

mà **S(đa giác k đỉnh)** đúng (theo giả thiết) và **S(tam giác $P_1P_kP_{k+1}$)** đúng (đã chứng minh ở trường hợp tam giác) ➔ Công thức (*) được chứng minh □.

2.2.2.2. Trường hợp tổng quát (Thuật toán Jordan)

Từ P(x, y) kẻ nửa đường thẳng ΔP không đi qua các đỉnh của đa giác S (Hình 2.6).



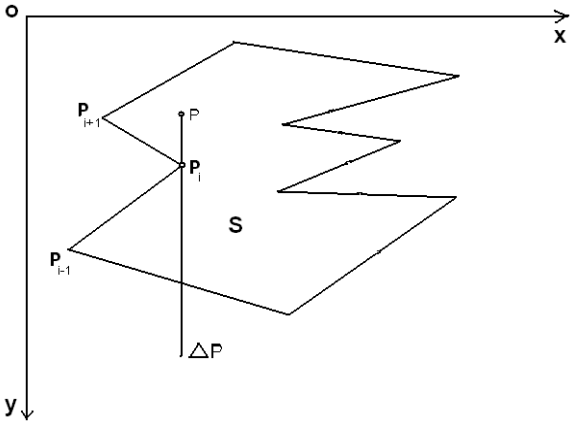
Hình 2.6

Gọi S(P) là số giao điểm của ΔP với các biên của S.

Nếu S(P) lẻ thì P ∈ S.

* Vấn đề còn lại là tìm S(P):

Bước 1: Kẻ nửa đường thẳng ΔP // 0y và hướng về phía y>0 (Hình 2.7).



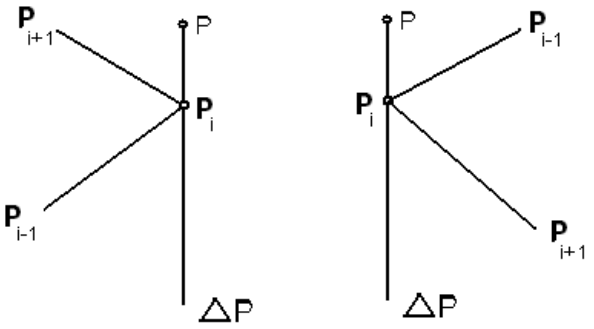
Hình 2.7

Bước 2: Với mỗi cạnh $C_i = P_iP_{i+1}$ của S:

+ Nếu $x=x_i$ (ΔP đi qua 1 đỉnh của đa giác) thì xét 2 cạnh có 1 đầu là P_i:

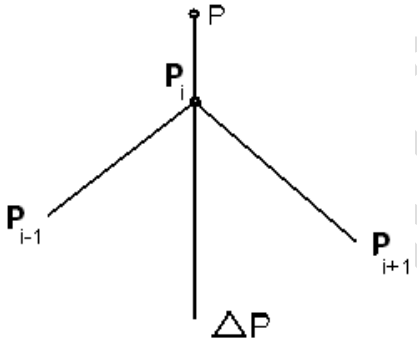
Nếu $y<y_i$ thì

- Nếu cả 2 điểm P_{i+1} và P_{i-1} ở cùng một phía đối với ΔP thì ta tính ΔP cắt cạnh C_i tại 2 điểm (Hình 2.8).



Hình 2.8

- Ngược lại : ΔP cắt C_i tại 1 điểm (Hình 2.9).



Hình 2.9

+ Ngược lại:

- Nếu $x>Max(x_i,x_{i+1})$ hoặc $x<Min(x_i,x_{i+1})$ thì ΔP không cắt C_i
- Ngược lại:
  - Nếu $y<= Min(y_i, y_{i+1})$ thì ΔP cắt C_i
  - Ngược lại: Xét tọa độ giao điểm (x0, y0) của ΔP với C_i

    Nếu $y >= y_0$ thì ΔP không cắt C_i. Ngược lại ΔP cắt C_i.

Thuật toán này có thể được cài đặt bằng đoạn chương trình như sau:

```
struct ToaDo2D
{
  int x, int y;
}
int SOGIAODIEM(ToaDo2D P[]; int n)
{
int dem,i,j,s;
dem:=0;
for(i=1;i<=n;i++)  { Tim so giao diem }
 {
  if(i==n)j=1; else j=i+1;
  if(i==1)s=n; else s=i-1;
  if(x==P[i].x)
   {
     if(y<P[i].y)
```

```
            if((x<=Min(P[s].x ,P[j].x))||(x>=Max(P[s].x,P[j].x)))
                  dem=dem+2
            else dem=dem+1;
         }
         else
          if((x>Min(P[i].x,P[j].x))&&(x<Max(P[j].x,P[i].x)))
            if (y<=Min(P[i].y,P[j].y)) dem=dem+1;
              else
             if(y<=(x-P[j].x)*(P[i].y-P[j].y)/
                                 (P[i].x-P[j].x)+P[j].y) dem=dem+1;
     }
   return dem;
   }
```

2.2.3. Thuật toán Scanline

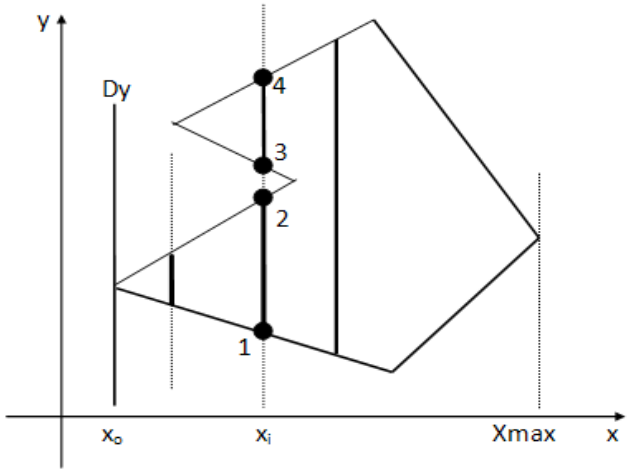
Thuật toán tô màu theo dòng quét (**Scanline**) được mô tả tóm tắt như sau:

Đặt $x_0 = Min(x_i)$, $i \in [1,n]$.

Bước 1: Kẻ Dy//0y đi qua x_0 (Hình 2.10).

Bước 2: Xác định các giao điểm $M_i(x,y)$ của Dy với các cạnh C_i.

Nếu có cạnh $C_i = P_iP_{i+1}$ song song và trùng với Dy thì xem như Dy cắt C_i tại 2 điểm P_i và P_{i+1}.



Hình 2.10

Bước 3: Sắp xếp lại các điểm M_i theo thứ tự tăng dần đối với y_i (điểm đầu tiên có thứ tự là 1).

Bước 4: Những điểm nằm trên Dy ***ở giữa giao điểm lẻ và giao điểm chẵn liên tiếp*** là những điểm nằm trong đa giác và những điểm này sẽ được tô.

Bước 5: Tăng x_0 lên một Pixel. Nếu $x_0 \le Max(x_i)$ thì quay lại bước 1.

Sau đây là chương trình cài đặt thuật toán Scanline:

```
#include <conio.h>
#include <iostream>
#include <graphics.h>
using namespace std;
```

```
struct Toado
{
   int x,y;
};
Toado P[100];
int minx,maxx,n,color1,color2;
//-----------------  -------------
void Nhaptoado(Toado a[],int &n)
{
/*
  cout<<endl<<"Nhap so dinh cho da giac:";
  cin>>n;
  for(int i=1;i<=n;i++)
  {
    cout<<endl<<"a["<<i<<"].x=";
    cin>>a[i].x;
    cout<<endl<<"a["<<i<<"].y=";
    cin>>a[i].y;
  }
 */
n=8;
P[1].x = 50;
P[1].y = 250;
P[2].x = 250;
P[2].y = 300;
P[3].x = 350;
P[3].y = 260;
P[4].x = 270;
P[4].y = 180;
P[5].x = 300;
P[5].y = 150;
P[6].x = 200;
P[6].y = 50;
P[7].x = 70;
P[7].y = 100;
P[8].x = 150;
P[8].y = 150;
minx=a[1].x;
maxx=a[1].x;
for (int i=2;i<=n;i++)
{
  if (a[i].x<minx) minx=a[i].x;
  if (a[i].x>maxx) maxx=a[i].x;
}
}
//-------------------------------
void Vedagiac(Toado a[],int n)
{
  setcolor(color1);
```

```
  for (int i=1;i<=n;i++)
  {
      int j;
      if (i==n) j=1;
       else  j=i+1;
      line(a[i].x,a[i].y,a[j].x,a[j].y);
  }
}
//-------------------------------
void Tomau(Toado a[],int n)
{
  //Duyệt Dy từ Min tới Max
  for(int i=minx+1;i<=maxx-1;i++)
  {
     int m=0,t,s,tg,z[100];
     //Tìm các giao điểm  z[]
     for (int j=1;j<=n;j++)
     {
          t=j+1;    if (j==n) t=1;
          s=j-1;    if (j==1) s=n;
          if (i==a[j].x)
          {
           if((i>min(a[s].x,a[t].x))&&(i<max(a[s].x,a[t].x)))
           {
             m++;
             z[m]=a[j].y;
           }
           else
               {
                    m++;
                    z[m]=a[j].y;
                    m++;
                    z[m]=a[j].y;
               }
          }
          if ((i>min(a[j].x,a[t].x))&&(i<max(a[t].x,a[j].x)))
          {
               ++m;
               float r=(float)(a[t].y-a[j].y)/(a[t].x-a[j].x);
               z[m]=(int)(r*(i-a[j].x))+a[j].y;
          }
     }
    //Sắp xếp các giao điểm z[] tăng dần theo y
     for(int j=1;j<=m-1;++j)
       for (int k=j+1;k<=m;++k)
          if (z[j]>z[k])
               swap([j],z[k]);

     //Tô màu các đoạn từ giao điểm lẻ tới giao điểm chẵn
```

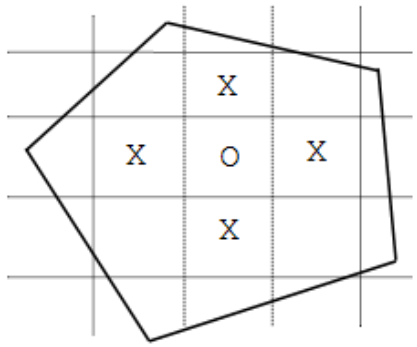
```
     setcolor(color2);
     for (int k=1;k<=m-1;k++)
          if (k%2!=0) line(i,z[k],i,z[k+1]);
     }
}
//------------------------------
main()
{
  int gd=0,gm;
  Nhaptoado(P,n);
  cout<<endl<<"Nhap mau duong vien:";
  cin>>color1;
  cout<<endl<<"Nhap mau to da giac:";
  cin>>color2;
  initgraph(&gd,&gm,"");
  Vedagiac(P,n);
  getch();
  Tomau(P,n);
  getch();
}
```

2.2.4. Thuật toán tô loang

Lấy P(x,y) ∈ S, tô màu P.

Xét các điểm lân cận của P (Hình 2.11).

Nếu các điểm lân cận đó vẫn còn thuộc S và chưa được tô màu thì tô màu các điểm lân cạn đó...



Hình 2.11

Thuật toán trên có thể được minh họa bằng hàm đệ qui:

```
void   ToLoang(int x,int y,int Color)
{
  If((x,y)∈S)&&((x,y)chưa  tô))
  {
       putpixel(x,y,Color);
       ToLoang(x+1,y,Color);
       ToLoang(x-1,y,Color);
       ToLoang(x,y+1,Color);
       ToLoang(x,y-1,Color);
```

```
   }
}
```

Sau đây là chương trình cài đặt thuật toán tô loang (đệ quy):

```
#include<conio.h>
#include<graphics.h>
#include<iostream>
#include <math.h>
using namespace std;
struct toado
{
    int x,y;
};
toado A[100];
int n,mauvien = 4;

//--------------------------------
void Nhap()
{
  cout<<"Nhap vao so dinh cua da giac:";
  cin>>n;
   for (int i=0;i<n;++i)
   {
   cout<<"\nA["<<i+1<<"].x=";  cin>>A[i].x;
   cout<<"\nA["<<i+1<<"].y=";  cin>>A[i].y;
   }
}
//--------------------------------
void Vedagiac(toado A[],int n,int color)
{
   int i,j;
   setcolor(color);
   for(i=0;i<n;++i)
   {
       if (i==n-1) j=0;
       else  j=i+1;
       line(A[i].x,A[i].y,A[j].x,A[j].y);
   }
}
void Toloang(int x,int y,int color)
{
   if (getpixel(x,y)!= mauvien && getpixel(x,y)!= color)
```

```
    {
       putpixel(x,y,color);
       Toloang(x+1,y,color);
       Toloang(x-1,y,color) ;
       Toloang(x,y+1,color);
       Toloang(x,y-1,color);
    }
}
//---------------------------
int main()
{
  Nhap();
  int gd= DETECT, gm;
  initgraph(&gd,&gm,"");
  Vedagiac(A,n,mauvien);
  getch();
  Toloang(10,10,14);
  getch();
}
```

NHẬN XÉT: Thuật toán tô loang đệ quy có ưu điểm là cài đặt đơn giản, ngắn gọn. Tuy nhiên nó có nhược điểm lớn là dễ tràng bộ nhớ nếu vùng tô hơi lớn, nhược điểm này có thể khắc phục bằng cách sử dụng phương pháp khử đệ quy.

Sau đây là chương trình cài đặt thuật toán tô loang (khử đệ quy):

```
#include<conio.h>
#include<graphics.h>
#include<iostream>
#include<stack>
#include <math.h>
using namespace std;
struct toado
{
    int x,y;
};
toado A[100];
int n,mauvien = 10;
//-----------------------
void Nhap()
{
  cout<<"So dinh cua da giac:";
  cin>>n;
  for (int i=0;i<n;++i)
```

```
   {
    cout<<"\nA["<<i+1<<"].x=";  cin>>A[i].x;
    cout<<"\nA["<<i+1<<"].y=";  cin>>A[i].y;
   }
}
//---------------------------------
void Vedagiac(toado A[],int n,int color)
 {
   int i,j;
   setcolor(color);
   for(i=0;i<n;++i)
   {
       if (i==n-1) j=0;
       else  j=i+1;
       line(A[i].x,A[i].y,A[j].x,A[j].y);
   }
 }
void Toloang(int x,int y,int color)
{
   stack<toado> S;
   toado P;
   P.x = x; P.y = y;
   S.push(P);
   while(!S.empty())
   {
       P = S.top();
       S.pop();
       if (getpixel(P.x,P.y)!= mauvien &&
                          getpixel(P.x,P.y)!= color)
       {
           putpixel(P.x,P.y,color);
           toado P1,P2,P3,P4;
           P1.x = P.x+1; P1.y = P.y;
           S.push(P1);
           P2.x = P.x-1; P2.y = P.y;
           S.push(P2);
           P3.x = P.x; P3.y = P.y+1;
           S.push(P3);
           P4.x = P.x; P4.y = P.y-1;
           S.push(P4);
       }
```

```
    }
  }
  //------------------------
  main()
  {
    Nhap();
    int gd= DETECT, gm;
    initgraph(&gd,&gm,"");
    Vedagiac(A,n,mauvien);
    getch();
    Toloang(10,10,14);
    getch();
  }
```

BÀI TẬP

1. Viết hàm **int STamGiac(ToaDo2D A, ToaDo2D B, ToaDo2D C)** để tính diện tích tam giác ABC.

2. Viết hàm **int DienTich(ToaDo2D P[ ]; int n)** để tính diện tích của đa giác lồi có n đỉnh lưu trong mảng P.

3. Viết hàm **int KiemTra(int x, int y, ToaDo2D P[ ]; int n)** để kiểm tra điểm (x,y) nằm trong hay ngoài đa giác có n đỉnh được lưu trong mảng P theo hai cách:

a) Dùng công thức tính diện tích đa giác (đối với đa giác lồi).

b) Dùng thuật toán Jordan (đối với đa giác bất kỳ).

4. Viết hàm **FillRec(int x1, int y1, int x2, int y2, int color)** để tô màu hình chữ nhật xác định bởi 2 đỉnh (x1,y1) và (x2,y2).

5. Viết hàm **FillEllipse(int x, int y, int Rx, int Ry, int color)** để tô màu Ellipse có tâm (x,y) và bán kính theo hai trục là Rx và Ry.

6. Viết hàm **FillSector(int x, int y, int Rx, int Ry, int g1, int g2, int color)** để tô màu hình quạt Ellipse có tâm (x,y), bán kính theo hai trục là Rx và Ry, góc bắt đầu là g1 và góc kết thúc là g2.

7. Viết hàm **Donut(int x, int y, int Rmin, int Rmax, int color)** để tô màu hình vành khăn có tâm (x,y) và bán kính hai đường tròn tương ứng là Rmin và Rmax.

CHƯƠNG 3 : XÉN HÌNH

3.1. BÀI TOÁN TỔNG QUÁT

Cho miền $D \subset R^n$ và F là một hình trong R^n ($F \subset R^n$). Gọi $F \cap D$ là hình có được bằng cách xén hình F vào trong miền D và ký hiệu là **$Clip_D(F)$**. Hãy xác định **$Clip_D(F)$**?

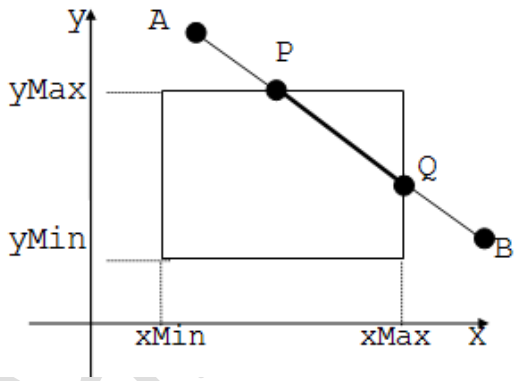
3.2. XÉN ĐOẠN THẲNG VÀO HÌNH CHỮ NHẬT

Lúc này:

$$D = \left\{ (x, y) \in R^2 \middle| \begin{matrix} X\min \le x \le X\max \\ Y\min \le y \le Y\max \end{matrix} \right\}$$

và F là đoạn thẳng nối 2 điểm (x_1,y_1), (x_2,y_2) nên phương trình tham số của F là:

$$\begin{cases} x = x_1 + (x_2 - x_1).t \\ y = y_1 + (y_2 - y_1).t \end{cases} \qquad t \in [0,1]$$



Hình 3.1

Do đó, F có thể được viết dưới dạng:

$F = \{(x,y) \in R^2 \mid x = x_1 + (x_2 - x_1).t;\ y = y_1 + (y_2 - y_1).t;\ 0 \le t \le 1\}$

Khi đó, giao điểm của F và D chính là nghiệm của hệ bất phương trình (theo t):

$$\begin{cases} Xmin \le x_1 + (x_2 - x_1).t \le Xmax \\ Ymin \le y_1 + (y_2 - y_1).t \le Ymax \\ 0 \le t \le 1 \end{cases}$$

Gọi N là tập nghiệm của hệ phương trình trên.

Nếu $N = \varnothing$ thì $Clip_D(F) = \varnothing$

Nếu $N \ne \varnothing$ thì $N = [t_1, t_2]$ $(t_1 \le t_2)$

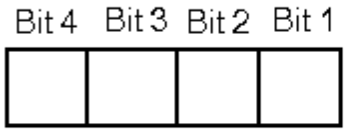
Gọi P, Q là 2 giao điểm được xác định bởi:

$$\begin{cases} x_P = x_1 + (x_2 - x_1).t_1 \\ y_P = y_1 + (y_2 - y_1).t_1 \end{cases} \quad \text{và} \quad \begin{cases} x_Q = x_1 + (x_2 - x_1).t_2 \\ y_Q = y_1 + (y_2 - y_1).t_2 \end{cases}$$

thì: **$Clip_D(F) = PQ$** (Hình 3.1).

3.2.1. Thuật toán Cohen - Sutherland

- Chia mặt phẳng ra làm 9 vùng, mỗi vùng đánh một mã nhị phân 4 bit (Hình 3.2).



Mã nhị phân

1001	1000	1010
0001	0000	0010
0101	0100	0110

Hình 3.2

Bit 1: Qui định vùng nằm bên trái cửa sổ
Bit 2: Qui định vùng nằm bên phải cửa sổ
Bit 3: Qui định vùng nằm bên dưới cửa sổ
Bit 4: Qui định vùng nằm bên trên cửa sổ

- Xét điểm P(x,y) $\in R^2$:

$$P_{Left} = \begin{cases} 1, & x_P < X\min \\ 0, & nguoclai \end{cases}$$

$$P_{Right} = \begin{cases} 1, & x_P > X\max \\ 0, & nguoclai \end{cases}$$

$$P_{Below} = \begin{cases} 1, & y_P < Y\min \\ 0, & nguoclai \end{cases}$$

$$P_{Above} = \begin{cases} 1, & y_P > Y\max \\ 0, & nguoclai \end{cases}$$

- Xét đoạn thẳng AB, ta có các trường hợp sau:

i/ Nếu Mã(A) = Mã(B) = 0000 thì AB $\in$ D $\Rightarrow$ **$Clip_D(F) = AB$**

ii/ Nếu Mã(A) AND Mã(B) $\neq$ 0000 thì đoạn AB nằm hoàn toàn bên ngoài hình chữ nhật $\Rightarrow$ **$Clip_D(F) = \varnothing$.**

Chú ý: Phép toán AND là phép toán logic giữa các bit.

Nếu (Mã(A) AND Mã(B) = 0000) thì: Giả sử Mã(A) $\neq$ 0000 $\Leftrightarrow$ A nằm ngoài hình chữ nhật.

- ♦ Nếu A_{Left} = 1: thay A bởi điểm nằm trên đoạn AB và cắt cạnh trái (nối dài) của hình chữ nhật.
- ♦ Nếu A_{Right} =1: thay A bởi điểm nằm trên đoạn AB cắt cạnh phải (nối dài) của hình chữ nhật.
- ♦ Nếu A_{Below} =1: thay A bởi điểm nằm trên đoạn AB và cắt cạnh dưới (nối dài) của hình chữ nhật.
- ♦ Nếu A_{Above} =1: thay A bởi điểm nằm trên đoạn AB và cắt cạnh trên (nối dài) của hình chữ nhật.

Chú ý: Quá trình này được lặp lại: Sau mỗi lần lặp, ta phải tính lại mã của A. Nếu cần, phải đổi vai trò của A và B để đảm bảo A luôn luôn nằm bên ngoài hình chữ nhật. Quá trình sẽ dừng khi xẩy ra một trong 2 trường hợp: i/ hoặc ii/.

Sau đây là chương trình cài đặt thuật toán Cohen-Sutherland:

```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#define LEFT 1
#define RIGHT 2
#define BELOW 4
#define ABOVE 8
#define TRUE 1
#define FALSE 0
struct ToaDo2D
{
  int  x,y;
};
ToaDo2D Tren,Duoi,A,B;
void nhap()
{
  Tren.x=150;Tren.y=100;
  Duoi.x=400;Duoi.y=300;
  printf("\nA.x=");scanf("%d",&A.x);
  printf("\nA.y=");scanf("%d",&A.y);
  printf("\nB.x=");scanf("%d",&B.x);
  printf("\nB.y=");scanf("%d",&B.y);
}
void vehinh()
{
  rectangle(Tren.x,Tren.y,Duoi.x,Duoi.y);
  setwritemode(XOR_PUT);
  line(A.x,A.y,B.x,B.y); //vẽ
  getch();
  line(A.x,A.y,B.x,B.y); //xóa
}
int  Ma( ToaDo2D P)
{
 int S=0;
 if (P.x<Tren.x)  S=S | LEFT;
 if (P.x>Duoi.x)  S=S | RIGHT;
 if (P.y>Duoi.y)  S=S | BELOW;
```

```
 if (P.y<Tren.y)  S=S | ABOVE;
 return S;
}
int round(float x)
{
 if(x>0) return int (x+0.5);
 else return int (x-0.5);
}
void Swap(ToaDo2D &A,ToaDo2D &B)
{
 ToaDo2D t=A; A=B; B=t;
}
void clip(ToaDo2D A,ToaDo2D B,ToaDo2D Tren,ToaDo2D Duoi)
{
 int stop=FALSE,draw=FALSE;
 while (stop!=TRUE)
 {
   if((Ma(A)==0)&&(Ma(B)==0))
   {
     stop=TRUE; draw=TRUE;
   }
   else
     if((Ma(A)&Ma(B))!=0) stop=TRUE;
     else
       {
        if(((Ma(A)&Ma(B))==0)
           &&((Ma(A)!=0)||(Ma(B)!=0)))
         {
           if(Ma(A)==0) Swap(A,B);
           float m=float(B.y-A.y)/(B.x-A.x);
           if((Ma(A) & LEFT)!=0)
           {
                A.y=A.y+round(m*(Tren.x - A.x));
                A.x=Tren.x;
           }
           else
            if((Ma(A) & RIGHT)!=0)
            {
                A.y=A.y+round(m*(Duoi.x - A.x));
                A.x=Duoi.x;
            }
```

```
                else
                     if((Ma(A) & ABOVE)!=0)
                     {
                          A.x=A.x+round(float(Tren.y-A.y)/m);
                          A.y=Tren.y;
                     }
                     else
                       if((Ma(A) & BELOW)!=0)
                       {
                          A.x=A.x+round(float(Duoi.y-A.y)/m);
                          A.y=Duoi.y;
                       }
                }
           }
     }
     if(draw) line(A.x,A.y,B.x,B.y);
   }
   main()
   {
     nhap();
     int driver = DETECT,mode;
     initgraph(&driver,&mode,"d:\\tc\\bgi");
     vehinh();
     clip(A,B,Tren,Duoi);
     getch();
     closegraph();
   }
```

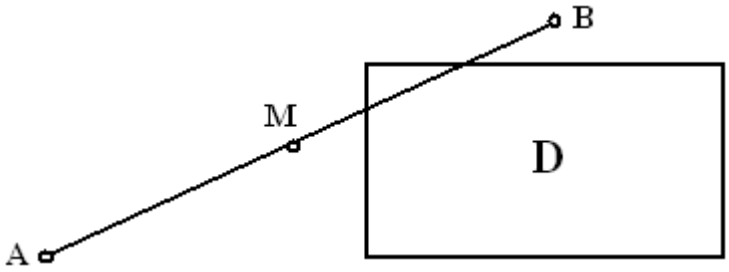
3.2.2. Thuật toán chia nhị phân

- Ý tưởng của thuật toán này tương tự như thuật toán tìm nghiệm bằng phương pháp chia nhị phân.
- **Mệnh đề**: Cho M: trung điểm của đoạn AB, Mã(A) ≠ 0000, Mã(B) ≠ 0000, Mã(M) ≠ 0000 thì ta có:

  [Mã(A) AND Mã(M)] ≠ 0000

  hoặc [Mã(M) AND Mã(B)] ≠ 0000.

✍ **Ý nghĩa hình học của mệnh đề**: *Nếu cả ba điểm A, B, M đều ở ngoài hình chữ nhật thì có ít nhất nửa đoạn hoàn toàn nằm ngoài hình chữ nhật (Hình 3.3)* .



Hình 3.3

- Ta phát thảo thuật toán như sau:

i/ Nếu Mã(A) = 0000 và Mã(B) = 0000 thì $\mathbf{Clip_D(F) = AB}$

ii/ Nếu Mã(A) AND Mã(B) ≠ 0000 thì $\mathbf{Clip_D(F) = \varnothing}$

iii/ Nếu Mã(A) = 0000 và Mã(B) ≠ 0000 thì:

P:=A; Q:=B;

Trong khi $|x_P - x_Q| + |y_P - y_Q| \geq 2$ thì:

- ♦ Lấy trung điểm M của PQ;
- ♦ Nếu Mã(M) ≠ 0000 thì Q:= M.

Ngược lại: P:= M.

$\Rightarrow \mathbf{Clip_D(F) = AP}$

iv/ Nếu Mã(A) ≠ 0000 và Mã(B) = 0000 thì: Đổi vai trò của A, B và áp dụng ii/

v/ Nếu Mã(A) ≠ 0000 ≠ Mã(B) và [Mã(A) AND Mã(B)]= 0000 thì:

P:=A; Q:=B;

Lấy M: trung điểm PQ;

Trong khi Mã(M) ≠ 0000 và $|x_P - x_Q| + |y_P - y_Q| \geq 2$ thì:

- ♦ Nếu Mã(M) AND Mã(Q) ≠ 0000 thì Q:=M.
  Ngược lại P:=M.
- ♦ Lấy M: trung điểm PQ.

Nếu Mã(M) ≠ 0000 thì $\mathbf{Clip_D(F) = \varnothing}$. Ngược lại, áp dụng ii/ ta có:

$\mathbf{Clip_D(MA) = MA_1}$

$\mathbf{Clip_D(MB) = MB_1}$

Suy ra: $\mathbf{Clip_D(F) = A_1B_1}$

Sau đây là chương trình cài đặt thuật toán chia nhị phân:

```
#include <graphics.h>
#include <stdio.h>
#include <conio.h>
#include <stdlib.h>
struct Toado2D
{
  int  x,y;
};
Toado2D Tren,Duoi,A,B;
void nhap()
```

```
{
  Tren.x=150;Tren.y=100;
  Duoi.x=400;Duoi.y=300;
  printf("\nA.x=");scanf("%d",&A.x);
  printf("\nA.y=");scanf("%d",&A.y);
  printf("\nB.x=");scanf("%d",&B.x);
  printf("\nB.y=");scanf("%d",&B.y);
}
void vehinh()
{
  rectangle(Tren.x,Tren.y,Duoi.x,Duoi.y);
  setwritemode(XOR_PUT);
  line(A.x,A.y,B.x,B.y);
  getch();
  line(A.x,A.y,B.x,B.y);
}
int  Ma( Toado2D P)
{
 int S=0;
 if (P.x<Tren.x)  S=S | 1;
 if (P.x>Duoi.x)  S=S | 2;
 if (P.y>Duoi.y)  S=S | 4;
 if (P.y<Tren.y)  S=S | 8;
 return S;
}
void trongngoai(Toado2D A,Toado2D B)
{
 Toado2D P,Q,M;
 P=A; Q= B;
 while (abs(P.x-Q.x)+abs(P.y- Q.y)> 2)
  {
   M.x=(P.x+ Q.x) / 2;
   M.y=(P.y+ Q.y) / 2;
   if (Ma(M)==0) P=M;  else Q=M;
  }
 line(A.x,A.y,P.x,P.y);
}
void clip(Toado2D A,Toado2D B)
{
 Toado2D P,Q,M;
 if((Ma(A)==0)&&(Ma(B)==0)) line(A.x,A.y,B.x,B.y);
```

```
  else if((Ma(A)==0)&&(Ma(B)!=0)) trongngoai(A,B);
    else if((Ma(A)!=0)&&(Ma(B)==0)) trongngoai(B,A);
      else
       if ((Ma(A) & Ma(B))==0)
       {
         P=A;Q=B;
         M.x=(P.x+Q.x)/2;
         M.y=(P.y+Q.y)/2;
         while((Ma(M)!=0)&&((Ma(P)& Ma(Q))==0))
         {
             if (Ma(P)&Ma(M)!=0) P=M; else Q=M;
             M.x=(P.x+Q.x)/2;
             M.y=(P.y+Q.y)/2;
         }
         if (Ma(M)==0)
         {
             trongngoai(M,P);
             trongngoai(M,Q);
         }
      }
}
main()
{
  nhap();
  int driver = DETECT,mode;
  initgraph(&driver,&mode,"d:\\tc\\bgi");
  vehinh();
  clip(A,B);
  getch();
  closegraph();
}
```

3.2.3. Thuật toán Liang - Barsky

Đặt $\Delta x = x2 - x1$ $\Delta y = y2 - y1$

$p_1 = -\Delta x$ $q_1 = x1 - xMin$

$p_2 = \Delta x$ $q_2 = xMax - x1$

$p_3 = -\Delta y$ $q_3 = y1 - yMin$

$p_4 = \Delta y$ $q_4 = yMax - y1$

thì hệ bất phương trình giao điểm của F và D có thể viết lại:

$$\begin{cases} p_k.t \le q_k, k = 1..4 \\ 0 \le t \le 1 \end{cases}$$

Xét các trường hợp sau:

i/ $\exists k$: $p_k = 0$ và $q_k < 0$: (Đường thẳng song song với các biên và nằm ngoài vùng hình chữ nhật)

$\Rightarrow$ **$Clip_D(F) = \varnothing$**

ii/ $\forall k \in \{1,2,3,4\}$: $p_k \neq 0$ hoặc $q_k \geq 0$:

Đặt $K_1 = \{k \mid p_k < 0\}$

$K_2 = \{k \mid p_k > 0\}$

$$u_1 = Max\left\{\left\{\frac{q_k}{p_k} | k \in K_1\right\} \cup \{0\}\right\}$$

$$u_2 = Min\left\{\left\{\frac{q_k}{p_k} | k \in K_2\right\} \cup \{1\}\right\}$$

Nếu $u_1 > u_2$ thì **$Clip_D(F) = \varnothing$**

Ngược lại: Gọi P, Q là 2 điểm thỏa mãn:

$$\begin{cases} x_P = x_1 + \Delta x.u_1 \\ y_P = y_1 + \Delta y.u_1 \end{cases} \quad \text{và} \quad \begin{cases} x_Q = x_1 + \Delta x.u_2 \\ y_Q = y_1 + \Delta y.u_2 \end{cases}$$

$\Rightarrow$ **$Clip_D(F) = PQ$**

Sau đây là chương trình cài đặt thuật toán Liang-Barsky:

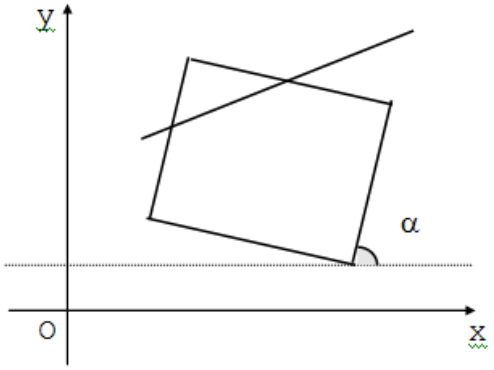
```
#include <conio.h>
#include <graphics.h>
#include <stdlib.h>
#define TRUE 1
#define FALSE 0
struct ToaDo2D
{
   int  x,y;
};
int Round(float x)
{
 if(x>0) return int (x+0.5);
 else return int (x-0.5);
}
int cliptest(float p,float q,float &u1,float &u2)
{
  float r;
  int clip = TRUE;
  if (p < 0)
  {
```

```
      r = q / p;
      if (r > u2)  clip = FALSE;
      else
        if (r > u1) u1 = r;
    }
    else
      if ( p > 0)
      {
        r = q / p;
        if (r < u1) clip = FALSE;
        else
             if (r < u2) u2 = r;
      }
      else
        if (q < 0) clip = FALSE;
    return clip;
}
void clip(ToaDo2D A,ToaDo2D B,ToaDo2D Tren, ToaDo2D Duoi)
{
    float u1 = 0, u2 = 1, dx, dy;
    dx = B.x - A.x;
    if (cliptest(-dx, A.x - Tren.x, u1, u2))
    if (cliptest(dx, Duoi.x-A.x, u1, u2))
    {
        dy = B.y - A.y;
        if (cliptest(-dy, A.y - Tren.y, u1, u2))
          if (cliptest(dy, Duoi.y - A.y, u1, u2))
          {
             if (u2 < 1)
             {
                  B.x = A.x + Round(u2*dx);
                  B.y = A.y + Round(u2*dy);
             }
             if (u1 > 0)
             {
                  A.x += Round(u1*dx);
                  A.y += Round(u1*dy);
             }
             line(A.x, A.y, B.x, B.y);
          }
    }
```

```
}
void khoitaodohoa()
{
  int gd = 0, gm;
  initgraph(&gd, &gm,"D:\\TC\\BGI");
}
main()
{
 khoitaodohoa();
 ToaDo2D Tren,Duoi,A,B;
 Tren.x=150;Tren.y=100;
 Duoi.x=400;Duoi.y=300;
 rectangle(Tren.x,Tren.y,Duoi.x,Duoi.y);
 setwritemode(XOR_PUT);
 A.x = 10; A.y = 10;
 B.x = 450;B.y = 400;
 line(A.x,A.y,B.x,B.y); //Vẽ AB
 getch();
 line(A.x,A.y,B.x,B.y); //Xóa AB
 clip(A,B,Tren,Duoi);   //Vẽ đoạn xén
 closegraph();
}
```

3.2.4. Khi cạnh của hình chữ nhật tạo với trục hoành một góc $\alpha \in (0,\pi/2)$

Ta dùng phép quay trục tọa độ để đưa bài toán về trường hợp các cạnh của hình chữ nhật song song với các trục tọa độ (Hình 3.4).



Hình 3.4

Gọi R là ma trận quay của phép đổi trục, ta có:

$$\begin{pmatrix} X\min \\ Y\min \end{pmatrix} = \mathrm{R}.\begin{pmatrix} X\min \\ Y\min \end{pmatrix}$$

$$\begin{pmatrix} X\max \\ Y\max \end{pmatrix} = \mathrm{R}.\begin{pmatrix} X\max \\ Y\max \end{pmatrix}$$

với $R = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}$

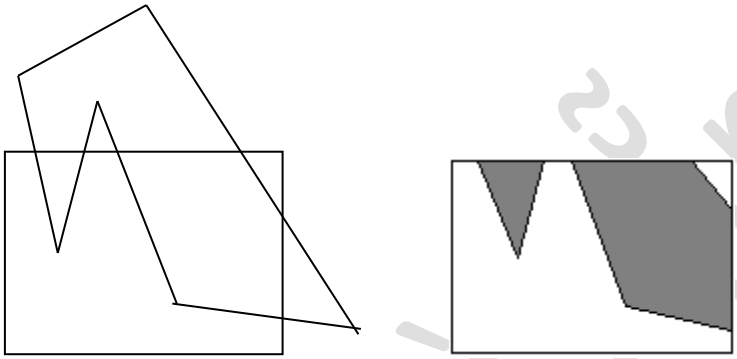
3.5. XÉN ĐA GIÁC VÀO HÌNH CHỮ NHẬT

*** Thuật toán Sutherland – Hodgman:**

i/ Nếu tất cả các đỉnh của đa giác đều nằm trong hình chữ nhật thì hình cần xén chính là đa giác và bài toán coi như đã được giải quyết.

ii/ Trường hợp ngược lại:

Với mỗi cạnh của đa giác, xét các trường hợp sau:

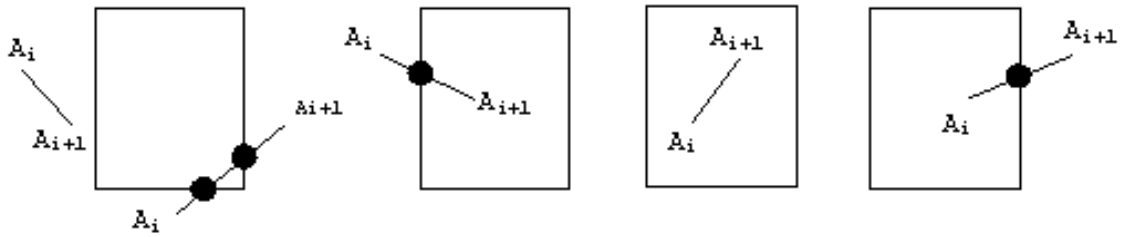


Hình 3.5. Xén đa giác vào hình chữ nhật

- Nếu cả hai đỉnh đều nằm ngoài hình chữ nhật thì:

  Nếu Ma(A_i) and Ma(A_{i+1}) ≠ 0000 thì không lưu đỉnh

  Ngược lại thì lưu hai giao điểm.
- A_i ngoài, A_{i+1} trong: lưu giao điểm P và A_{i+1}.
- Cả hai đỉnh đều nằm trong hình chữ nhật: lưu A_i và A_{i+1}.
- A_i trong, A_{i+1} ngoài: lưu A_i và giao điểm P.



Hình 3.6. Các trường hợp cần xét

Sau khi duyệt qua tất cả các cạnh của đa giác thì ta có được một dãy các đỉnh mới phát sinh: B_1, B_2, ..., B_n.

Nếu trong dãy các đỉnh mới này có hai đỉnh liên tiếp không nằm trên cùng một cạnh của hình chữ nhật , giả sử hai đỉnh đó là B_i và B_{i+1} thì ta đi dọc các cạnh của hình chữ nhật từ B_i đến B_{i+1} để tìm tất cả các đỉnh của hình chữ nhật nằm trong đa giác rồi bổ sung chúng vào giữa B_i và B_j.

Tập đỉnh mới tìm được chính là đa giác xén được.

Nếu tập đỉnh mới này là rỗng: Nếu có một đỉnh của hình chữ nhật nằm trong đa giác thì hình xén được chính là toàn bộ hình chữ nhật. Ngược lại, hình xén được là rỗng.

Sau đây là chương trình cài đặt thuật toán Sutherland - Hodgman:

```
#include<conio.h>
#include<iostream.h>
```

```
#include<graphics.h>
#define LEFT 1
#define RIGHT 2
#define ABOVE 3
#define BELOW 4
struct ToaDo2D
{
  int x,y;
};
ToaDo2D h[100],luu[100];
ToaDo2D tren,duoi,giaodiem;
int n,dem;
int round(float x)
{
  if(x>0) return int(x+0.5);
  else return int(x-0.5);
}
void NhapDinh(ToaDo2D P[])
{
  cout<<"\nNhap so dinh da giac:";cin>>n;
  for(int i=0;i<n;i++)
  {
    cout<<"Nhap toa do dinh thu "<<i+1<<" :\n";
    cout<<" x = ";cin>>P[i].x;
    cout<<" y = ";cin>>P[i].y;
  }
  P[n]=P[0];
  tren.x=100;
  tren.y=100;
  duoi.x=300;
  duoi.y=300;
}
void VeHinh(ToaDo2D P[])
{
  line(tren.x,tren.y,duoi.x,tren.y);
  line(duoi.x,tren.y,duoi.x,duoi.y);
  line(duoi.x,duoi.y,tren.x,duoi.y);
  line(tren.x,duoi.y,tren.x,tren.y);
  setwritemode(XOR_PUT);
  setcolor(14);
  for(int i=0;i<n;i++)
```

```
    line(P[i].x,P[i].y,P[i+1].x,P[i+1].y);
  getch();
  for(i=0;i<n;i++)
    line(P[i].x,P[i].y,P[i+1].x,P[i+1].y);
  setwritemode(COPY_PUT);
}
void giao(ToaDo2D p1,ToaDo2D p2,int canh)
{
  float k;
  switch (canh)
  {
    case 1://trai
       giaodiem.x=tren.x;
       k=(float)(p1.y-p2.y)/(p1.x-p2.x);
       giaodiem.y=round((tren.x-p1.x)*k +p1.y);
       break;
    case 2://phai
       giaodiem.x=duoi.x;
       k=(float)(p1.y-p2.y)/(p1.x-p2.x);
       giaodiem.y=round((duoi.x-p1.x)*k +p1.y);
       break;
    case 3://tren
       giaodiem.y=tren.y;
       k=(float)(p1.x-p2.x)/(p1.y-p2.y);
       giaodiem.x=round((tren.y-p1.y)*k +p1.x);
       break;
    case 4://duoi
       giaodiem.y=duoi.y;
       k=(float)(p1.x-p2.x)/(p1.y-p2.y);
       giaodiem.x=round((duoi.y-p1.y)*k +p1.x);
       break;
  }
}
int trong(ToaDo2D P,int canh)
{
  int tam = 0;
  switch (canh)
  {
    case 1: if (P.x>tren.x) tam = 1;
            break;
    case 2: if (P.x<duoi.x) tam = 1;
```

```
            break;
     case 3: if (P.y>tren.y) tam = 1;
            break;
     case 4: if (P.y<duoi.y) tam = 1;
            break;
   }
   return tam;
}
void xencanh(int canh)
{
   int truoc=0, dem=0;
   if (trong(h[0],canh))
   {
     luu[dem]=h[0];
     truoc=1;
     dem++;
   }
   for(int i=1;i<n;i++)
   {
    if (truoc)
      if (trong(h[i],canh))
      {
        luu[dem]=h[i];
        truoc=1;
        dem++;
      }
      else
    {
      giao(h[i-1],h[i],canh);
      luu[dem]=giaodiem;
      truoc=0;
      dem++;
    }
    else
       if (trong(h[i],canh))
       {
            giao(h[i-1],h[i],canh);
            luu[dem]=giaodiem;
            truoc=1;
            dem++;
            luu[dem]=h[i];
```

```
            dem++;
        }
    }
    if(truoc)
     if (trong(h[0],canh)) truoc=1;
     else
        {
            giao(h[n-1],h[0],canh);
            luu[dem]=giaodiem;
            truoc=0;
            dem++;
        }
     else
       if (trong(h[0],canh))
       {
         giao(h[n-1],h[0],canh);
         luu[dem]=giaodiem;
         truoc=0;
         dem++;
       }
       else truoc=0;
 for(i=0;i<(dem);i++) h[i]=luu[i];
 h[dem]=luu[0];
 n=dem;
}
void xendagiac()
{
  xencanh(LEFT);
  xencanh(RIGHT);
  xencanh(ABOVE);
  xencanh(BELOW);
}
void ve(ToaDo2D luu[100])
{
  setcolor(14);
  for(int i=0;i<n;i++)
     line(luu[i].x,luu[i].y,luu[i+1].x,luu[i+1].y);
}
Int main()
{
  NhapDinh(h);
```

```
    int gd=0,gm;
    initgraph(&gd,&gm,"");
    VeHinh(h);
    xendagiac();
    ve(h);
    getch();
    closegraph;
}
```

BÀI TẬP

1. Cài đặt thuật toán xén một đoạn thẳng vào một hình tròn.
2. Cài đặt thuật toán xén một hình tròn vào hình chữ nhật.
3. Cài đặt thuật toán xén đoạn thẳng vào hình chữ nhật có cạnh tạo với trục hoành một góc α.

CHƯƠNG 4: THIẾT KẾ ĐƯỜNG CONG BEZIER VÀ B-SPLINE

Khác với những phương pháp biểu diễn mặt và đường bởi các công thức toán học tường minh, ở đây ta sẽ bàn đến các công cụ cho phép chỉ ra các dạng đường và mặt khác nhau dựa trên các dữ liệu.

Điều này có nghĩa là với một đường cong cho trước mà ta chưa xác định được công thức toán học của nó thì làm thế nào để có thể nắm bắt được dạng của đường cong đó một cách tương đối chính xác qua việc sử dụng một tập nhỏ các điểm P_0, P_1,... cùng với một phương pháp nội suy nào đó từ tập điểm này để tạo ra đường cong mong muốn với một độ chính xác cho phép.

Có nhiều cách để nắm bắt được đường cong cho trước, chẳng hạn:

- Lấy một mẫu đường cong chừng vài chục điểm cách nhau tương đối ngắn rồi tìm một hàm toán học và chỉnh hàm này sao cho nó đi qua các điểm này và khớp với đường cong ban đầu. Khi đó, ta có được công thức của đường và dùng nó để vẽ lại đường cong.
- Cách khác là dùng một tập các điểm kiểm soát và dùng một thuật toán để xây dựng nên một đường cong của riêng nó dựa trên các điểm này. Có thể đường cong ban đầu và đường cong tạo ra không khớp nhau lắm, khi đó ta có thể di chuyển một vài điểm kiểm soát và lúc này thuật toán lại phát sinh một đường cong mới dựa trên tập điểm kiểm soát mới. Tiến trình này lặp lại cho đến khi đường cong tạo ra khớp với đường cong ban đầu.

Ở đây, ta sẽ tiếp cận vấn đề theo phương pháp thứ hai, dùng đến các đường cong Bezier và B-Spline để tạo các đường và mặt.

Giả sử một điểm trong không gian được biểu diễn dưới dạng vector tham số p(t). Với các đường cong 2D, p(t) = (x(t), y(t)) và các đường 3D, p(t) = (x(t), y(t), z(t)).

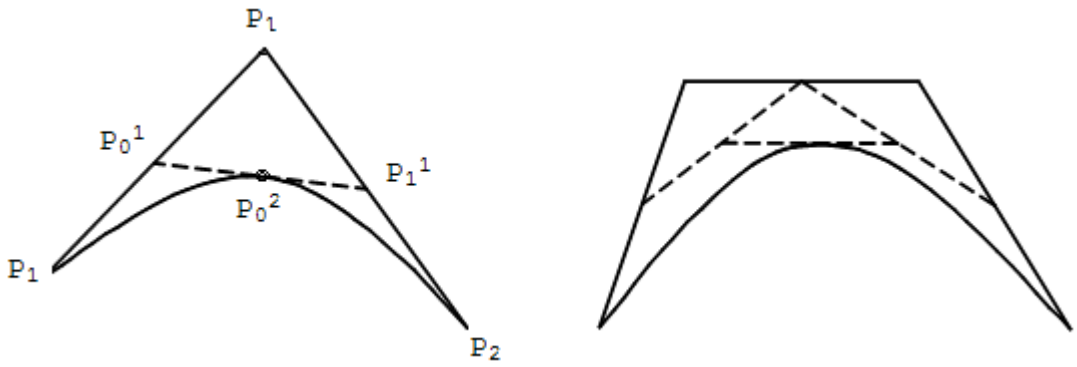
4.1. ĐƯỜNG CONG BEZIER VÀ MẶT BEZIER

4.1.1. Thuật toán Casteljau

Để xây dựng đường cong p(t), ta dựa trên một dãy các điểm cho trước rồi tạo ra giá trị p(t) ứng với mỗi giá trị t nào đó. Việc thay đổi các điểm này sẽ làm thay đổi dạng của đường cong. Phương pháp này tạo ra đường cong dựa trên một dãy các bước nội suy tuyến tính hay ***nội suy khoảng giữa*** (In-Betweening).

Ví dụ: Với 3 điểm P_0, P_1, P_2 ta có thể xây dựng một Parabol nội suy từ 3 điểm này bằng cách chọn một giá trị $t \in [0, 1]$ nào đó rồi chia đoạn P_0P_1 theo tỉ lệ t, ta được điểm P_0^1 trên P_0P_1 . Tương tự, ta chia tiếp P_1P_2 cũng theo tỉ lệ t, ta được P_1^1. Nối P_0^1 và P_1^1, lại lấy điểm trên $P_0^1P_1^1$ chia theo tỉ lệ t, ta được P_0^2.

Với cách làm này, ta sẽ lấy những giá trị t khác $\in [0, 1]$ thì sẽ được tập điểm P_0^2. Đó chính là đường cong p(t) (Hình 4.1).



Đường cong Bezier bậc 2 Đường cong Bezier bậc 3

Hình 4.1

Ta biểu diễn bằng phương trình:

$$P_0^1(t) = (1-t).P_0 + t.P_1 \quad (1)$$

$$P_1^1(t) = (1-t).P_1 + t.P_2 \quad (2)$$

$$P_0^2(t) = (1-t).P_0^1 + t.P_1^1 \quad (3)$$

Thay (1), (2) vào (3) ta được:

$$P(t) = P_0^2(t) = (1-t)^2.P_0 + 2t.(1-t).P_1 + t^2.P_2$$

Đây là một đường cong bậc 2 theo t nên nó là một Parabol.

Tổng quát hóa ta có thuật toán Casteljau cho (L+1) điểm:

Giả sử ta có tập điểm: $P_0, P_1, P_2, ..., P_L$

Với mỗi giá trị t cho trước, ta tạo ra điểm $P_i^r(t)$ ở thế hệ thứ r, từ thế hệ thứ (r-1) trước đó, ta có:

$$P_i^r(t) = (1-t).P_i^{r-1}(t) + t.P_{i+1}^{r-1}(t) \quad (3')$$

r = 0,1,...,L và i = 0,...,L-r

Thế hệ cuối cùng $P_0^L(t)$ được gọi là **đường cong Bezier** của các điểm $P_0, P_1, P_2,...,P_L$.

Các điểm P_i , i=0,1,...,L được gọi là các **điểm kiểm soát** hay các điểm Bezier.

Đa giác tạo bởi các điểm kiểm soát này gọi là **đa giác kiểm soát** hay đa giác Bezier.

4.1.2. Dạng Bernstein của các đường cong Bezier

Đường cong Bezier dựa trên (L+1) điểm kiểm soát $P_0 ,P_1 , ...,P_L$ được cho bởi công thức:

$$\mathbf{P(t)} = \sum_{k=0}^{L} \mathbf{P_k.B_k^L(t)}$$

Trong đó, P(t) là một điểm trong mặt phẳng hoặc trong không gian.

$\mathbf{B_k^L(t)}$ được gọi là đa thức Bernstein, được cho bởi công thức:

$$B_k^L(t) = \frac{L!}{k!(L-k)!}(1-t)^{L-k}.t^k \qquad \text{với } L \geq k$$

Mỗi đa thức Bernstein có bậc là L, thông thường ta gọi các $B_k^L(t)$ là các **hàm trộn** (blending function).

Tương tự, đối với mặt Bezier ta có phương trình sau:

$$\mathbf{P(u,v)} = \sum_{i=0}^{M} \sum_{i=0}^{L} \mathbf{P_{i,k}.B_i^M(u).B_k^L(v)}$$

Trong trường hợp này, khối đa diện kiểm soát sẽ có (M+1).(L+1) đỉnh.

4.1.3. Tạo và vẽ các đường Bezier

Để tạo ra một đường cong Bezier từ một dãy các điểm kiểm soát ta sẽ áp dụng phương pháp lấy mẫu hàm p(t) ở các giá trị cách đều nhau của tham số $t \in [0,1]$, ví dụ có thể lấy $t_i = i/N$, i=0,1,...,N. Khi đó ta sẽ được các điểm $P(t_i)$ từ công thức Bezier. Nối các điểm này bằng các đoạn thẳng ta sẽ được đường cong Bezier gần đúng.

Sau đây là chương trình vẽ đường cong Bezier trong mặt phẳng:

```
#include <conio.h>
#include <graphics.h>
struct ToaDo
{
 int x,y;
};
void Thietlapdohoa()
{
   int gd=0,gm;
   initgraph(&gd,&gm,"");
}
int round(float x)
{
   float n;
   n=x-(int)x;
   if (n>0.5) return (int(x)+1);
   else return x;
}
float tich(int n)
{
   int c;
   float f = 1.0;
   for (c=n;c>1;c--) f *= round(c);
   return(f);
}
float CLK(int L,int k)
{
   return(tich(L)/(tich(k)*tich(L-k)));
}
float BKL(int k,int L,float t,float nt)
{
   float s;
   s = nt;    //C^k_n
   for(int i=1;i<=k;i++) s*=t; //t^k
```

```
    for (int i=1;i<=L-k;i++) s*=(1.0-t); //(1-t)^(L-k)
    return s;
}
void Pt(float &x,float &y,float t,float A[],int L,
                                              ToaDo diem[])
{
    int k;
    float s;
    x = y = 0;
    for (k=0;k<=L;k++)
    {
        s = BKL(k,L,t,A[k]);
        x+=(s*diem[k].x);
        y+=(s*diem[k].y);
    }
}
void VeBezier(ToaDo A[],int L,int SoDiem)
{
    float u, x,y;
    float p[20];
    for(int i=0;i<=L;i++) p[i]=CLK(L,i);
    for (int i=0;i<=SoDiem;i++)
    {
        u=(float)i/SoDiem;
        Pt(x,y,u,p,L,A);
        if(i==0) moveto(round(x),round(y));
        else lineto(round(x),round(y));
    }
}
main()
{
    ToaDo A[6];
    int L=5;
    A[0].x=50; A[0].y=75;
    A[1].x=120;A[1].y=100;
    A[2].x=150;A[2].y=180;
    A[3].x=220;A[3].y=300;
    A[4].x=290;A[4].y=150;
    A[5].x=400;A[5].y=80;
    Thietlapdohoa();
    for(int i=0;i<=L;i++) //Vẽ các điểm kiểm soát
```

```
        bar(round(A[i].x)-3,round(A[i].y)-3,
                    round(A[i].x)+3,round(A[i].y)+3);
    setcolor(YELLOW);
    VeBezier(A,L,100); //Vẽ đường Bezier
    getch();
}
```

4.1.4. Các tính chất của đường cong Bezier

i/ Nội suy được các điểm đầu và cuối (đi qua điểm đầu và điểm cuối của tập điểm kiểm soát).

Chứng minh:

Ta có: $P(t) = \sum_{k=0}^{L} P_k.B_k^L(t)$

Do đó $P(0) = \sum_{k=0}^{L} P_k.B_k^L(0)$

trong đó: $B_k^L(0) = \frac{L!}{k!(L-k)!}(1-0)^{L-k}.0^k \qquad \forall k \neq 0 \text{ và } k \neq L$

$= \frac{L!}{k!(L-k)!}.0 = 0$

Vì vậy, $P(0) = P_0.B_0^L(0) + P_L.B_L^L(0)$

$= P_0 + 0 = P_0$

Lý luận tương tự cho P(1). Ta có $P(1) = P_L$.

ii/ Bất biến Affine:

Khi biến đổi một đường cong Bezier, ta không cần biến đổi mọi điểm trên đường cong một cách riêng rẻ mà chỉ cần biến đổi các điểm kiểm soát của đường cong đó rồi sử dụng công thức Bernstein để tái tạo lại đường cong Bezier đã được biến đổi.

Chứng minh:

Giả sử điểm P(t) biến đổi Affine thành P'(t)

$$P'(t) = P(t).N + tr = \sum_{k=0}^{L} P_k.B_k^L(t).N + tr$$

Trong đó:

N: ma trận biến đổi.

tr: vector tịnh tiến.

Xét đường cong

$$\sum_{k=0}^{L} (P_k.N + tr).B_k^L(t) \qquad (*)$$

được tạo ra bằng cách biến đổi Affine các vector P_k. Ta sẽ chứng minh đường cong này chính là P'(t).

Khai triển (*) ta có:

$$\sum_{k=0}^{L} P_k.N.B_k^L(t) + \sum_{k=0}^{L} tr.B_k^L(t) = \sum_{k=0}^{L} P_k.N.B_k^L(t) + tr.\sum_{k=0}^{L} B_k^L(t) \qquad (**)$$

Nhưng theo đa thức Bernstein thì $\sum_{k=0}^{L} B_k^L(t) = (1-t+t)^L = 1$ nên số hạng thứ hai của (**) sẽ là tr.

Vì vậy, P'(t) nằm trên đường cong Bezier tạo ra bởi các điểm kiểm soát P_k.

iii/ Tính chất của bao lồi: đường cong Bezier P(t) không bao giờ đi ra ngoài bao lồi của nó.

Ở đây, bao lồi của các điểm kiểm soát là tập đỉnh nhỏ nhất chứa tất cả các điểm kiểm soát đó.

Chứng minh:

Bao lồi của các điểm kiểm soát cũng chính là tập hợp các tổ hợp lồi của các điểm kiểm soát.

Ta biểu diễn tổ hợp tuyến tính của các điểm P_k:

$$P(t) = \sum_{k=0}^{L} a_k.P_k \quad , \quad a_k \geq 0$$

Do P(t) là tổ hợp lồi của các điểm kiểm soát $\forall t \in [0,1]$ và $\sum_{k=0}^{L} B_k^L(t) = 1$

Nên đường cong Bezier sẽ nằm trong bao lồi của các điểm kiểm soát.

iv/ Độ chính xác tuyến tính:

Đường cong Bezier có thể trở thành một đường thẳng khi tất cả các điểm kiểm soát nằm trên một đường thẳng vì khi đó bao lồi của chúng là một đường thẳng nên đường Bezier bị kẹp vào bên trong bao lồi nên nó cũng trở thành đường thẳng.

v/ Đạo hàm của các đường Bezier:

Ta có: $(P(t))' = L.\sum_{k=0}^{L-1} \Delta P_k.B_k^{L-1}(t)$, $\Delta P_k = P_{k+1} - P_k$

Do đó, đạo hàm của đường cong Bezier là một đường cong Bezier khác được tạo ra từ các vector kiểm soát ΔP_k (Ta chỉ cần lấy các điểm kiểm soát gốc theo từng cặp để tạo ra các điểm kiểm soát cho (P(t))'.

4.1.5. Đánh giá các đường cong Bezier

Bằng các điểm kiểm soát, ta có thể tạo ra các dạng đường cong khác nhau bằng cách hiệu chỉnh các điểm kiểm soát cho tới khi tạo ra được một dạng đường cong mong muốn. Công việc này lặp đi lặp lại cho đến khi toàn bộ đường cong thỏa yêu cầu.

Tuy nhiên, khi ta thay đổi bất kỳ một điểm kiểm soát nào thì toàn bộ đường cong bị thay đổi theo. Nhưng trong thực tế, ta thường mong muốn chỉ thay đổi một ít về dạng đường cong ở gần khu vực đang hiệu chỉnh các điểm kiểm soát.

Tính cục bộ yếu của đường cong Bezier được biểu hiện qua các đa thức $B_k^L(t)$ đều khác 0 trên toàn khoảng [0,1]. Mặt khác đường cong p(t) lại là một tổ hợp tuyến tính của các điểm kiểm soát được gia trọng bởi các hàm $B_k^L(t)$ nên ta kết luận rằng mỗi điểm kiểm soát có ảnh hưởng đến đường cong ở tất cả các giá trị $t \in [0,1]$. Do đó, hiệu chỉnh bất kỳ một điểm kiểm soát nào cũng sẽ ảnh hưởng đến dạng của toàn thể đường cong.

Để giải quyết bài toán này, ta sử dụng một tập hợp các hàm trộn khác nhau. Các hàm trộn này có ***giá mang*** (***support***: khoảng mà trên đó hàm lấy giá trị khác 0) chỉ là một phần của khoảng [0,1]. Ngoài giá mang này chúng có giá trị là 0.

Thường ta chọn các hàm trộn là các đa thức trên các giá mang đó, các giá mang này kề nhau. Như vậy, các hàm trộn chính là một *tập các đa thức được định nghĩa trên những khoảng kề nhau* được nối lại với nhau để tạo nên một đường cong liên tục. Các đường cong kết quả được gọi là *đa thức riêng phần* hay từng phần (piecewise polynomial).

Ví dụ: ta định nghĩa hàm g(t) gồm 3 đa thức a(t), b(t), c(t) như sau:

$$g(t) = \begin{cases} a(t) = \frac{1}{2}t^2, \text{ có giá mang } [0,1] \\ b(t) = \frac{3}{4} - (t - \frac{3}{2})^2, \text{ có giá mang } [1,2] \\ c(t) = \frac{1}{2}(3-t)^2, \text{ có giá mang } [2,3] \end{cases}$$

Giá mang của g(t) là [0,3]

Các giá trị của t ứng với *các chỗ nối* của các đoạn gọi là *nút* (knut), chẳng hạn t=0,1,2,3 là bốn nút của g(t). Hơn nữa, tại các chỗ nối của đường cong g(t) là trơn, không bị gấp khúc. Do đó, ta gọi đó là hàm **Spline**.

Vậy, một hàm Spline cấp m là đa thức riêng phần cấp m có đạo hàm cấp m -1 liên tục ở mỗi nút.

Dựa trên tính chất của hàm Spline, ta có thể dùng nó như các hàm trộn để tạo ra đường cong p(t) dựa trên các điểm kiểm soát $P_0,...,P_L$. Khi đó:

$$P(t) = \sum_{k=0}^{L} P_k.g_k(t)$$

Tổng quát hóa, ta xây dựng một hàm p(t) với L+1 điểm kiểm soát như sau:

Với mỗi điểm kiểm soát P_k , ta có một hàm trộn tương ứng $R_k(t)$ và *tập các nút gọi là vector nút* $T=(t_0,t_1,...,t_n)$ với $t_i \in R$, $t_i \le t_{i+1}$. Khi đó:

$$P(t) = \sum_{k=0}^{L} P_k.R_k(t)$$

4.2. ĐƯỜNG CONG SPLINE VÀ B-SPLINE

4.2.1. Định nghĩa

Theo trên ta có:

$$P(t) = \sum_{k=0}^{L} P_k.R_k(t) \qquad (*)$$

trong đó

P_k với k=1..L là các điểm kiểm soát.

$R_k(t)$ là các hàm trộn liên tục trong mỗi đoạn con $[t_i , t_{i+1}]$và liên tục trên mỗi nút. Mỗi $R_k(t)$ là một đa thức riêng phần.

Do đó đường cong p(t) là tổng của các đa thức này, lấy trên các điểm kiểm soát.

Các đoạn đường cong riêng phần này gặp nhau ở các điểm nút và tạo cho đường cong trở nên liên tục. Ta gọi những đường cong như vậy là **SPLINE.**

Cho trước một vector nút thì có thể có nhiều họ hàm trộn được dùng để tạo ra một đường cong Spline có thể định nghĩa trên vector nút đó. Một họ các hàm như vậy được gọi là *cơ sở* cho các Spline.

Trong số các họ hàm này, có một cơ sở cụ thể mà các hàm trộn của nó có giá mang nhỏ nhất và nhờ vậy nó đem lại khả năng kiểm soát cục bộ lớn nhất. Đó là các **B-Spline**, với B viết tắt của chữ Basic (cơ sở).

Đối với các hàm B-Spline, mỗi đa thức riêng phần tạo ra nó ***có một cấp m*** nào đó. Do đó, thay vì dùng ký hiệu Rk(t) cho các hàm riêng phần này ta sẽ ký hiệu các hàm trộn này là $\mathbf{N_{k,m}(t)}$.

Do đó các đường cong B-Spline có thể biểu diễn lại:

$$\mathbf{P(t)} = \sum_{k=0}^{L} \mathbf{P_k}.\mathbf{N_{k,m}(t)}$$

TÓM LẠI

Để xây dựng các đường cong B-Spline ta cần có:

- Một véc tơ nút $T=(t_0, t_1, t_2, ...,t_{k+m-1})$.
- (L+1) điểm kiểm soát.
- Cấp m của các hàm B-Spline và công thức cơ bản cho hàm B-Spline $N_{k,m}(t)$ là:

$$N_{k,m}(t) = \left(\frac{t - t_k}{t_{k+m-1} - t_k}\right).N_{k,m-1}(t) + \left(\frac{t_{k+m} - t}{t_{k+m} - t_{k+1}}\right).N_{k+1,m-1}(t) \text{ với } k=0..L$$

Đây là một công thức đệ quy với $N_{k,L}(t) = \begin{cases} 1 & t_k \prec t \le t_{k+1} \\ 0 & \text{ng-îc l¹i} \end{cases}$

(Hàm hằng bằng 1 trên đoạn (t_k , t_{k+1}))

Đối với các mặt B-Spline, ta có công thức biểu diễn tương tự:

$$P(u,v) = \sum_{i=0}^{M} \sum_{k=0}^{L} P_{i,k}.N_{i,m}(u).N_{k,m}(v)$$

Nhận xét: C*ác đường Bezier là các đường B-Spline.*

Sau đây là chương trình vẽ đường Spline:

```
#include<conio.h>
#include<graphics.h>
const float t=0;
struct ToaDo
{
  float x,y;
};
void thietlapdohoa()
{
  int gd=0,gm;
  initgraph(&gd,&gm,"d:\\tc\\bgi ");
}
int round(float x)
{
```

```
  if(x>0) return int(x+0.5);
  else return int(x-0.5);
}
void tinh(float u,ToaDo p[],int k, ToaDo &pu)
{
  float u2,u3,s;
  u2=u*u;
  u3=u2*u;
  s=(1-t)/2.0;
  pu.x=p[k-1].x*(-s*u3+2.0*s*u2-s*u)
       +p[k].x*((2.0-s)*u3+(s-3.0)*u2+1.0)
       +p[k+1].x*((s-2.0)*u3+(3.0-2.0*s)*u2+s*u)
       +p[k+2].x*(s*u3-s*u2);
  pu.y=p[k-1].y*(-s*u3+2.0*s*u2-s*u)
       +p[k].y*((2.0-s)*u3+(s-3.0)*u2+1.0)
       +p[k+1].y*((s-2.0)*u3+(3.0-2.0*s)*u2+s*u)
       +p[k+2].y*(s*u3-s*u2);
}
void khoitao(int &n,ToaDo p[])
{
  n=12;
  p[1].x=10;  p[1].y=100;
  p[2].x=40;  p[2].y=120;
  p[3].x=80;  p[3].y=100;
  p[4].x=120; p[4].y=220;
  p[5].x=160; p[5].y=100;
  p[6].x=200; p[6].y=120;
  p[7].x=240; p[7].y=20;
  p[8].x=280; p[8].y=120;
  p[9].x=320; p[9].y=100;
  p[10].x=360;p[10].y=120;
  p[11].x=400;p[11].y=100;
  p[12].x=440;p[12].y=300;
}
void Spline(int n,ToaDo p[])
{
  float u,du=1.0/20;
  ToaDo pu;
  int batdau=1;
  for(int i=1;i<=n;i++)
     bar(round(p[i].x)-3,round(p[i].y)-3,
```

```
            round(p[i].x)+3,round(p[i].y)+3);
    for (int k=1;k<n;k++)
    {
      u=0;
      do
        {
         tinh(u,p,k,pu);
         if (batdau)
         {
              moveto(round(pu.x),round(pu.y));
              batdau=0;
         }
         else
              lineto(round(pu.x),round(pu.y));
         u=u+du;
        }
      while(u<1);
    }
}
int main()
{
  ToaDo p[100];
  int n;
  khoitao(n,p);
  thietlapdohoa();
  setcolor(RED);
  Spline(n,p);
  getch();
  closegraph();
}
```

4.2.2. Các tính chất hữu ích trong việc thiết kế các đường cong B-Spline

i/ Các đường B-Spline cấp m là các đa thức riêng phần cấp m. Chúng là các Spline do chúng có m-2 cấp đạo hàm liên tục ở mọi điểm trong giá mang của chúng.

Các hàm B-Spline cấp m tạo thành **một cơ sở** cho bất kỳ Spline nào có **cùng cấp** được định nghĩa trên **cùng các nút**. Các Spline có thể được biểu diễn như một tổ hợp tuyến tính của các B-Spline.

ii/ Hàm trộn B-Spline $N_{k,m}(t)$ bắt đầu ở tk và kết thúc ở t_{k+m} . Giá mang của nó là $[t_k,t_{k+m}]$. Giá mang của họ các hàm $N_{k,m}(t)$ với k=0,...L là khoảng $[t_0,t_{m+L}]$.

iii/ Một đường cong B-Spline đóng dựa trên L+1 điểm kiểm soát có thể được tạo ra bằng cách dùng phương trình đường B-Spline tuần hoàn sau:

$$P(t) = \sum_{k=0}^{L} P_k.N_{0,m}((t-k) \bmod (L+1))$$

Với giả thiết các nút cách đều nhau trong định nghĩa của hàm $N_{0,m}(...)$.

iv/ Nếu dùng vector chuẩn thì đường cong B-Spline sẽ nội suy các điểm kiểm soát đầu tiên và cuối cùng. Các hướng khởi đầu và kết thúc của đường cong đó sẽ nằm dọc theo các cạnh đầu tiên và cuối cùng của đa giác kiểm soát.

v/ Mỗi hàm B-Spline $N_{k,m}(t)$ là không âm $\forall t$, và tổng các họ hàm bằng 1:

$$\sum_{k=0}^{L} N_{k,m}(t) = 1 \qquad \forall t \in [t_0 , t_{m+L}]$$

vi/ Các đường cong dựa trên các B-Spline là **bất biến Affine**. Do đó, để biến đổi một đường cong B-Spline, chỉ cần biến đổi các điểm kiểm soát, sau đó khởi tạo lại đường cong từ các điểm kiểm soát đã được biến đổi này.

vii/ Một đường cong B-Spline sẽ nằm trong bao lồi của các điểm kiểm soát

Mạnh hơn: Ở bất kỳ t nào, chỉ có m hàm B-Spline là khác 0. Vì vậy, ở mỗi t đường cong phải nằm trong bao lồi của hầu hết m điểm kiểm soát kích hoạt kế nhau. (Các điểm kiểm soát kích hoạt là các điểm mà tại đó hàm B-Spline khác 0)

viii/ Độ chính xác tuyến tính của đường cong B-Spline: Nếu m điểm kiểm soát kề nhau là tuyến tính cùng nhau thì bao lồi của chúng là một đường thẳng. Do đó đường cong cũng sẽ trở thành đường thẳng.

ix/ Tính chất giảm độ biến thiên: Số giao điểm giữa đường cong B-Spline với bất kỳ một mặt phẳng nào (nếu có) luôn luôn nhỏ hơn số giao điểm (nếu có) giữa đa giác kiểm soát của nó với mặt phẳng đó.

4.2.3. Thiết kế các mặt Bezier và B-Spline

Ta có thể dùng các hàm trộn Bezier và B-Spline để mô tả và vẽ các mặt cong. Đối với các mặt cong, ta biểu diễn chúng dưới dạng tham số qua một hàm vector với 2 tham số là u, v. Dạng tổng quát của một mặt cong là:

$$p(u,v) = (X(u,v), Y(u,v), Z(u,v))$$

$$\Leftrightarrow \quad (u,v) = X(u,v).i + Y(u,v).j + Z(u,v).k$$

Khi u, v biến thiên trên một khoảng nào đó thì các hàm X(u,v), Y(u,v) và Z(u,v) thay đổi giá trị, do đó làm cho vị trí của p(u,v) thay đổi trong không gian 3 chiều.

Chúng ta sẽ không biểu diễn các mặt qua các hàm toán học tường minh mà sẽ biểu diễn chúng qua các điểm kiểm soát.

Ví dụ: $p(u,v) = (1-v).((1-u).P_{00} + u.P_{10}) + v.((1-u).P_{01} + u.P_{11})$ dùng 4 điểm kiểm soát ở 4 góc là Pij với các hàm trộn là tuyến tính theo u, v.

CHƯƠNG 5: CÁC PHÉP BIẾN ĐỔI TRONG MẶT PHẲNG

5.1. CƠ SỞ TOÁN HỌC

Phép biến đổi Affine 2D biến điểm P(x,y) thành điểm P'(x',y') theo hệ phương trình sau:

$$\begin{cases} x' = Ax + Cy + E \\ y' = Bx + Dy + F \end{cases} \quad \textbf{(*)}$$

Dưới dạng ma trận, hệ này có dạng:

$$\textbf{(x' y')} = \textbf{(x y)}.\begin{pmatrix} A & B \\ C & D \end{pmatrix} + \textbf{(E F)} \quad \textbf{(1)}$$

Hay viết gọn hơn:

$$\textbf{X' = X.M + tr} \quad \textbf{(2)}$$

với X'=(x',y'), X=(x,y), tr=(E,F) - vector tịnh tiến và $M = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$ - ma trận biến đổi.

Chú ý: ***Từ nay về sau, để thuận tiện cho việc biểu diễn dưới dạng ma trận, tất cả các tọa độ và ma trận đều được biểu diễn dưới dạng Homogen.***

Ví dụ: (x y) → (x y 1)

$$\begin{pmatrix} A & B \\ C & D \end{pmatrix} \rightarrow \begin{pmatrix} A & B & 0 \\ C & D & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Như vậy (*) có thể viết lại:

$$\textbf{(x' y' 1)} = \textbf{(x y 1)}.\begin{pmatrix} A & B & 0 \\ C & D & 0 \\ E & F & 1 \end{pmatrix} \quad (**)$$

5.2. CÁC PHÉP BIẾN ĐỔI CƠ BẢN

5.2.1. Phép tịnh tiến

Cho phép di chuyển đối tương theo véc tơ (E,F).

$$M = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ E & F & 1 \end{pmatrix} \Rightarrow \begin{cases} x' = x + E \\ y' = y + F \end{cases}$$

5.2.2. Phép đồng dạng

$$M = \begin{pmatrix} A & 0 & 0 \\ 0 & D & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \Rightarrow \quad \begin{cases} x' = A.x \\ y' = D.y \end{cases}$$

Cho phép phóng to hay thu nhỏ đối tượng theo một hoặc hai chiều.

5.2.3. Phép đối xứng

Đây là trường hợp đặc biệt của phép đồng dạng.

$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ đối xứng qua Oy.

$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ đối xứng qua Ox.

$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$ đối xứng qua gốc tọa độ.

5.2.4. Phép quay

$$M = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \Rightarrow \begin{cases} x' = x.\cos(\alpha) - y.\sin(\alpha) \\ y' = x.\sin(\alpha) + y.\cos(\alpha) \end{cases}$$

Chú ý:

- Tâm của phép quay được xét ở đây là gốc tọa độ.
- Định thức của ma trận phép quay luôn luôn bằng 1.

5.2.5. Phép biến dạng

$$M = \begin{pmatrix} 1 & g & 0 \\ h & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \Rightarrow \begin{cases} x' = x + h.y \\ y' = g.x + y \end{cases}$$

Trong đó:

g = 0: biến dạng theo trục x.

h = 0: biến dạng theo trục y.

5.2.6. Hợp của các phép biến đổi

Có ma trận là tích của các ma trận biến đổi.

Ví dụ 1: Phép quay quanh một điểm bất kỳ trong mặt phẳng có thể thực hiện bởi tích của các phép biến đổi sau:

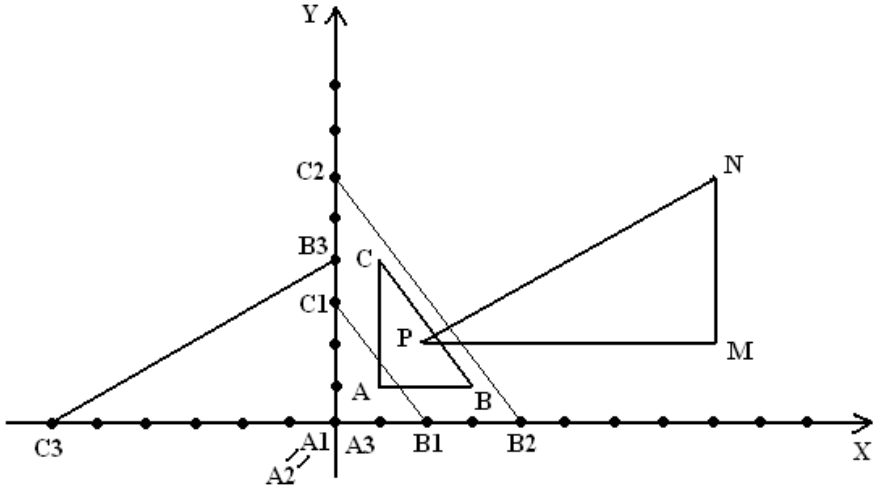
- Phép tịnh tiến tâm quay đến gốc tọa độ.
- Phép quay với góc đã cho.

° Phép tịnh tiến kết quả về tâm quay ban đầu.

Như vậy, ma trận của phép quay quanh một điểm bất kỳ được thực hiện bởi tích của ba phép biến đổi sau:

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -M & -N & 1 \end{pmatrix} . \begin{pmatrix} Cos(\alpha) & Sin(\alpha) & 0 \\ -Sin(\alpha) & Cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} . \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ M & N & 1 \end{pmatrix}$$

Ví dụ 2: Cho tam giác ABC với A(1,1) B(3,1) C(1,4). Tịnh tiến tam giác ABC theo vector (-1,-1) để được tam giác $A_1B_1C_1$. Phóng to tam giác $A_1B_1C_1$ theo hệ số 2 để được tam giác $A_2B_2C_2$. Quay tam giác $A_2B_2C_2$ một góc 90^0 để được tam giác $A_3B_3C_3$. Tịnh tiến tam giác $A_3B_3C_3$ theo vector (8,2) để được tam giác MNP. Vẽ hình và tìm ma trận biến đổi tam giác ABC thành tam giác MNP.



Hình 5.1

Bước 1: Ma trận tịnh tiến tam giác ABC theo vector (-1,-1) để được tam giác $A_1B_1C_1$.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -1 & -1 & 1 \end{pmatrix}$$

Bước 2: Ma trận phóng to tam giác $A_1B_1C_1$ theo hệ số 2 để được tam giác $A_2B_2C_2$.

$$\begin{pmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Bước 3: Ma trận quay tam giác $A_2B_2C_2$ một góc 90^0 để được tam giác $A_3B_3C_3$.

$$\begin{pmatrix} Cos(90^0) & Sin(90^0) & 0 \\ -Sin(90^0) & Cos(90^0) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Bước 4: Ma trận tịnh tiến tam giác $A_3B_3C_3$ theo vector (8,2) để được tam giác MNP.

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 8 & 2 & 1 \end{pmatrix}$$

Vậy, ma trận biến đổi tam giác ABC thành tam giác MNP là:

$$\begin{pmatrix}1&0&0\\0&1&0\\-1&-1&1\end{pmatrix}.\begin{pmatrix}2&0&0\\0&2&0\\0&0&1\end{pmatrix}.\begin{pmatrix}Cos(90^0)&Sin(90^0)&0\\-Sin(90^0)&Cos(90^0)&0\\0&0&1\end{pmatrix}.\begin{pmatrix}1&0&0\\0&1&0\\8&2&1\end{pmatrix}=\begin{pmatrix}0&2&0\\-2&0&0\\10&0&1\end{pmatrix}$$

5.3. CÁC VÍ DỤ MINH HỌA

Ví dụ 1: Viết chương trình vẽ và điều khiển các phím ←→↑↓ để tịnh tiến hình chữ nhật trên màn hình.

```
#include<conio.h>
#include<math.h>
#include<graphics.h>
#define Step 5;
int x0,y0;
void thietlapdohoa()
{
  int gd=0,gm;
  initgraph(&gd,&gm,"");
}
void vehinh(int x0,int y0)
{
 rectangle(x0,y0,x0+50,y0+30);
}
int main()
{
  char ch;
  thietlapdohoa();
  x0=getmaxx()/2;
  y0=getmaxy()/2;
  setwritemode(XOR_PUT);
  vehinh(x0,y0);
  do
  { ch=getch();
    switch(ch)
    {
       case 75: //phím ←
          vehinh(x0,y0);
                x0-=Step;
                vehinh(x0,y0);
                break;
       case 77: //phím →
          vehinh(x0,y0);
                x0+=Step;
```

```
                    vehinh(x0,y0);
                    break;
          case 72: //phím ↑
              vehinh(x0,y0);
                    y0-=Step;
                    vehinh(x0,y0);
                    break;
          case 80: //phím ↓
              vehinh(x0,y0);
                    y0+=Step;
                    vehinh(x0,y0);
                    break;
       }
      } while(ch!=27);
    closegraph();
}
```

Ví dụ 2: Viết chương trình điều khiển các phím ←→ để quay tam giác quanh gốc tọa độ.

```
#include<conio.h>
#include<math.h>
#include<graphics.h>
#define PI 3.1416
struct ToaDo
{
 float x,y;
};
int x0,y0;
float k;
int round(float x)
{
 if (x>0) return int (x+0.5);
 else return int (x-0.5);
}
void thietlapdohoa()
{
  int mh=0,mode=0;
  initgraph(&mh,&mode,"");
}
void vetruc()
{
  line(0,y0,2*x0,y0);
```

```
  line(x0,0,x0,2*x0);
}
void vehinh(ToaDo p1,ToaDo p2,ToaDo p3)
{
 line(x0+round(p1.x*k),y0-round(p1.y*k),
        x0+round(p2.x*k),y0-round(p2.y*k));
 line(x0+round(p2.x*k),y0-round(p2.y*k),
        x0+round(p3.x*k),y0-round(p3.y*k));
 line(x0+round(p3.x*k),y0-round(p3.y*k),
        x0+round(p1.x*k),y0-round(p1.y*k));
}
void QuayDiem(ToaDo p,float alpha,ToaDo &pmoi)
{
  pmoi.x=p.x*cos(alpha)-p.y*sin(alpha);
  pmoi.y=p.x*sin(alpha)+p.y*cos(alpha);
}
void QuayHinh(ToaDo p1,ToaDo p2,ToaDo p3,float alpha,
              ToaDo &p1moi,ToaDo &p2moi,ToaDo &p3moi)
{
  QuayDiem(p1,alpha,p1moi);
  QuayDiem(p2,alpha,p2moi);
  QuayDiem(p3,alpha,p3moi);
}
int main()
{
  float goc,alpha;
  char ch;
  ToaDo p,pp,ppp,p1,p2,p3;
  thietlapdohoa();
  k=getmaxy()/30;
  x0=getmaxx()/2;
  y0=getmaxy()/2;
  vetruc();
  p.x=5;p.y=3;pp.x=2;pp.y=6;ppp.x=6;ppp.y=-4;
  p1=p; p2=pp; p3=ppp;
  goc=PI/180; alpha=0;
  setwritemode(XOR_PUT);
  vehinh(p1,p2,p3);
  do
  { ch=getch();
    switch(ch)
```

```
        {
      case 75://phím ←
             vehinh(p1,p2,p3);
                   alpha=alpha+goc;
                   QuayHinh(p,pp,ppp,alpha,p1,p2,p3);
                   vehinh(p1,p2,p3);
                   break;
      case 77://phím →
             vehinh(p1,p2,p3);
                 alpha=alpha-goc;
                   QuayHinh(p,pp,ppp,alpha,p1,p2,p3);
                   vehinh(p1,p2,p3);
                   break;
        }
      } while(ch!=27);
    closegraph();
   }
```

BÀI TẬP

1. Cho 3 tam giác sau:

ABC với	A(1,1)	B(3,1)	C(1,4)
EFG với	E(4,1)	F(6,1)	G(4,4)
MNP với	M(10,1)	N(10,3)	P(7,1)

   a. Tìm ma trận biến đổi tam giác ABC thành tam giác EFG.

   b. Tìm ma trận biến đổi tam giác ABC thành tam giác MNP.

2. Cài đặt thuật toán xén một đoạn thẳng vào một hình chữ nhật có cạnh không song song với trục tọa độ.

3. Viết chương trình vẽ một Ellipse có các trục không song song với các trục tọa độ. Mở rộng: hãy mô phỏng quá trình quay của một Ellipse xung quanh tâm của nó.

4. Viết chương trình mô phỏng chuyển động của trái đất xung quanh mặt trời đồng thời mô tả chuyển động của mặt trăng xung quanh trái đất.

5. Viết chương trình vẽ đồng hồ đang hoạt động.

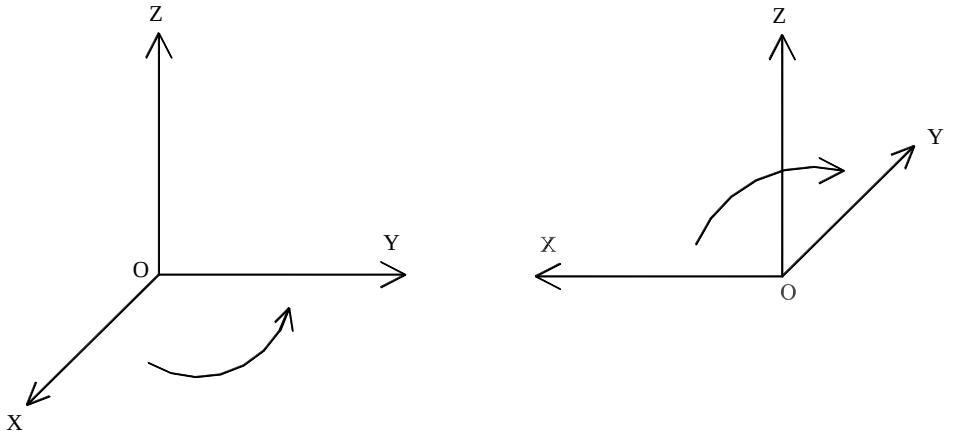
6. Viết chương trình trò chơi "rắn săn mồi".

CHƯƠNG 6: VẼ CÁC ĐỐI TƯỢNG BA CHIỀU

6.1. CÁC PHÉP BIẾN ĐỔI TRONG KHÔNG GIAN

6.1.1. Các hệ trục tọa độ

Để định vị một điểm trong không gian, ta có thể chọn nhiều hệ trục tọa độ:

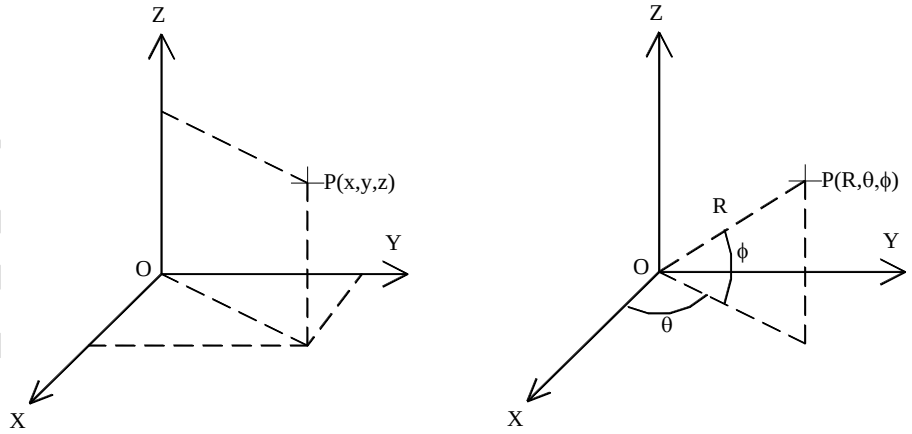


Hệ trực tiếp Hệ gián tiếp

Hình 6.1

- *Hệ tọa độ trực tiếp* : nếu tay phải cầm trục Z sao cho ngón cái hướng theo chiều dương của trục Z thì bốn ngón còn lại sẽ quay từ trục X sang trục Y (Qui tắc bàn tay phải).
- *Hệ tọa độ gián tiếp* : ngược lại (Qui tắc bàn tay trái).

Thông thường, ta luôn định vị một điểm trong không gian thông qua hệ trực tiếp.



Hệ tọa độ Descarter Hệ cầu

Hình 6.2

Hệ tọa độ trực tiếp được chia ra làm 2 loại: Hệ tọa độ Descarter và hệ cầu (Hình 6.2).

Ta có công thức chuyển đổi tọa độ giữa hai hệ này như sau:

x = R.Cos(θ).Cos(Φ)

y = R.Sin(θ).Cos(Φ)

$\mathbf{z = R.Sin(\Phi)}$

$\mathbf{R^2 = x^2 + y^2 + z^2}$

Để thuận tiện cho việc tính toán, tất cả các điểm trong không gian đều được mô tả dưới dạng ma trận 1x4, tức là (x,y,z,1). Vì vậy, tất cả các phép biến đổi trong không gian đều được biểu diễn bởi các ma trận vuông 4x4 (Ma trận Homogen).

6.1.2. Các phép biến đổi cơ bản

Phép biến đổi Affine 3D có dạng: **X'=X.M + tr**

với X'=(x',y',z'), X=(x,y,z), M - ma trận biến đổi, tr=(E,F,G) – véc tơ tịnh tiến.

6.1.2.1. Phép đồng dạng

$$M = \begin{pmatrix} A & 0 & 0 & 0 \\ 0 & B & 0 & 0 \\ 0 & 0 & C & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad \Leftrightarrow \quad \begin{cases} x' = A.x \\ y' = B.y \\ z' = C.z \end{cases}$$

6.1.2.2. Phép đối xứng

$$M_z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ đối xứng qua mặt phẳng (XY).

$$M_y = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ đối xứng qua mặt phẳng (XZ)

$$M_x = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ đối xứng qua mặt phẳng (YZ)

6.1.2.3. Phép tịnh tiến

$$M = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ M & N & P & 1 \end{pmatrix} \quad \Leftrightarrow \quad \begin{cases} x' = x + M \\ y' = y + N \\ z' = z + P \end{cases}$$

6.1.2.4. Phép quay

Ta nhận thấy rằng, nếu phép quay quay quanh một trục nào đó thì tọa độ của vật thể tại trục đó sẽ không thay đổi. Do đó, ta có ma trận của các phép quay như sau:

- Ma trận quay quanh trục Z: $R_Z = \begin{pmatrix} Cos(\theta) & Sin(\theta) & 0 & 0 \\ -Sin(\theta) & Cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

- Ma trận quay quanh trục X: $R_X = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos(\theta) & Sin(\theta) & 0 \\ 0 & -Sin(\theta) & Cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

- Ma trận quay quanh trục Y: $R_Y = \begin{pmatrix} Cos(\theta) & 0 & Sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -Sin(\theta) & 0 & Cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$

Chú ý: Tích của 2 ma trận nói chung không giao hoán nên kết quả của 2 phép quay liên tiếp tùy thuộc vào thứ tự thực hiện tích số.

Ví dụ: $R_X.R_Y \neq R_Y.R_X$

6.1.3. Ma trận nghịch đảo

Định nghĩa: Hai ma trận được gọi là nghịch đảo của nhau nếu tích của chúng bằng ma trận đơn vị.

Ký hiệu: Ma trận nghịch đảo của ma trận M là M^{-1}.

Ví dụ:

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 3 \\ 1 & 2 & 4 \end{pmatrix} . \begin{pmatrix} 6 & -2 & -3 \\ -1 & 1 & 0 \\ -1 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Người ta chứng minh được rằng: Tất cả các ma trận của các phép biến đổi đã nêu ở trên đều có ma trận nghịch đảo, cụ thể:

- Ma trận nghịch đảo của phép tịnh tiến có được bằng cách thay M, N, P bằng -M, -N, -P.
- Ma trận nghịch đảo của phép thay đổi tỉ lệ có được bằng cách thay A, B, C bằng 1/A, 1/B, 1/C.
- Ma trận nghịch đảo của phép quay có được bằng cách thay góc θ bằng $-\theta$.

6.2. PHÉP CHIẾU VẬT THỂ TRONG KHÔNG GIAN LÊN MẶT PHẲNG

6.2.1. Phép chiếu phối cảnh

Phép chiếu phối cảnh (perspective) cho hình ảnh giống như khi nhìn vật thể.

Để tìm hình chiếu P'(x',y') của P(x,y,z), nối P với mắt (**tâm chiếu**). Giao điểm của đường này với mặt quan sát chính là P' (Hình 6.3).

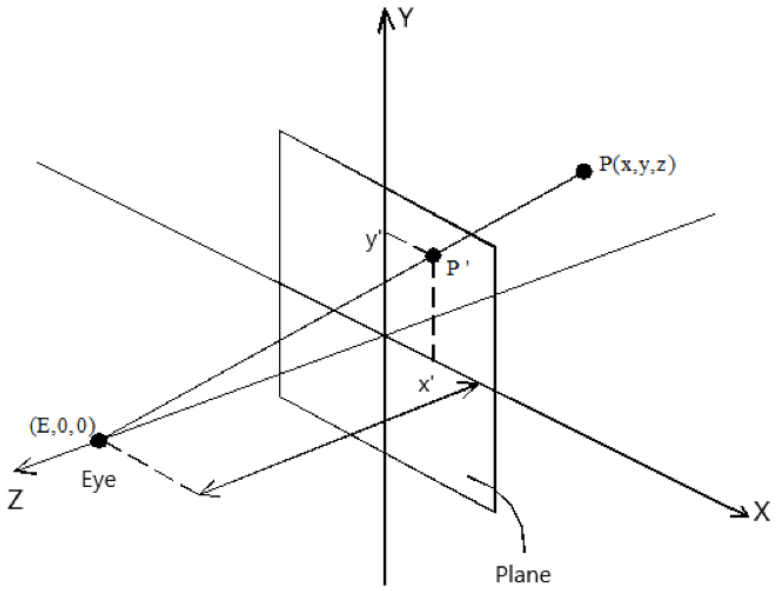
Giả sử P nằm phía trước của mắt, tức là P.z < E.

Phương trình của tia đi qua mắt và P là: r(t) = (0,0,E).(1-t) + (x,y,z).t (*)

Giao điểm với mặt phẳng quan sát có thành phần z' = 0.

Do thành phần z' của tia r là E.(1-t) + z.t = 0 nên t = $\frac{1}{1-z/E}$. Thay t vào (*) ta tính được:

$$x' = \frac{x}{1-z/E} \quad \text{và} \quad y' = \frac{y}{1-z/E}$$



Hình 6.3

NHẬN XÉT

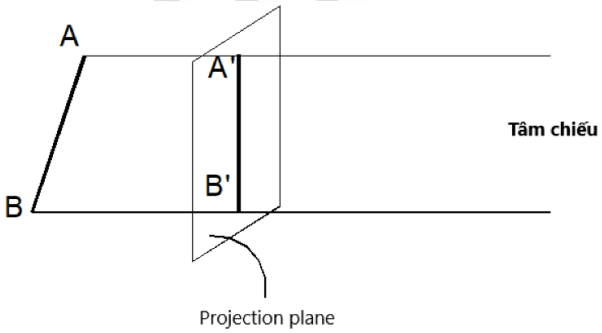
i/ Phép chiếu phối cảnh không giữ nguyên hình dạng của vật thể.

ii/ Chỉ có những đường thẳng song song với mặt phẳng chiếu thì mới song song với nhau.

6.2.2. Phép chiếu song song

Phép chiếu song song (parallel) có tâm chiếu đặt ở vô cực nên y'=y, z'=z (Hình 6.4).

⇒ Tính song song được bảo toàn.



Hình 6.4

6.3. CÔNG THỨC CỦA CÁC PHÉP CHIẾU LÊN MÀN HÌNH

Khi quan sát một vật thể trong không gian dưới một góc độ nào đó, có hai khả năng chọn lựa:

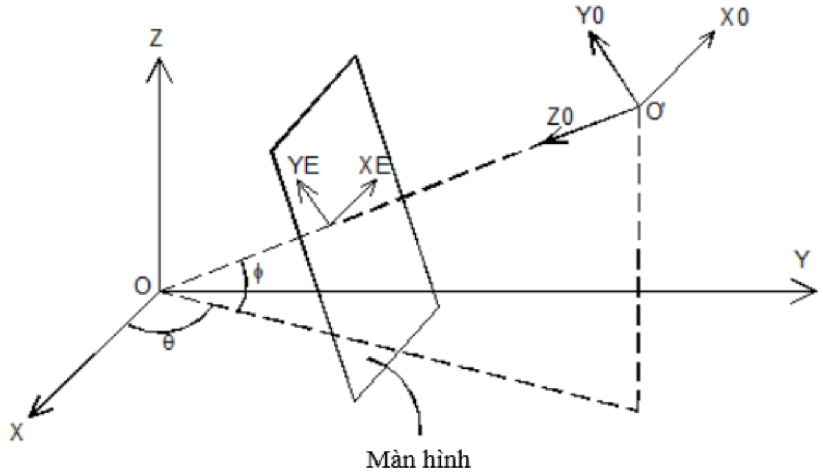
- Điểm nhìn (màn hình) đứng yên và vật thể di động.
- Vật thể đứng yên và điểm nhìn sẽ được bố trí thích hợp.

Ta thường chọn giải pháp thứ hai vì nó sát với thực tế hơn.

Khi quan sát một vật thể bất kỳ trong không gian, ta phải tuân thủ các nguyên tắc sau (Hình 6.5):

- Vật thể phải được chiếu lên một *hệ trực tiếp* (O,X,Y,Z).
- Con mắt phải nằm ở gốc của một hệ gián tiếp thứ hai (O',X_0,Y_0,Z_0)

- Màn hình là mặt phẳng vuông góc với đường thẳng OO'.
- Trục Z_0 của hệ quan sát chỉ đến gốc O.



Hình 6.5

Nếu dùng hệ tọa độ cầu để định vị mắt của người quan sát thì dễ dàng thay đổi góc ngắm bằng cách thay đổi góc θ và Φ.

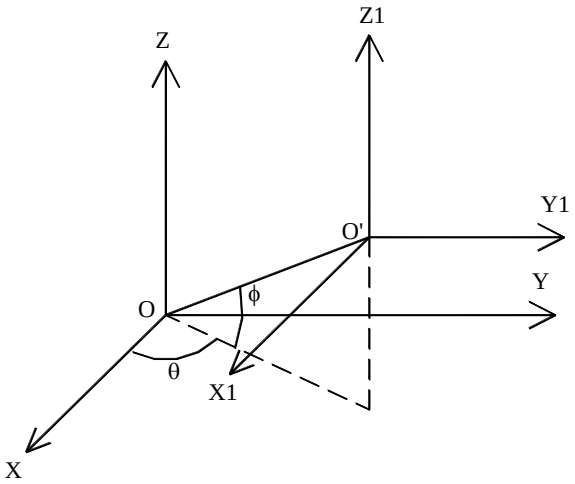
Bây giờ, ta khảo sát phép biến đổi mà vật thể (X,Y,Z) phải chịu để cho nó trùng với hệ quan sát (X_0,Y_0,Z_0) và cuối cùng tạo ra hệ tọa độ màn hình (x_E,y_E).

Bước 1: Tịnh tiến gốc O thành O' (Hình 6.6).

Ma trận của phép tịnh tiến (Lấy nghịch đảo):

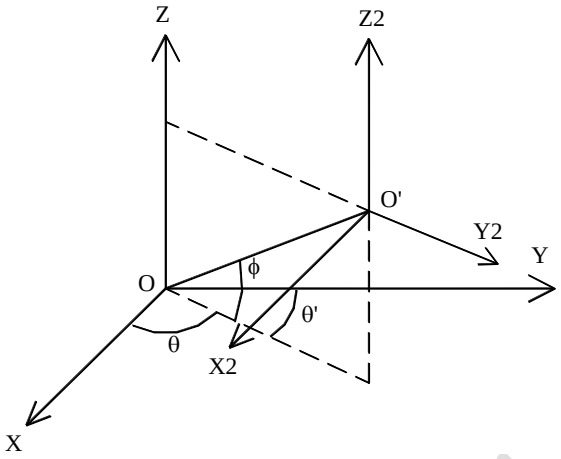
$$A=\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -M & -N & -P & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ -R.Cos(\theta).Cos(\phi) & -R.Sin(\theta).Cos(\phi) & -R.Sin(\phi) & 1 \end{pmatrix}$$

và hệ (X,Y,Z) biến đổi thành hệ (X_1,Y_1,Z_1).



Hình 6.6

Bước 2: Quay hệ (X_1,Y_1,Z_1) một góc **-θ'** (**θ'=90^0 - θ**) quanh trục Z_1 theo chiều kim đồng hồ. Phép quay này làm cho trục âm của Y_1 cắt trục Z (Hình 6.7).



Hình 6.7

Gọi Rz là ma trận tổng quát của phép quay quanh trục Z. Vì đây là phép quay hệ trục nên phải dùng ma trận nghịch đảo R^{-1}_z.

$$Rz = \begin{pmatrix} Cos(\alpha) & Sin(\alpha) & 0 & 0 \\ -Sin(\alpha) & Cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{R}^{-1}_z = \begin{pmatrix} Cos(\alpha) & -Sin(\alpha) & 0 & 0 \\ Sin(\alpha) & Cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

thay $\alpha = -\theta'$, ta có:

$Sin(-\theta') = -Sin(\theta') = -Sin(90^0 - \theta) = \mathbf{-Cos(\theta)}$

$Cos(-\theta') = Cos(\theta') = Cos(90^0-\theta) = Sin(\theta)$

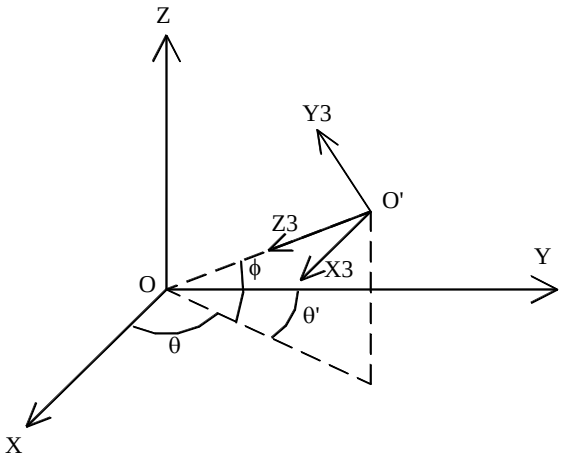
Nên ma trận của phép quay tìm được sẽ có dạng:

$$\mathbf{B} = \begin{pmatrix} Sin(\theta) & Cos(\theta) & 0 & 0 \\ -Cos(\theta) & Sin(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ và hệ (X_1,Y_1,Z_1) biến đổi thành hệ (X_2,Y_2,Z_2).

Bước 3: Quay hệ (X_2,Y_2,Z_2) một góc $90^0 + \mathbf{\Phi}$ quanh trục X_2. Phép biến đổi này sẽ làm cho trục Z_2 hướng đến gốc O (Hình 6.8).

Ta có:

$$Rx = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos(a) & Sin(a) & 0 \\ 0 & -Sin(a) & Cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \qquad R^{-1}_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos(a) & -Sin(a) & 0 \\ 0 & Sin(a) & Cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$



Hình 6.8

Thay góc $\alpha = 90^0 + \Phi$, ta có:

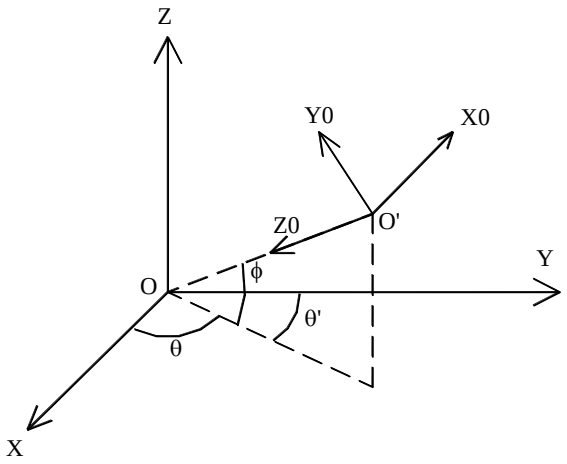
$Cos(90^0 + \Phi) = -Sin(\Phi)$ và $Sin(90^0 + \Phi) = Cos(\Phi)$

nên ma trận tìm được có dạng:

$$C = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & -Sin(\phi) & -Cos(\phi) & 0 \\ 0 & Cos(\phi) & -Sin(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Lúc này, hệ (X_2,Y_2,Z_2) biến đổi thành hệ (X_3,Y_3,Z_3).

Bước 4: Biến đổi hệ trực tiếp (X_3,Y_3,Z_3) thành hệ gián tiếp bằng cách đổi hướng của trục X_3. Ta nhận được ma trận:



Hình 6.9

$D = \begin{pmatrix} -1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$ và hệ (X_3,Y_3,Z_3) biến đổi thành hệ (X_0,Y_0,Z_0) (Hình 6.9).

TÓM LẠI

Các điểm trong không gian sẽ nhận trong hệ quan sát một tọa độ có dạng:

$$\mathbf{(x_0\ y_0\ z_0\ 1) = (x\ y\ z\ 1).A.B.C.D}$$

Gọi T = A.B.C.D, ta tính được:

$$\mathbf{T} = \begin{pmatrix} -\sin(\theta) & -Cos(\theta).Sin(\phi) & -Cos(\theta).Cos(\phi) & 0 \\ Cos(\theta) & -Sin(\theta).Sin(\phi) & -Sin(\theta).Cos(\phi) & 0 \\ 0 & Cos(\phi) & -Sin(\phi) & 0 \\ 0 & 0 & R & 1 \end{pmatrix}$$

Ta có:

$$(x_0, y_0, z_0, 1) = (x\ y\ z\ 1).T$$

hay:

$$x_0 = -x.Sin(\theta) + y.Cos(\theta)$$

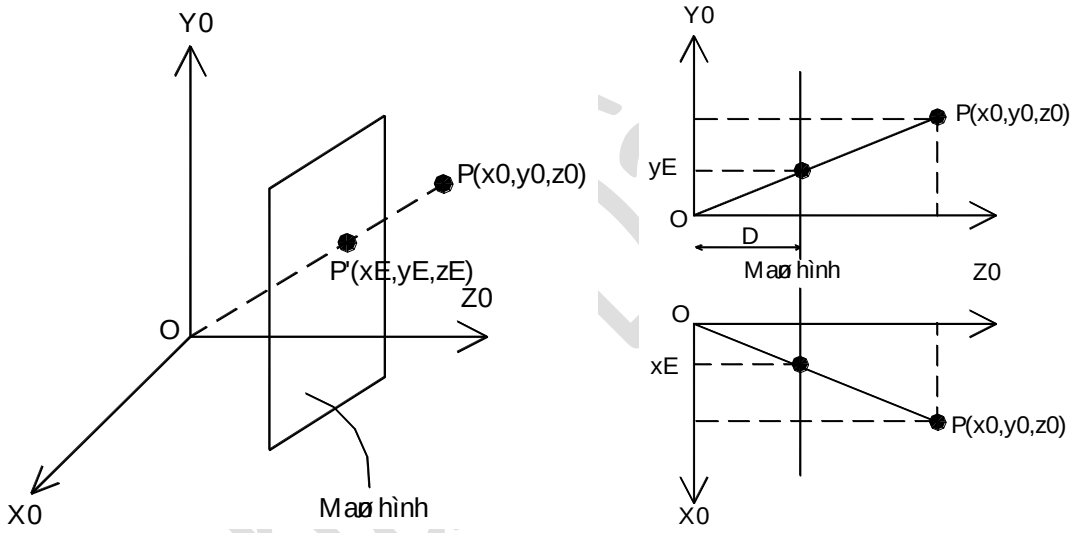
$$y_0 = -x.Cos(\theta).Sin(\Phi) - y.Sin(\theta).Sin(\Phi) + z.Cos(\Phi)$$

$$z_0 = -x.Cos(\theta).Cos(\Phi) - y.Sin(\theta).Cos(\Phi) - z.Sin(\Phi) + R$$

Cuối cùng, ta chiếu ảnh của hệ quan sát lên màn hình.

1. Phép chiếu phối cảnh

Cho điểm P(x,y,z) và hình chiếu P'(x0,y0,z0) của nó trên mặt phẳng.



Hình 6.10

Gọi D là khoảng cánh từ mặt phẳng đến mắt (gốc tọa độ). (Hình 6.10)

Xét các tam giác đồng dạng, ta có:

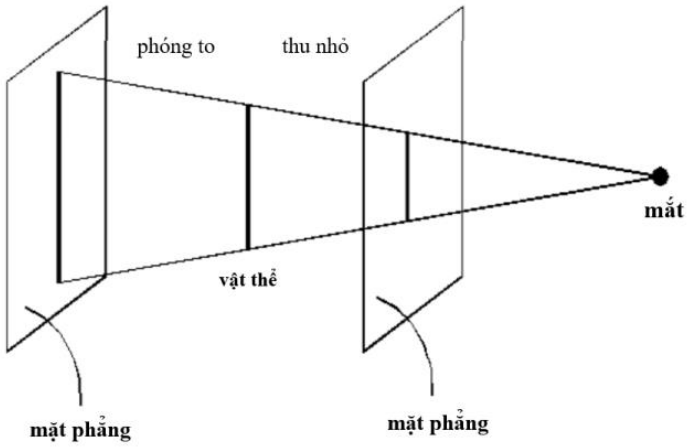
$x_E/D = x_0/z_0$ và $y_E/D = y_0/z_0$

$\Rightarrow$ $\boxed{x_E = D.x_0/z_0}$ và $\boxed{y_E = D.y_0/z_0}$

2. Phép chiếu song song

Tọa độ quan sát (x_0,y_0,z_0) và tọa độ màn hình thỏa mãn công thức:

$$x_E = x_0 \text{ và } y_E = y_0$$



Hình 5.11

KẾT LUẬN

Có 4 giá trị ảnh hưởng đến phép chiếu vật thể 3D là: các góc θ , Φ , khoảng cách R từ O đến O' và khoảng cách D từ O' đến mặt phẳng quan sát.

Cụ thể:

- Tăng giảm θ sẽ quay vật thể trong mặt phẳng (XY).
- Tăng giảm Φ sẽ quay vật thể lên xuống.
- Tăng giảm R để quan sát vật từ xa hay gần.
- Tăng giảm D để phóng to hay thu nhỏ ảnh.

6.4. PHỤ LỤC

Tạo thư viện DOHOA3D (DOHOA3D.H).

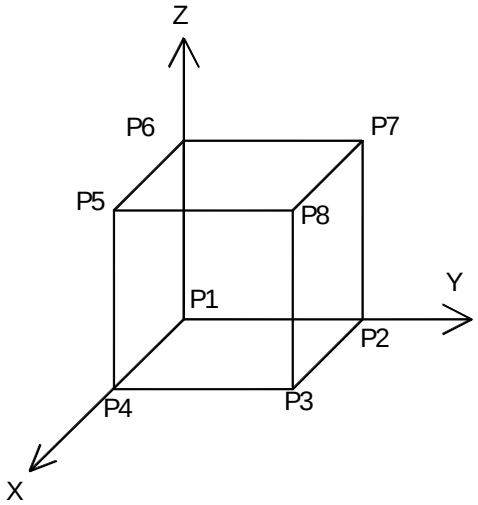
```
#ifndef DoHoa3D_h
#define DoHoa3D_h
#include<math.h>
#define IncAng 5
struct ToaDo3D
{
  float x,y,z;
};
struct ToaDo2D
{
  int x,y;
};
enum PhepChieu{PhoiCanh,SongSong};
float R,D,theta,phi;
float temp1,temp2,temp3,temp4;
float temp5,temp6,temp7,temp8 ;
char  ch;
ToaDo2D PC,PE;
ToaDo3D Obs;
```

```
PhepChieu project;
float Xproj,Yproj;
void ThietLapDoHoa()
{
 int gd=0,gm;
 initgraph(&gd,&gm,"D:\\TC\\BGI");
}
int round(float x)
{
 if(x>0) return int(x+0.5);
 else return int(x-0.5);
}
void KhoiTaoPhepChieu()
{
  float th,ph;
  th = M_PI*theta/180;
  ph = M_PI*phi/180;
  temp1 = sin(th);
  temp2 = sin(ph);
  temp3 = cos(th);
  temp4 = cos(ph);
  temp5 = temp3*temp2;
  temp6 = temp1*temp2;
  temp7 = temp3*temp4;
  temp8 = temp1*temp4;
  PC.x = getmaxx()/2;
  PC.y = getmaxy()/2;
}
void  Chieu(ToaDo3D P)
{
   Obs.x = -P.x*temp1 + P.y*temp3;
   Obs.y = -P.x*temp5 - P.y*temp6 + P.z*temp4;
   if( project== PhoiCanh)
   {
     Obs.z =-P.x*temp7 - P.y*temp8 - P.z*temp2 + R;
     Xproj = D*Obs.x/Obs.z;
     Yproj = D*Obs.y/Obs.z;
   }
   else
   {
     Xproj = D*Obs.x;
```

```
      Yproj = D*Obs.y;
    }
}
void VeDen(ToaDo3D P)
{
   Chieu(P);
   PE.x = PC.x + round(Xproj);
   PE.y = PC.y - round(Yproj);
   lineto (PE.x,PE.y);
}
void DiDen(ToaDo3D P)
{
   Chieu(P);
   PE.x = PC.x + round(Xproj);
   PE.y = PC.y - round(Yproj);
   moveto (PE.x,PE.y);
}
void TrucToaDo()
{
 ToaDo3D OO,XX,YY,ZZ;
 setcolor(LIGHTRED);
 OO.x=0;  OO.y=0;  OO.z=0;
 XX.x=2;  XX.y=0;  XX.z=0;
 YY.x=0;  YY.y=2;  YY.z=0;
  ZZ.x=0;  ZZ.y=0;  ZZ.z=2;
 DiDen(OO);VeDen(XX);
 outtextxy(PE.x+2,PE.y,"X");
  DiDen(OO);VeDen(YY);
 outtextxy(PE.x+2,PE.y,"Y");
 DiDen(OO);VeDen(ZZ);
 outtextxy(PE.x+2,PE.y,"Z");
}
void DieuKhienQuay()
{
 ch=getch();
 if(ch==0) ch=getch();
 cleardevice();
 switch(ch)
 {
   case 75: theta = theta + IncAng;
                 break;
```

```
    case 77: theta = theta - IncAng;
             break;
    case 72: phi = phi + IncAng;
                  break;
    case 80: phi = phi - IncAng;
                  break;
    }
}
#endif
```

Ví dụ: Viết chương trình mô tả phép quay của một hình lập phương quanh các trục (Hình 6.12).



Hình 6.12

```
#include<stdio.h>
#include<conio.h>
#include<graphics.h>
#include"DoHoa3D.h"
ToaDo3D P1,P2,P3,P4,P5,P6,P7,P8;
void KhoiTaoBien()
{
 D=100;R=15;
 theta=40;phi=20;
 P1.x=0;          P1.y=0;   P1.z=0;
 P2.x=0;          P2.y=1;   P2.z=0;
 P3.x=1;          P3.y=1;   P3.z=0;
 P4.x=1;          P4.y=0;   P4.z=0;
 P5.x=1;          P5.y=0;   P5.z=1;
 P6.x=0;          P6.y=0;   P6.z=1;
 P7.x=0;          P7.y=1;   P7.z=1;
 P8.x=1;          P8.y=1;   P8.z=1;
}
```

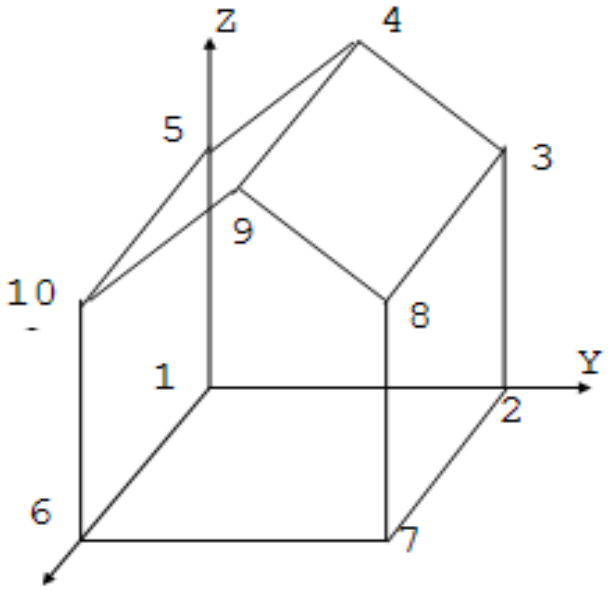
```
void VeLapPhuong()
{
 DiDen(P1); VeDen(P2);
 VeDen(P3); VeDen(P4);
 VeDen(P1); VeDen(P6);
 VeDen(P7); VeDen(P8);
 VeDen(P5); VeDen(P6);
 DiDen(P3); VeDen(P8);
 DiDen(P2); VeDen(P7);
 DiDen(P4); VeDen(P5);
}
void MinhHoa()
{
 KhoiTaoBien();
 KhoiTaoPhepChieu();
 //TrucToaDo();
 VeLapPhuong();
 do
  {
    DieuKhienQuay();
    KhoiTaoPhepChieu();
    cleardevice();
    //TrucToaDo();
    VeLapPhuong();
  }
 while (ch!=27);
}
int main()
{
 project=SongSong;
 ThietLapDoHoa();
 MinhHoa();
}
```

6.5. MÔ HÌNH WIREFRAME

Mô hình WireFrame mô tả đối tượng 3D bằng 2 danh sách:

- Danh sách các đỉnh: lưu tọa độ (x,y,z) của các đỉnh.
- Danh sách các cạnh: lưu số thứ tự đỉnh đầu và đỉnh cuối của từng cạnh.

Ví dụ: Biểu diễn 1 căn nhà thô sơ (Hình 6.13)



Hình 6.13

Danh sách các đỉnh

Vector	*x*	*y*	*z*
1	0	0	0
2	0	1	0
3	0	1	1
4	0	0.5	1.5
5	0	0	1
6	1	0	0
7	1	1	0
8	1	1	1
9	1	0.5	1.5
10	1	0	1

Danh sách các cạnh

Cạnh	*Đỉnh đầu*	*Đỉnh cuối*
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

6.5.1. Xây dựng cấu trúc dữ liệu

Có nhiều cách để lưu giữ mô hình WireFrame. Ở đây, chúng ta sẽ xây dựng một cấu trúc dựa trên 2 mảng:

```
struct wireframe
{
  int sodinh,socanh;
  ToaDo3D dinh[50];
  int canh[100][2];
};
wireframe WF;
```

Với biến WF ở trên, ta có thể gán giá trị như sau:

```
  WF.sodinh:=10;
  WF.socanh:=17;
  WF.dinh[1].x:=0;
  WF.dinh[1].y:=0;
  WF.dinh[1].z:=0;
...
  WF.canh[1,0]:=1;   {đỉnh đầu của cạnh số 1}
  WF.canh[1,1]:=2;   {đỉnh cuối của cạnh số 1}
  ...
```

Chú ý: Thông thường, dữ liệu của mô hình WireFrame được lưu trữ trong file text với cấu trúc như sau:

- Dòng đầu tiên lưu hai số nguyên dương m và n tương ứng với số đỉnh và số cạnh của mô hình.
- M dòng tiếp theo mỗi dòng lưu bộ ba (x,y,z) của từng đỉnh.
- N dòng tiếp theo mỗi dòng cặp số thứ tự đỉnh đầu và đỉnh cuối của từng cạnh.

Sau đây là hàm đọc dữ liệu từ file text để lưu vào biến WF của mô hình WireFrame:

```
void DocFile(char *FileName,wireframe &WF)
{
  FILE *f;
  int x,i,j;
  float xx;
  f=fopen(FileName,"rt");
  fscanf(f,"%d",&x);  WF.sodinh=x;
  fscanf(f,"%d",&x);  WF.socanh=x;
  for(i=1;i<= WF.sodinh;i++)
  {
    fscanf(f,"%f",&xx); WF.dinh[i].x=xx;
    fscanf(f,"%f",&xx); WF.dinh[i].y=xx;
    fscanf(f,"%f",&xx); WF.dinh[i].z=xx;
  }
```

```
    for(i=1;i<= WF.socanh;i++)
    {
      fscanf(f,"%d",&x);  WF.canh[i][0]=x;
      fscanf(f,"%d",&x);  WF.canh[i][1]=x;
    }
    fclose(f);
}
```

6.5.2. Vẽ mô hình WireFrame

Để vẽ đối tượng WireFrame, ta vẽ từng cạnh trong danh sách các cạnh của mô hình. Sau đây là hàm vẽ đối tượng WireFrame (sử dụng thư viện "Dohoa3D.h"):

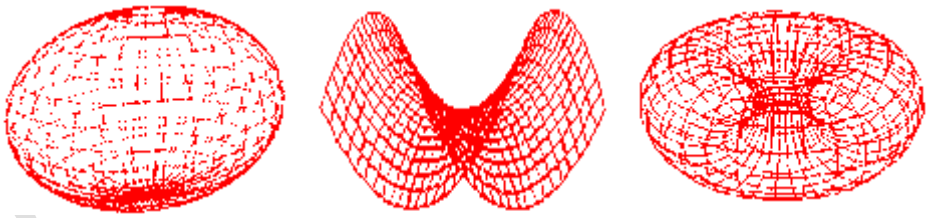
```
void Draw(wireframe wf)
{
  ToaDo3D d1,d2;
  for(int i=1;i<=wf.socanh;i++)
  {
    d1=wf.dinh[wf.canh[i][0]];
    d2=wf.dinh[wf.canh[i][1]];
    DiDen(d1);
    VeDen(d2);
  }
}
```

6.6. VẼ CÁC MẶT TOÁN HỌC

Phần này sẽ trình bày phương pháp vẽ các mặt cong dựa trên phương trình tham số của nó. Phương trình tham số của một mặt cong thường có dạng hàm hai biến:

$$F(u,v) = (X(u,v),\ Y(u,u),\ Z(u,v)) \qquad (*)$$

Ví dụ:



(a) (b) (c)

Hình 6.14

- Mặt Ellipsoid (Hình 6.14a)

  $X(u,v) = R_x.\cos(u).\cos(v)$

  $Y(u,v) = R_y.\sin(u).\cos(v)$

  $Z(u,v) = R_z.\sin(v)$

  Trong đó: $0 \le u \le 2\pi$ và $-\pi/2 \le v \le \pi/2$

- Mặt Hypeboloid: (Hình 6.14b)

  $X(u,v) = u$

Y(u,v) = v

$Z(u,v) = u^2 - v^2$

Trong đó u,v ∈[-1,1]

- Hình xuyến: (Hình 6.14c)

X(u,v) = (R + a.cos(v)).cos(u)

Y(u,v) = (R + a.cos(v)).sin(u)

Z(u,v) = a.sin(v)

Trong đó: $0 \le u \le 2\pi$ $-\pi/2 \le v \le \pi/2$

- Hình trụ tròn (Cylinder)

X(u,v) = R.cos(u)

Y(u,v) = R.sin(u)

Z(u,v) = h

- Hình nón (Cone)

$p(u,v) = (1-v).P_0 + v.P_1(u)$

trong đó:

P_0: đỉnh nón

$P_1(u)$: đường tròn $\begin{cases} x = R.\cos(u) \\ y = R.\sin(u) \end{cases}$ u,v ∈ [0,1]

- Chảo Parabol (Paraboloid)

X(u,v) = a.v.cos(u)

Y(u,v) = b.v.sin(u) u∈[-π,π], v ≥ 0

$Z(u,v) = v^2$

Ý tưởng chính của phương pháp này là cố định một biến (u hoặc v) để đưa về hàm một biến, sau đó vẽ các đường cong theo u và v.

Để vẽ một đường cong u tại giá trị u' khi v chạy từ VMin đến Vmax, ta thực hiện như sau:

- Tạo một tập hợp các giá trị v[i] ∈ [VMin ,VMax], xác định vị trí P[i] = (X(u',v[i]), Y(u',v[i]), Z(u',v[i])).
- Chiếu từng điểm P[i] lên mặt phẳng và nối chúng lại với nhau.

Sau đây là hàm vẽ họ đường cong theo u:

```
void DuongCongU()
{
  ToaDo3D P;
  u=uMin;
  while (u<=uMax)
  {
    v=vMin;
    P.x=fx(u,v);
    P.y=fy(u,v);
    P.z=fz(u,v);
    DiDen(P);
```

```
    while (v<=vMax)
    {
            P.x=fx(u,v);
            P.y=fy(u,v);
            P.z=fz(u,v);
            VeDen(P);
            v+=dv;
    }
    u+=du;
  }
}
```

Tương tự, ta có thể vẽ họ đường cong theo v.

Ví dụ: Viết chương trình để vẽ mặt Hypeboloid (mặt yên ngựa).

```
#include <conio.h>
#include <graphics.h>
#include "dohoa3d.h"
float v,vMin,vMax,dv,u,uMin,uMax,du;
float fx(float u,float v)
{
  return u;
}
float fy(float u,float v)
{
  return v;
}
float fz(float u,float v)
{
  return u*u-v*v;
}
void DuongCongV()
{
  ToaDo3D P;
  v=vMin;
  while (v<=vMax)
  {
    u=uMin;
    P.x=fx(u,v);
    P.y=fy(u,v);
    P.z=fz(u,v);
    DiDen(P);
    while (u<=uMax)
```

```
    {
            P.x=fx(u,v);
            P.y=fy(u,v);
            P.z=fz(u,v);
            VeDen(P);
            u+=du;
    }
    v+=dv;
  }
}
void DuongCongU()
{
  ToaDo3D P;
  u=uMin;
  while (u<=uMax)
  {
    v=vMin;
    P.x=fx(u,v);
    P.y=fy(u,v);
    P.z=fz(u,v);
    DiDen(P);
    while (v<=vMax)
    {
            P.x=fx(u,v);
            P.y=fy(u,v);
            P.z=fz(u,v);
            VeDen(P);
            v+=dv;
    }
    u+=du;
  }
}
int main()
{
  theta=60;phi=20;R=2;D=200;
  uMin=-1;uMax=1;du=0.1;
  vMin=-1;vMax=1;dv=0.1;
  ThietLapDoHoa();
  do
  {
    KhoiTaoPhepChieu();
```

```
        DuongCongU();
        DuongCongV();
        DieuKhienQuay();
       }
     while (ch!=27);
     closegraph();
   }
```

BÀI TẬP

1. Tạo ra các file text để lưu các khối đa diện đều: Tứ diện, lập phương, bát diện, thập nhị diện (12 mặt), nhị thập diện (20 mặt) theo mô hình WireFrame.

2. Viết chương trình vẽ các khối đa diện đều đã được lưu trong các file text ở bài tập 1.

3. Viết chương trình để vẽ các mặt toán học: mặt cầu, hình xuyến...

4. Lập trình mô phỏng chuyển động của trái đất xung quanh mặt trời đồng thời mô tả chuyển động của mặt trăng xung quanh trái đất trong không gian 3 chiều.

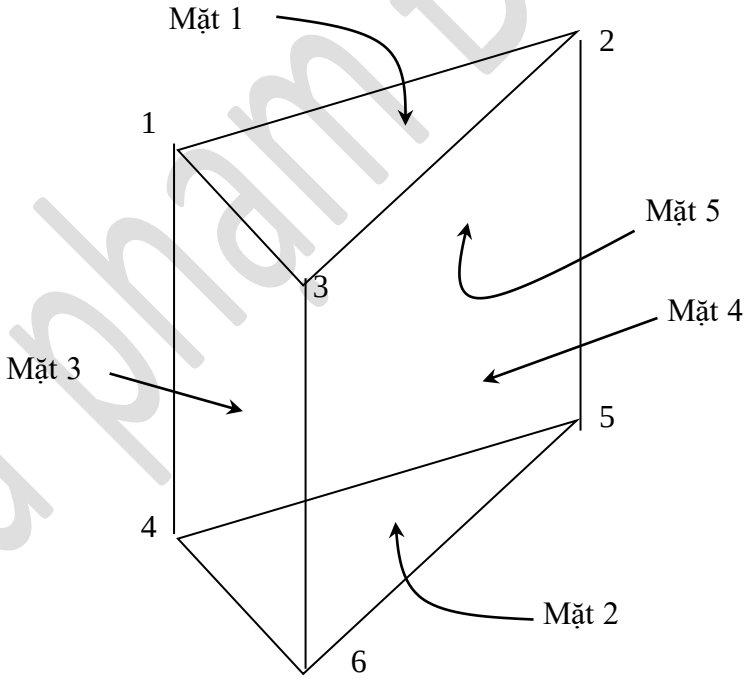
CHƯƠNG 7: KHỬ ĐƯỜNG VÀ MẶT KHUẤT

7.1. MÔ HÌNH CÁC MẶT ĐA GIÁC

Một vật thể 3D có thể biểu diễn trong máy tính bằng nhiều mô hình khác nhau, hai mô hình phổ biến nhất là mô hình khung dây (WireFrame) và mô hình ***các mặt đa giác*** (Polygon mesh model).

- Mô hình WireFrame: Đã trình bày ở chương 6, nó cho ta hình dáng của vật thể dưới dạng một bộ khung.
- Mô hình các mặt đa giác: một vật thể 3D được xác định thông qua các mặt (thay vì các cạnh như trong mô hình WireFrame), và mỗi một mặt lại được xác định thông qua các điểm mà các điểm này được xem như là các đỉnh của mặt đa giác. Với mô hình các mặt đa giác, ta không chỉ tạo ra được hình dáng của vật thể như mô hình Wireframe mà còn thể hiện được các đặc tính về màu sắc và nhiều tính chất khác của vật thể. Song để có thể mô tả vật thể 3D một cách trung thực (như trong thế giới thực) thì đòi hỏi người lập trình phải tính toán và giả lập nhiều thông tin, mà mấu chốt là vấn đề khử mặt khuất và nguồn chiếu sáng. Chương này sẽ tập trung nghiên cứu các vấn đề về khử mặt khuất.

Ví dụ: Mô tả vật thể hình lăng trụ (Hình 7.1).



Hình 7.1

- Danh sách các đỉnh: 1,2,3,4,5,6
- Danh sách các mặt được xác định theo bảng sau:

Mặt	Đỉnh	
1	3	1,2,3
2	3	4,5,6

3	4	1,3,6,4
4	4	3,2,5,6
5	4	1,2,5,4

*** Vấn đề khử mặt khuất**

Khi thể hiện vật thể 3D, một vấn đề nảy sinh là làm sao chỉ thể hiện các mặt có thể nhìn thấy được mà không thể hiện các mặt khuất phía sau. Việc một mặt bị khuất hay không bị khuất thì tuỳ thuộc vào cấu trúc các mặt của vật thể và vị trí của điểm nhìn cũng như bối cảnh mà vật thể đó được đặt vào.

7.2. CÁC PHƯƠNG PHÁP KHỬ MẶT KHUẤT

7.2.1. Giải thuật Depth-sorting

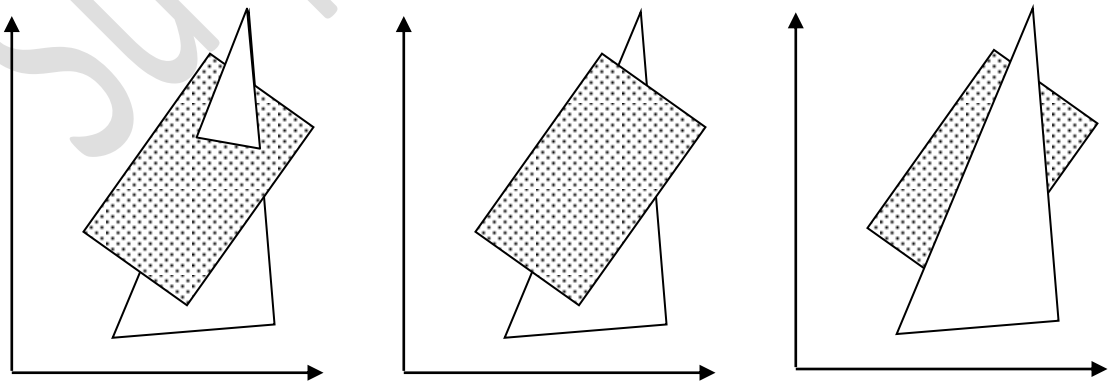
Giải thuật người thợ sơn hay Depth-sorting là tên của một thuật giải đơn giản nhất trong số các thuật toán vẽ ảnh thực 3 chiều. Nếu để ý người thợ sơn làm việc, ta sẽ thấy anh ta sơn bức tranh từ trong ra ngoài, với các cảnh vật từ xa đến gần. Chúng ta có thể áp dụng một cách tương tự để vẽ các đa giác trong danh sách các đa giác. Song có một vấn đề cần phải chọn lựa, đó là một đa giác tồn tại trong không gian 3D có tới ba bốn đỉnh, và những đỉnh này có thể có các giá trị z (giá trị độ sâu) khác nhau. Chúng ta sẽ không biết chọn giá trị nào trong số chúng. Từ những kinh nghiệm trong thực tế, người ta cho rằng nên sử dụng giá trị z trung bình sẽ cho kết quả tốt trong hầu hết các trường hợp.

Do đó, cần phải sắp xếp các mặt theo thứ tự từ xa đến gần, sau đó vẽ các mặt từ xa trước, rồi vẽ các mặt ở gần sau, như thế thì các mặt ở gần sẽ không bị che khuất bởi các mặt ở xa, mà chỉ có các mặt ở xa mới có thể bị các mặt ở gần che khuất, do các mặt ở gần vẽ sau nên có thể được vẽ chồng lên hình ảnh của các mặt xa.

Như vậy, giải thuật Depth-Sorting được thực hiện một cách dễ dàng khi chúng ta xác định một giá trị độ sâu (là giá trị z trong hệ toạ độ quan sát) đại diện cho cả mặt. Các mặt dựa vào độ sâu đại diện của mình để so sánh rồi sắp xếp theo một danh sách giảm dần (theo độ sâu đại diện). Bước tiếp theo là vẽ các mặt lên mặt phẳng theo thứ tự trong danh sách.

Giải thuật còn một số vướng mắc sau (Hình 7.2):

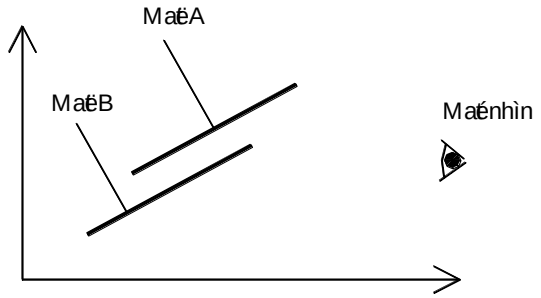
- Khi hai mặt cắt nhau thì thuật giải này chỉ thể hiện như chúng chồng lên nhau.
- Khi hai mặt ở trong cùng một khoảng không gian về độ sâu và hình chiếu của chúng lên mặt phẳng chiếu chồng lên nhau (hay chồng một phần lên nhau). Chẳng hạn như Hình 7.3.



Hình ảnh thật Khi vẽ bằng giải thuật trên

Hình 7.2

Từ những ví dụ trên ta thấy rằng, có những trường hợp các đa giác được sắp xếp sai dẫn đến kết quả hiển thị không đúng. Liệu có thể khắc phục được vấn đề này không? Câu trả lời dĩ nhiên là có nhưng cũng đồng nghĩa là phải xử lý thêm rất nhiều các trường hợp và làm tăng độ phức tạp tính toán.



Hình 7.3

- **Phép kiểm tra phần kéo dài Z**

Phép kiểm tra này nhằm xác định phần kéo dài z của hai đa giác có gối lên nhau hay không? Nếu các phần kéo dài Z là gối lên nhau rất có thể các đa giác này cần được hoán đổi. Vì thế phép kiểm tra tiếp theo phải được thực hiện.

- **Phép kiểm tra phần kéo dài X**

Phép kiểm tra này tương tự như phép kiểm tra trước, nhưng nó sẽ kiểm tra phần kéo dài X của hai đa giác có gối lên nhau hay không? Nếu có, thì rất có thể các đa giác này cần được hoán đổi. Vì thế phép kiểm tra tiếp theo phải được thực hiện.

- **Phép kiểm tra phần kéo dài Y**

Phép kiểm tra này kiểm tra phần kéo dài Y của hai đa giác có gối lên nhau hay không? Nếu có, thì rất có thể các đa giác này cần được hoán đổi. Vì thế phép kiểm tra tiếp theo phải được thực hiện.

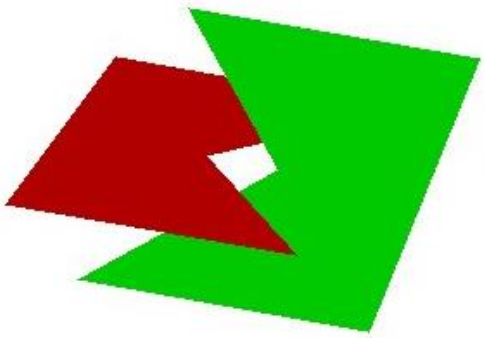
- **Phép kiểm tra cạnh xa**

Giả sử A và B là hai đa giác mà sau khi sắp xếp theo độ sâu trung bình thì A đứng trước B. Song qua 3 phép kiểm tra trên mà vẫn không xác định được liệu trật tự trên là đúng hay chưa. Lúc này chúng phải tiến hành phép kiểm tra cạnh xa. Phép kiểm tra cạnh xa nhằm xác định xem đa giác B có nằm phía sau cạnh xa của đa giác A hay không? Nếu có thì trật tự này là đúng, ngược lại thì phải qua bước kiểm tra tiếp theo.

Để kiểm tra đa giác B có nằm sau cạnh xa của đa giác A hay không, chúng ta thực hiện việc kiểm tra mỗi đỉnh của đa giác B. Các đỉnh này đều nằm về cùng một phía của đa giác A theo chiều trục Z không? Nếu đúng thì kết quả trật tự trên là đúng. Ngược lại, có thể xảy ra một trong hai tình huống như Hình 6.2 hoặc Hình 6.3, để xác định được ta phải tiếp tục sang bước kiểm tra tiếp theo.

- **Phép kiểm tra cạnh gần**

Phép kiểm tra cạnh gần nhằm xác định xem đa giác A có nằm phía sau cạnh gần của đa giác B hay không? Nếu có thì trật tự xác định trước đây không đúng, chúng ta cần phải hoán đổi lại trật tự. Ngược lại thì rõ ràng hai đa giác đang cắt nhau (như Hình 6.2) hoặc chéo vào nhau (Hình 6.4), lúc này chúng ta phải tiến hành chia nhỏ hai đa giác A và B thành 3 (hoặc 4) đa giác con, đường chia cắt chính là đường giao cắt của 2 đa giác. Sau phép chia chúng ta tiến hành sắp xếp lại các đa giác con.



Hình 7.4

7.2.2. Giải thuật BackFace

Giải thuật này sử dụng các kiến thức tích vô hướng và tích véc tơ để kiểm tra một mặt nào đó là thấy được hay không.

*** Tích vô hướng**: Cho hai véc tơ $\vec{v}$(v1,v2,v3) và $\vec{n}$(n1,n2,n3) thì tích vô hướng

$$\vec{v}.\vec{n} = |\vec{v}|.|\vec{n}|.\mathbf{Cos(\theta) = v1.n1 + v2.n2 + v3.n3}$$

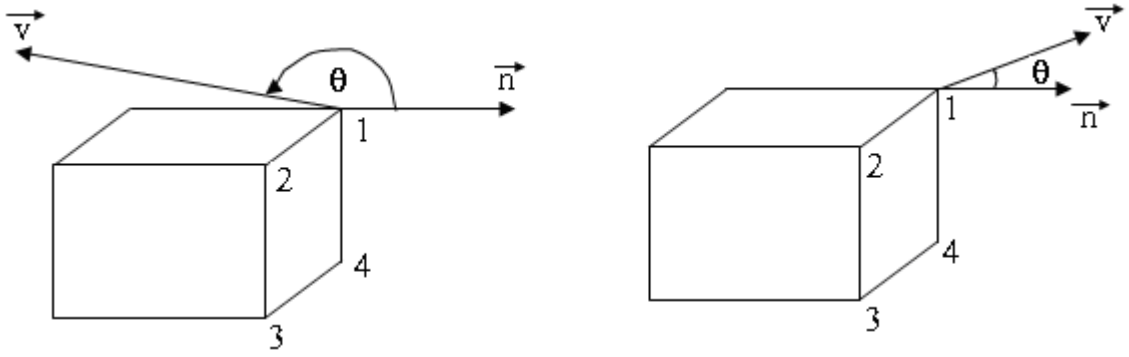
*** Tích véc tơ**: Cho hai véc tơ $\vec{p}$(p1,p2,p3) và $\vec{q}$(q1,q2,q3) trong hệ tọa độ trực tiếp. Tích véc tơ của $\vec{p}$ và $\vec{q}$ là véc tơ $\vec{n}$(n1,n2,n3) vuông góc với mặt phẳng được tạo bởi hai véc tơ $\vec{p}$ và $\vec{q}$: [p,q] = $\vec{n}$ với

n1 = p2.q3 - q2.p3

n2 = p3.q1 - q3.p1

n3 = p1.q2 - q1.p2

Ví dụ: Cho hình hộp chữ nhật như Hình 7.5. Xét xem mặt được đánh số (1,2,3,4) là thấy được hay không thấy được?



Hình 7.5

Bước 1: Gọi $\vec{v}$: Vector xuất phát từ một đỉnh nào đó của mặt (1,2,3,4) hướng đến mắt của người quan sát (Giả sử chọn đỉnh số 1).

$\vec{n}$: Vector chuẩn của mặt (1,2,3,4) hướng ra phía ngoài vật thể.

Bước 2: Gọi θ: góc giữa $\vec{v}$ **và** $\vec{n}$**.**

- Nếu cos(θ) > 0 thì Mặt (1,2,3,4) thấy được.
- Ngược lại: Mặt (1,2,3,4) bị khuất.

Sau đây là chương trình cài đặt thuật toán BackFace:

```
#include <conio.h>
#include <graphics.h>
```

```
#include <stdio.h>
#include "Dohoa3d.h"
#define MaxSoDinh 50;
#define MaxSoMat 30;
#define MaxDinh  10;
typedef  ToaDo3D TapDinh[50];
typedef  int TapMat[30][10];
struct FaceModel
{
   int SoDinh;
   TapDinh Dinh;
   int SoMat;
   TapMat Mat;
};
FaceModel Hinh;
ToaDo3D O;
void KhoiTao()
{
  project=PhoiCanh;
  R=150; theta=20;
  phi=15; D=3000;
}
void VectorNhin(int Dinh1, ToaDo3D &v)
{
  v.x=O.x - Hinh.Dinh[Dinh1].x;
  v.y=O.y -Hinh.Dinh[Dinh1].y;
  v.z=O.z -Hinh.Dinh[Dinh1].z;
}
void VectorChuan(int Dinh1,int Dinh2,int Dinh3, ToaDo3D &N)
{
  ToaDo3D A,B;
  A.x=Hinh.Dinh[Dinh2].x - Hinh.Dinh[Dinh1].x;
  A.y=Hinh.Dinh[Dinh2].y - Hinh.Dinh[Dinh1].y;
  A.z=Hinh.Dinh[Dinh2].z - Hinh.Dinh[Dinh1].z;
  B.x=Hinh.Dinh[Dinh3].x - Hinh.Dinh[Dinh1].x;
  B.y=Hinh.Dinh[Dinh3].y - Hinh.Dinh[Dinh1].y;
  B.z=Hinh.Dinh[Dinh3].z - Hinh.Dinh[Dinh1].z;
  N.x=A.y*B.z - B.y*A.z;
  N.y=A.z*B.x - B.z*A.x;
  N.z=A.x*B.y - B.x*A.y;
}
```

```
float TichVoHuong(ToaDo3D v,ToaDo3D n)
{
  return v.x*n.x + v.y*n.y + v.z*n.z;
}
void ToaDoQuanSat()
{
  KhoiTaoPhepChieu();
  O.x= R*temp7;
  O.y= R*temp8;
  O.z= R*temp2;
}
void DocFile(char *FileName,FaceModel & WF)
{
  FILE *f;
  int x,i,j;
  f=fopen(FileName,"rt");
  fscanf(f,"%d",&x);  WF.SoDinh=x;
  fscanf(f,"%d",&x);  WF.SoMat=x;
  for(i=1;i<=WF.SoDinh;i++)
  {
    fscanf(f,"%d",&x); WF.Dinh[i].x=x;
    fscanf(f,"%d",&x); WF.Dinh[i].y=x;
    fscanf(f,"%d",&x); WF.Dinh[i].z=x;
  }
  for(i=1;i<=WF.SoMat;i++)
  {
    fscanf(f,"%d",&x);  WF.Mat[i][0]=x;
    for(j=1;j<= WF.Mat[i][0];j++)
    {
       fscanf(f,"%d",&x);  WF.Mat[i][j]=x;
    }
  }
  fclose(f);
}
void VeMat(int k)
{
  int SoDinh,i,j;
  ToaDo3D P,P0;
  SoDinh=Hinh.Mat[k][0];
  for(i=1;i<=SoDinh;i++)
  {
```

```
    j=(Hinh.Mat[k][i]);
    P.x=Hinh.Dinh[j].x;
     P.y=Hinh.Dinh[j].y;
    P.z=Hinh.Dinh[j].z;
    if(i==1)
    {
      DiDen(P);
      P0.x=P.x; P0.y=P.y; P0.z=P.z;
    }
    else VeDen(P);
  }
  VeDen(P0);
}
void VeVatThe(FaceModel Hinh)
{
  int i,Dinh1,Dinh2,Dinh3 ;
  ToaDo3D v,n;
  for(i=1;i<=Hinh.SoMat;i++)
  {
     Dinh1=Hinh.Mat[i][1];
     Dinh2=Hinh.Mat[i][2];
    Dinh3=Hinh.Mat[i][3];
     VectorNhin(Dinh1,v);
     VectorChuan(Dinh1,Dinh2,Dinh3,n);
     if( TichVoHuong(v,n)>0 )
     {
          setlinestyle(SOLID_LINE,0,NORM_WIDTH);
          VeMat(i);
     }
     else
          {
               setlinestyle(DOTTED_LINE,0,NORM_WIDTH);
               VeMat(i);
          }
  }
}
void DieuKhien()
{
   ToaDoQuanSat();
   VeVatThe(Hinh);
   do
```

```
    {
      DieuKhienQuay();
      ToaDoQuanSat();
      VeVatThe(Hinh);
    }
    while (ch!=27);
}
int main()
{
  DocFile("D:\\ltdh\\c6\\BATDIEN.txt",Hinh);
  ThietLapDoHoa();
  KhoiTao();
  DieuKhien();
  closegraph();
}
```

7.2.3. Giải thuật vùng đệm độ sâu (Z-Buffer)

Bằng cách tính giá trị độ sâu (là giá trị Z trong hệ toạ độ quan sát) của mỗi điểm trong tất cả các mặt đa giác, tại mỗi điểm trên mặt phẳng chiếu có thể có ảnh của nhiều điểm trên nhiều mặt đa giác khác nhau, song hình vẽ chỉ được thể hiện hình ảnh của điểm có độ sâu thấp nhất (tức là điểm ở gần nhất). Với cách thực hiện này giải thuật có thể khử được tất cả các trường hợp mà các giải thuật khác mắc phải.

Giới hạn của phương pháp này là đòi hỏi nhiều bộ nhớ và thực hiện nhiều tính toán. Z-Buffer là một bộ đệm dùng để lưu độ sâu cho mỗi pixel trên hình ảnh của vật thể, thông thường ta tổ chức nó là một ma trận hình chữ nhật. Nếu dùng 1 byte để biểu diễn độ sâu của một pixel, thì một vật thể có hình ảnh trên mặt phẳng chiếu là 100x100 sẽ cần 10000 byte dùng để làm Depth Buffer, và khi đó vùng đệm độ sâu sẽ cho phép ta phân biệt được 256 mức sâu khác nhau, điều này có nghĩa là nếu có 257 pixel ở 257 độ sâu khác nhau thì khi đó buột ta phải quy 2 pixel nào đó về cùng một độ sâu. Nếu ta dùng 4 byte để biểu diễn độ sâu của một pixel, thì khi đó vùng đệm độ sâu sẽ cho phép ta phân biệt được 4294967296 (2^{32}) mức sâu khác nhau, song lúc đó sẽ phải cần 40000 byte cho một bộ đệm kích thước 100x100. Do tính chất 2 mặt này nên tuỳ vào tình huống và yêu cầu mà ta có thể tăng hay giảm số byte để lưu giữ độ sâu của 1 pixel. Và thông thường người ta dùng 4 byte để lưu giữ độ sâu của một điểm, khi đó thì độ chính xác rất cao.

Một câu hỏi có thể đặt ra là làm sao có thể tính độ sâu của mỗi điểm trong đa giác. Ở đây có 2 phương pháp: phương pháp trực tiếp và phương pháp gián tiếp.

- Phương pháp trực tiếp sẽ tính độ sâu của mỗi điểm dựa vào phương trình mặt phẳng chứa đa giác. Với phương pháp này chúng ta cần duyệt qua tất các điểm của đa giác (tất nhiên chỉ hữu hạn điểm), bằng cách cho các thành phần x và y, nếu cặp giá trị (x,y) thoả trong miền giới hạn của đa giác thì chúng ta sẽ tìm thành phần thứ 3 là z bằng cách thay thế x và y vào phương trình mặt phẳng để tính ra thành phần z. Về mặt toán học thì phương pháp trực tiếp rõ ràng là rất khoa học, song khi áp dụng ta sẽ gặp phải vướng mắc:

  Cần phải tính bao nhiêu điểm để hình ảnh thể hiện của đa giác lên mặt phẳng chiếu đủ mịn và cũng không bị tình trạng quá mịn (tức là vẽ rất nhiều điểm chồng chất lên

nhau không cần thiết mà lại gây ra tình trạng chậm chạp và tăng độ phức tạp tính toán. Cũng nên nhớ rằng khi thể hiện một đa giác lên mặt phẳng chiếu thì ảnh của nó có thể được phóng to hay thu nhỏ).

- Phương pháp gián tiếp: Chúng ta sẽ tính độ sâu của một điểm gián tiếp thông qua độ sâu của các điểm lân cận. Để thực hiện chúng ta tiến hành theo các bước sau:

  Gọi G là một mặt đa giác được biểu diễn bởi tập các điểm P_1, P_2, … P_n và G' là hình chiếu của G xuống mặt phẳng chiếu với tập các đỉnh P_1',P_2',… P_n'.

  Để thể hiện hình ảnh của G lên mặt phẳng chiếu thì rõ ràng là chúng ta phải tiến hành tô đa giác G'. Song như thuật toán đã phát biểu, chúng ta cần xác định xem mỗi điểm M' bất kỳ thuộc G' là ảnh của điểm M nào trên G và dựa vào độ sâu của M để so sánh với độ sâu đã có trong z-buffer để quyết định là có vẽ điểm M' hay không. Nếu ta gán thêm cho các điểm ảnh một thành phần nữa, đó là giá trị độ sâu của điểm tạo ảnh (tức là điểm đã tạo ra điểm ảnh sau phép chiếu) thì lúc này ta không cần thiết phải xác định M để tính độ sâu, mà ta có thể tính được giá trị độ sâu này qua công thức sau:

  Nếu M' nằm trên đoạn thẳng P'Q' với tỷ lệ là: P'M'/P'Q'=t và nếu biết được độ sâu của P' và Q' lần lượt là z(P') và z(Q') thì độ sâu mà điểm ảnh M' nhận được là

  $$\mathbf{z(M')=z(P')+(z(Q')-z(P'))t} \qquad (7.1)$$

  Ta có thể sử dụng được công thức trên với tất cả các phép chiếu có bảo toàn đường thẳng. Từ đó ta có thể xác định quy trình vẽ đa giác G' là ảnh của G như sau: Gán thêm cho mỗi điểm đỉnh của đa giác G' một thành phần z có giá trị bằng độ sâu của điểm tạo ảnh. Có nghĩa là P'_1 sẽ chứa thêm giá trị $z(P_1)$, P'_2 sẽ chứa thêm giá trị $z(P_2)$, hay một cách tổng quát P'_i sẽ chứa thêm giá trị $z(P_i)$ với i=1..n.

  Tiến hành tô đa giác G' theo một quy trình tương tự như thuật toán tô đa giác theo dòng quét. Có nghĩa là cho một dòng quét chạy ngang qua đa giác, tại mỗi vị trí bất kỳ của dòng quét, chúng ta tiến hành tìm tập các giao điểm của dòng quét với đa giác. Gọi $\{x_m\}$ là tập các giao điểm, một điều cần chú ý là ta cần tính độ sâu cho các giao điểm này. Giả sử x_i là giao điểm của đường quét với cạnh $P_i'P_j'$ thế thì ta có thể tính ra độ sâu của x_i thông qua công thức (6.1) như sau:

  Nếu gọi y_{scan} là giá trị tung độ của dòng quét thế thì:

  $$\mathbf{z(x_i) = z(P_i')+z(P_j')*[(y_{scan} - y(P_i'))/(y(P_j')-y(P_i'))]} \qquad (7.2)$$

  {trong đó y(P) là thành phần toạ độ y của điểm P}

  Rõ ràng qua công thức trên ta thấy, nếu x_i là trung điểm của $P_i'P_j'$ thì $z(x_i) = z(P_i')+z(P_j')*1/2$.

BÀI TẬP

1. Viết chương trình cài đặt giải thuật Depth-Sorting để biểu diễn và quan sát vật thể 3D theo mô hình "các mặt đa giác".

2. Viết chương trình cài đặt giải thuật vùng đệm độ sâu Z-Buffer.

TÀI LIỆU THAM KHẢO

[1]. Francis S. Hill , *Computer Graphics*, 1990 (Vol. I & II).

[2]. Nguyễn Xuân Huy, *Giáo trình Đồ hoạ Máy tính*, Trường Đại học Bách Khoa Hà Nội, 1994.

[3]. Phan Hữu Phúc, *Cơ sở Đồ hoạ Máy tính*, NXB Giáo dục Hà Nội, 1998.

[4]. Lê Tấn Hùng, Huỳnh Quyết Thắng, *Kỹ thuật đồ họa*, Nhà xuất bản khoa học và kỹ thuật, Hà nội, 2000.