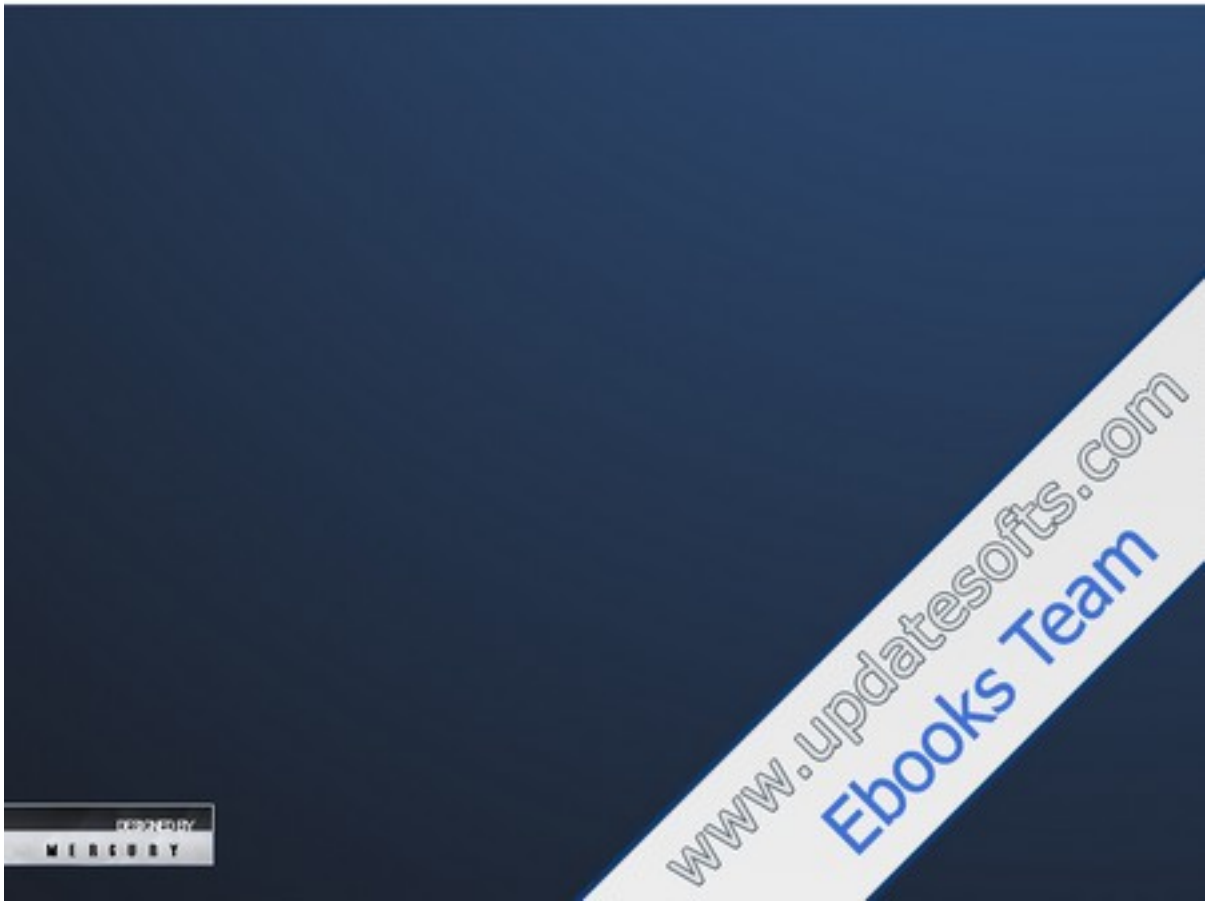
Giáo Trình

Lý Thuyết Đồ Họa



MỤC LỤC

Chương 1: CÁC YẾU TỐ CƠ SỞ CỦA ĐỒ HỌA
1.1. Tổng quan về đồ họa máy tính 1
1.1.1. Giới thiệu về đồ họa máy tính 1
1.1.2. Các kỹ thuật đồ họa 1
1.1.2.1. Kỹ thuật đồ họa điểm 1
1.1.2.2. Kỹ thuật đồ họa vector 2
1.1.3. Ứng dụng của đồ họa máy tính 2
1.1.4. Các lĩnh vực của đồ họa máy tính 3
1.1.5. Tổng quan về một hệ đồ họa 4
1.2. Màn hình đồ họa 6
1.3. Các khái niệm 6
1.3.1. Điểm 6
1.3.2. Các biểu diễn tọa độ 8
1.3.3. Đoạn thẳng 8
1.4. Các thuật toán vẽ đoạn thẳng 8
1.4.1. Bài toán 8
1.4.2. Thuật toán DDA 9
1.4.3. Thuật toán Bresenham 10
1.4.4. Thuật toán MidPoint 12
1.5. Thuật toán vẽ đường tròn 14
1.5.1. Thuật toán Bresenham 14
1.5.2. Thuật toán MidPoint 16
1.6. Thuật toán vẽ Ellipse 17
1.6.1. Thuật toán Bresenham 17
1.6.2. Thuật toán MidPoint 20
1.7. Phương pháp vẽ đồ thị hàm số 21
Bài tập 23

Chương 2: TÔ MÀU
2.1. Giới thiệu các hệ màu 25
2.2. Các thuật toán tô màu 28
2.2.1. Bài toán 28
2.2.2. Thuật toán xác định $P \in S$ 28
2.2.3. Thuật toán tô màu theo dòng quét 30
2.2.4. Thuật toán tô màu theo vết dầu loang 30
Bài tập 31

Chương 3: XÉN HÌNH
3.1. Đặt vấn đề 32

3.2. Xén đoạn thẳng vào vùng hình chữ nhật 32
3.2.1. Cạnh của hình chữ nhật song song với các trục tọa độ 32
3.2.1.1. Thuật toán Cohen – Sutherland 33
3.2.1.2. Thuật toán chia nhị phân 34
3.2.1.3. Thuật toán Liang – Barsky 35
3.2.2. Khi cạnh của hình chữ nhật tạo với trục hoành một góc α 36
3.3. Xén đoạn thẳng vào hình tròn 37
3.4. Xén đường tròn vào hình chữ nhật 38
3.5. Xén đa giác vào hình chữ nhật 39
Bài tập 40

Chương 4: CÁC PHÉP BIẾN ĐỔI
4.1. Các phép biến đổi trong mặt phẳng 41
4.1.1. Cơ sở toán học 41
4.1.2. Ví dụ minh họa 43
4.2. Các phép biến đổi trong không gian 45
4.2.1. Các hệ trục tọa độ 45
4.2.2. Các công thức biến đổi 46
4.2.3. Ma trận nghịch đảo 48
4.3. Các phép chiếu của vật thể trong không gian lên mặt phẳng 48
4.3.1. Phép chiếu phối cảnh 48
4.3.2. Phép chiếu song song 50
4.4. Công thức của các phép chiếu lên màn hình 50
4.5. Phụ lục 56
4.6. Ví dụ minh họa 59
Bài tập 61

Chương 5: BIỂU DIỄN CÁC ĐỐI TƯỢNG BA CHIỀU
5.1. Mô hình WireFrame 63
5.2. Vẽ mô hình WireFrame với các phép chiếu 64
5.3. Vẽ các mặt toán học 65
Bài tập 68

Chương 6: THIẾT KẾ ĐƯỜNG VÀ MẶT CONG BEZIER VÀ B-SPLINE
6.1. Đường cong Bezier và mặt Bezier 69
6.1.1. Thuật toán Casteljau 70
6.1.2. Dạng Bernstein của đường cong Bezier 70
6.1.3. Dạng biểu diễn ma trận của đường Bezier 71
6.1.4. Tạo và vẽ đường cong Bezier 72
6.1.5. Các tính chất của đường Bezier 74
6.1.6. Đánh giá các đường cong Bezier 76
6.2. Đường cong Spline và B-Spline 77
6.2.1. Định nghĩa 77

6.2.2. Các tính chất hữu ích trong việc thiết kế các đường cong B-Spline 78
6.2.3. Thiết kế các mặt Bezier và B-Spline 79
6.2.4. Các băng Bezier 80
6.2.5. Dán các băng Bezier với nhau 81
6.2.6. Các băng B-Spline 81

Chương 7: KHỬ ĐƯỜNG VÀ MẶT KHUẤT
7.1. Các khái niệm 83
7.2. Các phương pháp khử mặt khuất 85
7.2.1. Giải thuật sắp xếp theo chiều sâu 85
7.2.2. Giải thuật BackFace 88
7.2.3. Giải thuật vùng đệm độ sâu 90
Bài tập 103

Chương 8: TẠO BÓNG VẬT THỂ 3D
8.1. Khái niệm 104
8.2. Nguồn sáng xung quanh 104
8.3. Nguồn sáng định hướng 105
8.4. Nguồn sáng điểm 109
8.5. Mô hình bóng Gouraud 110
Bài tập 121

Phụ lục: MỘT SỐ CHƯƠNG TRÌNH MINH HỌA
I. Các thuật toán tô màu 122
II. Các thuật toán xén hình 129
III. Vẽ các đối tượng 3D 136

Tài liệu tham khảo 143

LỜI MỞ ĐẦU

Đồ họa là một trong những lĩnh vực phát triển rất nhanh của ngành Công nghệ thông tin. Nó được ứng dụng rộng rãi trong nhiều lĩnh vực khoa học và công nghệ. Chẳng hạn như y học, kiến trúc, giải trí... Đồ họa máy tính đã giúp chúng ta thay đổi cách cảm nhận và sử dụng máy tính, nó đã trở thành những công cụ trực quan quan trọng không thể thiếu trong đời sống hằng ngày. Vì vậy môn "Đồ họa" đã trở thành một trong những môn học chính trong các chuyên ngành Công nghệ thông tin ở các trường đại học.

Cuốn sách "Giáo trình lý thuyết đồ họa" được biên soạn theo sát nội dung chương trình đào tạo cử nhân Công nghệ thông tin. Nội dung của giáo trình này cung cấp một số kiến thức cơ bản về lý thuyết và thuật toán xây dựng các công cụ đồ họa 2D và 3D. Từ đó giúp sinh viên có thể độc lập xây dựng những thư viện đồ họa cho riêng mình và phát triển các phần mềm ứng dụng đồ họa cao hơn.

Giáo trình được chia làm 8 chương và phần phụ lục, sau mỗi chương đều có phần bài tập để kiểm tra kiến thức và rèn luyện khả năng lập trình cho bạn đọc. Để thuận tiện cho việc trình bày thuật toán một cách dễ hiểu, các giải thuật trong giáo trình được viết trên ngôn ngữ "tựa Pascal" và các mã nguồn được cài đặt trên Turbo Pascal 7.0. Nhằm giúp bạn đọc bớt lúng túng trong quá trình cài đặt các giải thuật, phần phụ lục liệt kê một số mã nguồn cài đặt các thuật toán trong các chương. Tuy nhiên, bạn đọc nên tự cài đặt các thuật toán ở phần lý thuyết, nếu cảm thấy khó khăn lắm mới nên tham khảo phần phụ lục này.

Chương 1, 2 và 3 trình bày về các yếu tố cơ sở của đồ họa như: màn hình đồ họa, điểm, đoạn thẳng, đường tròn, các hệ màu và các thuật toán tô màu, xén hình ... Chương 4 trang bị các kiến thức toán học về các phép biến đổi trong không gian 2D và 3D. Chương 5, 6 và 7 giới thiệu các mô hình đồ họa 3D, các giải thuật khử mặt khuất và tạo bóng cho vật thể... Chương 8 trình bày về phương pháp thiết kế các đường cong Bezier và B-Spline.

Mặc dù đã rất cố gắng trong quá trình biên soạn nhưng chắc chắn giáo trình này vẫn không thể tránh khỏi những thiếu sót. Chúng tôi rất mong nhận được những ý kiến đóng góp của bạn đọc cũng như các bạn đồng nghiệp trong lĩnh vực Đồ họa để giáo trình ngày càng được hoàn thiện hơn trong lần tái bản sau. Địa chỉ liên lạc:

Khoa Công nghệ Thông tin, trường Đại học Khoa học Huế.

Điện thoại: 054.826767. Email: paphuong@hueuni.edu.vn

nhtai@hueuni.edu.vn

Huế, tháng 08 năn 2003

Các tác giả

CHƯƠNG I

CÁC YẾU TỐ CƠ SỞ CỦA ĐỒ HỌA

1.1. TỔNG QUAN VỀ ĐỒ HỌA MÁY TÍNH

Đồ họa máy tính là một lãnh vực phát triển nhanh nhất trong Tin học. Nó được áp dụng rộng rãi trong nhiều lãnh vực khác nhau thuộc về khoa học, kỹ nghệ, y khoa, kiến trúc và giải trí.

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.

1.1.1 Giới thiệu về đồ họa máy tính

Đồ họa máy tính là một **ngành khoa học Tin học** 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 hổ 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, dể dùng,... kích thích khả năng sáng tạo và nâng cao năng suất làm việc.

1.1.2. CÁC KỸ THUẬT ĐỒ HỌA

Dựa vào các phương pháp xử lý dữ liệu trong hệ thống, ta phân ra làm hai kỹ thuật đồ họa:

1.1.2.1. 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.

1.1.2.2. 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.

Ở kỹ thuật này, chúng ta 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.

1.1.3. Ứng dụng của đồ họa máy tính hiện nay

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:

1.1.3.1. ***Tạo giao diện*** (User Interfaces): như các chương trình ứng dụng WINDOWS, WINWORD, EXCEL ... đang được đa số người sử dụng ưa chuộng nhờ tính thân thiện, dể sử dụng.

1.1.3.2. ***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 hổ trợ đắc lực cho việc phân tích thông tin và trợ giúp ra quyết định.

1.1.3.3. ***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 ...

1.1.3.4. ***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 ...

1.1.3.5. ***Lĩnh vực giải trí, nghệ thuật***: cho phép các họa sĩ tạo ra các hình ảnh ngay 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 ... như các phần mềm PAINTBRUSH, CORELDRAW,...

1.1.3.6. ***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).

1.1.4. Các lĩnh vực của đồ họa máy tính

1.1.4.1. 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...

1.1.4.2. 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, ...

1.1.4.3. 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...

1.1.4.4. 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...

1.1.4.5. 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,...

1.1.5. Tổng quan về một hệ đồ họa (Graphics System)

1.1.5.1. 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, ... hổ 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.

1.1.5.2. 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.

1.2. 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 $12: chế độ phân giải 640x480x16
- Mode $13: chế độ phân giải 320x200x256

Ta có thể viết một thủ tục để khởi tạo chế độ đồ họa như sau:

```
Procedure InitGraph(Mode:Word);
 var Reg:Registers;
 Begin
  reg.ah := 0;
  reg.al := mode;
  intr($10,reg);
 End;
```

Các bạn có thể tham khảo thêm ở các tài liệu về lập trình hệ thống.

1.3. CÁC KHÁI NIỆM

1.3.1. Điểm (Pixel)

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

Ví du: 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, ta dùng hàm $0C ngắt $10:

```
Procedure PutPixel(Col,Row:Word; Color:Byte);
 Var reg:Registers;
 Begin
   reg.ah:=$0C;
   reg.al:=Color;
```

```
    reg.bh:=0;
    reg.cx:=Col;
    reg.dx:=Row;
    Intr($10,reg);
End;
```

- 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 bắt đầu từ địa chỉ segment A000h và địa chỉ offset được tính theo công thức: **Offset = y*320 + x**.

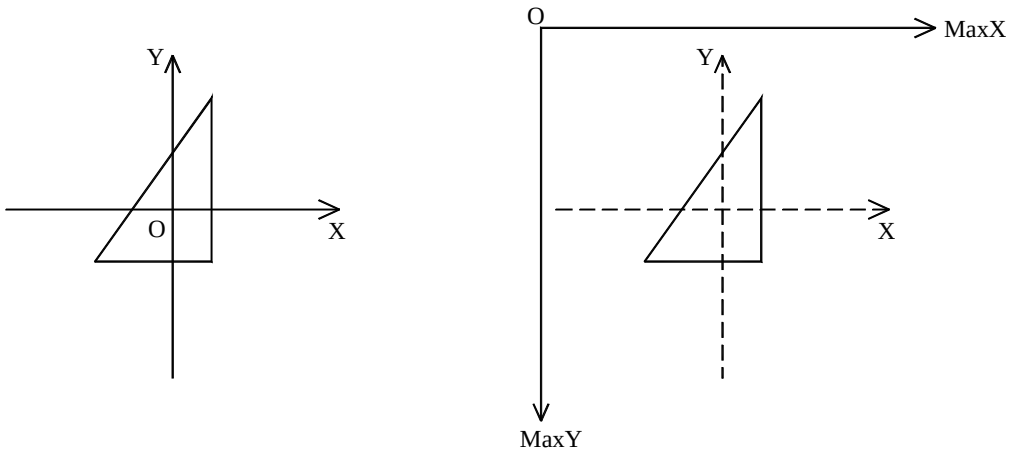
Ta có thể viết thủ tục như sau:

```
Procedure PutPixel(x,y:Word; Color:Byte);
 Var Offset:Word;
 Begin
  Offset:=(y shl 8) + (y shl 6) + x;
  Mem[$A000:Offset]:=Color;
 End;
```

1.3.2. Các biểu diễn tọa độ

Hầu hết các chương trình đồ họa đều dùng hệ tọa độ Decartes (Hình 1.1).

Ta biến đổi:



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

Hình 1.1

1.3.3. 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.

1.4. CÁC THUẬT TOÁN VẼ ĐOẠN THẲNG

1.4.1. 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.

1.4.2. Thuật toán DDA (Digital differential analyzer)

Xét đường thẳng có hệ số góc $0<m\leq1$(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: ta 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:

$y_{k+1} = y_k - m$ $(0<m\leq1, \Delta x= -1)$

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

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 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 thủ tục vẽ đoạn thẳng theo thuật toán DDA như sau:

```
Procedure DDALine(x1,y1,x2,y2:Integer);
var dx,dy,step,i:integer;
```

```
    xInc,yInc,x,y:real;
Begin
  dx:=x2-x1; dy:=y2-y1;
  If abs(dx)>abs(dy) Then step:=abs(dx)
   else step:=abs(dy);
  xInc:=dx/step;
  yInc:=dy/step;
  x:=x1;
  y:=y1;
  Putpixel(round(x),round(y),15);
  for i:=1 to step do
   Begin
     x:=x+xInc;
     y:=y+yInc;
     Putpixel(round(x),round(y),15);
   End;
End;
```

1.4.3. Thuật toán Bresenham

Phương trình đường thẳng có thể phát biểu dưới dạng: y = m.x + b (1)

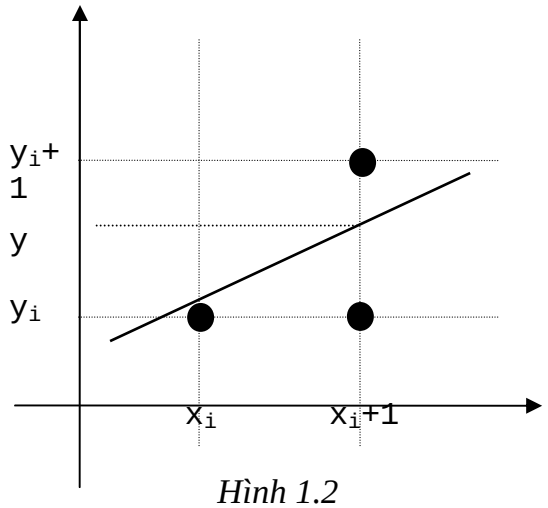
Phương trình đường thẳng qua 2 điểm:

$$\frac{x - x1}{x2 - x1} = \frac{y - y1}{y2 - y1} \quad (*)$$

Đặt $\Delta x = x2 - x1$

$\Delta y = y2 - y1$

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



Hình 1.2

Suy ra $m = \frac{\Delta y}{\Delta x}$ nên $\Delta y = m.\Delta x$ (2)

$b = y1 - m.x1$ (3)

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

Giả sử điểm (xi,yi) đã được vẽ. Ta phải chọn điểm kế tiếp là:

$(x_i + 1, y_i)$ hoặc $(x_i + 1, y_i + 1)$ (Xem hình 1.2)

Xét khoảng cách giữa 2 điểm chọn với điểm nằm trên đường thực. Nếu khoảng cách nào bé hơn thì ta lấy điểm đó.

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 $p_i = \Delta x(d_1 - d_2)$ và $C = 2\Delta y + \Delta x.(2b - 1)$

thì $p_i = 2\Delta y.x_i - 2\Delta x.y_i + C$ (4)

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

Suy ra:

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

$= 2\Delta y - 2\Delta x(y_{i+1} - y_i)$ (5)

(vì $x_{i+1} - x_i = 1$)

* ***Nhận xét***:

. Nếu $p_i < 0$: Chọn $\mathbf{y_{i+1} = y_i}$ Từ (5) $\Rightarrow p_{i+1} = p_i + 2\Delta y.$ $(d_1<d_2)$

. Nếu $p_i \geq 0$: Chọn $\mathbf{y_{i+1} = y_i + 1}$ Từ (5) $\Rightarrow p_{i+1} = p_i + 2\Delta y - 2\Delta x.$ $(d_1>d_2)$

Với điểm mút đầu tiên, theo (4) ta có:

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

Từ đó, ta có thể tóm tắt thuật toán vẽ đường thẳng theo Bresenham cho trường hợp hệ số góc $0<m<1$ như sau:

- **Bước 1**: Nhập các điểm đầu mút. Điểm đầu mút bên trái chứa tọa độ (x1,y1), điểm đầu mút bên phải chứa tọa độ (x2,y2).
- **Bước 2**: Điểm được chọn để vẽ đầu tiên là (x1,y1).
- **Bước 3**: Tính $\Delta x = |x2 - x1|$, $\Delta y = |y2 - y1|$ và $P_1 = 2\Delta y - \Delta x$

  Nếu $p_i < 0$ thì điểm kế tiếp là $(x_i + 1, y_i)$

  Ngược lại: điểm kế tiếp là $(x_i + 1, y_i + 1)$

- **Bước 4**: Tiếp tục tăng x lên 1 Pixel. Ở vị trí x_i +1, ta tính:

  $p_{i+1} = p_i + 2\Delta y$ nếu $p_i < 0$

  $p_{i+1} = p_i + 2.(\Delta y - \Delta x)$ nếu $p_i \geq 0$

  Nếu $p_{i+1} < 0$ thì ta chọn toạ độ y kế tiếp là y_{i+1}

  Ngược lại thì ta chọn y_{i+1} +1

- **Bước 5**: Lặp lại bước 4 cho đến khi x = x2.

Sau đây là thủ tục cài đặt thuật toán:

```
Procedure LINE(x1,y1,x2,y2:integer); { 0<m<1}
var dx,dy,x,y,p,c1,c2,xMax:integer;
Begin
  dx:=abs(x1-x2);
  dy:=abs(y1-y2);
  c1:=2*dy;
  c2:=2*(dy-dx);
  p:=2*dy-dx;
  if x1>x2 then
    begin
       x:=x2; y:=y2; xMax:=x1;
    end
  else
    begin
      x:=x1;y:=y1;xMax:=x2;
    end;
  putpixel(x,y,red);
  while x<xMax do
   begin
     x:=x+1;
     if p<0 then p:=p+c1
     else begin
            y:=y+1;
            p:=p+c2;
          end;
```

```
      putpixel(x,y,red);
    end;
end;
```

1.4.4. Thuật toán MidPoint

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

Thuật toán này đưa ra cách chọn điểm $S(x_i+1,y_i)$ hay $P(x_i+1,y_i+1)$ bằng cách so sánh điểm thực $Q(x_i+1,y)$ với điểm M (trung điểm của S và P).

- Nếu điểm Q nằm dưới điểm M thì chọn điểm S
- Ngược lại, chọn điểm P. (Xem hình 1.3)

Ta có dạng tổng quát của phương trình đường thẳng:

$$Ax + By + C = 0$$

với $A = y2 – y1$, $B = –(x2 – x1)$,

$$C = x2.y1 – x1.y2$$

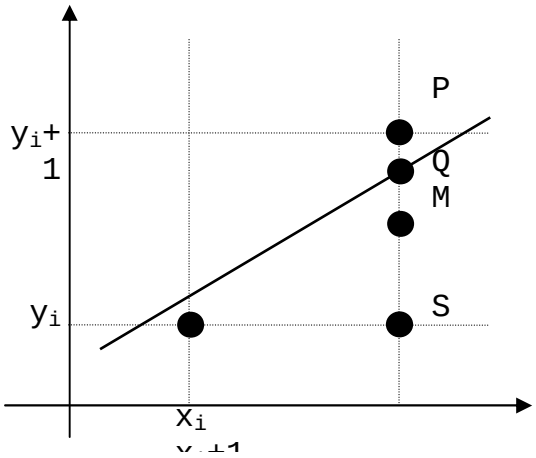
Đặt $F(x,y) = Ax + By + C$, ta có nhận xét:

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

Lúc này, việc chọn các điểm S hay P được đưa về việc xét dấu của:

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

- Nếu $p_i < 0 \Rightarrow$ M nằm trên đoạn thẳng $\Rightarrow$ Q nằm dưới M $\Rightarrow$ Chọn S
- Nếu $p_i \geq 0 \Rightarrow$ M nằm dưới đoạn thẳng $\Rightarrow$ Q nằm trên M $\Rightarrow$ Chọn P



Hình 1.3

Mặt khác:

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

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

nên

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

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

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

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

Suy ra:

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

***Nhận xét:**

. Nếu $p_i < 0$: Chọn điểm S: $\mathbf{y_{i+1} = y_i}$ Từ (*) suy ra $p_{i+1} = p_i + A$

. Nếu $p_i \geq 0$: Chọn điểm P: $\mathbf{y_{i+1} = y_i + 1}$ Từ (*) suy ra $p_{i+1} = p_i + A + B$

Với điểm mút đầu tiên, ta có:

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

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

Thuật toán MidPoint cho kết quả tương tự như thuật toán Bresenham.

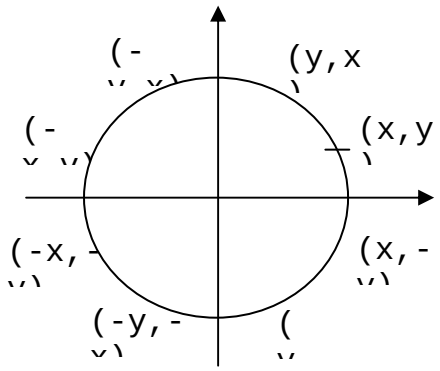
1.5. THUẬT TOÁN VẼ ĐƯỜNG TRÒN

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

* *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) (Hình 1.4) Vì vậy, ta chỉ cần vẽ một phần tám cung tròn rồi lấy đối xứng qua gốc O và 2 trục toạ độ thì ta có được toàn bộ đường tròn.

1.5.1. Thuật toán Bresenham

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

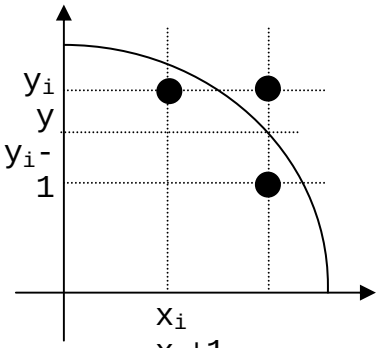
Từ phương trình: $x^2 + y^2 = R^2$

ta tính được giá trị y thực ứng với x_i +1 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

Suy ra:

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

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

Từ (2) và (3) ta có:

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

$\Rightarrow p_{i+1} = p_i + 4x_i + 6 + 2.(y^2_{i+1} - y_i^2) - 2.(y_{i+1} - y_i)$ (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 (0,R), theo (2) ta có: p1 = 3 - 2R

Tóm lại: Ta có thuật toán vẽ đường tròn:

- **Bước 1**: Chọn điểm đầu cần vẽ (x1,y1) = (0,R)
- **Bước 2**: Tính P đầu tiên: p1 = 3 - 2R

  Nếu p < 0: chọn điểm kế tiếp là $(x_i + 1, y_i)$. Ngược lại chọn điểm $(x_i + 1, y_i - 1)$
- **Bước 3**: x:=x + 1, tính lại p:

  Nếu $p_i < 0$: $p_{i+1} = p_i + 4x_i + 6$. Ngược lại: $p_{i+1} = p_i + 4.(x_i - y_i) + 10$

  Khi đó:

  Nếu $p_{i+1} < 0$: chọn điểm kế tiếp là $(x_i + 1, y_{i+1})$. Ngược lại chọn điểm $(x_i + 1, y_{i+1} - 1)$
- **Bước 4**: Lặp lại bước 3 cho đến khi x = y.

Sau đây là thủ tục để cài đặt thuật toán:

```
Procedure Circle(x0,y0,r:Integer);
 Var p,x,y:Integer;
  Procedure VeDiem;
    Begin
      PutPixel( x0 + x , y0 + y , color);
      PutPixel( x0 - x , y0 + y , color);
      PutPixel( x0 + x , y0 - y , color);
      PutPixel( x0 - x , y0 - y , color);
```

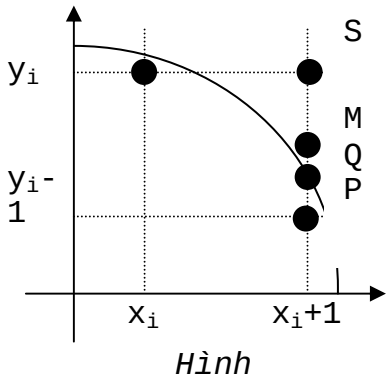
```
      PutPixel( x0 + y , y0 + x , color);
      PutPixel( x0 - y , y0 + x , color);
      PutPixel( x0 + y , y0 - x , color);
      PutPixel( x0 - y , y0 - x , color);
    End;
Begin
  x:=0; y:=r;
  p:=3 - 2*r;
  While x<=y do
   Begin
     VeDiem;
     If p<0 then p:=p + 4*x + 6
     Else
       Begin
         p:=p + 4*(x-y) + 10;
         y:=y-1;
       End;
     x:=x+1;
   End;
End;
```

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:

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



Hình 1.6

Lúc này, việc chọn các điểm $S(x_i+1,y_i)$ hay $P(x_i+1,y_i-1)$ được đưa về việc xét dấu của:

$$p_i = F(M) = F(x_i + 1, y_i - \frac{1}{2})$$ (Hình 1.6)

- Nếu $p_i < 0 \Rightarrow$ M nằm trong đường tròn $\Rightarrow$ Q gần S hơn $\Rightarrow$ Chọn S
- Nếu $p_i \geq 0 \Rightarrow$ M nằm ngoài đường tròn $\Rightarrow$ Q gần P hơn $\Rightarrow$ Chọn P

Mặt khác:

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

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

nên

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

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

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

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

Suy ra:

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

Nhận xét:

. Nếu $p_i < 0$: Chọn điểm S : $\mathbf{y_{i+1} = y_i}$ Từ (*) $\Rightarrow$ $\mathbf{p_{i+1} = p_i + 2x_i + 3}$

. Nếu $p_i \geq 0$: Chọn điểm P: $\mathbf{y_{i+1} = y_i - 1}$ Từ (*) $\Rightarrow$ $\mathbf{p_{i+1} = p_i + 2(x_i - y_i) + 5}$

Với điểm đầu tiên (0,R), ta có:

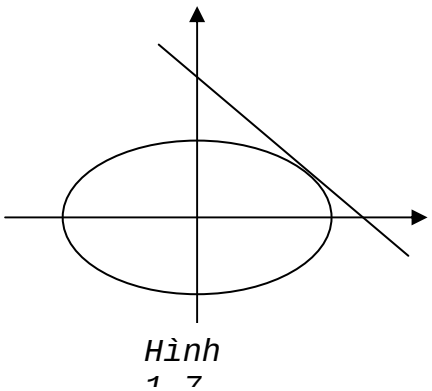
$$p_1 = F(x_1 + 1, y_1 - \frac{1}{2}) = F(1, R - \frac{1}{2}) = 1 + (R - \frac{1}{2})^2 - R^2 = \frac{5}{4} - R$$

1.6. THUẬT TOÁN VẼ ELLIPSE

Để đơn giản, ta chọn Ellipse 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 ta 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 = -1 (Hình 1.7).

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

Ở đây, ta chỉ 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:

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

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

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

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

Suy ra:

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

****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 $y_{i+1} = y_i - 1$

$$(**) \Rightarrow p_{i+1} = p_i + 2.\frac{b^2}{a^2}.(2x+3) - 4y_i$$

Với điểm đầu tiên (0,b), ta có:

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

Từ đó, ta có thủ tục vẽ Ellipse như sau:

```
Procedure Ellipse(xc,yc,a,b:Integer;Color:Byte);
Var p,a2,b2:real;
    x,y:integer;
 (*-------------------*)
  Procedure VeDiem;
   Begin
     PutPixel(xc+x,yc+y,Color);
     PutPixel(xc-x,yc+y,Color);
     PutPixel(xc-x,yc-y,Color);
     PutPixel(xc+x,yc-y,Color);
   End;
 (*-------------------*)
Begin
  a2:=a*a;
  b2:=b*b;
  {Nhanh 1}
  x:=0; y:=b;
  p:=2*b2/a2 - 2*b + 1;
  While (b2/a2)*(x/y)<1 do
    Begin
      VeDiem;
      If p<0 then p:=p + 2*(b2/a2)*(2*x+3)
      else Begin
             p:=p - 4*y + 2*(b2/a2)*(2*x+3);
             y:=y-1;
           End;
      x:=x+1;
    End;
  {Nhanh 2}
  y:=0; x:=a;
  p:=2*(a2/b2) - 2*a + 1;
  While (a2/b2)*(y/x)<=1 do
```

```
    Begin
      VeDiem;
      If p<0 then p:=p + 2*(a2/b2)*(2*y+3)
      else Begin
             p:=p - 4*x + 2*(a2/b2)*(2*y+3);
             x:=x-1;
           End;
      y:=y+1;
    End;
End;
```

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

```
Procedure MidEllipse(xc,yc,a,b:Integer;Color:Byte);
Var p,a2,b2:real;
    x,y:Integer;
 (*-------------------*)
  Procedure VeDiem;
   Begin
     PutPixel(xc+x,yc+y,Color);
     PutPixel(xc-x,yc+y,Color);
     PutPixel(xc-x,yc-y,Color);
```

```
     PutPixel(xc+x,yc-y,Color);
   End;
 (*-------------------*)
Begin
   a2:=a*a;
   b2:=b*b;
  {Nhanh 1}
  x:=0; y:=b;
  Vediem;
  p:=b2 - a2*b + 0.25*a2;
  While (b2/a2)*(x/y)<1 do
    Begin
      x:=x+1;
      If p<0 Then p:=p + b2 + 2*b2*x
      else begin
            y:=y-1;
            p:=p + b2 + 2*b2*x - 2*a2*y;
           end;
      Vediem;
    End;
  {Nhanh 2}
  p:=b2*(x+0.5)*(x+0.5) + a2*(y-1)*(y-1)- a2*b2 ;
  While y>0 do
    Begin
      y:=y-1;
      If p>0 Then p:=p + a2 - 2*a2*y
      else begin
            x:=x+1;
            p:=p + a2 + 2*b2*x - 2*a2*y;
           end;
      Vediem;
    End;
```

```
End;
```

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 đầu đến cuối 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 số f(x) = Sin(x)

```
Uses crt,Graph;
Var dau,cuoi:real;
     Gd,Gm:Integer;
Function F(x:real):real;
 Begin
   F:=Sin(x);
 End;
Procedure VeHinhSin(ChukyDau,ChuKyCuoi:real);
```

```
 var x1,y1,x2,y2:integer;
     a,x,k:real;
     x0,y0:word;
 Begin
   x0:=GetMaxX div 2;
   y0:=GetMaxY div 2;
   K:=GetMaxX/30;
   a:=Pi/180;
   x:=ChuKyDau;
   x1:=x0 + Round(x*k);
   y1:=y0 - Round(F(x)*k);
   Moveto(x1,y1);
   While x<ChuKyCuoi do
     Begin
       x:=x+a;
       x2:=x0 + Round(x*k);
       y2:=y0 - Round(F(x)*k);
       LineTo(x2,y2);
     End;
 End;
BEGIN
 Gd:=0;
 InitGraph(Gd,Gm,'C:\BP\BGI');
 Dau:=-4*Pi; cuoi:=4*Pi;
 VeHinhSin(Dau,cuoi);
 repeat  until KeyPressed;
 CloseGraph;
END.
```

BÀI TẬP

1. Cho hai điểm A(20,10) và B(25,13). Hãy tính các giá trị Pi, xi, yi 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

Pi	?	?	?	?	?	?
xi	?	?	?	?	?	?
yi	?	?	?	?	?	?

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

3. Viết thủ tục **LineTo(x,y:Integer);** để vẽ đoạn thẳng từ vị trí hiện thời đến điểm có tọa độ (x,y).

4. Viết thủ tục **LineRel(dx,dy:Integer);** để vẽ đoạn thẳng từ vị trí hiện thời đến điểm mới cách điểm hiện thời một khoảng theo trục x là dx và theo trục y là dy.

5. Cài đặt thủ tục vẽ đường tròn theo thuật toán MidPoint.

6. Viết thủ tục **Arc(x0,y0,g1,g2:Integer; R:Word);** để vẽ cung tròn có tâm (x0,y0) bán kính R với góc bắt đầu là g1 và góc kết thúc là g2.

7. Viết thủ tục **Sector(x0,y0,g1,g2:Integer; Rx,Ry:Word);** 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.

8. Viết thủ tục **DrawPoly(P:Array; n:Byte; xc,yc,R:Word);** để vẽ một đa giác đều có n đỉnh lưu trong mảng P nội tiếp trong đường tròn tâm (xc,yc) bán kính R.

9. Viết thủ tục **Circle3P(A,B,C:ToaDo2D);** để vẽ đường tròn đi qua 3 điểm A,B,C.

10. Viết thủ tục **Arc3P(A,B,C:ToaDo2D);** để vẽ cung tròn đi qua 3 điểm A,B,C.

11. Vẽ đồ thị các hàm số sau:

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

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

12. 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 chương trình khảo sát và vẽ đồ thị các hàm số sơ cấp ở bài tập số 11.

CHƯƠNG 2

TÔ MÀU

2.1. GIỚI THIỆU VỀ CÁC HỆ MÀU

Giác quan của con người cảm nhận được các vật thể xung quanh thông qua các tia sáng màu tốt hơn rất 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.

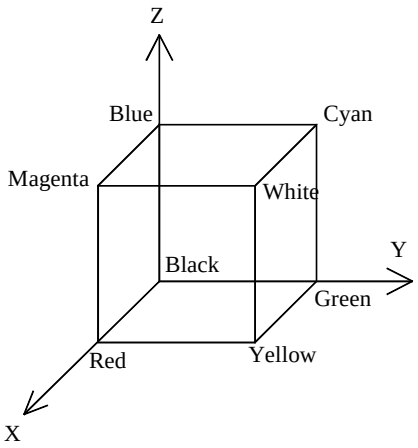
Ba mô hình màu được sử dụng và phát triển nhiều trong các phần cứng là: RGB - dùng với các màn hình CRT (Cathode ray bube), YIQ – dùng trong các hệ thống ti vi màu băng tần rộng và CMY - sử dụng trong một số thiết bị in màu.

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.

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).



Hình 2.1. Hệ màu RGB

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 củ

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

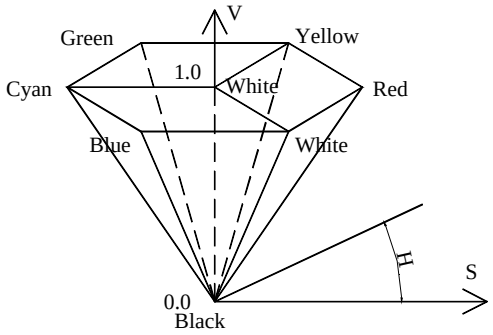
Ma trận nghịch đảo của ma trận biến đổi RGB thành hệ YIQ được sử dụng cho phép biến đổi từ hệ YIQ thành RGB.

2.1.4. Hệ HSV (Hue, Saturation, Value)

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 $0 \rightarrow 1$



Hình 2.2. Hệ màu HSV

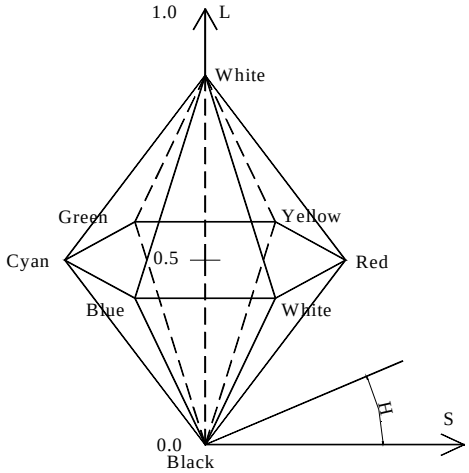
Ví dụ:

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).

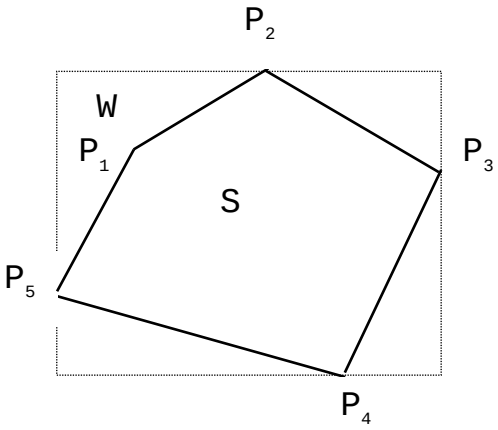


Hình 2.3. Hệ màu HSL

2.2. CÁC THUẬT TOÁN TÔ MÀU

2.2.1. Bài toán

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



Hình 2.4

* 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:

$$S=\frac{1}{2}\left|\sum_{i=1}^{n}(x_iy_{i+1}-x_{i+1}y_i)\right|$$

2.2.2.2. Trường hợp tổng quát (Thuật toán Jordan)

Lấy P(x, y) ∈ W, kẻ nửa đường thẳng ΔP xuất phát từ P và không đi qua các đỉnh của đa giác S.

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.

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

+ Nếu x=x_i thì xét 2 cạnh có 1 đầu là P_i:

Nếu y<y_i thì

- Nếu cả 2 đầu kia ở cùng một phía của ΔP thì ta tính ΔP cắt cả 2 cạnh.
- Ngược lại : ΔP cắt 1 cạnh.

+ Ngược lại:

- Nếu $x>Max(x_i,x_{i+1})$ hoặc $x<Min(x_i,x_{i+1})$ thì ΔP không cắt Ci
- 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:

```
Type   ToaDo=record
           x,y:integer;
          End;
       Mang=array[0..30] of ToaDo;

Function SOGIAODIEM(a:Mang; n:Byte):Integer;
 var dem,i,j,s:Integer;
 Begin
   dem:=0;
   for i:=1 to n do     { Tim so giao diem }
    begin
     if i=n then j:=1 else j:=i+1;
     if i=1 then s:=n else s:=i-1;
     if x=a[i].x then
      begin
       if y<a[i].y then
        if (x<=Min(a[s].x ,a[j].x))OR
      (x>=Max(a[s].x,a[j].x)) then dem:=dem+2
        else dem:=dem+1;
      end
     else
       if (x>Min(a[i].x,a[j].x)) and
     (x<Max(a[j].x,a[i].x)) then
        if y<=Min(a[i].y,a[j].y) then dem:=dem+1
        else if y <= (x-a[j].x)*(a[i].y-a[j].y)/
       (a[i].x-a[j].x)+a[j].y then dem:=dem+1;
    end;
 SOGIAODIEM:=dem;
End;
```

2.2.3. Thuật toán tô màu theo dòng quét (Scanline)

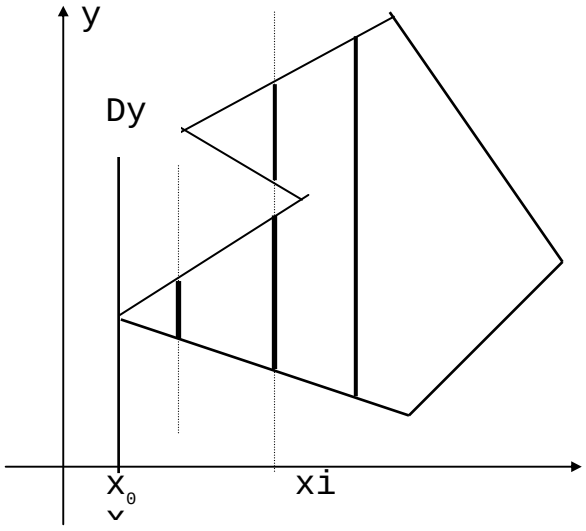
Đặt $x_0 = Min(x_i)$, $i \in [1,n]$.

Bước 1: Kẻ Dy//0y đi qua x_0 (hình 2.5).

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 Ci tại 2 điểm P_i và P_{i+1}.

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).



Hình 2.5

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.

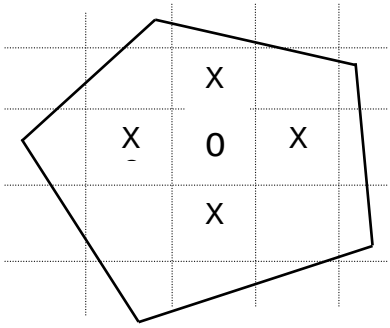
2.2.4. Thuật toán tô màu theo vết dầu loang

Lấy $P(x,y) \in S$, tô màu P.

Xét các điểm lân cận của P (Hình 2.6).

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

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



Hình 2.6

```
ProcedureTOLOANG(x,y:Integer; Color:Word);
  Begin
    If (P(x,y)∈S)and(P(x,y)chưa tô) Then
      Begin
        PutPixel(x,y,Color);
        TOLOANG(x+1,y,Color);
        TOLOANG(x-1,y,Color);
```

```
        TOLOANG(x,y+1,Color);
        TOLOANG(x,y-1,Color);
      End;
  End;
```

BÀI TẬP

1. Viết hàm **DienTich(P:Array; n:Byte);** để tính diện tích của đa giác lồi có n đỉnh lưu trong mảng P.

2. Viết hàm **KiemTra(x,y:Integer; P:Array; n:Byte):Boolean;** để kiểm tra điểm (x,y) nằm trong hay ngoài đa giác có n đỉnh được lưu trong mảng P theo hai cách:

- Dùng công thức tính diện tích đa giác (đối với đa giác lồi).
- Dùng thuật toán Jordan (đối với đa giác bất kỳ).

2. Viết chương trình cài đặt thuật toán tô màu một đa giác theo thuật toán Scanline.

3. Viết chương trình cài đặt thuật toán tô màu một đa giác theo vết dầu loang theo hai phương án:

a. Đệ qui.

b. Khử đệ qui.

4. Viết thủ tục **FillRec(x1,y1,x2,y2:Integer; color:Byte);** để tô màu hình chữ nhật xác định bởi 2 đỉnh (x1,y1) và (x2,y2).

5. Viết thủ tục **FillEllipse(x,y,Rx,Ry:Integer; color:Byte);** để tô màu Ellipse có tâm (x,y) và bán kính theo hai trục là Rx và Ry.

6. Viết thủ tục **FillSector(x,y,Rx,Ry,g1,g2:Integer; color:Byte);** 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 thủ tục **Donut(x,y,Rmin,Rmax:Integer; color:Byte);** để tô màu hình vành khăn có tâm (x,y) và bán kính hai đường tròn tương ứng là Rmin và Rmax.

Bài tập lớn: Xây dựng một thư viện đồ họa **MYGRAPH** tương tự như thư viện **GRAPH.TPU** của Turbo Pascal.

CHƯƠNG III

XÉN HÌNH

3.1. ĐẶT VẤN ĐỀ

Cho một miền D $\subset R^n$ và F là một hình trong R^n ($F \subset R^n$). Ta gọi $F \cap D$ là hình có được từ F bằng cách xén vào trong D và ký hiệu là **$Clip_D(F)$**.

Bài toán đặt ra là xác định **$Clip_D(F)$**.

3.2. XÉN ĐOẠN THẲNG VÀO VÙNG HÌNH CHỮ NHẬT CỦA R^2

3.2.1. Cạnh của hình chữ nhật song song với các trục tọa độ

Lúc này:

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

và F là đoạn thẳng nối 2 điểm (x1,y1), (x2,y2) nên phương trình của F là:

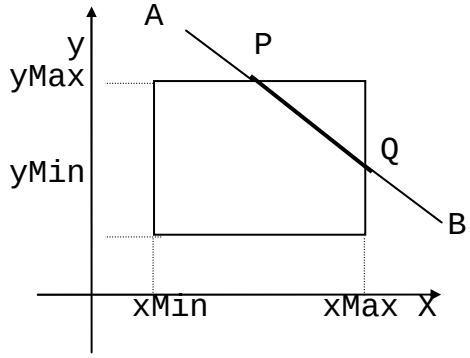
$$\begin{cases} x = x1 + (x2 - x1).t \\ y = y1 + (y2 - y1).t \end{cases} \qquad t \in [0,1]$$

Do đó, F có thể được viết dưới dạng:

$$F = \{(x,y) \in R^2 \mid x = x1 + (x2 - x1).t;\ y = y_1 + (y2 - y1).t;\ 0 \le t \le 1\}$$

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

$$\begin{cases} Xmin \le x1 + (x2 - x1).t \le Xmax \\ Ymin \le y1 + (y2 - y1).t \le Ymax \\ 0 \le t \le 1 \end{cases}$$



Hình 3.1

Gọi N là tập nghiệm của hệ phương trình trên.

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

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

Gọi P, Q là 2 giao điểm xác định bởi:

$$\begin{cases} Px = x1 + (x2 - x1).t1 \\ Py = y1 + (y2 - y1).t1 \end{cases} \qquad \begin{cases} Qx = x1 + (x2 - x1).t2 \\ Qy = y1 + (y2 - y1).t2 \end{cases}$$

thì: **$Clip_D(F) = PQ$** (Hình 3.1)

3.2.1.1. Thuật toán Cohen - Sutherland

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

1001	1000	1010
0001	0000	0010
0101	0100	0110

Hình 3.2

- Xét điểm $P \in R^2$:

$$Pleft = \begin{cases} 1 & \text{nếu } P_x < X_{min} \\ 0 & \text{Ngược lại} \end{cases}$$

$$PRight = \begin{cases} 1 & \text{nếu } P_x > X_{max} \\ 0 & \text{Ngược lại} \end{cases}$$

$$PBelow = \begin{cases} 1 & \text{nếu } P_y < Y_{min} \\ 0 & \text{Ngược lại} \end{cases}$$

$$PAbove = \begin{cases} 1 & \text{nếu } P_y > Y_{max} \\ 0 & \text{Ngược lại} \end{cases}$$

- 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) ≠ 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.

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

  Giả sử Mã(A) ≠ 0000 $\Leftrightarrow$ A nằm ngoài hình chữ nhật.

  - Nếu Aleft = 1 : thay A bởi điểm nằm trên đoạn AB và cắt cạnh trái (nối dài) của hình chữ nhật.

- Nếu Aright = 1: thay A bởi điểm nằm trên đoạn AB cắt cạnh phải (nối dài) của hình chữ nhật.
- Nếu ABelow = 1: thay A bởi điểm nằm trên đoạn AB và cắt cạnh dưới (nối dài) của hình chữ nhật.
- Nếu AAbove = 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/

3.2.1.2. 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 một đoạn hoàn toàn nằm ngoài hình chữ nhật.*

- 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.

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

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 **$Clip_D(F) = \varnothing$** . Ngược lại, áp dụng ii/ ta có:

$Clip_D(MA) = MA_1$

$Clip_D(MB) = MB_1$

Suy ra: **$Clip_D(F) = A_1B_1$**

3.2.1.3. Thuật toán Liang - Barsky

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

$p1 = -\Delta x$ $q1 = x1 - xMin$

$p2 = \Delta x$ $q2 = xMax - x1$

$p3 = -\Delta y$ $q3 = y1 - yMin$

$p4 = \Delta y$ $q4 = yMax - y1$

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

$$\begin{cases} P_k.t \leq Q_k, \ k = 1..4 \\ 0 \leq t \leq 1 \end{cases}$$

Xét các trường hợp sau:

i/ ∃k: Pk = 0 và Qk < 0: (Đường thẳng song song với các biên và nằm ngoài vùng hình chữ nhật)

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

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

Đặt $K_1 = \{k \mid Pk > 0\}$

$K_2 = \{k \mid Pk < 0\}$

$$u_1 = Max(\{\frac{Q_k}{P_k} \mid k \in K_2\} \cup \{0\})$$

$$u_2 = Min(\{\frac{Q_k}{P_k} \mid k \in K_1\} \cup \{1\})$$

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

Ngược lại: Gọi P, Q là 2 điểm thỏa

$$\begin{cases} Px = x1 + \Delta x.u_1 \\ Py = y1 + \Delta y.u_1 \end{cases} \text{ và } \begin{cases} Qx = x1 + \Delta x.u_2 \\ Qy = y1 + \Delta y.u_2 \end{cases}$$

thì **$Clip_D(F) = PQ$**

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

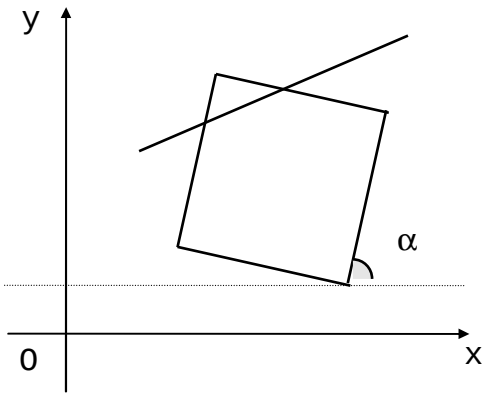
Ta dùng phép quay trục tọa độ để đưa bài toán về trường hợp các cạnh của hình chữ nhật song song với các trục tọa độ (hình 3.3).

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

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

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

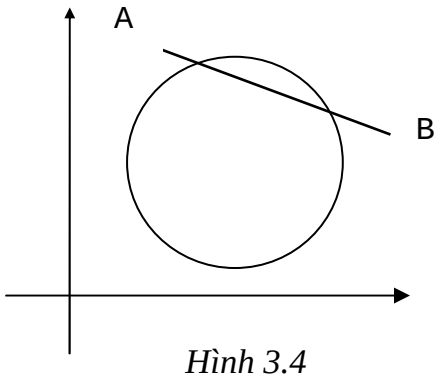
với $R = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix}$



Hình 3.3

3.3. XÉN ĐOẠN THẲNG VÀO HÌNH TRÒN

Giả sử ta có đường tròn tâm O(xc,yc) bán kính R và đoạn thẳng cần xén là AB với A(x1,y1), B(x2,y2) (Hình 3.4).



Hình 3.4

*** Thuật toán:**

- Tính d(O,AB)
- Xét các trường hợp:

  i/ Nếu d > R thì $\mathbf{Clip_D(F) = \varnothing}$

  ii/ Nếu d = R thì $\mathbf{Clip_D(F) = A_0}$ với A_0 là chân đường vuông góc hạ từ O xuống AB.

  iii/ Nếu d < R thì xét các trường hợp sau:

  - ♦ (OA < R) AND (OB < R) thì ClipD(F) = AB
  - ♦ Nếu một điểm nằm trong và điểm kia nằm ngoài hình tròn, chẳng hạn OA<R và OB>R thì $\mathbf{Clip_D(F) = AI}$ với I là giao điểm duy nhất giữa AB và đường tròn.
  - ♦ (OA > R) AND (OB > R) thì ClipD(F) = IJ với I, J là hai giao điểm của AB với đường tròn.

Sau đây là phương pháp tìm giao điểm giữa đoạn thẳng và đường tròn:

◊ Phương trình đường tròn: $(x - xc)^2 + (y - yc)^2 = R^2$ (1)

◊ Phương trình đoạn AB: $\begin{cases} x = x1 + (x2 - x1).\lambda \\ y = y1 + (y2 - y1).\lambda \\ 0 \le \lambda \le 1 \end{cases}$ (2)

◊ Thay (2) vào (1) ta suy ra: $\lambda = \dfrac{-a \pm \sqrt{a^2 - bc}}{b}$

Trong đó:

$a = \Delta x.(x1 - x_c) + \Delta y.(y1 - yc)$

$b = (\Delta x)^2 + (\Delta y)^2$

$c = (x1 - x_c)^2 + (y1 - yc)^2 - R^2$

$\Delta x = x2 - x1$

$\Delta y = y2 - y1$

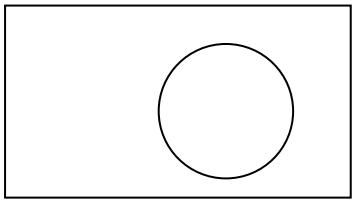
◊ Dựa vào điều kiện $0 \leq \lambda \leq 1$ để chọn giao điểm.

3.4. XÉN ĐƯỜNG TRÒN VÀO HÌNH CHỮ NHẬT CÓ CÁC CẠNH SONG SONG VỚI TRỤC TỌA ĐỘ

Lúc này:

$D = \{(x,y)|\ xMin \leq x \leq xMax\ ;\ yMin \leq y \leq yMax\ \}$

$F = \{\ (x,y)|\ (x - x_C)^2 + (y - y_C)^2 = R^2\}$



Hình 3.5

*Trước hết, ta kiểm tra các trường hợp đặc biệt sau:

i/ Nếu $xMin \leq x_C - R$; $x_C + R \leq xMax$;

$yMin \leq y_C - R$; $y_C + R \leq yMax$;

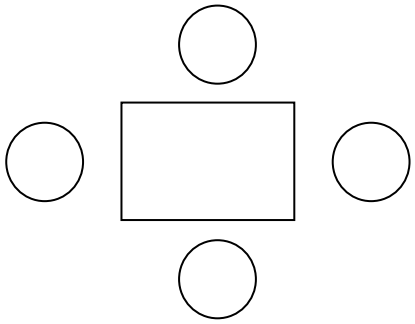
thì $\mathbf{Clip_D(F) = F}$ (Hình 3.5)

ii/ Nếu $x_C + R < xMin$

hoặc $x_C - R > xMax$

hoặc $y_C + R < yMin$

hoặc $y_C - R > yMax$



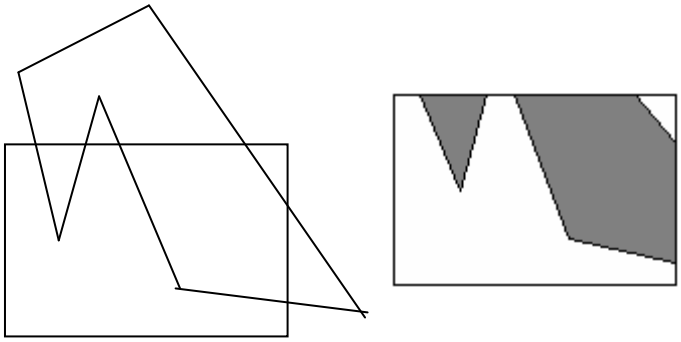
Hình 3.6

thì $\mathbf{Clip_D(F) = \varnothing}$ (Hình 3.6)

*Xét trường hợp còn lại: Tìm các giao điểm của F và D. Sắp xếp các giao điểm đó theo chiều ngược kim đồng hồ.

- Các cung tròn được tạo bởi 2 giao điểm liên tiếp sẽ hoàn toàn nằm trong D hoặc hoàn toàn nằm bên ngoài D.
- Để xác định các cung này nằm trong hay ngoài D, ta chỉ cần lấy trung điểm M của cung đó. Nếu $M \in D$ thì cung đó nằm trong D, ngược lại thì nó nằm ngoài D.

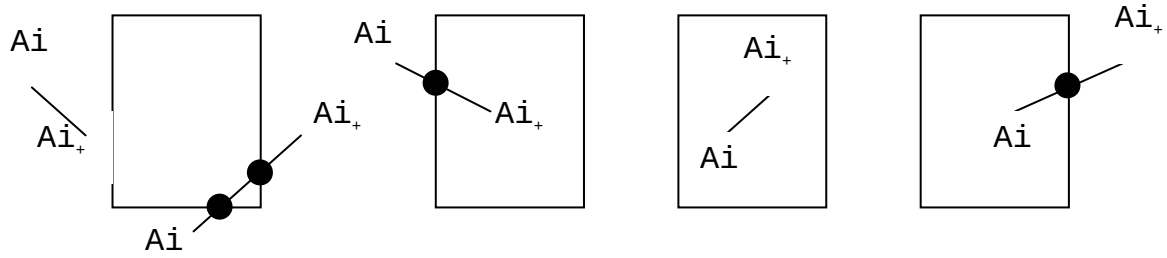
3.5. XÉN ĐA GIÁC VÀO HÌNH CHỮ NHẬT



Hình 3.7. 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.



Hình 3.8. Các trường hợp cần xét

ii/ Trường hợp ngược lại:

- Xuất phát từ một đỉnh nằm ngoài hình chữ nhật, ta chạy theo dọc biên của đa giác. Với mỗi cạnh của đa giác, ta có các trường hợp sau:

- 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.

- Ai 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.

- 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.

BÀI TẬP

1. Viết hàm **MA(P:ToaDo):Byte;** để đánh mã cho điểm P.

2. Cài đặt thuật toán xén một đoạn thẳng vào một hình chữ nhật theo các thuật toán: Liang-Barsky, Cohen-Sutherland, chia nhị phân.

3. Cài đặt thuật toán xén một đoạn thẳng vào một hình tròn.

4.Cài đặt thuật toán xén một đa giác vào một vùng hình chữ nhật.

CHƯƠNG IV

CÁC PHÉP BIẾN ĐỔI

4.1. CÁC PHÉP BIẾN ĐỔI TRONG MẶT PHẲNG

4.1.1. Cơ sở toán học

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

x’ = Ax + Cy + trx

y’ = Bx + Dy + try

Dưới dạng ma trận, hệ này có dạng:

$$\mathbf{(x'\ y') = (x\ y).}\begin{pmatrix} A & B \\ C & D \end{pmatrix} \mathbf{+ (trx\ \ try)}$$

Hay viết gọn hơn: **X’ = X.M + tr**

với X’=(x’,y’); X=(x,y); tr=(trx,try) - vector tịnh tiến;

M = $\begin{pmatrix} A & B \\ C & D \end{pmatrix}$ - ma trận biến đổi.

4.1.1.1. Phép đồng dạng

Ma trận của phép đồng dạng là:

$$M = \begin{pmatrix} A & 0 \\ 0 & D \end{pmatrix} \quad \Leftrightarrow \begin{cases} x' = Ax \\ y' = Dy \end{cases}$$

Cho phép ta phóng to hay thu nhỏ hình theo một hay hai chiều.

4.1.1.2. Phép đối xứng

Đây là trường hợp đặc biệt của phép đồng dạng với A và D đối nhau.

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

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

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

4.1.1.3. Phép quay

Ma trận tổng quát của phép quay là $R = \begin{pmatrix} Cos(\alpha) & Sin(\alpha) \\ -Sin(\alpha) & Cos(\alpha) \end{pmatrix}$

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.

4.1.1.4. Phép tịnh tiến

Biến đổi (x,y) thành (x',y') theo công thức sau

x' = x + M

y' = y + N

Để thuận tiện biểu diễn dưới dạng ma trận, ta có thể biểu diễn các tọa độ dưới dạng tọa độ thuần nhất (Homogen):

$$(x \;\; y \;\; 1).\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ M & N & 1 \end{pmatrix} = (x + M \qquad y + N \qquad 1)$$

4.1.1.5. Phép biến dạng

Ma trận tổng quát là: $M = \begin{pmatrix} 1 & g \\ h & 1 \end{pmatrix}$

Trong đó:

g = 0: biến dạng theo trục x.

h = 0: biến dạng theo trục y.

4.1.1.6. Hợp của các phép biến đổi

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

Ví dụ: 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:

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

4.2. Ví dụ minh họa

Viết chương trình mô phỏng phép quay một tam giác quanh gốc tọa độ.

```
Uses crt,Graph;
Type ToaDo=Record
       x,y:real;
      End;
 var k,Alpha,goc:real;
     P,PP,PPP,P1,P2,P3:ToaDo;
     x0,y0:word;
     ch:char;
Procedure VeTruc;
  Begin
     Line(GetMaxX div 2,0,GetMaxX div 2,GetMaxY);
     Line(0,GetMaxY div 2,GetMaxX,GetMaxY div 2);
  End;
Procedure VeHinh(P1,P2,P3:ToaDo);
  Begin
   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),
```

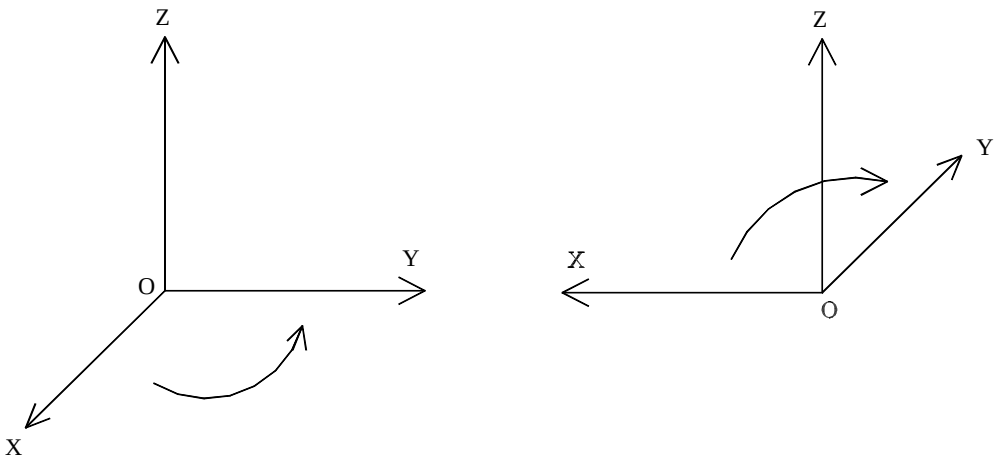
```
        x0+Round(P3.x*k),y0- Round(P3.y*k));
   Line(x0+Round(P3.x*k),y0-Round(P3.y*k),
        x0+Round(P1.x*k),y0- Round(P1.y*k));
  End;
Procedure QuayDiem(P:ToaDo;Alpha:real; var PMoi:ToaDo);
  Begin
    PMoi.x:=P.x*cos(Alpha)-P.y*sin(Alpha);
    PMoi.y:=P.x*sin(Alpha)+P.y*cos(Alpha);
  End;
Procedure QuayHinh(P1,P2,P3:ToaDo;Alpha:real;
                   var P1Moi,P2Moi,P3Moi:ToaDo);
  Begin
   QuayDiem(P1,Alpha,P1Moi);
   QuayDiem(P2,Alpha,P2Moi);
   QuayDiem(P3,Alpha,P3Moi);
  End;
BEGIN
  ThietLapDoHoa;
  x0:=GetMaxX div 2;
  y0:=GetMaxY div 2;
  k:=GetMaxX/50;
  Vetruc;
  P.x:=5; P.y:=3; PP.x:=2; PP.y:=6; PPP.x:=6; PPP.y:=-4;
  P1.x:=5; P1.y:=3; P2.x:=2; P2.y:=6; P3.x:=6; P3.y:=-4;
  Alpha:=0; goc:=Pi/180;
  SetWriteMode(XORPut);
  VeHinh(P,PP,PPP);
  Repeat
    ch:=readkey;
    if ord(ch)=0 then ch:=readkey;
    case Upcase(ch) of
       #75: Begin
```

```
            VeHinh(P1,P2,P3);
            Alpha:=Alpha-goc;
            QuayHinh(P,PP,PPP,Alpha,P1,P2,P3);
            VeHinh(P1,P2,P3);
          End;
      #77: Begin
            VeHinh(P1,P2,P3);
            Alpha:=Alpha+goc;
            QuayHinh(P,PP,PPP,Alpha,P1,P2,P3);
            VeHinh(P1,P2,P3);
          End;
    End;
  Until ch=#27;
  CloseGraph;
END.
```

4.2. CÁC PHÉP BIẾN ĐỔI TRONG KHÔNG GIAN

4.2.1. 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

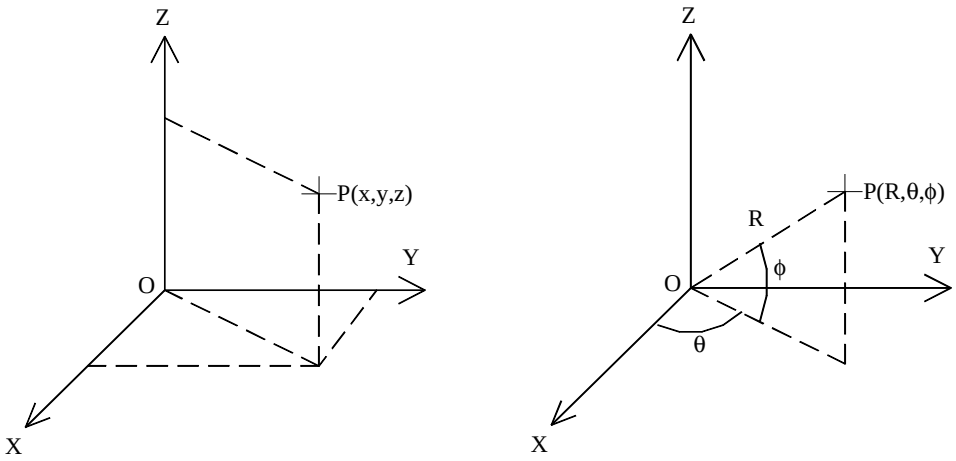
Hình 4.1

- 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 luôn định vị một điểm trong không gian qua hệ trực tiếp.

Trong hệ tọa độ trực tiếp, ta chia ra làm 2 loại sau:



Hệ tọa độ Descarter Hệ cầu

Hình 4.2

Ta có công thức chuyển đổi tọa độ từ hệ này sang hệ khác:

$$\mathbf{x = R.Cos(\theta).Cos(\Phi)}$$

$$\mathbf{y = R.Sin(\theta).Cos(\Phi)}$$

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

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

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).

4.2.2. Các công thức biến đổi

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=(trx,try,trz) - vector tịnh tiến

4.2.2.1. Phép thay đổi tỉ lệ

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

4.2.2.2. Phép đối xứng

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

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

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

4.2.2.3. Phép tịnh tiến

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

4.2.2.4. 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:

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

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

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

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: $R_X.R_Y \neq R_Y.R_X$

4.2.3. Ma trận nghịch đảo

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

Ma trận nghịch đảo của ma trận M ký hiệu 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.

- 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 $-\theta$.

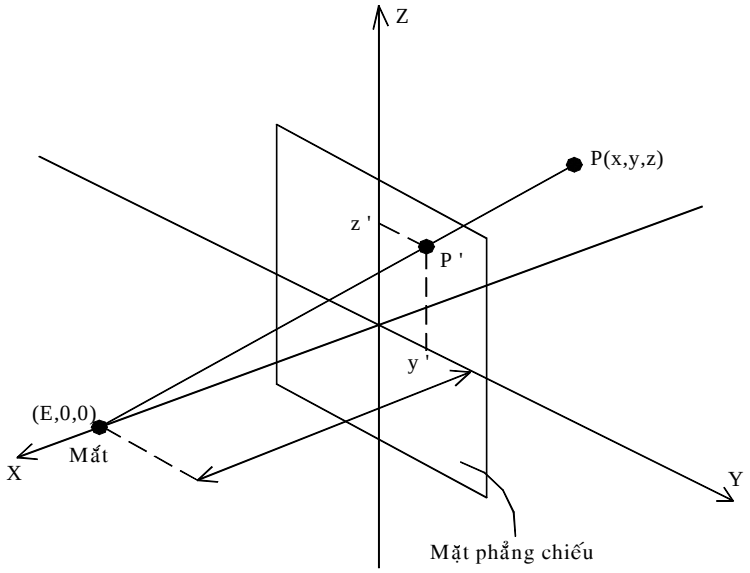
4.3. CÁC PHÉP CHIẾU CỦA VẬT THỂ TRONG KHÔNG GIAN LÊN MẶT PHẲNG

4.3.1. Phép chiếu phối cảnh (Perspective)

Phép chiếu này cho hình ảnh giống như khi nhìn vật thể.

Để tìm hình chiếu P'(y',z') của P(x,y,z), ta nối P với mắt (tâm chiếu). Giao điểm của đường này với mặt quan sát chính là P' (hình 4.3).

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



Hình 4.3

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

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

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

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

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.

iii/ Phép chiếu phối cảnh được qui định bởi 5 biến:

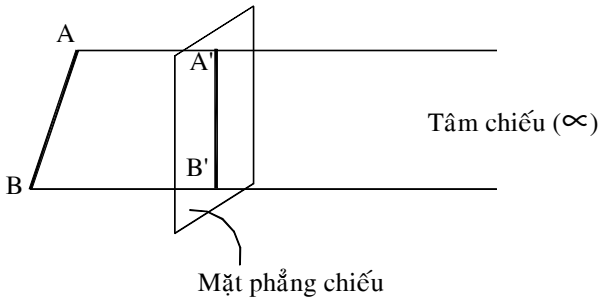
- Hướng của mặt phẳng chiếu so với vật thể.
- Độ cao của tâm chiếu so với vật thể.
- Khoảng cách từ tâm chiếu đến vật thể (R).
- Khoảng cách từ mặt phẳng chiếu đến tâm chiếu (D).
- Độ dịch chuyển ngang của tâm chiếu so với vật thể.

Chú ý: Với tọa độ cầu, ta chỉ cần 4 tham số: R, **Φ**, **θ**, D.

4.3.2. Phép chiếu song song (Parallel)

Phép chiếu này có tâm chiếu ở vô cực và y'=y, z'=z.(Hình 4.4)

Tính song song được bảo toàn.



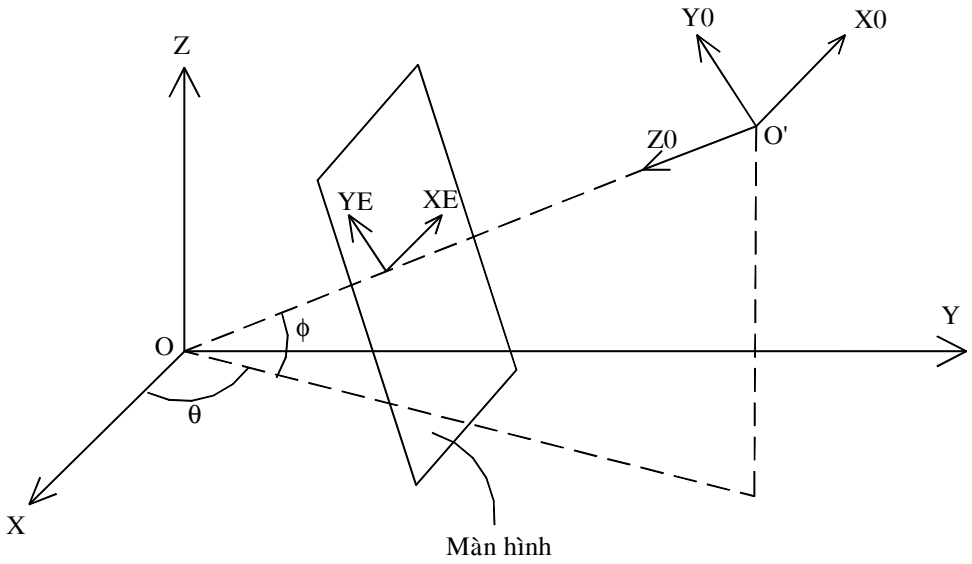
Hình 4.4

4.4. CÔNG THỨC CỦA CÁC PHÉP CHIẾU LÊN MÀN HÌNH

Khi quan sát một vật thể trong không gian dưới một góc độ nào đó, ta có 2 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.



Hình 4.5

Khi quan sát một vật thể bất kỳ trong không gian, ta phải tuân thủ các nguyên tắc sau (hình 4.5):

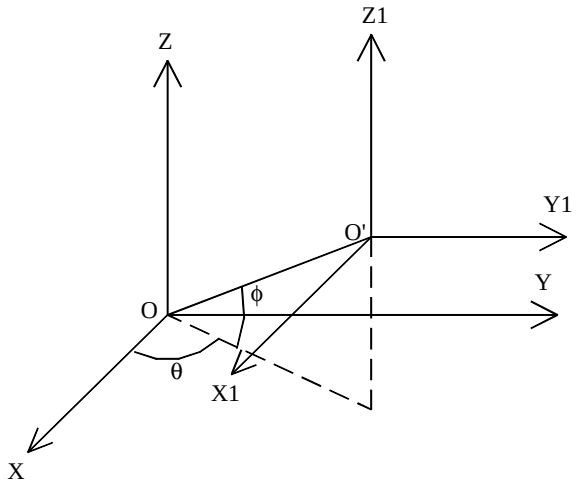
- Vật thể phải được chiếu lên một *hệ trực tiếp* (O,X,Y,Z).

- 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.

Nếu dùng hệ tọa độ cầu để định vị mắt của người quan sát thì ta 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) để cuối cùng tạo ra hệ tọa độ màn hình (Xe,Ye).

Bước **1**: Tịnh tiến gốc O thành O' (hình 4.6).



Hình 4.6

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 (X1,Y1,Z1).

Bước **2**: Quay hệ (X1,Y1,Z1) một góc **-θ'** (**$\theta'=90^0 - \theta$**) quanh trục Z1 theo chiều kim đồng hồ. Phép quay này làm cho trục âm của Y1 cắt trục Z (hình 4.7).

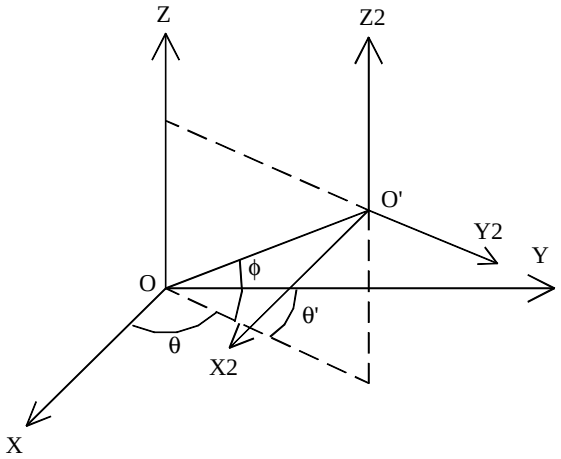
Ta gọi Rz là ma trận tổng quát của phép quay quanh trục Z. Vì đây là phép quay hệ trục nên phải dùng ma trận nghịch đảo R^{-1}_z.

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

ta thay góc a = -**θ' .** Theo các phép toán lượng giác:

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

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



Hình 4.7

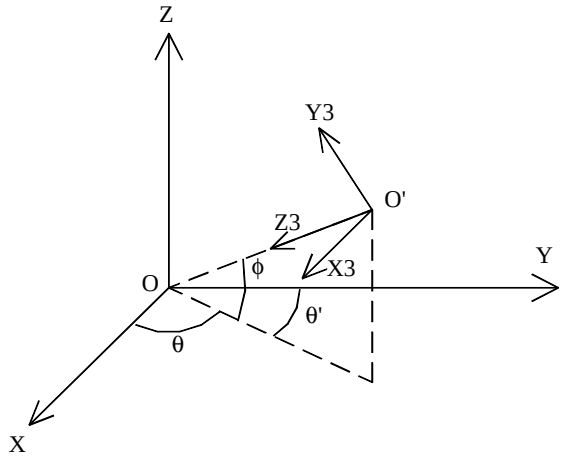
Nên ma trận của phép quay tìm được sẽ có dạng:

$$\mathbf{B} = \begin{pmatrix} Sin(\theta) & Cos(\theta) & 0 & 0 \\ -Cos(\theta) & Sin(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$ và hệ (X1,Y1,Z1) biến đổi thành hệ (X2,Y2,Z2).

Bước 3: Quay hệ (X2,Y2,Z2) một góc $90^0 + \boldsymbol{\Phi}$ quanh trục X2. Phép biến đổi này sẽ làm cho trục Z2 hướng đến gốc O (hình 4.8).

Ta có:

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



Hình 4.8

$$R^{-1}_{x} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos(a) & -Sin(a) & 0 \\ 0 & Sin(a) & Cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Thay góc a = 90^0 + **Φ** , ta có: Cos(90^0 + **Φ**) = -Sin(**Φ**) và Sin(90^0 + **Φ**) = Cos(**Φ**) nên ma trận tìm được sẽ có dạng:

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

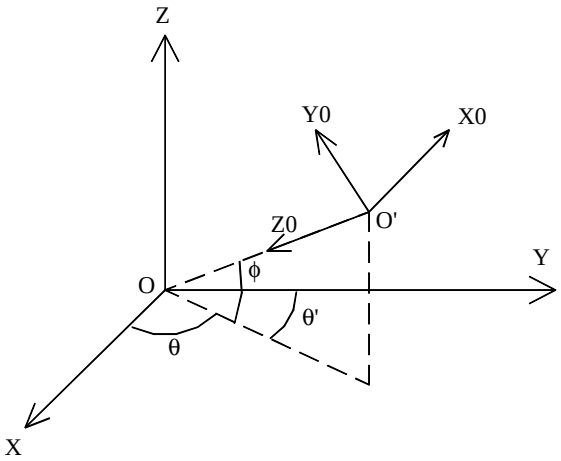
Lúc này, hệ (X2,Y2,Z2) biến đổi thành hệ (X3,Y2,Z3).

Bước 4: Biến đổi hệ trực tiếp (X3,Y3,Z3) thành hệ gián tiếp (hình 4.9).

Trong bước này, ta phải đổi hướng trục X3 bằng cách đổi dấu các phần tử của cột X. Ta nhận được ma trận:

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

và hệ (X3,Y3,Z3) biến đổi thành hệ (X_0,Y_0,Z_0).



Hình 4.9

TÓM LẠI

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

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

Gọi T = A.B.C.D, ta tính được:

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

Cuối cùng ta có:

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

hay:

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

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

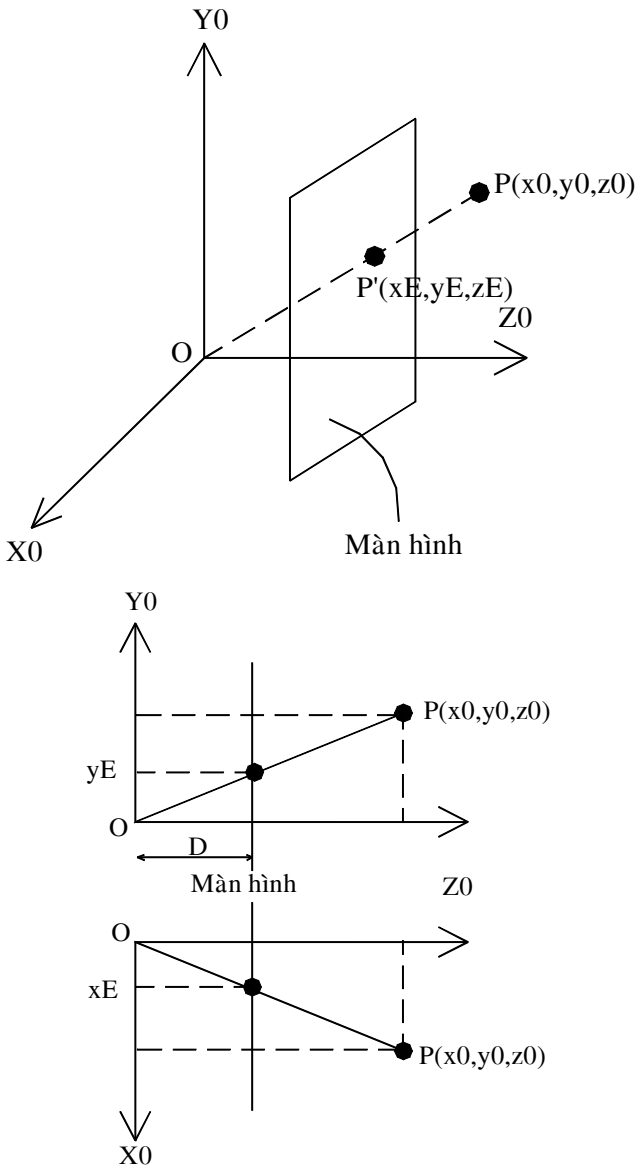
$$z_0 = -x.Cos(\theta).Cos(\Phi) - y.Sin(\theta).Cos(\Phi) - z.Sin(\Phi) + R$$

**** Bây giờ 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.

Gọi D là khoảng cánh từ mặt phẳng đến mắt (gốc tọa độ). (Hình 4.10)



Hình 4.10

Xét các tam giác đồng dạng, ta có:

$xE/D = x_0/z_0$ và $yE/D = y_0/z_0$

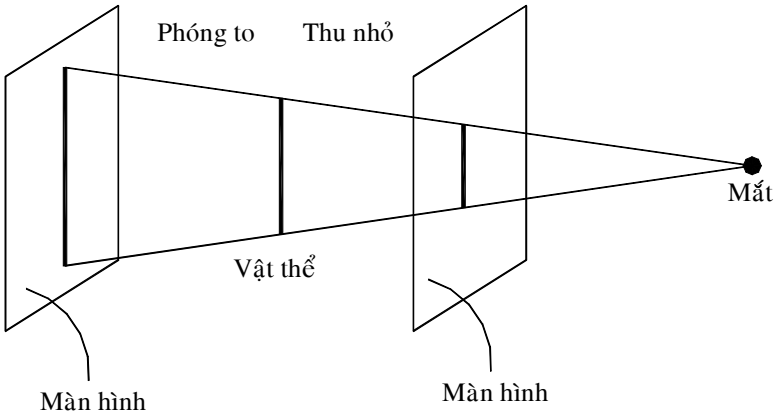
$\Rightarrow$ **$x_E = D.x_0/z_0$ và $y_E = D.y_0/z_0$**

Chú ý: z_0 bao hàm việc phóng to hay thu nhỏ vật thể.

2. Phép chiếu song song

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

$x_E = x_0$ và **$y_E = y_0$**



Hình 4.11

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.

4.5. PHỤ LỤC

Tạo UNIT DOHOA3D (DOHOA3D.PAS).

```
UNIT   DOHOA3D;
INTERFACE
USES  graph,crt;
{ Cac hang de quay hinh }
Const  IncAng   = 5;      {Tang goc}
Type ToaDo3D=Record
                x,y,z:real;
            End;
    ToaDo2D=Record
                x,y:integer;
            End;
```

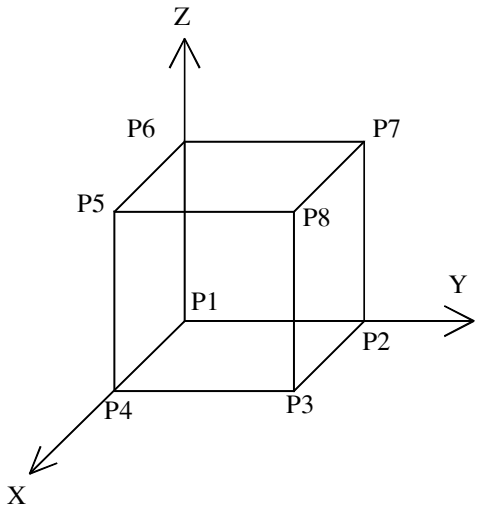
```
     PhepChieu = (PhoiCanh,SongSong);
 VAR R,d,theta,phi  : real;
     aux1,aux2,aux3,aux4 : real;
     aux5,aux6,aux7,aux8 : real;
     projection         : PhepChieu;
     xproj,yproj        : real;
     Obs,O              : ToaDo3D;
     PE,PC              : ToaDo2D;
     { cac bien dung quay hinh }
      ch : char;
PROCEDURE ThietLapDoHoa;
PROCEDURE KhoiTaoPhepChieu;
PROCEDURE Chieu(P :ToaDo3D);
PROCEDURE VeDen(P :ToaDo3D);
PROCEDURE DiDen(P :ToaDo3D);
PROCEDURE TrucToaDo;
PROCEDURE DieuKhienQuay;  {dung de quay hinh}
IMPLEMENTATION
Procedure ThietLapDoHoa;
 var gd,gm:integer;
 Begin
   Gd:=0;
   InitGraph(gd,gm,'C:\BP\BGI');
  End;
PROCEDURE KhoiTaoPhepChieu;
 VAR   th,ph :real;
 BEGIN
   th := pi*theta/180;
   ph := pi*phi/180;
   aux1 := sin(th);
   aux2 := sin(ph);
   aux3 := cos(th);
```

```
   aux4 := cos(ph);
   aux5 := aux3*aux2;
   aux6 := aux1*aux2;
   aux7 := aux3*aux4;
   aux8 := aux1*aux4;
    PC.x := getmaxx div 2;
   PC.y := getmaxy div 2;
 END;
PROCEDURE Chieu(P :ToaDo3D);
 BEGIN
   Obs.x := -P.x*aux1 + P.y*aux3 ;
   Obs.y := -P.x*aux5 - P.y*aux6 + P.z*aux4  ;
   IF projection = PhoiCanh  THEN
     BEGIN
       obs.z:=-P.x*aux7 -P.y*aux8 -P.z*aux2 + R;
       Xproj := d*obs.x/obs.z;
       Yproj := d*obs.y/obs.z;
     END
   ELSE BEGIN
          Xproj := d*obs.x;
          Yproj := d*obs.y;
        END;
 END;
PROCEDURE VeDen(P :ToaDo3D);
 BEGIN
   Chieu(P);
   PE.x := PC.x + round(xproj);
   PE.y := PC.y - round(yproj);
   lineto (PE.x,PE.y);
 END;
PROCEDURE Diden(P :ToaDo3D);
 BEGIN
```

```
   Chieu(P);
   PE.x := PC.x + round(xproj);
   PE.y := PC.y - round(yproj);
   moveto (PE.x,PE.y);
 END;
PROCEDURE TrucToaDo;        { Ve 3 truc }
 var OO,XX,YY,ZZ:ToaDo3D;
 Begin
   OO.x:=0;     OO.y:=0;     OO.z:=0;
   XX.x:=3;     XX.y:=0;     XX.z:=0;
   YY.x:=0;     YY.y:=3;     YY.z:=0;
   ZZ.x:=0;     ZZ.y:=0;     ZZ.z:=3;
    DiDen(OO);   VeDen(XX);
    DiDen(OO);   VeDen(YY);
    DiDen(OO);   VeDen(ZZ);
 END;
PROCEDURE DieuKhienQuay;  {Dieu khien Quay/Zoom hinh}
 BEGIN
   ch := readkey;
   IF ch = #0 THEN ch := readkey;
   cleardevice;
   CASE UpCase(ch) OF
   #72 : phi := phi + incang;
   #80 : phi := phi - incang;
   #75 : theta := theta + incang;
   #77 : theta := theta - incang;
    END; {of case ch}
 END; {of Procedure}
END. {Of UNIT}
```

4.6. VÍ DỤ MINH HỌA

Viết chương trình mô tả phép quay của một hình lập phương quanh các trục (hình 4.12).



Hình 4.12

```
Uses crt,graph,Dohoa3D;
var P1,P2,P3,P4,P5,P6,P7,P8:ToaDo3D;
Procedure KhoiTaoBien;
 Begin
  D:=70;      R:=5;
  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;
 End;
Procedure VeLapPhuong;
begin
  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);
end;
Procedure MinhHoa;
 BEGIN
  KhoiTaoBien;
  KhoiTaoPhepChieu;
  TrucToaDo;
  VeLapPhuong;
  Repeat
   DieuKhienQuay;
   KhoiTaoPhepChieu;
   ClearDevice;
   TrucToado;
   VeLapPhuong;
  until ch=#27;
 END;
BEGIN   { Chuong Trinh Chinh }
  Projection:=SongSong{Phoicanh};
  ThietLapDoHoa;
  MinhHoa;
  CloseGraph;
END.
```

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 hệ trục tọa độ.

4. Dựa vào bài tập 2, hãy mô phỏng quá trình quay của một Ellipse xung quanh tâm của nó.

5. Viết chương trình mô phỏng quá trình quay, đối xứng, tịnh tiến, phóng to, thu nhỏ, biến dạng của một hình bất kỳ trong mặt phẳng.

6. Mô phỏng chuyển động của trái đất xung quanh mặt trời đồng thời mô tả chuyển động của mặt trăng xung quanh trái đất.

Mở rộng trong không gian 3 chiều.

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

8. Viết chương trình vẽ các khối đa diện đều trong không gian.

Mở rộng: điều khiển phóng to, thu nhỏ, quay các khối đa diện quanh các trục...

CHƯƠNG V

BIỂU DIỄN CÁC ĐỐI TƯỢNG BA CHIỀU

5.1. MÔ HÌNH WIREFRAME

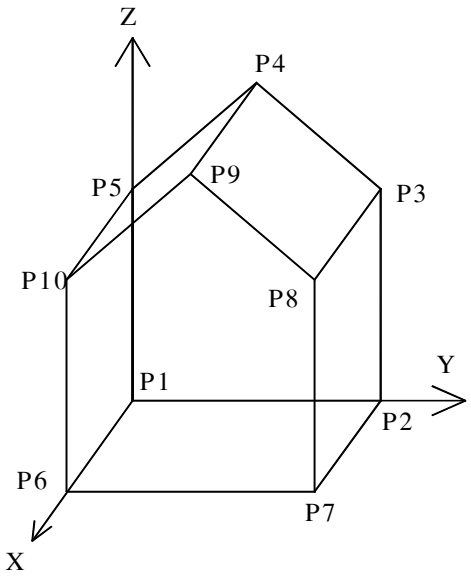
Mô hình WireFrame thể hiện hình dáng của đối tượng 3D bằng 2 danh sách :

- Danh sách các đỉnh : lưu tọa độ của các đỉnh.
- Danh sách các cạnh : lưu các cặp điểm đầu và cuối của từng cạnh.

Các dỉnh và các cạnh được đánh số thứ tự cho thích hợp.

Ví dụ: Biểu diễn 1 căn nhà thô sơ (hình 5.1)

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



Hình 5.1

Có nhiều cách để lưu giữ mô hình WireFrame. Ở đây, chúng ta dùng cấu trúc record dựa trên 2 mảng:

```
Const  MaxDinh = 50; { Số đỉnh tối đa}
       MaxCanh = 100; {Số cạnh tối đa}
Type   ToaDo3D = record
                   x, y, z:real;
               end;
       WireFrame = Record
```

```
        Sodinh: 0..MaxDinh;
        Dinh: array [1..MaxDinh] of ToaDo3D;
        Socanh : 0..Maxcanh;
        Canh :array[1..Maxcanh, 1..2] of 1..MaxDinh;
    end;
```

Khi đó, ta dùng một biến để mô tả căn nhà :

```
Var   House : WireFrame;
```

với biến house ở trên, ta có thể gán giá trị như sau:

```
With House Do
Begin
   sodinh:=10;
   socanh:=17;
   dinh[1].x:=0;
   dinh[1].y:=0;
   dinh[1].z:=0;
...
   canh[1, 1]:=1;  {Số đỉnh thứ nhất của
                    cạnh số 1}
   canh[1, 2]:=2;  {Số đỉnh thứ hai của
                    cạnh số 1}
   ...
end;
```

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
16	2	5
17	1	3

5.2. VẼ MÔ HÌNH WIREFRAME VỚI CÁC PHÉP CHIẾU

Để vẽ một đối tượng WireFrame, ta vẽ từng cạnh trong danh sách các cạnh của mô hình. Vấn đề là làm thế nào để vẽ 1 đường thẳng trong không gian 3 chiều vào mặt phẳng?

Để làm điều này, ta phải bỏ bớt đi 1 chiều trong mô hình biểu diễn, tức là ta phải dùng phép chiếu từ 3D $\rightarrow$ 2D .

Kỹ thuật chung để vẽ một đường thẳng 3D là:

- Chiếu 2 điểm đầu mút thành các điểm 2D.
- Vẽ đường thẳng đi qua 2 điểm vừa được chiếu.

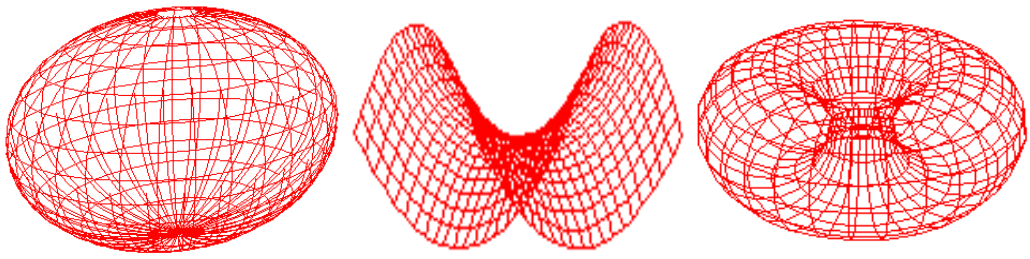
Sau đây là thủ tục xác định hình chiếu của một điểm qua phép chiếu phối cảnh:

```
Procedure Chieu(P3D:ToaDo3D; E:Real; Var P2D:ToaDo2D);
Var t:Real;
Begin
If (P3D.x >=E) OR (E=0) Then
   Writeln('Điểm nằm sau mắt hoặc mắt nằm trên mặt phẳng
            nhìn');
Esle
   Begin
      t := 1/(1 - P3D.x/E);
      P2D.y := t*P3D.y;
      P2D.z := t*P3D.z;
   End;
End;
```

5.3. VẼ CÁC MẶT TOÁN HỌC

Ta sẽ vẽ các mặt cong dựa trên phương trình tham số của các mặt đó.

Ví dụ:



(a) (b) (c)

Hình 5.2

- Mặt Ellipsoid: (hình 5.2.a)

  x=Rx.cos(u).cos(v)

  y=Ry.sin(u).cos(v)

z=Rz.sin(v)

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

- Mặt Hypeboloid: (hình 5.2.b)

  x=u

  y=v

  $z=u^2 - v^2$

  Trong đó $u,v \in [-1,1]$

- Hình xuyến: (hình 5.2.c)

  x=(R + a.cos(v)).cos(u)

  y=(R + a.cos(v)).sin(u)

  z= a.sin(v)

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

- Hình trụ tròn (Cylinder)

  x = R.cos(u)

  y = R.sin(u)

  z = 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 $\begin{cases} x = R.\cos(u) \\ y = R.\sin(u) \end{cases}$ $u,v \in [0,1]$

- Chảo Parabol (Paraboloid)

  x = a.v.cos(u)

  y = b.v.sin(u) $u \in [-\pi,\pi]$, $v \ge 0$

  $z = v^2$

Phương pháp chính ở đây cũng là vẽ các đường viền theo u và v.

Để vẽ một đường viền u tại giá trị u’ khi v chạy từ VMin đến VMax ta làm 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.
- Vẽ các đường gấp khúc dựa trên các điểm 2D P’[i].

Sau đây là thủ tục vẽ họ đường cong theo u:

```
ProcedureHoDuongCongU;
  Var  P: ToaDo3D;
       u,v,du,dv:Real;
Begin
  u:=UMin; du:=0.05;  dv:=0.05;
  While u<=UMax do
    Begin
      v:=Vmin;
      P.x:=fx(u,v);
      P.y:=fy(u,v);
      P.z:=fz(u,v);
      DiDen(P);{ Đi đến điểm xuất phát ban đầu }
      While v<=VMax do
        Begin
          v:=v+dv;
          P.x:=fx(u,v);
          P.y:=fy(u,v);
          P.z:=fz(u,v);
          VeDen(P);{ Vẽ đến điểm mới }
        End;
      u:=u + du;
    End;
End;
```

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

TÓM LẠI: Muốn vẽ một mặt cong, ta thực hiện các bước sau

- Nhập các hệ số của phương trình mặt: a, b, c, d, Umin, Umax, Vmin, Vmax.
- Tính các hàm 2 biến: X(u,v), Y(u,v), Z(u,v).
- Khởi tạo phép chiếu: Song song/Phối cảnh.
- Vẽ họ đường cong u.

  Vẽ họ đường cong v.

BÀI TẬP

1. Hãy xây dựng một cấu trúc dữ liệu để lưu trữ mô hình WireFrame.

2. Tạo file text để lưu các đỉnh và cạnh của một vật thể trong không gian 3D theo mô hình WireFrame với cấu trúc như sau:

- Dòng đầu tiên chứa hai số nguyên m, n dùng để lưu số đỉnh và số cạnh của mô hình.
- m dòng tiếp theo, mỗi dòng lưu tọa độ x, y, z của từng đỉnh trong mô hình.
- n dòng tiếp theo, mỗi dòng lưu hai số nguyên là đỉnh đầu và đỉnh cuối của từng cạnh trong mô hình.

3. Viết thủ tục để đọc các giá trị trong file text lưu vào mô hình WireFrame.

4. Viết thủ tục để vẽ vật thể từ mô hình WireFrame.

5. Viết chương trình biểu diễn các khối đa diện sau: Tứ diện đều, Khối lập phương, Bát diện đều, Thập nhị diện đều, Nhị thập diện đều.

6. Viết chương trình để mô phỏng các mặt toán học: yên ngựa, mặt cầu, hình xuyến...

CHƯƠNG VI

THIẾT KẾ ĐƯỜNG VÀ MẶT 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)).

6.1. ĐƯỜNG CONG BEZIER VÀ MẶT BEZIER

6.1.1. 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).

Ta biểu diễn bằng phương trình:

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

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

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

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:

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

$$r = 0,1,...,L \quad \text{và} \quad i = 0,...,L-r$$

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.

6.1.2. 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:

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

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

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

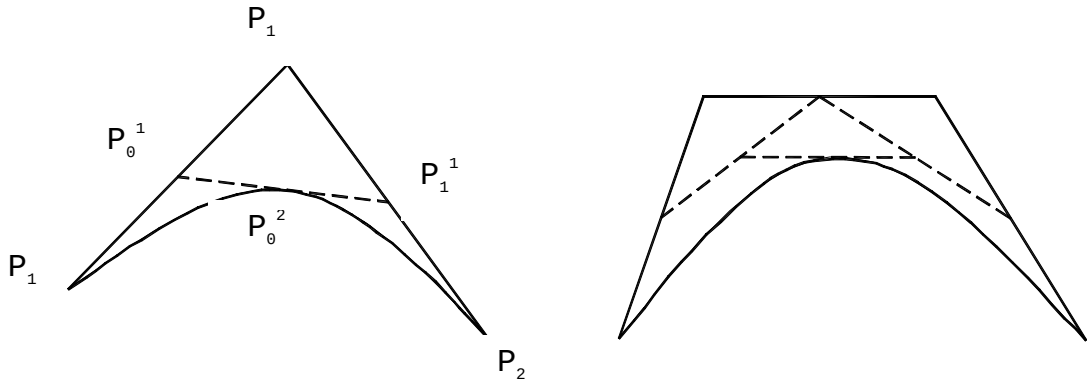
$$B_k^L(t) = \frac{L!}{k!(L-k)!}(1-t)^{L-k}.t^k \qquad \text{với } L \geq k$$

Mỗi đa thức Bernstein có bậc là L. Thông thường ta còn 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.



Đường cong Bezier bậc 2 *Đường cong Bezier bậc 3*

Hình 6.1

6.1.3. Dạng biểu diễn ma trận của đường Bezier

Để thích hợp cho việc xử lý trên máy tính, ta biểu diễn hai mảng BL(t) và P như sau:

$$B^L(t) = (B_0^L(t), B_1^L(t), ..., B_L^L(t))$$

$$P = (P_0, P_1, ..., P_L)$$

Do đó: $P(t) = B^L(t).P$ (tích vô hướng)

hay $P(t) = B^L(t).P^T$ (P^T là dạng chuyển vị của P)

Dưới dạng đa thức, có thể biểu diễn $B_k^L(t)$ như sau:

$$B_k^L(t) = a_0 + a_1.t + a_2.t^2 + ... + a_L.t^L = (t^0, t^1, ..., t^L).(a_0, a_1, ..., a_L)$$

Do đó P(t) có thể biểu diễn lại như sau:

$$\mathbf{P(t) = Pow^L(t).Bez^L.P^T}$$

Với:

- $Pow^L(t) = (t^0, t^1, ..., t^L)$

• Bez^L là ma trận biểu diễn mảng $B^L(t)$ trong đó mỗi hàng i của ma trận ứng với các hệ số tương ứng $(a_0, a_1, ..., a_L)$ của đa thức $B_i^L(t)$ và tại vị trí (i,j) trong ma trận Bez^L có giá trị $Bez^L(i,j) = (-1)^{j-i}.C_n^i.C_i^j$

Ví dụ: Ma trận Bez^3 cho các đường Bezier bậc 3

$$Bez^3 = \begin{pmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{pmatrix}$$

6.1.4. 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, ví dụ có thể lấy ti = 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. Để tính $P(t_i)$ ta có thể áp dụng ma trận của P(t) ở trên trong đó chỉ có thành phần $Pow^L(t_i)$ là thay đổi, còn tích $Bez^L.P^T$ với $P = (P_0, P_1, ..., P_L)$ là không thay đổi.

Sau đây là thủ tục minh họa việc vẽ đường cong Bezier trong mặt phẳng:

```
Type Mang = array[0..50] of PointType;
function tich(x,y:word):real;
 var s:real;i:word;
 begin
  if y<=1 then tich:=1
  else begin
        s:=1;
        for i:=x to y do s:=s*i;
        tich:=s;
       end;
 end;
function CLK(l,k:word):real;
 begin
   CLk:=tich(k+1,l)/tich(1,l-k);
 end;
function Xmu(x:real;mu:word):real;
```

```
 var i:word;s:real;
 begin
   if mu=0 then s:=1
   else  begin
           s:=1;
           for i:=1 to mu do s:=s*x;
         end;
   Xmu:=s;
 end;
function BKL(t:real;l,k:word):real;
 begin
   BKL:=CLK(l,k)*xmu(1-t,l-k)*xmu(t,k);
 end;
procedure Pt(t:real;L:word;A:Mang;var diem:PointType);
 var k:word;s,x,y:real;
 begin
   x:=0; y:=0;
   for k:=0 to L do
     begin
       s:=BKL(t,l,k);
       x:=x+A[k].x*s;
       y:=y+A[k].y*s;
     end;
   diem.x:=round(x);
   diem.y:=round(y);
 end;
procedure Vebezier(A:Mang;L:integer);
 var i,SoDiem:word; Diem:PointType;
     dx,x:real;
begin
sodiem:=100;
 dx:=1/sodiem;
```

```
 x:=0;
 if L>0 then
  begin
    for i:=1 to sodiem+1 do
     begin
       Pt(x,L,A,Diem);
       if i=1 then moveto(round(diem.x),round(diem.y))
       else        lineto(round(diem.x),round(diem.y));
       x:=x+dx;
     end;
  end
end;
```

6.1.5. Các tính chất của đường cong Bezier

i/ Nội suy được các điểm đầu và cuối.

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

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

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

Nhưng theo đa thức Bernstein thì $\sum_{k=0}^{L} B_k^L(t) = (1\text{-}t+t)^L = 1$ 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 Pk:

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

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/ Bất kỳ một đường thẳng hay mặt phẳng nào cũng luôn luôn cắt đường cong Bezier ít lần hơn so với cắt đa giác kiểm soát.

vi/ 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))'.

6.1.6. 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:

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

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:

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

6.2. ĐƯỜNG CONG SPLINE VÀ B-SPLINE

6.2.1. Định nghĩa

Theo trên ta có: $P(t) = \sum_{k=0}^{L} P_k.R_k(t)$ (*)

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 vector 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 $N_{k,L}(t) = \begin{cases} 1 & t_k \pi t \le t_{k+1} \\ 0 & \text{ngæåüc laûi} \end{cases}$

(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.*

6.2.2. 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 này 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 Affin**. 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.

6.2.3. 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:

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

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

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.

6.2.4. Các băng Bezier

Đường cong Bezier trong không gian 3 chiều có thể được viết dưới dạng là một hàm của tham số v với L+1 điểm kiểm soát tùy thuộc vào tham số u theo một kiểu nào đó: Chẳng hạn $P(u,v) = \sum_{k=0}^{L} P_k(u).B_k^L(v)$ (*)

Nghĩa là mỗi đường viền u là một đường cong Bezier chuẩn, nhưng ở những giá trị u khác nhau thì các điểm kiểm soát cũng nằm ở những vị trí khác nhau.

Khi u biến thiên thì mỗi điểm kiểm soát $P_k(u)$ sẽ chạy trên một đường cong cụ thể. Do đó, mặt cong có thể xem như là một sự dịch chuyển đường Bezier trong không gian.

Ta tưởng tượng một đa giác kiểm soát chuyển động trong không gian và thay đổi dạng khi chuyển động. Ở mỗi vị trí, đa giác này tạo nên một đường cong Bezier và mặt cong tạo thành chính là cái ***vết*** còn để lại bên dưới của đường cong này.

Ví dụ: Phép chiếu phối cách của một mặt được tạo ra bởi việc nội suy tuyến tính giữa 2 đường cong Bezier dựa trên 2 đa giác kiểm soát là P_0 và P_1. Mỗi đường cong kiểm soát $p_k(u)$ được nội suy tuyến tính giữa 2 điểm kiểm soát P_k^0 và P_k^1 khi u biến thiên giữa 0 và 1:

$$p_k(u) = (1-u).P_k^0 + u.P_k^1 \qquad k=0,1,2,3$$

Giả sử các đường cong kiểm soát $p_k(u)$ chính là các đường cong Bezier, mỗi đường cong này dựa trên m +1 điểm kiểm soát của chúng.

Vì vậy: $P_k(u) = \sum_{i=0}^{M} P_{i,k}.B_i^M(u)$

Kết hợp $p_k(u)$ này vào phương trình (*) ta được:

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

Ta gọi đây là dạng **tích Tensor** cho băng Bezier.

Cũng giống như các đa giác kiểm soát trong 2D, một khối đa diện kiểm soát là một mạng gồm có (M+1).(L+1) đỉnh.

Tóm lại, để tạo ra một băng ta chỉ cần chỉ ra các vị trí của các đỉnh này rồi sau đó áp dụng phương trình (**) để vẽ các đường viền hay định nghĩa dạng mặt cong.

6.2.5. Dán các băng Bezier với nhau

Mục đích là để tạo ra các dạng mặt phức tạp gồm nhiều băng Bezier kết lại với nhau một cách trơn tru ở các biên chung.

Khi nối 2 băng Bezier lại với nhau, mỗi băng có một khối đa diện kiểm soát riêng và đều được tạo ra từ phương trình (*) với u, v biến thiên trong khoảng [0,1]. Vấn đề là làm sao cho 2 băng có thể dán vào nhau một cách trơn tru.

- Hai băng sẽ gặp nhau ở tất cả các điểm dọc theo biên chung nếu các khối đa diện kiểm soát của chúng khớp nhau ở biên. Như vậy, ta chỉ cần chọn các đa giác kiểm soát biên để cho 2 băng đồng nhất nhau ở biên. Có thể thấy được điều này khi thay u=0 vào trong phương trình (*) ở trên.
- Một điều kiện đủ nữa là mỗi cặp cạnh của khối đa diện mà nó gặp nhau ở biên phải tuyến tính cùng nhau.

6.2.6. Các băng B-Spline

Các hàm B-Spline có thể được sử dụng trong dạng tích Tensor thay cho các đa thức Bernstein để đạt được tính kiểm soát cao hơn khi thiết kế mặt cong. Điều đó có nghĩa ta sẽ thay phương trình (**) thành:

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

Khối đa diện kiểm soát gồm có (L+1).(M+1) điểm kiểm soát; u,v biến thiên từ 0 tới giá trị nút lớn nhất trong các vector nút tương ứng của chúng.

Đối với các băng B-Spline, người ta vẫn dùng các B-Spline bậc 4. Do việc chọn số điểm kiểm soát là không giới hạn nên có thể tạo ra nhiều dạng mặt cong rất phức tạp.

Tất nhiên khi thiết kế, ta phải chọn khối đa diện nút để tạo ra mặt có dạng mong muốn.

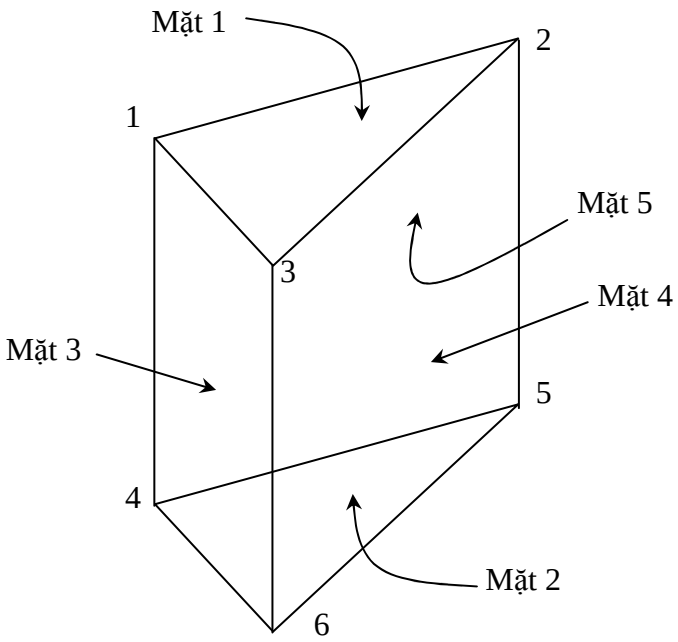
CHƯƠNG VII

KHỬ ĐƯỜNG VÀ MẶT KHUẤT

7.1. CÁC KHÁI NIỆM

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

- Mô hình WireFrame: Đã trình bày ở chương 5, nó cho ta hình dáng của vật thể dưới dạng một bộ khung
- Mô hình các mặt đa giác: ở đây 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 thì chúng 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 chiếu sáng.Trong chương này chúng ta sẽ tập trung nghiên cứu vấn đề khử mặt khuất.



Hình 7.1

Ví dụ: Mô tả vật thể như trong 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	1,2,3
2	4,5,6
3	1,3,6,4
4	3,2,5,6
5	1,2,5,4

Chúng ta có thể đưa ra nhiều cấu trúc dữ liệu khác nhau để lưu trữ cho đa giác. Dưới đây là phát thảo một kiểu cấu trúc:

```
Type Point3D = Record   {Điểm 3 chiều}
                 x,y,z:real;
               end;
     Vector3D = Record       {Vector 3 chiều. Mặc dù nó giống với
                 x,y,z:real; Point3D song ta vẫn khai để các thuật toán
               end;          được tường minh}
     RGBColor = Record  {Cấu trúc màu sắc của một mặt}
                 B,G,R:Byte;
               end;
     KieuMat = Record
              PhapVT:Vector3D;   {Pháp vector của mặt}
              Sodinh:cardinal;   {Số đỉnh của mặt}
              List:array of integer;{Danh sách thứ tự các đỉnh tạo
                                     nên mặt. Ở đây ta dùng mảng động}
              Color:RGBColor;    {màu sắc của mặt}
            end;
     Obj3D = record    {Đối tượng 3 chiều}
              ObjName:string;    {Tên của đối tượng}
              Sodinh:cardinal;   {Số đỉnh}
              Dinh: array of point3d; {Danh sách đỉnh. Ở đây ta dùng
                                       kiểu mảng động}
              SoMat:cardinal;    {Số mặt}
              Mat:array of KieuMat; {Danh sách mặt}
```

```
        Xworld,Yworld,Zworld,Zoom:Real;   {Toạ độ và kích
                                          thước thật của vật trong hệ toạ độ thế giới}
    end;
```

Khi cài đặt cho một ứng dụng cụ thì việc sử dụng mảng cố định có thể gây ra các trở ngại về kích thước tối đa hay tối thiểu, cũng như việc sử dụng bộ nhớ không tối ưu. Vì thế ngoài cách dùng mảng cố định, ta có thể dùng mảng động trong một số ngôn ngữ như Visual Basic, Delphi hay Visual C++,… hoặc dùng cấu trúc danh sách móc nối. Song song với điều đó là việc bớt đi hay đưa thêm các thuộc tính cần thiết để biểu diễn các đặc tính khác của mặt hay của đối tượng.

*** 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 người thợ sơn và sắp xếp theo chiều sâu (Depth-Sorting)

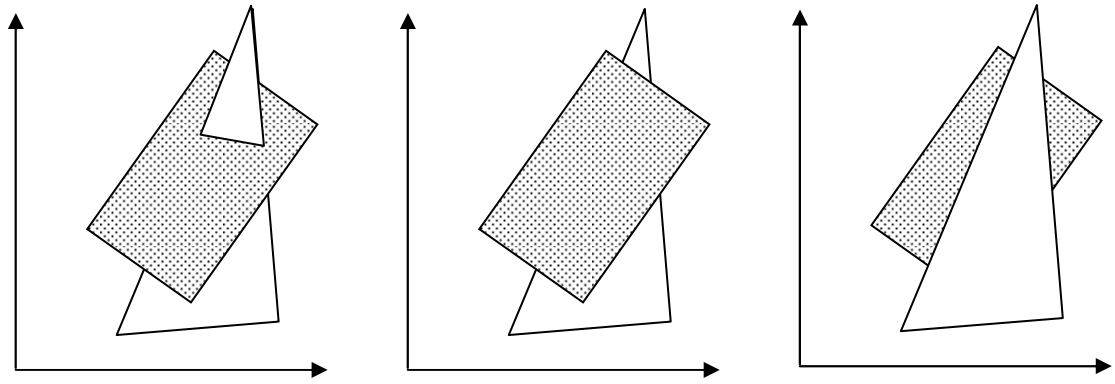
Người thợ sơn (hay Depth-sorting) là tên của một thuật giải đơn giản nhất trong số các thuật toán vẽ ảnh thực 3 chiều. Nếu để ý người thợ sơn làm việc, chúng ta sẽ thấy anh ta sơn bức tranh từ trong ra ngoài, với các cảnh vật từ xa đến gần. Chúng ta có thể áp dụng một cách tương tự để vẽ các đa giác 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.

Như vậy, chúng ta cần phải sắp xếp các mặt theo thứ tự từ xa đến gần, rồi 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, thuật giải 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):

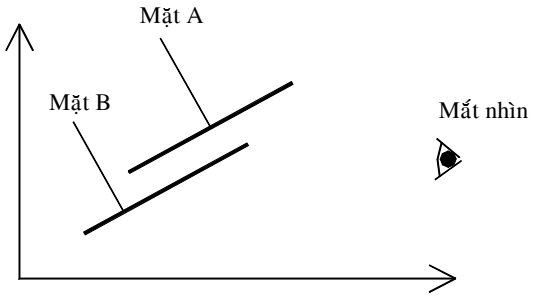
- Khi hai mặt cắt nhau thì thuật giải này chỉ thể hiện như chúng chồng lên nhau.



Hình ảnh thật Khi vẽ bằng giải thuật trên

Hình 7.2

- 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

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

- **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**

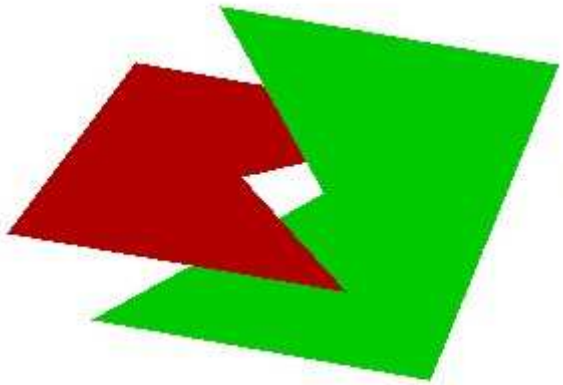
Giả sử A và B là hai đa giác mà sau khi sắp xếp theo độ sâu trung bình thì A đứng trước B. Song qua 3 phép kiểm tra trên mà vẫn không xác định được liệu trật tự trên là đúng hay chưa. Lúc này chúng phải tiến hành phép kiểm tra cạnh xa. Phép kiểm tra cạnh xa nhằm xác định xem đa giác B có nằm phía sau 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 (7.2) hoặc hình (7.3), 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 7.2) hoặc chéo vào nhau (hình 7.4), 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.



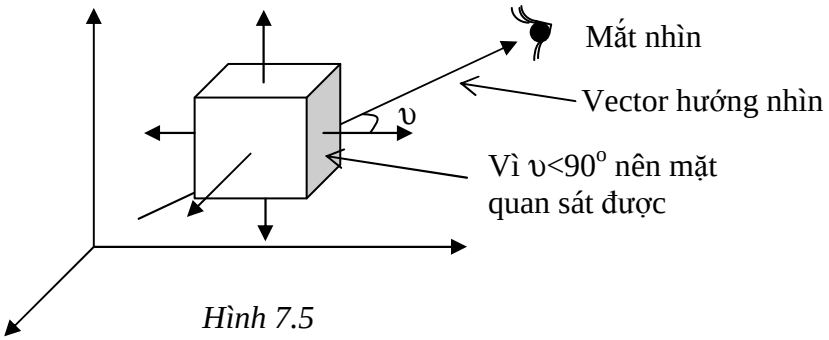
Hình 7.4

7.2.2. Giải thuật BackFace

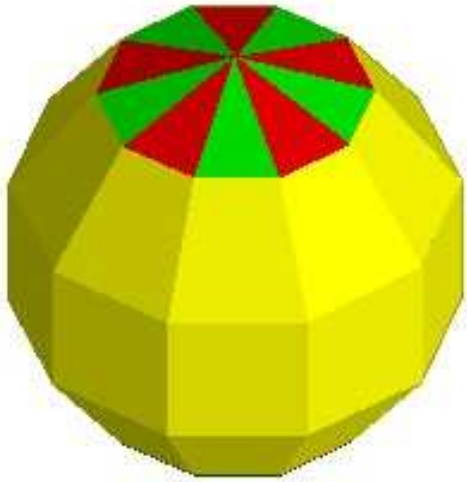
Sẽ rất đơn giản nếu ta dùng Vector pháp tuyến để khử các mặt khuất của một đối tượng 3D đặc và lồi. Ta sẽ tính góc giữa véc tơ hướng nhìn V và pháp vector N của mặt, nếu góc này là lớn hơn 90^o thì mặt là không nhìn thấy (bị khuất), ngược lại thì mặt là khả kiến.

Dấu của tích vô hướng của 2 vector là dương nếu góc giữa chúng nhỏ hơn hay bằng 90^o. Vậy thuật toán để xét một mặt bị khuất hay không chỉ đơn giản là:

If V.N >= 0 then Mặt thấy
Else Mặt không thấy (mặt khuất);



Hình 7.5



Hình 7.6

Cài đặt minh hoạ cho thuật toán chọn lọc mặt sau

```
Function Tich_vo_huong(v,n:Vector3D):real;
```

{*Tính tích vô hướngcủa 2 vector*}

```
Begin
  Tich_vo_huong:=v.x*n.x+v.y*n.y+v.z*n.z;
End;
Procedure DrawObj_FilterRearFace(Obj:Obj3D;
            Canvas:TCanvas;Width,Height:integer;
            Zoom:real;V:Vector3D);
```

{*Vẽ đối tượng theo thuật toán chọn lọc mặt sau.*

Trong đó:

+ Obj: chứa đối tượng 3D cần vẽ

+ Canvas: Vải vẽ (hay vùng đệm khung)

+ Width, Height: Kích thước của Canvas

+ Zooom: Hệ số tỷ lệ khi vẽ đối tượng (Hay hệ số thu phóng)

+ V: Vector hướng nhìn. Nếu Obj đã được chuyển sang hệ toạ độ quan sát O'UVN thì V=(0,0,-1)}

```
Var i,k,P,cx,cy:integer;
Poly:array of TPoint;
begin
cx:=Width div 2;cy:=Height div 2;
```

{*Duyệt qua tất cả các mặt của đối tượng*}

```
For k:=0 to Obj.SoMat-1 do
  if Tich_vo_huong(v,Obj.Mat[K].PhapVT)>= 0 then
  {Mặt khả kiến}
    begin
      setlength(Poly,Obj.Mat[K].Sodinh);{Thiết lập độ dài của
                                        mảng Poly bằng số đỉnh của đa giác}
      For i:=0 to Obj.Mat[K].Sodinh -1 do
      {Đưa toạ độ các đỉnh của đa giác vào Poly}
        begin
          P:=Obj.Mat[K].list[i];
          Poly[i].X:=round(Obj.dinh[P].x*zoom)+cx;
          Poly[i].Y:=-round(Obj.dinh[P].y*zoom)+cy;
        end;
      {Thiết lập màu cho bút tô trước khi tô}
      canvas.Brush.Color:=rgb(Obj.Mat[K].Color.R,
                  Obj.Mat[K].Color.G,Obj.Mat[K].Color.G);
      Canvas.Polygon(poly); {Tô đa giác với màu đã được thiết lập}
    end;
setlength(poly,0);
end;
```

Rõ ràng, thuật toán rất đơn giản và độ phức tạp tính toán không cao. Song khi sử dụng phải luôn đảm bảo rằng đối tượng có đặt tính là "***đặc và lồi***", nếu đối tượng không thoả mãn điệu kiện đó thì chúng ta phải áp dụng một thoật toán khác hay có những sửa đổi cần thiết để tránh sự thể hiện sai lạc.

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

$$\mathbf{z(M')=z(P')+(z(Q')-z(P'))t} \qquad (2.3.1)$$

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 (2.3.1) như sau:

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

$$\mathbf{z(x_i) = z(P_i')+z(P_j')*[(y_{scan} - y(P_i'))/(y(P_j')-y(P_i'))]} \quad (2.3.2)$$

{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 $z(x_i) = z(P_i')+z(P_j')*1/2$

Cài đặt minh hoạ cho giải thuật "vùng đệm độ sâu"

Từ những phân tính trên chúng ta có thể tiến hành khai báo các cấu trúc dữ liệu cần thiết và cài đặt cho thuật toán.

- Khai báo các cấu trúc dữ liệu cần thiết:

Sau đây là các khai báo cần thiết để cho phép lưu trữ một đối tượng 3D theo mô hình các mặt đa giác, cùng các khai báo cần thiết để tiến hành khử mặt khuất theo thuật toán z-Buffer theo ngôn ngữ Pascal trong môi trường của trình biên dịch Delphi

{Bắt đầu phần khai báo phục vụ cho giải thuật Z-buffer}

```
Type Z_BufferType=Array of Array of cardinal; {Kiểu bộ đệm Z, đây là một mảng động 2 chiều mà mỗi phần tử có kiểu cardinal, điều đó có nghĩa là vùng đệm độ sâu sẽ cho phép ta phân biệt được 4294967296 (2^32) mức sâu khác nhau}

NutPoly_Z=record      {Cấu trúc của một đỉnh của đa giác chiếu G' }
        x,y:Integer; {Toạ độ của ảnh trên mặt phẳng chiếu}
        z:real;      {Thành phần độ sâu đi kèm (là độ sâu của tạo ảnh)}
       end;
Polygon_Z =array of NutPoly_Z;  {Đa giác chiếu là một mảng
                                 động. Như một đa giác 2 chiều, song mỗi một đỉnh
                                 có chứa thêm thành phần độ sâu của đỉnh}
CanhCat_Z=record      {Cấu trúc của các cạnh đa giác được xây dựng
                       nhằm phục vụ cho quá trình tính giao điểm}
      y1,y2:Integer;  {Tung độ bắt đầu và kết thúc của một cạnh
                       (y1<=y2)}
```

```
        xGiao:real; {hoành độ xuất phát của cạnh. Song trong quá trình
                     tính toán nó sẽ là tung độ giao điểm của cạnh với
                     đường quét ngang}
        xStep:real; {Giá trị thay đổi của x khi y thay đổi 1 đơn vị, nó cho
                     biết độ dốc của cạnh}
        zGiao:real; {Giá trị độ sâu tại điểm xuất phát của cạnh. Song
                     trong quá trình tính toán nó sẽ là giá trị độ sâu của
                     giao điểm với đường quét ngang}
        zStep:real; {Giá trị độ sâu của giao điểm tiếp theo so với giá trị
                     độ sâu của giao điểm trước đó sẽ chênh lệch nhau
                     một khoảng là zStep}
    end;
DanhSachCanhCat_Z=array of CanhCat_Z; {Danh sách các cạnh
được tạo ra từ đa giác chiếu G', danh sách này nhằm phụ vụ cho quá trình tính
toán các giao điểm với đường quét cũng như độ sâu của mỗi giao điểm}
GiaoDiem_Z=record   {Lưu toạ độ giao điểm và độ sâu tương ứng với
                     giao điểm đó}
    x,y:Integer;      {Toạ độ giao điểm}
    z:real;           {Giá trị độ sâu}
    ChiSoCanh:integer; {Chỉ số cạnh cắt tạo ra giao điểm (Nhằm mục
                        đích khử các giao điểm thừa)}
    end;
DanhsachGiaoDiem_Z=array of GiaoDiem_Z;
{Kết thúc phần khai báo phục vụ cho giải thuật  Z-buffer}

Procedure DrawObj(Obj:Obj3D; Zmin,ZMax:Real;
                  Z_Buffer:Z_BufferType; Canvas:TCanvas;
                  Width,Height:integer; Zoom:real);
{Đầu vào:  + Đối tượng 3D chứa trong Obj
    + Giới hạn độ sâu trong không gian mà chương trình xử lý là từ Zmin đến
Zmax. Ta sẽ thực hiện ánh xạ các giá trị độ sâu tính được của các điểm trên đa
```

giác sang đoạn 0..4294967294. Biết rằng độ sâu Zmin ứng với 0 và Zmax ứng với 4294967294. (độ sâu 4294967295 làm giá trị mặc định cho các điểm nền

+ Z_Buffer: là ma trận chứa độ sâu các điểm ảnh của các đối tượng đã thể hiện trên Canvas (xem như là mặt phẳng chiếu). Nếu ta chưa vẽ đối tượng nào trước đó thì Z_Buffer được khởi động là 4294967295

Canvas: Tấm vải vẽ. Chúng ta sẽ thực hiện vẽ hình ảnh của đối tượng lên Canvas.

Width,Height: Là chiều rộng và cao của Canvas

+ Zoom: tỷ lệ thể hiện đối tượng lên Canvas sau khi thực hiện phép chiếu, ta có thể hiểu nôm na là tỷ lệ thu phóng.}

```
Var i,k,P,cx,cy:integer;
Poly:Polygon_Z;
CuongDoSang:Real;
Color:Tcolor;
Begin
  cx:=Width div 2;cy:=Height div 2;
  For k:=0 to Obj.SoMat-1 do {Duyệt qua tất cả các mặt đa giác}
    begin
      setlength(Poly,Obj.Mat[K].Sodinh);
     {Thiết lập số phần tử của Poly bằng số đỉnh của mặt mà nó sắp chứa}
      For i:=0 to Obj.Mat[K].Sodinh -1 do
    {Duyệt qua tất cả các đỉnh của mặt và thiết lập giá trị cho mỗi đỉnh
     của Poly}
        begin
          P:=Obj.Mat[K].list[i]; {Đỉnh thứ i trong đa giác K sẽ
                                  là đỉnh thứ P trong danh sách đỉnh của Obj}
        {Dùng phép chiếu trực giao để chiếu điểm Obj.dinh[P] xuống mặt
        phẳng OXY ta được tọa độ ảnh là (Obj.dinh[P].y,Obj.dinh[P].x),
        rồi sau đó phóng theo tỷ lệ là Zoom và tịnh tiến theo vector (cx,cy)
        nhằm giúp đưa hình ảnh ra vùng giữa Canvas}
          Poly[i].X:=round(Obj.dinh[P].x*zoom)+cx;
```

```
          Poly[i].Y:=-round(Obj.dinh[P].y*zoom)+cy;
          Poly[i].Z:=((Obj.dinh[P].z-ZMin)/(ZMax-Zmin)
                    *4294967294); //MaxCardinal=4294967295
```

{*Giá trị độ sâu của đỉnh Poly[i] là giá trị Obj.dinh[P].z song được ánh xạ vào đoạn 0..4294967294*}

```
        end;
Color:=RGB(Obj.Mat[K].Color.R,Obj.Mat[K].Color.G,
            Obj.Mat[K].Color.B);

 FillPolygon3D(Poly,Color,Z_Buffer,CanVas);
end;
setlength(poly,0);
end;
Procedure FillPolygon3D(Poly:Polygon_Z;Color:TColor;
              Z_Buffer:Z_BufferType;Canvas:TCanvas);
```

{*Thủ tục tô màu một đa giác theo thuật toán Z_Buffer*}

```
var L,H,ND,NG,i,j,Y,MaxY,MinY:integer;
D:DanhSachCanhCat_Z;
G:DanhsachGiaoDiem_Z;
Z_BufferW,Z_BufferH:Integer;
```

{*L,H:Giới hạn chỉ số của mảng Poly*

D:Danh sách các cạnh được tạo ra từ Poly, chứa những thông tin cần thiết để tính giao điểm và độ sâu của giao điểm một cách nhanh chóng

ND: Số phần tử của mảng D

G: Chứa danh sách các giao điểm có được sau mỗi lần dòng quét thay đổi

NG:số phần tử của mảng G}

```
Procedure TaoDanhSachCanhCat;
```

{*Thủ tục này tạo ra danh sách D, là danh sách các cạnh của đa giác từ thông tin đầu vào Poly*}

```
Var i,d1,d2,Dem,Dy,Cuoi:integer;
begin
```

```
{Xác định số cạnh của đa giác}
 If (Poly[L].x<>Poly[H].x)or
     (Poly[L].y<>Poly[H].y) then
   begin
     ND:=H-L+1;
     setlength(D,ND);
     Cuoi:=H;
   end
 else
   begin
     ND:=H-L;
     setlength(D,ND);
     Cuoi:=H-1;
   end;
 Dem:=0;
 {Tạo ra các cạnh}
 For i:=L to Cuoi do
   begin
     If i<H then j:=i+1 else j:=L;
 {Xác định điểm đầu và điểm cuối của cạnh, điểm đầu là điểm có giá trị y nhỏ}
     If Poly[i].y<=Poly[j].y then
       begin d1:=i;d2:=j end
     else
       begin d1:=j;d2:=i end;
     D[dem].y1:=Poly[d1].y;D[dem].y2:=Poly[d2].y;
     {Lưu trữ tung độ xuất phát và kết thúc}
     D[dem].xGiao:=Poly[d1].x;
{Tung độ xuất phát. Khởi đầu thì (D[dem].y1,D[dem].xGiao) chính là toạ độ
của điểm đầu của cạnh}
     D[dem].zGiao:=Poly[d1].z;
{Độ sâu của giao điểm tại điểm điểm đầu của cạnh}
    Dy:=(Poly[d2].y-Poly[d1].y);
```

```
{Độ chênh lệch tung độ của điểm đầu và điểm cuối}
     If Dy<>0 then
      begin
        D[dem].xStep:=(Poly[d2].x-Poly[d1].x)/Dy;
        D[dem].zStep:=(Poly[d2].z-Poly[d1].z)/Dy;
 {Từ độ chênh lệch Dy ta suy ra gia trọng của x và độ sâu z  khi giá trị y tăng 1
đơn vị. Nếu khi dòng quét đi qua điểm đầu thì toạ độ giao điểm là
(D[dem].y1,D[dem].xGiao) với độ sâu là D[dem].zGiao, nếu sau đó dòng quét
tăng 1 đơn vị thì rõ ràng  toạ độ giao điểm sẽ là
(D[dem].y1+1,D[dem].xGiao+D[dem].xStep) và độ sâu sẽ là
(D[dem].zGiao+D[dem].zStep)}
       end
     else
       begin
         D[dem].xStep:=0;
         D[dem].zStep:=0;
       end;
     Dem:=Dem+1;
  end;
end;
Procedure TaoDanhSachGiaoDiem;
{Tạo danh sách các giao điểm với đường quét có tung độ y hiện thời}
Var i:integer;
Begin
Setlength(G,ND);
NG:=0;
{Duyệt qua tất cả các cạnh}
for i:=0 to ND-1 do
  begin
    If (D[i].y1<=y)and(y<=D[i].y2) then
     {Có giao điểm với đường quét y}
      Begin
```

```
          {Lưu lại toạ độ giao điểm và độ sâu}
           G[NG].x:=round(D[i].xGiao);
           G[NG].y:=y;
           G[NG].z:=D[i].zGiao;
           G[NG].ChiSoCanh:=i;
           {Chỉ số cạnh đã tạo ra giao điểm. Nhằm phục vụ cho quá trình lọc
            bỏ các giao điểm không cần thiết}
           {Lưu lại Tung độ và độ sâu của giao điểm với đường quét tiếp theo
          (y+1) vào chính D[i].xGiao và D[i].zGiao}
           D[i].xGiao:=D[i].xGiao+D[i].xStep;
           D[i].zGiao:=D[i].zGiao+D[i].zStep;
           NG:=NG+1;
        end;
    end;
end;
Procedure SapXepVaLoc;
{Sắp xếp lại các giao điểm và lọc bỏ các giao điểm thừa}
Var i,j,C1,C2:integer;
Tg:GiaoDiem_Z;
Begin
  {Sắp xếp lại các giao điểm}
   for i:=0 to NG-2 do
      For j:=i+1 to NG-1 do
        If G[i].x>G[j].x then
          begin
            Tg:=G[i];G[i]:=G[j];G[j]:=Tg;
          end;
  i:=0;
  {Khử những Giao điểm thừa}
  While i<(NG-2) do
   begin
    If G[i].x=G[i+1].x then {2 giao điểm trùng nhau}
```

```
      begin
        C1:=G[i].ChiSoCanh;
        C2:=G[i+1].ChiSoCanh;
        {C1 và C2 là hai cạnh đã tạo nên 2 giao điểm trùng nhau đang xét}
        If (D[C1].y1<>D[C2].y1)and(D[C1].y2<>D[C2].y2))
        or(D[C1].y1=D[C1].y2)or(D[C2].y1=D[C2].y2) then
         {Xoá bớt một giao điểm nếu như: 2 cạnh tạo nên 2 giao điểm này
         nằm về hai phía của đường quét hoặc có một cạnh là nằm ngang}
          begin
            For j:=i to NG-2 do G[j]:=G[j+1];
            NG:=NG-1;
          end;
      end;
     i:=i+1;
   end;
end;
Procedure ToMauCacDoan;
{Thực hiện tô màu các đoạn thẳng là phần giao của đường quét với đa giác.
Đó là các đoạn x1x2, x3x4,…}
Var i,x,K:integer;Dz:real;
Z:Cardinal;
begin
  i:=0;
  While i<NG-1 do
    begin
      K:=G[i+1].x - G[i].x;
      If k<>0 then Dz:=(G[i+1].z-G[i].z)/K
      else Dz:=0;
      For x:=G[i].x to G[i+1].x do
      {Với mỗi đoạn ta thực hiện tính độ sâu của từng điểm rồi so sánh với
      giá trị có trong Z_Buffer}
        begin
```

```
                If (0<=x)and(x<=Z_BufferW)and(0<=y)
                    and(y<=Z_BufferH) then
                  begin
                    z:=round(G[i].z);
                    If Z_Buffer[x,G[i].y]>Z then
```

{So sánh độ sâu của điểm tính được với độ sâu đã có }

{Nếu độ sâu của điểm tính được nhỏ hơn độ sâu đã có trong Z_Buffer thì rõ ràng là điểm tính được ở gần hơn điểm đã vẽ trước đó trong vùng đệm Z và Canvas}

```
                      Begin
                        Canvas.Pixels[x,G[i].y]:=Color;
```

{*Vẽ điểm lên Canvas*}

```
                        Z_Buffer[x,G[i].y]:=Z;
```

{*Cập nhật độ sâu của điểm vừa vẽ vào vùng đệm Z*}

```
                      end;
                  end;
                G[i].z:=G[i].z+Dz;
```

{*Gán giá trị độ sâu của điểm tiếp theo vào trong G[i].z*}

```
              end;
            i:=i+2;
        end;
    end;
```

{*Thủ tục chính*}

```
Begin
L:=low(Poly);
H:=High(Poly);
```

{*Xác định giới hạn trên và giới hạn dưới của Poly*}

```
Z_BufferW:=high(Z_Buffer);
```

{*Xác định các chiều của ma trận Z_Buffer*}

```
Z_BufferH:=high(Z_Buffer[0]);
```

{*Z_BufferW+1:Chiều rộng (từ 0..Z_BufferW)*

Z_BufferH+1:Chiều cao (từ 0..Z_BufferH)}

```
{ Tìm giá trị y lớn nhất và nhỏ nhất của đa giác Poly để từ đó cho dòng quét thực hiện
quét từ trên min đến max}
MaxY:=Poly[L].y;
MinY:=MaxY;
For i:=L+1 to H do
  if MaxY<Poly[i].y then MaxY:=Poly[i].y
  else If MinY>Poly[i].y then MinY:=Poly[i].y;
TaoDanhSachCanhCat; {Tạo danh sách các cạnh của đa giác Poly với các tham
số thiết lập nhằm giúp cho việc tính toán giao điểm được dễ dàng}
For y:=MinY to MaxY do  {Cho dòng quét chạy từ MinY đến MaxY }
  begin
     TaoDanhSachGiaoDiem;  {Tìm danh sách các giao điểm của đường quét y
                           với các cạnh của Poly}
     SapXepVaLoc; {Sắp xếp lại các giao điểm và lọc bỏ các giao điểm thừa}
     ToMauCacDoan;  {Dựa vào các giao điểm để xác định ra các đoạn nằm
                    trong đa giác, từ đó tô màu từng điểm trên đoạn đó dựa
                 vào độ sâu so sánh với giá trị độ sâu tương ứng trên Z_Buffer}
    end;
Setlength(D,0);     {Giải phóng mảng D}
Setlength(G,0);     {Giải phóng mảng G}
end;
```

BÀI TẬP

1. Cài đặt cho thuật giải Depth-Sorting

Cài đặt chương trình cho phép biểu diễn và quan sát vật thể 3D theo mô hình "các mặt đa giác" trong đó sử dụng thuật giải Depth-Sorting để khử các mặt khuất

2. Cài đặt cho thuật giải chọn lọc mặt sau

Cài đặt chương trình cho phép biểu diễn và quan sát vật thể 3D theo mô hình "các mặt đa giác" trong đó sử dụng thuật giải chọn lọc mặt sau để khử các mặt khuất. Với đối tượng là các hình lập phương, tứ diện, bát diện, cầu,…

3. Cài đặt cho thuật giải vùng đệm độ sâu

Cài đặt chương trình cho phép biểu diễn và quan sát vật thể 3D theo mô hình "các mặt đa giác" trong đó sử dụng thuật giải chọn lọc mặt sau để khử các mặt khuất. Với đối tượng là các mặt cắt nhau, các hình lồi lõm bất kỳ.

CHƯƠNG VIII

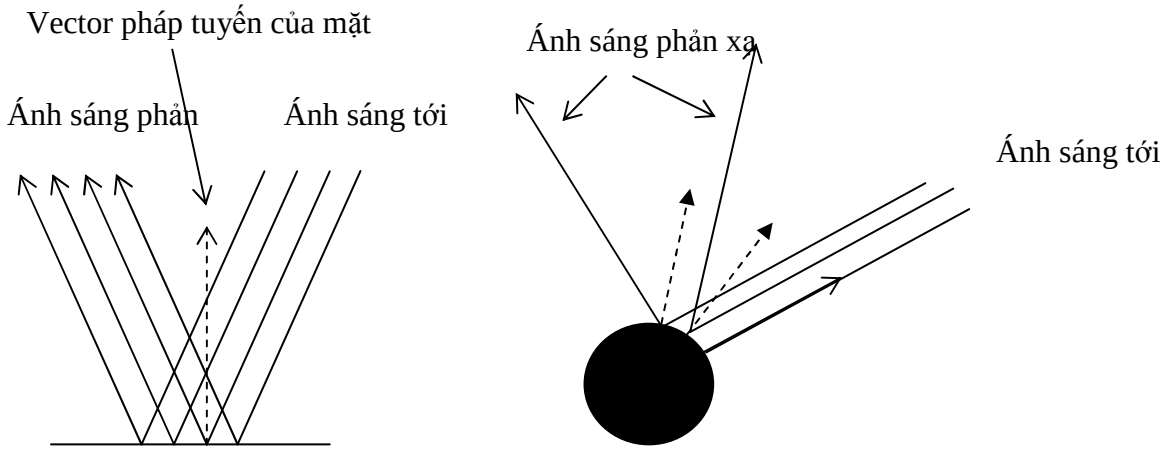
TẠO BÓNG VẬT THỂ 3D

8.1. KHÁI NIỆM

Khi biểu diễn các đối tượng 3 chiều, một yếu tố không thể bỏ qua để tăng tính thực của đối tượng đó là tạo bóng sáng cho vật thể. Để thực hiện được điều này, chúng ta cần phải lần lượt tìm hiểu các dạng nguồn sáng có trong tự nhiên, cũng như các tính chất đặc trưng khác nhau của mỗi loại nguồn sáng. Từ đó đưa ra các giải pháp kỹ thuật khác nhau nhằm thể hiện sự tác động của các nguồn sáng khác nhau lên đối tượng.

8.2. NGUỒN SÁNG XUNG QUANH

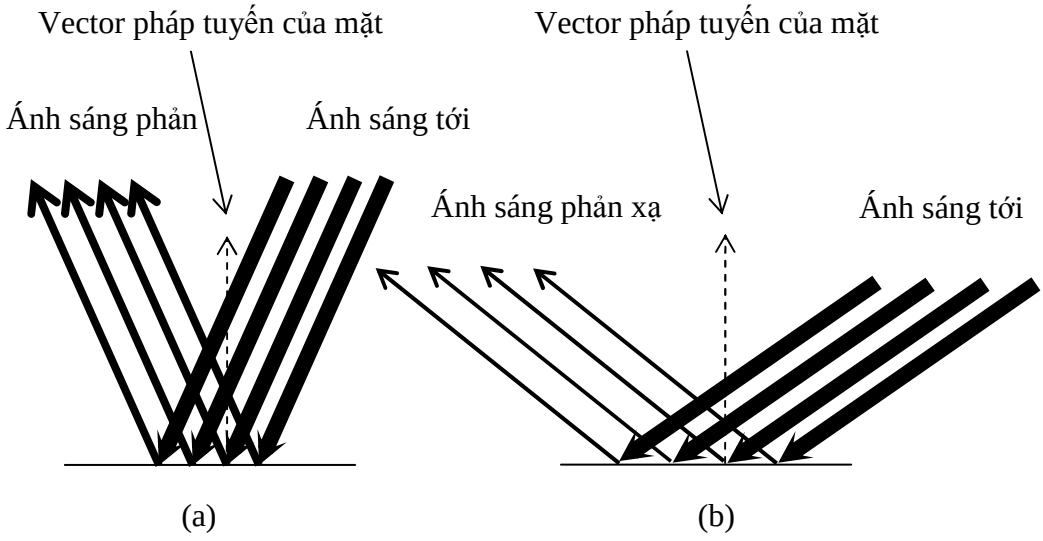
Ánh sáng xung quanh là mức sáng trung bình, tồn tại trong một vùng không gian. Một không gian lý tưởng là không gian mà tại đó mọi vật đều được cung cấp một lượng ánh sáng lên bề mặt là như nhau, từ mọi phía ở mọi nơi. Thông thường ánh sáng xung quanh được xác định với một mức cụ thể gọi là mức sáng xung quanh của vùng không gian mà vật thể đó cư ngụ, sau đó ta cộng với cường độ sáng có được từ các nguồn sáng khác để có được cường độ sáng cuối cùng lên một điểm hay một mặt của vật thể.



Hình 8.1. Sự phản xạ của ánh sáng

8.3. NGUỒN SÁNG ĐỊNH HƯỚNG

Nguồn sáng định hướng giống như những gì mà mặt trời cung cấp cho chúng ta. Nó bao gồm một tập các tia sáng song song, bất kể cường độ của chúng có giống nhau hay không. Có hai loại kết quả của ánh sáng định hướng khi chúng chiếu đến bề mặt là: khuyếch tán và phản chiếu. Nếu bề mặt phản xạ toàn bộ (giống như mặt gương) thì các tia phản xạ sẽ có hướng ngược với hướng của góc tới (Hình 8.1). Trong trường hợp ngược lại, nếu bề mặt là không phản xạ toàn phần (có độ nhám, xù xì) thì một phần các tia sáng sẽ bị toả đi các hướng khác hay bị hấp thụ, phần còn lại thì phản xạ lại, và lượng ánh sáng phản xạ lại này tỷ lệ với góc tới. Ở đây chúng ta sẽ quan tâm đến hiện tượng phản xạ không toàn phần vì đây là hiện tượng phổ biến (vì chỉ có những đối tượng được cấu tạo từ những mặt như mặt gương mới xảy ra hiện tượng phản xạ toàn phần), và đồng thời tìm cách tính cường độ của ánh sáng phản xạ trên bề mặt.



Hình 8.2. Sự phản xạ không toàn phần của ánh sáng

Trong hình 8.2 thể hiện sự phản xạ ánh sáng không toàn phần. Độ đậm nét của các tia ánh sáng tới thể hiện cường độ sáng cao, độ mảnh của các tia phản xạ thể hiện cường độ sáng thấp. Nói chung, khi bề mặt là không phản xạ toàn phần thì cường độ của ánh sáng phản xạ (hay tạm gọi là tia phản xạ) luôn bé hơn so với cường độ của ánh sáng tới (hay gọi là tia tới), và cường độ của tia phản xạ còn tỷ lệ với góc giữa tia tới với vector pháp tuyến của bề mặt, nếu góc này càng nhỏ thì cường độ phản xạ càng cao (hình II.2 (a)), nếu góc này lớn thì cường độ phản xạ rất thấp (hình II.2 (b)). Ở đây ta chỉ quan tâm đến thành phần ánh sáng khuyếch tán và tạm bỏ qua hiện tượng phản

xạ toàn phần. Để cho tiện trong việc tính toán ta tạm đổi hướng của tia tới thực sự, vậy bây giờ hướng của tia tới được xem là hướng ngược lại của tia sáng tới.

Nếu gọi θ là góc giữa tia tới với vector pháp tuyến của bề mặt thì Cos(θ) phụ thuộc vào tia tới a và vector pháp tuyến của mặt n theo công thức:

$$Cos(\theta) = \frac{\vec{a}.\vec{n}}{|\vec{a}|.|\vec{n}|} \qquad (8.1)$$

Trong công thức trên Cos(θ) bằng tích vô hướng của a và n chia cho tích độ lớn của chúng. Nếu ta đã chuẩn hoá độ lớn của các vector a và n về 1 từ trước thì ta có thể tính giá trị trên một cách nhanh chóng như sau:

Cos(θ) = tích vô hướng của $\vec{a}$ và $\vec{n}$ = a.x*n.x+a.y*n.y+a.z*n.z

Vì Cos(θ) có giá trị từ +1 đến -1 nên ta có thể suy ra công thức tính cường độ của ánh sáng phản xạ là:

Cường độ ánh sáng phản xạ = Cường độ của ánh sáng định hướng * [(Cos(θ)+1)/2] (8.2)

Trong đó [(Cos(θ)+1)/2] có giá trị trong khoảng từ 0 đến 1. Vậy qua công thức (8.1) và (8.2) chúng ta có thể tính được cường độ của ánh sáng phản xạ trên bề mặt khi biết được cường độ của ánh sáng định hướng cũng như các vector pháp tuyến của mặt và tia tới.

Cài đặt thuật toán

Dưới đây là phần trình bày các thủ tục phục vụ cho việc vẽ đối tượng 3D đặc lồi, theo thuật toán chọn lọc mặt sau có tính đến vấn đề chiếu sáng của nguồn sáng xung quanh và nguồn sáng định hướng.

```
Function Cuong_Do_Anh_Sang_Dinh_Huong(v,n:Vector3D):real;
{Thủ tục tính cường độ ánh sáng phản xạ trên bề mặt của đa giác khi biết được tia tới
v và vector pháp tuyến n}
var    s,t:real;
begin
   s:=sqrt(v.x*v.x+v.y*v.y+v.z*v.z)*sqrt(n.x*n.x+n.y*n.y+n.z*n.z);
   {Gán S bằng tích của |v|*|n|}
```

```
   if s=0 then {Một trong hai vector bằng 0 do đó tạm xem cường độ sáng bằng 1}
     begin Cuong_Do_Anh_Sang_Dinh_Huong:=1;end
   else
      Begin t:=tich_vo_huong(v,n); {Tính tích vô hướng của v và n}
        If t>0 then  {Nếu góc giữa v và n nằm trong khoảng từ 0 đến 90 độ thì}
           Cuong_Do_Anh_Sang_Dinh_Huong:=(T/s)
        else
           Cuong_Do_Anh_Sang_Dinh_Huong:=0;
      end;
end;
Procedure DrawObj_FilterRearFace(Obj:Obj3D; Canvas:TCanvas;
                                 Width,Height:integer;
                                  Zoom:real;
                                 AnhSangNen,AnhSangDinhHuong:real;
                                 VectorChieuSang:vector3D; V:Vector3D);
```

{Vẽ đối tượng 3D đặc lồi theo thuật toán chọn lọc mặt sau có tính đến vấn đề chiếu sáng của nguồn sáng xung quanh và nguồn sáng định hướng.

Trong đó:

+ Obj: chứa đối tượng 3D cần vẽ

+ Canvas: Vải vẽ (hay vùng đệm khung)

+ Width, Height: Kích thước của Canvas

+ Zooom: Hệ số tỷ lệ khi vẽ đối tượng (Hay hệ số thu phóng)

+ V: Vector hướng nhìn. Nếu Obj đã được chuyển sang hệ toạ độ quan sát O'UVN thì V=(0,0,-1)

+ AnhSangNen: Giá trị cường độ của ánh sáng xung quanh mà đối tượng có thể thu nhận được

+ AnhSangDinhHuong: Giá trị cường độ của ánh sáng định hướng mà đối tượng có thể thu nhận được

**Chú ý: AnhSangNen + AnhSangDinhHuong <=1. Ở đây ta xem tổng cường độ của các nguồn sáng tạo ra giới hạn trong khoảng 0..1. Từ đó chúng ta có thể điều chỉnh cường độ chiếu sáng của các nguồn sáng bằng cách tăng hệ số cường độ của nó song vẫn phải luôn luôn thoả mãn tổng của chúng nhỏ hơn hay bằng 1. Khi một mặt*

nhận được tổng cường độ sáng là 1 từ các nguồn sáng khác nhau cung cấp thì mặt sẽ cho màu thực của nó. Nếu tổng cường độ sáng mà nó thu được từ các nguồn sáng nhỏ hơn 1 mặt sẽ hơi tối đi.

+ VectorChieuSang: Đây là vector biểu diễn tia tới (chú ý nó có hướng ngược với hướng của ánh sáng chiếu tới như đã nói trong phần lý thuyết}

```
Var i,k,P,cx,cy:integer;
Poly:array of TPoint;
CuongDoSang:Real;
R,G,B:byte;
Begin
cx:=Width div 2;cy:=Height div 2;
For k:=0 to Obj.SoMat-1 do
   if Tich_vo_huong(v,Obj.Mat[K].PhapVT)>= 0 then
      begin
```

{*Thiết lập đa giác là hình chiếu của mặt xuống mặt phẳng OXY (có tịnh tiến và đổi hướng trục Y)*}

```
         setlength(Poly,Obj.Mat[K].Sodinh);
         For i:=0 to Obj.Mat[K].Sodinh -1 do
            begin
               P:=Obj.Mat[K].list[i];
               Poly[i].X:=round(Obj.dinh[P].x*zoom)+cx;
               Poly[i].Y:=-round(Obj.dinh[P].y*zoom)+cy;
```

{*Toạ độ của đỉnh sau khi chiếu là (Obj.dinh[P].x,Obj.dinh[P].y), song được biến đổi tỷ lệ với hệ số là zoom rồi đổi hướng trục Y và tịnh tiến theo vector (cx,cy)*}

```
            end;
```

{*Tính cường độ sáng mà mặt nhận được: bằng tổng cường độ sáng do nguồn sáng xung quanh (ánh sáng nền) và nguồn sáng định hướng cung cấp*}

```
         CuongDoSang:=AnhSangNen  +  AnhSangDinhHuong *
         Cuong_Do_Anh_Sang_Dinh_Huong(VectorChieuSang,Obj.Mat[K].PhapVT);
```

{*Ở đây cường độ sáng mà mặt nhận được do nguồn sáng định hướng cung cấp phụ thuộc không chỉ vào cường độ sáng mà nguồn phát ra, mà còn phụ thuộc vào hướng đón ánh sáng của mặt và được biểu diễn bởi biểu thức: AnhSangDinhHuon*Cuong_Do_Anh_Sang_Dinh_Huong(VectorChieuSang,Obj.Mat[K].PhapVT)*}

```
R:=round(Obj.Mat[K].Color.R*CuongDoSang);
G:=round(Obj.Mat[K].Color.G*CuongDoSang);
B:=round(Obj.Mat[K].Color.B*CuongDoSang);
```

{*Thiết lập màu sắc cho mặt bằng cách: lấy cường độ màu sắc mặt định của mặt Obj.Mat[K].Color, nhân với cường độ sáng mà nó nhận được trong thực tế tính toán được CuongDoSang. Từ đó ta thấy, nếu mặt nhận được đầy đủ ánh sáng (cường độ sáng =1) thì mặt sẽ có màu mặt định của nó (xác định khi thiết kế đối tượng), ngược lại thì mặt sẽ có màu sắc tối hơn. Nếu mặt không được chiếu sáng từ các nguồn sáng (cường độ sáng =0) thì mặt sẽ có màu đen*}

```
canvas.Brush.Color :=rgb(R,G,B);
Canvas.Pen.Color:=canvas.Brush.Color;
Canvas.Polygon(poly);
```

{*vẽ đa giác với màu sắc đã được xác định trước bởi bút tô (Brush.Color) và bút vẽ (Pen.Color)*}

```
    end;
setlength(poly,0);
end;
```

8.4. NGUỒN SÁNG ĐIỂM

Nguồn sáng định hướng là tương đương với nguồn sáng điểm đặt ở vô tận. Nhưng khi nguồn sáng điểm được mang đến gần đối tượng thì các tia sáng từ nó phát ra không còn song song nữa mà được toả ra theo mọi hướng theo dạng hình cầu. Vì thế, các tia sáng sẽ rơi xuống các điểm trên bề mặt dưới các góc khác nhau. Giả sử vector pháp tuyến của mặt là n=(xn, yn, zn), điểm đang xét có toạ độ là (x_0, y_0, z_0) và nguồn sáng điểm có tọa độ là (plx, ply, plz) thì ánh sáng sẽ rọi đến điểm đang sét theo vector (x_0-plx, y_0-ply, z_0-plz), hay tia tới:

a = (plx - x_0, ply - y_0,plz - z_0).

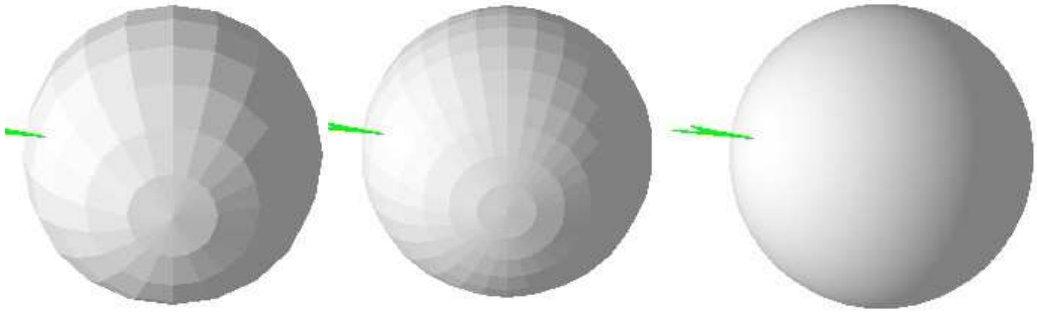
Từ đó cường độ sáng tại điểm đang xét sẽ phụ thuộc vào Cos(θ) giữa n và a như đã trình bày trong phần nguồn sáng định hướng.

Vậy với nguồn sáng định hướng, chúng ta cần tính tia tới cho mọi điểm trên mặt, từ đó kết hợp với vector pháp tuyến của mặt để tính được cường độ sáng tại điểm đó, nếu tính toán trực tiếp thì có thể mất khá nhiều thời gian do phải tính vector a và tính Cos(θ) thông qua công thức (8.1) với tất cả các điểm trên mặt. Nên nhớ rằng trong tình hướng nguồn sáng điểm thì chúng ta buộc lòng phải tính Cos(θ) thông qua công thức (8.1) vì vector a sẽ thay đổi khi mặt hay nguồn sáng thay đổi (trừ khi mặt tĩnh, song nếu mặt tĩnh và nguồn sáng cố định thì suy ra chúng ta chỉ cần tính cường độ sáng một lần).

8.5. MÔ HÌNH BÓNG GOURAUD

Mô hình bóng Gouraud là một phương pháp vẽ bóng, tạo cho đối tượng 3D có hình dáng cong có một cái nhìn có tính thực hơn. Phương pháp này đặt cơ sở trên thực tế sau: đối với các đối tượng 3D có bề mặt cong thì người ta thường xấp sỉ bề mặt cong của đối tượng bằng nhiều mặt đa giác phẳng, ví dụ như một mặt cầu có thể sấp sỉ bởi một tập các mặt đa giác phẳng có kích thước nhỏ sắp xếp lại, khi số đa giác xấp xỉ tăng lên (có nghĩa là diện tích mặt đa giác nhỏ lại) thì tính thực của mặt cầu sẽ tăng, sẽ cho ta cảm giác mặt cầu trông tròn trịa hơn, mịn và cong hơn. Tuy nhiên, khi số đa giác xấp xỉ một mặt cong tăng thì khối lượng tính toán và lưu trữ cũng tăng theo tỷ lệ thuận theo số mặt, điều đó dẫn đến tốc độ thực hiện sẽ trở nên chậm chạp hơn. Chúng ta hãy thử với một ví dụ sau: Để mô phỏng một mặt cầu người ta xấp xỉ nó bởi 200 mặt thì cho ta một cảm giác hơi gồ ghề, nhưng với 450 mặt thì ta thấy nó mịn và tròn trịa hơn, song khi số mặt là 16200 thì cho ta cảm giác hình cầu rất tròn và mịn (hình 8.3). Tuy hình ảnh mặt cầu với 16200 mặt đa giác thì mịn hơn so với 200 mặt, song lượng tính toán phải thực hiện trên mỗi đa giác cũng tăng lên gấp 16200/200= 81 lần.

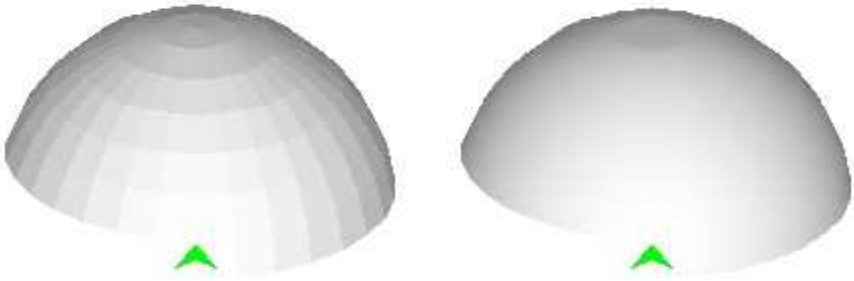
Song vấn đề vẫn còn nảy sinh một khi ta phóng lớn hay thu nhỏ vật thể. Nếu ta phóng lớn thì rõ ràng là các đa giác cũng được phóng lớn theo cùng tỷ lệ, dẫn đến hình ảnh về các mặt đa giác lại hiện rõ và gây ra cảm giác không được trơn mịn. Ngược lại, khi ta thu nhỏ thì nếu số đa giác xấp xỉ lớn thì sẽ dẫn đến tình trạng các đa giác quá nhỏ, chồng chất lên nhau không cần thiết.



200 mặt 450 mặt 16200 mặt

Hình 8.3

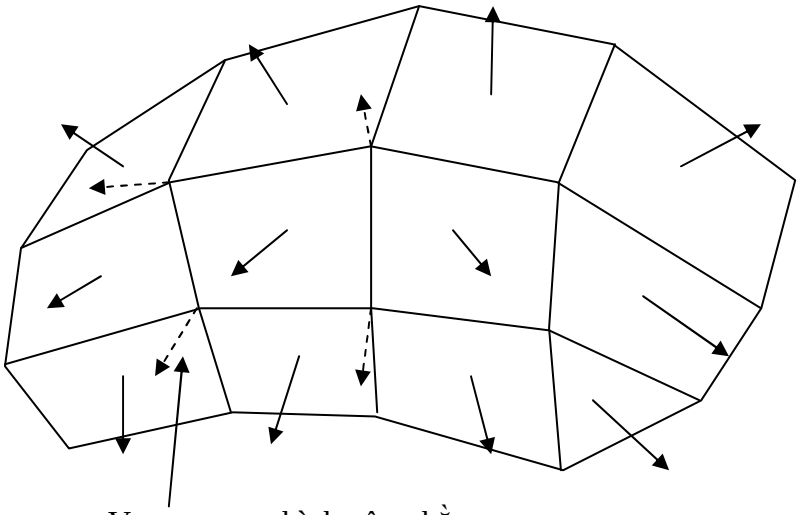
Để giải quyết vấn đề trên, chúng ta có thể tiến hành theo phương pháp tô bóng Gouraud. Mô hình bóng Gouraud tạo cho đối tượng một cái nhìn giống như là nó có nhiều mặt đa giác bằng cách vẽ mỗi mặt không chỉ với một cường độ sáng mà vẽ với nhiều cường độ sáng khác nhau trên các vùng khác nhau, làm cho mặt phẳng nom như bị cong. Bởi thực chất ta cảm nhận được độ cong của các mặt cong do hiệu ứng ánh sáng khi chiếu lên mặt, tại các điểm trên mặt cong sẽ có pháp vector khác nhau nên sẽ đón nhận và phản xạ ánh sáng khác nhau, từ đó chúng ta sẽ cảm nhận được các độ sáng khác nhau trên cùng một mặt cong.



Tô bóng thường Tô bóng theo Gouraud

Hình 8.4

Thường thì mỗi mặt đa giác có một vector pháp tuyến, và như phần trên đã trình bày, vector pháp tuyến đó được dùng để tính cường độ của ánh sáng phản xạ trên bề mặt của đa giác từ đó suy ra cường độ sáng của mặt. Tuy nhiên mô hình Gouraud lại xem một đa giác không chỉ có một vector pháp tuyến, mà mỗi đỉnh của mặt đa giác lại có một vector pháp tuyến khác nhau, và từ vector pháp tuyến của các đỉnh chúng ta sẽ nội suy ra được vector pháp tuyến của từng điểm trên mặt đa giác, từ đó tính được cường độ sáng của điểm. Như thế, các điểm trên cùng một mặt của đa giác sẽ có cường độ sáng khác nhau và cho ta cảm giác mặt đa giác không phải là mặt phẳng mà là mặt cong.



Hình 8.5

Thực chất thì mỗi mặt đa giác chỉ có một vector pháp tuyến, song phương pháp Gouraud tính toán vector trung bình tại mỗi đỉnh của đa giác bằng cách: lấy trung bình cộng các vector pháp tuyến của các đa giác có chứa đỉnh đang xét.

Việc nội suy vector pháp tuyến của từng điểm trên mặt đa giác được thực hiện tương tự như việc nội suy độ sâu trong giải thuật "vùng đệm độ sâu" đã được trình bày trong chương trước.

Cài đặt thuật toán

Dưới đây là phần trình bày các thủ tục phục vụ cho việc vẽ đối tượng 3D đặc lồi và cong, theo thuật toán chọn lọc mặt sau có tính đến vấn đề chiếu sáng của nguồn sáng xung quanh và nguồn sáng định hướng, song mỗi đa giác sẽ được tô bóng theo phương pháp Gouraud.

{Bắt đầu phần khai báo phục vụ cho giải thuật tô đa giác theo phương pháp Gouraud}

```
Type NutPolyGourand=record {1 đỉnh của đa giác chiếu (là ảnh của mặt
                                đa giác xuống mặt phẳng OXY}
       N:Vector3D;      {Pháp vector tại 1 đỉnh của đa giác}
       x,y:Integer;     {Toạ độ của đỉnh}
     end;
```

```
      PolygonGourand =array of NutPolyGourand;
      {mảng động dùng để chứa các đỉnh của đa giác chiếu}
      CanhCat=record {Một cạnh của đa giác được tạo ra nhằm phục vụ cho
                      quá trình tính giao điểm nhanh}
         y1,y2:Integer;
         xGiao:real;
         XStep:real;
         NGiao:Vector3D; {Pháp vector tại đỉnh, song nó sẽ được chứa pháp
                          vector tai giao điểm với đường quét}
         NStep:Vector3D; {độ biến thiên của Pháp vector khi di chuyển dọc
                          theo cạnh, mỗi lần y thay đổi 1 đơn vị thì pháp
                          vector thay đổi một lượng là NStep}
       end;
      DanhSachCanhCat=array of CanhCat;
      GiaoDiem=record   {Cấu trúc chứa giao điểm của đường quét ngang
                         với cạnh đa giác chiếu }
        x,y:Integer;    {Toạ độ giao điểm}
        NGiao:Vector3D;    {Pháp vector tại giao điểm}
        ChiSoCanh:integer;{Chỉ số cạnh đã tạo ra giao điểm}
       end;
      DanhsachGiaoDiem=array of GiaoDiem;
Procedure DrawObjGouraud_FilterRearFace(Obj:Obj3D;
           Canvas:TCanvas;Width,Height:integer;Zoom:real;
           AnhSangNen,AnhSangDinhHuong:real;
           VectorChieuSang:vector3D;V:Vector3D);
{Vẽ đối tượng 3D đặc lồi và cong}
Var i,j,k,Dem,P,Q,cx,cy:integer;
    Poly:PolygonGourand;
    DinhLinkMat:array of Record{Chứa Danh sách các mặt có liên kết
                                đến một đinh}
        SoMat:Integer;
```

```
       A:array of integer;
     end;
  CMLD:array of integer;     {Số mặt có liên kết với đỉnh thứ i}
begin
cx:=Width div 2;cy:=Height div 2;
Setlength(DinhLinkMat,Obj.Sodinh);
Setlength(CMLD,Obj.Sodinh);
For i:=0 to obj.Sodinh-1 do CMLD[i]:=0;
{Xác định mỗi đỉnh có bao nhiêu mặt liên kết đến}
For i:=0 to obj.SoMat -1 do
  For j:=0 to obj.mat[i].Sodinh-1 do
    begin
      K:=obj.mat[i].List[j];
      CMLD[k]:=CMLD[k]+1; {Số mặt liên kết đến đỉnh thứ k được tăng lên
                           khi có một mặt có liên kết đến nó }
    end;
{Thiết lập danh sách các măt liên kết đến từng đỉnh của đối tượng}
For i:=0 to obj.Sodinh-1 do
  begin
    setlength(DinhLinkMat[i].A,CMLD[i]);{Số mặt liên kết với đỉnh i
                                         là CMLD[i]}
     DinhLinkMat[i].SoMat:=0;
  end;
{Quá trình xác định rõ những mặt nào  liên kết với đỉnh i}
For i:=0 to obj.SoMat -1 do
  For j:=0 to obj.mat[i].Sodinh-1 do
    begin
      K:=obj.mat[i].List[j];
      Dem:=DinhLinkMat[K].SoMat;
      DinhLinkMat[K].A[Dem]:=i;
      DinhLinkMat[K].SoMat:=DinhLinkMat[K].SoMat+1;
    end;
```

```
Setlength(CMLD,0);
For k:=0 to Obj.SoMat-1 do
  if Tich_vo_huong(v,Obj.Mat[K].PhapVT)>= 0 then
    begin
      setlength(Poly,Obj.Mat[K].Sodinh);
      For i:=0 to Obj.Mat[K].Sodinh -1 do
        begin
         {thiết lập các thuộc tính của đỉnh thứ i của Poly (đa giác chiếu) }
         P:=Obj.Mat[K].list[i];
         Poly[i].X:=round(Obj.dinh[P].x*zoom)+cx;
         Poly[i].Y:=-round(Obj.dinh[P].y*zoom)+cy;
         {Tính Vector pháp tuyến tại đỉnh Poly bằng cách tính tổng của các
         vector pháp tuyến của các mặt có liên kết với đỉnh đó}
          Poly[i].N.x :=0;Poly[i].N.y :=0;Poly[i].N.z :=0;
         For j:=0 to DinhLinkMat[P].SoMat-1 do
           begin
             Q:=DinhLinkMat[P].A[j];//Mat ke voi dinh P
             Poly[i].N.x:=Poly[i].N.x+obj.Mat[Q].PhapVT.x;
             Poly[i].N.y:=Poly[i].N.y+obj.Mat[Q].PhapVT.y;
             Poly[i].N.z:=Poly[i].N.z+obj.Mat[Q].PhapVT.z;
           end;
      end;
  FillPolygonGourand(poly,VectorChieuSang,AnhSangNen,
                AnhSangDinhHuong,CanVas,Obj.Mat[K].Color);
{Tô đa giác Poly theo thuật toán tô đa giác theo dòng quét, song có nội suy vector
pháp tuyến cho mỗi điểm, và tính cường độ sáng của mỗi điểm trong đa giác dựa vào
vector pháp tuyến tại điểm đó.}
 end;
setlength(poly,0);
For i:=0 to obj.Sodinh-1 do
  setlength(DinhLinkMat[i].A,0);
Setlength(DinhLinkMat,0);
```

```
end;
Procedure FillPolygonGourand(Poly:PolygonGourand;
           Vector_Chieu_Sang:Vector3D;
           Anh_Sang_Nen,Anh_Sang_Chieu:real;
           Canvas:TCanvas;Color:RGBColor);
```

{*Tô đa giác Poly theo thuật toán tô đa giác theo dòng quét, song có nội suy vector pháp tuyến cho mỗi điểm, và tính cường độ sáng của mỗi điểm trong đa giác dựa vào vector pháp tuyến tại điểm đó.*

Thủ tục này tương tự như thủ tục tô đa giác theo thuật toán Z-Buffer song thay vì nội suy độ sâu z của mỗi điểm, thì thủ tục này sẽ nội suy pháp vector của mỗi điểm, rồi dựa vào pháp vector của điểm đó mà tính ra cường độ sáng mà nó có được do nguồn sáng định hướng cung cấp.}

```
var L,H,ND,NG,i,j,Y,MaxY,MinY:integer;
```

{L,H:Gioi han chi so cua mang Poly

ND: So phan tu cua mang D:DanhSachCanhCat

NG:So phan tu cua mang G:DanhsachGiaoDiem}

```
D:DanhSachCanhCat;
G:DanhsachGiaoDiem;
Procedure TaoDanhSachCanhCat;
Var i,d1,d2,Dem,Kc,Cuoi:integer;
begin
  If (Poly[L].x<>Poly[H].x)or(Poly[L].y<>Poly[H].y) then
    begin
      ND:=H-L+1;
      setlength(D,ND);
      Cuoi:=H;
    end
else
    begin
      ND:=H-L;
      setlength(D,ND);
      Cuoi:=H-1;
```

```
    end;
Dem:=0;
For i:=L to Cuoi do
  begin
    If i<H then j:=i+1 else j:=L;
    If Poly[i].y<=Poly[j].y then
       begin d1:=i;d2:=j end
    else
       begin d1:=j;d2:=i end;
    D[dem].y1:=Poly[d1].y;D[dem].y2:=Poly[d2].y;
    D[dem].xGiao:=Poly[d1].x;
    D[dem].NGiao:=Poly[d1].N;
    Kc:=(Poly[d2].y-Poly[d1].y);
    If Kc<>0 then
      begin
        D[dem].XStep:=(Poly[d2].x-Poly[d1].x)/Kc;
        D[dem].NStep.x:=(Poly[d2].N.x-Poly[d1].N.x)/Kc;
        D[dem].NStep.y:=(Poly[d2].N.y-Poly[d1].N.y)/Kc;
        D[dem].NStep.z:=(Poly[d2].N.z-Poly[d1].N.z)/Kc;
      end
    else
      begin
        D[dem].XStep:=0;
        D[dem].NStep.x:=0;
        D[dem].NStep.y:=0;
        D[dem].NStep.z:=0;
      end;
   Dem:=Dem+1;
 end;
end;
Procedure TaoDanhSachGiaoDiem;
Var i,Dy:integer;
```

```
begin
Setlength(G,ND);
NG:=0;
for i:=0 to ND -1 do
  begin
    If (D[i].y1<=y)and(y<=D[i].y2) then
      begin
       Dy:=y-D[i].y1;
       G[NG].x:=round(D[i].xGiao+D[i].XStep*Dy);
       G[NG].y:=y;
       G[NG].NGiao.x:=D[i].NGiao.x+D[i].NStep.x*Dy;
       G[NG].NGiao.y:=D[i].NGiao.y+D[i].NStep.y*Dy;
       G[NG].NGiao.z:=D[i].NGiao.z+D[i].NStep.z*Dy;
       G[NG].ChiSoCanh:=i;
       NG:=NG+1;
      end;
    end;
end;
Procedure SapXepVaLoc;
Var i,j,C1,C2:integer;
Tg:GiaoDiem;
begin
  for i:=0 to NG-2 do
    For j:=i+1 to NG-1 do
      If G[i].x>G[j].x then
        begin
          Tg:=G[i];G[i]:=G[j];G[j]:=Tg;
        end;
  i:=0;
  {Khu nhung Giao Diem thua}
  While i<(NG-2) do
    begin
```

```
      If G[i].x=G[i+1].x then //Trung nhau
        begin
          C1:=G[i].ChiSoCanh;
          C2:=G[i+1].ChiSoCanh;
          If ((D[C1].y1<>D[C2].y1)and(D[C1].y2<>D[C2].y2))
          or (D[C1].y1=D[C1].y2)or(D[C2].y1=D[C2].y2) then
           //Xoa bo bot 1 giao diem
            begin
              For j:=i to NG-2 do G[j]:=G[j+1];
              NG:=NG-1;
            end;
        end;
       i:=i+1;
     end;
end;
Procedure ToMauCacDoan;
Var i,x,K:integer;CuongDoSang,Dx,Dy,Dz:real;
Re,Gr,Bl,Cd:byte;
begin
  If Red then Re:=1 else Re:=0;
  If Green then Gr:=1 else Gr:=0;
  If Blue then Bl:=1 else Bl:=0;
  i:=0;
  While i<NG-1 do
    begin
      K:=G[i+1].x - G[i].x;
      If k<>0 then
        begin
          Dx:=(G[i+1].NGiao.x-G[i].NGiao.x)/K;
          Dy:=(G[i+1].NGiao.y-G[i].NGiao.y)/K;
          Dz:=(G[i+1].NGiao.z-G[i].NGiao.z)/K;
        end
```

```
      else
        begin
          Dx:=0;Dy:=0;Dz:=0;
        end;
      For x:=G[i].x to G[i+1].x do
        begin
          CuongDoSang:=Anh_Sang_Nen + Anh_Sang_Chieu*
           Cuong_Do_Anh_Sang_Dinh_Huong(Vector_Chieu_Sang,
                                        G[i].NGiao);
```

{*Cường độ sáng tại mỗi điểm được tính bằng tổng của* ***cường độ ánh sáng nền*** *cộng với cường độ có được từ* ***nguồn sáng định hướng***, *song để tính được cường độ sáng có được từ nguồn định hướng cung cấp thì chúng ta dựa vào tia tới (Vector_Chieu_Sang) và pháp vector của điểm G[i].Ngiao.*}

```
           Cd:=round(255*CuongDoSang);
          Canvas.Pixels[x,G[i].y]:=rgb(Cd*Re,Cd*Gr,Cd*Bl);
```

{Nội suy pháp vector của điểm tiếp theo và gán vào G[i].NGiao}

```
          G[i].NGiao.x:=G[i].NGiao.x+dx;
          G[i].NGiao.y:=G[i].NGiao.y+dy;
          G[i].NGiao.z:=G[i].NGiao.z+dz;
        end;
      i:=i+2;
   end;
end;
Begin
  L:=low(Poly); H:=High(Poly);
  MaxY:=Poly[L].y;MinY:=MaxY;
  For i:=L+1 to H do
   if MaxY<Poly[i].y then MaxY:=Poly[i].y
   else If MinY>Poly[i].y then MinY:=Poly[i].y;
  TaoDanhSachCanhCat;
  For y:=MinY to MaxY do
```

```
    begin
      TaoDanhSachGiaoDiem;
      SapXepVaLoc;
      ToMauCacDoan;
    end;
  Setlength(D,0);
  Setlength(G,0);
end;
```

BÀI TẬP

1. Xây dựng một chương trình cho phép quan sát vật thể 3D đặc lồi. Chương trình cho phép thay đổi vị trí quan sát, cho phép thể hiện tác động của các nguồn sáng xung quanh và định hướng lên đối tượng.

Nâng cao: Cho thép thay đổi cường độ của các nguồn sáng, cũng như thay đổi hướng chiếu của nguồn sáng định hướng.

2. Hãy xây dựng chương trình với các chức năng như **Bài 1** song sử dụng phương pháp tô bóng Gouraud.

3. Hãy tổng hợp các kiến thức đã biết để xây dựng một chương trình mô phỏng thế giới thực trong đó có nhiều đối tượng khác nhau vận động.

PHỤ LỤC

MỘT SỐ CHƯƠNG TRÌNH MINH HỌA

I. CÁC THUẬT TOÁN TÔ MÀU

1. Thuật toán tô màu theo loạt

```
Type Toado=record
          x,y:integer;
          end;
     mang=array[1..20] of Toado;

var  minx,miny,maxx,maxy,n,mau1,mau2:integer;
     a:mang;

Procedure NhapDuLieu(var a:Mang; var n:Byte);
var i:Byte;
Begin
  write('nhap vao so dinh : ');readln(n);
  for i:=1 to n do
    begin
      write('x',i,' =  ');readln(a[i].x);
      write('y',i,' =  ');readln(a[i].y);
    end;
  write('mau vien da giac: ');  readln(mau1);
  write('mau to da giac: ');    readln(mau2);
End;

Procedure vedagiac(P:mang;sodinh:byte);
var  i,j:byte;
Begin
  setcolor(mau1);
  for i:=1 to sodinh do
    begin
      if i=n then j:=1 else j:=i+1;
      line(P[i].x,P[i].y,P[j].x,p[j].y);
    end;
End;

Function  min(c,d:integer):integer;
  begin
    if c<d then min:=c else min:=d
  end;

Function  max(g,h:integer):integer;
  begin
    if g<h then max:=h else max:=g
  end;

Procedure Tomau(P:mang; n:Byte);
```

```
var   j,i,k,m,truoc,sau,tg:integer;
      r:real;
      z:array[1..15] of integer;
Begin
  for i:=minx+1 to maxx-1 do
    begin
      m:=0;
      for j:=1 to n do
        begin
          truoc:=j+1;
          if j=n then truoc:=1;
          sau:=j-1;
          if j=1 then sau:=n;
          if i=P[j].x then
          begin
            if (i>min(P[sau].x,P[truoc].x))and
               (i<max(P[sau].x,P[truoc].x)) then
              begin
                inc(m);
                z[m]:=P[j].y;
              end
            else
              begin
                inc(m);
                z[m]:=P[j].y;
                inc(m);
                z[m]:=P[j].y;
              end;
          end;
          if (i>min(P[j].x,P[truoc].x))and
             (i<max(P[truoc].x,P[j].x)) then
            begin
              inc(m);
              r:=(P[truoc].y-P[j].y)/(P[truoc].x-P[j].x);
              z[m]:=P[j].y+trunc(r*(i-P[j].x));
            end;
        end;

      for j:=1 to m-1 do
        for k:=j+1 to m do
          if z[j]>z[k] then
            begin
              tg:=z[j];z[j]:=z[k];z[k]:=tg;
            end;

      setcolor(mau2);
      For k:=1 to m-1 do
        if k mod 2<>0 then line(i,z[k],i,z[k+1]);
    end;
```

```
End;
Procedure ThietLapDoHoa;
var Gd,Gm:Integer;
Begin
  Gd:=0;
  InitGraph(Gd,Gm,'C:\BP\BGI');
End;
Begin
     CLRSCR;
     NhapDuLieu(a,n);
      minx:=a[1].x;
      maxx:=minx;
      miny:=a[1].y;
      maxy:=miny;
      for i:=1 to n do
        begin
          if minx>a[i].x then minx:=a[i].x;
          if miny>a[i].y then miny:=a[i].y;
          if maxx<a[i].x then maxx:=a[i].x;
          if maxy<a[i].x then maxy:=a[i].y;
        end;
     ThietLapDoHoa;
      vedagiac(a,n);
      Tomau(a,n);
      readln;
      closegraph;
end.
```

2. Thuật toán tô loang (Đệ qui)

```
uses crt, graph;
Type  ToaDo=record
           x,y:integer;
          End;
      Mang=array[0..30] of ToaDo;
Var a:Mang;
    x,y,n,Gd,Gm:Integer;
procedure NhapDaGiac(Var n:integer);
var i:integer;
begin
  clrscr;
  write('Nhap vao so dinh cua mot da giac n= ');
  readln(n);
  for i:=1 to n do
    begin
      writeln('Toa do dinh thu',i,'la:');
      write('a[',i,'].x=');
      readln(a[i].x);
```

```
      write('a[',i,'].y=');
      readln(a[i].y);
    end;
  Write('Nhap x= '); Readln(x);
  Write('Nhap y= '); Readln(y);
end;

Procedure VeDaGiac(n,color:integer);
  var i,j:byte;
  begin
    SetColor(Color);
     for i:=1 to n do
      begin
        if i=n then j:=1  else j:=i+1;
        line(a[i].x,a[i].y,a[j].x,a[j].y);
      end;
  end;

Function Max(a,b:integer):integer;
 Begin
   if a<b then  Max:=b  else   Max:=a;
 End;

Function Min(a,b:integer):integer;
 Begin
   if a<b then  Min:=a   else   Min:=b;
 End;

Function KiemTra(x,y:Integer;a:Mang):Boolean;
 var dem,i,j,s:Integer;
 Begin
   dem:=0;
   for i:=1 to n do     { Tim so giao diem }
    begin
     if i=n then j:=1 else j:=i+1;
     if i=1 then s:=n else s:=i-1;
     if x=a[i].x then
      begin
       if y<a[i].y then
        if (x<=Min(a[s].x ,a[j].x)) OR
                  (x>=Max(a[s].x,a[j].x)) then dem:=dem+2
        else dem:=dem+1;
      end
     else
       if (x>Min(a[i].x,a[j].x))and(x<Max(a[j].x,a[i].x))
then
        if y<=Min(a[i].y,a[j].y) then dem:=dem+1
        else if y <= (x-a[j].x)*(a[i].y-a[j].y)/(a[i].x-
                       a[j].x)+a[j].y then dem:=dem+1;
    end;
   if dem mod 2=1 then KiemTra:=True else KiemTra:=False;
```

```
 End;
Procedure ToLoang(x,y:Integer;color:Byte);
 Begin
   if KiemTra(x,y,a) and (GetPixel(x,y)<>color) then
    Begin
      PutPixel(x,y,color);
      ToLoang(x+1,y,color);
      ToLoang(x-1,y,color);
      ToLoang(x,y+1,color);
      ToLoang(x,y-1,color);
    End;
 End;
BEGIN
  Nhapdagiac(n);
  Gd:=Detect;
  InitGraph(Gd,Gm,'D:\TP\BGI');
  Vedagiac(n,4);
  Toloang(x,y,14);
  readln;
  closegraph;
END.
```

3. Thuật toán tô loang (Khử đệ qui)

```
Uses crt, graph;
Type  ToaDo=record
           x,y:integer;
          End;
     DANHSACH=^DS;
     DS=Record
         Data:ToaDo;
         Next:DANHSACH;
        End;
    Mang=array[0..30] of ToaDo;
Var Stack:DanhSach;
    a:Mang;
    x,y,n,Gd,Gm:Integer;
Procedure KhoiTaoStack;
 Begin
   Stack:=Nil;
 End;
Procedure PUSHStack(a:ToaDo;Var  Stack:DanhSach);
{ Nhap vao dau danh sach }
 Var p:DanhSach;
 Begin
     new(p);
     p^.Data:=a; p^.next:=nil;
     p^.next:=Stack;
```

```
     Stack:=p;
 End;
Procedure POPStack(Var Stack:DanhSach;var x,y:Integer);
{ Lay ra o dau danh sach }
 Var p:DanhSach;
 Begin
   If Stack<>nil then
    Begin
     p:=Stack;
     Stack:=Stack^.next;
     x:=p^.Data.x;
     y:=p^.Data.y;
     Dispose(p);
    End;
 End;
procedure NhapDaGiac(Var n:integer;var a:Mang);
var i:integer;
begin
  clrscr;
  write('Nhap vao so dinh cua mot da giac n= ');
  readln(n);
  for i:=1 to n do
    begin
      writeln('Toa do dinh thu',i,'la:');
      write('a[',i,'].x=');
      readln(a[i].x);
      write('a[',i,'].y=');
      readln(a[i].y);
    end;
  Write('Nhap x= '); Readln(x);
  Write('Nhap y= '); Readln(y);
end;
Procedure VeDaGiac(n,color:integer);
  var i,j:byte;
  begin
    SetColor(Color);
     for i:=1 to n do
      begin
        if i=n then j:=1  else j:=i+1;
        line(a[i].x,a[i].y,a[j].x,a[j].y);
      end;
  end;
Function Max(a,b:integer):integer;
 Begin
   if a<b then  Max:=b  else   Max:=a;
 End;
```

```
Function Min(a,b:integer):integer;
 Begin
   if a<b then  Min:=a   else   Min:=b;
 End;
Function KiemTra(x,y:Integer;a:Mang):Boolean;
 var dem,i,j,s:Integer;
 Begin
   dem:=0;
   for i:=1 to n do     { Tim so giao diem }
    begin
     if i=n then j:=1 else j:=i+1;
     if i=1 then s:=n else s:=i-1;
     if x=a[i].x then
      begin
       if y<a[i].y then
        if (x<=Min(a[s].x ,a[j].x))OR
           (x>=Max(a[s].x,a[j].x)) then dem:=dem+2
        else dem:=dem+1;
      end
     else
       if (x>Min(a[i].x,a[j].x)) and
          (x<Max(a[j].x,a[i].x)) then
        if y<=Min(a[i].y,a[j].y) then dem:=dem+1
        else if y <= (x-a[j].x)*(a[i].y-a[j].y)/
                (a[i].x-a[j].x)+a[j].y then dem:=dem+1;
    end;
   KiemTra:=dem mod 2=1;
 End;
Procedure ToLoang(x,y:Integer;color:Byte);
 Var B,C:ToaDo;
 Begin
   if KiemTra(x,y,a) and (GetPixel(x,y)<>color) then
    Begin
      PutPixel(x,y,color);
      B.x:=x+1; B.y:=y;
      PUSHStack(B,Stack);
      B.x:=x-1; B.y:=y;
      PUSHStack(B,Stack);
      B.x:=x; B.y:=y+1;
      PUSHStack(B,Stack);
      B.x:=x; B.y:=y-1;
      PUSHStack(B,Stack);
    End;
   While Stack<>nil do
    Begin
      POPStack(Stack,B.x,B.y);
      if KiemTra(B.x,B.y,a) and
```

```
        GetPixel(B.x,B.y)<>color) then
      Begin
        PutPixel(B.x,B.y,color);
        C.x:=B.x+1; C.y:=B.y;
        if KiemTra(C.x,C.y,a) and
                 (GetPixel(C.x,C.y)<>color) then
        PUSHStack(C,Stack);
        C.x:=B.x-1; C.y:=B.y;
        if KiemTra(C.x,C.y,a) and
                 (GetPixel(C.x,C.y)<>color) then
        PUSHStack(C,Stack);
        C.x:=B.x; C.y:=B.y+1;
        if KiemTra(C.x,C.y,a) and
                 (GetPixel(C.x,C.y)<>color) then
        PUSHStack(C,Stack);
        C.x:=B.x; C.y:=B.y-1;
        if KiemTra(C.x,C.y,a) and
                 (GetPixel(C.x,C.y)<>color) then
        PUSHStack(C,Stack);
      End;
    End;
 End;

BEGIN
  KhoiTaoStack;
  Nhapdagiac(n,a);
  Gd:=Detect;
  InitGraph(Gd,Gm,'D:\TP\BGI');
  Vedagiac(n,4);
  Toloang(x,y,14);
  readln;
  closegraph;
END.
```

II. CÁC THUẬT TOÁN XÉN HÌNH

1. Thuật toán Cohen Sutherland

```
Uses crt,graph;
Const LEFT=1;
      RIGHT=2;
      BELOW=4;
      ABOVE=8;

Type ToaDo2D=record
           x,y:integer;
          end;

var  Tren,Duoi,A,B:ToaDo2D;
     gd,gm:Integer;
     ch:char;
```

```
procedure NhapDinhHCN;
 begin
  Tren.x:=100;
  Tren.y:=100;
  Duoi.x:=450;
  Duoi.y:=350;
  randomize;
  a.x:=random(GetMaxx);
  a.y:=random(GetMaxY);
  b.x:=random(GetMaxx);
  b.y:=random(GetMaxY);
 end;

PROCEDURE VeHCN;
  begin
   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(xorput);
   line(a.x,a.y,b.x,b.y);
   ch:=readkey;
   line(a.x,a.y,b.x,b.y);
   setwritemode(orput);
 end;

FUNCTION MA(P:ToaDo2D):Byte;
 var s:Byte;
BEGIN
   s:=0;
   if P.x<Tren.x then  s:=s OR Left;
   if P.x>Duoi.x then  s:=s OR Right;
   if P.y<Tren.y then  s:=s OR Above;
   if P.y>Duoi.y then  s:=s OR Below;
   Ma:=s;
end;

Procedure Swap(Var A,B:ToaDo2D);
var t:ToaDo2D;
 Begin
   t:=a; a:=b; b:=t;
 End;

Procedure Clipping(A,B,Tren,Duoi:ToaDo2D);
Var stop,draw:Boolean;
    m:Real;
Begin
  stop:=False; draw:=False;
  While not stop do
   Begin
```

```
If (Ma(A)=0)and(Ma(B)=0) then
 Begin
   stop:=True; draw:=True;
 End
else
 If (Ma(A) and Ma(B)<>0) then stop:=True
 else
  Begin
  If (Ma(A)and Ma(B)=0)and
     (Ma(A)<>0)or(Ma(B)<>0)) then
     Begin
       if Ma(A)=0 then Swap(A,B); {A luon nam ngoai}
       if A.x=B.x then
        Begin
         if Ma(A) and ABOVE<>0 then A.y:=Tren.y
         else A.y:=Duoi.y;
         if Ma(B)<>0 then
          Begin
           if Ma(B) and ABOVE<>0 then B.y:=Tren.y;
           if Ma(B) and BELOW<>0 then B.y:=Duoi.y;
          End;
         stop:=True; draw:=True;
        End
       else {Ax<>Bx}
        Begin
          m:=(B.y-A.y)/(B.x-A.x);
          If Ma(A) and LEFT<>0 then
           Begin
             A.y:=round((Tren.x - A.x)*m + A.y);
             A.x:=Tren.x;
           End
          else
            If Ma(A) and RIGHT<>0 then
              Begin
                A.y:=round((Duoi.x - A.x)*m + A.y);
                A.x:=Duoi.x;
              End
            else
              If Ma(A) and ABOVE<>0 then
                Begin
                  A.x:=round((Tren.y - A.y)/m + A.x);
                  A.y:=Tren.y;
                End
              else
                If Ma(A) and BELOW<>0 then
                  Begin
                   A.x:=round((Duoi.y - A.y)/m +A.x);
                   A.y:=Duoi.y;
```

```
                        End;
                End;
            End;
         End;
    End;
  setcolor(14);
  If draw then Line(A.x,A.y,B.x,B.y);
  setcolor(15);
End;

BEGIN
 gd:=detect; Initgraph(gd,gm,'D:\TP\BGI');
 repeat
 NhapDinhHCN;
 VeHCN;
 Clipping(A,B,Tren,Duoi);
 until ch=#27;
 closegraph;
END.
```

2. Thuật toán chia nhị phân

```
Uses crt,graph;

type ToaDo2D=record
           x,y:integer;
          end;

var  Tren,Duoi,A,B:ToaDo2D;
     gd,gm:Integer;

procedure NhapDinhHCN;
 begin
  Tren.x:=100;
  Tren.y:=100;
  Duoi.x:=300;
  Duoi.y:=200;
  a.x:=352;
  a.y:=122;
  b.x:=22;
  b.y:=23;
 end;

PROCEDURE VeHCN;
  begin
   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(xorput);
   line(a.x,a.y,b.x,b.y);
   readln;
```

```
   line(a.x,a.y,b.x,b.y);
 end;

FUNCTION MA(P:ToaDo2D):Byte;
 var s:Byte;
BEGIN
   s:=0;
   if P.x<Tren.x then  s:=s OR Left;
   if P.x>Duoi.x then  s:=s OR Right;
   if P.y<Tren.y then  s:=s OR Above;
   if P.y>Duoi.y then  s:=s OR Below;
   Ma:=s;
end;

PROCEDURE XuLyATrongBNgoai(A,B:ToaDo2D);
 Var C,D,M:ToaDo2D;
   begin
    c:=a;d:=b;
    While abs(C.x-D.x)+abs(C.y-D.y)>2 do
     begin
      M.x:=round((C.x+D.x)/2);
      M.y:=round((C.y+D.y)/2);
      if ma(M)<>0 then D:=M  else C:=M;
     end;
    line(A.x,A.y,C.x,C.y);
 end;

PROCEDURE Clipping(A,B,Tren,Duoi:ToaDo2D);
Var C,D,M:ToaDo2D;
Begin
  if (ma(a)=0) and (ma(b)=0) then  line(a.x,a.y,b.x,b.y);
  if (ma(a)=0) and (ma(b)<>0) then  XulyATrongBNgoai(A,B);
  if (ma(a)<>0) and (ma(b)=0) then  XulyATrongBNgoai(B,A);
  if (ma(A)<>0) and (ma(B)<>0) and ((ma(A) and ma(B))=0)
then
   begin
     C:=A; D:=B;
     M.x:=(C.x+D.x)div 2;
     M.y:=(C.y+D.y)div 2;
     while (ma(M)<>0)and(abs(C.x-D.x)+abs(C.y-D.y)>2) do
       begin
         if (ma(C) and ma(M))<>0 then C:=M else D:=M;
         M.x:=(C.x+D.x)div 2;
         M.y:=(C.y+D.y)div 2;
       end;
    if ma(M)=0 then
     begin
      XulyATrongBNgoai(M,A);
      XulyATrongBNgoai(M,B);
     end;
```

```
   end;
End;

BEGIN
 NhapDinhHCN;
 gd:=detect; Initgraph(gd,gm,'D:\TP\BGI');
 VeHCN;
 Clipping(A,B,Tren,Duoi);
 readln;
 closegraph;
 END.
```

3. Thuật toán Liang-Barsky

```
Uses crt,graph;
var  PTop,PBottom,A,B:PointType;
     gd,gm:Integer;

procedure NhapDinhHCN;
var i:integer;
begin
  writeln('Nhap toa do dinh tren trai cua HCN:');
  write('x1=');readln(PTop.x);
  write('y1=');readln(PTop.y);
  writeln('Nhap toa do dinh duoi phai cua HCN:');
  write('x2=');readln(PBottom.x);
  write('y2=');readln(PBottom.y);

  writeln('Nhap toa do dinh thu nhat cua duong thang:');
  write('a.x=');readln(a.x);
  write('a.y=');readln(a.y);
  writeln('Nhap toa do dinh thu hai cua duong thang:');
  write('b.x='); readln(b.x);
  write('b.y='); readln(b.y);
end;

PROCEDURE VeHCN;
 begin
   line(PTop.x,PTop.y,PBottom.x,PTop.y);
   line(PBottom.x,PTop.y,PBottom.x,PBottom.y);
   line(PBottom.x,PBottom.y,PTop.x,PBottom.y);
   line(PTop.x,PBottom.y,PTop.x,PTop.y);
   setwritemode(xorput);
   line(a.x,a.y,b.x,b.y);
   readln;
   line(a.x,a.y,b.x,b.y);
 end;

Function Clip(p,q:real; Var u1,u2:real):Boolean;
 Var r:real;
 Begin
   Clip:=True;
```

```
   If p<0 then
    Begin
      r:=q/p;
      If r>u2 then Clip:=False else If r>u1 then u1:=r;
    End
   else If p>0 then
         Begin
          r:=q/p;
          If r<u1 then Clip:=False
           else If r<u2 then u2:=r;
         End
        else If q<0 then Clip:=False;
 End;

Procedure LiangBaskyClip(p1,p2,PTop,PBottom:PointType);
 Var u1,u2,dx,dy:real;
 Begin
   u1:=0; u2:=1;
   dx:=p2.x - p1.x;
   If Clip(-dx,p1.x - PTop.x,u1,u2) then
     If Clip(dx,PBottom.x - p1.x,u1,u2) then
       Begin
         dy:=P2.y - P1.y;
         If Clip(-dy,p1.y - PTop.y,u1,u2) then
           If Clip(dy,PBottom.y - p1.y,u1,u2) then
             Begin
               If u2<1 then
                Begin
                  p2.x:=p1.x + Round(u2*dx);
                  p2.y:=p1.y + Round(u2*dy);
                End;
               If u1>0 then
                Begin
                  p1.x:=p1.x + Round(u1*dx);
                  p1.y:=p1.y + Round(u1*dy);
                End;
               Line(p1.x,p1.y,p2.x,p2.y);
             End;
       End;
 End;

BEGIN
 clrscr;
 NhapDinhHCN;
 gd:=detect; Initgraph(gd,gm,'D:\TP\BGI');
 VeHCN;
 LiangBaskyClip(a,b,PTop,PBottom);
 readln;
 closegraph;
```

```
END.
```

III. VẼ CÁC ĐỐI TƯỢNG 3D

1. Vẽ mặt yên ngựa

```
USES crt, graph, DOHOA3d ; {Su dung Unit DoHoa3D}
VAR  u,uMin, uMax,du : real;
     v,vMin, vMax, dv : real;
     a1,a2,b1,b2,c1,c2,d  : integer;

PROCEDURE  Nhap_tham_so;
BEGIN
     projection := Phoicanh;
     rho := 50;     de := 2000;
     theta := 40;   phi := 20;
     uMin := -1;    uMax := 1 ;
     vMin := -1 ;   vMax:= 1 ;
     du := 0.095;   dv := 0.09;
     a1:= 0;  a2:=0;
     b1:= 0;  b2:=0;
     c1:= 0;  c2:=0;
     d := 1;
END;

FUNCTION fx(u,v:real): real;
BEGIN
  fx:=a1*cos(u) + b1*cos(v) + c1*cos(u)*cos(v) + d*u;
END;

FUNCTION fy(u,v:real): real;
BEGIN
  fy:=a1*cos(u) + b1*sin(v) +  c2*cos(u)*sin(v) + d*v ;
END ;

FUNCTION fz(u,v:real): real;
BEGIN
  fz := a2*sin(u) +b2*sin(v) + d*u*u - d*v*v   ;
END ;

PROCEDURE ho_duong_cong_u ;
VAR  P :ToaDo3D;
BEGIN
  u := uMin;   {Mat cat U ban dau}
  WHILE u<=uMax DO
    BEGIN
      v :=vMin;   {Mat cat V ban dau}
      P.x :=fx(u,v);
      P.y :=fy(u,v);
      P.z :=fz(u,v);
      DiDen(P); {Move to point (x,y,z) ban dau}
      WHILE v <= vMax DO  {Thay doi mat cat V}
```

```
      BEGIN
        P.x :=fx(u,v);
        P.y :=fy(u,v);
        P.z := fz(u,v);
        VeDen(P); {Ve den diem (x,y,z) moi}
        v := v+dv;    {tang gia tri mat cat V}
      END;
     u:=u+du;   {tang gia tri mat cat U}
   END;
END;

PROCEDURE ho_duong_cong_v  ;
VAR  P :ToaDo3D;
BEGIN
  v := vMin; {Mat cat V ban dau}
  WHILE v<=vMax DO
   BEGIN
     u :=vMin; {Mat cat U ban dau}
     P.x :=fx(u,v);
     P.y :=fy(u,v);
     P.z :=fz(u,v);
     DiDen(P);
     WHILE u <= uMax DO
       BEGIN
         P.x :=fx(u,v);
         P.y :=fy(u,v);
         P.z := fz(u,v);
         VeDen(P);
         u := u+du;  {tang gia tri mat cat U}
       END;
     v :=v+dv;  {tang gia tri mat cat V}
  END; {of while v}
END;

PROCEDURE DEMO;
BEGIN
  nhap_tham_so;
  REPEAT
    XoaManHinh;
    KhoiTaoPhepChieu;
    ho_duong_cong_u ;
    ho_duong_cong_v ;
    DieuKhienQuay;
    UNTIL  upcase(ch) = char(27);
END;

BEGIN { Main program }
   ThietLapDoHoa;
   demo;
   CloseGraph;
```

```
END.
```

2. Vẽ các đối tượng WireFrame

```
uses crt,Graph,DoHoa3D;
Const MaxDinh=50;
      MaxCanh=100;

Type  WireFrame=Record
          SoDinh:0..MaxDinh;
          Dinh:Array[1..MaxDinh] of ToaDo3D;
          SoCanh:0..MaxCanh;
          Canh:Array[1..MaxCanh,1..2] of 1..MaxDinh;
      End;

Var a:WireFrame;

Procedure KhoiTaoBien;
 Begin
  Rho:=5;  Theta:=20;
  Phi:=20;   De:=3;
 End;

Procedure DocFile(FileName:String; Var WF:WireFrame);
var f:Text;
    x,i:Integer;
 Begin
   assign(f,FileName);
   Reset(f);
   With WF do
    Begin
      read(f,x);  SoDinh:=x;
      read(f,x);  SoCanh:=x;
      For i:=1 to SoDinh do  {Doc so dinh}
       Begin
         read(f,x); Dinh[i].x:=x;
         read(f,x); Dinh[i].y:=x;
         read(f,x); Dinh[i].z:=x;
       End;
      For i:=1 to SoCanh do  {Doc so Canh}
       Begin
         read(f,x); Canh[i,1]:=x;
         read(f,x); Canh[i,2]:=x;
       End;
    End;
   Close(f);
 End;

Procedure VeWireFrame(WF:WireFrame);
 Var i:Byte;
     d1,d2:ToaDo3D;
 Begin
```

```
   With WF do
    Begin
      for i:=1 to SoCanh do
       Begin
         d1:=Dinh[Canh[i,1]];
         d2:=Dinh[Canh[i,2]];
         DiDen(d1);
         VeDen(d2);
       End;
    End;
 End;

Begin
  DocFile('bacdien.txt',a);
  Projection:=SongSong{PhoiCanh};
  ThietLapDoHoa;
  KhoiTaoBien;
  repeat
  KhoiTaoPhepChieu;
  VeWireFrame(a);
  DieuKhienQuay;
  until ch=#27;
  CloseGraph;
End.
```

3. Khử mặt khuất theo giải thuật BackFace

```
Uses  crt,graph,DoHoa3D;
Const MaxSoDinh=50;
      MaxSoMat =30;
      MaxDinh  =10;
Type TapDinh=Array[1..MaxSoDinh] of ToaDo3D;
     TapMat=Array[1..MaxSoMat,0..MaxDinh] of Integer;
     FaceModel=Record
                 SoDinh:Integer;
                 Dinh:TapDinh;
                 SoMat:Integer;
                 Mat:TapMat;
               End;
Var  Hinh:FaceModel;
     O:ToaDo3D;

Procedure KhoiTao;
 Begin
  Projection:=Phoicanh;
  Rho:=1500; Theta:=20;
  Phi:=15; DE:=3000;
 End;

Procedure VectorNhin(Dinh1,Dinh2,Dinh3:Integer;
                     Var v:toaDo3D);
```

```
 Begin
  With hinh do
   Begin
    v.x:=O.x - Dinh[Dinh1].x;
    v.y:=O.y - Dinh[Dinh1].y;
    v.z:=O.z - Dinh[Dinh1].z;
   end;
 End;
Procedure VectorChuan(Dinh1,Dinh2,Dinh3:Integer; Var
N:ToaDo3D);
 Var P,Q:ToaDo3D;
 Begin
  With hinh do
   Begin
    P.x:=Dinh[Dinh2].x - Dinh[Dinh1].x;
    P.y:=Dinh[Dinh2].y - Dinh[Dinh1].y;
    P.z:=Dinh[Dinh2].z - Dinh[Dinh1].z;
    Q.x:=Dinh[Dinh3].x - Dinh[Dinh1].x;
    Q.y:=Dinh[Dinh3].y - Dinh[Dinh1].y;
    Q.z:=Dinh[Dinh3].z - Dinh[Dinh1].z;
    N.x:=P.y*Q.z - Q.y*P.z;
    N.y:=P.z*Q.x - Q.z*P.x;
    N.z:=P.x*Q.y - Q.x*P.y;
   End;
 End;
Function TichVoHuong(v,n:ToaDo3D):Real;
 Begin
  TichVoHuong:=v.x*N.x + v.y*N.y + v.z*N.z;
 End;
Procedure ToaDoQuanSat;
 Begin
   KhoiTaoPhepChieu;
   O.x:= Rho*Aux7;
   O.y:= Rho*Aux8;
   O.z:= Rho*Aux2;
 End;
Procedure DocFile(FileName:String; Var WF:FaceModel);
var f:Text;
    x,i,j:Integer;
 Begin
   assign(f,FileName);
   Reset(f);
   With WF do
    Begin
      read(f,x);  SoDinh:=x;
      read(f,x);  SoMat:=x;
      For i:=1 to SoDinh do  {Doc so dinh}
```

```
       Begin
         read(f,x); Dinh[i].x:=x;
         read(f,x); Dinh[i].y:=x;
         read(f,x); Dinh[i].z:=x;
       End;
      For i:=1 to SoMat do  {Doc so Mat}
       Begin
         read(f,x); read(f,x); Mat[i,0]:=x;
         For j:=1 to Mat[i,0] do
          Begin
           read(f,x); Mat[i,j]:=x;
          End;
       End;
    End;
   Close(f);
 End;
Procedure VeMat(f:Integer);
 Var SoCanh,i,j:Integer;
     P,P0:ToaDo3D;
 Begin
  With hinh do
   Begin
    SoCanh:=Mat[f,0];
    For i:=1 to SoCanh do
     Begin
      j:=Mat[f,i];
      P.x:=Dinh[j].x; P.y:=Dinh[j].y; P.z:=Dinh[j].z;
      If i=1 Then
        Begin
          DiDen(P);
          P0.x:=P.x; P0.y:=P.y; P0.z:=P.z;
        End
      Else VeDen(P);
     End;
   VeDen(P0);
  End;
 End;
Procedure VeVatThe(Hinh:FaceModel);
 Var f,Dinh1,Dinh2,Dinh3:Integer;
     v,n:ToaDo3D;
 Begin
  With hinh do
   Begin
    For f:=1 to SoMat do
     Begin
      Dinh1:=Mat[f,1]; Dinh2:=Mat[f,2]; Dinh3:=Mat[f,3];
      VectorNhin(Dinh1,Dinh2,Dinh3,v);
```

```
      VectorChuan(Dinh1,Dinh2,Dinh3,N);
      If TichVoHuong(v,n)>0 Then
       Begin
         SetLineStyle(SolidLN,0,NormWidth);
         VeMat(f);
       End
      Else  Begin
              SetLineStyle(DottedLN,0,NormWidth);
              VeMat(f);
            End;
     End;
   End;
 End;

PROCEDURE DieuKhien;
 BEGIN
   ToaDoQuanSat;
   VeVatThe(Hinh);
   Repeat
     DieuKhienQuay;
     ToaDoQuanSat;
     VeVatThe(Hinh);
  Until ch=#27;
 END;

BEGIN   { Chuong Trinh Chinh }
  DocFile('Batdien.txt',Hinh);
  ThietLapDoHoa;
  KhoiTao;
  DieuKhien;
  CloseGraph;
END.
```