

**Trường Đại học Khoa học Huế**  
**Khoa CNTT**  
C&I&C

**Giáo trình**

**Lý Thuyết Đồ Họa**



Huế, tháng 10 năm 2008

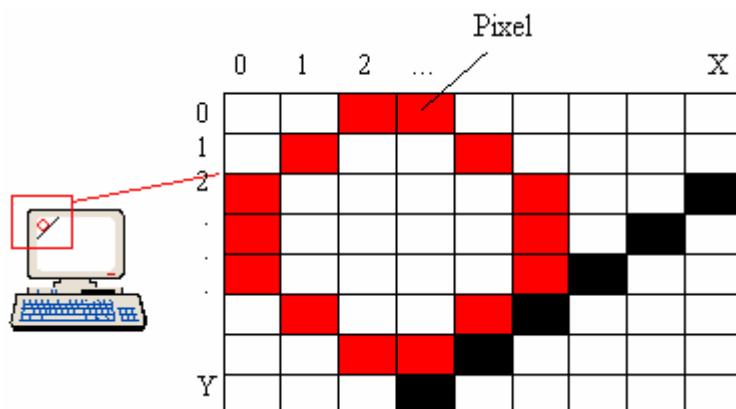
# Chương I: Các yếu tố cơ sở của đồ họa



## I. Các khái niệm cơ bản

### I.1. Thiết bị đồ họa và điểm ảnh (Pixel)

Các thiết bị đồ họa thông dụng như màn hình máy tính, máy in,... cho phép chúng ta biểu diễn các hình vẽ trên đó. Các thiết bị đồ họa này tạo ra mặt phẳng, đó là một tập hữu hạn các điểm mà mỗi điểm được đánh một cặp chỉ số nguyên gọi là tọa độ, thông thường mặt phẳng đồ họa do thiết bị tạo ra là một ma trận điểm, mỗi điểm gọi là một Pixel có các thành phần tọa độ là x và y.



(Hình I.1)

### I.2. Điểm và Đoạn thẳng trong mặt phẳng

Về mặt toán học thì một đoạn thẳng bao gồm một tập vô hạn các điểm trong mặt phẳng với cặp tọa độ thực. Song do đặc điểm của các thiết bị hiển thị nên khi biểu diễn trên thiết bị hiển thị của máy tính (như màn hình, máy in,...) thì được nguyên hoà thành một tập hữu hạn các cặp tọa độ nguyên (Hình I.1).

## II. Các thuật toán vẽ đoạn thẳng

Phương trình tổng quát của một đường thẳng được viết dưới dạng:

$$y = a * x + b$$

với

a: là hệ số góc hay còn gọi là độ dốc, nó phản ánh mối tương quan giữa 2 biến số x và y.

b: là khoảng chẵn trên trục hoành

Phương trình đường thẳng đi qua 2 điểm A(x<sub>a</sub>, y<sub>a</sub>) và B(x<sub>b</sub>, y<sub>b</sub>) là:

$$\frac{y - y_a}{y_b - y_a} = \frac{x - x_a}{x_b - x_a} \quad (\text{II.a})$$

(với x<sub>a</sub> ≠ x<sub>b</sub> và y<sub>a</sub> ≠ y<sub>b</sub>.

Khi x<sub>a</sub> = x<sub>b</sub> thì phương trình là x = x<sub>a</sub>

còn khi y<sub>a</sub> = y<sub>b</sub> phương trình là y = y<sub>a</sub>)

Đặt  $\frac{\Delta x}{\Delta y} = \frac{x_b - x_a}{y_b - y_a}$  thì (II.a) trở thành  $y = \frac{\Delta y}{\Delta x}x - \frac{\Delta y}{\Delta x}x_a + y_a$

$$\Leftrightarrow y = ax + b \text{ với } \begin{cases} a = \frac{\Delta y}{\Delta x} \\ b = -ax_a + y_a \end{cases} \quad (\text{II.b})$$

### II.1. Vẽ đoạn thẳng dựa vào phương trình

Khi biết được phương trình của một đường chúng ta hoàn toàn có thể vẽ được đường biểu diễn cho đường đó nhờ vào các tính toán trên phương trình. Ở đây đường mà ta cần biểu diễn là một đoạn thẳng AB với A(x<sub>a</sub>, y<sub>a</sub>) và B(x<sub>b</sub>, y<sub>b</sub>). Phương trình biểu diễn được cho bởi (II.b) với

$$x \in [x_a, x_b], y \in [y_a, y_b]$$

**Quy trình có thể tóm tắt như sau:**

- ❖ Nếu  $|\Delta y| \leq |\Delta x|$ : nghĩa là biến số x biến thiên nhanh hơn biến số y, lúc này để đảm bảo tính liên tục của các điểm vẽ ta cho biến số x thay đổi tuần tự và tính biến số y qua phương trình. Cụ thể như sau:

Cho x nhận các giá trị nguyên lần lượt từ  $x_a$  đến  $x_b$ , với mỗi giá trị x ta thực hiện:

- Tính  $y=ax+b$  thông qua phương trình
- Vẽ điểm  $(x, \text{Round}(y))$ .

Ở đây điểm trên đoạn thẳng có tọa độ là  $(x,y)$  song ta không thể vẽ điểm đó vì giá trị y là một giá trị thực, mà như đã nói ở mục I là các hệ thống biểu diễn đồ họa chỉ có hữu hạn điểm và mỗi điểm có tọa độ nguyên. Vì thế ta buộc phải minh họa cho điểm  $(x,y)$  trên đường thẳng thực bởi một điểm trên hệ thống thiết bị đồ họa gần với nó nhất đó là điểm có tọa độ  $(x, \text{round}(y))$ .

- ❖ Ngược lại: nghĩa là biến số y biến thiên nhanh hơn biến số x, lúc này để đảm bảo tính liên tục của các điểm vẽ ta cho biến số y thay đổi tuần tự và tính biến số x qua phương trình. Cụ thể như sau:

Cho y nhận các giá trị nguyên lần lượt từ  $y_a$  đến  $y_b$ , với mỗi giá trị y ta thực hiện:

- Tính  $x = \frac{y-b}{a}$  (hay  $x = \frac{\Delta x}{\Delta y}y - \frac{\Delta x}{\Delta y}y_a + x_a$ )
- Vẽ điểm  $(\text{Round}(x), y)$ .

- ❖ **Ví dụ:** Cho A(5,4) đến B(10,7) để vẽ đoạn thẳng AB ta thực hiện các bước sau:

$$\begin{aligned} \text{Tính } \Delta x &= x_b - x_a = 10 - 5 = 5 \\ \Delta y &= y_b - y_a = 7 - 4 = 3 \end{aligned}$$

$$\begin{cases} a = \frac{\Delta y}{\Delta x} = \frac{3}{5} \\ b = -ax_a + y_a = 1 \end{cases}$$

Vì  $|\Delta y| \leq |\Delta x|$  nên ta thực hiện theo cách 1 là cho x nhận các giá trị nguyên lần lượt từ  $x_a$  đến  $x_b$ , với mỗi giá trị x ta thực hiện:

- Tính  $y=ax+b$  thông qua phương trình
- Vẽ điểm  $(x, \text{Round}(y))$ .

Cụ thể như sau:

$$\text{Khi } x = x_a = 5: \quad \Rightarrow y = ax+b = 4; \quad \text{Vẽ điểm } (5,4)$$

$$\text{Khi } x = 6: \quad \Rightarrow y = 23/5 = 4.6; \quad \text{Vẽ điểm } (6,5)$$

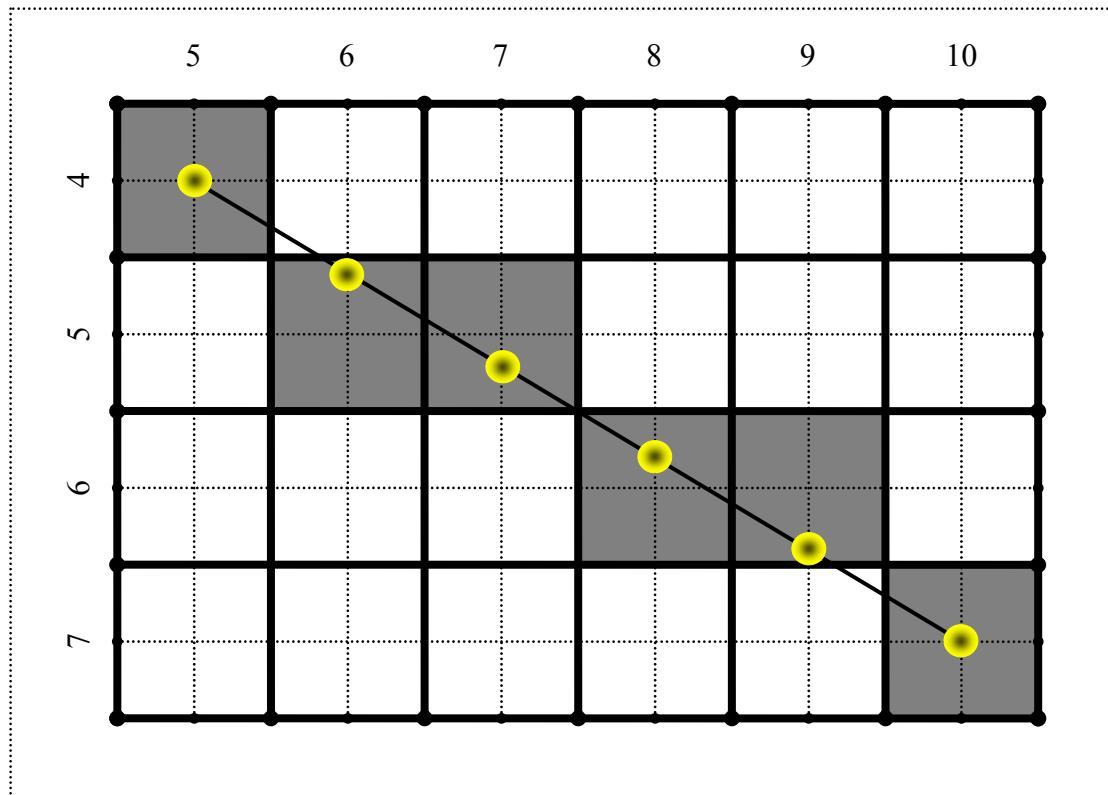
$$\text{Khi } x = 7: \quad \Rightarrow y = 26/5 = 5.2; \quad \text{Vẽ điểm } (7,5)$$

Khi  $x = 8$ :  $\Rightarrow y = 29/5 = 5.8$ ; Vẽ điểm  $(8, 6)$

Khi  $x = 9$ :  $\Rightarrow y = 32/5 = 6.4$ ; Vẽ điểm  $(9, 6)$

Khi  $x = 10$ :  $\Rightarrow y = 7$ ; Vẽ điểm  $(10, 7)$

Kết quả ta có hình vẽ đoạn thẳng AB có thể minh họa như sau:



Hình ảnh minh họa một đoạn thẳng từ  $A(5,4)$  đến  $B(10,7)$

## II.2. Vẽ đoạn thẳng dựa vào thuật toán Bresenham

Từ quy trình vẽ đoạn thẳng trên (II.1) ta thấy đoạn thẳng AB có thể được vẽ một cách dễ dàng. Song số phép tính cần phải thực hiện trong mỗi bước vẽ còn lớn, cụ thể là ta còn phải tính 1 phép nhân và 1 phép cộng trên trường số thực và một phép tính làm tròn (Round). Cũng với tư tưởng như trên song thuật toán Bresenham hướng tới một sự phân tích bài toán sâu sắc hơn để tìm ra một quy trình vẽ được các điểm song ít tính toán hơn.

Trong phần này ta chỉ trình bày giải thuật trong trường hợp hệ số góc của đoạn thẳng  $a \in [0,1]$ . Các trường hợp còn lại của hệ số góc như  $a \in [1,+\infty]$ ;  $a \in [-\infty, -1]$ ;  $a \in [-1, 0]$  chúng ta có thể lấy đối xứng đoạn thẳng qua các đường phân giác, OX, hay OY để quy về trường hợp  $a \in [0,1]$ .

Rõ ràng là vì  $a \in [0,1]$  nên quy trình ở đây là cho x nhận các giá trị nguyên lần lượt từ  $x_a$  đến  $x_b$ , với mỗi giá trị x ta cần phải tìm ra một giá trị y nguyên để  $(x,y)$  chính là toạ độ của điểm cần minh họa trên thiết bị, song giá trị y tìm ra ở đây phải thông qua ít phép tính toán hơn quy trình ở II.1.

Giả thiết với hai điểm đầu mút  $A(x_a, y_a)$  và  $B(x_b, y_b)$  có toạ độ nguyên và  $x_a < x_b$ . Rõ ràng điểm đầu tiên cần biểu diễn trên thiết bị đó là điểm có toạ độ  $(x_a, y_a)$ . Nếu gọi điểm chọn được đầu tiên là  $(x_0, y_0)$  thì

❖  $(x_0, y_0) = (x_a, y_a)$

theo lập luận quy nạp ta:

- ❖ Giả thiết rằng đến bước thứ i ta đã chọn được điểm thứ i, hay nói cách khác là điểm chọn thứ i là  $(x_i, y_i)$  đã được xác định giá trị.
- ❖ Vậy đến bước tiếp theo  $(i+1)$  ta sẽ chọn điểm nào? Hay nói cách khác là điểm chọn thứ  $(i+1)$  là  $(x_{i+1}, y_{i+1})$  sẽ có toạ độ bằng bao nhiêu.

(Chú ý:  $x_i, y_i$  là tên gọi của toạ độ điểm chọn thứ i, ví dụ như  $(x_0, y_0)$  là tên gọi của điểm chọn đầu tiên ( $i=0$ ) và nó có giá trị là  $(x_a, y_a)$ )

Để trả lời câu hỏi này ta cần dựa vào một số lập luận sau:

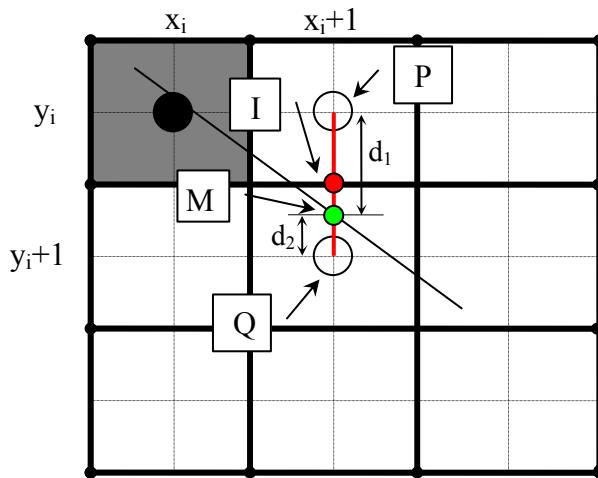
Như trên đã trình bày thì điểm chọn thứ  $i+1$  sẽ phải có hoành độ x bằng hoành độ của điểm trước đó cộng thêm 1:

Hay  $x_{i+1} = x_i + 1$

Gọi M là điểm thuộc AB sao cho  $x_M = x_{i+1} = x_i + 1$

thì  $y_M = ax_M + b = a(x_i + 1) + b = (ax_i + b) + a$

vậy điểm tiếp theo thuộc đoạn thẳng mà ta cần tìm minh họa trên thiết bị là  $M(x_i + 1, (ax_i + b) + a)$ . Câu hỏi đặt ra là ta sẽ chọn điểm nào trong 2 điểm  $P(x_i + 1, y_i)$  và  $Q(x_i + 1, y_i + 1)$  để minh họa cho M trên thiết bị đồ họa.



Để trả lời câu hỏi này ta đi xét một biểu thức trung gian:

$$\text{Đặt } d_1 = y_M - y_P; \quad \text{và} \quad d_2 = y_Q - y_M$$

Xét biểu thức:

$$d_1 - d_2 = (y_M - y_P) - (y_Q - y_M) = 2y_M - (y_P + y_Q) = 2\left[y_M - \frac{y_P + y_Q}{2}\right]$$

Nếu gọi I là trung điểm củaQP thì:  $d_1 - d_2 = 2[y_M - y_I]$

Rõ ràng là:

- ❖ Nếu  $d_1 - d_2 < 0$  thì điểm  $y_M < y_I$ , suy ra P gần điểm M hơn Q, vậy ta sẽ chọn điểm P là điểm minh họa cho M trên thiết bị đồ họa
- ❖ Nếu  $d_1 - d_2 > 0$  thì điểm  $y_M > y_I$ , suy ra Q gần điểm M hơn P, vậy ta sẽ chọn điểm Q là điểm minh họa cho M trên thiết bị đồ họa
- ❖ Nếu  $d_1 - d_2 = 0$  thì điểm  $y_M = y_I$ , suy ra khả năng lựa chọn P và Q là như nhau, song ta phải quyết định chọn một điểm. Trong tình huống này ta quyết định chọn điểm Q.

Vậy để tìm được điểm minh họa tiếp theo ta cần xét dấu của biểu thức  $d_1 - d_2$ . Song ta thấy biểu thức  $d_1 - d_2$  còn khá phức tạp và phải thực hiện tính toán trên trường số thực do trong đó có xuất hiện phép chia:

$$y_M = (ax_i + b) + a = (ax_i + (-ax_a + y_a)) + a = \frac{\Delta y}{\Delta x}x_i - \frac{\Delta y}{\Delta x}x_a + y_a + \frac{\Delta y}{\Delta x} \quad (*)$$

Để tránh tính biểu thức  $d_1 - d_2$  trên trường số thực người ta hướng tới một biểu thức tương đương về dấu đó là

$$P_i = \Delta x(d_1 - d_2)$$

Việc đưa  $\Delta x$  vào nhầm loại bỏ mảng số trong biểu thức ( $d_1-d_2$ ) để thu được biểu thức  $P_i$  tính trên trường số nguyên. Thật vậy:

$$\begin{aligned} P_i &= \Delta x(d_1 - d_2) = \Delta x(2y_M - (y_P + y_Q)) = \Delta x(2y_M - (y_i + y_{i+1})) = \Delta x(2y_M - 2y_i - 1) \\ P_i &= 2\Delta xy_M - 2\Delta xy_{i+1} - \Delta x \end{aligned}$$

Thay  $y_M$  bởi giá trị ở (\*) ta được:

$$\begin{aligned} P_i &= 2\Delta yx_i - 2\Delta yx_{i+1} + 2\Delta xy_a + 2\Delta y - 2\Delta xy_i - \Delta x \\ &= 2\Delta yx_i - 2\Delta xy_i - 2\Delta yx_{i+1} + 2\Delta xy_a + 2\Delta y - \Delta x \end{aligned} \quad (\text{a})$$

Ta thấy biểu thức  $P_i$  được xác lập từ tọa độ của điểm chọn thứ  $i$  là  $(x_i, y_i)$ . Vậy  $P_{i+1}$  sẽ được xác lập từ điểm chọn thứ  $i+1$  là  $(x_{i+1}, y_{i+1})$  như sau:

$$P_{i+1} = 2\Delta yx_{i+1} - 2\Delta xy_{i+1} - 2\Delta yx_a + 2\Delta xy_a + 2\Delta y - \Delta x \quad (\text{b})$$

Vì dấu của  $P_i$  và dấu của  $(d_1-d_2)$  là tương đương nên có thể tóm tắt quy tắc chọn điểm tiếp theo như sau:

❖ Nếu  $P_i < 0$ : Thì chọn điểm  $P$  làm điểm minh họa cho  $M$  trên thiết bị đồ họa

Hay nói cách khác là điểm chọn thứ  $i+1$  là  $(x_{i+1}, y_{i+1})$  sẽ có giá trị bằng  $P$

Nghĩa là:  $(x_{i+1}, y_{i+1}) = (x_i + 1, y_i)$

Thay vào (b) ta có:

$$\begin{aligned} P_{i+1} &= 2\Delta y(x_i + 1) - 2\Delta xy_i - 2\Delta yx_a + 2\Delta xy_a + 2\Delta y - \Delta x \\ &= P_i + 2\Delta y \end{aligned}$$

❖ Nếu  $P_i \geq 0$ : Thì chọn điểm  $Q$  là điểm minh họa cho  $M$  trên thiết bị đồ họa

Hay nói cách khác là điểm chọn thứ  $i+1$  là  $(x_{i+1}, y_{i+1})$  sẽ có giá trị bằng  $Q$

Nghĩa là:  $(x_{i+1}, y_{i+1}) = (x_i, y_i + 1)$

Thay vào (b) ta có:

$$\begin{aligned} P_{i+1} &= 2\Delta y(x_i + 1) - 2\Delta x(y_i + 1) - 2\Delta yx_a + 2\Delta xy_a + 2\Delta y - \Delta x \\ &= P_i + 2\Delta y - 2\Delta x \end{aligned}$$

Khi  $i=0$  thì ta có  $(x_0, y_0) = (x_a, y_a)$  thay vào (a) ta có:

$$\begin{aligned} P_0 &= 2\Delta yx_0 - 2\Delta xy_0 - 2\Delta yx_a + 2\Delta xy_a + 2\Delta y - \Delta x \\ &= 2\Delta y - \Delta x \end{aligned}$$

Vậy từ đây ta thấy được quy trình chọn ra các điểm trên thiết bị để minh họa cho đoạn thẳng AB theo thuật toán Bresenham như sau:

- ❖ Điểm chọn đầu tiên ( $i=0$ ) là  $(x_0, y_0) = (x_a, y_a)$  và giá trị  $P_0 = 2\Delta y - \Delta x$
- ❖ Dựa vào giá trị của  $P_0$  là âm hay dương mà ta lại chọn được điểm tiếp theo  $(x_1, y_1)$  và tính được giá trị  $P_1$
- ❖ Dựa vào giá trị của  $P_1$  là âm hay dương mà ta lại chọn được điểm tiếp theo  $(x_2, y_2)$  và tính được giá trị  $P_2$
- ❖ Cứ như vậy ta tìm ra được tập các điểm trên thiết bị đồ họa để minh họa cho đoạn thẳng AB.

### II.2.a. Tóm tắt thuật toán Bresenham:

- ❖ Bước 1:

Tính  $\Delta x; \Delta y$

$$\text{Const1} = 2\Delta y; \text{Const2} = 2\Delta y - 2\Delta x$$

$$P_0 = 2\Delta y - \Delta x; (x_0, y_0) = (x_a, y_a)$$

Vẽ điểm  $(x_0, y_0)$

- ❖ Bước 2: Với mỗi giá trị  $i$  ( $i=0, 1, 2, \dots$ ) ta xét dấu  $P_i$ 
  - Nếu  $P_i < 0$ : thì chọn điểm tiếp theo là

$$(x_{i+1}, y_{i+1}) = (x_i + 1, y_i)$$

$$P_{i+1} = P_i + \text{Const1}$$

- Ngược lại (tức  $P_i \geq 0$ ): thì chọn điểm tiếp theo là

$$(x_{i+1}, y_{i+1}) = (x_i + 1, y_i + 1)$$

$$P_{i+1} = P_i + \text{Const2}$$

Vẽ điểm  $(x_{i+1}, y_{i+1})$  vừa tìm được

- ❖ Bước 3: Lặp lại bước 2 với những giá trị  $i$  tiếp theo, cho đến khi điểm tìm được trùng với B, nghĩa là  $x_{i+1} = x_b$  thì thuật toán kết thúc.

### II.2.b. Ví dụ:

Cho đoạn thẳng AB với A(5,6) và B(10,10). Sử dụng thuật toán Bresenham chúng ta có thể tìm được các Pixel cần vẽ để biểu diễn đoạn AB trên màn hình như sau:

#### Bước 1:

$$\Delta x = 10 - 5 = 5; \quad \Delta y = 10 - 6 = 4;$$

$$\text{Const1} = 2\Delta y = 8; \quad \text{Const2} = 2\Delta y - 2\Delta x = 8 - 10 = -2;$$

$$P_0 = 2\Delta y - \Delta x = 8 - 5 = 3; \quad (x_0, y_0) = (x_a, y_a) = (5, 6)$$

Vẽ điểm  $(x_0, y_0)$

❖ **Bước 2:** Bước lặp:

i = 0:

Ta có  $P_0 = 3 \geq 0$  nên:

$$(x_1, y_1) = (x_0 + 1, y_0 + 1) = (6, 7)$$

$$P_1 = P_0 + \text{Const2} = 3 + (-2) = 1$$

Vẽ điểm  $(x_1, y_1) = (6, 7)$

i = 1:

Ta có  $P_1 = 1 \geq 0$  nên:

$$(x_2, y_2) = (x_1 + 1, y_1 + 1) = (7, 8)$$

$$P_2 = P_1 + \text{Const2} = 1 + (-2) = -1$$

Vẽ điểm  $(x_2, y_2) = (7, 8)$

i = 2:

Ta có  $P_2 = -1 < 0$  nên:

$$(x_3, y_3) = (x_2 + 1, y_2 + 1) = (8, 8)$$

$$P_3 = P_2 + \text{Const2} = -1 + 8 = 7$$

Vẽ điểm  $(x_3, y_3) = (8, 8)$

i = 3:

Ta có  $P_3 = 7 \geq 0$  nên:

$$(x_4, y_4) = (x_3 + 1, y_3 + 1) = (9, 9)$$

$$P_4 = P_3 + \text{Const2} = 7 + (-2) = 5$$

Vẽ điểm  $(x_4, y_4) = (9, 9)$

i = 4:

Ta có  $P_4 = 5 \geq 0$  nên:

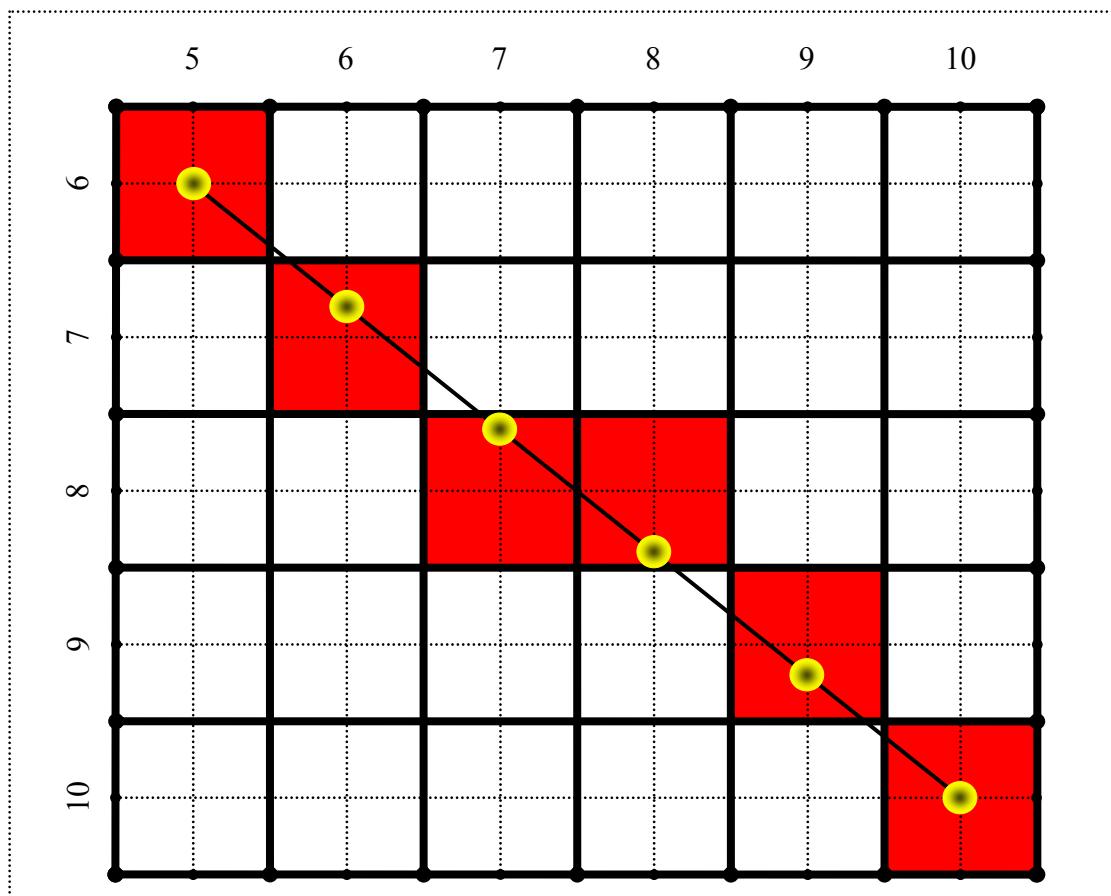
$$(x_5, y_5) = (x_4 + 1, y_4 + 1) = (10, 10)$$

$$P_5 = P_4 + \text{Const2} = 5 + (-2) = 3$$

Vẽ điểm  $(x_5, y_5) = (10, 10)$

Vì  $x_5 = x_b = 10$  nên kết thúc vòng lặp và cũng là kết thúc thuật toán.

Hình vẽ minh họa:



(Các Pixel vuông màu đỏ chính là hình ảnh thể hiện của đoạn thẳng AB trên màn hình máy tính)

### II.2.c. Hướng dẫn cho các trường hợp hệ số góc ngoài khoảng [0,1]

Giả sử cho A(0,50), B(100,10) để dựng đoạn thẳng AB ta cần tiến hành một số phân tích:

$$\Delta x = x_b - x_a = 100 - 0 = 100$$

$$\Delta y = y_b - y_a = 10 - 50 = -40$$

Suy ra hệ số góc  $a = \Delta y / \Delta x = -0.4 \in [-1, 0]$

Lúc này ta cần lấy đối xứng của AB qua trục OX để được CD với C(0, -50) D(100, -10) nên xét trên đoạn thẳng CD ta có

$$\Delta x = x_d - x_c = 100 - 0 = 100$$

$$\Delta y = y_d - y_c = -10 - (-50) = 40$$

Suy ra hệ số góc  $a = \Delta y / \Delta x = 0.4 \in [1, 0]$  thỏa mãn điều kiện của thuật toán Bresenham. Từ đó chúng ta có thể áp dụng thuật toán Bresenham để tính toán ra các điểm cần vẽ trên CD nhưng chúng ta sẽ không vẽ nó (vì mục đích chúng ta là

vẽ AB) mà lại lấy đối xứng qua trục OX (tức đối xứng ngược lại với lúc đầu) rồi mới vẽ, thì lúc này các điểm vẽ ra sẽ là hình ảnh của đoạn thẳng AB. Như thế CD chỉ đóng vai trò trung gian để áp dụng được thuật toán còn kết quả sau cùng ta thu được vẫn là hình ảnh minh họa cho đoạn AB.

Áp dụng tương tự:

+ trường hợp hệ số góc  $a \in (1, +\infty)$ , lúc này chúng ta cần lấy đối xứng qua đường phân giác của góc phần tư thứ nhất để hệ số góc được quy về  $[0, 1]$ .

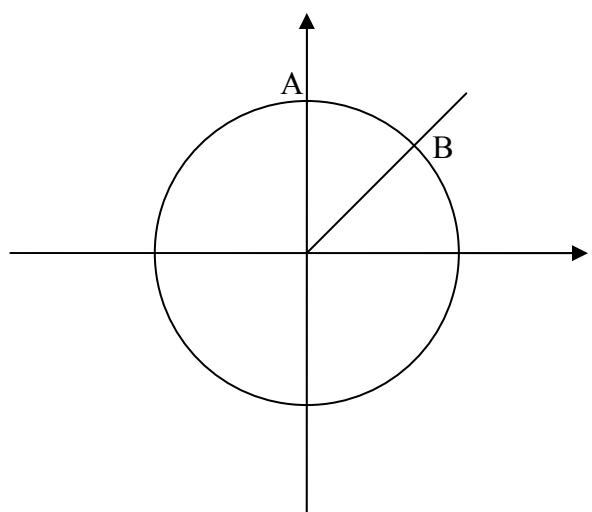
+ trường hợp hệ số góc  $a \in (-\infty, -1)$ , lúc này chúng ta cần lấy 2 lần đối xứng. Đầu tiên lấy đối xứng qua OX rồi tiếp đến đối xứng qua đường phân giác.

Xét điểm M(x,y). Đối xứng qua OX ta được tọa độ mới là (x,-y). Tiếp đến lấy đối xứng qua tia phân giác góc phần tư thứ nhất ta được (-y,x).

#### II.2.d. Cài đặt thuật toán

- ❖ Sinh viên cần xây dựng một thủ tục vẽ đoạn thẳng AB với giả thiết đầu vào thỏa hệ số góc thuộc đoạn  $[0, 1]$
- ❖ Sinh viên cần xây dựng một thủ tục vẽ đoạn thẳng tổng quát cho phép vẽ đoạn thẳng AB trong mọi trường hợp, và một chương trình minh họa có sử dụng thủ tục này.

### III. Các thuật toán vẽ đường tròn



Phương trình đường tròn tâm O (gốc toạ độ) bán kính R (nguyên) là:

$$X^2 + Y^2 = R^2$$

Trong mục này ta chỉ cần tìm phương pháp vẽ đường tròn tâm tại gốc toạ độ. Nếu ta vẽ được đường tròn tâm tại gốc toạ độ thì bằng cách thêm vào phép tịnh tiến ta được đường tròn tâm  $(x, y)$  bất kỳ.

Ở đây ta thấy để vẽ được đường tròn tâm gốc toạ độ ta chỉ cần tìm phương pháp vẽ cung một phần tam AB, và với các phép lấy đối xứng ta sẽ có các phần còn lại của đường tròn.

Với cung AB thì rõ ràng độ dốc của nó thuộc đoạn  $[-1,0]$ . Điều này ta có thể dễ dàng thấy qua góc của tiếp tuyến với cung AB hay qua đạo hàm phương trình biểu diễn cung AB.

Vì cung AB có độ dốc trong khoảng  $[-1,0]$ , nên ta suy ra rằng trên toàn bộ cung AB khi biến số x tăng thì biến số y giảm, và tốc độ thay đổi của y chậm hơn của x. Từ đây ta có thể đề ra một quy trình dựng cung AB là:

- ❖ Cho biến số x nhận lần lượt các giá trị nguyên từ  $x_a$  đến  $x_b$ . Với mỗi giá trị x ta thực hiện:
  - Tìm giá trị y nguyên tương ứng để điểm có tọa độ nguyên  $(x,y)$  sẽ là điểm gần nhất điểm  $(x,y_{circle})$  thuộc đường tròn
  - Vẽ điểm  $(x,y)$  tìm được và các đối xứng của nó để có được đường tròn

Trong mục này ta sẽ đi tìm hiểu 2 thuật toán cho phép dựng đường tròn (thực chất là dựng cung AB và các đối xứng của nó) một cách hiệu quả về mặt tốc độ.

### III.1. Thuật toán vẽ đường tròn MidPoint

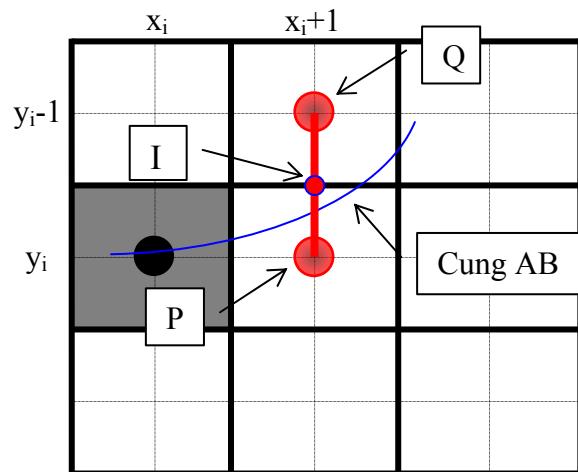
Thuật toán MidPoint hay còn gọi là thuật toán xét điểm giữa.

- ❖ Điểm đầu tiên được chọn để vẽ sẽ là điểm  $A(0,R)$ , nghĩa là:  $(x_0,y_0)=(0,R)$
- ❖ Giả sử đến bước thứ i ta đã chọn được điểm  $(x_i,y_i)$  để vẽ. Câu hỏi đặt ra là đến bước thứ  $i+1$  ta sẽ chọn điểm  $(x_{i+1},y_{i+1})$  có giá trị bằng bao nhiêu?

Vì điểm tiếp theo sẽ chọn theo quy tắc đã nói trên, nên có hoành độ x tăng một giá so với giá trị của điểm chọn trước, hay nói cách khác là:

$$x_{i+1} = x_i + 1$$

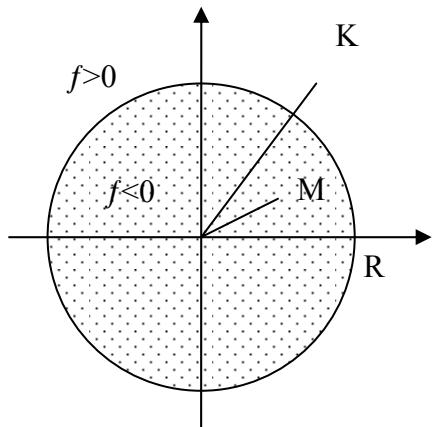
Đồng thời vì trên cung AB khi x tăng thì y giảm và tốc độ thay đổi của y chậm hơn của x, nên rõ ràng ta thấy là với giá trị x tăng 1 thì giá trị y sẽ giảm đi một lượng  $-1 \leq \Delta y \leq 0$ . Mà điểm chọn bước trước là  $(x_i,y_i)$  nên điểm chọn tiếp theo  $(x_{i+1},y_{i+1})$  chỉ có thể là một trong hai điểm  $P(x_i+1,y_i)$  và  $Q(x_i+1,y_i-1)$ .



Để quyết định được điểm chọn là P hay Q chúng ta hướng đến một biểu thức mà dấu của nó cho phép chúng ta ra quyết định chọn điểm nào.

Trước hết chúng ta xét hàm:

$$f_{circle}(x, y) = x^2 + y^2 - R^2$$



Với một điểm  $M(x, y)$  thì rõ ràng ta có:

- ❖  $f_{circle}(M) = f(x, y) = x^2 + y^2 - R^2 < 0$  khi và chỉ khi điểm M nằm trong đường tròn (tâm O bán kính  $R$ )
- ❖  $f_{circle}(M) = f(x, y) = x^2 + y^2 - R^2 > 0$  khi và chỉ khi điểm M nằm ngoài đường tròn (tâm O bán kính  $R$ )
- ❖  $f_{circle}(M) = f(x, y) = x^2 + y^2 - R^2 = 0$  khi và chỉ khi điểm M nằm trên đường tròn

Từ kết quả trên, nếu ta gọi I là trung điểm của PQ thì  $I(x_i+1, y_i-0.5)$  và:

$$\text{Đặt } P_i = f_{circle}(I) = f_{circle}(x_i+1, y_i-0.5) = (x_i+1)^2 + (y_i - 0.5)^2 - R^2 \quad (\text{a})$$

- ❖ Khi  $P_i = f_{circle}(I) < 0$  thì điểm I nằm trong đường tròn (tâm O bán kính  $R$ ), vì thế điểm P sẽ gần với đường tròn hơn điểm Q, do đó ta sẽ chọn điểm P làm điểm biểu diễn (vẽ).
- ❖ Khi  $P_i = f_{circle}(I) > 0$  thì điểm I nằm ngoài đường tròn, vì thế điểm Q sẽ gần với đường tròn hơn điểm P, do đó ta sẽ chọn điểm Q làm điểm biểu diễn.
- ❖ Khi  $P_i = f_{circle}(I) = 0$  thì điểm I nằm trên đường tròn, suy ra khả năng lựa chọn P và Q là như nhau, song ta phải quyết định chọn một điểm. Trong tình huống này thuật toán quy định chọn điểm Q.

Vậy từ đây ta thấy có thể dựa vào dấu của biểu thức  $P_i$  để ra quyết định chọn điểm tiếp theo.

Để thuật toán được đơn giản người ta tối ưu hóa việc tính  $P_i$  theo công thức truy hồi:

$$P_{i+1} = f(x_{i+1}+1, y_{i+1}-0.5) = (x_{i+1}+1)^2 + (y_{i+1} - 0.5)^2 - R^2 \quad (\text{b})$$

Dấu của  $P_i$  sẽ quyết định giá trị  $P_{i+1}$  cụ thể như sau:

- ❖ Nếu  $P_i < 0$ : thì điểm chọn tiếp theo là  $P(x_i+1, y_i)$ , nghĩa là  $(x_{i+1}, y_{i+1}) = (x_i+1, y_i)$ .

Thay vào (b) ta được:

$$P_{i+1} = (x_i+1+1)^2 + (y_i - 0.5)^2 - R^2 = P_i + 2(x_i+1)+1 = P_i + 2x_i + 3$$

- ❖ Nếu  $P_i \geq 0$ : thì điểm chọn tiếp theo là  $Q(x_i+1, y_i-1)$ , nghĩa là  $(x_{i+1}, y_{i+1}) = (x_i+1, y_i-1)$ .

Thay vào (b) ta được:

$$P_{i+1} = (x_i + 1 + 1)^2 + (y_i - 1 - 0.5)^2 - R^2 = P_i + 2(x_i + 1) + 1 - 2(y_i - 0.5) + 1 = P_i + 2(x_i - y_i) + 5$$

Đầu tiên ta chọn điểm A(0,R), nghĩa là  $(x_0, y_0) = (0, R)$ , Thay vào (a) ta có:

$$P_0 = 1 - R + \frac{1}{4} = \frac{5}{4} - R$$

Vậy quy trình vẽ được thực hiện như sau:

- ❖ Tính  $P_0$ , vẽ điểm  $(x_0, y_0) = (0, R)$
- ❖ Dựa vào dấu của  $P_0$  ta lại chọn được điểm vẽ tiếp theo  $(x_1, y_1)$  và giá trị  $P_1$
- ❖ Dựa vào dấu của  $P_1$  ta lại chọn được điểm vẽ tiếp theo  $(x_2, y_2)$  và giá trị  $P_2$
- ❖ Quá trình trên được lặp đi lặp lại cho đến khi ta vẽ được điểm nguyên gần nhất với B.
- ❖ Một điểm đáng chú ý ở đây các giá trị  $P$  tiếp theo có được bằng cách cộng với giá trị  $P$  trước đó với một lượng nguyên  $2x_i + 3$  hoặc  $2(x_i - y_i) + 5$  tùy theo dấu của  $P$ . Song nếu giá trị  $P$  khởi đầu là  $P_0 = 1 - R + \frac{1}{4}$  là một giá trị thực sẽ làm cho việc tính các giá trị  $P$  tiếp theo cũng phải xử lý trên trường số thực. Một điều dễ thấy là nếu ta thay đổi giá trị  $P_0$  khởi đầu là  $1 - R$  thì dấu của  $P_0$  và các  $P_i$  có được sau đó không hề thay đổi về dấu (mặc dù có bị giảm một lượng 0.25) do đó kết quả thuật toán không hề bị thay đổi, song các tính toán giá trị  $P$  chỉ phải tính trên trường số nguyên.

### III.1.a. Tóm tắt thuật toán vẽ đường tròn MidPoint :

- ❖ **Bước 1:**  $P_0 = 1 - R$ ;  $(x_0, y_0) = (0, R)$

Vẽ điểm  $(x_0, y_0)$

- ❖ **Bước 2:** Với mỗi giá trị  $i$  ( $i=0, 1, 2, \dots$ ) ta xét dấu  $P_i$ 
  - Nếu  $P_i < 0$ : thì chọn điểm tiếp theo là

$$(x_{i+1}, y_{i+1}) = (x_i + 1, y_i)$$

$$P_{i+1} = P_i + 2x_i + 3$$

- Ngược lại (tức  $P_i \geq 0$ ): thì chọn điểm tiếp theo là

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

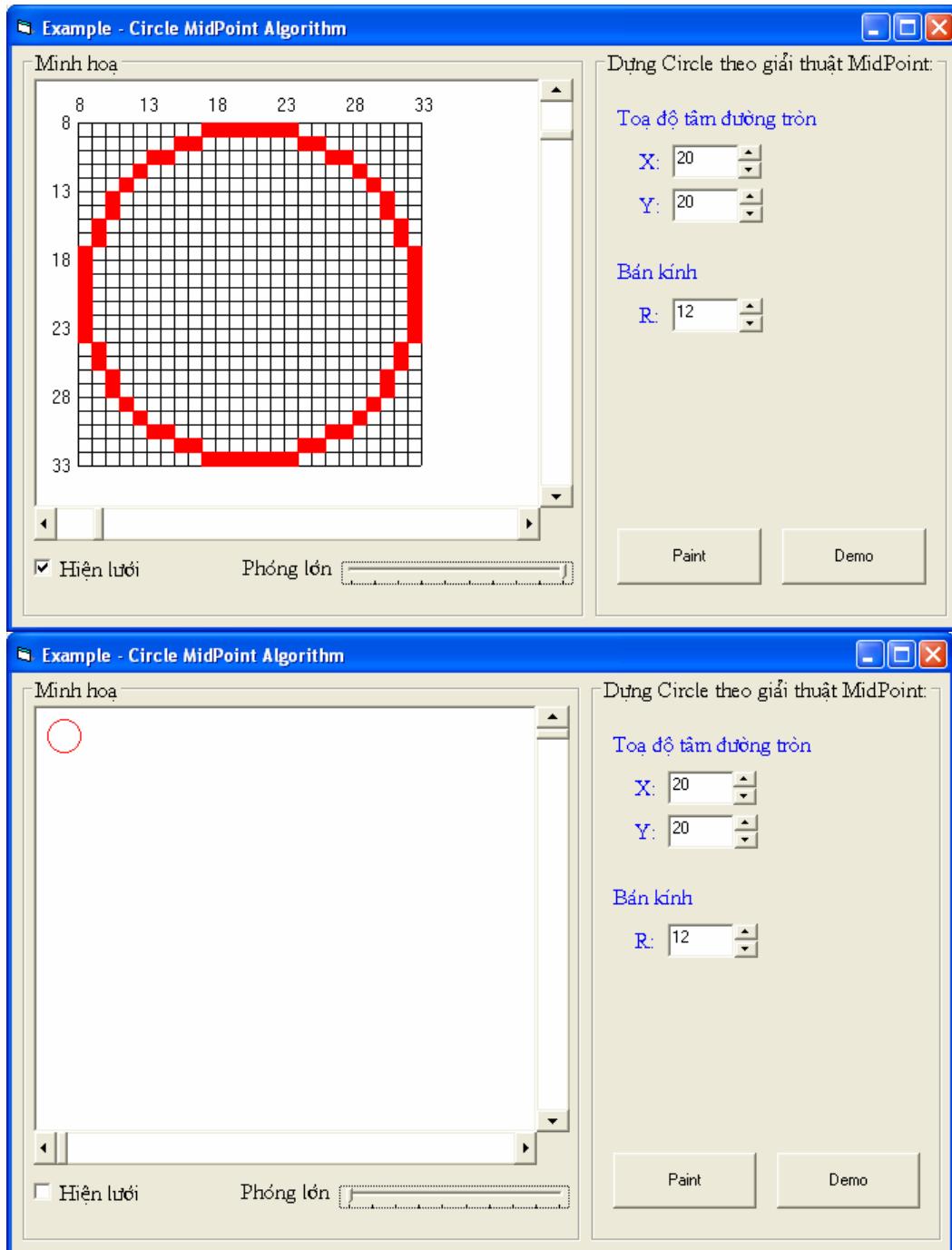
$$P_{i+1} = P_i + 2(x_i - y_i) + 5$$

Vẽ điểm  $(x_{i+1}, y_{i+1})$  vừa tìm được

- ❖ **Bước 3:** Lặp lại bước 2 với những giá trị i tiếp theo, cho đến khi ta vẽ được điểm nguyên gần nhất với B, nghĩa là  $x_{i+1} = \text{Trunc}(x_b) = \text{Trunc}\left(\frac{R}{\sqrt{2}}\right)$  thì thuật toán kết thúc.

### III.1.b. Cài đặt

- ❖ Sinh viên cần xây dựng một thủ tục vẽ đường tròn theo thuật toán đã trình bày trên.



(Hình vẽ minh họa)

### III.2. Thuật toán vẽ đường tròn Bresenham

Lập luận tương tự thuật toán trên song không dùng hàm  $f_{circle}$  mà dùng biểu thức ( $d_1 - d_2$ ). Có thể trình bày thuật giải như sau:

- ❖ Điểm đầu tiên được chọn để vẽ sẽ là điểm A(0,R), nghĩa là:  $(x_0, y_0) = (0, R)$
- ❖ Giả sử đến bước thứ i ta đã chọn được điểm  $(x_i, y_i)$  để vẽ. Câu hỏi đặt ra là đến bước thứ  $i+1$  ta sẽ chọn điểm  $(x_{i+1}, y_{i+1})$  có giá trị bằng bao nhiêu?

Vì điểm tiếp theo sẽ chọn theo quy tắc đã nói trên sẽ có hoành độ x tăng một giá so với giá trị của điểm chọn trước, hay nói cách khác là:

$$x_{i+1} = x_i + 1$$

Đồng thời vì trên cung AB khi x tăng thì y giảm và tốc độ thay đổi của y chậm hơn của x, nên rõ ràng ta thấy là với giá trị x tăng 1 thì giá trị y sẽ giảm đi một lượng  $-1 \leq \Delta y \leq 0$ . Mà điểm chọn bước trước là  $(x_i, y_i)$  nên điểm chọn tiếp theo  $(x_{i+1}, y_{i+1})$  chỉ có thể là một trong hai điểm P( $x_i + 1, y_i$ ) và Q( $x_i + 1, y_i - 1$ ).

Để quyết định được điểm chọn là P hay Q chúng ta hướng đến một biểu thức mà dấu của nó cho phép chúng ta ra quyết định chọn điểm nào.

$$\text{Đặt: } d_1 = y_P^2 - y^2 \quad \text{và} \quad d_2 = y^2 - y_Q^2$$

(giá trị y ở đây chính là tung độ của cung AB ứng với hoành độ x =  $x_i + 1$ )

$$\begin{aligned} \text{Đặt } P_i &= d_1 - d_2 = y_P^2 + y_Q^2 - 2y^2 = y_i^2 + (y_i - 1)^2 - 2(R^2 - x_{i+1}^2) = y_i^2 + (y_i - 1)^2 - 2(R^2 - (x_i + 1)^2) \\ &= y_i^2 + (y_i - 1)^2 - 2R^2 + 2(x_i + 1)^2 \end{aligned} \tag{a}$$

Dấu của biểu thức  $P_i$  cho phép xác định điểm chọn tiếp theo là P hay Q.

- ❖ Khi  $P_i < 0$ : thì điểm P sẽ gần với đường tròn hơn điểm Q, do đó ta sẽ chọn điểm P làm điểm biểu diễn (vẽ).
- ❖ Khi  $P_i > 0$ : thì điểm Q sẽ gần với đường tròn hơn điểm P, do đó ta sẽ chọn điểm Q làm điểm biểu diễn.
- ❖ Khi  $P_i = 0$ : khoảng cách từ P và Q đến đường tròn đều bằng nhau, nên ta có thể chọn P hay Q đều được. Trong tình huống này thuật toán quy ước chọn điểm Q làm điểm biểu diễn.

Vậy từ đây ta thấy có thể dựa vào dấu của biểu thức  $P_i$  để ra quyết định chọn điểm tiếp theo.

Để thuật toán được đơn giản người ta tối ưu hóa việc tính  $P_i$  theo công thức truy hồi:

$$P_{i+1} = y_{i+1}^2 + (y_{i+1}-1)^2 - 2R^2 + 2(x_{i+1}+1)^2 \quad (b)$$

Dấu của  $P_i$  sẽ quyết định giá trị  $P_{i+1}$  cụ thể như sau:

- ❖ Nếu  $P_i < 0$ : thì điểm chọn tiếp theo là  $P(x_i+1, y_i)$ , nghĩa là  $(x_{i+1}, y_{i+1}) = (x_i+1, y_i)$ .

Thay vào (b) ta được:

$$P_{i+1} = y_i^2 + (y_i-1)^2 - 2R^2 + 2((x_i+1)+1)^2 = P_i + 2(2(x_i+1)+1) = P_i + 4x_i + 6$$

- ❖ Nếu  $P_i \geq 0$ : thì điểm chọn tiếp theo là  $Q(x_i+1, y_i-1)$ , nghĩa là  $(x_{i+1}, y_{i+1}) = (x_i+1, y_i-1)$ .

Thay vào (b) ta được:

$$\begin{aligned} P_{i+1} &= (y_i-1)^2 + (y_i-2)^2 - 2R^2 + 2((x_i+1)+1)^2 = P_i + (-4y_i+4) + 2(2(x_i+1)+1) \\ &= P_i + 4(x_i - y_i) + 10 \end{aligned}$$

Đầu tiên ta chọn điểm  $A(0,R)$ , nghĩa là  $(x_0, y_0) = (0, R)$ , Thay vào (a) ta có:

$$\begin{aligned} P_0 &= y_0^2 + (y_0-1)^2 - 2R^2 + 2(x_0+1)^2 = R^2 + (R-1)^2 - 2R^2 + 2 \\ &= R^2 + R^2 - 2R + 1 - 2R^2 + 2 = 3 - 2R \end{aligned}$$

Vậy quy trình vẽ được thực hiện như sau:

- ❖ Tính  $P_0$ , vẽ điểm  $(x_0, y_0) = (0, R)$
- ❖ Dựa vào dấu của  $P_0$  ta lại chọn được điểm vẽ tiếp theo  $(x_1, y_1)$  và giá trị  $P_1$
- ❖ Dựa vào dấu của  $P_1$  ta lại chọn được điểm vẽ tiếp theo  $(x_2, y_2)$  và giá trị  $P_2$
- ❖ Quá trình trên được lặp đi lặp lại cho đến khi ta vẽ được điểm nguyên gần nhất với B.

### III.2.a. Tóm tắt thuật toán vẽ đường tròn Bresenham :

- ❖ **Bước 1:**  $P_0 = 3 - 2R$ ;  $(x_0, y_0) = (0, R)$

Vẽ điểm  $(x_0, y_0)$

- ❖ **Bước 2:** Với mỗi giá trị  $i$  ( $i=0, 1, 2, \dots$ ) ta xét dấu  $P_i$ 
  - Nếu  $P_i < 0$ : thì chọn điểm tiếp theo là

$$(x_{i+1}, y_{i+1}) = (x_i+1, y_i)$$

$$P_{i+1} = P_i + 4x_i + 6$$

- Ngược lại ( $P_i \geq 0$ ): thì chọn điểm tiếp theo là

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

$$P_{i+1} = P_i + 4(x_i - y_i) + 10$$

Vẽ điểm  $(x_{i+1}, y_{i+1})$  vừa tìm được

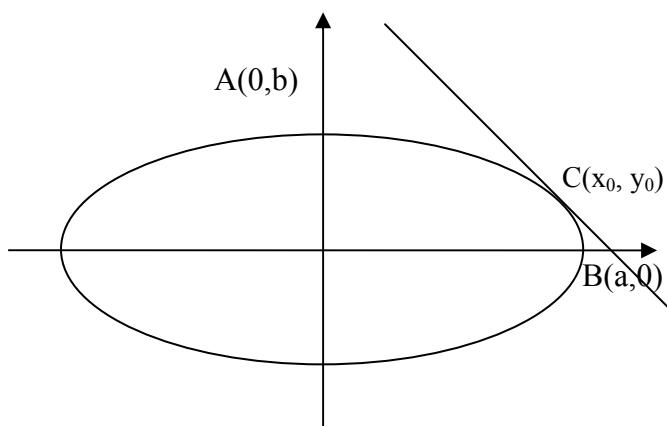
- ❖ **Bước 3:** Lặp lại bước 2 với những giá trị  $i$  tiếp theo, cho đến khi ta vẽ được điểm nguyên gần nhất với  $B$ , nghĩa là  $x_{i+1} = \text{Trunc}(x_b) = \text{Trunc}\left(\frac{R}{\sqrt{2}}\right)$  thì thuật toán kết thúc.

### III.2.b. Cài đặt

- ❖ Sinh viên cần cài đặt một thủ tục vẽ đường tròn theo thuật toán Bresenham và chương trình sử dụng thủ tục để vẽ các đường tròn ngẫu nhiên.

## IV. Thuật toán vẽ Ellipse

Phương trình chính tắc của Ellipse có dạng:  $\frac{x^2}{a^2} + \frac{y^2}{b^2} = 1$  (I)



Để dựng được ellipse rõ ràng là ta chỉ cần tìm cách dựng cung AB, còn các phần còn lại dễ dàng có được bằng cách lấy đối xứng. Song với tư tưởng chung để dựng một đường bất kỳ là cần phải xác định ra các miền mà trên toàn miền đó một biến số biến thiên nhanh hơn một biến số khác.

Rõ ràng trên cung AB thì độ dốc giảm liên tục từ điểm A (độ dốc bằng 0) đến B (độ dốc tiến đến  $-\infty$ ). Xét về tốc độ biến thiên của 2 biến số thì:

- Tốc độ biến thiên của biến số X giảm dần từ A đến B.
- Tốc độ biến thiên của biến số Y tăng dần từ A đến B.

Rõ ràng trên cung AB phải có một điểm mà tại đó tốc độ biến thiên của X và Y là bằng nhau (song x tăng thì y giảm), đó chính là điểm mà tại đó có độ dốc bằng  $-1$ .

Gọi  $C(x_0, y_0)$  là điểm nằm trên cung AB của ellipse mà tiếp tuyến tại đó có độ dốc bằng  $-1$ . Khi đó tiếp tuyến d của ellipse sẽ có dạng:

$$\frac{xx_0}{a^2} + \frac{yy_0}{b^2} = 1$$

Mặt khác do độ dốc của d là:  $-\frac{x_0 b^2}{y_0 a^2} = -1$  nên ta suy ra

$$y_0 = \frac{b^2}{a^2} x_0 \quad (\text{ở đây ta có } x_0 \geq 0) \Rightarrow y_0^2 = \frac{b^4}{a^4} x_0^2 \quad (\text{a})$$

mặt khác do C thuộc ellipse nên ta có:  $\frac{x_0^2}{a^2} + \frac{y_0^2}{b^2} = 1 \quad (\text{b})$

Từ (a) và (b) ta suy ra:  $x_0 = \frac{a^2}{\sqrt{a^2 + b^2}}$  và  $y_0 = \frac{b^2}{\sqrt{a^2 + b^2}}$

Ta sẽ trình bày giả thuật để vẽ cung AC (Đi từ A đến C theo chiều kim đồng hồ). Cung CB được thực hiện một cách tương tự khi ta đổi vai trò của x và y. Các phần còn lại của ellipse có được bằng cách lấy đối xứng.

- ❖ Trên cung AC độ dốc nằm trong đoạn [0,-1], nghĩa là x tăng thì y giảm và tốc độ biến thiên của x lớn hơn của y. Vậy nên tư tưởng của thuật giải dựng cung AC sẽ là cho tham số x biến thiên tuần tự từ  $x_a$  đến  $x_c$  với các giá trị nguyên, và với mỗi giá trị x như vậy ta tìm một giá trị y nguyên gần nhất với giá trị y thực của ellipse tương ứng với x.

Sau đây chúng ta sẽ tìm hiểu thuật toán Bresenham áp dụng cho dựng ellipse.

#### IV.1. Thuật toán Bresenham cho vẽ hình Ellipse

- ❖ Rõ ràng điểm đầu tiên được chọn để vẽ sẽ là điểm A(0,b), nghĩa là:  $(x_0, y_0) = (0, b)$
- ❖ Giả sử đến bước thứ i ta đã chọn được điểm  $(x_i, y_i)$  để vẽ. Câu hỏi đặt ra là đến bước thứ  $i+1$  ta sẽ chọn điểm  $(x_{i+1}, y_{i+1})$  có giá trị bao nhiêu?

Vì điểm tiếp theo sẽ có hoành độ x tăng một giá trị so với giá trị của điểm chọn trước, hay nói cách khác là:

$$x_{i+1} = x_i + 1$$

Đồng thời vì trên cung AC khi x tăng thì y giảm và tốc độ thay đổi của y chậm hơn của x, nên rõ ràng ta thấy là với giá trị x tăng 1 thì giá trị y sẽ giảm đi một lượng  $-1 \leq \Delta y \leq 0$ . Mà điểm chọn bước trước là  $(x_i, y_i)$  nên điểm chọn tiếp theo  $(x_{i+1}, y_{i+1})$  chỉ có thể là một trong hai điểm P( $x_i + 1, y_i$ ) và Q( $x_i + 1, y_i - 1$ ).

Để quyết định được điểm chọn là P hay Q chúng ta hướng đến một biểu thức mà dấu của nó cho phép chúng ta ra quyết định chọn điểm nào.

$$\text{Đặt: } d_1 = y_P^2 - y^2 \quad \text{và} \quad d_2 = y^2 - y_Q^2$$

(giá trị y ở đây là tung độ của cung AC ứng với hoành độ đang xét  $x_{i+1}$ )

$$\begin{aligned} \text{Đặt } P_i &= (d_1 - d_2) \cdot a^2 \\ &= [y_P^2 + y_Q^2 - 2y^2]a^2 = [y_i^2 + (y_i-1)^2 - 2\left(\frac{b^2(a^2 - x_{i+1}^2)}{a^2}\right)]a^2 = [y_i^2 + \\ &\quad (y_i-1)^2 - \frac{2b^2(a^2 - (x_i+1)^2)}{a^2}]a^2 \\ &= y_i^2 \cdot a^2 + (y_i-1)^2 \cdot a^2 - 2b^2[a^2 - (x_i+1)^2] \\ &= y_i^2 \cdot a^2 + (y_i-1)^2 \cdot a^2 - 2b^2a^2 + 2b^2(x_i+1)^2 \end{aligned} \quad (\text{a})$$

(lượng  $a^2$  được đưa vào nhằm mục đích khử mẫu của  $(d_1-d_2)$  song không làm cho  $P_i$  và  $(d_1-d_2)$  khác dấu)

Dấu của biểu thức  $P_i$  cho phép xác định điểm chọn tiếp theo là P hay Q.

- ❖ Khi  $P_i < 0$ : thì điểm P sẽ sát với cung AC hơn điểm Q, do đó ta sẽ chọn điểm P làm điểm biểu diễn (vẽ).
- ❖ Khi  $P_i > 0$ : thì điểm Q sẽ sát với cung AC hơn điểm P, do đó ta sẽ chọn điểm Q làm điểm biểu diễn.
- ❖ Khi  $P_i = 0$ : khoảng cách từ P và Q đến cung AC đều bằng nhau, nên ta có thể chọn P hay Q đều được. Trong tình huống này thuật toán quy ước chọn điểm Q làm điểm biểu diễn

Vậy từ đây ta thấy có thể dựa vào dấu của biểu thức  $P_i$  để ra quyết định chọn điểm tiếp theo.

Để thuật toán được đơn giản người ta tối ưu hóa việc tính  $P_i$  theo công thức truy hồi:

$$P_{i+1} = y_{i+1}^2 \cdot a^2 + (y_{i+1}-1)^2 \cdot a^2 - 2b^2a^2 + 2b^2(x_{i+1}+1)^2 \quad (\text{b})$$

Dấu của  $P_i$  sẽ quyết định giá trị  $P_{i+1}$  cụ thể như sau:

- ❖ Nếu  $P_i < 0$ : thì điểm chọn tiếp theo là  $P(x_i+1, y_i)$ , nghĩa là  $(x_{i+1}, y_{i+1}) = (x_i+1, y_i)$ .

Thay vào (b) ta được:

$$P_{i+1} = y_i^2 \cdot a^2 + (y_i-1)^2 \cdot a^2 - 2b^2a^2 + 2b^2[(x_i+1)+1]^2$$

$$= P_i + 2b^2[2(x_i+1)+1]$$

$$= P_i + 2b^2(2x_i + 3)$$

- ❖ Nếu  $P_i \geq 0$ : thì điểm chọn tiếp theo là  $Q(x_i+1, y_i-1)$ , nghĩa là  $(x_{i+1}, y_{i+1}) = (x_i+1, y_i-1)$ .

Thay vào (b) ta được:

$$P_{i+1} = (y_i-1)^2 \cdot a^2 + (y_i-2)^2 \cdot a^2 - 2b^2a^2 + 2b^2[(x_i+1)+1]^2$$

$$= P_i + a^2(-4y_i + 4) + 2b^2[2(x_i+1)+1]$$

$$= P_i + 4a^2(1-y_i) + 2b^2(2x_i + 3)$$

$$= P_i + 2b^2(2x_i + 3) + 4a^2(1-y_i)$$

Đầu tiên ta chọn điểm  $A(0,b)$ , nghĩa là  $(x_0, y_0) = (0, b)$ , Thay vào (a) ta có:

$$P_0 = y_0^2 \cdot a^2 + (y_0-1)^2 \cdot a^2 - 2b^2a^2 + 2b^2(x_0+1)^2 = b^2a^2 + (b-1)^2a^2 - 2a^2b^2 + 2b^2$$

$$= b^2a^2 + a^2b^2 - 2a^2b + a^2 - 2a^2b^2 + 2b^2 = -2a^2b + a^2 + 2b^2$$

$$= a^2(1-2b) + 2b^2$$

Vậy quy trình vẽ được thực hiện như sau:

- ❖ Tính  $P_0$ , vẽ điểm  $(x_0, y_0) = (0, b)$
- ❖ Dựa vào dấu của  $P_0$  ta lại chọn được điểm vẽ tiếp theo  $(x_1, y_1)$  và giá trị  $P_1$
- ❖ Dựa vào dấu của  $P_1$  ta lại chọn được điểm vẽ tiếp theo  $(x_2, y_2)$  và giá trị  $P_2$
- ❖ Quá trình trên được lặp đi lặp lại cho đến khi ta vẽ được điểm nguyên gần nhất với C.

#### IV.1.a. Tóm tắt thuật toán Bresenham cho vẽ Ellipse:

- ❖ **Bước 1:**  $P_0 = a^2(1-2b) + 2b^2 ; (x_0, y_0) = (0, b)$

Vẽ điểm  $(x_0, y_0)$

- ❖ **Bước 2:** Với mỗi giá trị i ( $i=0, 1, 2, \dots$ ) ta xét dấu  $P_i$ 
  - Nếu  $P_i < 0$ : thì chọn điểm tiếp theo là

$$(x_{i+1}, y_{i+1}) = (x_i+1, y_i)$$

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

- Ngược lại ( $P_i \geq 0$ ): thì chọn điểm tiếp theo là

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

$$P_{i+1} = P_i + 2b^2(2x_i + 3) + 4a^2(1-y_i)$$

Vẽ điểm  $(x_{i+1}, y_{i+1})$  vừa tìm được

- ❖ **Bước 3:** Lặp lại bước 2 với những giá trị i tiếp theo, cho đến khi ta vẽ được điểm nguyên gần nhất với C, nghĩa là  $x_{i+1} = \text{Trunc}(x_C) = \text{Trunc}\left(\frac{a^2}{\sqrt{a^2 + b^2}}\right)$  thì thuật toán kết thúc.

### Chú ý:

- Tóm tắt thuật toán trên chỉ áp dụng cho đoạn AC. Để dựng đoạn BC ta cần có sự thay đổi vai trò của của x và y cũng như a và b. Cụ thể để dựng được cung BC cần hoán đổi trong toàn bộ thuật toán: x thành y và y ngược lại thành x, a thành b và b ngược lại thành a.
- Vì thuật toán chỉ vẽ đến  $\text{Trunc}(x_c)$  nên nếu phần lẻ của  $x_c$  lớn hơn 0.5 (ví dụ  $\text{Trunc}(x_c=7.65)=7$ ). Nếu điều này được thực hiện trên cả 2 cung AC và BC thì sẽ dẫn đến hình ảnh ghép nối của 2 cung là cung AB sẽ thiếu 1 điểm tại C. Để tránh tình trạng này thì chúng ta có thể áp dụng  $\text{Trunc}()$  trên một cung, còn cung còn lại áp dụng  $\text{Round}()$ .

### IV.1.b. Cài đặt thuật toán Bresenham cho dựng Ellipse

Sau đây là một chương trình ví dụ cho thuật toán. Chương trình cài đặt thủ tục vẽ Ellipse có tên là Bre\_Ellipse theo thuật toán trình bày ở trên, và chương trình sử dụng thủ tục Bre\_Ellipse để vẽ các hình Ellipse một cách ngẫu nhiên.

```
uses graph,crt;

procedure init;
var
  grDriver: Integer;
  grMode: Integer;
  ErrCode: Integer;
begin
  grDriver := Detect;
  InitGraph(grDriver, grMode, '');
  ErrCode := GraphResult;
  if ErrCode <> grOk then
    begin Writeln('Graphics error:', GraphErrorMsg(ErrCode));readln;halt end;
end;
procedure Bre_Ellipse(xt,yt:Integer;A,B:longint);
var x,y,P,Const1, Const2:longint; color:byte;
```

```

Procedure Put(x,y:integer);
begin
putpixel(x+xt,-y+yt,color);
putpixel(x+xt,y+yt,color);
putpixel(-x+xt,-y+yt,color);
putpixel(-x+xt,y+yt,color);
end;
begin
Color:=Getcolor;

{Ve cung AC}
const1:=trunc(sqr(a) / sqrt(sqr(a)+sqr(b)));
x:=0;y:=b;P:=a*a*(1-2*b)-2*b*b;put(x,y);
repeat
  if p<0 then
    p:=p+2*b*b*(2*x+3)
  else
    begin
      p:=p+2*b*b*(2*x+3)+4*a*a*(1-y);
      y:=y-1;
    end;
    x:=x+1;
    put(x,y);
until x=const1;

```

```

{Ve cung BC}
{const2:=trunc(sqr(b) / sqrt(sqr(a)+sqr(b)));}
Const2:=Const1+1;

```

{ *Thay vì quá trình lặp được xét trên y, chúng ta có thể làm điều tương tự bằng cách xét trên x. Biết rằng trên toàn bộ cung AB thì x sẽ biến thiên từ 0 đến a, mà trước đó khi dựng cung AC ta đã cho x biến thiên trong đoạn [0, Const1], vậy trên cung BC x phải biến thiên trong đoạn [Const1+1, a] thì sẽ đảm bảo hai cung AC và CB ghép nối liên tục với nhau.* }

```

y:=0;x:=a;P:=b*b*(1-2*a)-2*a*a;put(x,y);
repeat
  if p<0 then
    p:=p+2*a*a*(2*y+3)
  else
    begin
      p:=p+2*a*a*(2*y+3)+4*b*b*(1-x);
      x:=x-1;
    end;
    y:=y+1;
    put(x,y);
until x=const2;
end;
var a,b:longint;

```

```
BEGIN
init;
randomize;
repeat
a:=random(100)+50;b:=random(100)+50;
setcolor(random(15)+1);
Bre_Ellipse(getmaxx div 2,getmaxy div 2,a,b);
until readkey=#27;
closegraph;
END.
```

## V. Bài tập cuối chương

1. Cho điểm A(5,7) và B(15,15). Sử dụng thuật toán Bresenham để cho đê tìm toạ độ các điểm vẽ  $(x_i, y_i)$ .
2. Cài đặt một thủ tục vẽ đường thẳng tổng quát (*xử lý được với tất cả mọi tình huống*) theo thuật toán Bresenham.
3. Sử dụng thuật toán vẽ đường tròn Midpoint để tính giá trị các điểm vẽ  $(x_i, y_i)$  biết rằng  $R=20$ .
4. Cài đặt một thủ tục vẽ đường tròn theo thuật toán MidPoint.
5. Cài đặt một thủ tục cho phép tô màu phần diện tích bên trong của đường tròn. Thủ tục có dạng **FillCircle(x,y:integer; R:word; FillColor: byte);**
6. Cài đặt một thủ tục tô màu phần bên trong của một Ellipse.
7. Hãy xây dựng một thuật toán để dựng đường cong bậc 2 (Parabol) dạng tổng quát:

$$y = ax^2 + bx + c$$

trên một khoảng xác định  $[x_1, x_2]$

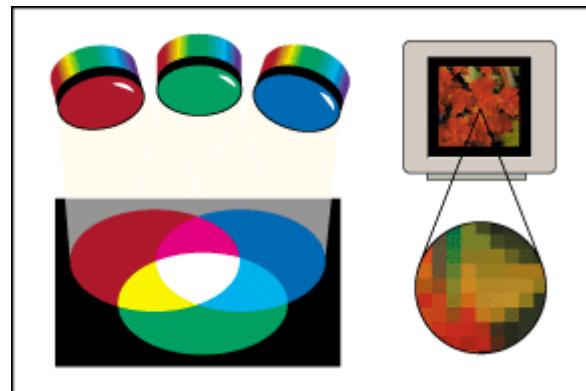
8. Viết chương trình vẽ Parabol:
9. Hãy xây dựng một thư viện đồ họa riêng với các thủ tục vẽ các đường cơ bản do bạn tự viết.

## Chương II: Các hệ màu & cơ chế tổ chức bộ nhớ

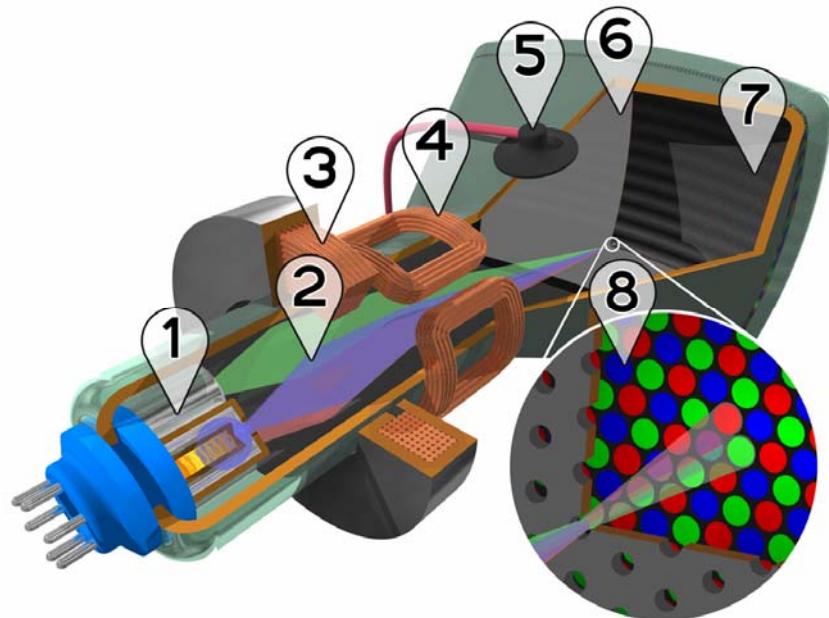
### Card màn hình



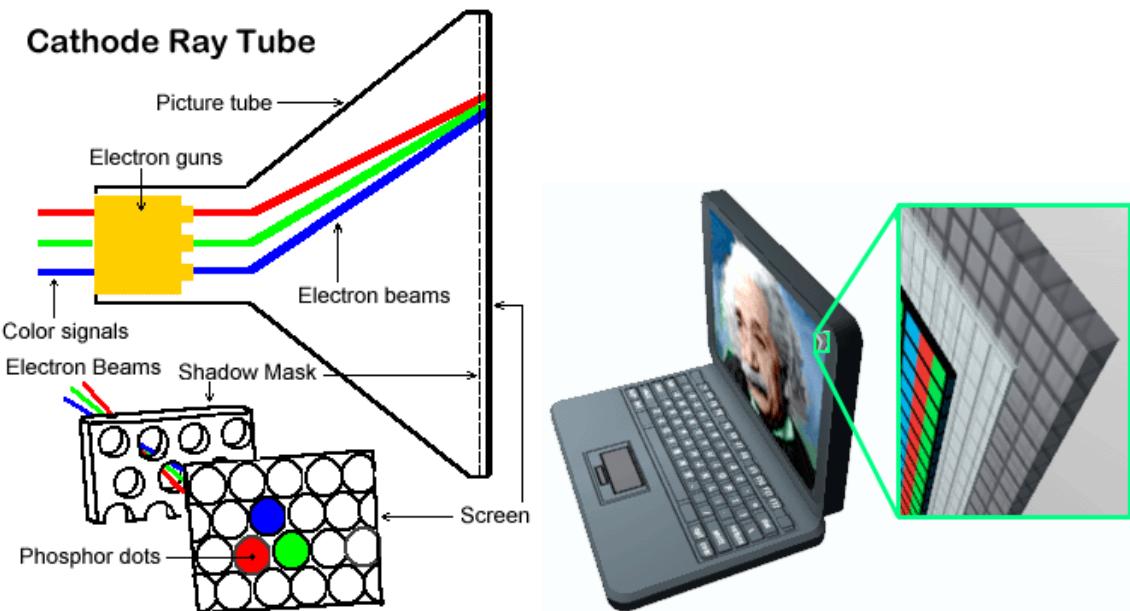
#### I. Đôi nét về cấu trúc màn hình màu



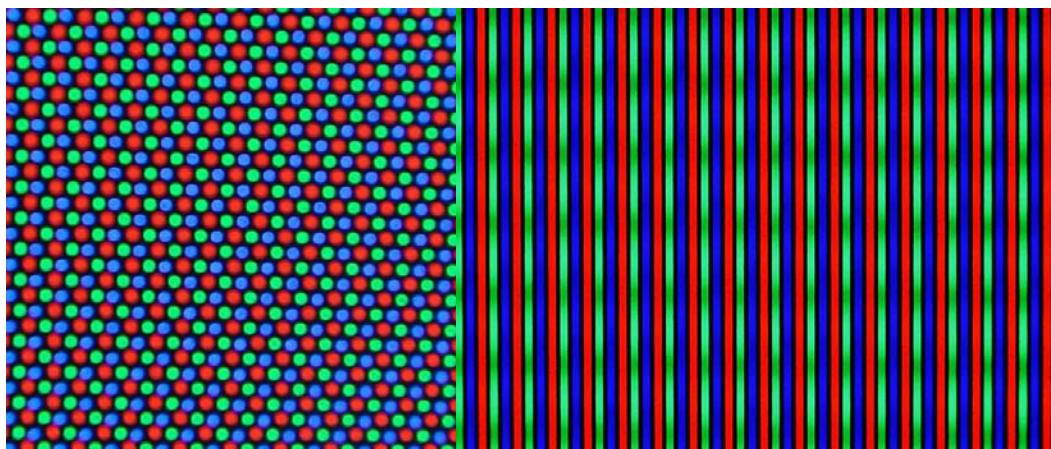
(Màu sắc và sự giao thoa)



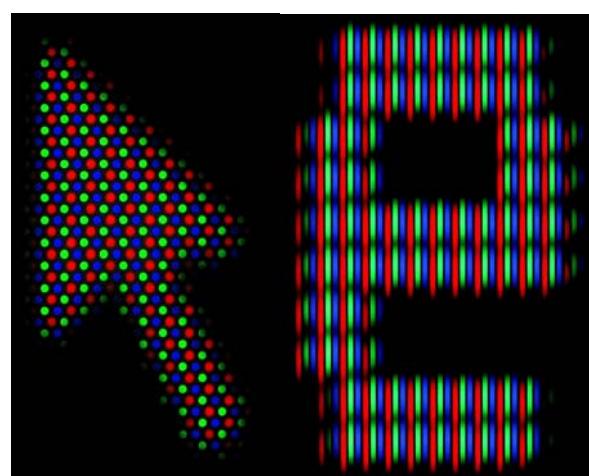
(Hình ảnh cấu tạo chi tiết bên trong một màn hình màu)



2 loại cấu trúc màn hình  
 (a) shadow mask color CRT (b) aperture grille color CRT



Hình ảnh phóng đại cấu trúc 2 loại màn hình màu  
 (a) shadow mask color CRT (b) aperture grille color CRT



(ảnh biểu diễn của một mũ tên màu trắng, và một chữ E trên máy tính được phóng lớn)

## II. Các hệ màu

Trên các thiết bị hiển thị như màn hình máy tính, màn hình Tivi và phần lớn các thiết bị hiển thị màu thông dụng khác người ta chọn 3 màu Red, Green và Blue để biểu diễn tất cả các màu sắc khác nhau của hình ảnh, cũng như người họa sĩ chỉ dùng một số màu cơ bản song các bức tranh được vẽ ra lại rất phong phú về màu sắc. Một câu hỏi đặt ra là tại sao lại chọn 3 màu trên mà không phải là một nhóm màu nào khác. Để trả lời câu hỏi này chúng ta hãy tìm hiểu qua về cấu tạo mắt người.

Mắt con người chúng ta cảm nhận màu sắc thông qua các tế bào võng mạc hình nón. Ba màu Red, Green và Blue được mắt con người cảm nhận rõ nhất, chúng có bước sóng dài lần lượt là 580nm, 545nm và 440nm. Sự hòa trộn của 3 bước sóng này sẽ cho ta được những màu sắc khác. Năm 1981 The Commission International de l'Eclairage (gọi tắt là CIE) đã xây dựng một chuẩn là tất cả các màu nên xây dựng thông qua thành phần màu chính đó là Red, Green và Blue.

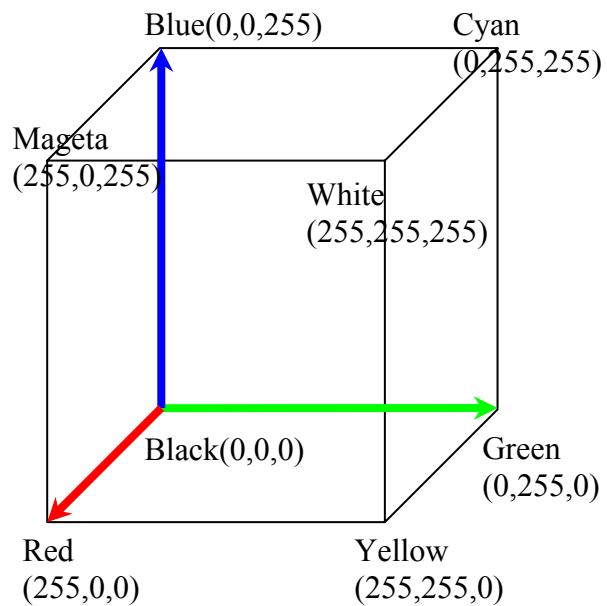
Chuẩn này đầu tiên được xây dựng cho các hệ vô tuyến truyền hình và ngay cả trong các máy tính. Dĩ nhiên không phải tất cả các màu đều có thể biểu diễn thông qua 3 thành phần này, nhưng hầu hết tất cả đều có thể chuyển được.

Hiện nay không phải chỉ có hệ màu RGB mà còn có những hệ khác như: HSV, HSL, YIQ, và một hệ màu mới đây là hệ HVC (Hue, Value, Color). Nay chúng ta hãy xem xét từng hệ một.

### II.1. Hệ RGB

RGB là chữ viết tắt của 3 từ Red, Green và Blue. Hệ này có miền không gian giá trị là một khối 3 chiều. Mỗi màu xác định trên 3 thành phần là R, G, B. Sự gia giảm các thành phần này sẽ tạo ra các màu sắc khác nhau tạo nên một không gian màu.

Cường độ của mỗi thành phần R,G,B được mã hóa trong các mức khác nhau. Có các mức mã hóa khá phổ biến là: Mã hóa 16, 64 và 256 mức. Hiện nay mức mã hóa 256 mức là phổ biến (từ 0 đến 255). Nếu cường độ của mỗi thành phần được mã hóa trong 256 mức thì cần 8bit để mã hóa, vậy một màu biểu diễn bởi 3 thành phần RGB sẽ lưu trữ bởi 24 bit, chế độ này thường được gọi là chế độ màu thực (True color -24 bit) bởi vì nó có thể biểu diễn đến khoảng 16,7 triệu màu.



Không gian màu trong chế độ True color

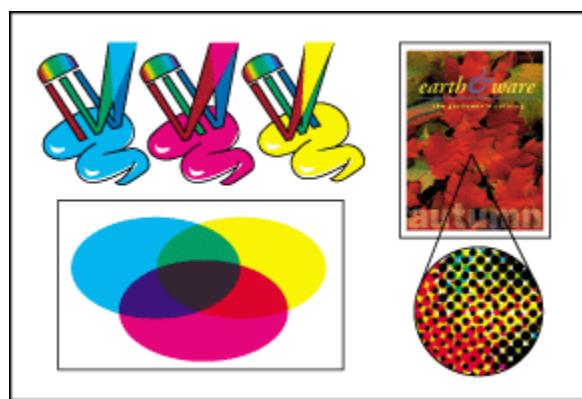
## II.2. Hệ màu CMY

Đó là 3 chữ viết tắt Cyan (màu lục lam), Magenta (màu đỏ tươi), Yellow. Công thức chuyển đổi từ hệ RGB sang hệ CMY như sau:

$$C = 1-R$$

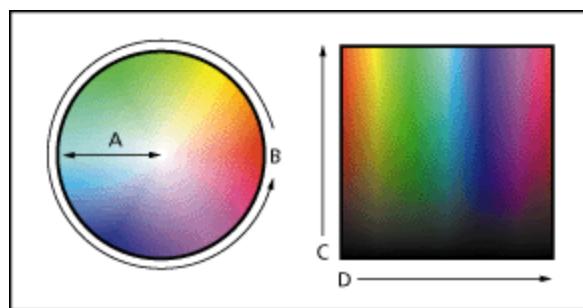
$$M = 1-G$$

$$Y = 1-B$$

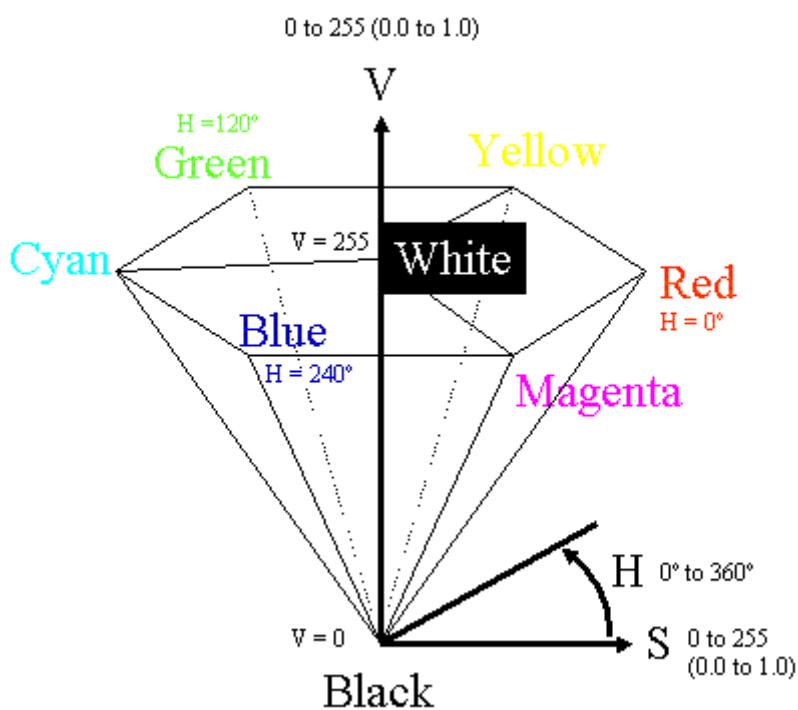


(Hệ màu CMYK chuyên dùng trong in ấn)

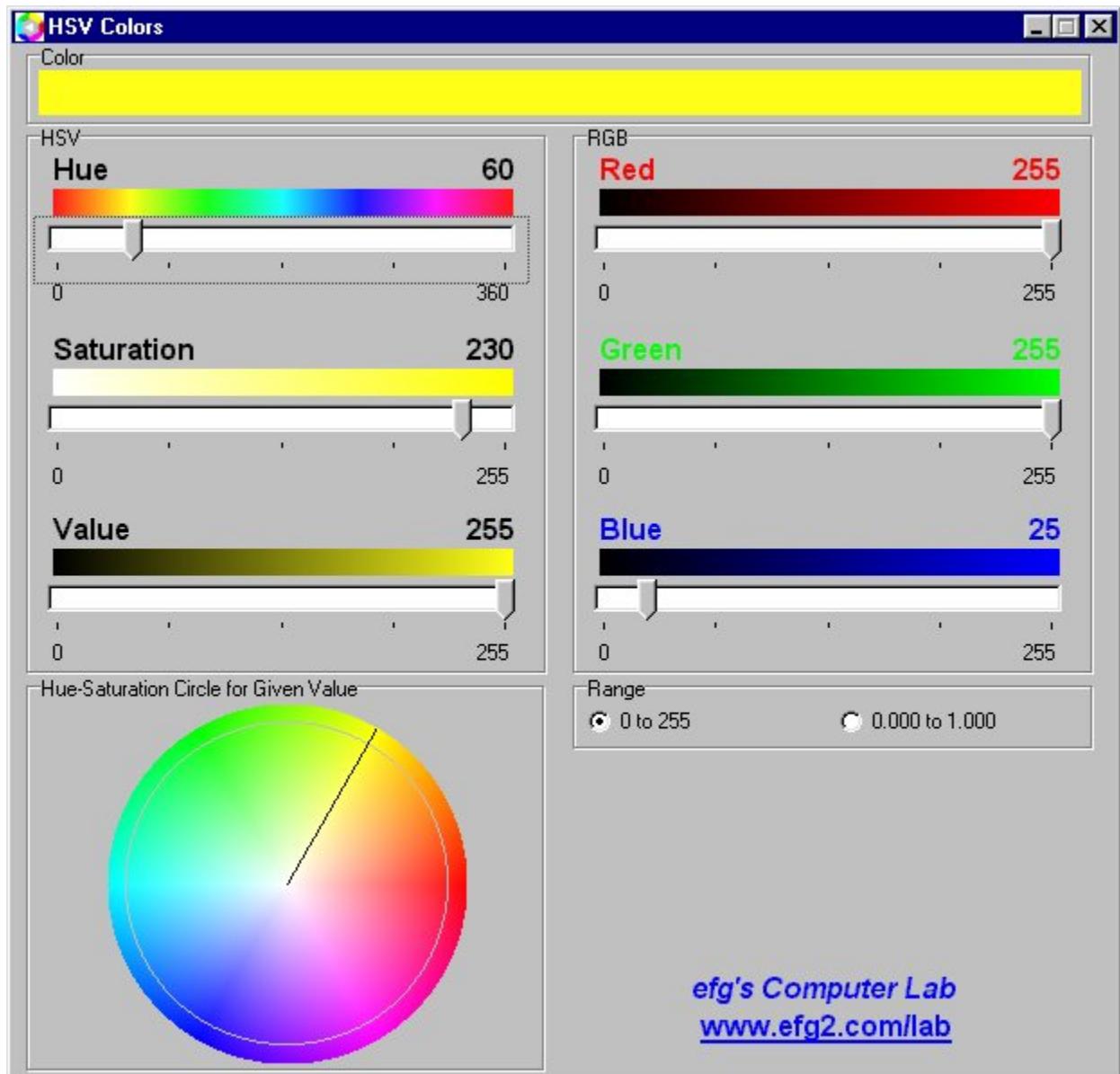
### II.3. Hệ màu HSV



Đó là những chữ viết tắt cho 3 chữ sau: Hue (màu sắc), Saturation (sự bão hòa) và Value. Ba màu Red, Green và Blue có lẽ là 3 màu mà mắt con người cảm nhận ánh sáng chứ không phải là màu mà mắt con người cảm nhận màu sắc. Màu sắc (Hue, hay color) được đo bởi tần số ánh sáng, còn độ sáng được đo bởi cường độ. Màu càng sáng thì cường độ càng lớn. Hệ HSV được biểu diễn bằng hình vẽ sau:

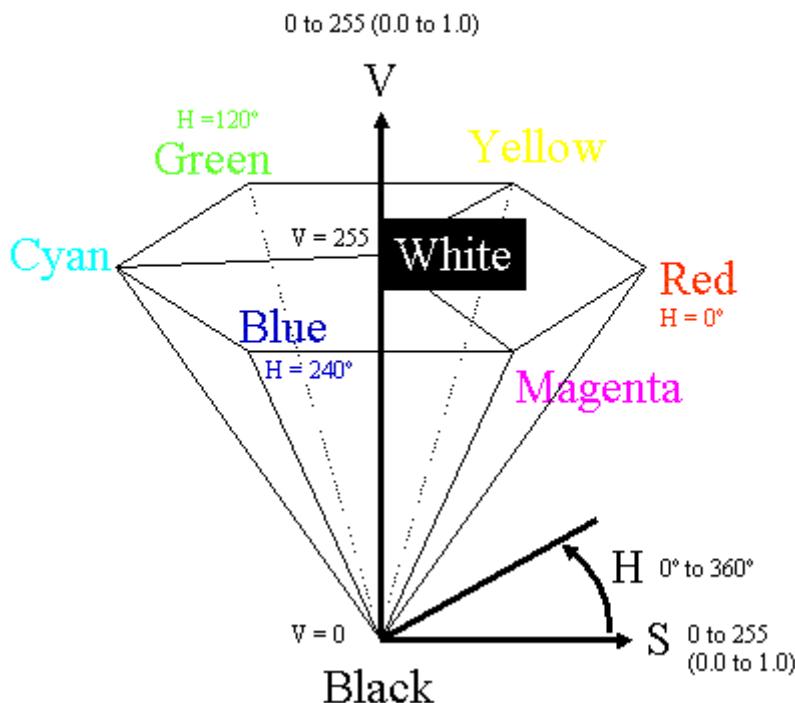


Hue có giá trị từ  $0 \rightarrow 360$ , trong khi S và V có giá trị trong khoảng  $0 \rightarrow 255$  (hoặc ở dạng số thực là  $0.0 \rightarrow 1.0$ ). Màu đỏ hoàn toàn được biểu diễn là  $(0 \text{ độ}, 255, 255)$ , màu xanh (green) được biểu diễn là  $(120 \text{ độ}, 255, 255)$ .



Dưới đây là phần tham khảo về các hàm chuyển đổi qua lại giữa 2 hệ màu RGB và HSV

## Hue-Saturation-Value Hexcone



### RGBtoHSV

USES

```
Math,      // MaxValue
IEEE754, // NaN, IsNaN
SysUtils; // Exception
```

TYPE

```
EColorError = CLASS(Exception);
TReal = DOUBLE;
```

...

```
// RGB, each 0 to 255, to HSV.
// H = 0.0 to 360.0 (corresponding to 0..360.0 degrees around hexcone)
// S = 0.0 (shade of gray) to 1.0 (pure color)
// V = 0.0 (black) to 1.0 {white)
```

```
// Based on C Code in "Computer Graphics -- Principles and Practice,"
// Foley et al, 1996, p. 592.
```

PROCEDURE RGBToHSV (CONST R,G,B: TReal; VAR H,S,V: TReal);

VAR

```
Delta: TReal;
Min : TReal;
```

BEGIN

```
Min := MinValue( [R, G, B] ); // USES Math
V := MaxValue( [R, G, B] );
```

```
Delta := V - Min;  
  
// Calculate saturation: saturation is 0 if r, g and b are all 0  
IF      V = 0.0  
THEN S := 0  
ELSE S := Delta / V;  
  
IF      S = 0.0  
THEN H := NaN // Achromatic: When s = 0, h is undefined  
ELSE BEGIN    // Chromatic  
    IF      R = V  
    THEN // between yellow and magenta [degrees]  
        H := 60.0 * (G - B) / Delta  
    ELSE  
        IF      G = V  
        THEN // between cyan and yellow  
            H := 120.0 + 60.0 * (B - R) / Delta  
        ELSE  
            IF      B = V  
            THEN // between magenta and cyan  
                H := 240.0 + 60.0 * (R - G) / Delta;  
  
    IF H < 0.0  
    THEN H := H + 360.0  
END  
END {RGBtoHSV};
```

### HSVtoRGB

```
// Based on C Code in "Computer Graphics -- Principles and Practice,"  
// Foley et al, 1996, p. 593.
```

```
//
```

```
// H = 0.0 to 360.0 (corresponding to 0..360 degrees around hexcone)
```

```
// NaN (undefined) for S = 0
```

```
// S = 0.0 (shade of gray) to 1.0 (pure color)
```

```
// V = 0.0 (black) to 1.0 (white)
```

```
PROCEDURE HSVtoRGB (CONST H,S,V: TReal; VAR R,G,B: TReal);
```

```
VAR
```

```
    f : TReal;
```

```
    i : INTEGER;
```

```
    hTemp: TReal; // since H is CONST parameter
```

```
    p,q,t: TReal;
```

```
BEGIN
```

```
    IF      S = 0.0 // color is on black-and-white center line
```

```
    THEN BEGIN
```

```
        IF      IsNaN(H)
```

```

THEN BEGIN
  R := V;           // achromatic: shades of gray
  G := V;
  B := V
END
ELSE RAISE EColorError.Create('HSVtoRGB: S = 0 and H has a value');
END

ELSE BEGIN // chromatic color
  IF      H = 360.0    // 360 degrees same as 0 degrees
  THEN hTemp := 0.0
  ELSE hTemp := H;

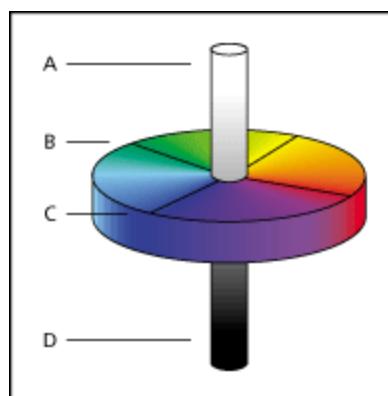
  hTemp := hTemp / 60;   // h is now IN [0,6)
  i := TRUNC(hTemp);    // largest integer <= h
  f := hTemp - i;        // fractional part of h

  p := V * (1.0 - S);
  q := V * (1.0 - (S * f));
  t := V * (1.0 - (S * (1.0 - f)));

CASE i OF
  0: BEGIN R := V; G := t; B := p END;
  1: BEGIN R := q; G := V; B := p END;
  2: BEGIN R := p; G := V; B := t END;
  3: BEGIN R := p; G := q; B := V END;
  4: BEGIN R := t; G := p; B := V END;
  5: BEGIN R := V; G := p; B := q END
END
END
END {HSVtoRGB};

```

Ngoài một số hệ màu trên còn có một vài hệ màu khác:



(Hệ màu Lab)

### **III. Cơ chế tổ chức bộ nhớ Card màn hình**

Bộ nhớ màn hình được đánh địa chỉ lôgic bắt đầu từ:

B800:0000 với chế độ text

A000:0000 với các chế độ đồ họa

Trong phần này ta chỉ đề cập đến chế độ đồ họa.

Thông tin hiển thị trên màn hình được bố trí trong bộ nhớ bắt đầu từ A000:0000, song cách bố trí như thế nào lại phụ thuộc vào chế độ (mode) màn hình mà ta chọn để hoạt động. Thông thường một card điều khiển đồ họa có thể hoạt động trong nhiều mode khác nhau, và để ra chỉ thị cho card hoạt động theo mode ta cần gọi đến các phục vụ của Bios với hàm phục vụ đặt chế độ màn hình và tham số mode.

Do có rất nhiều Mode đồ họa khác nhau và mỗi mode lại có một cơ chế bố trí thông tin riêng. Trong phần này chỉ trình bày hai mode:

- Một mode khá thông dụng, được dùng chủ yếu trong các trò chơi đó là mode 13Hexa, đây là mode với độ phân giải là 320x200 với 256 màu, trong chế độ này một byte trong vùng nhớ màn hình lưu trữ thông tin của một điểm, hay nói cách khác là có sự tương ứng 1 – 1 giữa điểm và byte nhớ trong vùng nhớ màn hình.
- Chuẩn Vesa (**Video Electronics Standards Association**): Hầu như tất cả các Card điều khiển màn hình thông dụng đều hỗ trợ mode này. Với mode bạn có thể đặt độ phân giải từ 640x400, 640x480,... lên đến 1024x800 hay cao hơn nữa tùy vào khả năng kỹ thuật của Card mà bạn có. Tương tự màu sắc có thể từ 16 màu, 256 màu, Hight color -16 bit hay True color - 24 bit hay 32 bít.

#### **III.1. Cơ chế hoạt động của chế độ màn hình 320x200x256 màu**

Trong chế độ này màn hình được chia ra 320 cột và 200 hàng tạo nên 320x200 pixel, tuy độ phân giải thấp song nó lại có ưu điểm là truy cập nhanh chóng vì cơ chế bố trí thông tin đơn giản. Pixel[cột 0,dòng 0] (ở góc trên bên trái màn hình) tương ứng với byte nhớ có địa chỉ A000:0000, tương tự Pixel[cột 1,dòng 0] tương ứng với byte nhớ có địa chỉ A000:0001,...

Hay nói một cách tổng quát thì pixel tại cột x dòng y tương ứng với byte nhớ [A000:(y\*320+x)]

Lệnh sau cho phép vẽ màu có số 255 cho điểm có toạ độ (x,y):

Mem[\$A000:(y\*320+x)]:=255

Câu lệnh trên cho phép vẽ ra màn hình 1 điểm có màu 255 tại vị trí (x,y). Song màu có số 255 là màu như thế nào lại phụ thuộc vào thanh ghi màu số 255 lưu trữ các giá trị RGB mô tả cách hoà ra màu 255 như thế nào.

Để quyết định màu cho một thanh ghi màu ta cần đến các phục vụ đặt giá trị cho thanh ghi màu của Bios.

### III.1.a. Cài đặt

Sau đây là một thư viện đồ họa (*mang tính minh họa*) xây dựng trên mode 13 hexa với độ phân giải và màu sắc là 320x200x256 màu.

```
unit Graph_13;
Interface
{*****}
uses crt,dos;

type screen=array[0..199,0..319] of byte;
const MaxPolygon=1000;
type pointtype=record
    x,y:integer;
end;

Polygo=array[1..MaxPolygon] of pointtype;
Polygon=^PolyGo;
FillPatternType = array[1..8] of byte;
FillStyle=array[1..8,1..8] of byte;
var   _Scr:^screen;_Color,_BKColor,_FillColor:byte;_X,_Y:integer;
      _FillPattern:FillPatternType;_Fill:FillStyle;
      _Page:array[1..5] of ^screen;_CurrentPage,_CountPage:byte;
      (*****)
procedure Init(mode:byte);
Procedure setRGB(num:byte;R,G,B:byte);
procedure SetGroupcolorRegs(starcolorReg,num:integer; segment,offset:word);
Function NewPage:byte;
Function ReleasePage(Page:byte):boolean;
Function SetActive(page:byte):boolean;
Procedure Displace(page:byte);
Function GetmaxX:integer;
Function GetmaxY:integer;
procedure setcolor(color:byte);
Function GetColor:byte;
Procedure MakeFillStyle;
procedure SetFillPattern(Pattern: FillPatternType; Color: Word);
Procedure SetfillStyle(Pattern:word;Color:byte);
Procedure MoveTo(x,y:integer);
Procedure putpixel(x,y:integer;Color:byte);
Procedure lineNgang(x1,x2,y:integer);
procedure line(x1,y1,x2,y2:integer);
Procedure LineTo(x,y:integer);
Procedure Circle(xt,yt,R:integer);
Procedure FillCircle(xt,yt,R:integer);
Procedure Bar(x1,y1,x2,y2:integer);
Procedure FillTamGiac(x1,y1,x2,y2,x3,y3:integer);
Procedure drawpoly(N:integer;A:polygon);
procedure fillpoly(N:integer;A:polygon);
procedure Fillpoly1(N:integer;A:polygon);
Procedure Spline(N:integer;P:polygon);
```

```
{*****  
Implementation  
{*****  
procedure Init(mode:byte);  
begin  
asm  
    mov ah,0  
    mov al,mode  
    int $10  
end;  
if mode=$13 then  
    begin _scr:=ptr($a000,$0000);_X:=0;_Y:=0;  
        _Color:=255;_Color:=0;_FillColor:=255;  
        _CurrentPage:=0;_CountPage:=0;  
    end;  
end;  
Procedure setRGB(num:byte;R,G,B:byte);  
(* Dat noi dung thanh ghi mau *)  
var Regs:registers;mau:longint;  
begin  
    with Regs do  
    begin  
        ah:=$10;  
        al:=$10;  
        bx:=num;  
        dh:=R;  
        ch:=G;  
        cl:=B;  
        intr($10,regs);  
    end  
end;  
procedure SetGroupcolorRegs(starcolorReg,num:integer; segment,offset:word);  
(* Dat mot nhom thanh ghi mau *)  
var Regs:registers;  
begin  
    with regs do  
    begin  
        ah:=$10; al:=$12;  
        bx:=starcolorreg;cx:=num;  
        es:=segment; dx:=offset;  
        intr($10,regs);  
    end;  
end;  
Function NewPage:byte;  
  
begin  
if _CountPage<5 then  
    begin  
        _CountPage:= _CountPage+1;
```

```
Getmem(_Page[_CountPage],320*200);
NewPage:=_CountPage;
fillchar(_Page[_CountPage]^,320*200,0);
end
else
  NewPage:=0;
end;

Function ReleasePage(Page:byte):boolean;
var i:byte;
begin
if (Page>0)and(Page<=_CountPage) then
  begin
    if (Page=_CountPage) then
      Freemem(_Page[Page],320*200)
    else
      begin
        Freemem(_Page[Page],320*200);
        for i:=Page to _CountPage-1 do
          _Page[i]:= _Page[i+1];
        end;
      _CountPage:= _CountPage-1;
      ReleasePage:=true
    end
  else
    ReleasePage:=false;
end;

Function SetActive(page:byte):boolean;
begin
  if (page>0)and(page<=_CountPage) then
    begin
      _Scr:=@_Page[Page]^;
      _CurrentPage:=Page;
    end
  else
    if page=0 then _scr:=ptr($a000,$0000)
end;
Procedure Displace(page:byte);
begin
  if (page>0)and(page<=_CountPage) then
    begin
      Move(_Page[Page]^,mem[$A000:0000],320*200);
    end;
end;
Function GetmaxX:integer;
begin
  GetmaxX:=319;
end;
Function GetmaxY:integer;
```

```
begin
    GetmaxY:=199;
end;
procedure setcolor(color:byte);
begin
    _Color:=color;
end;
Function GetColor:byte;
begin
    GetColor:=_Color;
end;
Procedure MakeFillStyle;
var i,j:byte;
begin
for i:=1 to 8 do
    for j:=1 to 8 do
        if ((_FillPattern[i] shr (8-j)) and 1)=1 then
            _Fill[i,j]:= _FillColor
        else _Fill[i,j]:= _BKColor;
end;
procedure SetFillPattern(Pattern: FillPatternType; Color: Word);
begin
    _FillPattern:=Pattern;
    MakeFillStyle;
end;
Procedure MakePattern(Pattern:word);
var p:FillPatternType;
begin
    Pattern:=Pattern mod 12;
    case Pattern of
        0:begin p[1]:=$0;p[2]:=0;p[3]:=0;p[4]:=0;p[5]:=0;
              p[6]:=0;p[7]:=0;p[8]:=0;
            end;
        1:begin p[1]:=$ac;p[2]:=$bc;p[3]:=$ce;p[4]:=$ee;p[5]:=ff;
              p[6]:=$fe;p[7]:=1;p[8]:=f1;
            end;
        2:begin
            end;
        3:begin
            end;
        4:begin
            end;
        5:begin
            end;
        6:begin
            end;
        7:begin
            end;
        8:begin
            end;
    end;
```

```
9:begin
  end;
10:begin
  end;
11:begin
  end;
end;
_FillPattern:=P;
MakeFillStyle;
end;
Procedure SetfillStyle(Pattern:word;Color:byte);
begin
  _FillColor:=Color;MakePattern(Pattern);
end;
Procedure MoveTo(x,y:integer);
begin
  _X:=x;_Y:=y;
end;
Procedure putpixel(x,y:integer;Color:byte);
begin
  _Scr^[_Y,x]:=Color;
end;
{***** Thu tuc LineNgang *****}
procedure lineNgang(x1,x2,y:integer);
var segment,offset,l:word;tam:integer;
begin
if x1>x2 then begin tam:=x1;x1:=x2;x2:=tam end;
segment:=seg(_Scr^[_Y,x1]);offset:=ofs(_Scr^[_Y,x1]);l:=x2-x1+1;
```

ASM

```
push ES
mov DI,offset
mov AX,segment
mov ES,AX
mov AL,_Color
mov CX,l
clc
rep stosb
pop ES
end;
end;
{***** Ket thuc Thu tuc LineNgang *****}
{***** Thu tuc Line *****}
procedure line(x1,y1,x2,y2:integer);
{***** Thu tuc hoan doi 2 dau mut *****}
procedure HoanDoiDau(var x1,y1,x2,y2:integer);
var tg:integer;
begin
  tg:=x1;x1:=x2;x2:=tg;
  tg:=y1;y1:=y2;y2:=tg;
```

```

end;
{***** Than cua thu tuc Line *****}
var P,dtx,dty,const1,const2,i:longint;segment,ofset,l:word;
begin
dtx:=x2-x1;dty:=y2-y1;
if dtx=0 then
begin
  if y1>y2 then begin i:=y1;y1:=y2;y2:=i end;
  for y1:=y1 to y2 do _Scr^[y1,x1]:=Color;
end
else
if dty=0 then linengang(x1,x2,y1)
else
if (dty*dtx>0)and(abs(dty)<=abs(dtx)) then
begin {Line (0,1)}
  if x1>x2 then HoanDoiDau(x1,y1,x2,y2);
  dtx:=x2-x1;dty:=y2-y1;
  const1:=dty*2; const2:=const1-2*dtx;
  p:=const1-dtx; _Scr^[y1,x1]:=Color;
  while x1<x2 do
    begin
      x1:=x1+1;
      if p<0 then
        begin
          _Scr^[y1,x1]:=Color;
          p:=p+const1;
        end
      else
        begin
          y1:=y1+1;
          _Scr^[y1,x1]:=Color;
          p:=p+const2;
        end;
    end;
  end
else
if (dtx*dty>0) then
begin {Line (1,vocung)}
  HoanDoiDau(x1,x2,y1,y2);
  if x1>x2 then HoanDoiDau(x1,y1,x2,y2);
  dtx:=x2-x1;dty:=y2-y1;
  const1:=dty*2; const2:=const1-2*dtx;
  p:=const1-dtx; _Scr^[x1,y1]:=Color;
  while x1<x2 do
    begin
      x1:=x1+1;
      if p<0 then
        begin
          _Scr^[x1,y1]:=Color;
          p:=p+const1;
        end
    end;
  end;
end;

```

```
        end
    else
        begin
            y1:=y1+1;
            _Scr^ [x1,y1]:= _Color;
            p:=p+const2;
        end;
    end;
end
else
if (abs(dty)<=abs(dtx)) then
begin {Line [-1,0]}
    y1:=-y1;y2:=-y2;
    if x1>x2 then HoanDoiDau(x1,y1,x2,y2);
    dtx:=x2-x1;dty:=y2-y1;
    const1:=dty*2; const2:=const1-2*dtx;
    p:=const1-dtx; _Scr^ [-y1,x1]:= _Color;
    while x1<x2 do
        begin
            x1:=x1+1;
            if p<0 then
                begin
                    _Scr^ [-y1,x1]:= _Color;
                    p:=p+const1;
                end
            else
                begin
                    y1:=y1+1;
                    _Scr^ [-y1,x1]:= _Color;
                    p:=p+const2;
                end;
        end;
    end
else
begin{Line (-1,am vo cung)}
    HoanDoiDau(x1,x2,y1,y2);
    y1:=-y1;y2:=-y2;
    if x1>x2 then HoanDoiDau(x1,y1,x2,y2);
    dtx:=x2-x1;dty:=y2-y1;
    const1:=dty*2; const2:=const1-2*dtx;
    p:=const1-dtx; _Scr^ [x1,-y1]:= _Color;
    while x1<x2 do
        begin
            x1:=x1+1;
            if p<0 then
                begin
                    _Scr^ [x1,-y1]:= _Color;
                    p:=p+const1;
                end
            else
```

```

begin
  y1:=y1+1;
  _Scr^[_x1,-y1]:=_Color;
  p:=p+const2;
end;
end;
{***** Ket thuc Thu tuc Line *****}

{***** Thu tuc LineTo *****}
Procedure LineTo(x,y:integer);
begin
  Line(_X,_Y,x,y);_X:=x;_Y:=y;
end;
{***** Ket thuc Thu tuc LineTo *****}
{***** Thu tuc Circle *****}
procedure Circle(xt,yt,R:integer);
var x,y,p:integer;mau:byte;s1:string;t,t1,t2:real;ch:char;
procedure putdiem(c,h:integer);
begin
  _Scr^[_h+yt,c+xt]:=_Color;
  _Scr^[_c+yt,h+xt]:=_Color;
  _Scr^[_-h+yt,-c+xt]:=_Color;
  _Scr^[_-c+yt,-h+xt]:=_Color;
  _Scr^[_h+yt,-c+xt]:=_Color;
  _Scr^[_c+yt,-h+xt]:=_Color;
  _Scr^[_-h+yt,c+xt]:=_Color;
  _Scr^[_-c+yt,h+xt]:=_Color;
end;
begin
  P:=1-r;
  putdiem(0,R);x:=0;y:=R;
  while x<y do
    begin
      if p<0 then
        p:=p+2*x+3
      else
        begin
          p:=p+2*(x-y)+5;
          y:=y-1;
        end;
      x:=x+1;
      putdiem(x,y);
    end;
end;
{***** Ket thu Thu tuc Circle *****}
{***** Thu tuc FillCircle *****}
procedure FillCircle(xt,yt,R:integer);
var x,y,p:integer;mau:byte;s1:string;t,t1,t2:real;ch:char;

```

```

procedure LineScan4(c,h:integer);
begin
line(c+xt,h+yt,-c+xt,h+yt);
line(c+xt,-h+yt,-c+xt,-h+yt);
line(h+xt,c+yt,-h+xt,c+yt);
line(h+xt,-c+yt,-h+xt,-c+yt);
end;
begin
P:=1-r;
LineScan4 (0,R);x:=0;y:=R;
while x<y do
begin
  if p<0 then
    p:=p+2*x+3
  else
    begin
      p:=p+2*(x-y)+5;
      y:=y-1;
    end;
  x:=x+1;
  LineScan4 (x,y);
end;
end;
{***** Ket thu Thu tuc FillCircle *****}
{***** Thu tuc Bar *****}
Procedure Bar(x1,y1,x2,y2:integer);
var tam:integer;
begin
if y1>y2 then begin tam:=y1;y1:=y2;y2:=tam end;
for tam:=y1 to y2 do lineNgang(x1,x2,tam);
end;
{***** Ket thuc Thu tuc Bar *****}
{***** Thu tuc FillTamGiac *****}
Procedure FillTamGiac(x1,y1,x2,y2,x3,y3:integer);
procedure FillFlatTop(x1,x2,x3,y1,y2,y3:integer);
{Qui dinh y1=y2}
var X1_Change_Per_Pixel,X2_Change_Per_Pixel,x1_now,x2_now:real;y:integer;
begin
X1_Change_Per_Pixel:=(x3-x1)/(y3-y1);
X2_Change_Per_Pixel:=(x3-x2)/(y3-y2);
x1_now:=x1;x2_now:=x2;
  for y:=y1 to y3 do
begin
  linengang(round(x1_now),round(x2_now),y);
  x1_now:=x1_now+X1_Change_Per_Pixel;
  x2_now:=x2_now+X2_Change_Per_Pixel;
end;
end;
procedure FillFlatBottom(x1,x2,x3,y1,y2,y3:integer);
{Qui dinh y1=y3}

```

```

var X1_Change_Per_Pixel,X2_Change_Per_Pixel,x1_now,x2_now:real;y:integer;
begin
  X1_Change_Per_Pixel:=(x2-x1)/(y2-y1);
  X2_Change_Per_Pixel:=(x3-x2)/(y3-y2);
  x1_now:=x2;x2_now:=x2;
  for y:=y2 to y1 do
    begin
      linengang(round(x1_now),round(x2_now),y);
      x1_now:=x1_now+X1_Change_Per_Pixel;
      x2_now:=x2_now+X2_Change_Per_Pixel;
    end;
  end;
procedure Sap(var x1,y1,x2,y2:integer);
var tam:integer;
begin
if y1>y2 then
  begin
    tam:=y1; y1:=y2; y2:=tam;
    tam:=x1; x1:=x2; x2:=tam;
  end;
end;
var x4,y4:integer;
begin
if (y1=y2)and(y2=y3) then
  begin linengang(x1,y1,x2);linengang(x1,y1,x3);exit; end;
  sap(x1,y1,x2,y2);sap(x1,y1,x3,y3);sap(x2,y2,x3,y3);
if y1=y2 then
  FillFlatTop(x1,x2,x3,y1,y2,y3)
else
  if y2=y3 then
    FillFlatBottom(x2,x1,x3,y2,y1,y3)
  else
    begin
      y4:=y2;x4:=x1+round((x3-x1)*(y2-y1)/(y3-y1));
      FillFlatBottom(x2,x1,x4,y2,y1,y4);
      FillFlatTop(x2,x4,x3,y2,y4,y3);
    end;
  end;
{***** Ket thuc Thu tuc FillTamGiac *****}
{***** Thu tuc Spline *****}
Procedure Spline(N:integer;P:polygon);
procedure tinh(u:real;var Pu:Pointtype;P:Polygon;K:integer);
const t=0;
var u2,u3,s:real;
begin
  u2:=u*u;u3:=u2*u;s:=(1-t)/2;
  Pu.x:=round(P^[k-1].x*(-s*u3+2*s*u2-s*u)+P^[k].x*((2-s)*u3+(s-3)*u2+1)
  +P^[k+1].x*((s-2)*u3+(3-2*s)*u2+s*u)+P^[k+2].x*(s*u3-s*u2));
  Pu.y:=round(P^[k-1].y*(-s*u3+2*s*u2-s*u)+P^[k].y*((2-s)*u3+(s-3)*u2+1)
  +P^[k+1].y*((s-2)*u3+(3-2*s)*u2+s*u)+P^[k+2].y*(s*u3-s*u2));

```

```

end;
Procedure TaoDuLieu(N:integer;P:polygon;var Pt:Polygon);
begin
    Getmem(Pt,sizeof(pointtype)*(N+2));
    move(P^,Pt^[2],n*sizeof(pointtype));Pt^[1]:=Pt^[2];Pt^[n+2]:=Pt^[n];
end;
{***** Than Thu tuc Spline *****}
var u,Bn:real;Pu:Pointtype;bd:boolean;
k:integer;Pt:polygon;
Begin
TaoDuLieu(n,p,pt);
{for k:=2 to n+1 do
with Pt^[k] do
    bar(round(x)-3,round(y)-3,round(x)+3,round(y)+3);}

Bn:=1/(20);bd:=true;
for k:=2 to n do
begin
    u:=0;
    repeat
        tinh(u,Pu,Pt,K);
        if bd then
            begin
                moveto(round(Pu.x),round(Pu.y));
                bd:=false;
            end
        else
            lineto(round(Pu.x),round(Pu.y));
        u:=u+bn;
    until U>1;
end;
Freemem(Pt,sizeof(pointtype)*(N+2));
End;
{***** Ket thuc Thu tuc Spline *****}
Procedure drawpoly(N:integer;A:polygon);
var i:integer;
begin
for i:=1 to N-1 do
    line(A^[i].x,A^[i].y,A^[i+1].x,A^[i+1].y);
If (A^[1].x<>A^[n].x)and(A^[1].y<>A^[n].y) then
    line(A^[1].x,A^[1].y,A^[n].x,A^[n].y);
end;
{***** Thu tuc FillPoly *****}
procedure Fillpoly1(N:integer;A:polygon);
var i,j,ymid,min,max,chiso,detal:integer;tam:pointtype;ch:char;
daui,dauj:shortint;
begin
if n=1 then putpixel(A^[1].x,A^[1].y,_Color)
else
if n=2 then line(A^[1].x,A^[1].y,A^[2].x,A^[2].y)

```

---

```

else
begin
    min:=A^[1].y;max:=A^[1].y;
    for i:=2 to n do
        begin
            if A^[i].y<min then min:=A^[i].y;
            if A^[i].y>max then max:=A^[i].y;
            end;
        ymid:=(min+max) div 2;chiso:=1;detal:=abs(A^[1].y-ymid);
        for i:=2 to n do
            if detal >abs(A^[i].y-ymid) then
                begin detal:=abs(A^[i].y-ymid); chiso:=i; end;
            i:=chiso;
            if chiso<>n then j:=chiso+1 else j:=1;
            while (j<>chiso) do
                begin
                    if (i<n) then i:=i+1 else i:=1;
                    if (j<n) then j:=j+1 else j:=1;
                    FillTamGiac(A^[chiso].x,A^[chiso].y,A^[i].x,A^[i].y,A^[j].x,A^[j].y);
                    ch:=readkey;
                end;
            end;
        end;
    {***** Thu tuc FillPoly *****}
procedure fillpoly(N:integer;A:polygon);
type diemcat=record
    x:real;
    canhthu:byte;
end;
type dlcanh=record
    y_top:integer;
    x :integer;
    delta_y :integer;
    x_change_per_scan:real;
end;
dsc=array[1..MaxPolygon] of dlcanh;
DSCanh=^dsc;
{*****}
procedure taodscanh(var A:polygon;n:integer;var B:dscanh;var m:integer);
var i:integer;
begin
Getmem(B,sizeof(Pointtype)*n);
for i:=1 to n-1 do
begin
    if A^[i].y<A^[i+1].y then
        begin B^[i].y_top:=A^[i].y;B^[i].x:=A^[i].x end
    else
        begin B^[i].y_top:=A^[i+1].y;B^[i].x:=A^[i+1].x end;
    B^[i].delta_y:=abs(A^[i].y-A^[i+1].y);

```

```

if (B^i.delta_y)<>0 then B^i.x_change_per_scan:=(A^i.x-A^{i+1}.x)/(A^i.y-
A^{i+1}.y);
end;
m:=n-1;
if (a^1.x<>a^n.x)or(a^1.y<>a^n.y) then
begin m:=m+1;
  if A^n.y<A^1.y then
    begin B^n.y_top:=A^n.y;B^n.x:=A^n.x end
  else
    begin B^n.y_top:=A^1.y;B^n.x:=A^1.x end;
    B^n.delta_y:=abs(A^n.y-A^1.y);
    if (B^n.delta_y)<>0 then B^n.x_change_per_scan:=(A^n.x-A^1.x)/(A^n.y-
A^1.y);
  end;
end;
{*****}
procedure sortdscanh(m:integer;var B:dscanh);
var i,j:integer;
tam:dlcanh;
begin
for i:=1 to m-1 do
  for j:=i+1 to m do
    if B^i.y_top>B^j.y_top then
      begin tam:=B^i;B^i:=B^j;B^j:=tam end;
end;
{*****}
var B:dscanh;
j,top,bottom,m,y,tam,maxy:integer;
Gx:array[1..20] of diemcat;
tg:diemcat;
dem:byte;
ch:char;
trang,doan,rong:word;
l,i:longint;
begin
taodscanh(a,n,B,m);
maxy:=A^1.y;
for i:=1 to n do
  if maxy<A^i.y then maxy:=A^i.y;
sortdscanh(m,B);
for y:=B^1.y_top to maxy do
begin
tam:=0;
for i:=1 to m do
if (B^i.y_top<=y)and(y<=B^i.y_top+B^i.delta_y-1) then
begin tam:=tam+1;
  Gx[tam].x:=B^i.x+B^i.x_change_per_scan*(y-B^i.y_top);
  Gx[tam].canhthu:=i;
end;
{xawps keeps laij cacs giao ddieemr theo thws twj tawng daanf}

```

```
for i:=1 to tam-1 do
  for j:=i+1 to tam do
    if Gx[i].x>Gx[j].x then
      begin tg:=Gx[i];Gx[i]:=Gx[j];Gx[j]:=tg end;
    i:=0;
  {too cacs doanj nawmf trong dda giacs}
  while i<tam-1 do
    begin i:=i+1;
    if Gx[i].x=Gx[i+1].x then
      begin
        if (B^[Gx[i].canhthu].y_top<>B^[Gx[i+1].canhthu].y_top) then
          begin
            for j:=i to tam-1 do Gx[j]:=Gx[j+1];
            tam:=tam-1;
          end;
        end;
      end;
    end;
  i:=1;
  while i<tam do
    begin linengang(round(Gx[i].x),round(Gx[i+1].x),y);
      i:=i+2;
    end;
  end;
  Freemem(B,sizeof(Pointtype)*n);
end;
{***** Ket thuc Thu tuc FillPoly *****}
END.
```

### **III.2. Cơ chế hoạt động của màn hình theo chuẩn Vesa**

*Tham khảo tài liệu:*

1. VESA BIOS Extensions:

[http://en.wikipedia.org/wiki/VESA\\_BIOS\\_Extensions](http://en.wikipedia.org/wiki/VESA_BIOS_Extensions)

2. Vesa (Video Electronics Standards Association)

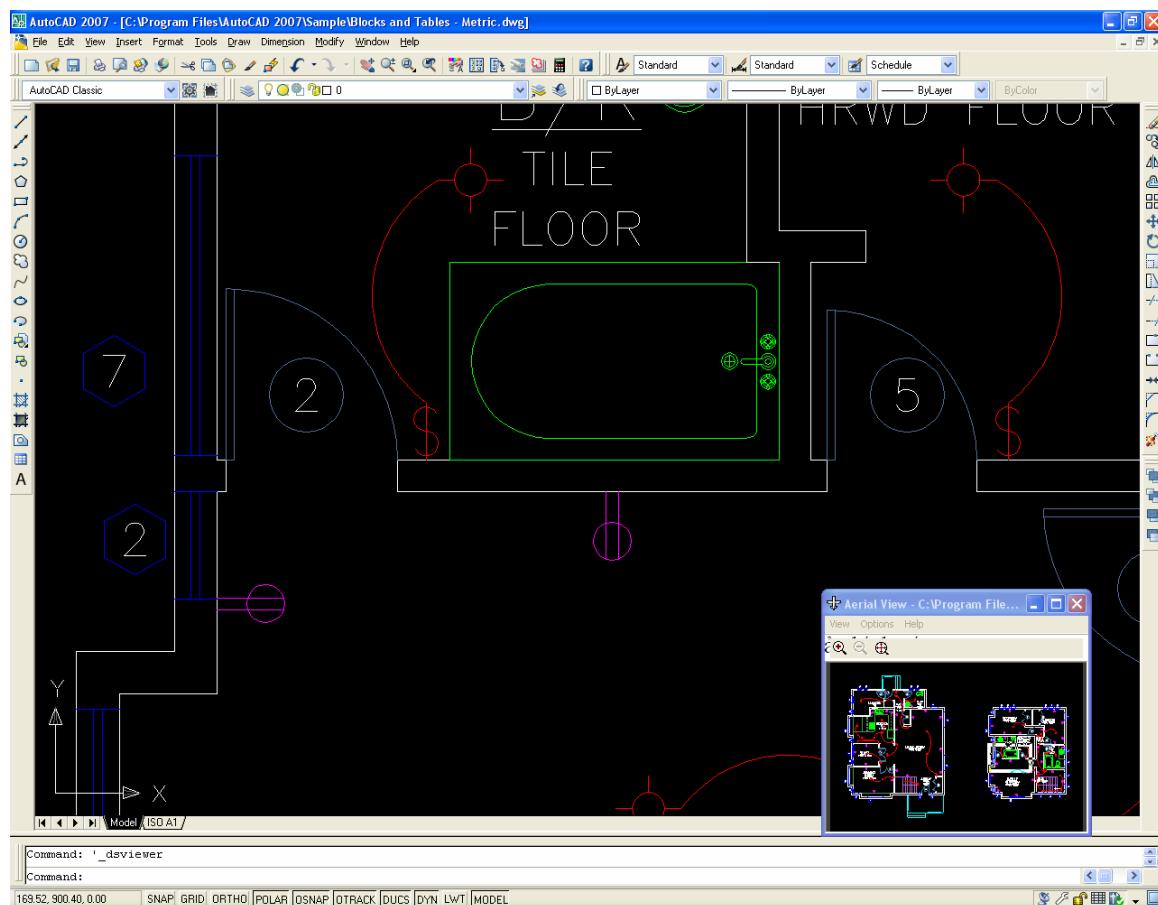
<http://www.vesa.org/public/VBE/vbe3.pdf>

3. Thư viện đồ họa hoạt động theo chuẩn Vesa: (*liên hệ tác giả*)

### Chương III: Các phép xén hình & tô màu



Trong kỹ thuật đồ họa máy tính rất nhiều bài toán đòi hỏi phải sử dụng các thuật toán xén hình (clipping). Các thuật toán xén hình nhằm cắt bỏ bớt các bộ phận của hình ảnh không nằm trong phạm vi cho trước nào đó.



Ví dụ như một bản thiết kế lớn cần xén vào một khung nhìn (viewport) cụ thể khi chúng ta thiết kế các chi tiết. Điều này giúp cho việc thể hiện bản vẽ được nhanh và hiệu quả.

Chúng ta có thể định nghĩa một cách tổng quát như sau:

**Định nghĩa:** Cho một miền  $D \subset R^n$  và  $F$  là một hình trong  $R^n$  (nghĩa là  $F \subset R^n$ ). Ta gọi  $F \cap D$  là hình có được từ  $F$  bằng cách cắt nó vào trong  $D$ . Ký hiệu  $F \cap D$  là  $\text{Clip}_D(F)$ .

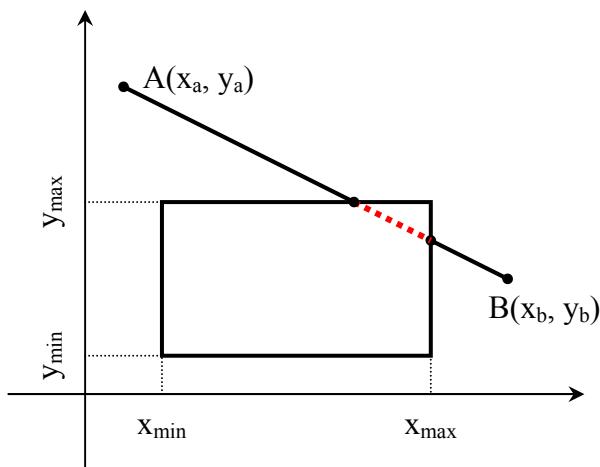
Trong từng ứng dụng cụ thể thì miền  $D$  và  $F$  sẽ được mô tả chính xác, chẳng hạn miền  $D$  là một vùng hình chữ nhật trong  $R^2$  và  $F$  là một đoạn thẳng hay một đường tròn trong  $R^2$ .

## I. Trường hợp $F$ là một tập hữu hạn điểm

Trong trường hợp này bài toán tương đương với việc tìm một thuật toán để xác định một điểm của  $F$  có nằm trong miền  $D$  hay không. Bài toán này đã được giải quyết khi  $D$  là một đa giác không tự cắt trong  $R^2$ , Khi  $D$  là một đa diện không tự cắt trong  $R^3$ , Khi  $D$  là hình tròn, hình cầu,...

## II. Trường hợp xén một đoạn thẳng vào một vùng hình chữ nhật của $R^2$ .

### II.1. Khi có một cạnh của hình chữ nhật song song với trục toạ độ



Khi đó hình chữ nhật  $D$  và đoạn thẳng  $F$  có thể được mô tả như sau:

$$D = \left\{ (x, y) \in R^2 \mid \begin{array}{l} x_{\min} \leq x \leq x_{\max} \\ y_{\min} \leq y \leq y_{\max} \end{array} \right\} \quad \text{và} \quad F = \left\{ (x, y) \in R^2 \mid \begin{array}{l} x = x_A + (x_B - x_A)t \\ y = y_A + (y_B - y_A)t \\ 0 \leq t \leq 1 \end{array} \right\}$$

Trong đó  $(x_{\min}, y_{\min}), (x_{\max}, y_{\max})$  là hai đỉnh xác định nên hình chữ nhật, và  $(x_A, y_A), (x_B, y_B)$  là toạ độ hai đầu mút của đoạn thẳng  $F$  kí hiệu là  $AB$ .

Trong trường hợp này  $\text{Clip}_D(F)$  có thể là rỗng hay là một đoạn thẳng (có thể suy biến thành 1 điểm). Theo định nghĩa thì thuật toán có thể mô tả như sau:

- ❖ **Bước 1:** Tìm nghiệm của hệ bất phương trình

$$\begin{cases} x_{\min} \leq x_A + (x_B - x_A)t \leq x_{\max} \\ y_{\min} \leq y_A + (y_B - y_A)t \leq y_{\max} \\ 0 \leq t \leq 1 \end{cases} \quad (\text{II.1})$$

Nghiệm của hệ này sẽ là  $[t_1, t_2]$  hay  $\emptyset$ , ta gọi  $N$  là tập các nghiệm của hệ

❖ **Bước 2:**

+ Nếu  $N = \emptyset$ : thì  $\text{Clip}_D(F) = \emptyset$

+ Ngược lại  $N = [t_1, t_2]$  (quy ước  $t_1 \leq t_2$ ). Gọi  $C, D$  là hai điểm xác định bởi:

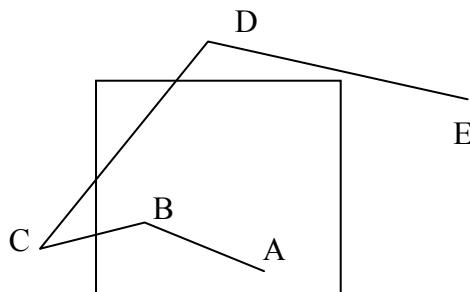
$$\begin{aligned} x_C &= x_A + (x_B - x_A)t_1; y_C = y_A + (y_B - y_A)t_1 \\ x_D &= x_A + (x_B - x_A)t_2; y_D = y_A + (y_B - y_A)t_2 \end{aligned}$$

Thì  $\text{Clip}_D(AB) = CD$

Thuật toán trên vừa đòi hỏi phải tính toán và lượng giá nhiều trên các số thực. Trong thực tế người ta dùng một thuật toán khác hiệu quả hơn. Ở đây chung ta sẽ xem xét đến các thuật toán như Cohen-Sutherland, thuật toán chia nhị phân, hay một thuật toán được cải biến và gọt giũa từ thuật toán tổng quát trên với tên gọi là thuật toán Liang-Barsky

### II.1.a. Thuật toán Cohen-Sutherland

Xét một cách tổng quát sẽ có các tình huống sau cho bài toán xén một đoạn thẳng vào trong một hình chữ nhật.



+ Trường hợp cả hai đầu mút của đoạn thẳng đều nằm trong hình chữ nhật (như đoạn AB) thì  $\text{Clip}_D(F)$  sẽ là đoạn thẳng đã cho.

+ Trường hợp có một đầu mút của đoạn thẳng nằm ngoài hình chữ nhật (như đoạn BC) thì  $\text{Clip}_D(F)$  sẽ là một phần đoạn thẳng đã cho.

+ Trường hợp cả hai đầu mút của đoạn thẳng nằm ở ngoài hình chữ nhật song có một phần của đoạn thẳng nằm trong hình chữ nhật thì đoạn thẳng F cắt hình chữ nhật D đúng hai điểm và khi đó  $\text{Clip}_D(F)$  sẽ là đoạn thẳng tạo bởi hai điểm đó.

+ Trường hợp cả hai đầu mút của đoạn thẳng nằm ở ngoài hình chữ nhật và không có phần nào của đoạn thẳng nằm trong hình chữ nhật (không có giao điểm với hình chữ nhật) thì  $\text{Clip}_D(F) = \emptyset$

Với các tình huống như trên giải thuật sẽ được tiếp cận theo hướng sau:  
Trước hết người ta đánh mã vùng cho không gian mặt phẳng đang xét để sau đó tiến hành thuật toán xén dựa trên cơ sở xét mã vùng của các điểm.

❖ Phương pháp đánh mã vùng: Mỗi vùng trong mặt phẳng Oxy được đánh một mã số tùy theo vị trí của chúng so với hình chữ nhật D như hình vẽ sau. Mỗi mã số có 4 bit, mỗi bit đóng vai trò như một cờ hiệu báo cho ta biết một trạng thái nào đó của vùng; cụ thể là: bit 0 chỉ được bật (bằng 1) khi vùng có mã đang xét nằm về bên Trái hình chữ nhật, bit 1 chỉ được bật khi vùng có mã đang xét nằm về bên Phải của hình chữ nhật, bit 2 được bật khi vùng nằm bên Dưới, và bit 3 được bật khi vùng nằm bên Trên.

1001	1000	1010
0001	0000	0010
0101	0100	0110

Bảng quy tắc đánh mã			
bít 3	bít 2	bít 1	bít 0
Trên	Dưới	Phải	Trái

Như vậy mặt phẳng OXY được chia làm 9 phần, mỗi phần được đánh một mã 4 bit biểu hiện trạng thái khác nhau của từng vùng.

Xét một điểm P ( $x_p, y_p$ ) trong mặt phẳng, ta gắn cho nó một mã trùng với mã của vùng chứa nó. Từ đó ta có thể suy ra cách xác định mã của P như sau:

$$\text{Bít 0} = \begin{cases} 1 & \text{nếu } x_p < x_{\min} \\ 0 & \text{nếu ngược lại} \end{cases}$$

$$\text{Bít 1} = \begin{cases} 1 & \text{nếu } x_p > x_{\max} \\ 0 & \text{nếu ngược lại} \end{cases}$$

$$\text{Bít 2} = \begin{cases} 1 & \text{nếu } y_p < y_{\min} \\ 0 & \text{nếu ngược lại} \end{cases}$$

$$\text{Bít 3} = \begin{cases} 1 & \text{nếu } y_p > y_{\max} \\ 0 & \text{nếu ngược lại} \end{cases}$$

Bây giờ xét một đoạn thẳng AB ta sẽ có các trường hợp sau:

i/ Đoạn AB nằm hoàn toàn bên trong hình chữ nhật nếu và chỉ nếu  $mã(A)=0000$  và  $mã(B)=0000$

ii/ Nếu  $[mã(A) \& mã(B)] \neq 0000$  thì đoạn AB nằm hoàn toàn bên ngoài hình chữ nhật (ở đây phép toán & được hiểu là phép AND lôgic của các bít tương ứng, ví dụ:  $1101 \& 1011 = 1001$ ). Thật vậy, từ giả thiết suy ra  $mã(A) \neq 0000$  và  $mã(B) \neq 0000$ , nghĩa là A và B cùng nằm bên ngoài hình chữ nhật, mặt khác do kết quả là khác 0000 có nghĩa là phải tồn tại ít nhất một bít tại vị trí nào đó bằng 1, giả sử đó là bít i, suy ra cả  $mã(A)$  và  $mã(B)$  đều có bít i có giá trị 1. Điều này có nghĩa là A và B nằm về cùng một phía nào đó của hình chữ nhật D. Cả A và B đều ở bên ngoài và nằm về cùng một phía của hình chữ nhật D nên hiển nhiên là đoạn AB không thể có phần chung với hình chữ nhật D.

iii/ Trường hợp  $[mã(A) \& mã(B)] = 0000$  và  $[mã(A) \neq 0000 \text{ hoặc } mã(B) \neq 0000]$

Ta có thể giả sử  $mã(A) \neq 0000$  (nếu cần có thể đổi vai trò của A và B để thoả mãn giả thiết trên), điều này tương đương với việc A nằm bên ngoài hình chữ nhật.

❖ Nếu  $x_A=x_B$  thì AB là một đoạn thẳng đứng, do đó ta suy ra ngay cách xác định  $\text{Clip}_D(AB)$

- Đặt điểm C là giao điểm của AB với hình D như sau:  $x_C=x_A$ 
  - Nếu  $A_{\text{Trên}}=1$  (hiểu rằng mã của A có bít 3 bằng 1) thì  $y_C=y_{\max}$
  - Ngược lại (tức  $A_{\text{Dưới}}=1$ ) thì  $y_C=y_{\min}$
- Đặt điểm D như sau:  $x_D=x_A$ 
  - Nếu  $mã(B)=0000$  thì  $y_D=y_B$
  - Ngược lại thì:  $y_D=y_{\max}$  nếu  $B_{\text{Trên}}=1$   
 $y_D=y_{\min}$  nếu  $B_{\text{Dưới}}=1$

➤  $\text{Clip}_D(AB)=CD$

❖ Khi  $x_A \neq x_B$ , Khi đó AB là một đoạn thẳng nằm xiên. Đặt  $C=A$ ,  $D=B$ . Bởi vì C nằm bên ngoài hình chữ nhật nên quy trình xén được thực hiện như sau:

- Nếu  $C_{\text{Trái}}=1$  thì có thể thay C bởi giao điểm của đoạn CD và **cạnh Trái nối dài** của hình chữ nhật D
- Nếu  $C_{\text{phải}}=1$  thì có thể thay C bởi giao điểm của đoạn CD và **cạnh Phải nối dài** của hình chữ nhật D
- Nếu  $C_{\text{Dưới}}=1$  thì có thể thay C bởi giao điểm của đoạn CD và **cạnh Dưới nối dài** của hình chữ nhật D

- Nếu  $C_{Trên} = 1$  thì có thể thay C bởi giao điểm của đoạn CD và **cạnh Trên nối dài** của hình chữ nhật D

Quá trình trên được lặp đi lặp lại, sau mỗi lần nhớ chú ý tính lại mã của C và đổi vai trò của C và D nếu cần để đảm bảo C luôn nằm bên ngoài hình chữ nhật. Quá trình trên sẽ dừng khi ta có các điểm CD rơi vào các trường hợp i/ hoặc ii/ tức tương ứng với  $Clip_D(AB) = CD$  hoặc  $Clip_D(AB) = \emptyset$ .

### **II.1.a.i. Kết thúc giải thuật**

- ❖ Hướng dẫn: Công thức tính giao điểm của CD với các cạnh Trái, Phải, Trên, Dưới có thể được suy ra từ phuong trình biểu diễn CD và các cạnh của hình chữ nhật:

$$\text{Phương trình AB: } \frac{y - y_A}{y_B - y_A} = \frac{x - x_A}{x_B - x_A} \text{ hay } \frac{y - y_A}{\Delta y} = \frac{x - x_A}{\Delta x} ;$$

Phương trình cạnh Trái  $x=x_{\min}$

Suy ra tọa độ giao điểm với cạnh trái nếu có là  $(x_{\min}, \frac{\Delta y}{\Delta x}(x_{\min} - x_A) + y_A)$ . Công thức tính tọa độ giao điểm với các cạnh còn lại được suy ra tương tự.

Việc tính mã có thể được thực hiện dễ dàng nhờ toán tử OR, với toán tử này cho phép ta dễ dàng bật các bit (cho bằng 1).

### **II.1.a.ii. Sau đây là một hàm tính mã**

{Vào : x,y chứa giá trị tọa độ của điểm P; xmin,ymin,xmax,ymax: biểu diễn tọa độ của hình chữ nhật D

Ra : Một byte có 4 bit đầu chứa mã của P và 4 bit sau bằng 0}

Function Ma(x,y,xmin,ymin,xmax,ymax:integer):Byte;

Var m:byte;

Begin

m:=0 {m=0000}

If  $x < xmin$  then  $m := m \text{ or } 1$ ; {Tức m or 0001 Bin, Đặt bit 0 bằng 1}

If  $x > xmax$  then  $m := m \text{ or } 2$ ; {Tức m or 0010 Bin, Đặt bit 1 bằng 1}

If  $y < ymin$  then  $m := m \text{ or } 4$ ; {Tức m or 0100 Bin, Đặt bit 2 bằng 1}

If  $y > ymax$  then  $m := m \text{ or } 8$ ; {Tức m or 1000 Bin, Đặt bit 3 bằng 1}

Ma:=m;

End;

#### **II.1.a.iii. Cài đặt thuật toán**

**Sinh viên cần xây dựng:**

+ Một thủ tục xén theo thuật toán Cohen-Sutherland

+ Một chương trình sử dụng thủ tục xén để minh họa các tình huống đã xét trong thuật toán.

#### **II.1.b. Thuật toán Chia nhị phân**

Thuật giải này xuất phát từ tư tưởng của phương pháp "giải phương trình bằng phương pháp chia nhị phân"

Tư tưởng của thuật giải như sau: lấy trung điểm của đoạn thẳng và kiểm tra mã của nó để loại dần các đoạn con không chứa giao điểm, và cuối cùng cho điểm giữa hội tụ về giao điểm của đoạn thẳng với hình chữ nhật, kết quả là ta thu được đoạn con nằm trong hình chữ nhật (nếu có)

Thuật toán được phác thảo như sau:

i/ Nếu  $mã(A)=0000$  và  $mã(B)=0000$  thì  $Clip_D(AB)=AB$

ii/ Nếu  $mã(A)=0000$  và  $mã(B) \neq 0000$  thì:

Đặt  $C:=A$ ;  $D:=B$ ;

while  $|x_C-x_D|+|y_C-y_D| \geq 1$  do

begin

lấy trung điểm M của CD

+ Nếu  $mã(M) \neq 0000$  thì  $D:=M$

+ Ngược lại:  $C:=M$

end;

$Clip_D(AB)=AC$

iii/ Nếu  $mã(A) \neq 0000$  và  $mã(B)=0000$ :

Đổi vai trò của A và B rồi áp dụng ii/

iv/ [Nếu  $mã(A) \& mã(B) \neq 0000$  thì  $Clip_D(AB)=\emptyset$ ]

v/ Nếu  $mã(A) \neq 0000$  và  $mã(B) \neq 0000$  và  $[mã(A) \& mã(B)]=0000$

Đặt  $C:=A$ ;  $D:=B$ ;

Lấy trung điểm M của CD

While ( $mā(M) \neq 0000$ )and( $|x_C - x_D| + |y_C - y_D| \geq 1$ ) do  
 begin  
     + Nếu  $mā(M) \& mā(C) \neq 0000$  thì  $C := M$  {đoạn CM ở ngoài theo iv/}  
     + Ngược lại (tức  $mā(M) \& mā(C) = 0000$ ) thì  $D := M$  {đoạn DM ở ngoài, tự chứng minh}  
         Lấy trung điểm M của CD  
     end;  
     + Nếu  $mā(M) \neq 0000$  thì  $\text{Clip}_D(AB) = \emptyset$   
     + Ngược lại: áp dụng ii/ cho các đoạn MA và MB ta được  
          $\text{Clip}_D(MA) = MA_1$   
          $\text{Clip}_D(MB) = MB_1$   
     Suy ra  $\text{Clip}_D(AB) = A_1B_1$

### **II.1.b.i. Cài đặt thuật toán**

**Sinh viên cần xây dựng:**

- + Một thủ tục xén theo thuật toán chia nhị phân
- + Một chương trình sử dụng thủ tục xén để minh họa các tình huống đã xét trong thuật toán.

### **II.1.c. Thuật toán Liang-Barsky**

Thuật toán này là một giải pháp cụ thể cho phương pháp chung đã đề cập ở phần đầu trong (mục II.1). Thuật toán này như sau:

Ta đặt	$\Delta x = x_B - x_A$	$\Delta y = y_B - y_A$
	$P_1 = -\Delta x$	$Q_1 = x_A - x_{\min}$
	$P_2 = \Delta x$	$Q_2 = x_{\max} - x_A$
	$P_3 = -\Delta y$	$Q_3 = y_A - y_{\min}$
	$P_4 = \Delta y$	$Q_4 = y_{\max} - y_A$

Thì hệ bất phương trình (II.1) có thể viết lại là:

$$\begin{cases} P_k t \leq Q_k; & k = 1, 2, 3, 4 \\ 0 \leq t \leq 1 \end{cases} \quad (\text{II.1.c})$$

Xét các trường hợp sau:

i/ Nếu  $\exists k \in \{1, 2, 3, 4\}$  sao cho ( $P_k = 0$  và  $Q_k < 0$ ) thì suy ra  $\text{Clip}_D(AB) = \emptyset$  (Vì hệ bất phương trình (II.1.c) vô nghiệm)

ii/  $\forall k \in \{1, 2, 3, 4\}$ ,  $P_k \neq 0$  hoặc  $Q_k \geq 0$ : thì khi đó các bất phương trình ứng với  $P_k = 0$  sẽ bị loại bỏ bởi vì chúng là hiển nhiên. Do vậy ta đặt

$$K_1 = \{k \mid P_k > 0\}$$

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

$$U_1 = \min \left( \left\{ \frac{Q_k}{P_k} \mid k \in K_1 \right\} \cup \{1\} \right)$$

$$U_2 = \max \left( \left\{ \frac{Q_k}{P_k} \mid k \in K_2 \right\} \cup \{0\} \right)$$

❖ Nếu  $U_1 < U_2$  thì  $\text{Clip}_D(AB) = \emptyset$

❖ Ngược lại: thì  $[U_2, U_1]$  chính là đoạn nghiệm của hệ bất phương trình trên (tương đương với  $[t_1, t_2]$  mà phương pháp tổng quát đã nêu). Nên gọi C,D là hai điểm thỏa

$$\begin{aligned} x_C &= x_A + \Delta x \cdot U_1; y_C = y_A + \Delta y \cdot U_1 \\ x_D &= x_A + \Delta x \cdot U_2; y_D = y_A + \Delta y \cdot U_2 \end{aligned}$$

Thì  $\text{Clip}_D(AB) = CD$

#### **II.1.c.i. Cài đặt thuật toán**

Sinh viên cần xây dựng:

- + Một thủ tục xén theo thuật toán trên Liang-Barsky
- + Một chương trình sử dụng thủ tục xén để minh họa các tình huống đã xét trong thuật toán.
- + Viết đánh giá so sánh ưu nhược điểm của 3 phương pháp xén hình trên.

#### **II.2. Khi 1 cạnh của hình chữ nhật tạo với trực hoành một góc $\alpha$ :**

Ta dùng phép quay hình để đưa bài toán về trường hợp xén đường thẳng vào hình chữ nhật có cạnh song song với trực toạ độ. Thuật toán được phác thảo như sau:

i/ Quay hình chữ nhật D và đường thẳng AB một góc  $-\alpha$  để được D' và A'B'.

ii/ Xác định  $\text{Clip}_{D'}(A'B')$

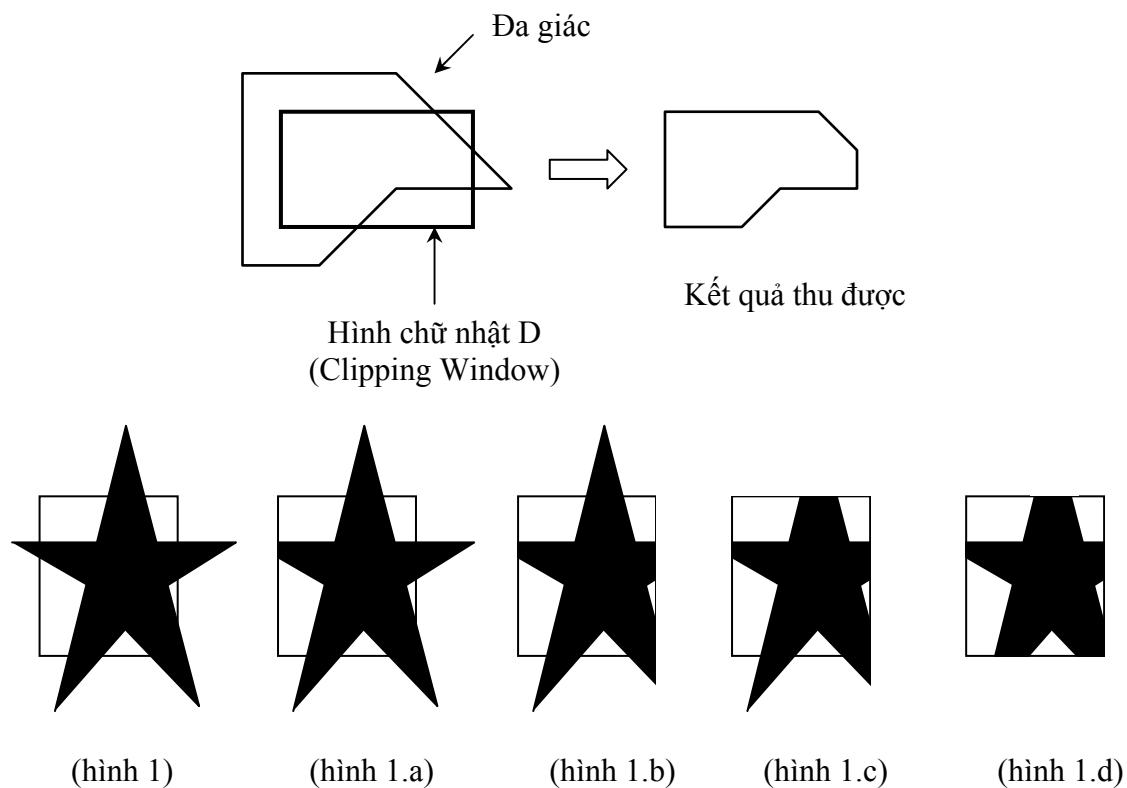
iii/ Sẽ có hai tình huống xảy ra:

- ❖ Nếu  $\text{Clip}_{D'}(A'B') = \emptyset$  thì kết quả  $\text{Clip}_D(AB) = \emptyset$ .
- ❖ Ngược lại nếu  $\text{Clip}_{D'}(A'B') = C'D'$  thì quay C'D' một góc  $\alpha$  ta được CD. Suy ra

$$\text{Clip}_D(AB) = CD$$

### III. Clipping một đa giác vào một vùng hình chữ nhật

#### III.1. Giải thuật Sutherland – Hodgman



*Tư tưởng của giải thuật như sau:*

Để Clipping một đa giác F vào trong một hình chữ nhật D ta tiến hành các bước sau:

**Bước 1:** Với đa giác F thực hiện cắt bỏ những phần nằm bên trái hình chữ nhật (nghĩa là bên trái của cạnh trái nối dài) ta thu được đa giác mới  $F_1$  (hình 1.a)

**Bước 2:** Với đa giác  $F_1$  thực hiện cắt bỏ những phần nằm bên phải hình chữ nhật ta thu được đa giác mới  $F_2$  (hình 1.b)

**Bước 3:** Với đa giác  $F_2$  thực hiện cắt bỏ những phần nằm bên trên hình chữ nhật ta thu được đa giác mới  $F_3$  (hình 1.c)

**Bước 4:** Với đa giác  $F_3$  thực hiện cắt bỏ những phần nằm bên dưới hình chữ nhật ta thu được đa giác mới  $F_4$  (hình 1.d)

**Kết quả:** Nếu  $F_4 = \emptyset$  thì  $\text{Clip}_D(F) = \emptyset$ . Ngược lại thì ta thu được kết quả xén là đa giác  $F_4$ , hay  $\text{Clip}_D(F) = F_4$

Để thực hiện các bước (1,2,3,4) ta sẽ tiến hành theo phương pháp sau:

**Chẳng hạn cho bước 1:** Xuất phát từ một đỉnh nào đó của đa giác, ta tiến hành đi dọc theo các cạnh đến các đỉnh khác cho đến khi về lại đỉnh đầu.

Trên quá trình di chuyển:

- ❖ Nếu gặp một đỉnh và đỉnh đó ở trên hay bên phải của cạnh trái (nối dài) hình chữ nhật thì ta lưu điểm đó vào  $F_1$ .
- ❖ Nếu gặp giao điểm với cạnh trái (hay cạnh trái nối dài) thì lưu giao điểm đó vào  $F_1$

Kết quả ta có  $F_1$  là một tập các điểm biểu diễn đa giác  $F$  khi đã xén đi mất phần bên trái.

Các bước còn lại thực hiện tương tự.

### **III.1.a. Cài đặt thuật toán**

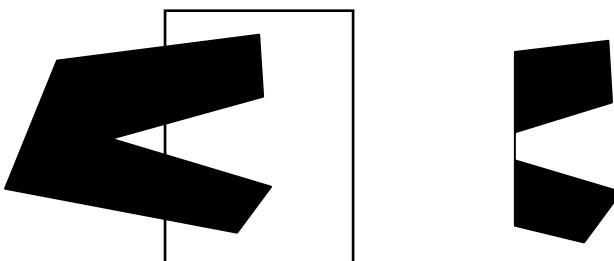
Sinh viên cần xây dựng:

- + Một thủ tục xén đa giác theo thuật toán trên
- + Một chương trình sử dụng thủ tục xén để minh họa

### **III.1.b. Nhược điểm thuật giải Sutherland-Hodgman và cách khắc phục**

Thuật giải xén đa giác Sutherland - Hodgeman còn mắc phải một nhược điểm đó là khi kết quả xén là 2 đa giác riêng biệt thì nó gộp lại làm một bởi một cạnh liên kết.

Ví dụ:

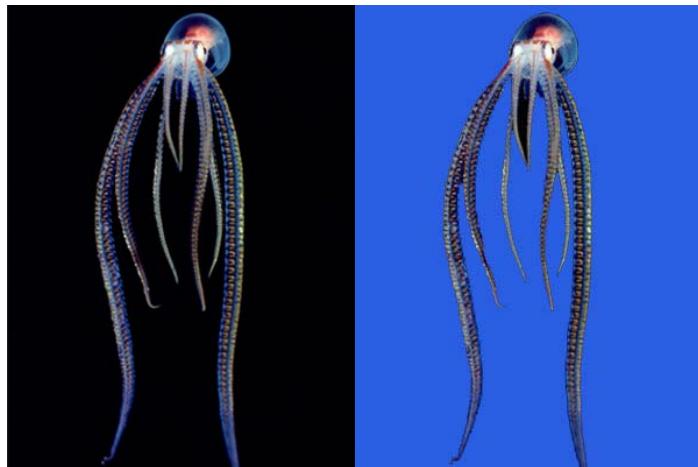


Phương pháp khắc phục: Sinh viên tự nghiên cứu.

## **IV. Một số thuật toán tô màu**

### **IV.1. Giải thuật vết dầu loang**

Thuật toán tô màu cho một vùng đồng nhất theo kiểu loang dầu thường được áp dụng để tô màu cho các vùng có đường biên phức tạp rối rắm, như tô màu cho một vùng ảnh, hay một biến thể của giải thuật là tìm một vùng liên thông đồng nhất màu (hay cùng tone màu) trên ảnh.



Ảnh gốc & Ảnh được tô nền xanh theo thuật toán vết dầu loang

### Bài toán tổng quát:

Cho một ma trận điểm M, mỗi điểm có tọa độ là  $(x,y)$  với  $x,y \in Z$  và có màu là  $\text{Color}(x,y)$ .

Từ một điểm  $P(x_0,y_0) \in M$  ta xác định một vùng liên thông đồng nhất (đồng màu) Q theo quy tắc sau:

- ❖ Đầu tiên tập Q chỉ có một phần tử đó là  $P(x_0,y_0)$
- ❖ Với mọi điểm  $T \in M$  sẽ được đưa vào Q nếu tồn tại trong Q một điểm K nào đó sao cho  $\text{Color}(K)=\text{Color}(T)$  và T là lân cận của K (*ở đây xét lân cận 4, có nghĩa là T ở sát trên hoặc sát dưới, hay sát trái hoặc sát phải của K*)

Biểu diễn về mặt tọa độ thì T là lân cận của K khi và chỉ khi:

$$|\text{ABS}(X_T-X_K)|+|\text{ABS}(Y_T-Y_K)|=1$$

Bài toán đặt ra là xuất phát từ một điểm  $P(x_0,y_0) \in M$  hãy xác định vùng Q và tô màu cho nó bởi một màu C nào đó (*dĩ nhiên là C phải khác màu hiện thời của vùng Q*).

Trong thực tế bài toán tô màu theo vết dầu loang được ứng dụng rất rộng rãi trong đồ họa để tô màu, để xác định vùng liên thông thỏa một thuộc tính nào đó. Hay tư tưởng và phương pháp của nó được ứng dụng cho: tìm một vùng liên thông khi biết trước 1 điểm, hay lọc ra từ tập hợp các phần tử cùng tính chất theo kiểu lần mắt xích quan hệ,...

Giải quyết bài toán này có hai phương pháp là đệ quy và phương pháp không đệ quy.

#### **IV.1.a. Phương pháp đệ quy**

Bước 1: Lưu giữ màu của điểm  $P(x_0,y_0)$  vào một biến:  $\text{OldColor}:=\text{Color}(P)$

Bước 2: Tô màu cho điểm  $P(x_0, y_0)$

Bước 3: Xác định các điểm lân cận  $T_i$  ( $i=1..4$ ) có màu OldColor (cùng màu với  $P$  trước khi tô)

Bước 4: Xem  $T_i$  như vai trò của  $P$  và thực hiện lại từ bước 1.

Kết luận: Thuật toán trên sẽ dừng khi không xác định được  $T_i$  ở tất cả các lần đệ quy.

#### **IV.1.a.i. Cài đặt thuật toán**

Sau đây là thủ tục minh họa cho thuật giải trên

{Vào: tọa độ của điểm  $P$  và màu cần tô cho vùng  $Q$

Ra : Không. Kết quả công việc là tô màu vùng  $Q$  trên màn hình}

Procedure LoangMauDeQuy(x,y:integer; NewColor:byte);

Var OldColor:byte;

Begin

    OldColor:=GetPixel(x,y)

    if OldColor=NewColor then

        exit

    else

        begin

            PutPixel(x,y,NewColor);

            If (x>0) and (GetPixel(x-1,y)=OldColor) then

                LoangMauDeQuy(x-1,y,NewColor);

            If (x<GetMaxX) and (GetPixel(x+1,y)=OldColor) then

                LoangMauDeQuy(x+1,y,NewColor);

            If (y>0) and (GetPixel(x,y-1)=OldColor) then

                LoangMauDeQuy(x,y-1,NewColor);

            If (y<GetMaxY) and (GetPixel(x,y+1)=OldColor) then

                LoangMauDeQuy(x,y+1,NewColor);

        end;

    End;

**Nhược điểm của phương pháp đệ quy là không thực hiện được khi vùng loang có diện tích lớn (dẫn đến tràn Stack).**

#### **IV.1.b. Phương pháp không đệ quy**

Trên cơ sở của ý tưởng trên song được xây dựng lại để tránh gọi đệ quy làm hạn chế không gian tó

**Bước 1:** Khởi tạo một hàng đợi Q với phần tử đầu tiên là P( $x_0, y_0$ ). Gọi OldColor là màu của điểm P (OldColor=Color(P))

**Bước 2:** Khi hàng đợi không rỗng thì: lấy ra từ hàng đợi Q một điểm T.

Nếu màu hiện thời của T là OldColor thì:

+ Tô màu điểm T

+ Tìm các điểm lân cận của T có màu là OldColor và đưa chúng vào hàng đợi Q.

Bước 2 này được lặp đi lặp lại cho đến khi hàng đợi Q rỗng.

Thuật toán trên có một nhược điểm là đôi khi một điểm được đưa vào hàng đợi nhiều hơn một lần, từ đó ta có thể tinh chỉnh ở chỗ các điểm lân cận cùng màu chỉ được đưa vào khi nó chưa có trong hàng đợi. Song việc kiểm tra này sẽ dẫn đến một sự hao phí về thời gian thực hiện kiểm tra hàng đợi, vậy nên ta có thể bỏ qua.

##### **IV.1.b.i. Cài đặt thuật toán**

❖ Sinh viên cần xây dựng thủ tục tô màu theo thuật giải “Vết dầu loang” sử dụng hàng đợi đệ khử đệ quy như đã nêu ở trên.  
Sau đây là một giải pháp cài đặt minh họa:

```
program Chuong_Trinh_To_Mau_Theo_Thuat_Toan_Vet_Dau_Loang;
uses crt,graph;
procedure init;
var
  grDriver: Integer;
  grMode: Integer;
  ErrCode: Integer;
begin
  grDriver := Detect;
  InitGraph(grDriver, grMode, '');
  ErrCode := GraphResult;
  if ErrCode <> grOk then
    Writeln('Graphics error:', GraphErrorMsg(ErrCode));
end;
procedure loangmau(x,y:integer;maumoi:byte);
type DS=^danh sach;
```

```

danh sach=record
    x,y:integer;
    next:DS;
    end;
var dau,cuoi:ds;maucu:byte;
Procedure Push(x,y:integer);
Var P:DS;
begin
    New(P);P^.x:=x;P^.y:=y;P^.next:=nil;
    If Dau=nil then
        Dau:=P
    else
        Cuoi^.next:=P;
        Cuoi:=p;
    end;
Procedure Pop(Var x,y:integer);
Var P:DS;
begin
    P:=Dau; Dau:=Dau^.next;
    x:=P^.x; y:=P^.y;
    dispose(p);
end;
begin
    maucu:=getpixel(x,y);
    if maucu=maumoi then exit;
    Dau:=Nil;Push(x,y);

    while (dau<>nil) do
    begin
        Pop(x,y);
        if (getpixel(x,y)=maucu)and(maxavail>sizeof(ds)) then
        begin
            putpixel(x,y,maumoi);
            if (x>0)and(getpixel(x-1,y)=maucu) then
                Push(x-1,y);
            if (x<getmaxx)and(getpixel(x+1,y)=maucu) then
                Push(x+1,y);
            if (y>0)and(getpixel(x,y-1)=maucu) then
                Push(x,y-1);
            if (y<getmaxy)and(getpixel(x,y+1)=maucu) then
                Push(x,y+1);
        end;
    end;
    BEGIN
    init;
    rectangle(50,50,480,380);
    line(50,50,155,370);line(155,370,165,55);

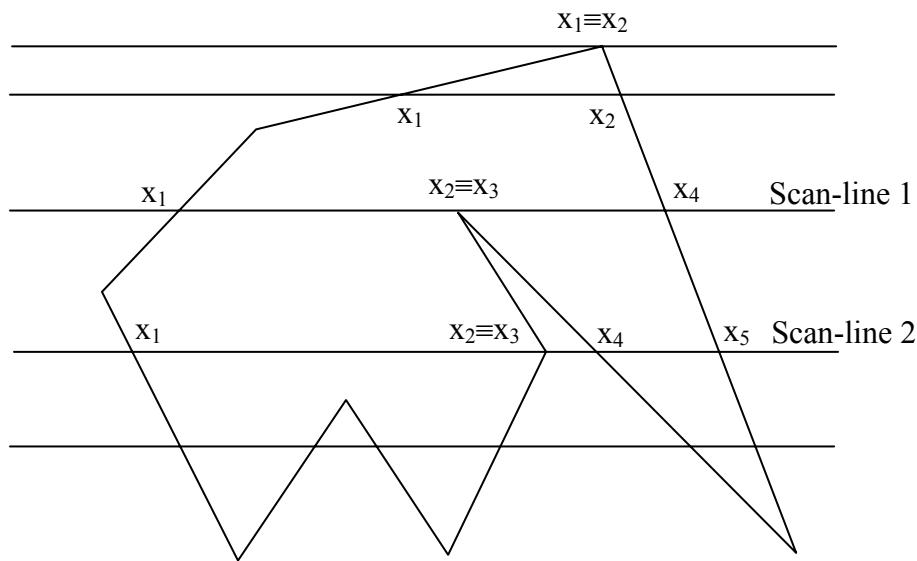
```

```

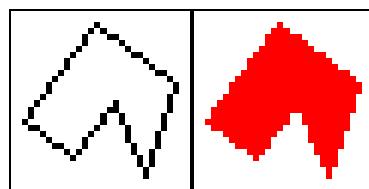
loangmau(60,60,4);
readln;
closegraph;
END.
```

#### IV.2. Thuật toán tô màu đa giác theo dòng quét (Scan-line Algorithm).

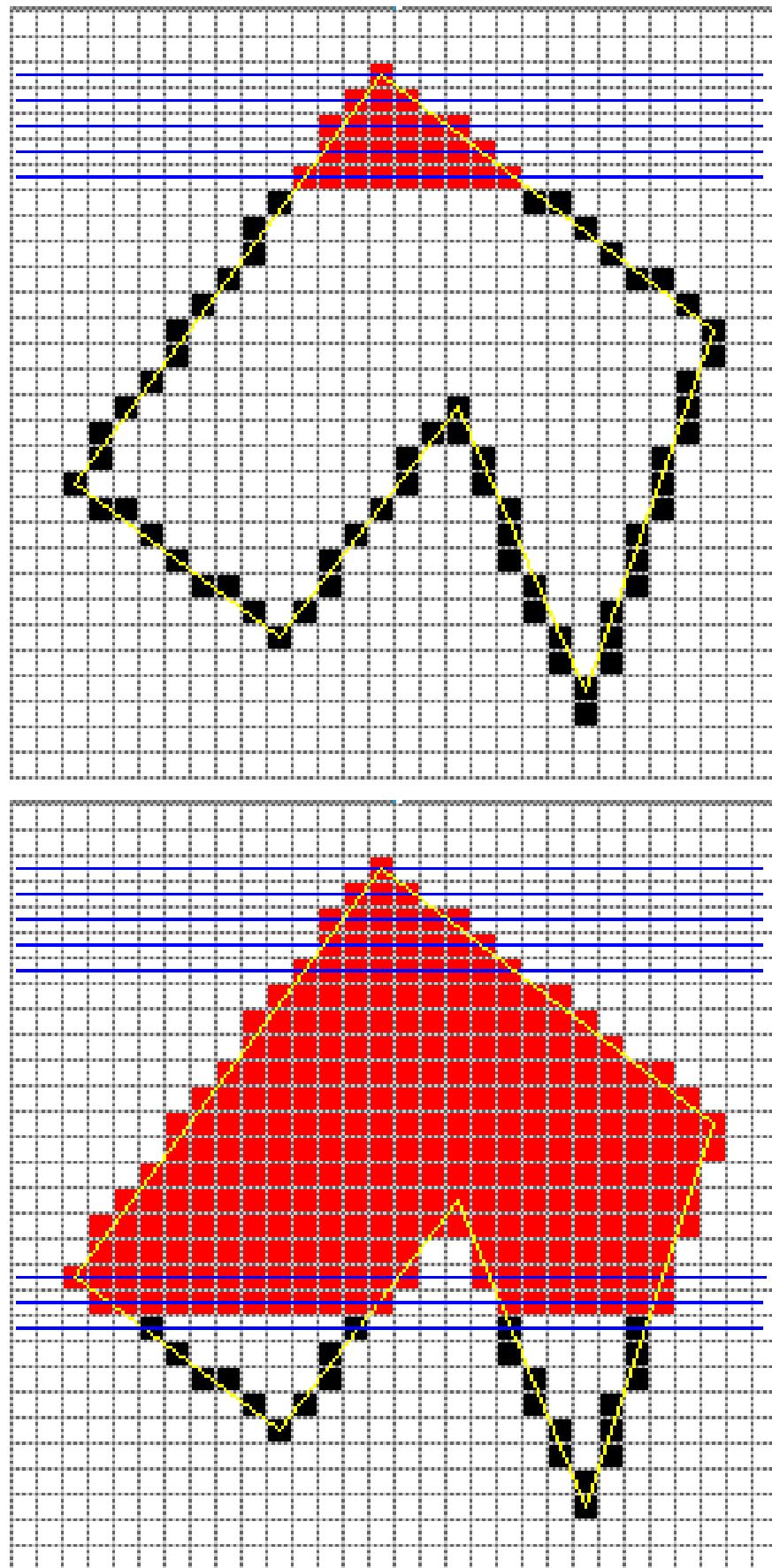
Thuật toán này sử dụng giao điểm giữa các cạnh biên của đa giác với các đường thẳng nằm ngang gọi là dòng quét di chuyển từ trên xuống dưới để xác định ra các đoạn con nằm trong đa giác và tô màu cho chúng.



Ví dụ:



(Đa giác trước và sau khi được tô)



(Hình ảnh phóng lớn mô tả quá trình tô màu đa giác)

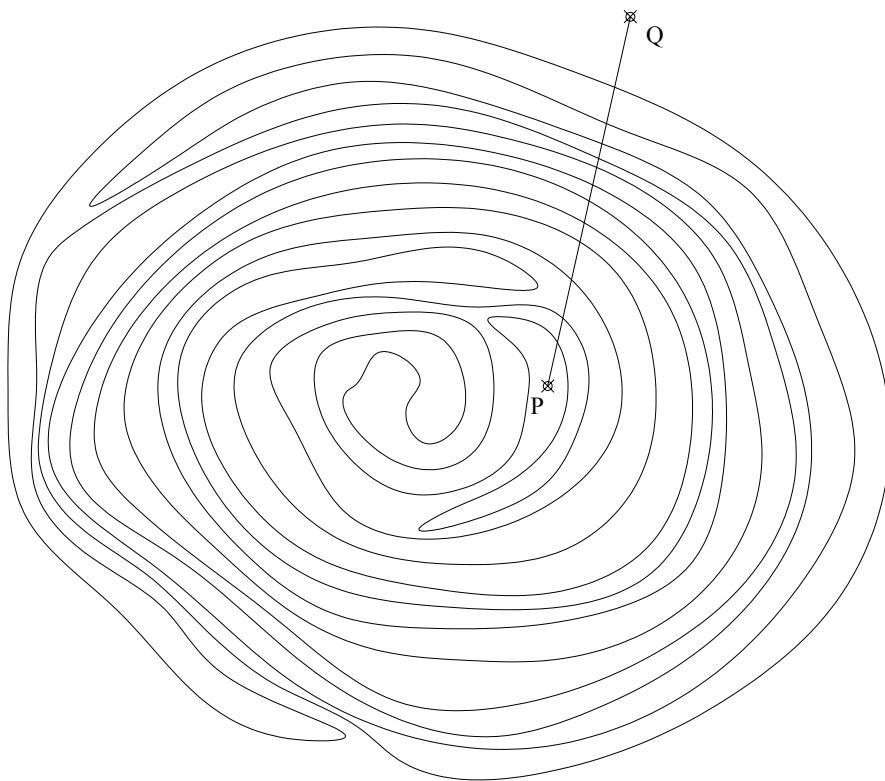
Để hiểu rõ hơn tư tưởng của giải thuật ta hãy tìm hiểu một ví dụ sau:

Cho miền D được bao bởi một đường cong kín không tự cắt G và một điểm P như trong hình vẽ, cần xác định điểm P là trong hay ngoài miền D?

Để trả lời câu hỏi này ta tiến hành như sau:

- + Xác định một điểm Q nào đó ở ngoài miền D
- + Tìm số giao điểm của đoạn thẳng PQ với G, nếu số giao điểm này là chẵn thì P không thuộc D, ngược lại thì P nằm trong miền D khi số giao điểm là lẻ.

Sở dĩ có kết luận như vậy là nhờ vào một bài toán vui cổ điển như sau:



Miền D xem như được bao bọc kín bởi tường thành G. Nếu một con kiến xuất phát từ Q và bò dọc theo đoạn thẳngQP để hướng đến P. Rõ ràng lúc đầu nó ở ngoài thành (ngoài miền D), sau lần vượt thành đầu tiên (qua giao điểm) thì nó sẽ vào trong, nếu nó vượt thành lần nữa thì rõ ràng là nó ra khỏi thành, nếu nó lại tiếp tục vượt thành thì nó lại vào trong, rồi lại vượt để ra ngoài... Rõ ràng ta thấy nên số lần vượt là lẻ thì nó ở trong thành, ngược lại nếu số lần vượt là chẵn thì nó ở ngoài thành.

Hay nhìn ở góc độ khác thì ta có các đoạn thẳng nằm trong hình chữ nhật là các đoạn được tạo bởi cặp giao điểm  $x_k$  với  $x_{k+1}$  với  $k=1,3,5,7\dots$

Từ bài toán trên ta thấy để tô màu đa giác ta chỉ việc cho dòng quét (Scan-line) chạy từ đỉnh cao nhất của đa giác đến đỉnh thấp nhất của đa giác, trong mỗi lần chạy ta xác định số giao điểm của đường chạy ngang với đa giác, các giao điểm được sắp theo thứ tự tăng dần theo hoành độ gọi là  $x_1, x_2, \dots, x_{2n}$  ( $n > 0$ ). và ta chỉ việc tô các đoạn  $x_1x_2, x_3x_4, \dots, x_{2n-1}x_{2n}$ .

Song để tìm giao điểm của dòng quét với đa giác ta phải tìm giao điểm với các cạnh của đa giác. Do đó điều tất yếu là tại những đỉnh sẽ có 2 giao điểm (giao điểm kép) và nó sẽ làm cho quy tắc tô trên sai nếu hai cạnh xuất phát từ đỉnh này nằm về hai phía của đường quét. Vậy trong tình huống dòng quét đi qua đỉnh thì nếu hai cạnh xuất phát từ đỉnh này nằm về hai phía của đường quét thì ta cần bỏ bớt đi một giao điểm.

#### **IV.2.a. Cài đặt thuật toán**

- ❖ Sinh viên cần viết một thủ tục tô màu cho một đa giác bất kỳ (với đầu vào là một mảng các đỉnh của đa giác)

### **V. Bài tập cuối chương**

1. Cài đặt thủ tục xén đoạn thẳng vào đa giác, xén đoạn thẳng vào Ellipse, Xén Ellipse vào hình chữ nhật.
2. Cài đặt thủ tục tô màu đa giác theo mẫu tô (tô có hoa văn)
  - a. Khi mẫu là một ma trận  $8 \times 8$  bít, nếu bít = 1 thì được tô ngược lại thì không tô
  - b. Khi mẫu tô là một ảnh bitmap kích thước  $m \times n$
3. Bổ sung vào thư viện do bạn cài đặt được ở chương trước các thủ tục xén và tô màu

# Chương IV: Các phép biến đổi hình học



## I. Các phép biến đổi Affine 2D (2- chiều)

Phép biến đổi Affine 2D sẽ biến điểm  $P(P_x, P_y)$  thành điểm  $Q(Q_x, Q_y)$  theo hệ phương trình sau:

$$Q_x = a.P_x + c.P_y + tr_x$$

$$Q_y = b.P_x + d.P_y + try$$

Dưới dạng ma trận, hệ này có dạng:

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

Hay viết gọn hơn:  $Q = P.M + Tr$  (I)

với  $Q=(Q_x, Q_y); P=(P_x, P_y);$

$Tr=(tr_x, tr_y)$  - vector tịnh tiến;

$$M = \begin{pmatrix} a & b \\ c & d \end{pmatrix}$$
 - ma trận biến đổi.

### I.1. Phép tịnh tiến

Phép tịnh tiến điểm  $P(P_x, P_y)$  thành điểm  $Q(Q_x, Q_y)$  theo Vector  $Tr = (tr_x, try)$  có hệ phương trình:

$$Q_x = P_x + tr_x$$

$$Q_y = P_y + try$$

hay theo dạng ma trận (I) thì

$$M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \text{ và } Tr = (tr_x, try)$$

## I.2. Phép đồng dạng

ở dạng phương trình là:

$$\begin{cases} Q_x = s_x P_x \\ Q_y = s_y P_y \end{cases}$$

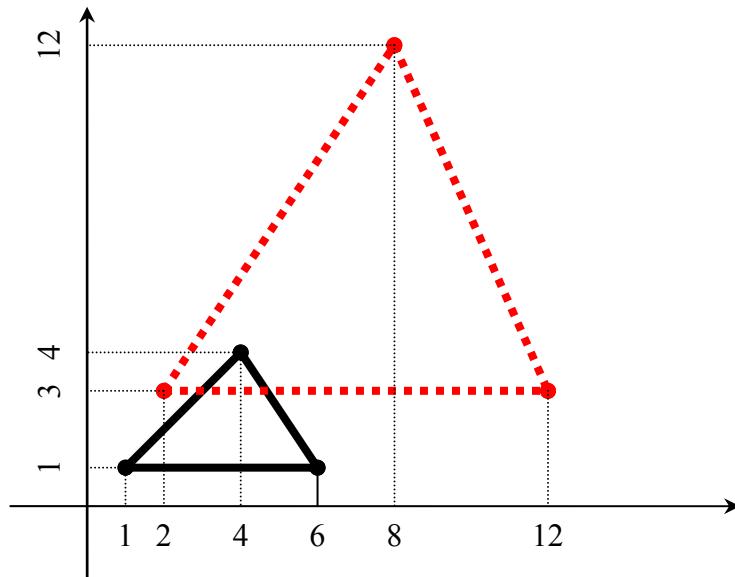
Dưới dạng ma trận:

$$M = \begin{pmatrix} s_x & 0 \\ 0 & s_y \end{pmatrix} \text{ và } Tr = (0,0)$$

Với:  $s_x$  là hệ số tỷ lệ theo trục x

$s_y$  là hệ số tỷ lệ theo trục y

Phép đồng dạng (hay còn gọi là phép biến đổi tỷ lệ) cho phép chúng ta phóng to hay thu nhỏ hình theo một hay hai chiều mà vẫn giữ được hình dáng cơ bản của chúng (đồng dạng).



(Hình minh họa phép biến đổi đồng dạng cho một tam giác)

## I.3. Phép đối xứng

Đây là trường hợp đặc biệt của phép đồng dạng với a và d đối nhau.

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

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

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

#### I.4. Phép quay quanh gốc tọa độ

Ma trận tổng quát của phép quay quanh gốc tọa độ một góc  $\alpha$  là:

$$M = \begin{pmatrix} \cos(\alpha) & \sin(\alpha) \\ -\sin(\alpha) & \cos(\alpha) \end{pmatrix} \text{ và } Tr = (0,0)$$

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.

#### I.5. Phép biến dạng (*Twist Transformation*)

Ma trận tổng quát là:  $M = \begin{pmatrix} 1 & t_y \\ t_x & 1 \end{pmatrix}$  và  $Tr = (0,0)$

Trong đó:

$t_x$ : là hệ số biến dạng theo trục x.

$t_y$ : là hệ số biến dạng theo trục y.

#### I.6. Toạ độ thuần nhất (Homogeneous Coordinates)

Để chỉ còn phép nhân ma trận trong các phép biến đổi, ta thêm vào một thành phần trong các ma trận và đưa nó về dạng

$$(Q_x, Q_y, 1) = (P_x, P_y, 1) \cdot \begin{pmatrix} a & b & 0 \\ c & d & 0 \\ tr_x & tr_y & 1 \end{pmatrix}$$

Ký hiệu  $(Q_x, Q_y, 1), (P_x, P_y, 1)$  được gọi là toạ độ Homogen, và công thức của phép biến đổi theo toạ độ Homogen là:

$$Q = P \cdot T$$

Với  $T$  là ma trận  $3 \times 3$  được gọi là ma trận biến đổi.

#### I.7. Tổng hợp các phép biến đổi Affine

Gọi  $f_1$  và  $f_2$  là 2 phép biến đổi như sau:

$$f_1 : P \rightarrow Q$$

$$f_2 : Q \rightarrow W$$

Ta cần tìm phép biến đổi  $P \rightarrow W$

Giả sử  $Q = P \cdot M_1 + Tr_1$  và  $W = Q \cdot M_2 + Tr_2$  thì:

$$\begin{aligned} W &= (P \cdot M_1 + Tr_1) \cdot M_2 + Tr_2 \\ &= P \cdot M_1 \cdot M_2 + Tr_1 \cdot M_2 + Tr_2 \end{aligned}$$

Đặt  $M = M_1 \cdot M_2$  và  $Tr = Tr_1 \cdot M_2 + Tr_2$  thì:

$$W = P \cdot M + Tr$$

Như vậy tổng hợp (hay tích) của hai phép biến đổi Affine cũng là một phép biến đổi Affine, có các ma trận biến đổi là:  $M = M_1 \cdot M_2$  và  $Tr = Tr_1 \cdot M_2 + Tr_2$

❖ Nếu sử dụng tọa độ Homogen thì:

$$Q = P \cdot T_1; \quad W = Q \cdot T_2$$

$$\text{Suy ra: } W = (P \cdot T_1) \cdot T_2 = P \cdot T \quad \text{với } T = T_1 \cdot T_2$$

Như vậy tổng hợp (hay tích) của hai phép biến đổi Homogen cũng là một phép biến đổi Homogen, có ma trận biến đổi là:  $T = T_1 \cdot T_2$

Lập luận tương tự ta có: Tổng hợp của  $N$  phép biến đổi Homogen cũng là một phép biến đổi Homogen, có ma trận biến đổi là:  $T = T_1 \times T_2 \times \dots \times T_N$

❖ **Chú ý:** Từ đây trở đi ta chỉ sử dụng tọa độ homogen trong các phép biến đổi để việc tìm tích của các phép biến đổi được trở nên đơn giản

Sau đây là ma trận biến đổi theo tọa độ Homogen của một số phép biến đổi cơ bản:

Phép biến đổi	Ma trận Homogeneous
phép tịnh tiến	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ tr_x & tr_y & 1 \end{pmatrix}$
phép quay	$\begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Phép đối xứng qua trục OX	$\begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
Phép đối xứng qua trục OY	$\begin{pmatrix} -1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

Phép đối xứng qua gốc tọa độ	$\begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$
phép đồng dạng	$\begin{pmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{pmatrix}$
phép biến dạng	$\begin{pmatrix} 1 & t_y & 0 \\ t_x & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

### I.8. Phép quay quanh điểm bất kỳ

Giả sử cho điểm P quay quanh điểm V( $V_x, V_y$ ) ta được ảnh của P qua phép biến đổi là Q.

Phép quay này bao gồm các phép biến đổi cơ bản sau:

- Tịnh tiến P theo vector  $\vec{VO} = (-V_x, -V_y)$  ta được  $P'$
- Quay  $P'$  quanh gốc tọa độ một góc  $\alpha$  ta được  $Q'$
- Tịnh tiến  $Q'$  theo vector  $\vec{OV} = (V_x, V_y)$  ta được Q.

Q chính là ảnh của P qua phép quay P quanh điểm V một góc  $\alpha$ . Ma trận biến đổi là:

$$T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -V_x & -V_y & 1 \end{pmatrix} \cdot \begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \cdot \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ V_x & V_y & 1 \end{pmatrix}$$

### I.9. Các ví dụ minh họa

#### I.9.a. Ví dụ 1:

Cho tam giác ABC có tọa độ lần lượt là A(0,0), B(6,0) và C(3,5). Quay tam giác ABC một góc 90° quanh điểm M(3,3) ta được tam giác  $A_1B_1C_1$ . Hãy tìm ma trận của phép biến đổi và tọa độ của  $A_1B_1C_1$ .

Giải: Phép quay quanh điểm M một góc 90° được biểu diễn qua 3 phép biến đổi cơ bản lần lượt là:

- + Tịnh tiến theo vector  $\vec{MO} = (x_O - x_M, y_O - y_M) = (0-3, 0-3) = (-3, -3)$ :

$$\text{Ma trận biểu diễn là } T_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & -3 & 1 \end{pmatrix}$$

+ Quay quanh gốc tọa độ một góc  $90^\circ$

$$\text{Ma trận biểu diễn là } T_2 = \begin{pmatrix} \cos(90^\circ) & \sin(90^\circ) & 0 \\ -\sin(90^\circ) & \cos(90^\circ) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

+ Tịnh tiến ngược trở lại theo vector  $\overrightarrow{OM} = (3, 3)$

$$\text{Ma trận biểu diễn là } T_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 3 & 1 \end{pmatrix}$$

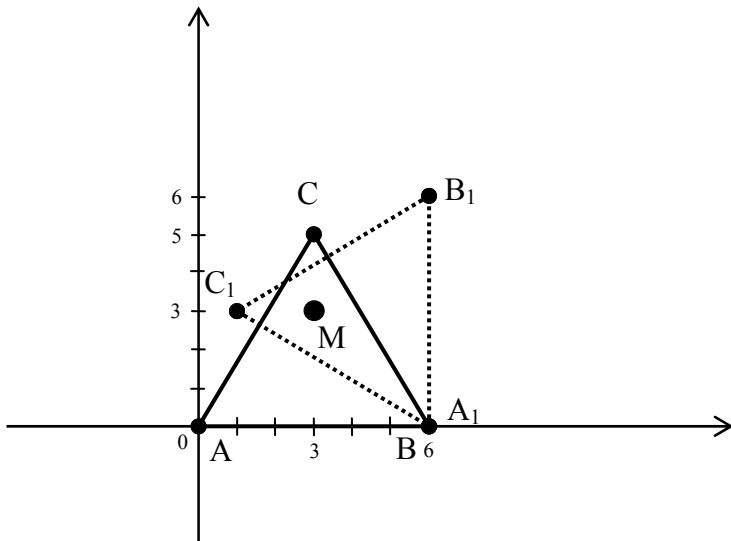
Như vậy, ma trận của phép quay quanh điểm M một góc  $90^\circ$  sẽ được tính bằng tích các ma trận biến đổi thành phần:

$$T = T_1 \cdot T_2 \cdot T_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -3 & -3 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 3 & -3 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 3 & 3 & 1 \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 6 & 0 & 1 \end{pmatrix}$$

$$A_1 B_1 C_1 \text{ (tọa độ homogen)} = ABC \text{ (tọa độ homogen)} \cdot T = \begin{pmatrix} 0 & 0 & 1 \\ 6 & 0 & 1 \\ 3 & 5 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 6 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 6 & 0 & 1 \\ 6 & 6 & 1 \\ 1 & 3 & 1 \end{pmatrix}$$

Vậy  $A_1(6, 0)$   $B_1(6, 6)$  và  $C_1(1, 3)$ .

Vẽ hình minh họa:



### I.9.b. Ví dụ 2:

Cho tứ giác ABCD với tọa độ lần lượt là  $A(3,0)$   $B(7,0)$   $C(6,4)$   $D(4,4)$ .

- Lấy đối xứng ABCD qua đường thẳng đi qua 2 điểm CD, rồi tiếp đến là quay một góc  $-90^\circ$  ta được tứ giác  $A_1B_1C_1D_1$ . Hãy tìm ma trận của phép biến đổi tổng hợp (biến đổi ABCD thành  $A_1B_1C_1D_1$ ) và tìm tọa độ của  $A_1B_1C_1D_1$ .
- Lấy đối xứng ABCD qua đường thẳng đi qua 2 điểm OC ( $O$  là gốc tọa độ) ta được  $A_2B_2C_2D_2$ . Tìm ma trận biến đổi và tọa độ của  $A_2B_2C_2D_2$ .

Giải:

- a) Phép lấy đối xứng qua CD được phân tích thành tích của 3 phép biến đổi cơ bản sau:

1. Phép tịnh tiến theo vector  $v_1 = (0, -4)$  (tịnh tiến CD về trùng với trục OX)

$$\text{Ma trận } T_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & -4 & 1 \end{pmatrix}$$

2. Phép lấy đối xứng qua trục OX

$$\text{Ma trận } T_2 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

3. Phép tịnh tiến ngược trở lại theo vector  $v_2 = -v_1 = (0, 4)$

$$\text{Ma trận } T_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 4 & 1 \end{pmatrix}$$

Vậy ma trận của phép lấy đối xứng qua CD là:

$$T_4 = T_1 x T_2 x T_3 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 8 & 1 \end{pmatrix}$$

Tiếp đến phép quay một góc  $-90^\circ$  có ma trận là:

$$T_5 = \begin{pmatrix} \cos(-90^\circ) & \sin(-90^\circ) & 0 \\ -\sin(-90^\circ) & \cos(-90^\circ) & 0 \\ 0 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Phép biến đổi tổng hợp biến đổi ABCD thành  $A_1B_1C_1D_1$  là tổng hợp của 2 phép biến đổi thành phần, gồm phép lấy đối xứng qua CD và phép quay một góc  $-90^\circ$ .

Vậy ma trận bằng tích của 2 ma trận biến đổi thành phần:

$$T_a = T_4 x T_5 = \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 8 & 0 & 1 \end{pmatrix}$$

$A_1B_1C_1D_1$  tọa độ homogen =  $ABCD$  tọa độ homogen  $T_a$

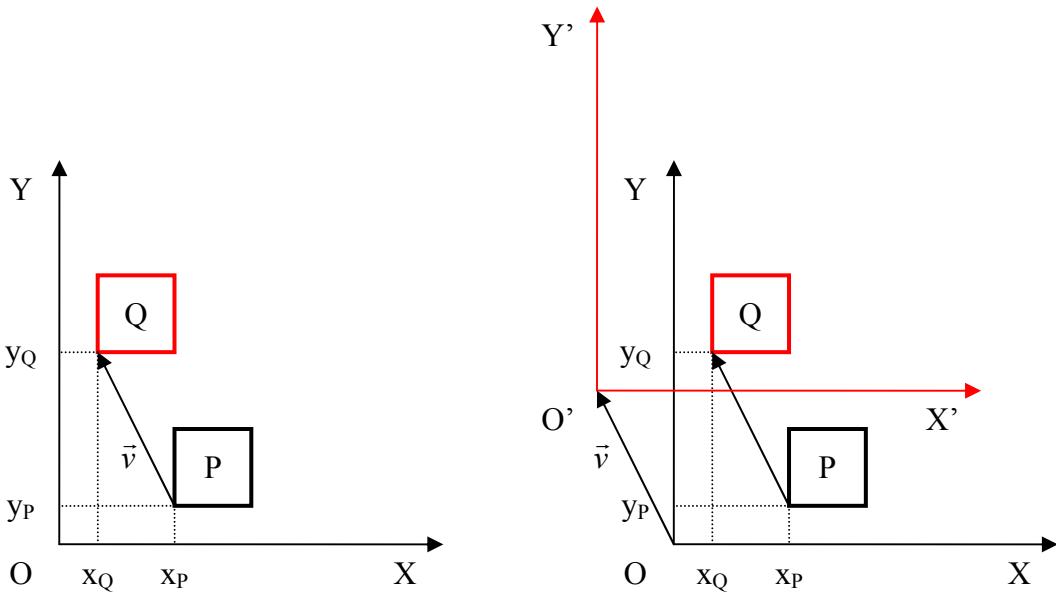
$$= \begin{pmatrix} 3 & 0 & 1 \\ 7 & 0 & 1 \\ 6 & 4 & 1 \\ 4 & 4 & 1 \end{pmatrix} x \begin{pmatrix} 0 & -1 & 0 \\ -1 & 0 & 0 \\ 8 & 0 & 1 \end{pmatrix} = \begin{pmatrix} 8 & -3 & 1 \\ 8 & -7 & 1 \\ 4 & -6 & 1 \\ 4 & -4 & 1 \end{pmatrix}$$

Vậy  $A_1 = (8, -3)$   $B_1 = (8, -7)$   $C_1 = (4, -6)$   $D_1 = (4, -4)$

- b) Sinh viên tự giải (xem như bài tập rèn luyện)

## I.10. Biến đổi hệ trực tọa độ (hay biến đổi ngược)

Xét ví dụ biến đổi **tịnh tiến** hình vuông P theo vector  $v(1, 2)$  thành hình vuông Q được minh họa như hình vẽ sau:



Tịnh tiến object

Tịnh tiến hệ trục tọa độ

Ta thấy khi tịnh tiến hình P theo vector  $\vec{v}$  ta được hình Q, đây được gọi là phép biến đổi thuận. Ngược lại, khi ta tịnh tiến hệ trục tọa độ OXY theo vector  $\vec{v}$  ta được hệ trục tọa độ mới O'X'Y'. Lúc này, tọa độ của hình P trong hệ trục OXY tương đương tọa độ của hình Q trong hệ tọa độ O'X'Y':

$$\text{Hay } P_{OXY} = Q_{O'X'Y'} \quad (a)$$

$$\text{mà } Q_{OXY} = P_{OXY} T \quad \text{với } T = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ v_x & v_y & 1 \end{pmatrix}$$

$$\text{suy ra: } Q_{OXY} \cdot T^{-1} = P_{OXY} \cdot T \cdot T^{-1} = P_{OXY} \quad (b)$$

từ (a) và (b) suy ra:

$$Q_{O'X'Y'} = Q_{OXY} \cdot T^{-1} \quad \text{với } T^{-1} = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ -v_x & -v_y & 1 \end{pmatrix}$$

Hay nói cách khác: là một điểm Q có tọa độ  $(x_Q, y_Q)$  trong hệ tọa độ OXY sẽ có tọa độ là  $(x'_Q, y'_Q)$  trong hệ tọa độ O'X'Y' và  
 $(x'_Q, y'_Q, 1) = (x_Q, y_Q, 1) \cdot T^{-1}$

Một cách tổng quát:

- Giả sử gọi  $f_1$  là một phép biến đổi Homogen biến điểm P thành điểm Q với ma trận biến đổi là  $T_{\text{thuận}}$ .
- Cũng sử dụng phép biến đổi  $f_1$  nhưng lần này là để biến đổi hệ trục OXY thành hệ trục O'X'Y'. Thì ma trận biến đổi sẽ là  $T_{\text{nghịch}} = T^{-1}$ .

Và lúc đó một điểm P trong hệ trục OXY sẽ có giá trị tọa độ trong hệ trục O'X'Y' là:

$$P' = P \cdot T_{\text{nghịch}} = P \cdot T^{-1}$$

## I.11. Cài đặt

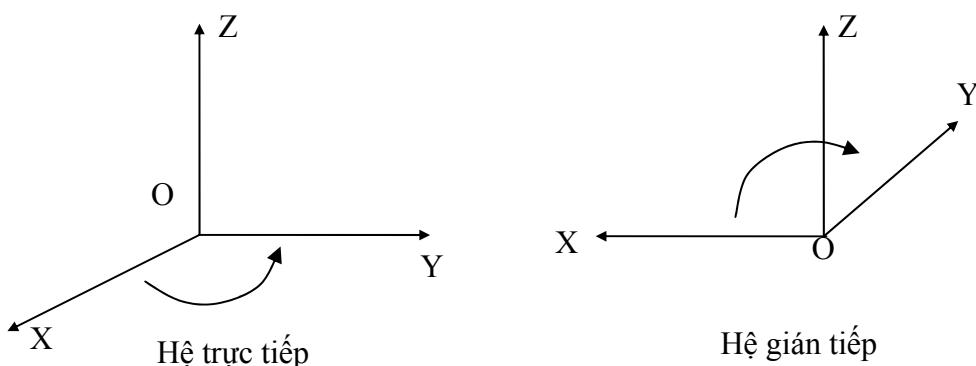
- Sinh viên cần xây dựng các thủ tục tìm ma trận của các phép biến đổi cơ bản trên. Mỗi thủ tục được cung cấp các đầu vào cần thiết và đầu ra là một ma trận biến đổi (mảng 2 chiều  $3 \times 3$  phần tử).
- Xây dựng một thủ tục với đầu vào là một đa giác và một ma trận biến đổi, đầu ra là đa giác đã được biến đổi (bằng cách nhân các đỉnh của đa giác với ma trận biến đổi).
- Viết chương trình sử dụng hai thủ tục trên để minh họa các phép biến đổi cơ bản. Chương trình cho phép biến đổi (như tịnh tiến, quay, biến đổi tỷ lệ, biến dạng,...) các hình cơ bản như tam giác, tứ giác, ngũ giác, ... hay một hình bất kỳ nào đó như hình các kí tự A, F, H, K.

## II. Các phép biến đổi Affine 3D

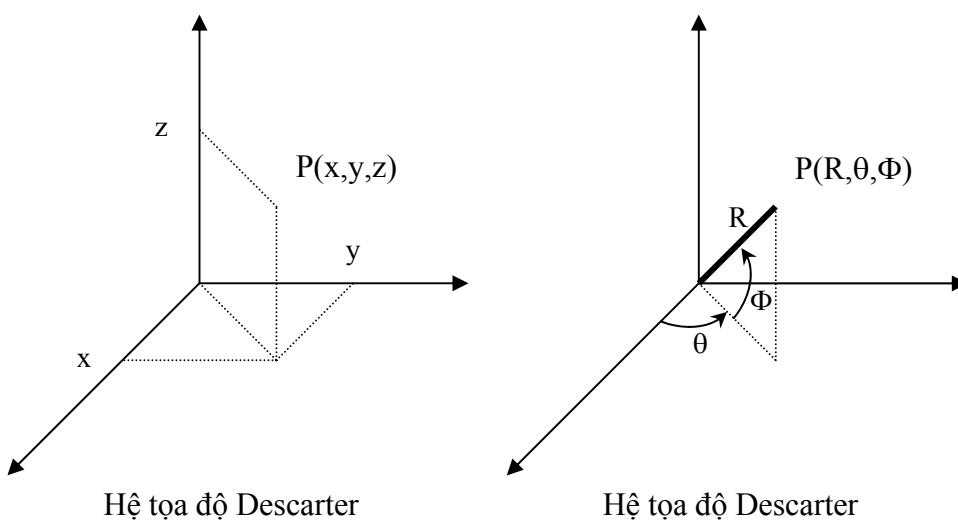
### II.1. Các hệ trục tọa độ

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

- *Hệ tọa độ trực tiếp* : nếu ta đang nhìn mặt phẳng (XY) thì trục Z hướng vào mắt chúng ta (Qui tắc bàn tay phải).
- *Hệ tọa độ gián tiếp* : Ngược lại (Qui tắc bàn tay trái).



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



Trong hệ tọa độ trực tiếp, Ta có công thức chuyển đổi tọa độ từ hệ tọa độ cũ sang hệ Đề-cát như sau:

$$x = R \cdot \cos(\theta) \cdot \cos(\Phi)$$

$$y = R \cdot \sin(\theta) \cdot \cos(\Phi)$$

$$z = R \cdot \sin(\Phi)$$

$$R^2 = x^2 + y^2 + z^2$$

Để thuận tiện cho việc tính toán chúng ta sẽ sử dụng hệ tọa độ thuần nhất (homogen), ma trận của các điểm có dạng  $(x, y, z, 1)$ . Ma trận biến đổi là ma trận vuông  $4 \times 4$ .

## II.2. Các công thức biến đổi

Phép biến đổi Affine 3D dạng thuần nhất có dạng:  $\mathbf{Q} = \mathbf{P} \cdot \mathbf{T}$

với  $\mathbf{P} = (P_x, P_y, P_z, 1)$ ;  $\mathbf{Q} = (Q_x, Q_y, Q_z, 1)$  và ma trận biến đổi  $\mathbf{T}$  có dạng:

$$\begin{pmatrix} t_{11} & t_{12} & t_{13} & 0 \\ t_{21} & t_{22} & t_{23} & 0 \\ t_{31} & t_{32} & t_{33} & 0 \\ tr_x & tr_y & tr_z & 1 \end{pmatrix}$$

### II.2.a. Phép tịnh tiến

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ tr_x & tr_y & tr_z & 1 \end{pmatrix}$$

Ma trận của phép tịnh tiến theo vector  $Tr = (tr_x, tr_y, tr_z)$  là:

$$T = \begin{pmatrix} a & 0 & 0 & 0 \\ 0 & b & 0 & 0 \\ 0 & 0 & c & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Trong đó:

a : hệ số tỷ lệ theo trục X

b : hệ số tỷ lệ theo trục Y

c : hệ số tỷ lệ theo trục Z

### II.2.c. Phép đối xứng

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

Đối xứng qua mặt (OXY):

Đối xứng qua mặt (OXZ):

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

Đối xứng qua mặt (0YZ):

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

### II.2.d. Phép quay

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

Quay quanh trục Z:

$$\begin{pmatrix} \cos(\alpha) & \sin(\alpha) & 0 & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Quay quanh trục X:

$$\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) & 0 \\ 0 & -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Quay quanh trục Y:

$$\begin{pmatrix} \cos(\alpha) & 0 & -\sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

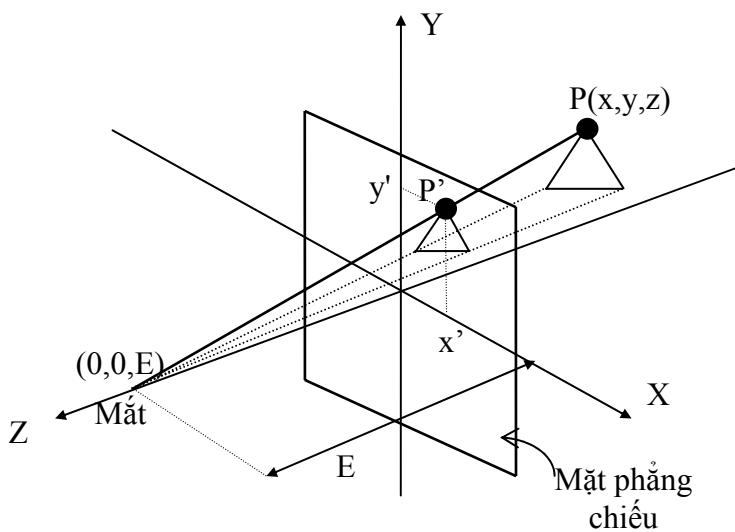
Chú ý: Tích của 2 ma trận nói chung không giao hoán.

## III. Các phép chiếu vật thể trong không gian lên mặt phẳng

### III.1. Phép chiếu phối cảnh (Perspective)

Phép chiếu này cho hình ảnh gần giống như khi ta nhìn vật thể trong thế giới thực.

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



Giả sử  $P$  nằm phía trước mắt, tức là  $P.z < E$ . Phương trình tham số của tia chiếu đi qua  $P$  được viết:

$$E*(1-t) + P*t \text{ với } t \in \mathbb{R}$$

Hay viết cụ thể là:  $P(t) = (0,0,E).(1-t) + (x,y,z).t$  (\*)

Với  $P(t)$  là điểm trên tia chiếu phụ thuộc vào tham số  $t$ .

$$(EP): \begin{cases} x_P = 0(1-t) + x.t \\ y_P = 0(1-t) + y.t \\ z_P = E(1-t) + z.t \end{cases} (*)$$

Giao điểm với mặt phẳng quan sát là  $P'$  sẽ có thành phần  $Z = 0$ . nên phương trình (\*) trở thành

$$\begin{cases} x_{P'} = 0(1-t) + x.t \\ y_{P'} = 0(1-t) + y.t \\ z_{P'} = E(1-t) + z.t = 0 \end{cases}$$

Từ  $E.(1-t) + z.t = 0$  suy ra  $t = \frac{1}{1 - \frac{z}{E}}$ . Thay  $t$  vào 2 phương trình trên ta tính được:

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

### Nhận xét

- i/ Phép chiếu phối cảnh không giữ nguyên hình dạng của vật thể.
- ii/ Chỉ có những đường thẳng song song với nhau đồng thời song song với mặt phẳng chiếu thì mới cho ảnh song song với nhau, còn tất cả các đường thẳng khác đều cho ảnh hội tụ đến tâm chiếu (mắt).
- iii/ Vật ở trước mặt phẳng chiếu thì được phóng lớn, sau mặt phẳng chiếu thì bị thu nhỏ. Vật ở xa thì trông nhỏ, ở gần thì trông lớn.

### III.2. Phép chiếu song song

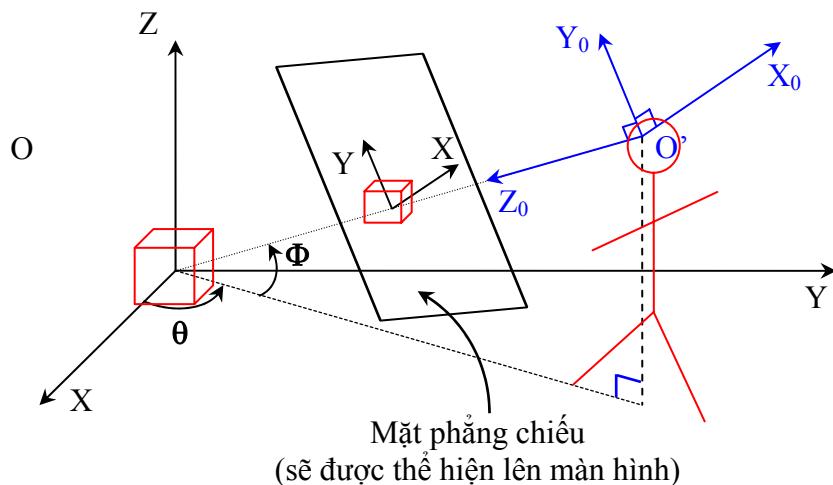
Nếu chúng ta chọn phương chiếu vuông góc với mặt phẳng chiếu (và cũng là vuông góc với mặt phẳng OXY) thì:

$$x' = x; \quad y' = y$$

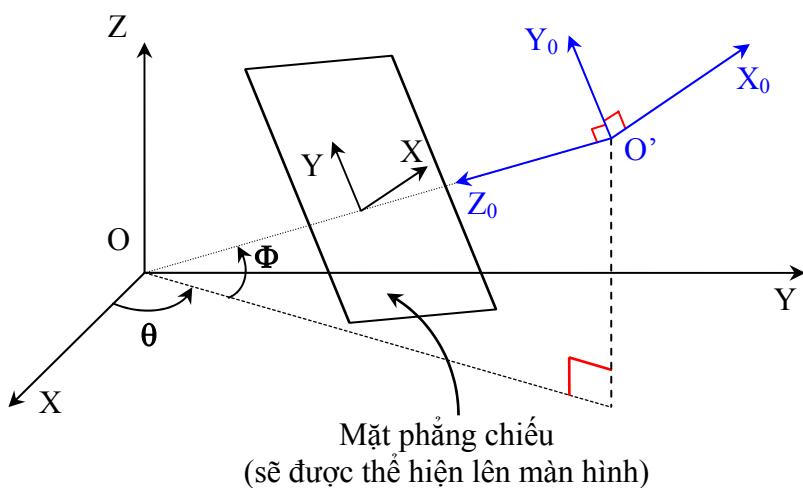
## IV. Quan sát vật thể 3D & Quay hệ quan sát

Khi mô tả việc quan sát một vật thể trong không gian ta cần lưu ý các điểm sau:

- Vật thể được chiếu lên một hệ trực tiếp ( $O, X, Y, Z$ )
- Mắt nằm ở gốc của một hệ gián tiếp thứ hai tạm gọi là ( $O', X_0, Y_0, Z_0$ )
- Mặt phẳng chiếu vuông góc với đường thẳng  $OO'$
- Trục  $Z_0$  của hệ toạ độ thứ 2 hướng vào gốc  $O$ . Hệ toạ độ thứ hai có tên là hệ toạ độ quan sát.

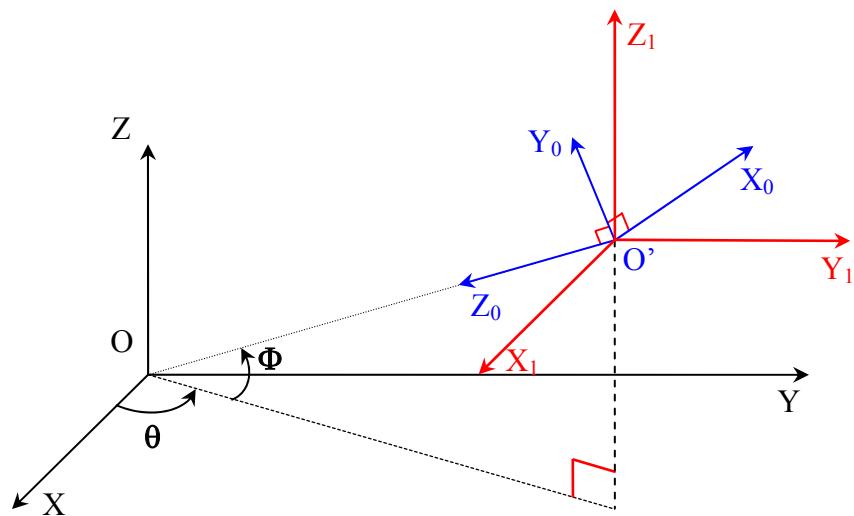


Mô hình quan sát vật thể đơn lẻ trong không gian.



Bây giờ, ta khảo sát phép biến đổi một điểm  $P(x,y,z)$  trong hệ toạ độ thứ nhất sang  $P'(x_0,y_0,z_0)$  trong hệ toạ độ thứ hai rồi chuyển sang toạ độ trên mặt phẳng quan sát.

**Bước 1:** Tịnh tiến hệ trục tọa độ từ gốc  $O$  thành  $O'$ .

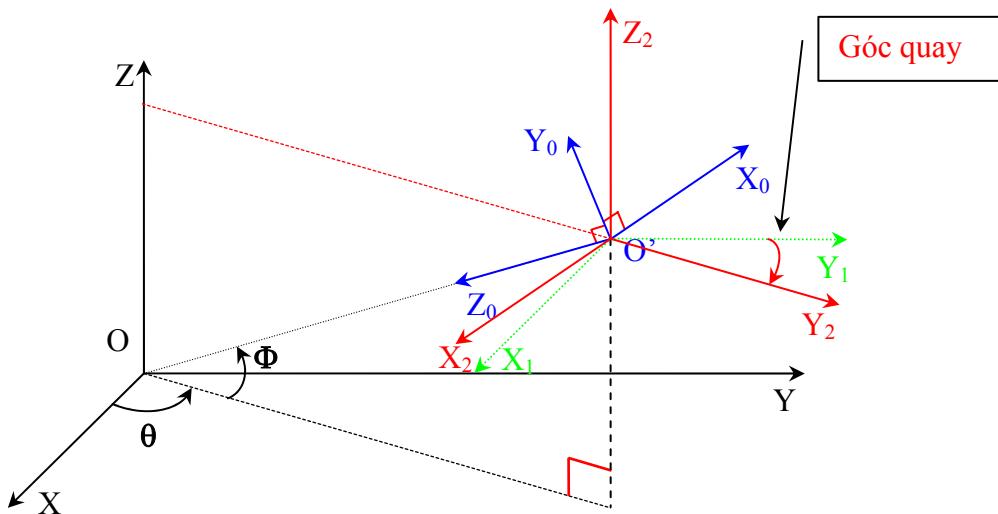


Ma trận của phép tịnh tiến hệ trục tọa độ theo vector  $\overrightarrow{OO'}$  là:

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

và hệ (OXYZ) biến đổi thành hệ ( $O'X_1Y_1Z_1$ ).

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



Ta gọi  $R_z$  là ma trận của phép quay quanh trục  $Z$  một góc  $a$

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

Vì đây là phép quay hệ trực, nên phải dùng ma trận nghịch đảo  $R_z^{-1}$

$$R_z^{-1} = \begin{pmatrix} \cos(a) & -\sin(a) & 0 & 0 \\ \sin(a) & \cos(a) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

ta thay góc  $a = -\theta'$ . Theo các phép toán lượng giác:

$$\sin(-\theta') = -\sin(\theta') = -\sin(90^\circ - \theta) = -\cos(\theta)$$

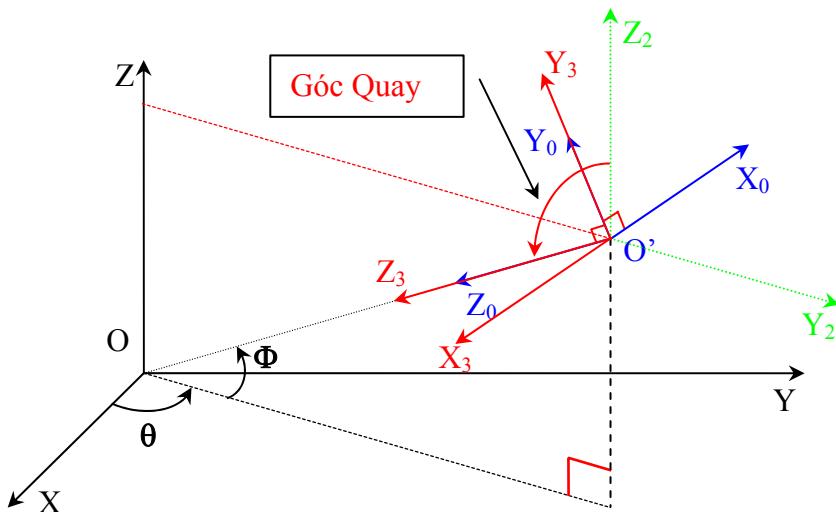
$$\cos(-\theta') = \cos(\theta') = \cos(90^\circ - \theta) = \sin(\theta)$$

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

$$\mathbf{B} = \begin{pmatrix} \sin(\theta) & \cos(\theta) & 0 & 0 \\ -\cos(\theta) & \sin(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

và hệ  $(O'X_1, Y_1, Z_1)$  biến đổi thành hệ  $(O'X_2, Y_2, Z_2)$ .

**Bước 3:** Quay hệ  $(O'X_2, Y_2, Z_2)$  một góc  $90^\circ + \Phi$  quanh trục  $X_2$ . Phép biến đổi này sẽ làm cho trục  $Z_2$  hướng đến gốc O.



Ta có:

$$R_x = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & \sin(a) & 0 \\ 0 & -\sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} R_x^{-1} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(a) & -\sin(a) & 0 \\ 0 & \sin(a) & \cos(a) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

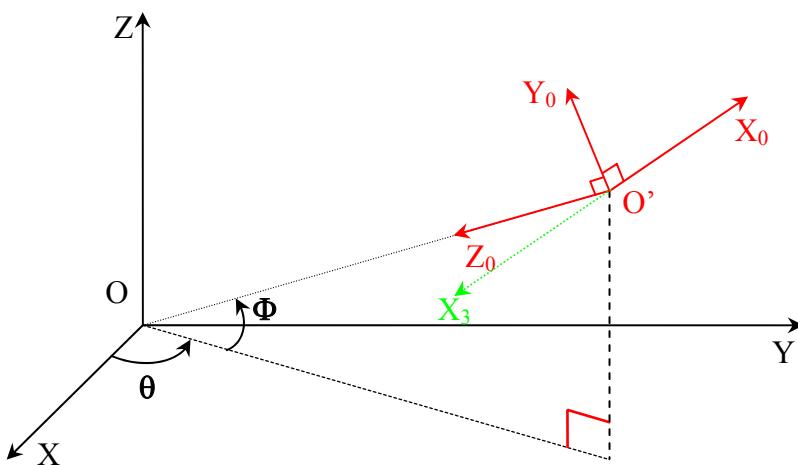
Thay góc  $a = 90^\circ + \Phi$ , ta có:  $\cos(90^\circ + \Phi) = -\sin(\Phi)$  và  $\sin(90^\circ + \Phi) = \cos(\Phi)$  nên ma trận tìm được sẽ có dạng:

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

Lúc này, hệ  $(O'X_2, Y_2, Z_2)$  biến đổi thành hệ  $(O'X_3, Y_2, Z_3)$ .

**Bước 4:** Biến đổi hệ trực tiếp  $(O'X_3, Y_3, Z_3)$  thành hệ gián tiếp.

Trong bước này, ta phải đổi hướng trục  $X_3$  bằng cách đổi dấu các phần tử của cột X. Ta nhận được ma trận:



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

### Tóm lại

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

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

Gọi  $T = A \cdot B \cdot C \cdot D$ , ta tính được:

$$T = \begin{pmatrix} -\sin(\theta) & -\cos(\theta) \cdot \sin(\phi) & -\cos(\theta) \cdot \cos(\phi) & 0 \\ \cos(\theta) & -\sin(\theta) \cdot \sin(\phi) & -\sin(\theta) \cdot \cos(\phi) & 0 \\ 0 & \cos(\phi) & -\sin(\phi) & 0 \\ 0 & 0 & R & 1 \end{pmatrix}$$

Cuối cùng ta có:

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

hay:

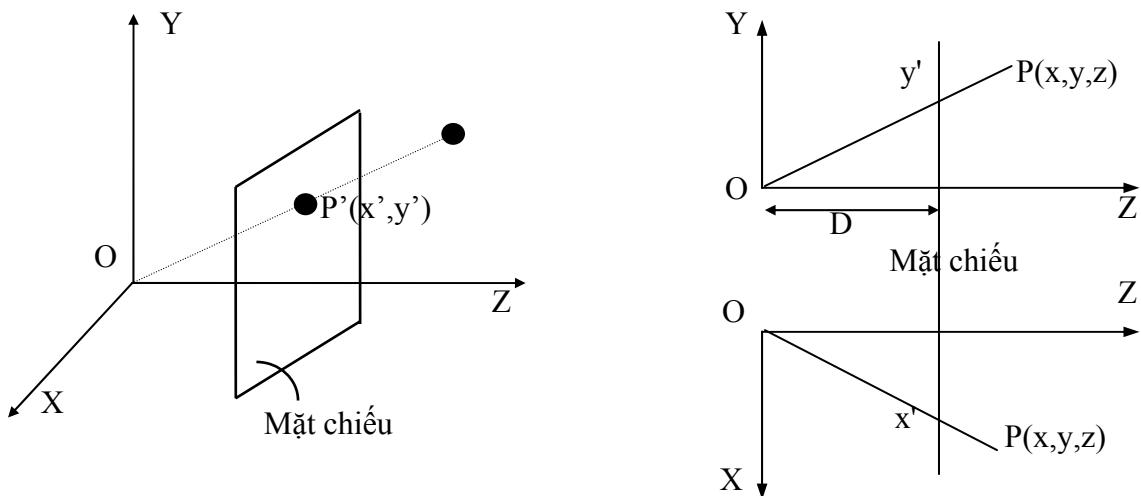
$$x_0 = -x \cdot \sin(\theta) + y \cdot \cos(\theta)$$

$$y_0 = -x \cdot \cos(\theta) \cdot \sin(\Phi) - y \cdot \sin(\theta) \cdot \sin(\Phi) + z \cdot \cos(\Phi)$$

$$z_0 = -x \cdot \cos(\theta) \cdot \cos(\Phi) - y \cdot \sin(\theta) \cdot \cos(\Phi) - z \cdot \sin(\Phi) + R$$

\* Vậy giờ ta lại chiếu ảnh của hệ quan sát lên màn hình.

#### IV.1. Phép chiếu phối cảnh

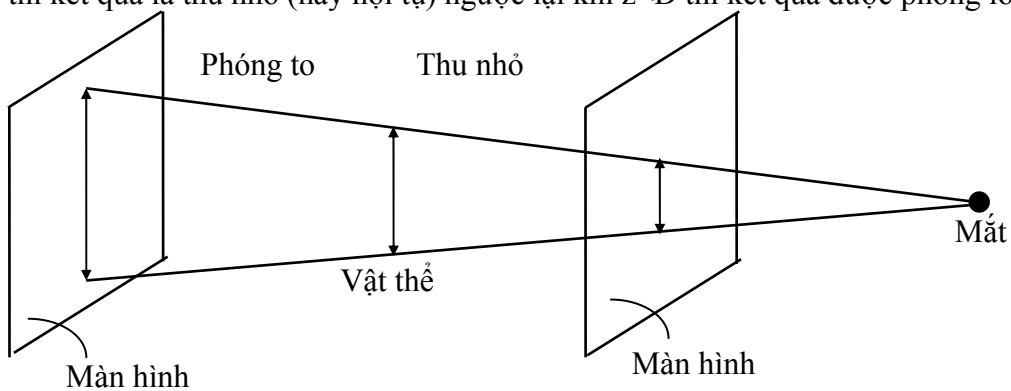


Cho một điểm  $P(x, y, z)$  và hình chiếu  $P'(x', y', z')$  của nó trên mặt phẳng chiếu. gọi  $D$  là khoảng cách từ mặt phẳng chiếu đến mắt (gốc tọa độ).

Xét các tam giác đồng dạng, ta có:

$$\begin{aligned} x'/x &= D/z & \text{và} & y'/y = D/z \\ \text{hay } x' &= x \cdot D/z & \text{và} & y' = y \cdot D/z \end{aligned}$$

**Chú ý:** nói một cách khác là tọa độ  $x$  và  $y$  được nhân với một hệ số  $D/z$ . Nên nếu  $z > D$  thì kết quả là thu nhỏ (hay hội tụ) ngược lại khi  $z < D$  thì kết quả được phóng lớn.



#### IV.2. Phép chiếu song song

Phép chiếu song song có thể được xem như tâm chiếu ở xa vô cực nên lúc này ta có:

$$x' = x \quad \text{và} \quad y' = y$$

#### KẾT LUẬN

Ta có 4 giá trị ảnh hưởng đến phép chiếu vật thể 3D là: các góc  $\theta$ ,  $\Phi$ , khoảng cách R từ O đến O' và khoảng cách D từ O' đến mặt phẳng quan sát.

Như vậy:

- Tăng giảm  $\theta$  sẽ giúp ta có thể di chuyển hệ quan sát quanh trục OZ để quan sát bốn mặt bên của vật thể.
- Tăng giảm  $\Phi$  sẽ giúp ta có thể di chuyển quanh gốc O theo hướng vuông góc với mặt phẳng OXY để quan sát được mặt trên hay mặt dưới của vật thể.
- Tăng giảm R để quan sát vật từ xa hay gần.
- Tăng giảm D để phóng to hay thu nhỏ ảnh.

#### **IV.3. Cài đặt**

- ❖ Sinh viên cần xây dựng một thủ tục cho phép chuyển tọa độ của đối tượng từ hệ tọa độ trực tiếp sang hệ tọa độ quan sát.
- ❖ Xây dựng một thủ tục cho phép chiếu vật thể từ hệ tọa độ quan sát lên mặt phẳng quan sát theo phép chiếu song song hay phôi cảnh rồi vẽ kết quả chiếu lên màn hình.
- ❖ Xây dựng chương trình sử dụng 2 thủ tục trên để minh họa hình ảnh của một hình hộp. Nâng cao hơn sinh viên cần theo dõi các góc  $\theta$  và  $\Phi$  để chương trình có thể thể hiện được hình hộp từ nhiều góc độ.

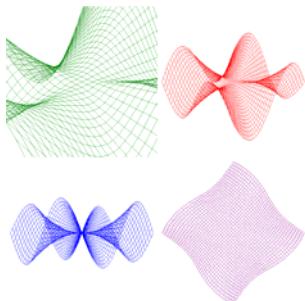
#### **IV.4. Ví dụ minh họa**

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

### **V. Bài tập cuối chương**

1. Cài đặt thuật toán xén một đoạn thẳng vào một hình chữ nhật có cạnh không song song với trục tọa độ.
2. Viết chương trình vẽ một Ellipse có các trục không song song với hệ trục tọa độ.
3. Dựa vào bài tập 2, hãy mô phỏng quá trình quay của một Ellipse xung quanh tâm của nó.
4. Mô phỏng quá trình quay, tịnh tiến của một hình bất kỳ trong mặt phẳng quanh trục tọa độ.
5. Mô phỏng chuyển động của trái đất xung quanh mặt trời đồng thời mô tả chuyển động của mặt trăng xung quanh trái đất.
6. Viết chương trình vẽ đồng hồ đang hoạt động.
7. Viết chương trình vẽ các khối đa diện đều trong không gian.

# Chương V: Các phương pháp dựng đường cong và mặt cong



Trong chương này chúng ta sẽ xem xét một số phương pháp để tạo ra các đường cong và mặt cong tron để biểu diễn các vật thể đó là phương pháp Bezier và B-Spline. Ở đây ta chỉ bàn đến các phương pháp xây dựng đường và mặt cong dựa trên một số đặc điểm dữ kiệu mô tả về đường cong đó. Các dữ liệu mô tả đó là các điểm kiểm soát để từ đó có thể nội suy ra đường cong và mặt cong mong muốn. Hay nói rõ hơn là với một đường cong bất kì ta có thể tiếp cận theo nhiều cách:

- ☒ Lấy một số điểm trên đường cong làm mẫu rồi tìm một hàm toán học và chỉnh hàm này sao cho nó đi qua các điểm mẫu và khớp với đường cong ban đầu. Khi đó ta có được công thức của đường cong và dùng nó để phát sinh lại.
- ☒ 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 khả năng là đường cong được tạo ra không khớp với đường cong ban đầu, lúc này ta sẽ di chuyển một vài điểm kiểm soát và khi đó thuật toán phát sinh một đường cong mới dựa trên các điểm kiểm soát mới thay đổi này. Tiến trình thay đổi các điểm kiểm soát được lặp đi lặp lại cho đến khi đường cong được tạo ra khớp với đường cong lúc ban đầu. Lúc này ta sẽ lưu trữ tập điểm kiểm soát này để phát sinh lại đường cong khi cần.

Ở đây ta sẽ tiếp cận vấn đề theo phương pháp thứ hai. Các đường cong hay mặt cong được xây dựng ở đây được gọi là các đường cong hoặc mặt cong Bezier hay một loại nữa là đường cong hoặc mặt cong Spline.

## I. Đường cong Bezier & mặt cong Bezier.

Trong phần này chúng ta sẽ trình bày phương pháp để định nghĩa một đường cong tham số  $P(t)$  dựa trên một tập các điểm. Trước tiên ta xem xét ý tưởng của thuật toán Casteljau để định nghĩa các đường cong loại này, sau đó sẽ đưa ra dạng toán học của đường cong Bezier qua công thức Bernstein.

## I.1. Thuật toán de Casteljau:

Để xây dựng đường cong  $P(t)$ , thuật toán này dựa trên một dãy các điểm cho trước rồi tạo ra các giá trị  $P(t)$  ứng với các giá trị  $t$  khác nhau. Khi thay đổi các điểm này thì sẽ kéo theo sự thay đổi dạng của đường cong. Phương pháp tạo đường cong ở đây là dựa trên một dãy các thao tác nội suy tuyến tính (hay nội suy khoảng giữa “in-between”).

Ví dụ với 3 điểm  $P_0, P_1, P_2$  ta có thể xây dựng một đường cong Parabol nội suy từ 3 điểm này bằng cách: chọn một giá trị  $t$  nào đó nằm giữa 0 và 1 rồi chia đoạn  $P_0P_1$  theo tỷ lệ  $t$ , ta được  $P_0^1(t)$  trên  $P_0P_1$ . Tương tự chia tiếp  $P_1P_2$  theo cùng tỷ lệ  $t$  ta được  $P_1^1(t)$ . Nối  $P_0^1(t)$  với  $P_1^1(t)$ , Lại lấy điểm trên  $P_0^1P_1^1$  chia theo tỷ lệ  $t$  ta được  $P_0^2(t)$ .

Với cách làm này, khi lấy  $t$  ở những giá trị khác nhau giữa 0 và 1 thì sẽ được tập điểm  $P_0^2(t)$ , và đó chính là đường cong  $P(t)$ .

Biểu diễn bằng phương trình:

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

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

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

Thay  $P_0^1$  và  $P_1^1$  bằng vé phái của hai phương trình trên 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  $t$  với các tham số  $P_0, P_1, P_2$  do đó nó là một Parabol.

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

Giả sử tập điểm là  $P_0, P_1, \dots, P_L$

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 theo phương trình:

$$P_i^r(t) = (1-t).P_i^{r-1}(t) + t.P_{i+1}^{r-1}(t)$$

Với  $r=0, 1, 2, \dots, L$ ;  $i=0, 1, 2, \dots, L-r$

Thế hệ cuối cùng  $P_0^L(t)$  được gọi là đường cong BEZIER, và các điểm  $P_0, P_1, \dots, P_L$  được gọi là các điểm kiểm soát hay đa giác BEZIER.

## I.2. Dạng BERNSTEIN của các đường cong BEZIER

Đường cong BEZIER dựa trên  $(L+1)$  điểm kiểm soát  $P_0, P_1, \dots, P_L$  được cho bởi công thức

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

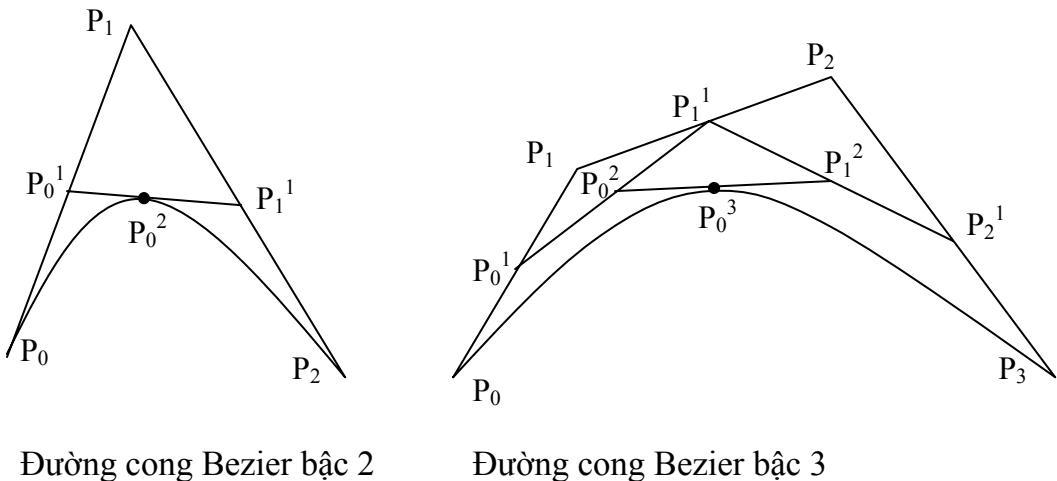
$P(t)$  ứng với một giá trị  $t$  nào đó nằm giữa 0 và 1 là một điểm trong 2D hay 3D.

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

$$B_k^L(t) = C_L^k (1-t)^{L-k} t^k = (1-t)^{L-k} t^k, \text{ với } L \geq k$$

Mỗi đa thức Bernstein có bậc là  $L$ .

Thông thường ta còn gọi các  $B_k^L(t)$  là các hàm trộn. Bởi ta có thể hiểu vectơ  $P(t)$  như là một sự pha trộn của các vectơ  $P_0, P_1, \dots, P_L$  theo một tỷ lệ được nhất định theo công thức Bernstein ở các giá trị  $t$  khác nhau giữa 0 và 1.



Đường cong Bezier bậc 3 được vẽ bởi chương trình Paint

Đối với mặt BEZIER ta có phương trình sau:

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

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

### I.3. Dạng biểu diễn ma trận của đường Bezier

Để thích hợp cho các tính toán trên máy tính (việc tính toán trở nên hợp lý và nhanh chóng) người ta thường biểu diễn công thức dưới dạng ma trận

$$\begin{aligned} B^L(t) &= (B_0^L(t), B_1^L(t), B_2^L(t), \dots, B_L^L(t)) \\ P &= (P_0, P_1, P_2, \dots, P_L) \end{aligned}$$

Do đó  $P(t) = B^L(t).P$

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

Mỗi  $B_k^L(t)$ , với  $k=0,1,\dots,L$  lại có thể xem là tích điểm của  $(t^0, t^1, t^2, \dots, t^L)$  với các hệ số tương ứng của các  $t^i$   $i=0..L$ . Vector  $(t^0, t^1, t^2, \dots, t^L)$  được gọi là cơ sở lũy thừa (Power basis). Dưới dạng đa thức có thể biểu diễn  $B_k^L(t)$  như sau:

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

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

$$P(t) = \text{Pow}^L(t) \cdot \text{Bez}^L \cdot P^T$$

Trong đó:

- ❖  $\text{Pow}^L(t) = (t^0, t^1, t^2, \dots, t^L)$
- ❖  $\text{Bez}^L$  là ma trận biểu diễn mảng  $B^L(t)$  trong đó mỗi hàng  $i$  của ma trận ứng với các hệ số tương ứng  $(a_0, a_1, a_2, \dots, a_L)$  của đa thức  $B_i^L(t)$ .

Ví dụ ma trận  $\text{Bez}^3$  cho các đường Bezier bậc 3:

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

Để tạo ra một đường cong Bezier từ một dãy các điểm kiểm soát người ta áp dụng phương pháp PolyLine gần đúng bằng cách lấy mẫu hàm  $P(t)$  ở các giá trị cách đều nhau của tham số  $T$ . Ví dụ như  $t_i = i/N$  với  $i=0,1,2,\dots,N$ . Khi đó ta sẽ được các điểm  $P(t_i)$  từ công thức, nối các điểm này lại bằng các đoạn thẳng ta sẽ được đường cong Bezier gần đúng. Để tính  $P(t_i)$  có thể áp dụng dạng ma trận của  $P(t)$  đã nói ở trên, khi đó chỉ có thành phần  $\text{Pow}^L(t)$  là thay đổi, còn tích

$$\text{Bez}^L \cdot P^T \quad \text{với } P = (P_0, P_1, \dots, P_L) \text{ là không đổi.}$$

## I.4. Các tính chất của đường cong BEZIER

Đường cong Bezier có một số tính chất quan trọng sau:

### I.4.a. Nội suy được các điểm đầu và điểm cuối:

Thật vậy:

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

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

$$\text{Mà } B_k^L(0) = \frac{L!}{k!(L-k)!} (1-0)^{L-k} \cdot 0^k = 0 \quad k \neq 0 \text{ và } k \neq L$$

$$\text{Do đó } P(0) = P_0 B_0^L(0) + P_L B_L^L(0) = P_0 + 0 = P_0$$

Tương tự cho  $P(1)$  ta có  $P(1)=P_L$

Vậy đường cong Bezier bắt đầu tại điểm  $P_0$  và kết thúc tại điểm  $P_L$

### I.4.b. Tính biến đổi Affine:

Khi biến đổi Affine một đường cong Bezier, ta không cần biến đổi mọi điểm trong đường cong một cách riêng rẽ mà chỉ cần biến đổi Affine các điểm kiểm soát của đường cong đó, rồi sử dụng công thức Bernstein để tái tạo đường

đường Bezier đã được biến đổi. Ta gọi đây là tính bất biến Affine của các đường Bezier.

Chứng minh:

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

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

N: ma trận biến đổi

Tr: vectơ 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 vectơ  $P_k$  ta sẽ chứng minh đường cong này chính là  $P'(t)$ .

Khai triển (\*) ta có:

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

nhưng do  $\sum_{k=0}^L B_k^L(t) = 1$  nên  $\sum_{k=0}^L Tr.B_k^L(t) = Tr \cdot \sum_{k=0}^L B_k^L(t) = Tr$

(Chứng minh  $\sum_{k=0}^L B_k^L(t) = 1$ . Thật vậy, ta đã biết khai triển nhị thức Newton  $(a+b)^L$ ,

ở đây áp dụng với  $a=(1-t)$  và  $b=t$  lúc này  $\sum_{k=0}^L B_k^L(t) = [(1-t)+t]^L = 1^L = 1$  ).

Vì vậy ta có  $P'(t)$  nằm trên đường cong Bezier tạo ra bởi các điểm kiểm soát  $P'_k$ . Với  $P'_k$  là ảnh của  $P_k$  qua phép biến đổi Affine.

#### I.4.c. Tính chất của bao lồi

Ta nhắc lại bao lồi của các điểm kiểm soát là tập đỉnh nhỏ nhất chứa tất cả các điểm kiểm soát đó. Đó cũng chính là tập tất cả các tổ hợp lồi của các điểm kiểm soát.

$$\sum_{k=0}^L \alpha_k P_k \text{ với } \alpha_k \geq 0 \text{ và } \sum_{k=0}^L \alpha_k = 1$$

Do  $P(t)$  là một tổ hợp lồi của các điểm kiểm soát  $t$ , vì không có hệ số Bernstein nào âm và  $\sum_{k=0}^L B_k^L(t) = 1$  nên mọi điểm trong đường cong Bezier sẽ nằm trong bao lồi của các điểm kiểm soát.

#### I.4.d. 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 do bị kẹp vào trong bao lồi nên nó cũng trở thành đường thẳng.

#### I.4.e. **Bất biến với những phép biến đổi Affine:**

Có thể đổi đường cong Bezier trong một khoảng tham số khác với khoảng  $[0,1]$ , ví dụ như một khoảng  $[a,b]$  nào đó bởi một ánh xạ nào đó, ví dụ: đường cong tham số  $t$  thành đường cong tham số  $u$  không làm thay đổi đường cong ban đầu (dùng tính bất biến Affine).

#### I.4.f. **Đạo hàm của các đường Bezier:**

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

là một đường cong Bezier khác được tạo ra từ các vectơ kiểm soát  $\Delta P_k$ .

### I.5. **Đánh giá các đường cong Bezier & sự khác biệt của các đường cong Spline:**

Bằng cách điều chỉnh 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 được lặp đi lặp lại cho đến khi toàn bộ đường cong thỏa mãn yêu cầu.

Tuy nhiên, có một vấn đề đối với đường cong Bezier là tính cục bộ yếu của nó, nghĩa là khi ta thay đổi bất kỳ một điểm kiểm soát nào thì toàn bộ đường cong bị thay đổi theo, nhưng trong thực tế thường ta 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 có thể thấy được qua việc tất cả các đa thức  $B_k^L(t)$  đều khác 0 trên khoảng  $[0,1]$ . Mặt khác đường cong  $P(t)$  bùn thân nó lại là một tổ hợp tuyến tính của các điểm kiểm soát được gia trọng bởi các hàm  $B_k^L(t)$  nên ta suy ra rằng mọi điểm kiểm soát đều có ảnh hưởng đến đường cong ở tất cả các giá trị  $t \in (0,1)$ . Do đó hiệu chỉnh bất kì điểm kiểm soát nào cũng đều ảnh hưởng đến dạng đường cong trên toàn thể.

Để giải quyết vấn đề này người ta đã đi đến sử dụng một tập các hàm trộn khác nhau thay vì chỉ một hàm trộn  $B_k^L(t)$  như của đường cong Bezier, các hàm trộn này có giá mang chỉ là một phần của khoảng  $[0,1]$ , hay nói cách khác là mỗi hàm trộn sẽ chỉ trộn với một số điểm kiểm soát để cho ra một phần của đoạn cong. Tập các đoạn cong do mỗi hàm trộn mang lại sẽ tạo nên một đường cong mà ta mong muốn. Như vậy 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.

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

$$a(t) = 1/2 t^2 \quad \text{có giá mang } [0,1]$$

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

Giá mang của  $g(t)$  là  $[0,3]$

Các giá trị của  $t$  ứng với các chốt nối của các đoạn gọi là nút (knot).

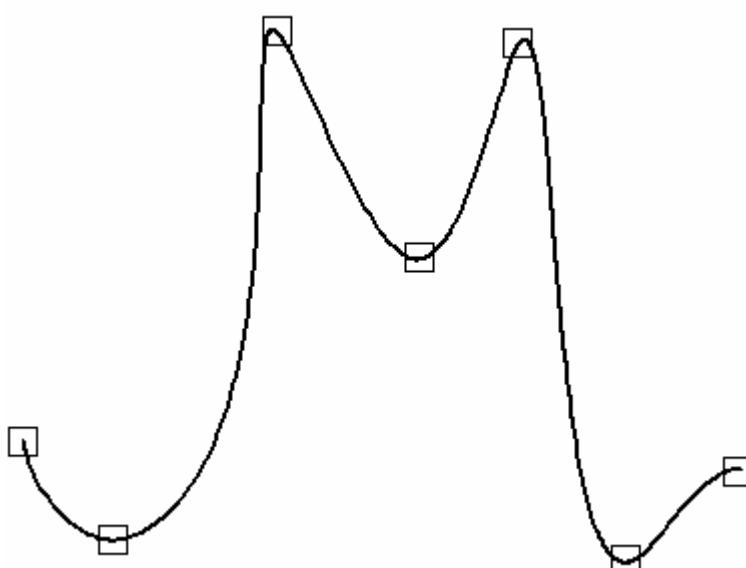
Hơn nữa tại các chốt nối đường cong  $g(t)$  là trơn, không bị gấp khúc. Ta gọi đó là hàm Spline.

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 một hàm trộn để tạo ra đường cong  $P(t)$ , dựa trên các điểm kiểm soát  $P_0, P_1, \dots, P_L$ .

Khi đó:

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

## II. Đường cong Spline và B-Spline:



Một đường cong Spline được vẽ bởi chương trình AutoCad

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 một hàm trộn để tạo ra đường cong  $P(t)$ , dựa trên các điểm kiểm soát  $P_0, P_1, \dots, P_L$ .

Khi đó:

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

Theo trên ta có

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

với :  $P_k : k=0,1,\dots,L$  là các điểm kiểm soát.

$R_k(t) : k=0,1,\dots,L$  là các hàm trộn, liên tục trong mỗi đoạn con  $[t_i, t_{i+1}]$  và ở mỗi nút. Mỗi  $R_k(t)$  là một đa thức riêng phần (piecewise polynomial). Các đoạn

đường cong riêng phần này gặp nhau ở mỗi 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 nhiều họ hàm trộn có thể được dùng để tạo ra các đường cong Spline có thể định nghĩa trên vector nút đó. Mỗi họ 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 mà nhờ vậy nó đem lại khả năng kiểm soát cục bộ lớn nhất. Đó là các B-Spline (B là viết tắt của Basic).

Đối với các hàm B-Spline, mỗi đa thức riêng phần tạo nên nó có một cấp nào đó, người ta gọi là m, do đó thay vì dùng ký hiệu  $R_k(t)$  cho các hàm riêng phần này bởi  $N_{k,m}(t)$ .

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

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

Trong các hàm B-Spline cấp m thì hàm B-Spline cấp 2 và cấp 3 là quang trọng nhất, nó được dùng trong hầu hết các chương trình xử lý đồ họa.

Tóm lại: để xây dựng các đường cong B-Spline ta cần có:

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

$$N_{k,m}(t) = N_{k, m-1}(t) + N_{k+1, m-1}(t)$$

Đây là một công thức đệ quy với

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

(Hàm hằng 1 trên đoạn  $[t_k, t_{k+1}]$ )

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

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

(tương tự như mặt Bezier)

\* **Ghi chú:** Các đường Bezier là các đường B-Spline.

Sau đây là một công thức cho đường cong Spline bậc 3 có tên là Hermite Spline với công thức sau:

$$P(u) = [u^3 \ u^2 \ u \ 1] M_c [P_{k-1} \ P_{k+2} \ P_{k+2}]$$

với ma trận cốt yếu là

$$M_c = \begin{bmatrix} -s & 2-s & s-2 & s \\ 2s & s-3 & 3-2s & -s \\ -s & 0 & s & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

với  $s = (1-t)/2$ .

Hay ở dạng tường minh sẽ là:

$$\begin{aligned} P(u) &= P_{k-1}(-su^3 + 2su^2 - su) + P_k [(2-s)u^3 + (s-3)u^2 + 1] + \\ &\quad P_{k+1} [(s-2)u^3 + (3-2s)u^2 + su] + P_{k+2} (su^3 - su^2) \\ &= P_{k-1} \text{CAR}_0(u) + P_k \text{CAR}_1(u) + P_{k+1} \text{CAR}_2(u) + P_{k+2} \text{CAR}_3(u) \end{aligned}$$

và ta gọi  $\text{CAR}_k(u)$  với  $k=0,1,2,3$  là các hàm trộn (blending).

## Chương VI: Mô hình WireFrame



Thế giới quanh ta hầu hết đều là những đối tượng 3 chiều (3D, gồm chiều rộng, chiều dài và chiều cao). Bằng những kỹ thuật như vẽ, chụp hình, quay phim... chúng ta thường có biểu diễn hình ảnh thực 3 chiều của đối tượng dưới những góc nhìn khác nhau bằng các hình vẽ trên mặt phẳng hai chiều của giấy, khung vẽ hay màn ảnh.v.v... để thể hiện lại các đối tượng thực tế. Các hình ảnh đó phải tuân theo một số quy luật về phối cảnh, sáng tối nhằm giúp người xem có thể tưởng tượng lại hình ảnh mong muốn.

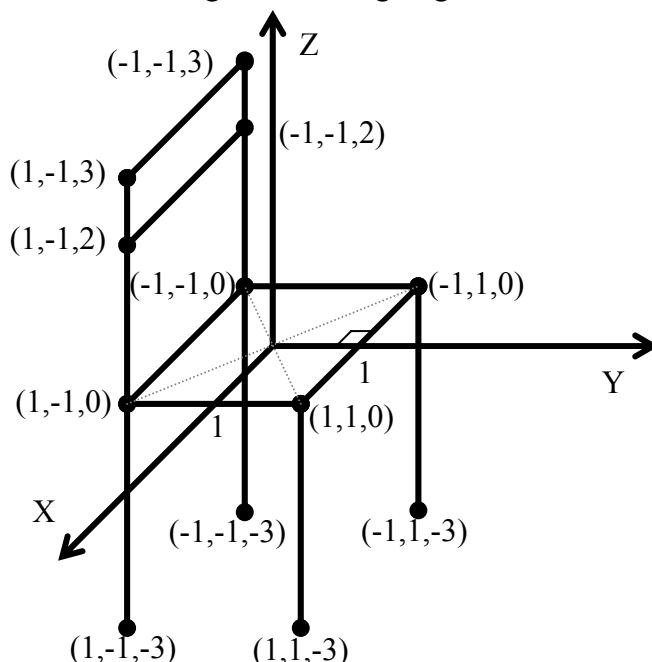
Ngày nay dưới sự tiến bộ của máy tính điện tử thì nhu cầu thể hiện các đối tượng bằng máy tính điện tử trở thành một nhu cầu thiết yếu. Thể hiện các đối tượng thực bằng máy tính đặc biệt là biểu diễn các đối tượng 3 chiều là một công việc đem lại lợi ích rất lớn, bởi nó dễ dàng cho ta những góc nhìn khác nhau, những cách thể hiện, những biến đổi và nhiều tác động khác một cách hiệu quả mà không mất nhiều thời gian và tiền của. Nếu như trước đây, một người kiến trúc sư làm việc trên bàn giấy với công cụ là bút vẽ và giấy, thì công việc là rất khó nhọc, đôi khi một bảng vẽ chỉ vì sai sót một vài chi tiết hay muốn sửa đổi một vài điểm trên bản vẽ, thì việc phải vẽ lại toàn bộ bản vẽ là điều không tránh khỏi. Ngày nay, với sự trợ giúp đắc lực của máy tính, các kiến trúc sư có thể nhanh chóng thể hiện các ý tưởng của mình trên máy tính, công việc thể hiện bằng vẽ trở nên dễ dàng hơn, các sai sót hay sửa đổi được thực hiện một cách dễ dàng mà không nhất thiết phải thực hiện vẽ lại toàn bộ bảng vẽ, ngoài ra máy tính còn có thể thể hiện ra nhiều bảng vẽ khác nhau cho cùng một đối tượng dưới nhiều góc nhìn, và nhiều chức năng tiện ích khác đi kèm tính kết cấu lực, tạo phối cảnh, tạo đoạn phim mô tả công trình,...

Trong phần này chúng ta sẽ giới thiệu một số khái niệm và kỹ thuật trong việc biểu diễn các đối tượng 3 chiều trên máy tính, bắt đầu từ các đối tượng đơn giản như các hình khối, các đa diện, các mặt mà ta có thể đơn giản hóa cách thể hiện nó bằng một tập các đỉnh và các cạnh nối liền.

## I. Mô hình Wireframe:

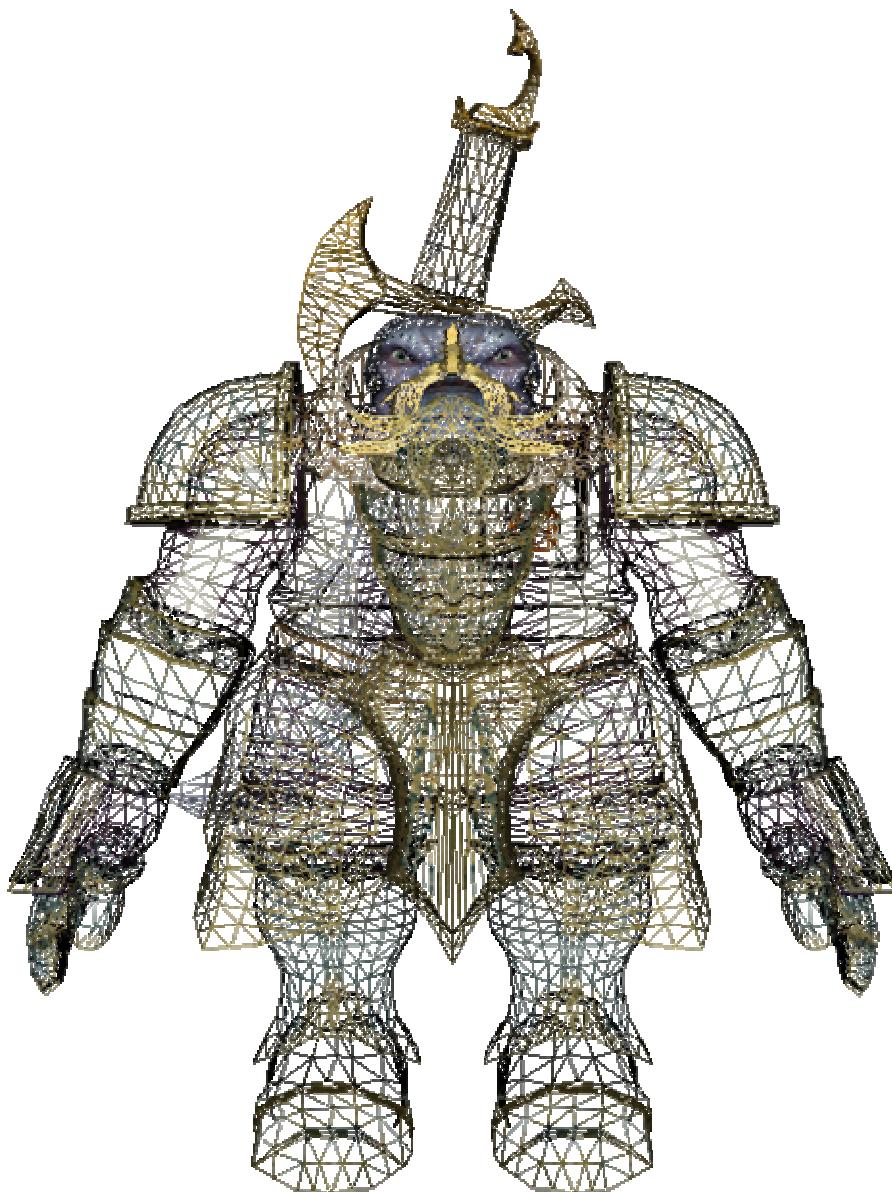
Mô hình Wireframe hay còn gọi là mô hình khung dây lưu trữ thông tin về hình dáng (bộ khung) của đối tượng. Thông tin lưu trữ được tổ chức thành 2 danh sách, một *danh sách đỉnh*, và một *danh sách cạnh* được tạo nên từ các đỉnh **chứa trong danh sách đỉnh**. Danh sách đỉnh lưu giữ toạ độ các đỉnh của đối tượng mà mỗi đỉnh bao gồm các thành phần toạ độ (x,y,z), danh sách cạnh lưu giữ danh sách các cặp thứ tự điểm tạo nên cạnh mà toạ độ của các điểm này chứa trong danh sách điểm.

Ví dụ: để lưu trữ hình dáng một chiếc ghế gỗ thô sơ như hình vẽ sau:



Danh sách đỉnh			
Stt	X	Y	Z
1	1	1	-3
2	-1	1	-3
3	-1	-1	-3
4	1	-1	-3
5	1	1	0
6	-1	1	0
7	-1	-1	0
8	1	-1	0
9	1	-1	2
10	-1	-1	2
11	1	-1	3
12	-1	-1	3

Danh sách cạnh		
Stt	đỉnh 1	đỉnh 2
1	1	5
2	2	6
3	3	12
4	4	11
5	5	6
6	6	7
7	7	8
8	8	5
9	9	10
10	11	12



Mô hình wireframe cho một nhân vật trong game

Có nhiều cách để lưu giữ một mô hình WireFrame trong một ứng dụng như: mảng, xâu, Danh sách móc nối,... và mỗi hình thức đều có những ưu nhược điểm riêng. Ở đây chúng ta sẽ giới thiệu một cấu trúc Record đơn giản dựa trên 2 mảng để lưu giữ thông tin về đỉnh và cạnh.

```
Const      MaxDinh=50;      {Số đỉnh lớn nhất cho phép lưu trữ}
          MaxCanh=100;     {Số cạnh lớn nhất cho phép lưu trữ}
Type       Pointer_3D=record
          x,y,z:real;
        end;
```

```
WireFrame=record
    SoDinh      : 0..MaxDinh; { chứa số đỉnh của đối tượng}
    Dinh        : Array[1..MaxDinh] of Pointer_3D;
    SoCanh      : 0..MaxCanh; { chứa số cạnh của đối tượng}
    Canh        : Array[1..MaxCanh] of record
                                D1,D2: 0..MaxDinh;
                                end;
end;
```

Trong đó mỗi cạnh là một phần tử của mảng Canh, phần tử Canh[i] chứa hai thông tin D1 và D2 là thứ tự của 2 đỉnh trong danh sách Dinh, mà hai đỉnh đó tạo nên cạnh.

## II. Vẽ hình dựa trên dữ liệu kiểu WireFrame với các phép chiếu

Vẽ một đối tượng ở dạng mô hình khung thì một cách đơn giản là ta vẽ các cạnh trong danh sách. Song để vẽ một cạnh trong không gian 3-chiều lên mặt phẳng 2-chiều thì ta cần thực hiện các phép chiếu để chiếu chúng lên mặt phẳng chiếu, hay nói một cách cụ thể là ta cần chiếu các điểm đầu mút của các cạnh lên mặt phẳng chiếu rồi sau đó nối chúng lại

Kỹ thuật chung để vẽ một đoạn thẳng 3D:

- + Chiếu mỗi điểm đầu mút thành từng điểm 2-chiều
- + Vẽ đoạn thẳng nối hai điểm chiếu ta được đoạn thẳng chiếu

điều này làm được bởi vì các phép chiếu nói chung bảo toàn đường thẳng. Vấn đề còn lại là tìm hình chiếu của một điểm trong không gian 3-chiều lên mặt phẳng chiếu. Ta xem xét hai phép chiếu dưới đây

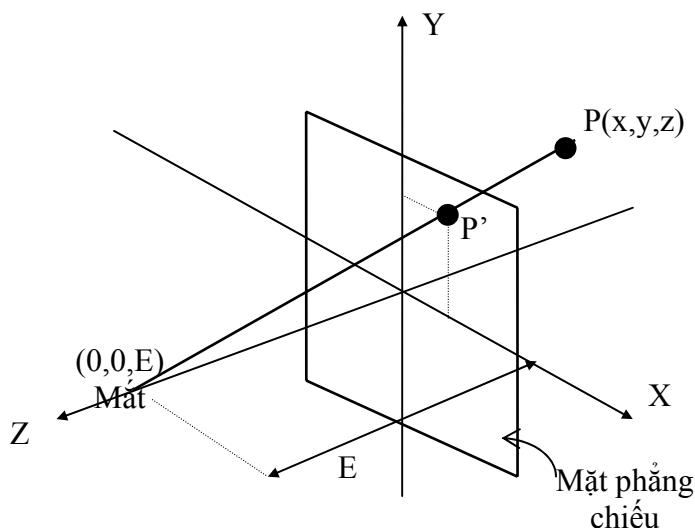
### II.1. Phép chiếu trực giao đơn giản

Chiếu một điểm P(x,y,z) lên một mặt phẳng thì đơn giản nhất là chọn mặt phẳng chiếu là mặt phẳng toạ độ. Ví dụ như chiếu lên mặt phẳng OXY, khi đó ta chỉ việc bỏ đi thành phần z trong toạ độ của điểm và giữ nguyên các thành phần x,y ta được toạ độ của điểm chiếu trên mặt phẳng OXY.

### II.2. Phép chiếu phối cảnh đơn giản

Phép chiếu phối cảnh được thực hiện khá đơn giản, song nó cho ta một cái nhìn khá tự nhiên hơn về đối tượng 3D so với phép chiếu trực giao. Chúng ta đã làm quen với cách nhìn phối cảnh trong đời sống hàng ngày

Phép chiếu phối cảnh phụ thuộc vào vị trí tương đối của mắt và mặt phẳng chiếu. Ví dụ như trong hình sau:



số  $t=1/(1-z/E)$ . Thì số này thuộc vào khoảng cách tương đối  $E$  giữa mặt nhìn và mặt phẳng chiếu, và nó sẽ làm cho vật thể (hay thành phần) ở gần mặt thì có hình dáng to ra còn những vật thể (hay thành phần) ở xa thì có hình dáng nhỏ lại.

### II.2.a. Cài đặt thuật toán

Sau đây là chương trình minh họa việc sử dụng mô hình WireFrame để biểu diễn một vật thể là một hình lập phương, đồng thời chương trình còn ứng dụng kiến thức của chương **Các phép biến đổi hình học**, để cho phép người quan sát có thể quan sát vật thể ở mọi góc cạnh nhờ chương trình đã thiết lập hệ quan sát và cho phép người sử dụng quay hệ quan sát bằng cách dùng các phím mũi tên.

```

Program View_3D_Object;
{Chương trình quan sát đối tượng 3D đơn giản}
uses graph, crt;
const maxdinh=100;
      maxcanh=200;
      gocdt:real=35;
      gocphi:real=30;
      R:real=5;
      d:real=17;
type point_3d=record
      x,y,z:real;
    end;
    wireframe=record
      sodinh:1..maxdinh;
      dinh:array[1..maxdinh] of point_3d;
      socanh:1..maxcanh;
    end;
  
```

Như đã trình bày trong chương IV (Các phép biến đổi hình học) ta có hình chiếu phối cảnh  $P'$  của  $P$  lên mặt phẳng OXY có các thành phần tọa độ là:

$$x'=x/(1-z/E) \text{ và } y'=y/(1-z/E)$$

Ta dễ dàng thấy rằng phép chiếu phối cảnh gần giống với phép chiếu trực giao trước đây ngoại trừ  $x$  và  $y$  được nhân lên một hệ

nhé

```

canh:array[1..maxcanh] of record
    D1,D2:1..maxdinh;
end;
end;
var wi,wic: wireframe;
hesophong:integer;
Color:byte;
procedure KhoiTaoDoHoa;
{Khởi tạo chế độ đồ họa}
var gd,gm,errorcode:integer;
begin
    gd:=detect;
    initgraph(gd,gm,'c:\bp\bgi\'');
    errorcode:=graphresult;
    if errorcode<>grok then
        writeln('graphics error');
end;
Procedure KhoiTaoGhe(var Wi:WireFrame);
{Khởi tạo dữ liệu mô tả về chiếc ghế tựa đã đà cập ở trên vào biến Wi}
begin
With Wi do
begin
    SoDinh:=12;
    SoCanh:=10;
    Dinh[1].x:= 1; Dinh[1].y:= 1; Dinh[1].z:=-3;
    Dinh[2].x:=-1; Dinh[2].y:= 1; Dinh[2].z:=-3;
    Dinh[3].x:=-1; Dinh[3].y:=-1; Dinh[3].z:=-3;
    Dinh[4].x:= 1; Dinh[4].y:=-1; Dinh[4].z:=-3;
    Dinh[5].x:= 1; Dinh[5].y:= 1; Dinh[5].z:= 0;
    Dinh[6].x:=-1; Dinh[6].y:= 1; Dinh[6].z:= 0;
    Dinh[7].x:=-1; Dinh[7].y:=-1; Dinh[7].z:= 0;
    Dinh[8].x:= 1; Dinh[8].y:=-1; Dinh[8].z:= 0;
    Dinh[9].x:= 1; Dinh[9].y:=-1; Dinh[9].z:= 2;
    Dinh[10].x:=-1; Dinh[10].y:=-1; Dinh[10].z:= 2;
    Dinh[11].x:= 1; Dinh[11].y:=-1; Dinh[11].z:= 3;
    Dinh[12].x:=-1; Dinh[12].y:=-1; Dinh[12].z:= 3;
    Canh[1].D1:=1; Canh[1].D2:=5;
    Canh[2].D1:=2; Canh[2].D2:=6;
    Canh[3].D1:=3; Canh[3].D2:=12;
    Canh[4].D1:=4; Canh[4].D2:=11;
    Canh[5].D1:=5; Canh[5].D2:=6;
    Canh[6].D1:=6; Canh[6].D2:=7;
    Canh[7].D1:=7; Canh[7].D2:=8;
    Canh[8].D1:=8; Canh[8].D2:=5;
    Canh[9].D1:=9; Canh[9].D2:=10;
    Canh[10].D1:=11; Canh[10].D2:=12;
end;
end;
procedure ChuyenSangHeTrucQuanSat(wi:wireframe;var
                                    wic:wireframe);

```

---

{Chuyển thông tin mô tả đổi tượng từ hệ trục OXY sang hệ trục Quan sát O'UVN, thông tin mô tả đổi tượng trong hệ trục Quan sát được chứa trong biến Wic}

```

var i:byte;
gdt,gphi:real;
begin
    wic:=wi;
    {Chuyển đơn vị đo góc sang Radian để sử dụng cho các hàm Sin và Cos}
    gdt:=gocdt/180*pi;
    gphi:=gocphi/180*pi;

for i:=1 to wi.sodinh do
    with wi.dinh[i] do
        begin
            wic.dinh[i].x:=-x*sin(gdt)+y*cos(gdt) ;
            wic.dinh[i].y:=-x*cos(gdt)*sin(gphi)-y*sin(gdt)*sin(gphi)+z*cos(gphi);
            wic.dinh[i].z:=-x*cos(gdt)*cos(gphi)-y*sin(gdt)*cos(gphi)-z*sin(gphi)+R;
        end;
end;
procedure ChieuTrucGiao_Ve(wic:wireframe;hesophong:integer);
{Thực hiện chiếu trực giao lên mặt phẳng OXY rồi ánh xạ lên màn hình}
var i:byte;
    xg,yg:integer;
    dx:real;
begin
    xg:=getmaxx div 2;
    yg:=getmaxy div 2;
    with wic do
        for i:=1 to socanh do
            begin
                setcolor(color);
                line(round(dinh[canh[i].D1].x*hesophong)+xg,
                    -round(dinh[canh[i].D1].y*hesophong)+yg,
                    round(dinh[canh[i].D2].x*hesophong)+xg,
                    -round(dinh[canh[i].D2].y*hesophong)+yg);
            end;
    end;
procedure xoay;
{Xoay quanh vật thể để quan sát được tại các vị trí khác nhau, bằng cách thay đổi
các góc θ và góc φ khi người sử dụng bấm các phím mũi tên}
var ch:char;
begin
repeat
    ch:=readkey;
    if ord(ch)=0 then
        begin
            ch:=readkey;
            case ch of
                #72:if gocphi>0 then gocphi:=gocphi-1

```

```

        else gocphi:=360;
#80;if gocphi>360 then gocphi:=gocphi+1
        else gocphi:=1;
#77;if gocdt>0 then gocdt:=gocdt-1
        else gocdt:=360;
#75;if gocdt<360 then gocdt:=gocdt+1
        else gocdt:=1;
end;
if ord(ch) in [72,75,77,80] then
begin
    {Xóa hình cũ bằng cách vẽ màu Black (màu Đen)}
    color:=Black;
    ChieuTrucGiao_Ve(wic,hesophong);
    {Vẽ hình mới bằng màu White (màu Trắng)}
    color:=White;
    ChuyenSangHeTrucQuanSat(wi,wic);
    ChieuTrucGiao_Ve(wic,hesophong);
end;
end
until ch=#27;
end;
BEGIN
KhoiTaoGhe(Wi);
KhoiTaoDoHoa;hesophong:=25;
setcolor(2);
outtextxy(30,30,'Press Across key to change View');
ChuyenSangHeTrucQuanSat(wi,wic);
color:=15;
ChieuTrucGiao_Ve(wic,hesophong);
xoay;
closegraph;
END.
```

### III. Bài tập cuối chương

- ◆ **Hướng dẫn nâng cao:** Sinh viên cần nâng cấp chương trình trên bằng việc xây dựng các mô-đun sau:
  - + Xây dựng chương trình con cho phép đọc thông tin mô tả đối tượng trong một file text với cấu trúc dạng:
    - Dòng đầu tiên: chứa tên của đối tượng
    - Dòng thứ 2: chứa 2 số nguyên là số đỉnh và số cạnh của đối tượng
    - Dòng thứ 3 trở đi mỗi dòng chứa 3 số thực lần lượt là 3 thành phần tọa độ x,y,z của đỉnh

- Tiếp theo là các dòng chứa toạ độ đỉnh là các dòng chứa thông tin mô tả về các cạnh, mỗi dòng gồm 2 số nguyên là thứ tự của hai đỉnh tạo nên cạnh.

Ví dụ thông tin mô tả chiếc ghế trên sẽ được mô tả trong file text như sau:

Hình cai ghe tua

12 10

1	1	-3
-1	1	-3
-1	-1	-3
1	-1	-3
1	1	0
-1	1	0
-1	-1	0
1	-1	0
1	-1	2
-1	-1	2
1	-1	3
-1	-1	3
1	5	
2	6	
3	12	
4	11	
5	6	
6	7	
7	8	
8	5	
9	10	
11	12	

+ Xây dựng chương trình con cho phép thực hiện chiếu phổi cảnh rồi ánh xạ lên màn hình thay thế cho chương trình con *ChieuTrucGiao\_Ve* dùng phép chiếu trực giao. Ở đây chúng ta dùng công thức chiếu phổi cảnh đã tìm ra trong chương **các phép biến đổi hình học** mục *Quan sát vật thể 3D & Quay hệ quan sát*, đó là công thức:

$$u' = D * u / n; \quad v' = D * v / n$$

+ Nâng cấp chương trình để người sử dụng có thể dùng các phím S và P để chuyển từ phép chiếu *trực giao* sang phép chiếu *phổi cảnh* và ngược lại.

# **Chương VII: Mô hình các mặt đa giác & Vấn đề khử mặt khuất**



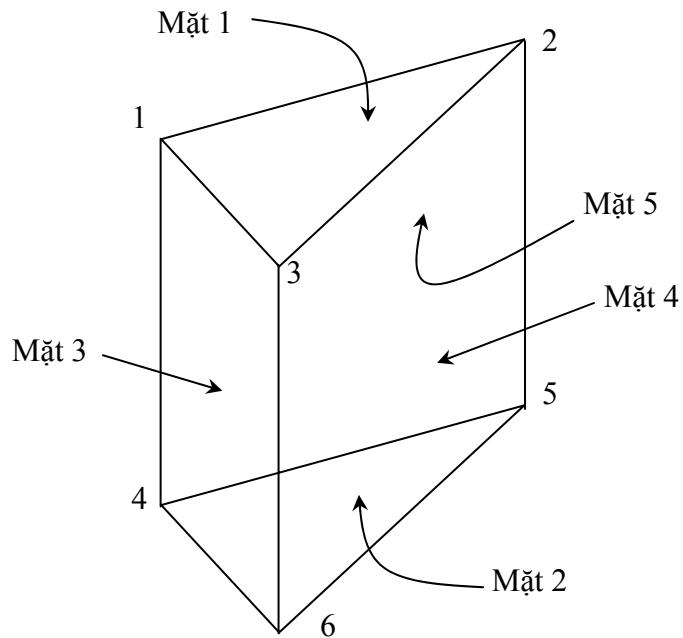
## **I. Các khái niệm chung**

### **I.1. Cấu trúc vật thể bằng mô hình "các mặt đa giác"**

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)

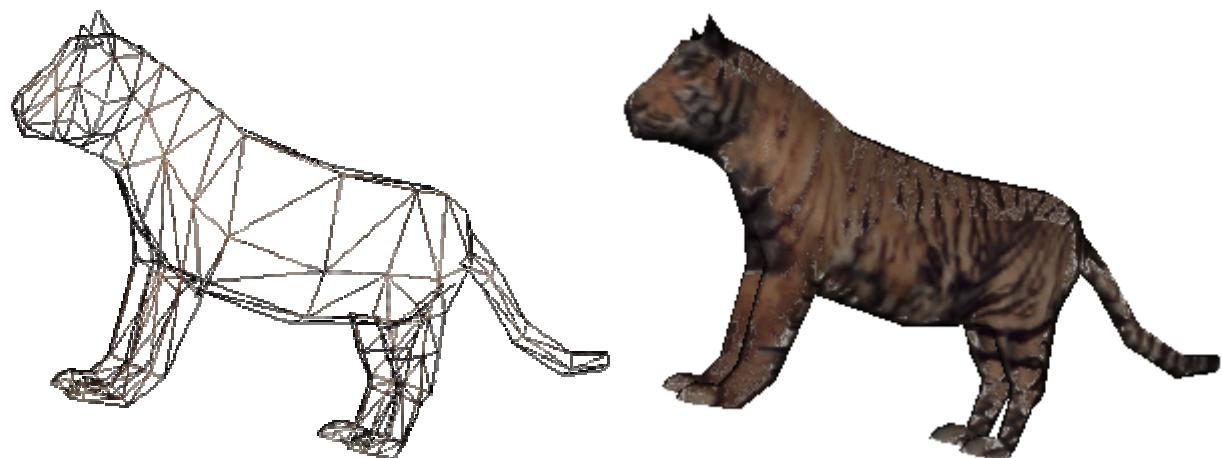
- Mô hình WireFrame: Đã trình bày ở chương trước, nó cho ta hình dáng của vật thể dưới dạng một bộ khung
- Mô hình các mặt đa giác: Ở đây một vật thể 3D được xác định thông qua các mặt (thay vì các cạnh như trong mô hình WireFrame), và mỗi một mặt lại được xác định thông qua các điểm mà các điểm này 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.

Ví dụ: Mô tả vật thể như trong hình vẽ sau:



- 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



(a) Một con Cọp với các mặt đa giác chưa tô màu. (b) Tô màu đa giác theo m�



Một nhân vật game theo mô hình các mặt đa giác cùng với phép ánh xạ bè mặt vật liệu lên các đa giác.

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:

```
Point3D=record  
    x,y,z:real;  
end;  
Vector3D=record  
    x,y,z:real;  
end;  
RGBColor=record
```

*{Điểm 3 chiều}*

*{Vector 3 chiều. Mặc dù nó giống với Point3D song ta vẫn khai để các thuật toán được tường minh}*

*{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
  ObjName:string;           {Đối tượng 3 chiều}
  Sodinh:cardinal;          {Tên của đối tượng}
  Dinh: array of point3d;   {Số đỉnh}
  SoMat:cardinal;           {Danh sách đỉnh. Ở đây ta dùng kiểu
                           mảng động}
  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 mọc 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 này 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 tùy 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.

## II. Các phương pháp khử mặt khuất

### II.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 3 chiều có tới 3 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.

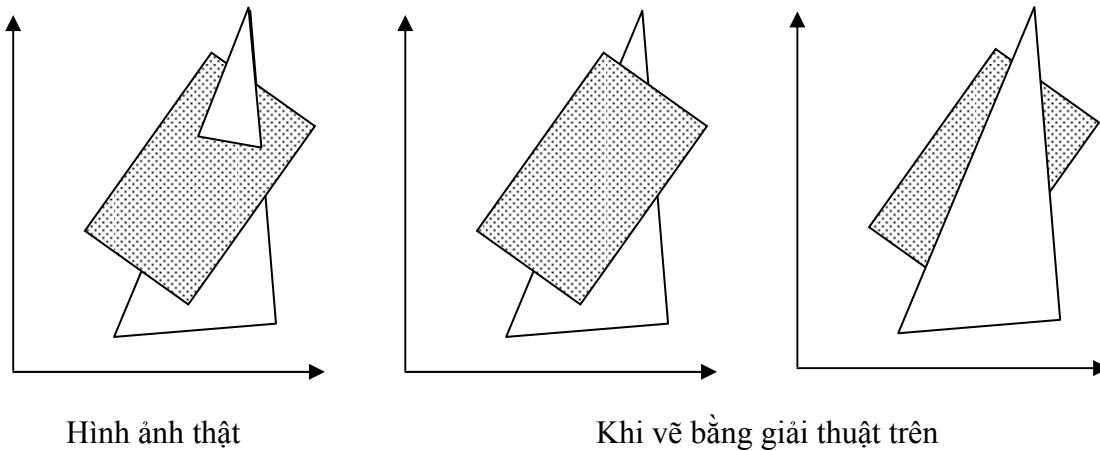
Từ những kinh nghiệm trong thực tế, người ta cho rằng nên sử dụng giá trị z trung bình sẽ cho kết quả tốt trong hầu hết các trường hợp.

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.

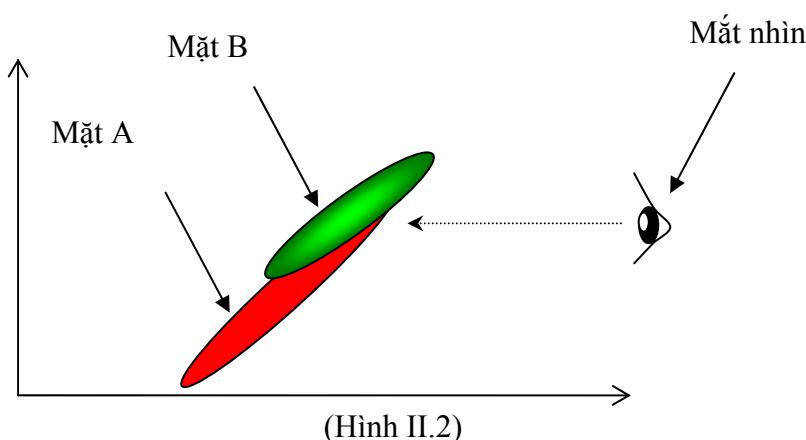
Như vậy, thuật giải Depth-Sorting được thực hiện một cách dễ dàng khi chúng ta xác định một giá trị độ sâu (là giá trị z trong hệ toạ độ quan sát) đại diện cho cả mặt. Các mặt dựa vào độ sâu đại diện của mình để so sánh rồi sắp xếp theo một danh sách giảm dần (theo độ sâu đại diện). Bước tiếp theo là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

Giải thuật còn một số vướng mắc sau:

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



- ☒ Khi hai mặt là ở trong cùng một khoảng không gian về độ sâu và hình chiếu của chúng lên mặt phẳng chiếu là chồng lên nhau (hay chồng một phần lên nhau). Ví dụ:



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 được nhưng cũng có 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.

Dể 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. Nếu các đỉnh này đều nằm về cùng một phía của đa giác A theo chiều trực Z hay không, nếu đúng thì kết quả trật tự trên là đúng. Ngược lại, có thể xảy ra một trong hai tình huống như hình (II.1) và (II.2), 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 II.1) hoặc chéo vào nhau, 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.

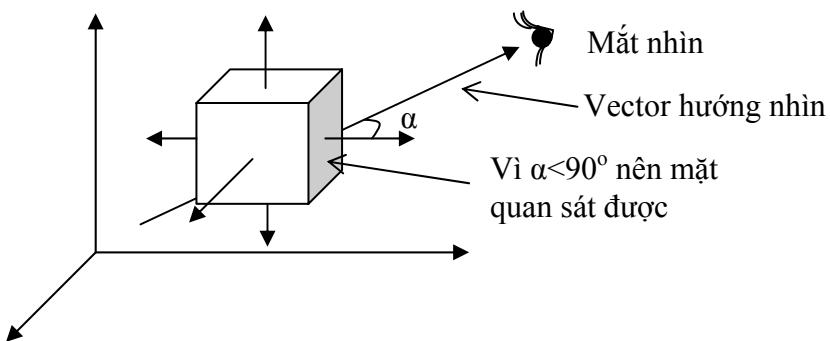


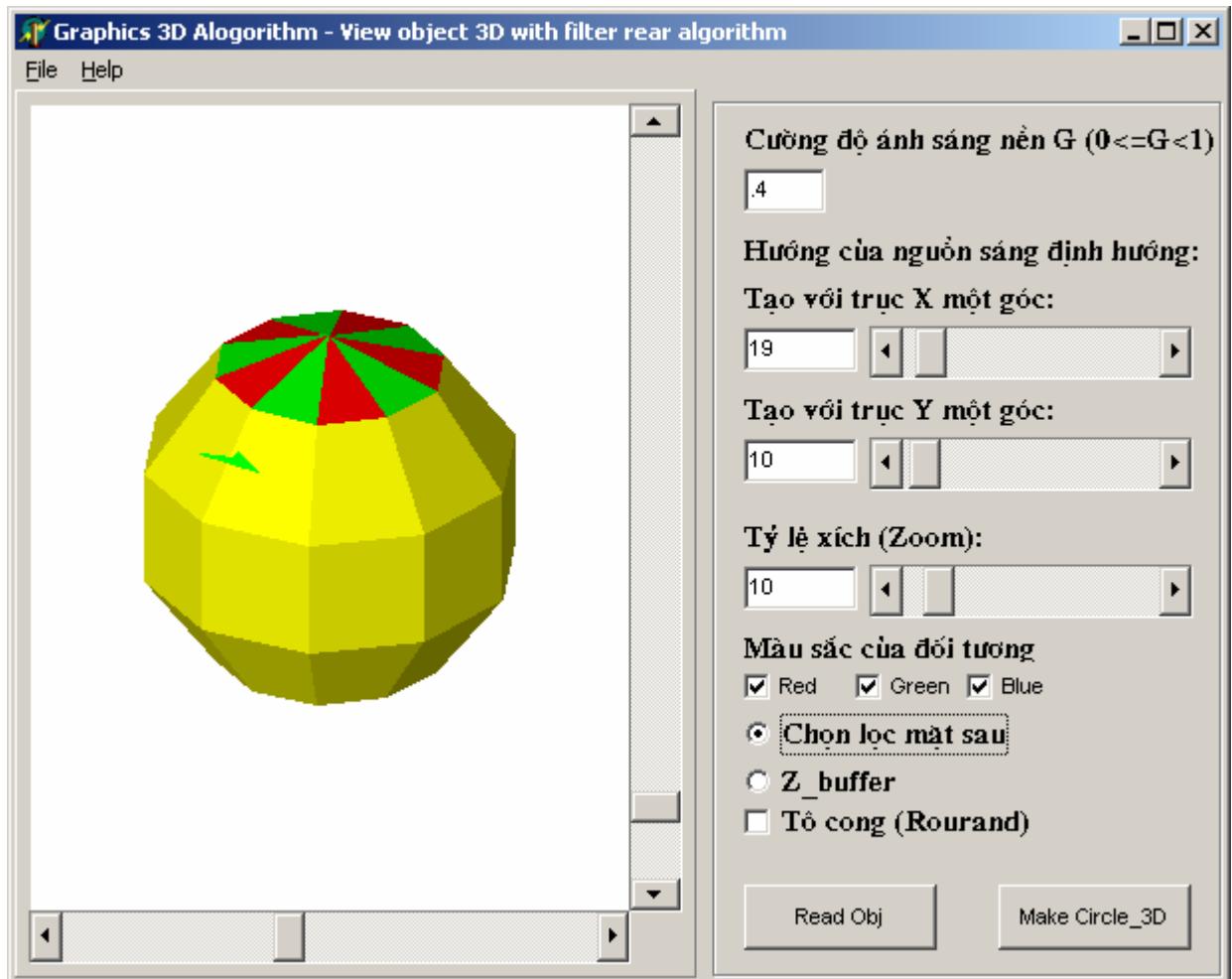
## II.2. Thuật toán chọn lọc mặt sau

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  $90^\circ$  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  $90^\circ$ . Vậy thuật toán để xét một mặt bị khuất hay không chỉ đơn giản là:

**If  $V.N \geq 0$  then Mặt thấy  
Else Mặt không thấy (mặt khuất)**





### II.2.1 Cài đặt minh họa cho thuật toán chọn lọc mặt sau

```
Function Tich_vo_huong(v,n:Vector3D):real;
{Tính tích vô hướng của 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*
- + *Zoom: 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.B);
        Canvas.Polygon(poly);      {Tô đa giác với màu đã được thiết lập}
      end;
    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 thỏa mãn điều kiện đó thì chúng ta phải áp dụng một thuật toán khác hay có những sửa đổi cần thiết để tránh sự thể hiện sai lạc.

### II.3. Thuật toán 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 buộc 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  $100 \times 100$ . 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 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 cả 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 một số vướng mắc đó là:

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  $P_1, P_2, \dots, P_n$  và  $G'$  là hình chiếu của G xuống mặt phẳng chiếu với tập các đỉnh  $P'_1, P'_2, \dots, P'_n$ .

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 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

$$z(M') = z(P') + (z(Q') - z(P'))t \quad (\text{II.3.1})$$

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. Đó là ta cho một dòng quét chạy ngang qua đa giác từ đỉnh đến đáy 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 (II.3.1) như sau:

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

$$z(x_i) = z(P_i') + [z(P_j') - z(P_i')] * [(y_{\text{scan}} - y(P_i')) / (y(P_j') - y(P_i'))] \quad (\text{II.3.2})$$
  
 {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  $z(x_i) = z(P_i') + z(P_j') * 1/2$

### II.3.a. Cài đặt minh họa cho thuật toán “vùng đệm độ sâu”

Từ những phân tích 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}

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}

<pre> NutPoly_Z=record   chiêu G';     x,y:Integer;     z:real;   end; </pre>	<p>Cấu trúc của một đỉnh của đa giác chiều <math>G'</math></p> <p>Toạ độ của ảnh trên mặt phẳng chiều</p> <p>Thành phần độ sâu đi kèm (là độ sâu của tạo ảnh)</p>
<pre> Polygon_Z=array of NutPoly_Z; </pre>	<p>Đa giác chiều là một mảng động. Như một đa giác 2 chiều, song mỗi một đỉnh có chứa thêm thành phần độ sâu của đỉnh</p>

```

CanhCat_Z=record
    y1,y2:Integer;
    cạnh (y1<=y2)
    xGiao:real;
    xStep:real;
    zGiao:real;
    zStep:real;
end;

```

*{Cấu trúc của các cạnh đa giác được xây dựng nhằm phục vụ cho quá trình tính giao điểm}*

*{Tung độ bắt đầu và kết thúc của một hành độ xuất phát của cạnh. Song trong quá trình tính toán nó sẽ là tung độ giao điểm của cạnh với đường quét ngang}*

*{Giá trị thay đổi của x khi y thay đổi 1 đơn vị, nó cho biết độ dốc của cạnh}*

*{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}*

*{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}*

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:* + Đôi 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 không gian 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 đôi 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 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=429496729

5

{Giá trị độ sâu của đỉnh Poly[i] là giá trị Obj.dinh[P].z song được ánh xạ vào không gian nguyên 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
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 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;
            end;
        Procedure ToMauCacDoan;

```

{Thực hiện tô màu các đoạn thẳng là phần giao của đường quét với đa giác. Đó là các đoạn  $x_1x_2, x_3x_4, \dots$ }

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_BufferW+1:Chiều rỗng (từ
Z_BufferH:=high(Z_Buffer[0]);   {Z_BufferW+1:Chiều rỗng (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 toMaxY do {Cho dòng quét chạy từ MinY đến MaxY }
begin
  TaoDanhSachGiaoDiem;      {Tim 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;

```

### III. Bài tập cuối chương

#### 1. Cài đặt cho thuật giải Depth-Sorting

Cài đặt chương trình cho phép biểu diễn và quan sát vật 3D thể 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 3D thể 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 3D thể 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: Các mô hình chiếu sáng



## I. 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.

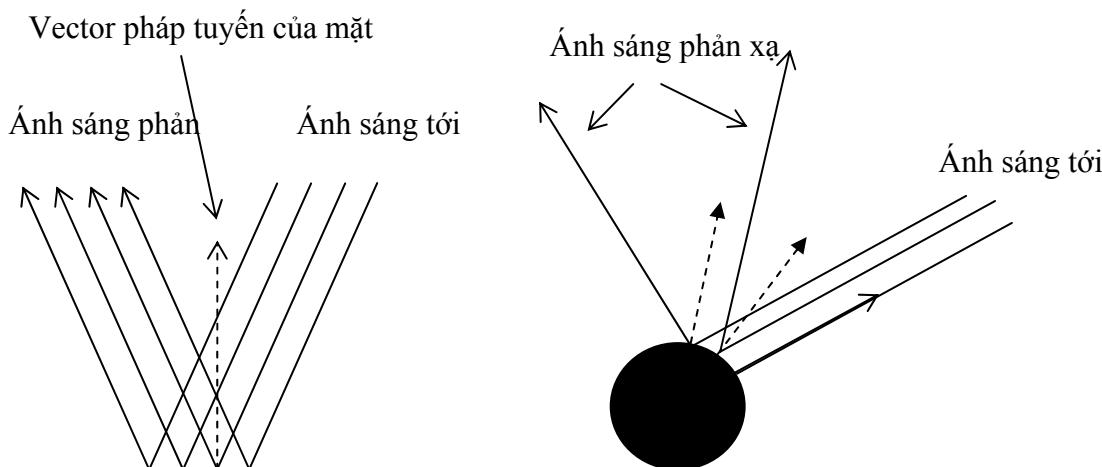
## II. Nguồn sáng xung quanh

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.

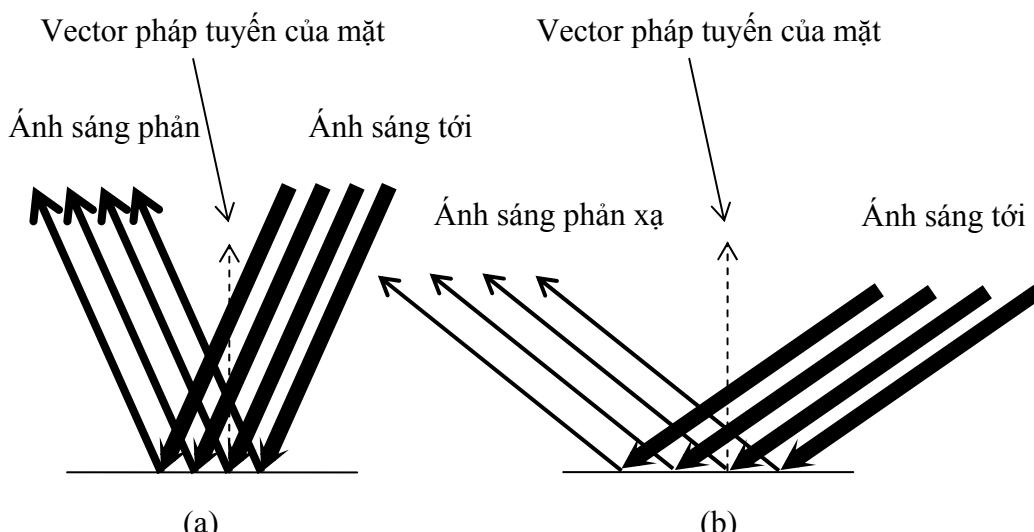
## III. Nguồn sáng định hướng

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 III.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 III.1 Sự phản xạ của ánh sáng trên các bề mặt



Hình III.2 Sự phản xạ không toàn phần của ánh sáng

Trong hình III.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 mờ nhạt 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 III.2 (a)), nếu góc này lớn thì cường độ phản xạ rất thấp (hình III.2 (b)). Ở đây ta sẽ quan tâm 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  $\theta$  là góc giữa tia tới với vector pháp tuyến của bề mặt thì  $\cos(\theta)$  phụ thuộc vào tia tới  $a$  và vector pháp tuyến của mặt  $n$  theo công thức:

$$\cos(\theta) = \frac{\vec{a} \cdot \vec{n}}{|\vec{a}| \cdot |\vec{n}|} \quad (\text{III.1})$$

Trong công thức trên  $\cos(\theta)$  bằng tích vô hướng của  $a$  và  $n$  chia cho tích độ lớn của chúng. Nếu ta đã chuẩn hóa 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:

$$\cos(\theta) = \text{tích vô hướng của } a \text{ và } n = a.x*n.x + a.y*n.y + a.z*n.z$$

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

$$\text{Cường\_độ\_ánh\_sáng\_phản\_xạ} := \text{cường\_độ\_của\_ánh\_sáng\_định\_hướng} * [(\cos(\theta)+1)/2] \quad (\text{III.2})$$

Trong đó  $[(\cos(\theta)+1)/2]$  có giá trị trong khoảng từ 0 đến 1. Vậy qua công thức (III.1) và (III.3) 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.

### III.1. 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
  - + Zoom: 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  $\leq 1$ . 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 thỏa 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) $\geq 0$  then {Nếu mặt là khả kiến (không bị khuất) thì}
    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 trực 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;

## IV. 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 tọa độ là  $(x_0, y_0, z_0)$  và nguồn sáng điểm có tọa độ là  $(plx, ply, plz)$  thì ánh sáng sẽ rời đến điểm đang xét theo vector  $(x_0-plx, y_0-ply, z_0-plz)$ , hay tia tới:

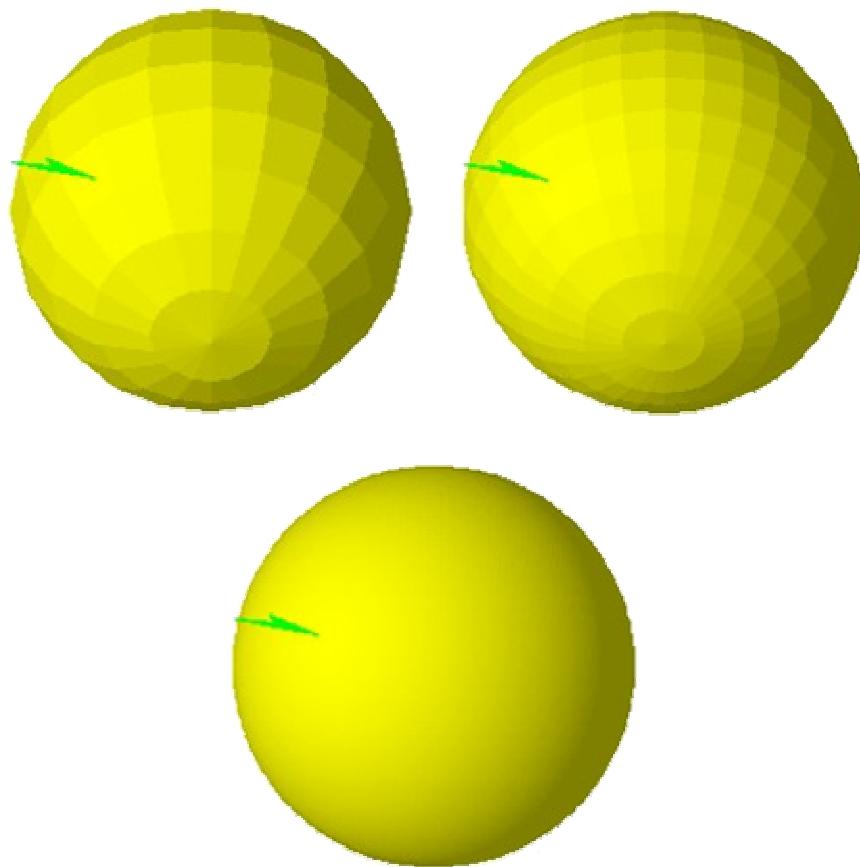
$$a = (plx - x_0, ply - y_0, plz - z_0).$$

Và từ đó cường độ sáng tại điểm đang xét sẽ phụ thuộc vào  $\cos(\theta)$  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  $\cos(\theta)$  thông qua công thức (II.1) với tất cả các điểm trên mặt. Nên nhớ rằng trong tình huống nguồn sáng điểm thì chúng ta buộc lòng phải tính  $\cos(\theta)$  thông qua công thức (II.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).

## **V. 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 tròn trịa hơn, mịn và cong hơn. Tuy nhiên, khi số đa giác xấp sỉ 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 sấp sỉ 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 III.1). 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.



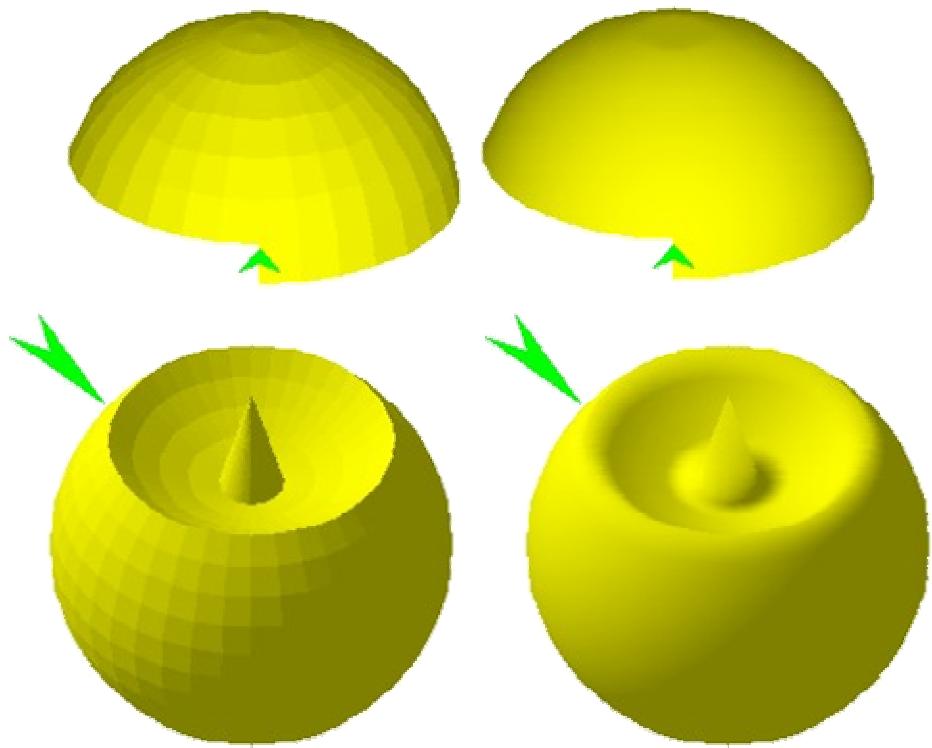
Hình V: (200 mặt)

(450 mặt)

(16200 mặt)

Xong 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 gồ ghề không 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.

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.

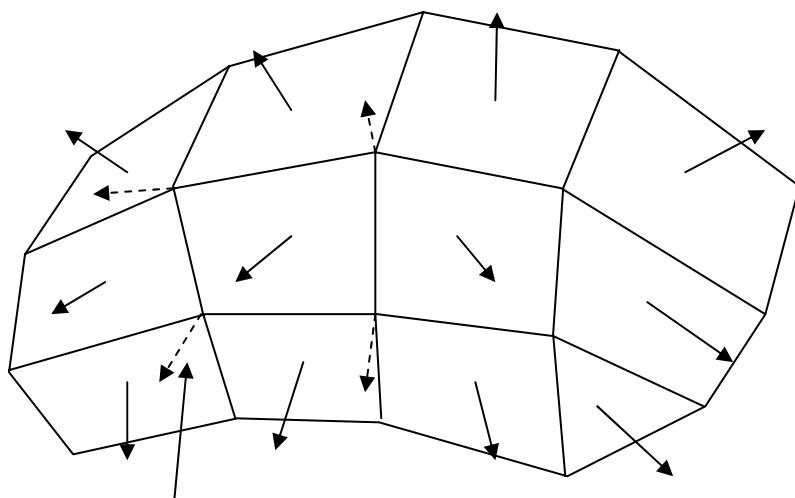


Hình III.2: (Tô bóng thường)

(Tô bóng theo Gouraud)

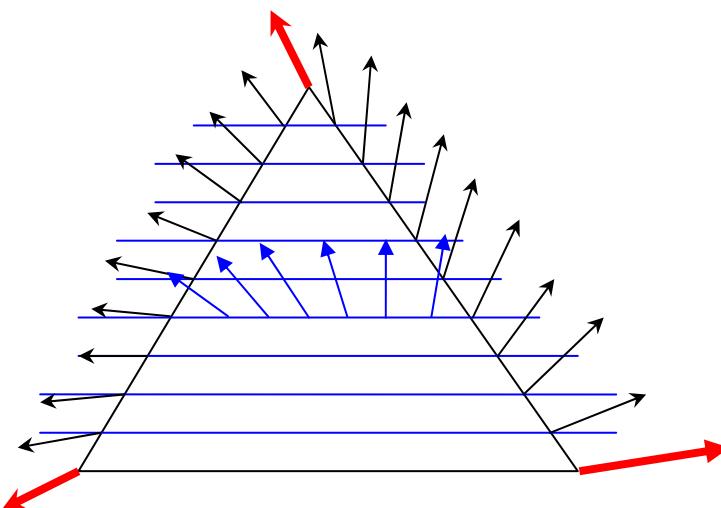
Thường thì mỗi mặt đa giác có một vector pháp tuyến, và như phần trên đã trình bày, vector pháp tuyến đó được dùng để tính cường độ của ánh sáng phản xạ trên bề mặt của đa giác từ đó suy ra cường độ sáng của mặt. Tuy nhiên mô hình Gouraud lại xem một đa giác không chỉ có một vector pháp tuyến, mà mỗi 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 dựa vào vector đó mà 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.

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.



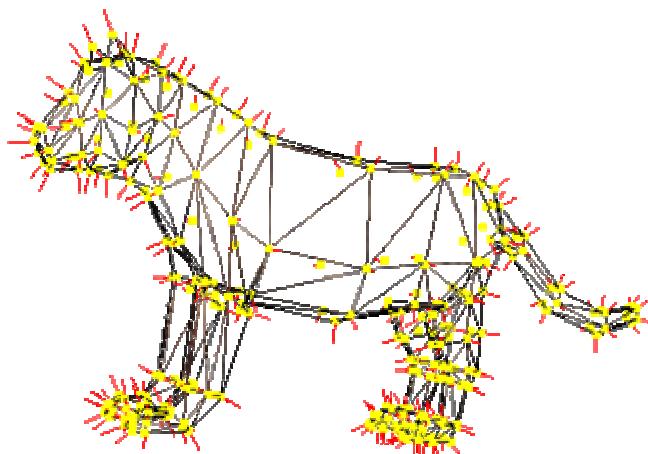
Vector trung bình cộng bằng trung bình cộng của các vector pháp tuyến lận cận

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.



Nội suy các vector pháp tuyến cho từng điểm trên mặt đa giác

Hiện nay, trong các tiện ích tạo mô hình, người ta cho phép chúng ta nhập và điều chỉnh hướng cho các vector nút tại các đỉnh của đa giác thay vì tính nội suy từ các vector pháp tuyến. Như hình minh họa sau:



## V.1. 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 xong 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
    N:Vector3D;
    x,y:Integer;
  end;
  PolygonGourand =array of NutPolyGourand; {mảng động dùng để chứa các đỉnh của đa giác chiếu}
  CanhCat=record
    y1,y2:Integer;
    xGiao:real;
    XStep:real;
    NGiao:Vector3D;
    NStep:Vector3D;
  end;
  DanhSachCanhCat=array of CanhCat;
  GiaoDiem=record {Cấu trúc chứa giao điểm của đường quét
    ngang với cạnh đa giác chiếu }
  end;
```

```

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
định}
    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 mặt 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 {Nếu mặt là khả kiến}
  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;
    end;

FillPolygonGourand(poly,VectorChieuSang,AnhSangNen,AnhSangDinhHuong,C
anVas,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\_Chiieu: 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_Cieu*
    
```

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;

## VI. Bài tập cuối chương

Bài 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.

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

Bài 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.



# Mục Lục

Chương I: Các yếu tố cơ sở của đồ họa.....	2
I. Các khái niệm cơ bản.....	2
I.1. Thiết bị đồ họa và điểm ảnh (Pixel).....	2
I.2. Điểm và Đoạn thẳng trong mặt phẳng .....	2
II. Các thuật toán vẽ đoạn thẳng .....	3
II.1. Vẽ đoạn thẳng dựa vào phương trình.....	3
II.2. Vẽ đoạn thẳng dựa vào thuật toán Bresenham.....	5
II.2.a. Tóm tắt thuật toán Bresenham: .....	9
II.2.b. Ví dụ: .....	9
II.2.c. Hướng dẫn cho các trường hợp hệ số góc ngoài khoảng [0,1] .....	11
II.2.d. Cài đặt thuật toán .....	12
III. Các thuật toán vẽ đường tròn .....	12
III.1. Thuật toán vẽ đường tròn MidPoint.....	13
III.1.a. Tóm tắt thuật toán vẽ đường tròn MidPoint : .....	15
III.1.b. Cài đặt .....	16
III.2. Thuật toán vẽ đường tròn Bresenham.....	17
III.2.a. Tóm tắt thuật toán vẽ đường tròn Bresenham : .....	18
III.2.b. Cài đặt .....	19
IV. Thuật toán vẽ Ellipse .....	19
IV.1. Thuật toán Bresenham cho vẽ hình Ellipse.....	20
IV.1.a. Tóm tắt thuật toán Bresenham cho vẽ Ellipse: .....	22
IV.1.b. Cài đặt thuật toán Bresenham cho dựng Ellipse .....	23
V. Bài tập cuối chương .....	25
Chương II: Các hệ màu & cơ chế tổ chức bộ nhớ Card màn hình .....	26
I. Đôi nét về cấu trúc màn hình màu .....	26
II. Các hệ màu.....	28
II.1. Hệ RGB.....	28
II.2. Hệ màu CMY .....	29
II.3. Hệ màu HSV .....	30
III. Cơ chế tổ chức bộ nhớ Card màn hình .....	35
III.1. Cơ chế hoạt động của chế độ màn hình 320x200x256 màu .....	35
III.1.a. Cài đặt .....	36
III.2. Cơ chế hoạt động của màn hình theo chuẩn Vesa .....	49
Chương III: Các phép xén hình & tô màu .....	50
I. Trường hợp F là một tập hữu hạn điểm .....	51
II. Trường hợp xén một đoạn thẳng vào một vùng hình chữ nhật của $R^2$ .....	51
II.1. Khi có một cạnh của hình chữ nhật song song với trục tọa độ .....	51
II.1.a. Thuật toán Cohen-Sutherland .....	52
II.1.a.i. Kết thúc giải thuật.....	55
II.1.a.ii. Sau đây là một hàm tính mã .....	55
II.1.a.iii. Cài đặt thuật toán .....	56
II.1.b. Thuật toán Chia nhị phân .....	56
II.1.b.i. Cài đặt thuật toán .....	57
II.1.c. Thuật toán Liang-Barsky .....	57
II.1.c.i. Cài đặt thuật toán .....	58
II.2. Khi 1 cạnh của hình chữ nhật tạo với trục hoành một góc $\alpha$ :.....	58

III.	Clipping một đa giác vào một vùng hình chữ nhật .....	59
III.1.	Giải thuật Sutherland – Hodgman.....	59
III.1.a.	Cài đặt thuật toán .....	60
III.1.b.	Nhược điểm thuật giải Sutherland-Hodgman và cách khắc phục .....	60
IV.	Một số thuật toán tô màu .....	60
IV.1.	Giải thuật vết dầu loang .....	60
IV.1.a.	Phương pháp đệ quy .....	61
IV.1.a.i.	Cài đặt thuật toán .....	62
IV.1.b.	Phương pháp không đệ quy .....	63
IV.1.b.i.	Cài đặt thuật toán .....	63
IV.2.	Thuật toán tô màu đa giác theo dòng quét (Scan-line Algorithm).....	65
IV.2.a.	Cài đặt thuật toán .....	68
V.	Bài tập cuối chương .....	68
Chương IV: Các phép biến đổi hình học .....		69
I.	Các phép biến đổi Affine 2D (2- chiều).....	69
I.1.	Phép tịnh tiến .....	69
I.2.	Phép đồng dạng .....	70
I.3.	Phép đối xứng .....	70
I.4.	Phép quay quanh gốc toạ độ .....	71
I.5.	Phép biến dạng ( <i>Twist Transformation</i> ) .....	71
I.6.	Toạ độ thuần nhất (Homogeneous Coordinates) .....	71
I.7.	Tổng hợp các phép biến đổi Affine .....	71
I.8.	Phép quay quanh điểm bất kỳ .....	73
I.9.	Các ví dụ minh họa .....	73
I.9.a.	Ví dụ 1: .....	73
I.9.b.	Ví dụ 2: .....	74
I.10.	Biến đổi hệ trực tọa độ (hay biến đổi ngược) .....	75
I.11.	Cài đặt .....	77
II.	Các phép biến đổi Affine 3D .....	77
II.1.	Các hệ trực tọa độ .....	77
II.2.	Các công thức biến đổi .....	78
II.2.a.	Phép tịnh tiến .....	78
II.2.b.	Phép biến đổi tỉ lệ .....	78
II.2.c.	Phép đối xứng .....	78
II.2.d.	Phép quay .....	79
III.	Các phép chiếu vật thể trong không gian lên mặt phẳng .....	79
III.1.	Phép chiếu phôi cảnh (Perspective) .....	79
III.2.	Phép chiếu song song .....	81
IV.	Quan sát vật thể 3D & Quay hệ quan sát .....	81
IV.1.	Phép chiếu phôi cảnh .....	85
IV.2.	Phép chiếu song song .....	85
IV.3.	Cài đặt .....	86
IV.4.	Ví dụ minh họa .....	86
V.	Bài tập cuối chương .....	86
Chương V: Các phương pháp dựng đường cong và mặt cong .....		87
I.	Đường cong Bezier & mặt cong Bezier .....	87
I.1.	Thuật toán de Casteljau:.....	88
I.2.	Dạng BERNSTEIN của các đường cong BEZIER .....	88
I.3.	Dạng biểu diễn ma trận của đường Bezier.....	89
I.4.	Các tính chất của đường cong BEZIER .....	90

I.4.a.	Nội suy được các điểm đầu và điểm cuối: .....	90
I.4.b.	Tính bất biến Affine:.....	90
I.4.c.	Tính chất của bao lồi.....	91
I.4.d.	Độ chính xác tuyênlính: .....	91
I.4.e.	Bất biến với những phép biến đổi Affine: .....	92
I.4.f.	Đạo hàm của các đường Bezier: .....	92
I.5.	Đánh giá các đường cong Bezier & sự khác biệt của các đường cong Spline: .....	92
II.	Đường cong Spline và B-Spline: .....	93
Chương VI: Mô hình WireFrame.....		96
I.	Mô hình Wireframe: .....	97
II.	Vẽ hình dựa trên dữ liệu kiểu WireFrame với các phép chiếu .....	99
II.1.	Phép chiếu trực giao đơn giản.....	99
II.2.	Phép chiếu phối cảnh đơn giản .....	99
II.2.a.	Cài đặt thuật toán .....	100
III.	Bài tập cuối chương .....	103
Chương VII: Mô hình các mặt đa giác & Ván đè khử mặt khuất .....		106
I.	Các khái niệm chung.....	106
I.1.	Cấu trúc vật thể bằng mô hình "các mặt đa giác" .....	106
II.	Các phương pháp khử mặt khuất .....	109
II.1.	Giải thuật người thợ sơn và sắp xếp theo chiều sâu (Depth-Sorting) .....	109
II.2.	Thuật toán chọn lọc mặt sau .....	112
II.3.	Thuật toán vùng đệm độ sâu (Z-Buffer) .....	114
II.3.a.	Cài đặt minh họa cho thuật toán “vùng đệm độ sâu” .....	116
III.	Bài tập cuối chương .....	123
Chương VIII: Các mô hình chiếu sáng .....		125
I.	Khái niệm.....	125
II.	Nguồn sáng xung quanh.....	125
III.	Nguồn sáng định hướng.....	125
III.1.	Cài đặt thuật toán .....	127
IV.	Nguồn sáng điểm .....	129
V.	Mô hình bóng Gouraud.....	130
V.1.	Cài đặt thuật toán .....	134
VI.	Bài tập cuối chương .....	140
Mục Lục .....		142