TRƯỜNG ĐẠI HỌC KỸ THUẬT – CÔNG NGHỆ CẦN THƠ

KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ HỌA MÁY TÍNH

MÃ SỐ HỌC PHẦN: TT083

SỐ TÍN CHỈ HỌC PHẦN: 03 TÍN CHỈ

SỐ TIẾT HỌC PHẦN: 30 TIẾT LÝ THUYẾT, 30 TIẾT THỰC HÀNH

Cần Thơ, năm 2023

Chương 2

CÁC ĐỐI TƯỢNG ĐỒ HỌA CƠ SỞ

1. Hệ tọa độ thế giới thực, hệ tọa độ thiết bị và hệ tọa độ chuẩn
2. Vẽ đoạn thẳng
3. Vẽ đường tròn
4. Vẽ đa giác và các đường cong khác
5. Sử dụng OpenGL vẽ các đối tượng hình học cơ bản

1 HỆ ĐỒ HỌA

- Các đối tượng trong thế giới thực được mô tả bằng tọa độ thực. Trong khi đó, hệ tọa độ thiết bị lại sử dụng hệ tọa độ nguyên để hiển thị các hình ảnh.
- Các thiết bị khác nhau có định nghĩa hiển thị khác nhau.

➔**Đây chính là vấn đề cơ bản cần giải quyết.**

Đồ họa là làm thế nào để có thể mô tả và **biến đổi** được các đối tượng trong thế giới thực trên máy tính.

1 HỆ ĐỒ HỌA

➔**Phương pháp chuyển đổi tương ứng giữa các hệ tọa độ và đối tượng phải được định nghĩa bởi các thành phần đơn giản như thế nào để có thể mô tả gần đúng với hình ảnh thực bên ngoài.**

1 HỆ ĐỒ HỌA

Một hệ tọa độ đồ họa được mô tả bao gồm 3 miền:

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

1

Hệ tọa độ thế giới thực

Hệ tọa độ thiết bị

Hệ tọa độ thiết bị chuẩn

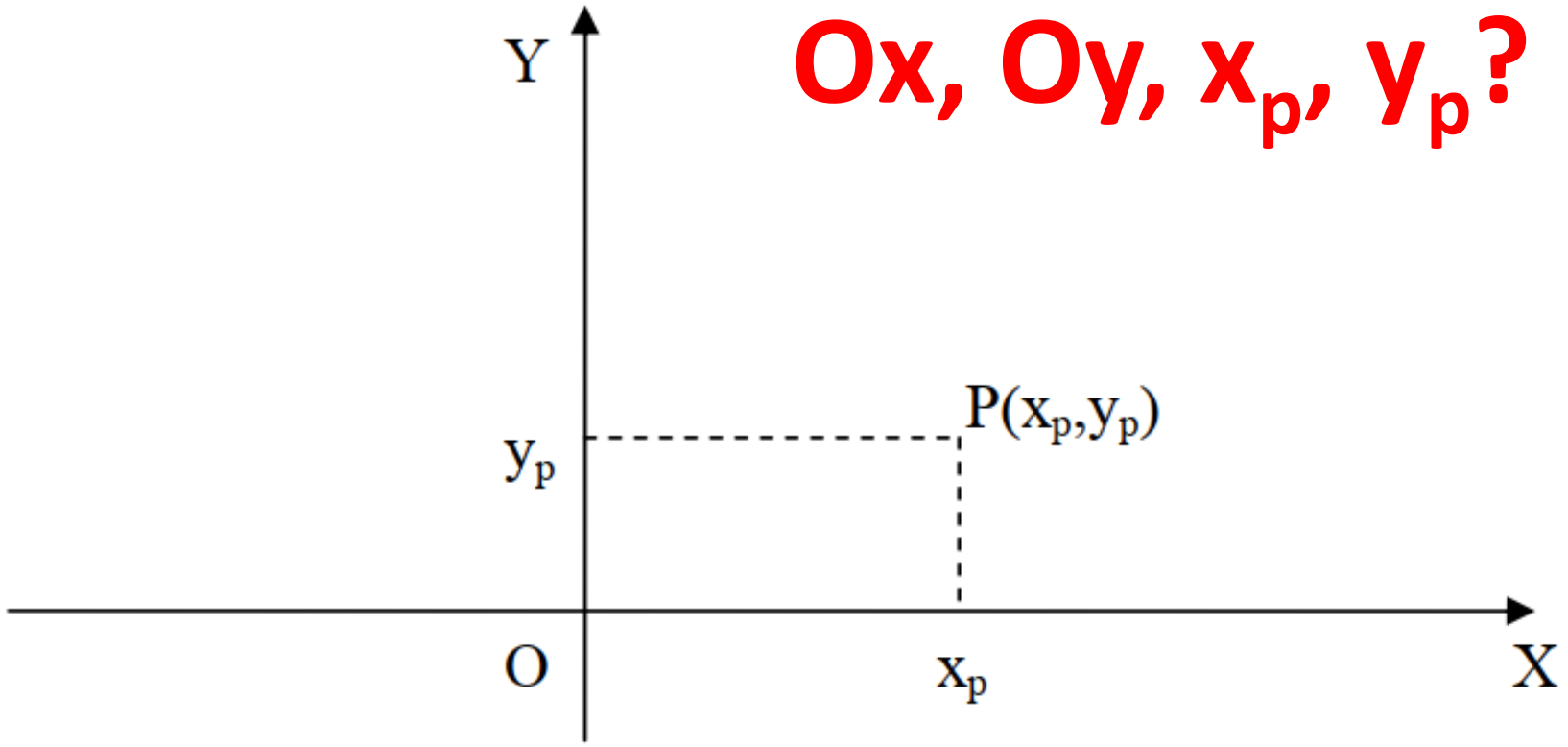
1 HỆ TỌA ĐỘ THẾ GIỚI THỰC

- Một trong những hệ tọa độ thực thường được dùng để mô tả các đối tượng trong thế giới thực là hệ tọa độ **Descartes**.
- Mỗi điểm P được biểu diễn bằng một cặp tọa độ (x_p,y_p) với x_p, y_p thuộc **R**

1 HỆ TỌA ĐỘ THẾ GIỚI THỰC

- **Descartes**

Ox, Oy, x_p, y_p?



Hình 2.1. Hệ toạ độ thực

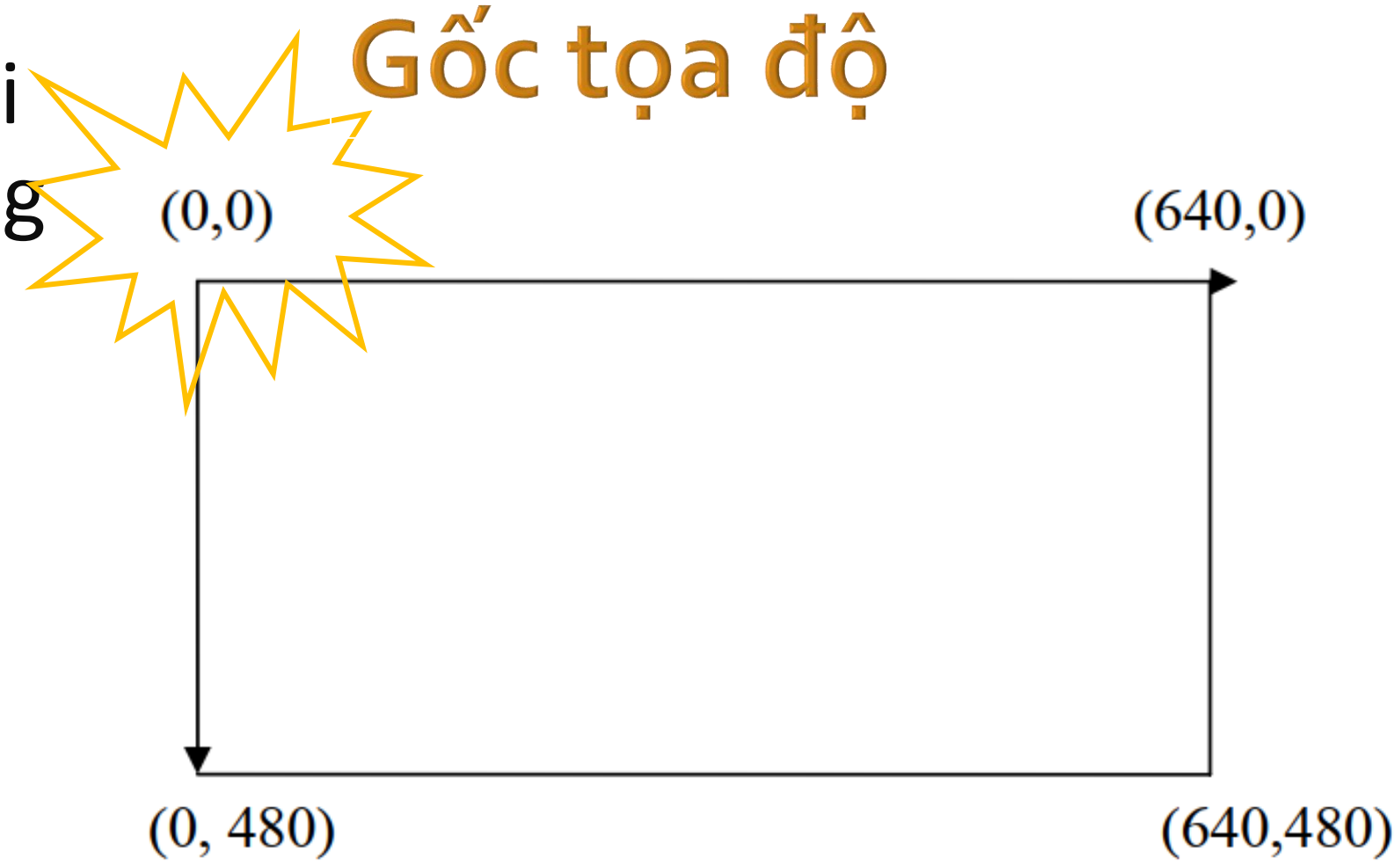
1 HỆ TỌA ĐỘ THIẾT BỊ (Device coordinates)

- Dùng cho một thiết bị xuất cụ thể nào đó, ví dụ như máy in, màn hình,…
- **Điểm** là thành phần cơ sở được định nghĩa trong một hệ toạ độ. Trong hệ toạ độ 2 chiều, điểm được mô tả bởi cặp tọa độ **(x,y).**
- **Điểm trong hệ toạ độ thiết bị** là rời rạc và thuộc *giới hạn* nào đó của N.

1 HỆ TỌA ĐỘ THIẾT BỊ

Ví dụ: Độ phân giải của màn hình trong chế độ đồ họa là 640x480. Khi đó,

- x thuộc (0,640)
- y thuộc (0,480)



Hình 2.2. Hệ toạ độ thiết bị

1 HỆ TỌA ĐỘ THIẾT BỊ CHUẨN

- Đại diện chung cho tất cả các thiết bị để có thể mô tả các hình ảnh mà không phụ thuộc vào bất kỳ thiết bị nào.
- Trong hệ tọa độ chuẩn, các tọa độ **x, y** sẽ được gán các giá trị trong đoạn từ **[0,1]**.

→ Vùng không gian của hệ tọa độ chuẩn chính là hình vuông đơn vị có góc trái dưới (0, 0) và góc phải trên là (1, 1).

1 HỆ TỌA ĐỘ THIẾT BỊ CHUẨN

Normalized device coordinates

Ảnh thực

Tọa độ chuẩn hóa

Tọa độ thiết bị

Thiết bị xuất

1

Vẽ điểm

1 ĐIỂM

- Xác định bằng tọa độ
- Hàm vẽ điểm:

GL_POINTS

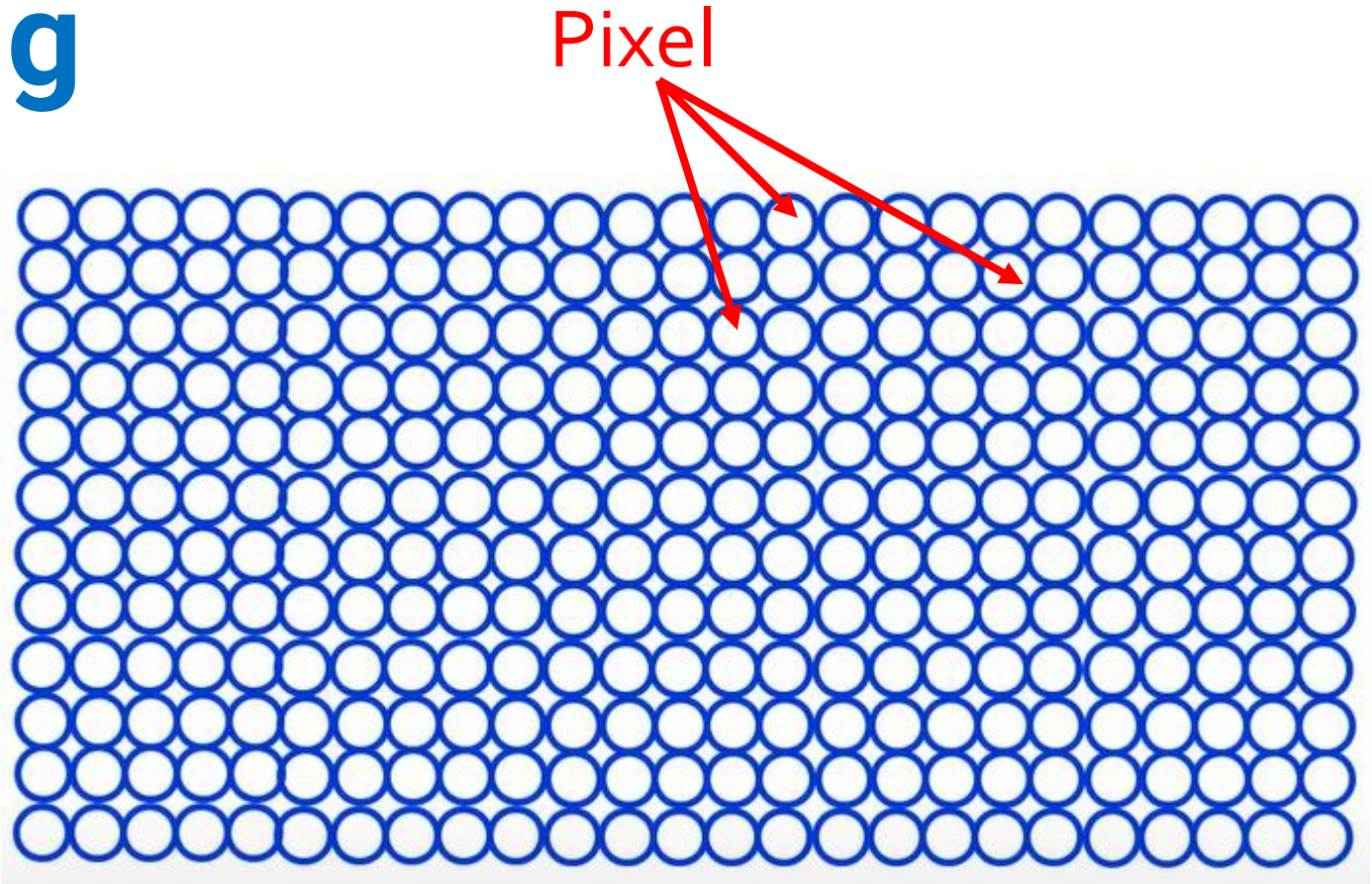
```
void display() {

    glColor3f(1.0,1.0,0.0);

    glPointSize(10);
    glBegin(GL_POINTS);
        glVertex2f(50,100);
        glVertex2f(200,200);
    glEnd();
    glFlush();
}
```

2

Vẽ đoạn thẳng

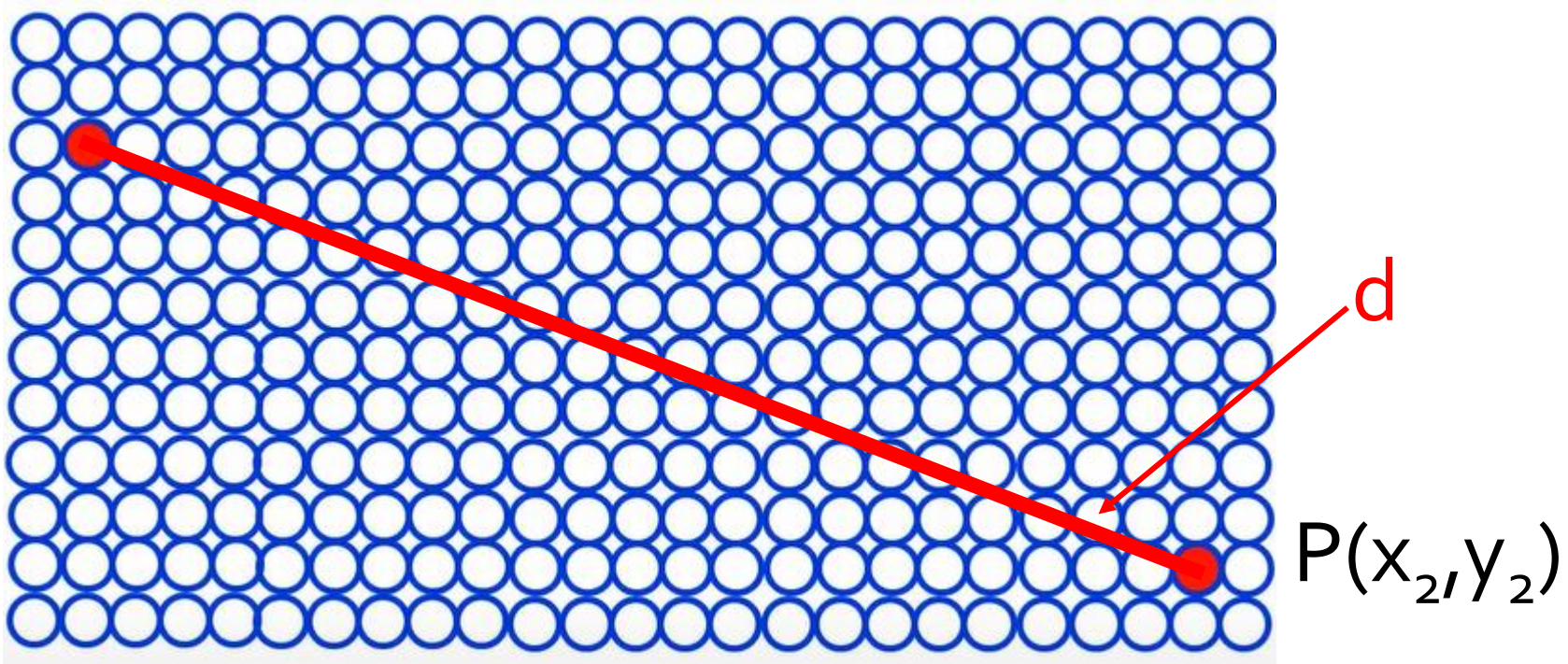
2 Vẽ đoạn thẳng

- Cấu trúc màn hình hiển thị là ma trận các pixel rời rạc.
- Các pixel cách nhau 1 đơn vị (quy ước).



Hình 2.3. Cấu trúc ma trận pixel của màn hình hiển thị

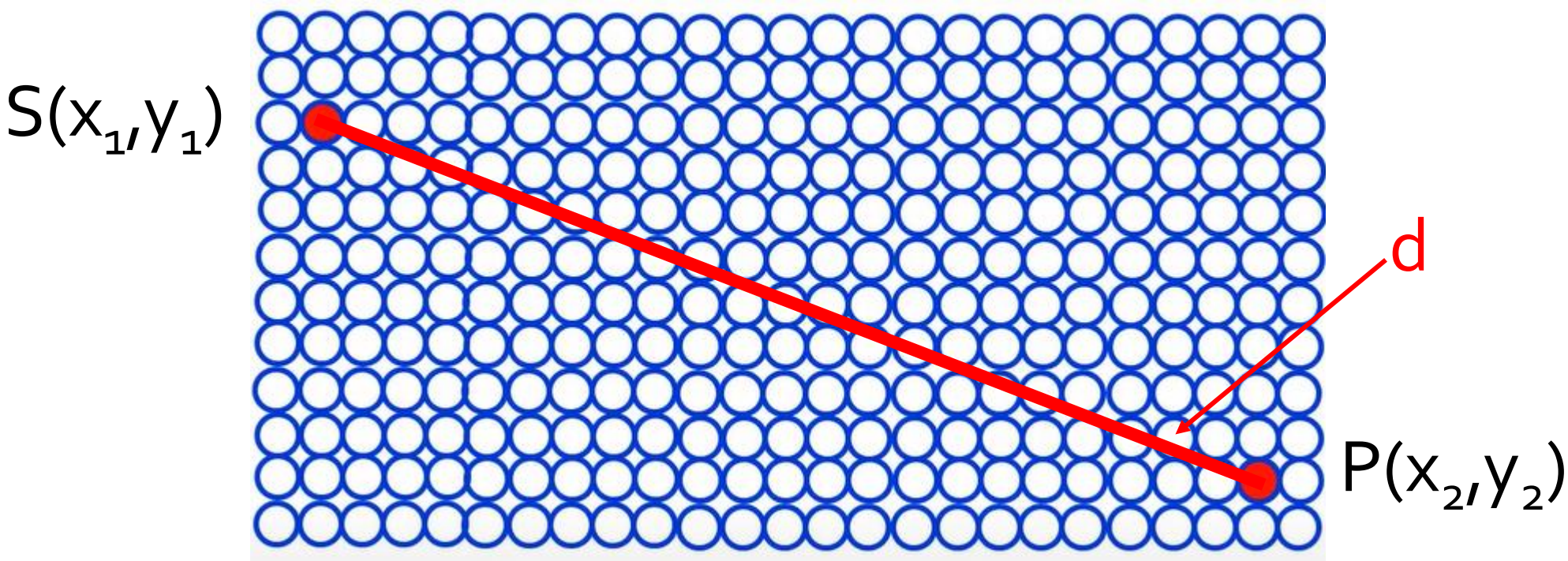
2 Vẽ đoạn thẳng



Hình 2.4. Vẽ đoạn thẳng d từ điểm S đến điểm P

- Vẽ một đoạn thẳng từ S đến P ta được đoạn thẳng liên tục d và gọi là đoạn thẳng "lý tưởng".

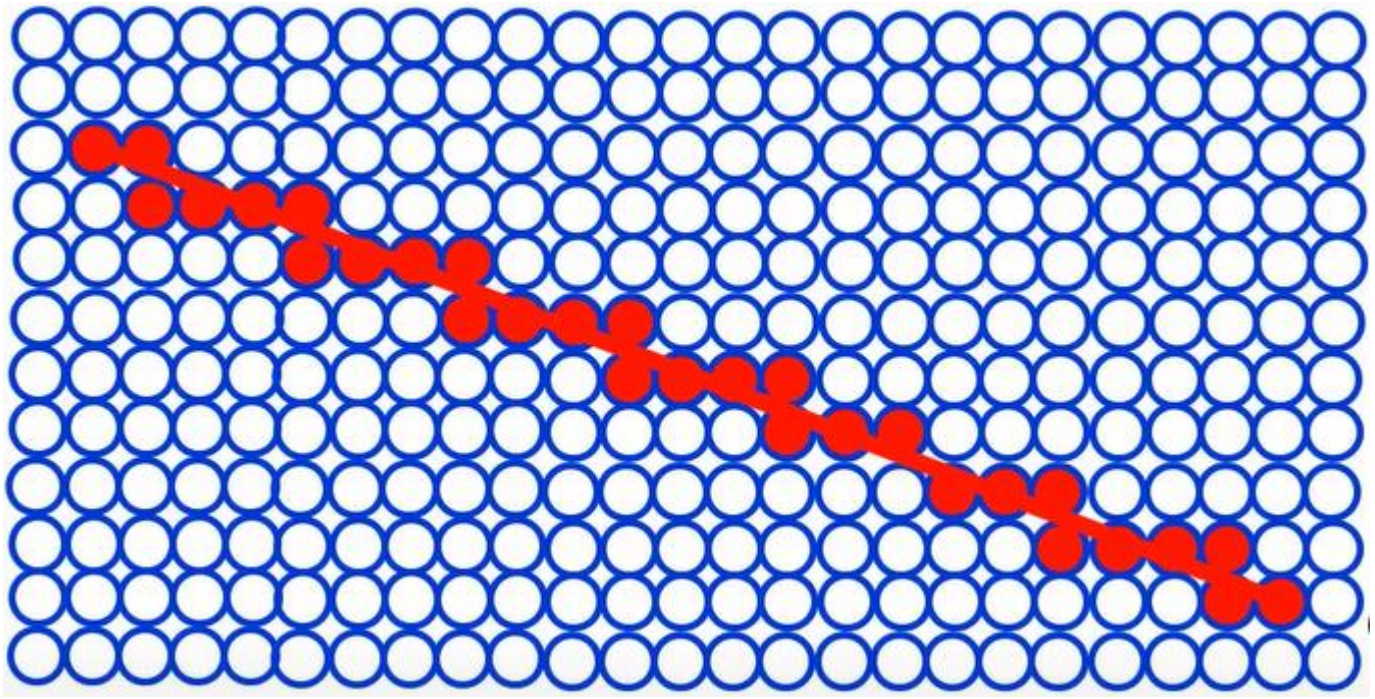
2 Vẽ đoạn thẳng



Hình 2.4. Vẽ đoạn thẳng d từ điểm S đến điểm P

- Nhưng hình ảnh hiển thị trên màn hình là tập hợp các pixel rời rạc nên ta phải chuyển đổi ảnh liên tục thành rời rạc.

2 Vẽ đoạn thẳng



Hình 2.5. Biến đổi đoạn thẳng liên tục thành rời rạc

2 Vẽ đoạn thẳng

❑Nguyên tắc vẽ đoạn thẳng

- ○Bài toán: Vẽ đoạn thẳng qua 2 điểm $S(x_1,y_1)$ và $P(x_n,y_n)$
- ○Giải pháp:
  - ○Đoạn thẳng là tập hợp các điểm có toạ độ nguyên, xấp xỉ toạ độ thực.

$$(x_i, y_i) \text{ với } i = 1 \dots n,\ x_i, y_i \in N$$

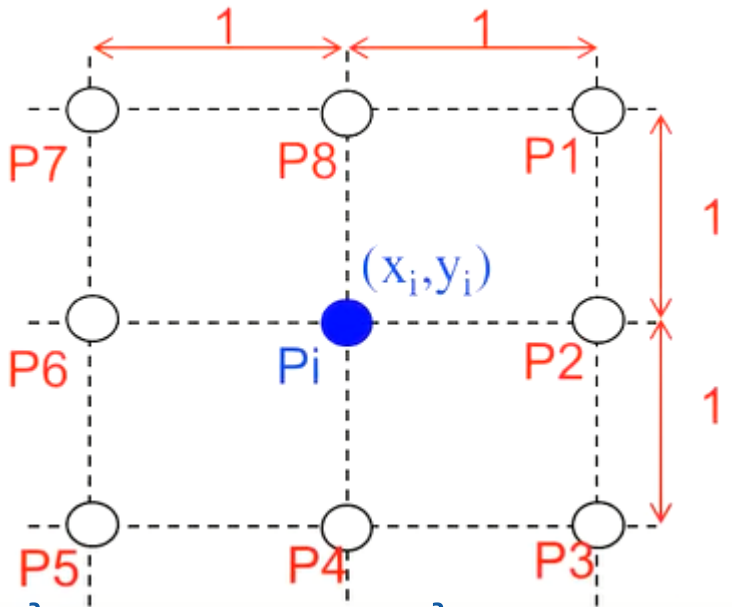
  - ○Tại bước i, xác định điểm (x_i, y_i) ➔ vẽ điểm đó
  - ○Tại bước i+1, xác định điểm (x_{i+1}, y_{i+1}) ➔ vẽ điểm kế tiếp
  - ○ Các điểm này phải gần các điểm lý tưởng

2 Vẽ đoạn thẳng

- Nguyên tắc xác định điểm lân cận
  - Tại bước i ta xác định được toạ độ nguyên (x_1,y_1)
  - Tại bước i+1, xác định điểm (x_{i+1}, y_{i+1}) là 1 trong 8 điểm lân cận.

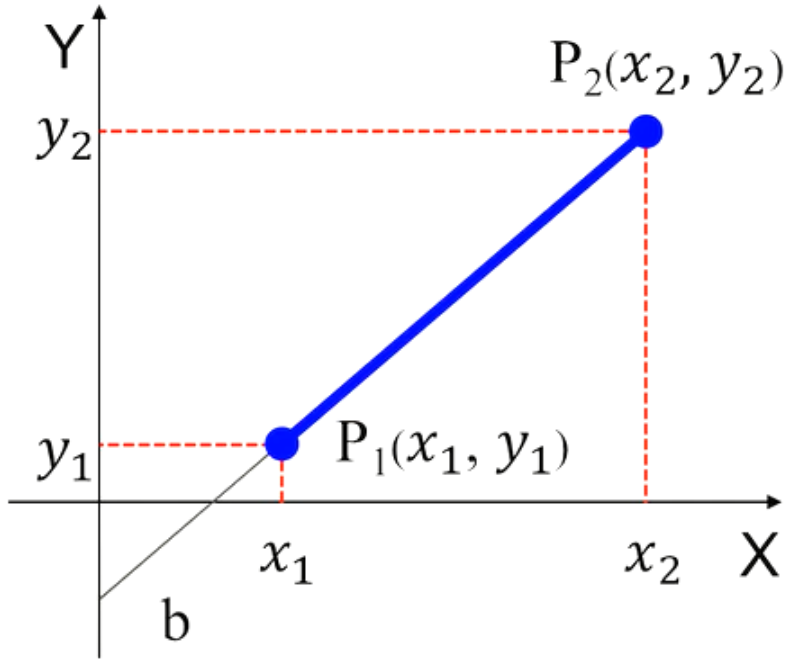
Toạ độ 8 điểm lân cận của điểm $P(x_i,y_i)$

$P1(x_i+1,y_i+1)$	$P5(x_i-1,y_i-1)$
$P2(x_i+1,y_i)$	$P6(x_i-1,y_i)$
$P3(x_i+1,y_i-1)$	$P7(x_i-1,y_i+1)$
$P4(x_i,y_i-1)$	$P8(x_i,y_i+1)$



Hình 2.5. Điểm lân cận điểm P(xi,yi)

2 Vẽ đoạn thẳng



- Để vẽ được đoạn thẳng ta sử dụng phương trình đoạn thẳng đi qua 2 điểm như sau:

$$y = mx + b$$

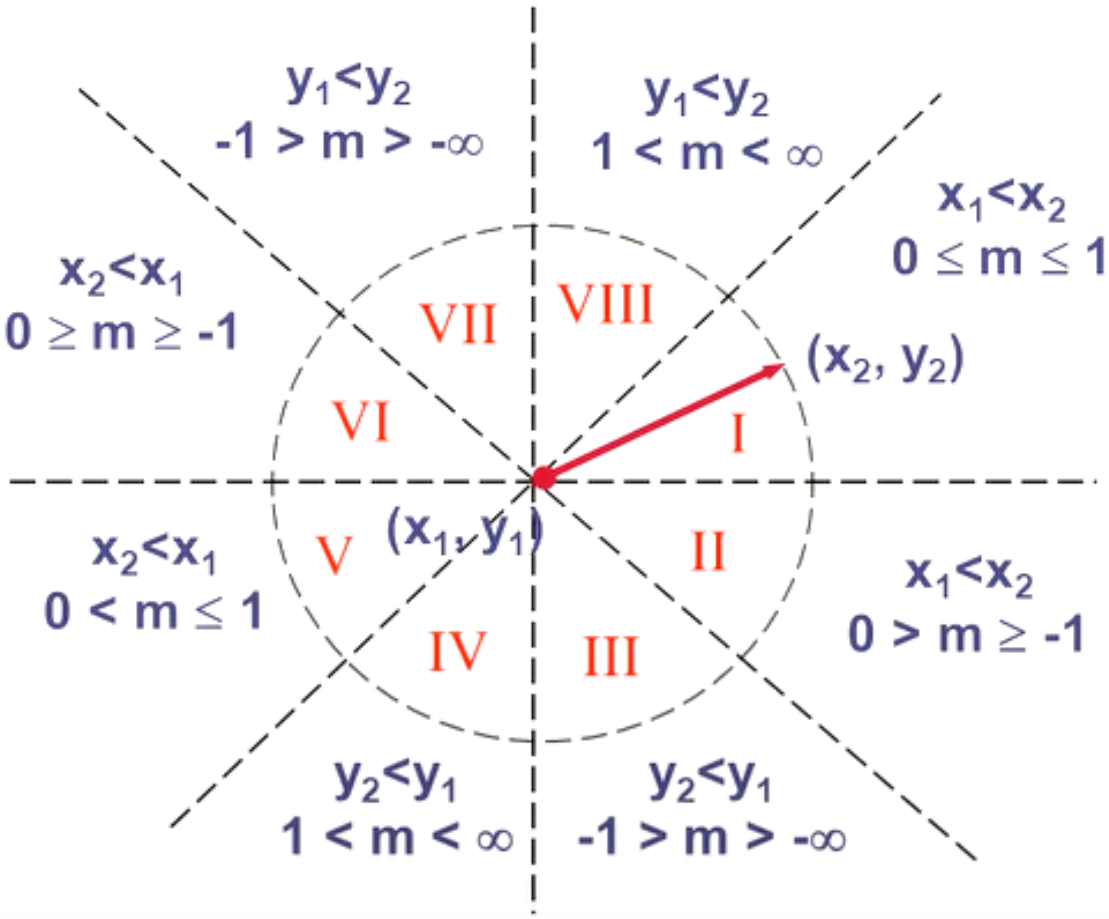
- $m = \frac{(y_2 - y_1)}{(x_2 - x_1)} = \frac{\Delta_y}{\Delta_x}$ gọi là hệ số góc (độ dốc) của đoạn thẳng và $\Delta_y = y_2 - y_1, \Delta_x = x_2 - x_1$
- **b:** gọi là đoạn chắn trên trục y

2 Vẽ đoạn thẳng

- Trong thực tế đoạn thẳng có nhiều dạng, phụ thuộc vào m.
- Ta chỉ xét đoạn thẳng nằm trong vùng I, có hệ số góc:

$0 \le m \le 1$ và

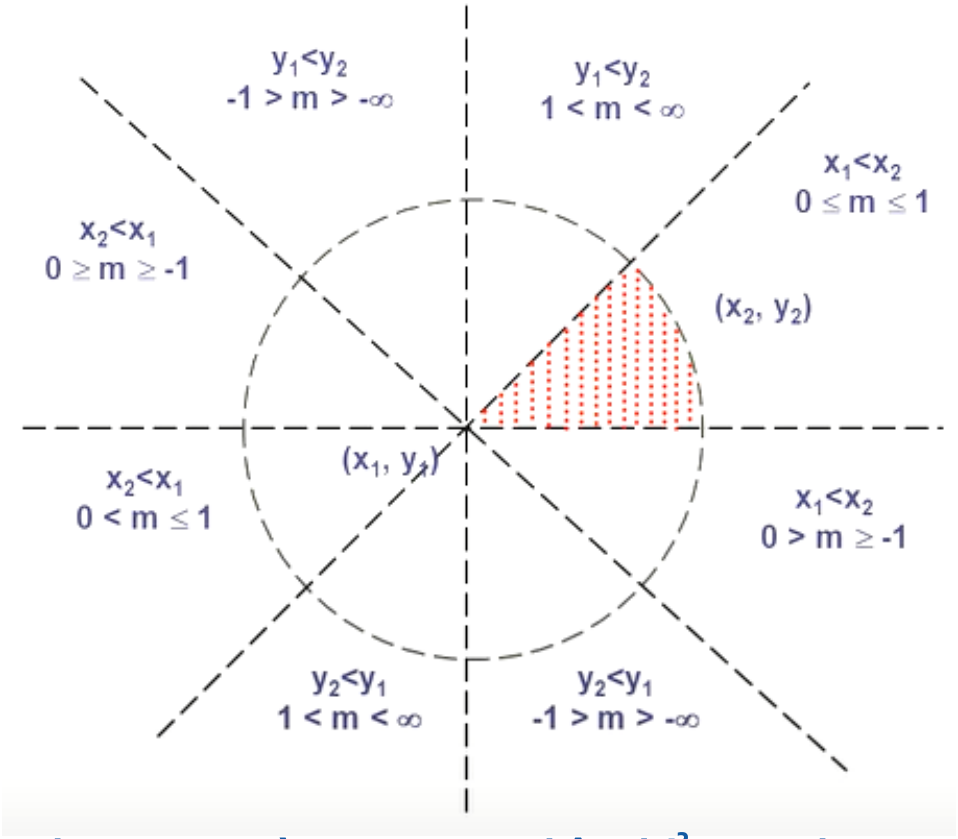
$x_1 < x_2 \ (\Delta_x > 0)$



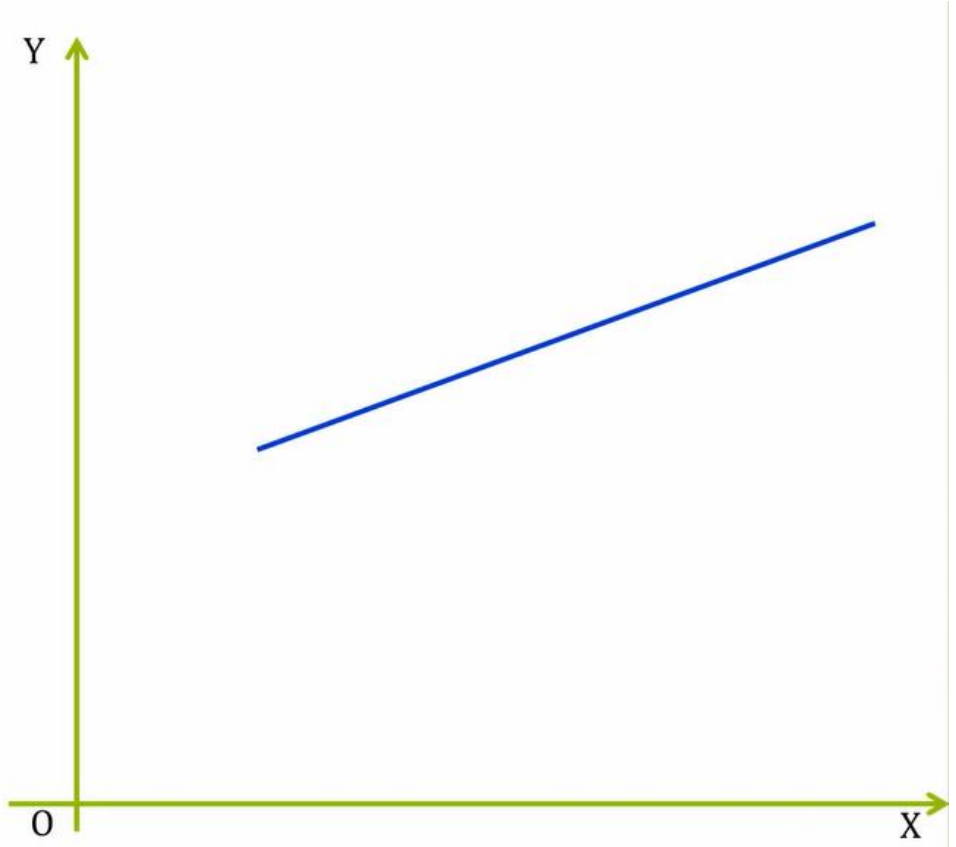
Hình 2.6. Các giá trị của m trong thực tế

2 Vẽ đoạn thẳng

- Xét đoạn thẳng $0 \leq m \leq 1$ và $\Delta_x > 0$



Hình 2.7. Vùng toạ độ để vẽ đoạn thẳng có $0 \leq m \leq 1$



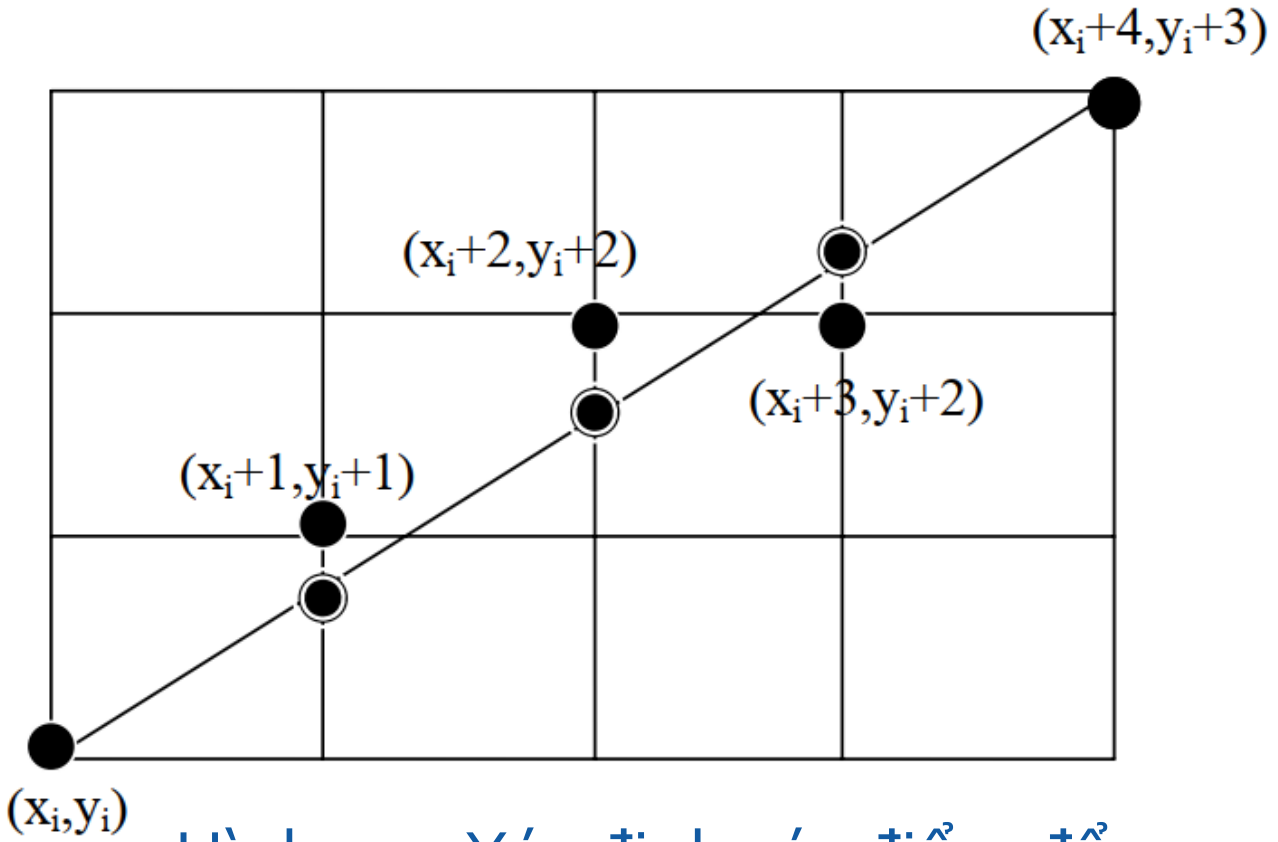
Hình 2.8. Đoạn thẳng có $0 \leq m \leq 1$

2 Thuật toán vẽ đoạn thẳng

Xét đoạn thẳng có hệ số góc 0<m<=1 và Δx>0.

Với các đoạn thẳng dạng này, nếu (x_i,y_i) là điểm đã được xác định ở bước thứ **i** thì điểm kế tiếp (x_{i+1},y_{i+1}) ở bước thứ **i+1** sẽ là một trong hai điểm sau:

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



Hình 2.9. Xác định các điểm để vẽ đoạn thẳng trên màn hình

2 Thuật toán vẽ đoạn thẳng

- Xác định 2 điểm
- Hàm vẽ đoạn thẳng:

GL_LINES

```
void display() {

    glColor3f(1.0,1.0,0.0);

   // glPointSize(100);
    glBegin(GL_LINES);
        glVertex2f(100,100);
        glVertex2f(400,400);
    glEnd();
    glFlush();
}
```

2 Thuật toán vẽ đoạn thẳng

- **Thuật toán DDA - Digital Differential Analyzer**

Là thuật toán tính toán các điểm vẽ dọc theo đường thẳng dựa vào **hệ số góc** của phương trình đường thẳng: y=mx+b.

Trong đó: $m = \frac{\Delta y}{\Delta x}, \Delta y = y_{i+1} - y_i, \Delta x = x_{i+1} - x_i$

Từ hình 2.9 nhận thấy rằng:

2 Thuật toán vẽ đoạn thẳng

- Toạ độ x tại bước thứ x_{i+1} tăng lên 1 đơn vị: $x_i + 1$
- Toạ độ y tại bước thứ y_{i+1} là y_i hay $y_i + 1$ phụ thuộc vào giá trị y sau khi làm tròn.
- Tính giá trị y ở bước thứ y_{i+1}

Từ phương trình đường thẳng: y=mx+b

Ta có:

$y_{i+1} = mx_{i+1} + b = m(x_i + 1) + b = mx_i + b + m$

2 Thuật toán vẽ đoạn thẳng

- Để tính phương trình này thì tại mỗi bước ta phải thực hiện phép toán nhân và cộng số thực ➔ Thực toán thực thi chậm.
- Để cải thiện tốc độ, ta phải khử phép nhân trên số thực như sau:
- Ta có:

$$y_i = mx_i + b$$

$$\Rightarrow \quad y_{i+1} = y_i + m \rightarrow Round(y_{i+1})$$

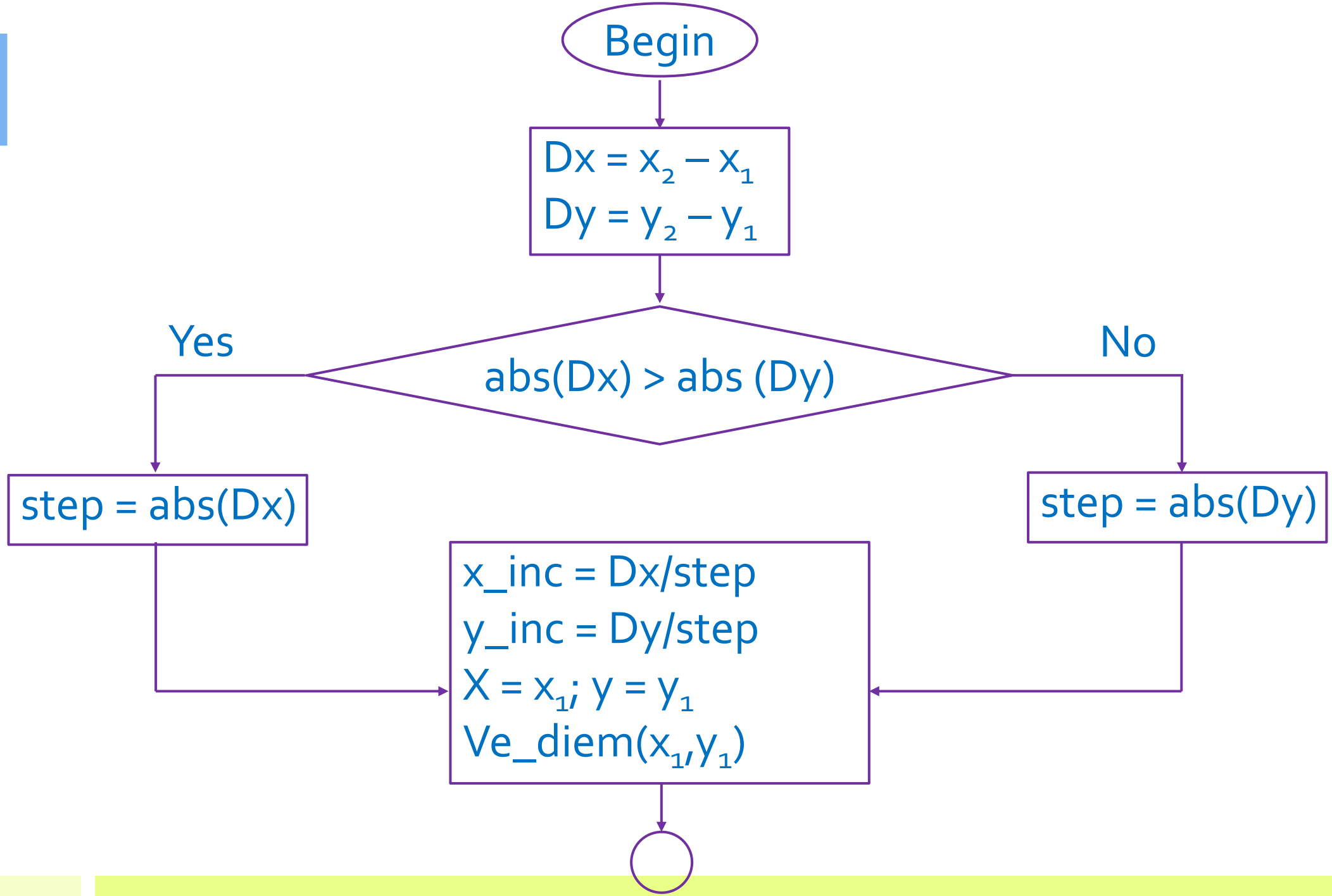
2 Thuật toán vẽ đoạn thẳng

- Như vậy, khi 0 < m <= 1 ta có:

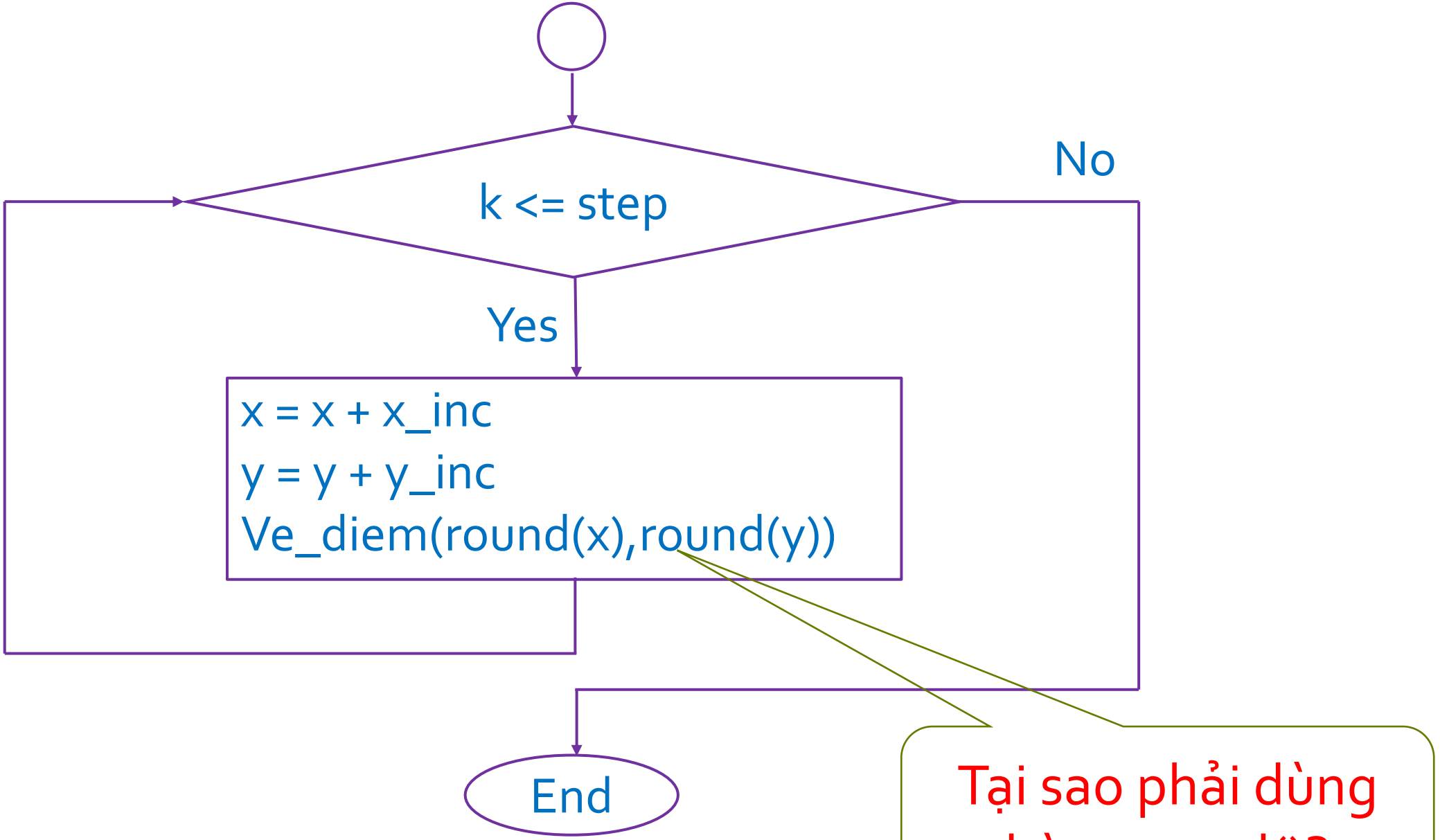
$$x_i = x_i + 1$$
$$y_{i+1} = y_i + m \rightarrow Round(y_{i+1})$$

LƯU ĐỒ THUẬT TOÁN DDA

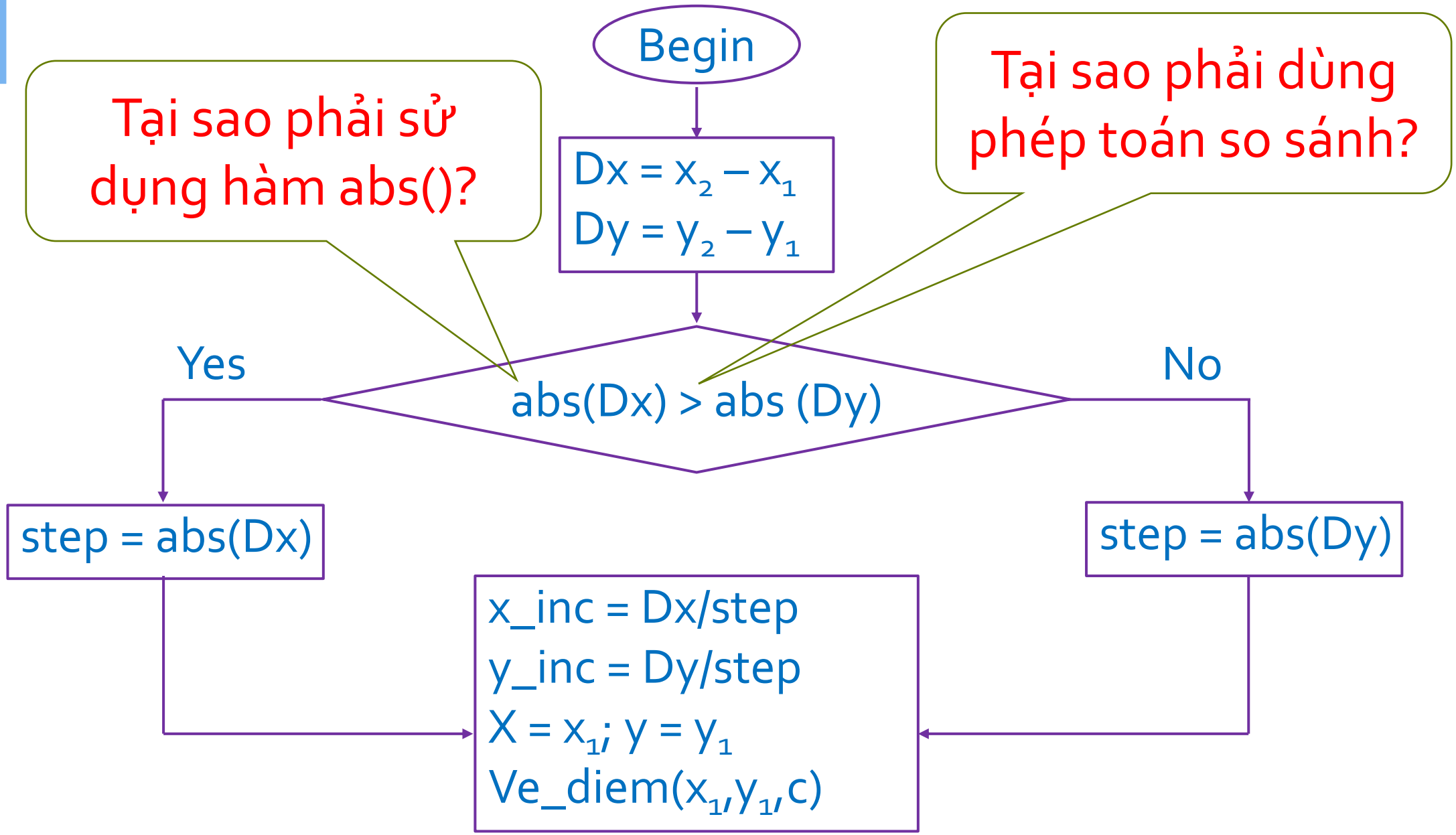
2


Begin
Dx = x2 – x1
Dy = y2 – y1
abs(Dx) > abs (Dy)
Yes
No
step = abs(Dx)
step = abs(Dy)
x_inc = Dx/step
y_inc = Dy/step
X = x1; y = y1
Ve_diem(x1,y1)

2


k <= step
No
Yes
x = x + x_inc
y = y + y_inc
Ve_diem(round(x),round(y))
End
Tại sao phải dùng hàm round()?

2


Begin
Dx = x2 – x1
Dy = y2 – y1
Tại sao phải sử dụng hàm abs()?
Tại sao phải dùng phép toán so sánh?
abs(Dx) > abs (Dy)
Yes
No
step = abs(Dx)
step = abs(Dy)
x_inc = Dx/step
y_inc = Dy/step
X = x1; y = y1
Ve_diem(x1,y1,c)

2 Thuật toán vẽ đoạn thẳng

Cài đặt thuật toán DDA

```
18  void lineDDA(int x1, int y1, int x2, int y2){
19      int Dx = x2 - x1;
20      int Dy = y2 - y1;
21      int steps;
22
23      if (abs(Dx) > abs(Dy))
24          steps = Dx;
25      else
26          steps = Dy;
27
28      float x_inc = static_cast<float>(Dx)/steps;
29      float y_inc = static_cast<float>(Dy)/steps;
```

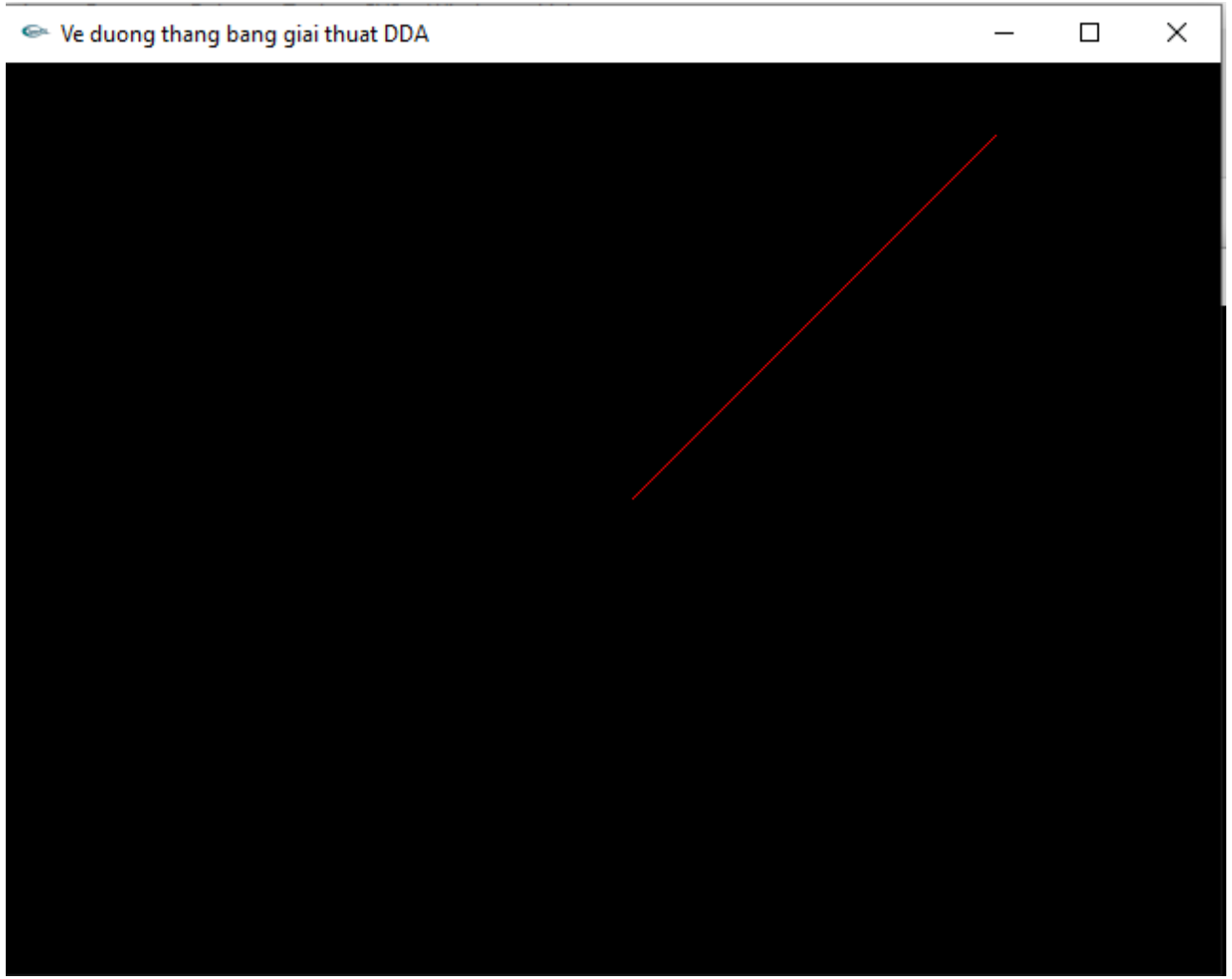
2 Thuật toán vẽ đoạn thẳng

Cài đặt thuật toán DDA

```
31      glBegin(GL_POINTS);
32      glVertex2f(x1,y1);
33
34      float x = x1;
35      float y = y1;
36      int k = 0;
37      while(k <= steps){
38          x = x + x_inc;
39          y = y + y_inc;
40          k++;
41          glVertex2f(round(x),round(y));
42      }
43      glEnd();
44  }
```

2 Thuật toán vẽ đoạn thẳng

Kết quả



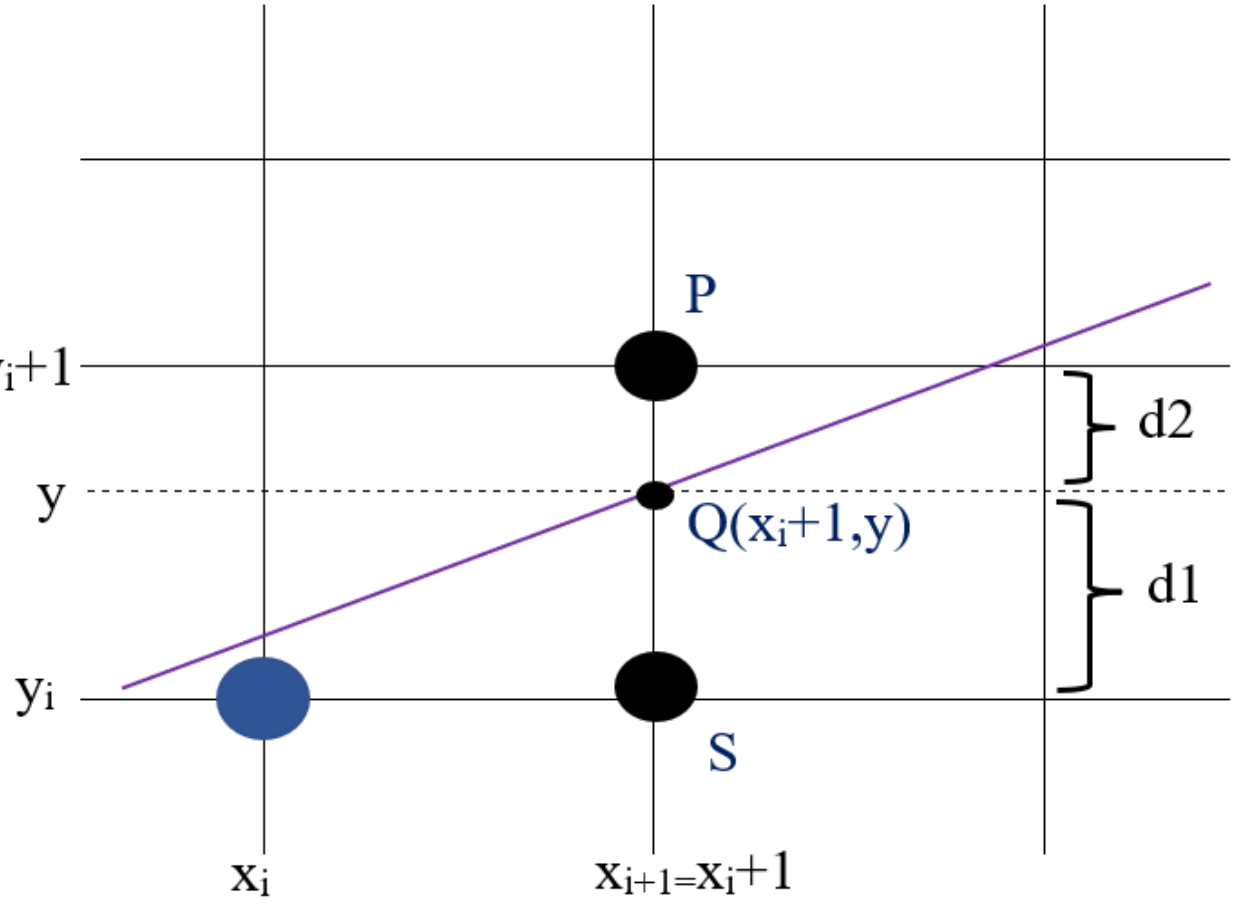
Hình 2.10. Đoạn thẳng được vẽ bằng thuật toán DDA

2 Thuật toán vẽ đoạn thẳng

- **Thuật toán Bresenham**

Xét đoạn thẳng $0 < m \leq 1$

- **Giả sử ở bước thứ i:** xác định được điểm (x_i, y_i).
- **Ở bước thứ i+1:** điểm cần chọn (x_{i+1}, y_{i+1}) sẽ là điểm $S(x_i+1, y_i)$ hoặc $P(x_i+1, y_i+1)$



Hình 2.11. Dạng đường thẳng 0<m<=1

2 Thuật toán vẽ đoạn thẳng

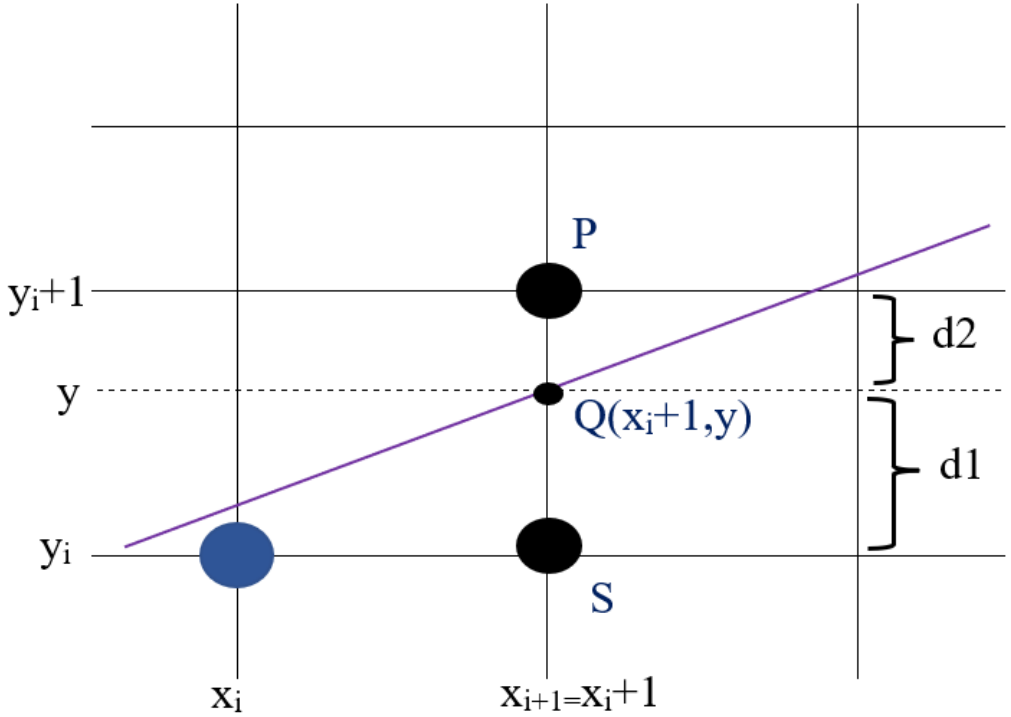
Xác định điểm S hay điểm P

- Giả sử ở bước i xác định được điểm (x_i,y_i)
- Xét điểm $Q(x_i+1,y)$
- Từ $y = mx + b \Rightarrow y = mx_i + b$

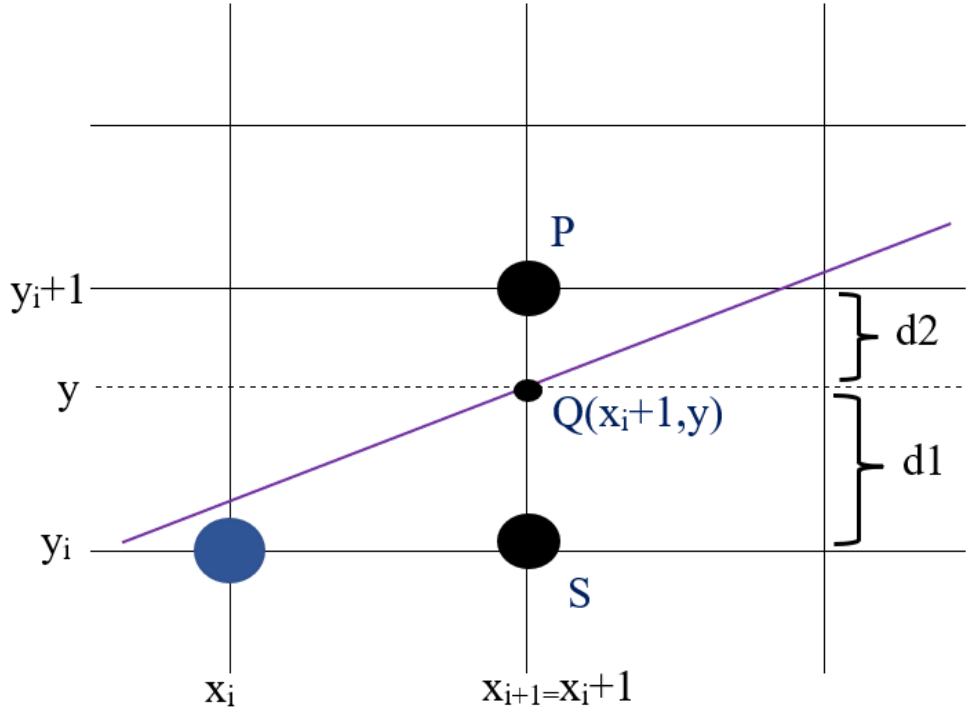
Q có toạ độ $Q(x_i+1, m(x_i+1)+b)$

- Bước i+1, xác định điểm (x_{i+1},y_{i+1})

dựa vào d_1-d_2



2 Thuật toán vẽ đoạn thẳng



Xác định điểm S hay điểm P

- Nếu d_1-d_2<0 ⇔d_1<d_2 ➔chọn điểm $S(x_i + 1, y_i)$
- Nếu d_1-d_2>=0 ⇔d_1>=d_2➔chọn điểm $P(x_i + 1, y_i + 1)$

- Tính: $d_1 = y - y_i = m(x_i + 1) + b - y_i$ (1)

$$d_2 = (y_i + 1) - y_i = y_i + 1 - m(x_i + 1) - b \quad (2)$$

2 Thuật toán vẽ đoạn thẳng

Xác định điểm S hay điểm P

- Nên: $d_1 - d_2 = 2mx_i - 2y_i + 2m + 2b - 1$ (3)

Với: $m = \frac{(y_2 - y_1)}{(x_2 - x_1)} = \frac{D_y}{D_x}$

- Từ (3) ta thấy m là số thực nên việc tính toán rất chậm, do đó thay vì xét dấu $d_1 - d_2$ thì ta xét dấu một đại lượng khác có liên quan đến $d_1 - d_2$ và khử D_x như sau:

2 Thuật toán vẽ đoạn thẳng

Xác định điểm S hay điểm P

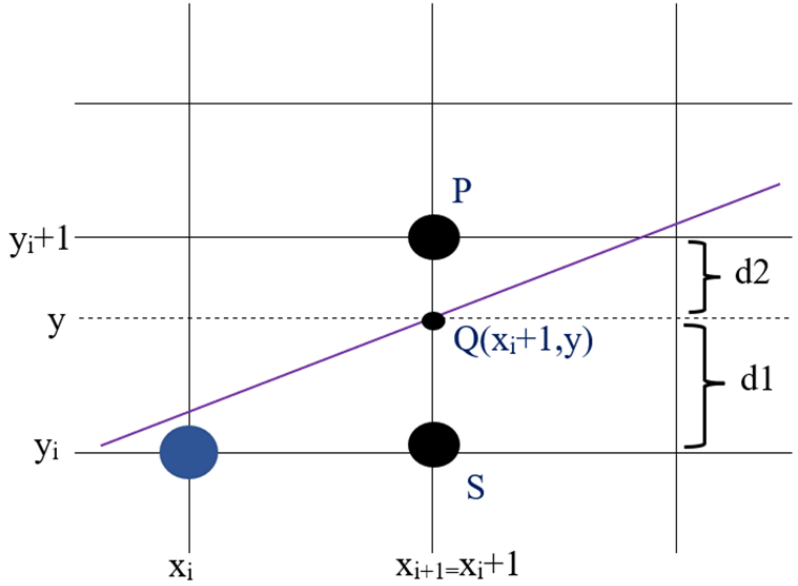
- Thay $m = \frac{D_y}{D_x}$ vào (3) và khử D_x ta có:

$$D_x(d_1 - d_2) = 2D_y x_i - 2D_x y_i + 2D_y + (2b - 1)D_x$$

- Vì $D_x > 0 \Rightarrow$ dấu của $D_x(d_1\text{-}d_2)$ phụ thuộc vào $(d_1\text{-}d_2)$
- Đặt $p_i = D_x(d_1 - d_2) = 2D_y x_i - 2D_x y_i + c$

với $c = 2D_y + (2b - 1)D_x$

2 Thuật toán vẽ đoạn thẳng



Xác định điểm S hay điểm P

- Vậy:

nếu $p_i < 0 \leftrightarrow d_1 < d_2 \Rightarrow chọn\ S(x_i + 1, y_i)$

nếu $p_i \geq 0 \leftrightarrow d_1 \geq d_2 \Rightarrow chọn\ P(x_i + 1, y_i + 1)$

- Như vậy để chọn điểm S hay P tại bước i ta dựa vào p_i
- Tại bước thứ i+1 ta xác định điểm tiếp theo như sau:

2 Thuật toán vẽ đoạn thẳng

Xác định điểm S hay điểm P

- Ta có: $p_{i+1} - p_i = \left(2D_y x_{i+1} - 2D_x y_{i+1} + c\right) - (2D_y x_i - 2D_x y_i + c)$ (4)
- Do $x_{i+1} = x_i + 1$ nên thay vào (4) ta được:
- $p_{i+1} - p_i = 2D_y - 2D_x(y_{i+1} - y_i)$

$$\Rightarrow p_{i+1} = p_i + 2D_y - 2D_x(y_{i+1} - y_i)$$

2 Thuật toán vẽ đoạn thẳng

Xác định điểm S hay điểm P

- Vậy:

○Nếu $p_i < 0 \leftrightarrow d_1 - d_2 < 0, chọn\ S(x_i + 1, y_i)$

Tức là $y_{i+1} = y_i$ nên $p_{i+1} = p_i + 2D_y$

○Nếu $p_i \geq 0 \leftrightarrow d_1 - d_2 \geq 0, chọn\ P(x_i + 1, y_i + 1)$

Tức là $y_{i+1} = y_i + 1$ nên $p_{i+1=}p_i + 2(D_y - D_x)$

2 Thuật toán vẽ đoạn thẳng

Xác định điểm S hay điểm P

- **Như vậy:**

○Tại bước i ta có $p_i = 2D_i x_i - 2D_x y_i + c$ (*)

Với $c = 2D_y + (2b - 1)D_x$ (**)

○Tại bước i+1 tính p_{i+1} dựa vào p_i

▪Nếu $p_i < 0 \Rightarrow p_{i+1} = p_i + 2D_y$

▪Nếu $p_i \geq 0 \Rightarrow p_{i+1} = p_i + 2(D_y - D_x)$

2 Thuật toán vẽ đoạn thẳng

Xác định điểm S hay điểm P

- **Từ đó ta tính điểm p_1 (điểm đầu tiên) có toạ độ (x_1,y_1) như sau:**

○Từ (*), (**) và điểm (x_1, y_1)

$$p_1 = 2D_y x_1 - 2D_x y_1 + 2D_y + (2b - 1)D_x \quad (***)$$

○Do (x_1,y_1) thuộc đường thẳng nên:

$y_1 = mx_1 + b = \frac{D_y}{D_x} x_1 + b$ thế vào (***)

2 Thuật toán vẽ đoạn thẳng

Xác định điểm S hay điểm P

$$\Rightarrow p_1 = 2D_y x_1 - 2D_x\left(\frac{D_y}{D_x}x_1 + b\right) + 2D_y + (2b-1)D_x$$

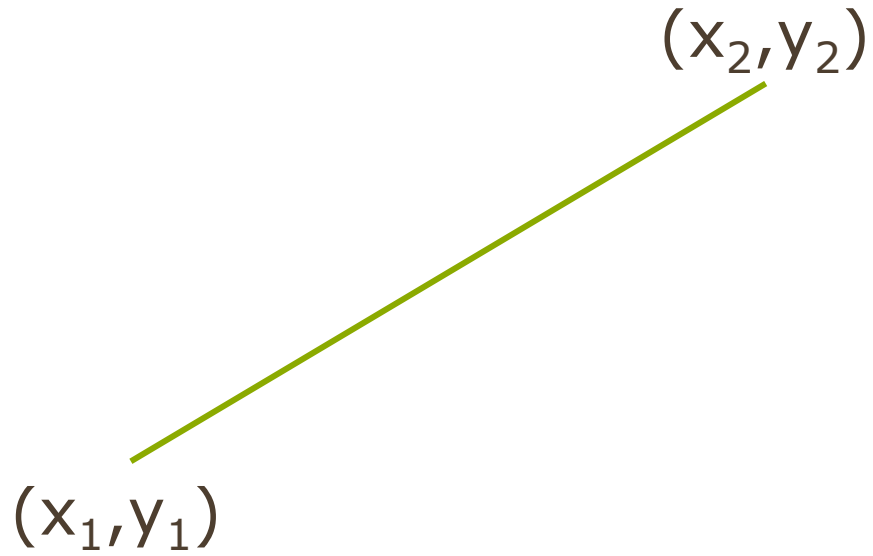
$$= 2D_y - D_x$$

Vậy: $p_1 = 2D_y - D_x$

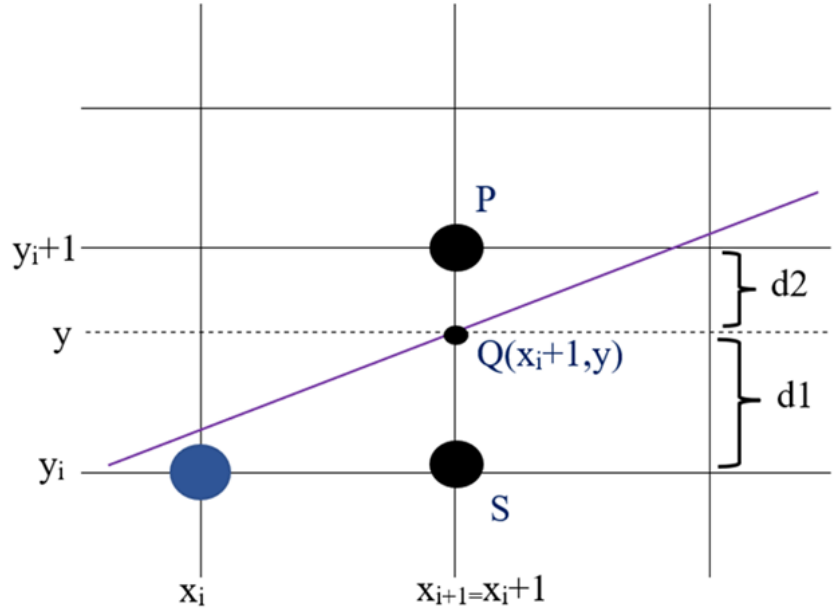
2 Thuật toán vẽ đoạn thẳng

Lưu đồ thuật toán Bresenham

- Điểm bắt đầu (x_1,y_1)
- Điểm kết thúc (x_2,y_2)



2



Begin

x = x1; y = y1;
Dx = x2-x1; Dy = $y_2 - y_1$;
p = 2Dy – Dx; Vẽ điểm (x,y);

$x \leq x2$

No

Yes

p<0

No (chọn P)

Yes (chọn S)

p = p + 2Dy;

p = p + 2(Dy – Dx);
y = y + 1;

x = x + 1; Vẽ điểm(x,y);

End

2

Cài đặt thuật toán Bresenham

```
7  void LineBres(int x1,int y1,int x2,int y2){
8      int Dx = x2-x1;
9      int Dy = y2-y1;
10     int p = 2*Dy-Dx;
11     int x = x1;
12     int y = y1;
13     glBegin(GL_POINTS);
14     glVertex2i(x,y);
15     while (x < x2)
16     {
17         if (p <0)
18             p+= 2*Dy;
19         else
20             {
21                 p+=2*(Dy-Dx);
22                 y++;
23             }
24         x++;
25         glVertex2i(x,y);
26     }
27     glEnd();
28 }
```

3 VẼ ĐƯỜNG TRÒN

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

Với tâm O(0,0) : $x^2 + y^2 = R^2$

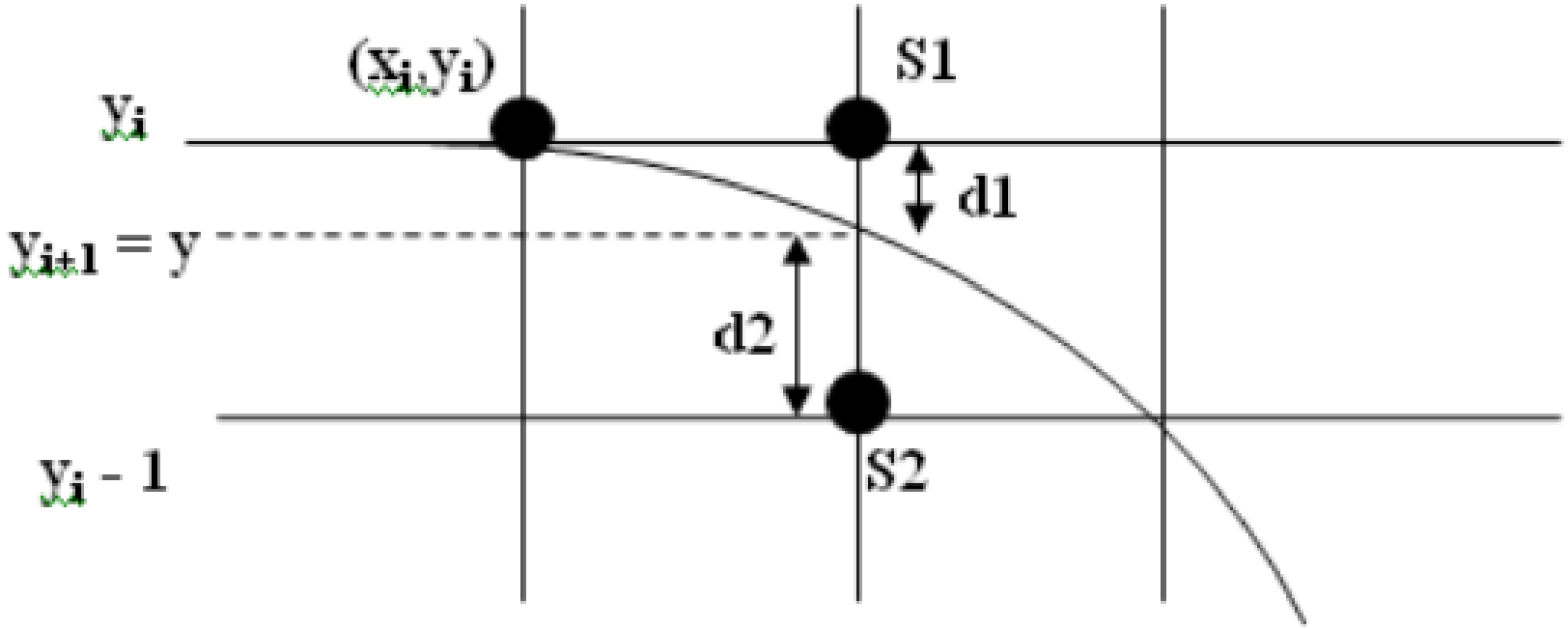
Với tâm C(x_c,y_c): $(x - x_c)^2 + (y - y_c)^2 = R^2$

Trong hệ tọa độ cực :

$$\begin{cases} x = x_c + R.\cos\theta \\ y = y_c + Y.\sin\theta \end{cases}$$

3 VẼ ĐƯỜNG TRÒN - BRESENHAM

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

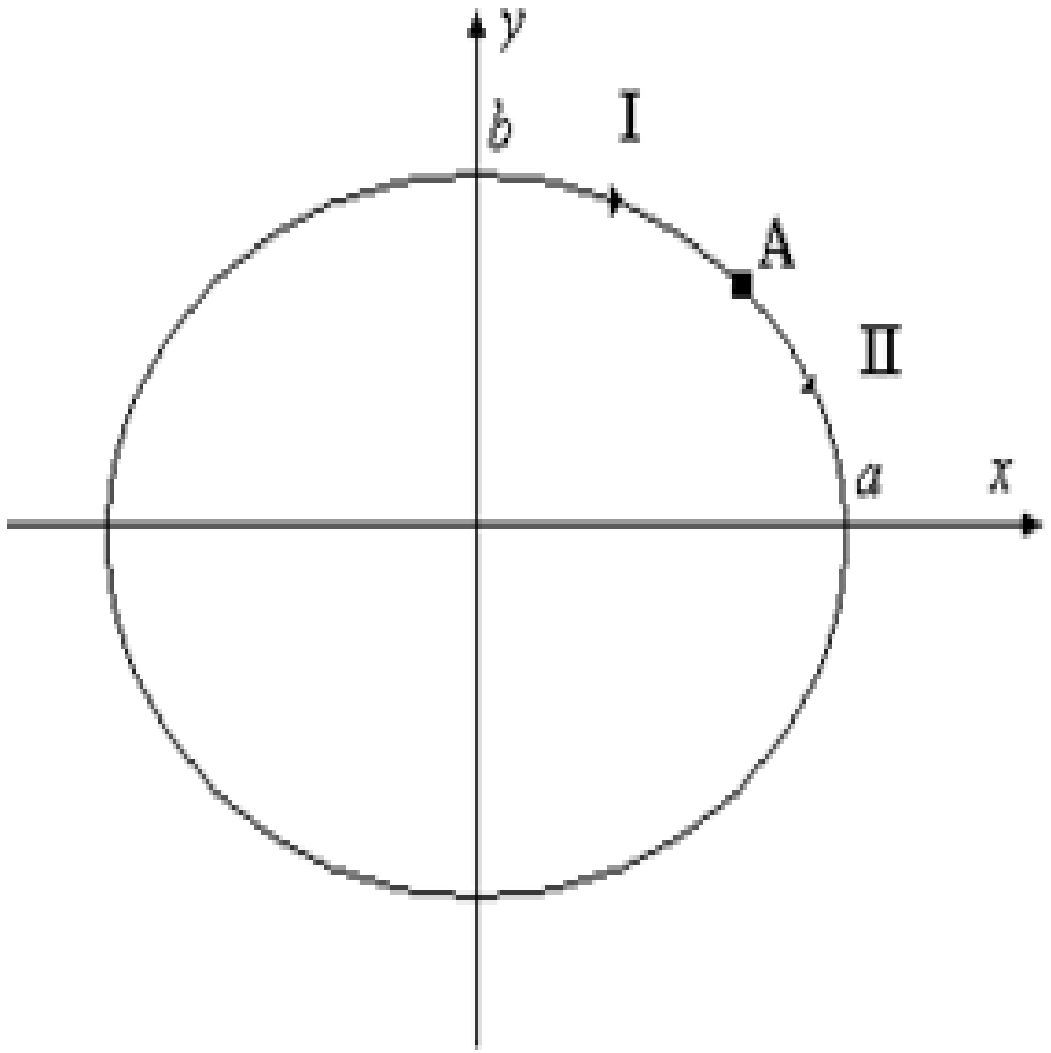


3 VẼ ĐƯỜNG TRÒN VỚI OPENGL

- Do tính chất đối xứng của đường tròn nên chỉ cần vẽ một cung, còn các cung còn lại ta lấy đối xứng từ cung I.
- Ở cung I , x tăng nhanh hơn y → dùng phương trình :

y = sqrt(r*r-x*x)

Khi lập trình cho x → tính y.



3 BÀI TẬP

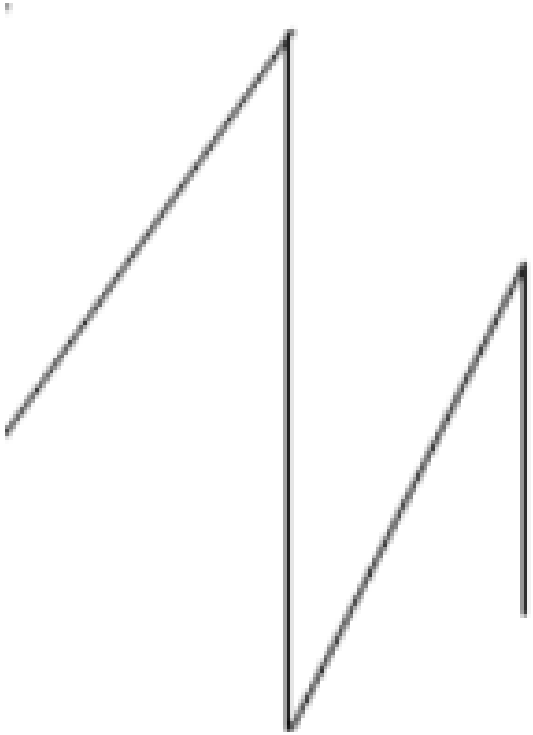
- Tìm hiểu phương pháp vẽ hình Ellipse và các đường Conic

4

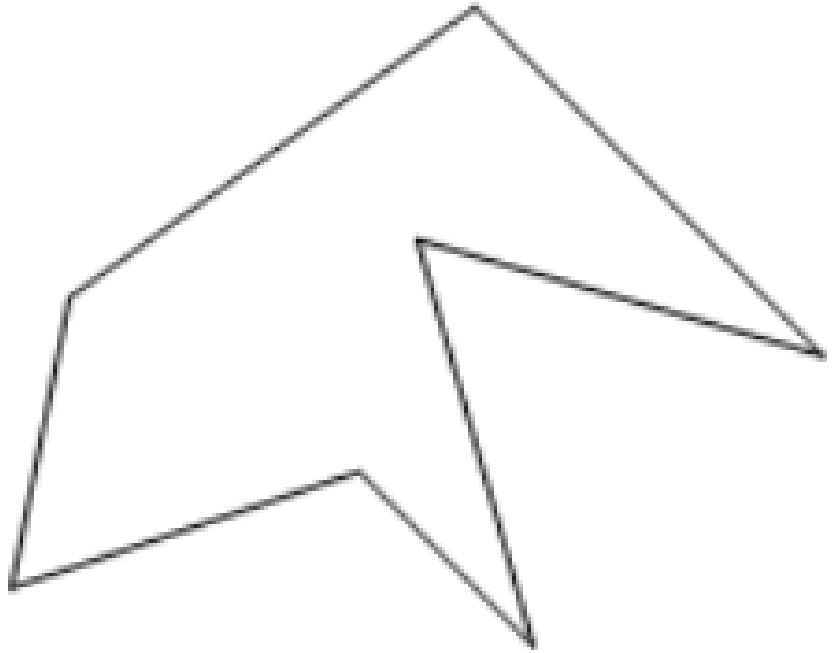
Vẽ đa giác POLYGON

4 Đa giác POLYGON

- Đa giác là một đường gấp khúc kín có đỉnh đầu và đỉnh cuối trùng nhau.



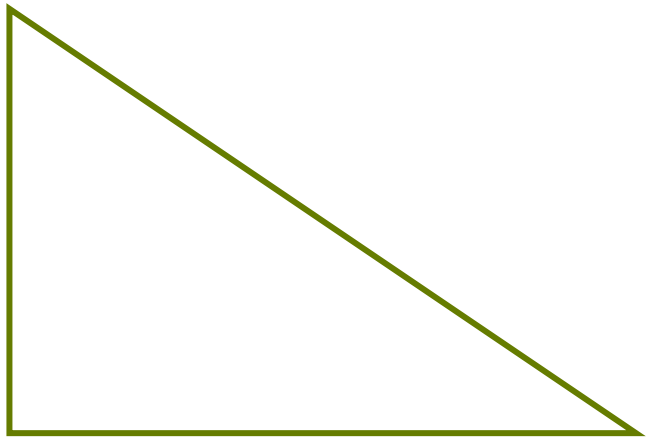
Đường gấp khúc hở

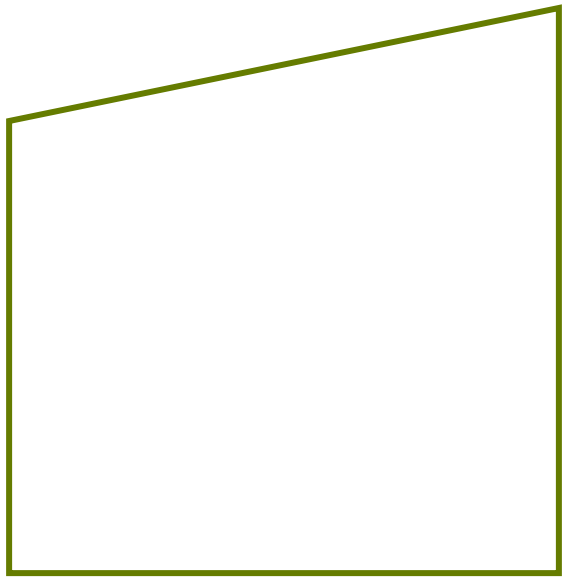


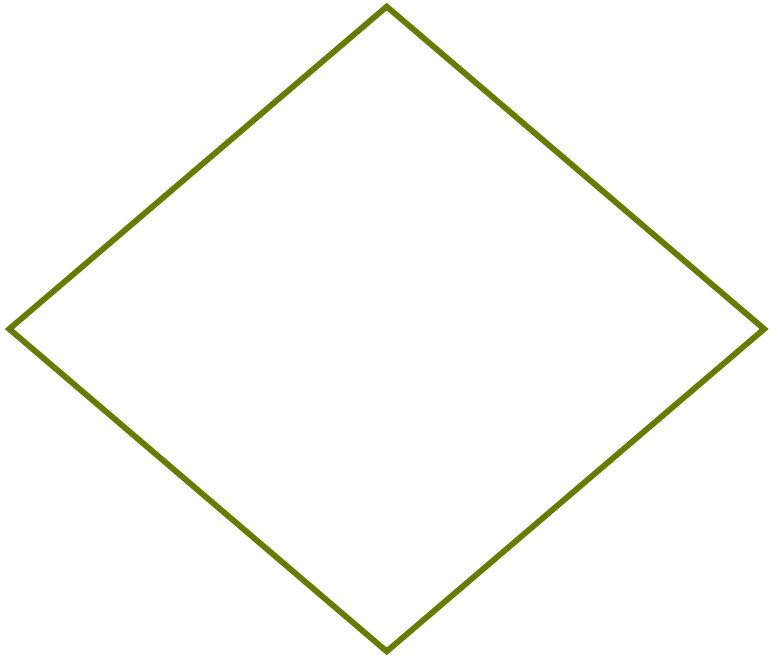
Đường gấp khúc kín

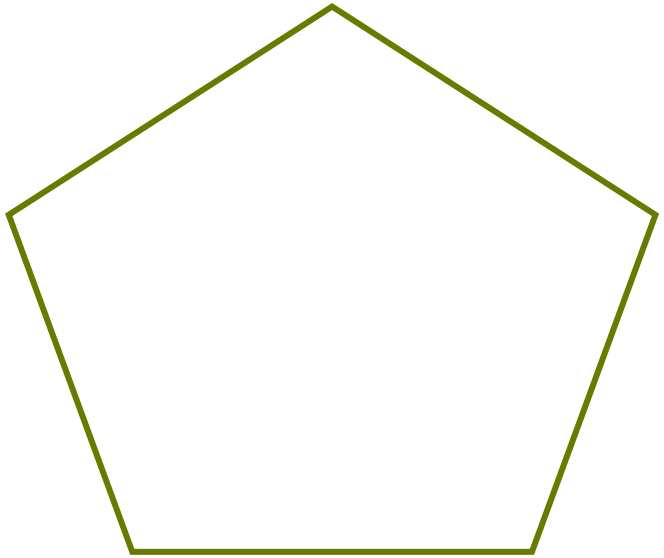
4 Đa giác POLYGON

- Vẽ đa giác với OpenGL
- Hàm: **GL_LINE_LOOP**
- Vị trí các đỉnh: **glVertex2f(x,y);**





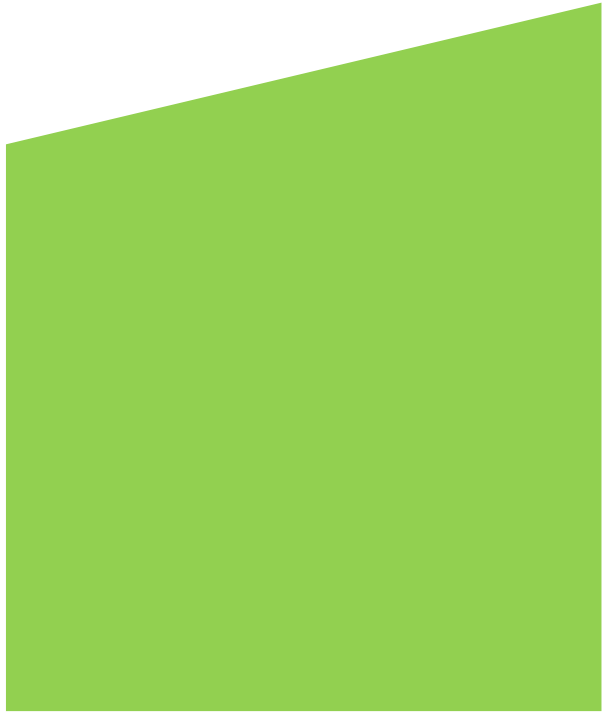




4 Tam giác

- Vẽ tam giác với OpenGL
- Hàm: **GL_TRIANGLES**
- Vị trí các đỉnh: **glVertex2f(x,y);**

4 Tứ giác



- Vẽ tứ giác với OpenGL
- Hàm: **GL_QUADS**
- Vị trí các đỉnh: **glVertex2f(x,y);**

6

CẤU TRÚC CHƯƠNG TRÌNH VỚI OPENGL

6 CẤU TRÚC CHƯƠNG TRÌNH VỚI OPENGL

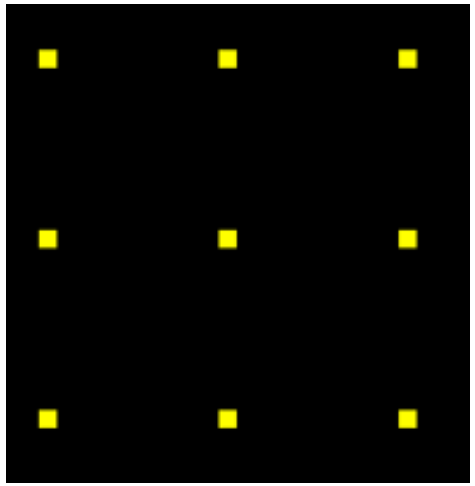
main.cpp

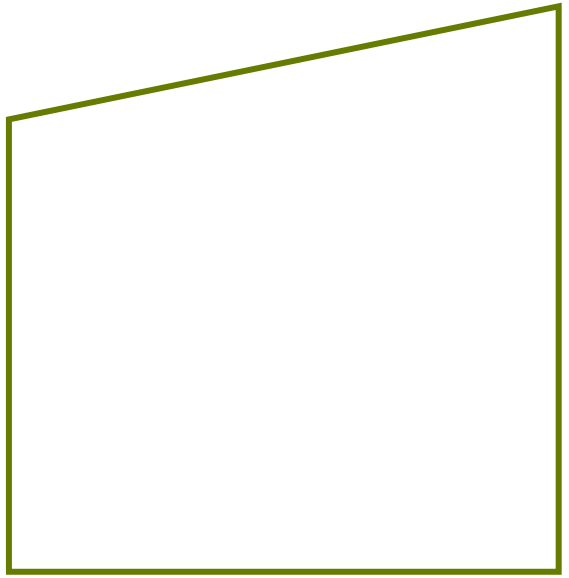
```
1   #include <iostream>
2   #include <gl\glut.h>
3   #include <gl\GL.h>
4   #include <gl\GLU.h>
5
6   void display() {
7
8       glColor3f(1.0,1.0,0.0);
9
10      glPointSize(10);
11      glBegin(GL_POINTS);
12          glVertex2f(50,100);
13          glVertex2f(200,200);
14      glEnd();
15      glFlush();
16  }
```

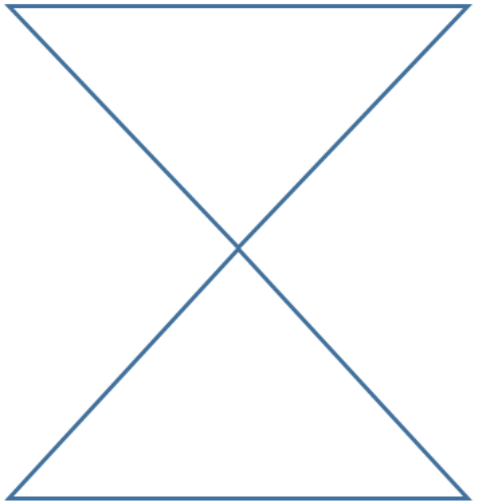
```
17  int main(int argc, char** argv) {
18      glutInit(&argc,argv);
19
20
21      glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
22      glutInitWindowPosition(0,0);
23      glutInitWindowSize(500,500);
24      glutCreateWindow("Ve");
25
26      glClearColor(0.0,0.0,0.0,0.0);
27      glClear(GL_COLOR_BUFFER_BIT);
28
29      glOrtho(0.0,500,0.0,500,-1.0,1.0);
30
31      glutDisplayFunc(display);
32      glutMainLoop();
33      return 0;
34  }
```

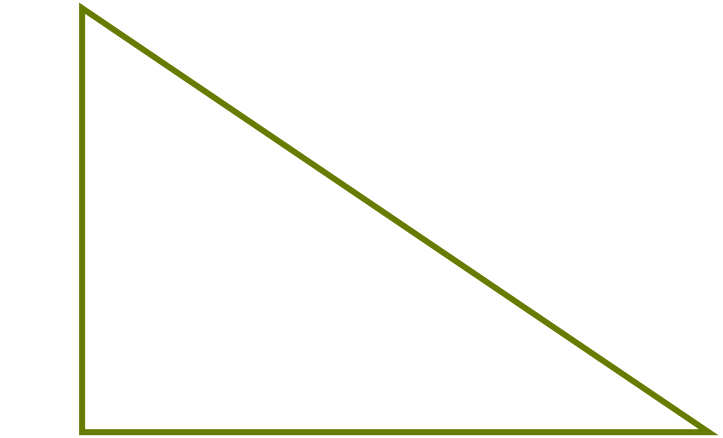
5 Sử dụng OpenGL vẽ các đối tượng hình học cơ bản

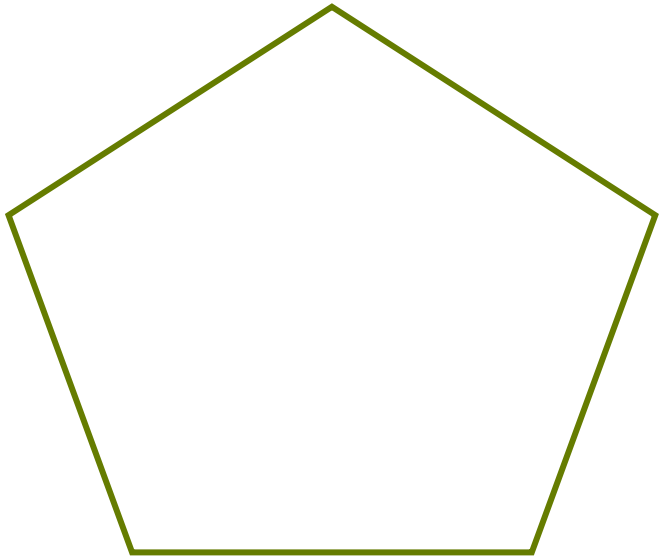
- Bài tập: Viết hàm display để vẽ đa giác sau:



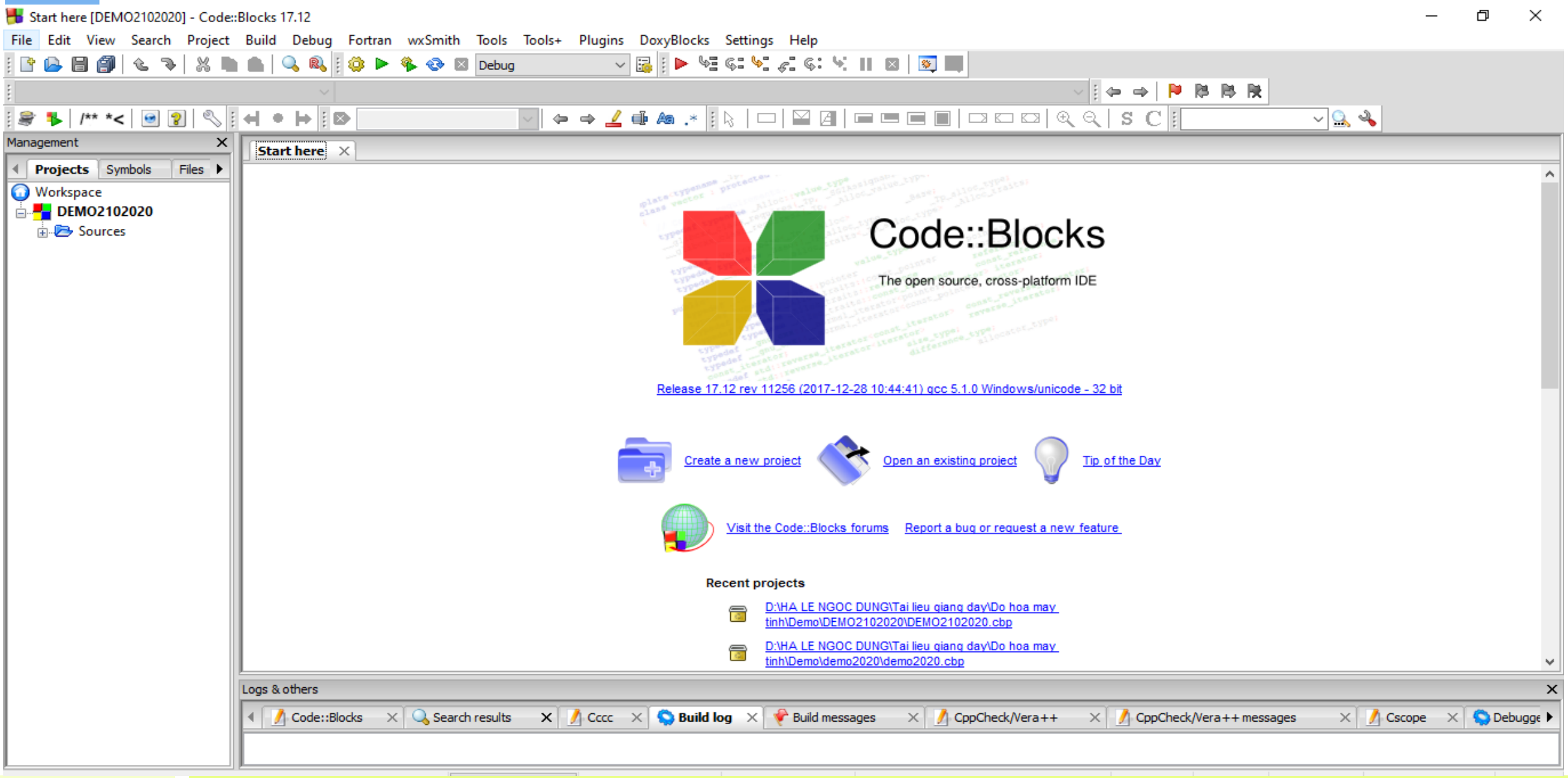




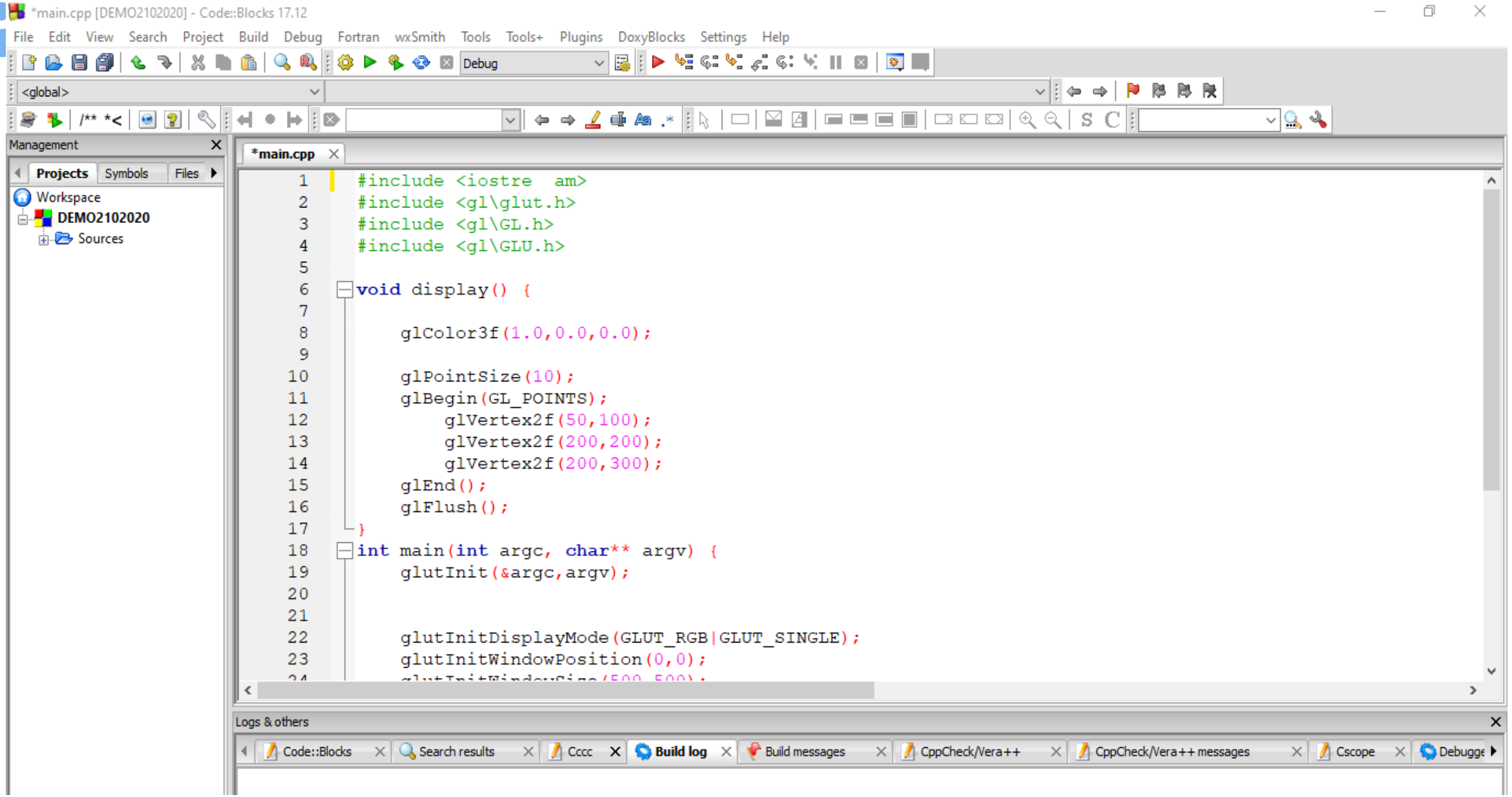




6 CÁC CÔNG CỤ TRONG GIAO DIỆN CODE::BLOCK



6 CÁC CÔNG CỤ TRONG GIAO DIỆN CODE::BLOCK



6

GIỚI THIỆU BỘ THƯ VIỆN GLUT

6 CẤU TRÚC CHƯƠNG TRÌNH VỚI OPENGL

STT	Nội dung	Ý nghĩa
1	`#include <...>`	Nhập các thư viện, file header để sử dụng các hàm glut, OpenGL.
2	**`glutInit`**`(int* pargc,char **argv)`	Khởi tạo thông số cho cửa sổ của glut. Các tham số pargc và argv được lấy từ hàm main()

6 THIẾT LẬP CHẾ ĐỘ MÀU, BUFFER CHO MÀN HÌNH

```
glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
```

glutInitDisplayMode(int mode)

GLUT_RGBA: thiết lập cho chế độ màn hình RGBA.

GLUT_RGB: tương tự GLUT_RGBA.

GLUT_INDEX: thiết lập chế độ màu mặc định.

GLUT_SINGLE: thiết lập chế độ màn hình buffer đơn.

GLUT_DOUBLE: thiết lập chế độ màn hình buffer đôi.

GLUT_ACCUM: thiết lập chế độ accumulation buffer.

GLUT_ALPHA: thiết lập chế độ đối tương anpha cho buffer màu.

GLUT_DEPTH: thiết lập cho depth buffer.

GLUT_STENCIL: thiết lập cho stencil buffer.

GLUT_MULTISAMPLE: thiết lập cho multisampling.

GLUT_STEREO: thiết lập chế độ stereo.

GLUT_LUMINANCE: thiết lập cho chế độ màu "luminance".

Sử dụng toán tử OR (ký hiệu |)để kết hợp các chế độ màu, buffer với nhau.

6 CẤU TRÚC CHƯƠNG TRÌNH VỚI OPENGL

4	`glutInitWindowPosition (int x, int y)`	Tạo độ cửa sổ màn hình
5	`glutInitWindowSize (int x, int y)`	thiết lập kích thước cửa sổ màn hình.
6	`glutCreateWindow (char* title)`	tạo cửa sổ màn hình với tiêu đề là title.
7	`glClearColor (Glclampf red, Glclampf green, Glclampf blue, Glclampf alpha)`	thiết lập chế độ màu mới cho toàn bộ ứng dụng. *(float)alpha=[0;1]*: độ mờ đục. 0: không nhìn thấy 1: nhìn thấy hoàn toàn

6 CẤU TRÚC CHƯƠNG TRÌNH VỚI OPENGL

8	**glClear** (GL_COLOR_BUFFER_BIT)	xóa mọi pixel
9	**glOrtho**(...) VD: glOrtho(0.0, 640.0, 0.0, 480.0)	Xác định một ma trận cho phép người dùng nhìn theo kiểu như hình vẽ
10	**glutDisplayFunc**(*display*)	gọi hàm để thực hiện việc vẽ khi cửa sổ màn hình hiển thị
11	**glutMainLoop**()	Hiển thị
12	**glColor3f** (red, green, blue)	thiết lập màu sắc cho đối tượng sắp vẽ.
13	**glVertex2f**(200,200)	Xác định 1 điểm với tọa độ (x,y)
14	**glPointSize**(float size)	kích thước điểm ảnh.

Giới thiệu bộ thư viện GLUT

15	**glBegin**(*int mode*) Với int mode: *GL POINT*: Vẽ điểm *GL LINES*: Vẽ đường thẳng nối hai điểm *GL LINE STRIP*: Tập hợp của những đoạn thẳng được nối	Vẽ các loại hình theo int mode
	với nhau *GL LINE LOOP*: Đường gấp khúc khép kín *GL TRIANGLES*: Vẽ hình tam giác *GL QUADS*: Vẽ tứ giác *GL TRIANGLES STRIP*: Vẽ một tập hợp các tam giác liền nhau, chung một cạnh *GL QUAD STRIP*: Vẽ một tập hợp các tứ giác liền nhau, chung một cạnh	
16	glEnd()	kết thúc quá trình vẽ.
17	glFlush()	đưa dữ liệu từ bộ nhớ tạm và màn hình.

Xin cảm ơn!