**Chương I: Thuật toán cơ sở trong đồ họa 2D**

**1.1. Thuật toán vẽ đường thẳng**

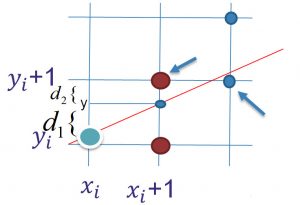
**1.1.1. Thuật toán Bressenham**

***+ Ý tưởng thuật toán:***

* Thay thế các phép toán trên số thực bằng các phép toán trên số nguyên.
* Giảm thời gian của thuật toán hơn so với DDA.
* Hạn chế phép toán được thực hiện để giảm tải thời gian.

***+ Xây dựng thuật toán***

Thuật toán Bresenham đưa ra cách chọn yi+1 là yi hay yi+1 theo một hướng khác. Đó là so sánh khoảng cách giữa điểm thực y với 2 điểm gần kề nó nhất. Nếu điểm nào nằm gần điểm thực hơn thì sẽ được chọn làm điểm vẽ tiếp theo.



Xét trường hơp 0<m<1

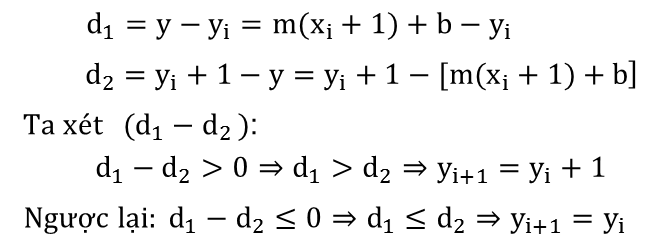
Gọi y là giá trị thực (giá trị chính xác) của đường thẳng tại x ở bước thứ i+1 .

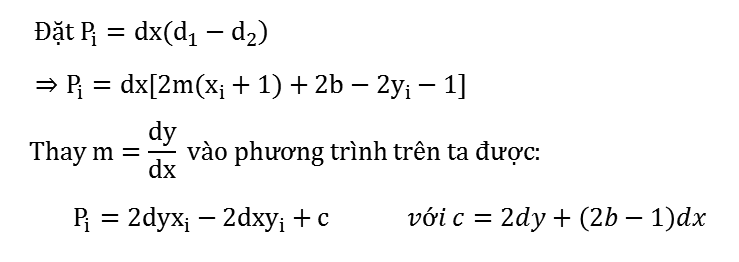
y = m(xi + 1)+b

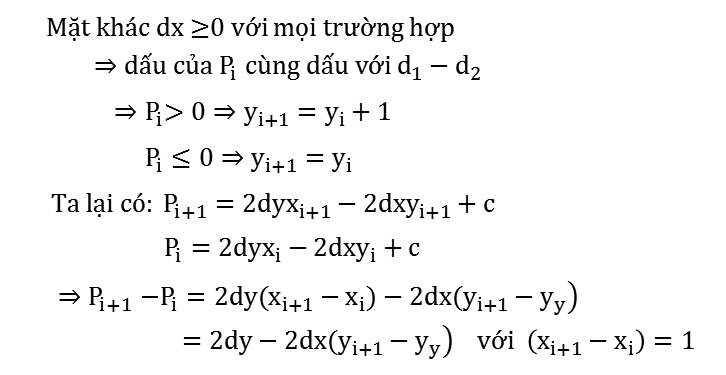
Gọi d1 là khoảng cách từ y đến yi .

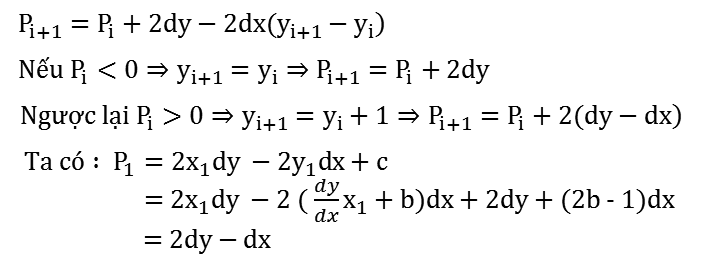
Gọi d2 là khoảng cách từ y đến yi+1 .

Ta có:

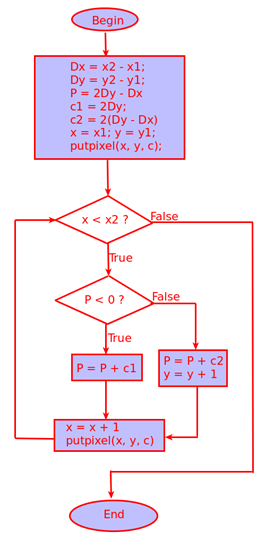
  
 Dễ thấy d1-d2 tồn tại phép toán với số thực m = dy/dx. Và để tuân thủ theo đúng ý tưởng thuật toán chỉ thực hiện các phép toán trên số nguyên, ta khử phân số (triệt tiêu mẫu số) bằng cách nhân 2 vế với dx :







***+ Lưu đồ thuật toán***



***+ Kết quả:*** Code minh họa

#include <winbgim.h>

#include<conio.h>

#define DELAY 10

int color = 2;

void Bresenham(int x1, int y1, int x2, int y2)

{

int Dx = abs(x2 - x1);

int Dy = abs(y2 - y1);

int p = 2\*Dy - Dx;

int c1 = 2\*Dy;

int c2 = 2\*(Dy-Dx);

int x = x1;

int y = y1;

int x\_unit = 1, y\_unit = 1;

putpixel(x,y,color);

while(x != x2)

{

delay(DELAY);

if (p<0) p += c1;

else

{

p += c2;

y += y\_unit;

}

x += x\_unit;

putpixel(x, y, color);

}

}

int main()

{

int x1,y1,x2,y2;

int gd,gm=VGAMAX; gd=DETECT;

initgraph(&gd,&gm,NULL);

//setbkcolor(4);

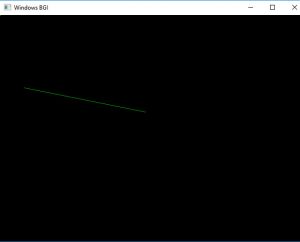
Bresenham(50,150, 300, 200);

delay(9000);

return 0;

}

Hình ảnh minh họa



**1.1.2. Thuật toán Midpoint**

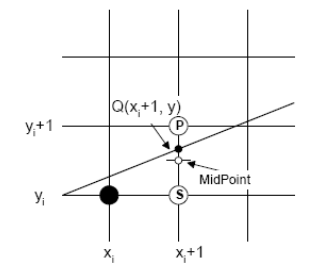
***+ Ý tưởng thuật toán***

Dù thuật toán Bresenham đã có bước chuyển mình thành công, đó là cải tiến việc tính toán trên số thực phức tạp của thuật toán DDA thành chỉ thao tác trên số nguyên, trên phép cộng và nhân, làm tăng tốc thuật toán đáng kể. Nhưng việc xây dựng trường hợp tổng quát cho thuật toán Bresenham có vẻ phức tạp hơn nhiều so với thuật toán DDA. Vì vậy, vào năm 1967, Pitteway đã công bố thuật toán MidPoint (được phát triển dựa trên thuật toán Bresenham), sau đó vào năm 1984 được Van Aken cải tiến. Thuật toán Midpoint cho kết quả hoàn toàn giống với thuật toán Bresenham, nhưng cách xây dựng thì đơn giản hơn rất nhiều.

***+ Xây dựng thuật toán***

Cho 2 điểm A(x1,y1) và B(x2,y2). Vẽ đường thẳng đi qua 2 điểm A,B

Xét 0<hệ số góc<1

  
Thuật toán Midpoint đưa ra cách chọn điểm yi+1 là yi hay yi+1 bằng cách so sánh điểm thực Q(xi+1, y) với điểm Midpoint là trung điểm của S và P.

Nếu điểm Q nằm dưới điểm Midpoint thì ta chọn điểm S là điểm vẽ tiếp theo. Ngược lại , nếu điểm Q nằm trên điểm Midpoint thì ta chọn P.

Ta có dạng tổng quát của PT đường thẳng:

Ax + By + C = 0

Với A= yo –y1; B= -(x2 – x1); C=x2y1 – x1y2

Đặt F(x,y)= Ax+ By +C

Ta có nhận xét:

Vị trí tương đối của điểm Midpoint (x,y) với đường thẳng:

* F( x, y ) < 0 nếu (x,y) nằm phía trên đường thẳng.
* F( x, y ) = 0 nếu (x,y) thuộc về đường thẳng.
* F( x, y ) > 0 nếu (x,y) nằm phía dưới đường thẳng.

Lúc này việc chọn các điểm S, P ở trên được đưa về việc xét dấu của

pi = 2F(Midpoint) = 2F(xi +1, yi +1/2)

* Nếu pi < 0 ⇒ Midpoint nằm phía trên đường thẳng ⇒Lúc này điểm thực Q nằm phía dưới điểm Midpoint ⇒ Chọn S(xi+1,yi) .
* Nếu pi >= 0 ⇒ Midpoint nằm phía dưới đường thẳng ⇒ Lúc này điểm thực Q nằm trên điểm Midpoint ⇒ Chọn P(xi+1,yi+1).

Mặt khác:

pi+1–pi = 2F(xi+1+1, yi+1+1/2) – 2F(xi+1, yi+1/2)

⇒ pi+1–pi=2[A(xi+1+1) + B(yi+1+1/2) +C] – 2[A(xi+1) + B(yi+1/2) +C]

⇒ pi+1–pi=2A + 2B(yi+1–yi)

⇒ pi+1–pi= 2Dy – 2Dx(yi+1–yi)

Vậy

pi+1=pi+ 2Dy nếu pi<0 do ta chọn yi+1=yi

pi+1=pi+ 2Dy – 2Dx nếu pi>=0 do ta chọn yi+1=yi+1

Ta tính giá trị p1 ứng với điểm ban đầu (x1, y1) với nhận xét rằng điểm (x1, y1) là điểm thuộc đường thẳng, tức là có Ax1+ By1+ C =0.

p1 = 2F(x1+1, y1+1/2) = 2[A(x1+1) +B(y1+1/2) +C]

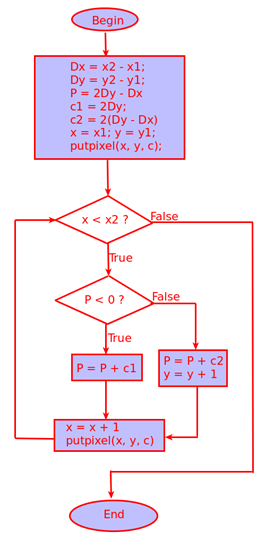
⇒ p1= 2(Ax1+ By1+C) +2A +B

⇒ p1= 2A + B

⇒ p1= 2Dy-Dx

Ta thấy kết quả của thuật toán Midpoint tương tự thuật toán Bresenham như đã nói ở trên.

***+ Lưu đồ thuật toán***



***+ Kết quả:*** Code minh họa:

#include<iostream>

#include<winbgim.h>

#include<math.h>

using namespace std;

void midpoint(int x1,int y1,int x2, int y2,int color)

{

//Truong hop 0<m<1 && x1<x2 && y1<y2

int a,b,pi,x,y,p;

a=y2-y1;

b=-(x2-x1);

y=y1;

x=x1;

putpixel(x,y,color); //Ve diem pixel dau tien

p=2\*a+b; //tinh vi tri tuong doi cua diem Midpoint so voi duong thang

while(x < x2)

{

if(p < 0)

{

p+=2\*a; // ta chon chon diem yi

}

else

{

y++;

p+=2\*(a+b);//ta chon diem yi +1

}

x++;

cout<<"Pixel (x,y) midpoint = ("<<x<<", "<<y<<")\n";

putpixel(x,y,color);

delay(10);

}

}

int main()

{

initwindow(800,600);

//khoi tao window có chieu rong x=400 và chieu cao y = 500

setwindowtitle("Thuat toan Line Midpoint");

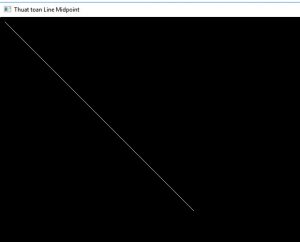
//thiet lap tieu de cho windows

midpoint(10,10,400,400,15);

getch();

}

Hình ảnh minh họa



**1.2. Thuật toán midpoint vẽ đường tròn**

***+ Ý tưởng thuật toán :***

Do tính đối xứng của đường tròn C nên ta chỉ cần vẽ 1/8 cung tròn, sau đó lấy đối xứng qua 2 trục tọa độ và 2 đường phân giác thì ta vẽ được cả đường tròn.

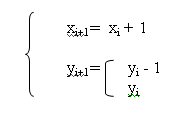
***+ Xây dựng thuật toán:***

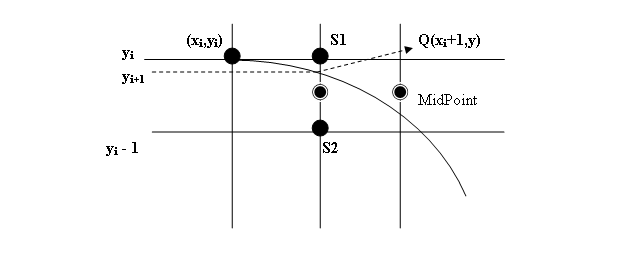
x2+ y2 = r2 => x2 + y2 – r2 = 0

Đặt f(x, y) = x2 + y2 - r2

Với mọi điểm P(x, y) nằm trong hệ tọa độ Oxy, ta có:

* P(x, y) nằm trên đường tròn O nếu f(x, y) = 0
* P(x, y) nằm ngoài đường tròn O nếu f(x, y) > 0
* P(x, y) nằm trong đường tròn O nếu f(x, y)< 0





Đặt F(x, y) = x2+ y2 - R2, ta có:

. F(x, y) < 0 , nếu điểm (x, y) nằm trong đường tròn.

. F(x, y) = 0 , nếu điểm (x, y) nằm trên đường tròn.

. F(x, y) > 0 , nếu điểm (x, y) nằm ngoài đường tròn.

Xét Pi = F(MidPoint) = F(xi+1, yi - 1/2). Ta có:

- Nếu Pi < 0: điểm MidPoint nằm trong đường tròn. Khi đó, điểm thực Q gần với điểm S1 hơn nên ta chọn yi+1 = yi.

- Nếu Pi >= 0: điểm MidPoint nằm ngòai đường tròn. Khi đó, điểm thực Q gần với điểm S2 hơn nên ta chọn yi+1 = yi - 1.

Mặt khác:

Pi+1- Pi= F(xi+1+1, yi +1- 1/2) – F(xi+1, yi -1/2)

= [(xi+1+1)2 + (yi+1 – 1/2)2 - R2] – [(xi+1)2 + (yi – 1/2)2 – R2]

= 2xi + 3 + ((yi+1)2 + (yi)2) - (yi + 1 - yi)

Vậy:

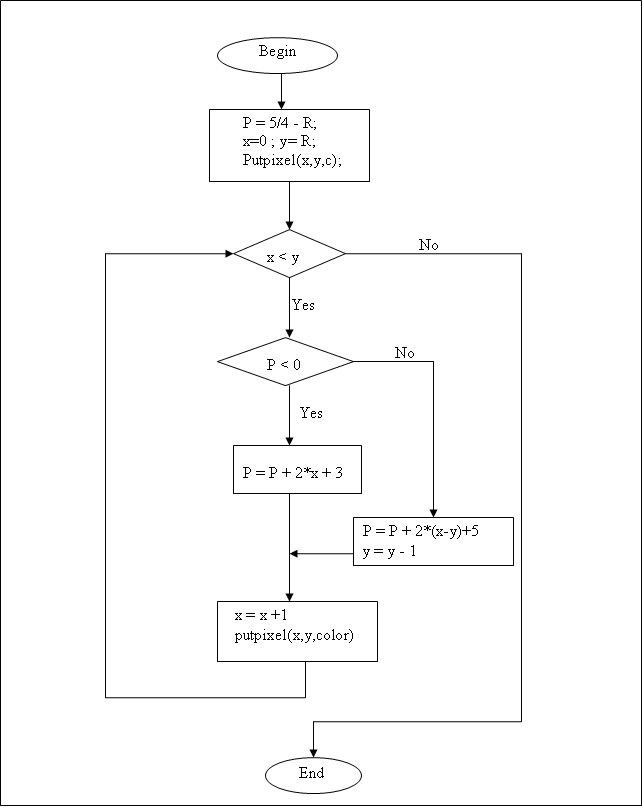
- Nếu Pi< 0 : chọn yi+1 = yi. Khi đó Pi+1 = Pi + 2xi +3

- Nếu Pi>= 0 : chọn yi+1= yi - 1. Khi đó Pi+1 = Pi + 2xi – 2yi +5.

- Pi ứng với điểm ban đầu (x0, y0) = (0,R) là:



***+ Lưu đồ thuật toán:***



***+ Kết quả:*** Code minh họa

Procedure DTR(xc, yc, r, mau : integer);

var x, y, p : integer ;

begin

x:=0 ; y:=r;

p:=1 - r;

while ( y > x) do

begin

doi\_xung;

if (p<0) then p:=p+2\*x+3

else begin

p:=p+2\*(x-y)+5 ;

y:=y-1;

end;

x:=x+1;

end;

{

while

}

end;

**1.3. Thuật toán tô màu**

**1.3.1. Thuật toán tô màu tràn**

***+ Ý tưởng thuật toán***

Thuật toán tô màu loang (hay còn gọi là tô màu theo đường biên, tô lân cận). Khác với thuật toán tô màu dựa theo dòng quét, đường biên của vùng tô màu ở thuật toán tô loang được xác định bởi tập các đỉnh của 1 đa giác, đường biên trong thuật toán được mô tả bằng một giá trị duy nhất, đó là màu của tất cả các điểm thuộc về đường biên.

***+ Lưu đồ giải thuật***

**\*Code đệ quy**

#include <conio.h>

#include <winbgim.h>

#include <iostream>

using namespace std;

struct ToaDo

{

int x, y;

};

int MauNen;

void NhapDaGiac(int &n,int &x,int &y,ToaDo a[])

{

cout<<"Nhap so dinh cua da giac n= "; cin>>n;

for (int i=1;i<=n;i++)

{

cout<<"Toa do dinh P["<<i<<"].x= "; cin>>a[i].x;

cout<<"Toa do dinh P["<<i<<"].y= "; cin>>a[i].y;

}

cout<<"Nhap diem (x,y) thuoc da giac:\n";

cout<<"nhap x="; cin>>x;

cout<<"nhap y="; cin>>y;

}

void VeDaGiac(int n,ToaDo a[],int color)

{

setcolor(color);

for (int i=1;i<=n;i++)

{

int j;

if (i==n) j=1; else j=i+1;

line(a[i].x,a[i].y,a[j].x,a[j].y);

}

}

void ToLoang(int x,int y,int color)

{

if (getpixel(x,y)==MauNen && x<getmaxx() && y<getmaxy())

{

putpixel(x,y,color);

ToLoang(x-1,y,color);

ToLoang(x,y-1,color);

ToLoang(x+1,y,color);

ToLoang(x,y+1,color);

}

delay(1);

}

int main()

{

int x,y,n,Gd,Gm=VGAMAX;

ToaDo a[10];

NhapDaGiac(n,x,y,a);

Gd=DETECT;

initgraph(&Gd,&Gm,"");

VeDaGiac(n,a,15);

MauNen=getpixel(x,y);

ToLoang(x,y,10);

getch();

closegraph();

}

**\*Code khử đệ quy**

#include <conio.h>

#include <winbgim.h>

#include <iostream>

#include <queue>

using namespace std;

struct ToaDo

{

int x,y;

};

int MauNen;

void NhapDaGiac(int &n,int &x,int &y,ToaDo a[])

{

cout<<"Nhap so dinh cua da giac n= "; cin>>n;

for (int i=1;i<=n;i++)

{

cout<<"Toa do dinh P["<<i<<"].x= "; cin>>a[i].x;

cout<<"Toa do dinh P["<<i<<"].y= "; cin>>a[i].y;

}

cout<<"Nhap diem (x,y) thuoc da giac:\n";

cout<<"nhap x="; cin>>x;

cout<<"nhap y="; cin>>y;

}

void VeDaGiac(int n,ToaDo a[],int color)

{

setcolor(color);

for (int i=1;i<=n;i++)

{

int j;

if (i==n) j=1; else j=i+1;

line(a[i].x,a[i].y,a[j].x,a[j].y);

}

}

void ToLoang(int x,int y,int color)

{

//Khai bao queue chua pixel chua duoc to mau

queue<ToaDo> Q;

ToaDo m, Tg;

if (getpixel(x,y)==MauNen  && x<getmaxx() && y<getmaxy())

{

m.x = x;

m.y = y;

putpixel(m.x, m.y, color);

Q.push(m); //Them 1 diem vao queue, queue size tang 1

while(Q.empty() == false) //Xet 4 diem xung quanh voi moi diem luu trong queue (neu queue con phan tu)

{

Q.pop(); //Xoa 1 diem phia dau queue, queue size giam 1

//Xet cac diem lan can cua 1 diem

if(getpixel(m.x+1, m.y) == MauNen)

{

putpixel(m.x+1, m.y, color );

Tg.x = m.x+1;

Tg.y = m.y;

Q.push(Tg);// Them 1 diem vao cuoi queue

}

if(getpixel(m.x-1, m.y) == MauNen)

{

putpixel(m.x-1, m.y, color);

Tg.x = m.x-1;

Tg.y = m.y;

Q.push(Tg);

}

if(getpixel(m.x, m.y+1) == MauNen)

{

putpixel(m.x, m.y+1, color);

Tg.x = m.x;

Tg.y = m.y+1;

Q.push(Tg);

}

if(getpixel(m.x, m.y-1) == MauNen)

{

putpixel(m.x, m.y-1, color);

Tg.x = m.x;

Tg.y = m.y-1;

Q.push(Tg);

}

m = Q.front();// Dua ve gia tri dau tien cho hang doi

delay(1);

}

}

}

int main()

{

int x,y,n,Gd,Gm=VGAMAX;

ToaDo a[10];

NhapDaGiac(n,x,y,a);

Gd=DETECT;

initgraph(&Gd,&Gm,"");

VeDaGiac(n,a,15);

MauNen=getpixel(x,y);

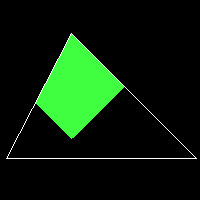
ToLoang(x,y,10);

getch();

closegraph();

}

Hình ảnh minh họa



**1.3.2. Thuật toán tô màu theo đường quét**

***+ Ý tưởng thuật toán***

Thuật toán tô màu Scanline (hay còn gọi là thuật toán tô màu theo dòng quét) được áp dụng để tô màu các đa giác lồi, lõm hay đa giác tự cắt.

Với mỗi dòng quét, ta sẽ xác định phần giao của đa giác và dòng quét, rồi tô màu các pixel thuộc đoạn giao đó. Để xác định các đoạn giao, ta tiến hành việc tìm giao điểm của dòng quét với các cạnh của đa giác, sau đó các giao điểm này sẽ được sắp theo thứ tự tăng dần của hoành độ giao điểm. Các đoạn giao chính là các đoạn thẳng được giới hạn bởi từng cặp giao điểm một.

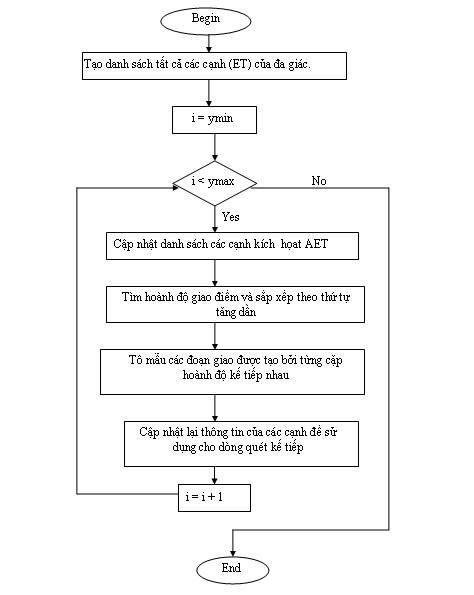
Cách thực hiện

- Tìm ymin, ymax lần lượt là giá trị nhỏ nhất, lớn nhất của tập các tung độ của các đỉnh của đa giác đã cho.

- Ứng với mỗi dòng quét y=k, k thay đổi từ ymin đến ymax, lặp :

* Tìm tất cả các hoành độ giao điểm của dòng quét y=k với các cạnh của đa giác.
* Sắp xếp các hoành độ giao điểm theo thứ tự tăng dần : x0, x1, …
* Tô màu các đoạn thẳng y=k trên đường thẳng lần lượt được giới hạn bởi các cặp ( x0, x1), ( x2, x3), ….(x2k, x2k+1).

***+ Lưu đồ thuật toán***



***+ Kết quả:*** Code minh họa

#include <conio.h>

#include <iostream>

#include <graphics.h>

#include <stdlib.h>

using namespace std;

struct ToaDo

{

int x,y;

};

void nhapDaGiac(ToaDo p[], int v)

{

int i;

for(i=0;i<v; i++){

cout<<"\nNhap toa do dinh "<<i+1<<" : ";

cout<<"\n\tx["<<(i+1)<<"] = "; cin>>p[i].x;

cout<<"\n\ty["<<(i+1)<<"] = "; cin>>p[i].y;

}

p[i].x=p[0].x;

p[i].y=p[0].y;

}

void veDaGiac(ToaDo p[], int v)

{

for(int i=0;i<v;i++)

line(p[i].x,p[i].y,p[i+1].x,p[i+1].y);

}

void ScanLine(ToaDo p[], int v)

{

int xmin,xmax,ymin,ymax,c,mang[50];

xmin=xmax=p[0].x;

ymin=ymax=p[0].y;

for(int i=0;i<v;i++){ if(xmin>p[i].x) xmin=p[i].x;

if(xmax<p[i].x) xmax=p[i].x; if(ymin>p[i].y) ymin=p[i].y;

if(ymax<p[i].y) ymax=p[i].y;

}

float y;

y=ymin+0.01;

while(y<=ymax){ //với y tăng dần từ ymin > ymax,tìm các giao điểm của từng y với các cặp cạnh

int x,x1,x2,y1,y2,tg;

c=0; //chỉ số của mảng phần tử

for(int i=0;i<v;i++)

{ //xét trên tất cả các đỉnh

//xét 2 đỉnh liền kề nhau

x1=p[i].x;

y1=p[i].y;

x2=p[i+1].x;

y2=p[i+1].y;

if(y2<y1)

{ //sắp xếp lại y của 2 điểm liên tiếp

tg=x1;x1=x2;x2=tg;

tg=y1;y1=y2;y2=tg;

}

//mảng giao điểm

if(y<=y2&&y>=y1)

{

if(y1==y2) x=x1; //nếu y của 2 đỉnh liên tiếp trùng nhau => bỏ qua

else

{

x=((y-y1)\*(x2-x1))/(y2-y1); //hệ số góc

x+=x1; //300

}

if(x<=xmax && x>=xmin)

mang[c++]=x; //cho phần tử c = x sau đó c++

}

}

//với từng y tăng dần ta vẽ luôn đường thằng nối 2 giao điểm

for(int i=0; i<c;i+=2)

{

delay(30);

line(mang[i],y,mang[i+1],y);

} //line(302,91,300,91

y++;

}

}

//nhap 10 dinh:

//1.(75,250),2.(210,250),3.(250,128),4.(291,250),5.(425,250)

//6.(318,331),7.(360,460),8.(249,380),9.(140,460),10.(182,331)

//3 7 1 5 9

int main()

{

int cl,v;

do

{

cout<<"\n Nhap so dinh da giac:"; cin>>v;

}while(v<3);

ToaDo p[v];

nhapDaGiac(p,v);

cout<<"\nChon mau (0-15) : "; cin>>cl;

initwindow(500,600);

veDaGiac(p,v);

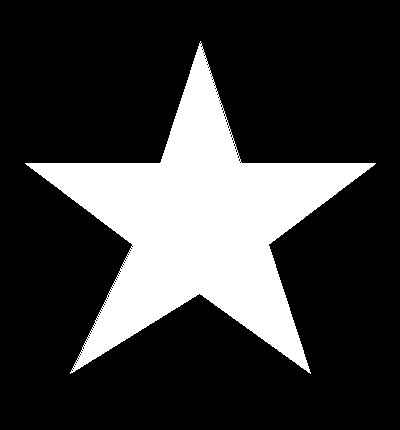
setcolor(cl);

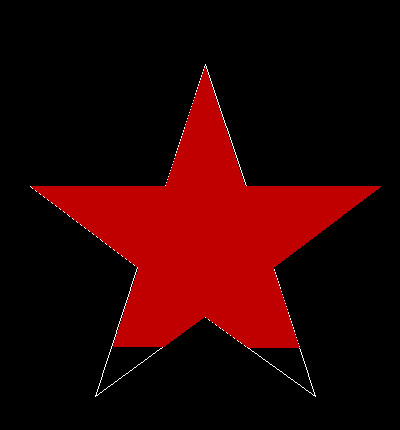
ScanLine(p, v);

getch();

}

Hình ảnh minh họa





**Chương 2: Xây dựng ứng dụng đồ họa**

**2.1. Phát biểu bài toán**

- Qua tìm hiểu lí thuyết về một số kỹ thuật đồ họa 2D và 3D của OpenGL. Chúng em đã vận dụng những kiến thức đã thu thập được để sử dụng kỹ thuật đồ họa 2D và 3D để thiết kế và xây dựng chương trình sử dụng tập hợp các khối cầu cách đều nhau để mô phỏng chuyển động của lốc xoáy.

- Mục tiêu của chương trình là mô phỏng giao diện, chuyển động của lốc xoáy. Đây là một mô phỏng hạt được lập trình trong Visual Studio bằng OpenGL và GLUT. Chương trình mô phỏng có khả năng thay đổi tốc độ xoay của cơn lốc xoáy, chiều cao của cơn lốc, kích thước của cơn lốc…

**2.2. Các bước giải quyết**

*- Vấn đề 1:* Tạo giao diện cho chương trình.

*- Giải quyết:* Dựa vào kiến thức 2D và 3D đã học khai báo các thư viện cần thiết, khởi tạo khung nhìn, đặt ra tọa độ, góc nhìn, ánh sáng, màu nền, đổ bóng đối tượng... cho chương trình.

#include <math.h>

#include<GL/glut.h>

#define WIDTH 700

#define HEIGHT 700

#define INTERVAL 15

#define MAX 20

int POS\_X, POS\_Y;

GLfloat BG\_Color[] = { 0.2f, 0.6f, 0.6f, 1.0f };

GLfloat Light\_Pos[] = { 0.0f, 50.0f, 80.0f, 0.0f };

GLfloat Light\_Dif[] = { 1.0f, 1.0f, 1.0f, 1.0f };

GLfloat Ambient[] = { 1.0f, 1.0f, 1.0f, 1.0f };

GLfloat Diffuse[] = { 0.6f, 0.6f, 0.6f, 1.0f };

GLfloat Specular[] = { 1.0f, 1.0f, 1.0f, 1.0f };

float Angle = 0.0f;

void Init()

{

glMatrixMode(GL\_PROJECTION);

glLoadIdentity();

glFrustum(-1.0, 1.0, -1.3, 0.7, 1, 50);

glTranslatef(0.0f, -2.5f, -7.5f);

glRotatef(16.0f, 1.0f, 0.0f, 0.0f);

glViewport(0, 0, WIDTH, HEIGHT);

glMatrixMode(GL\_MODELVIEW);

glClearColor(BG\_Color[0], BG\_Color[1], BG\_Color[2], BG\_Color[3]);

glEnable(GL\_DEPTH\_TEST);

glEnable(GL\_LIGHT0);

glLightfv(GL\_LIGHT0, GL\_DIFFUSE, Light\_Dif);

glLightfv(GL\_LIGHT0, GL\_POSITION, Light\_Pos);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_AMBIENT, Ambient);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_DIFFUSE, Diffuse);

glMaterialfv(GL\_FRONT\_AND\_BACK, GL\_SPECULAR, Specular);

glMaterialf(GL\_FRONT, GL\_SHININESS, 90.0f);

glEnable(GL\_LIGHTING);

}

*- Vấn đề 2:* Vẽ lốc xoáy tĩnh

*- Giải quyết:* Lốc xoáy được tạo bởi các vòng tròn là tập hợp các khối cầu bằng nhau, cách đều nhau. Các vòng tròn cũng cách đều nhau và có bán kính giảm dần từ trên xuống dưới.

void Display()

{

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glLoadIdentity();

int i, j, k;

float r = 3.0f, h = 3.0f, Offset\_Angle;

for (i = 0; i < 35; i++)

{

k = 60 - i \* 1.2f;

Offset\_Angle = 360.0f / k;

for (j = 0; j < k; j++)

{

glLoadIdentity();

glRotatef(j \* Offset\_Angle - Angle, 0.0f, 1.0f, 0.0f);

glTranslatef(r, h, 0.0f);

glutSolidSphere(0.05, 10, 10);

}

h -= 0.4f;

r \*= 0.93f;

}

glLoadIdentity();

glutSwapBuffers();

}

*- Vấn đề 3:* Tạo chuyển động cho lốc xoáy

*- Giải quyết:* Sử dụng hàm timer để cố định khoảng thời gian vẽ lại để tạo chuyển động xoay cho lốc xoáy.

void Timer(int value)

{

Angle -= 1.0f;

glutPostRedisplay();

glutTimerFunc(INTERVAL, Timer, 0);

}

*\* Hàm main của chương trình:*

int main(int argc, char\*\* argv)

{

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_RGB | GLUT\_DOUBLE | GLUT\_DEPTH);

POS\_X = (glutGet(GLUT\_SCREEN\_WIDTH) - WIDTH) >> 1;

POS\_Y = (glutGet(GLUT\_SCREEN\_HEIGHT) - HEIGHT) >> 1;

glutInitWindowPosition(POS\_X, POS\_Y);

glutInitWindowSize(WIDTH, HEIGHT);

glutCreateWindow("Tornado");

Init();

glutTimerFunc(0, Timer, 0);

glutDisplayFunc(Display);

glutMainLoop();

return 0;

}

2.3. Kết quả thực nghiệm.

- Sau khi thực hiện giải quyết các vấn đề trên chúng ta sẽ có được một chương trình cho ra kết quả là một cơn lốc xoáy chuyển động trong môi trường 3D như sau:

