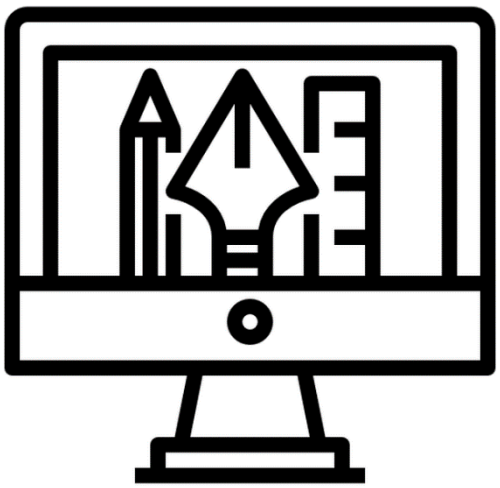
ĐẠI HỌC NHA TRANG
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO MÔN HỌC

KĨ THUẬT ĐỒ HỌA



- Nha Trang -
- 2021 -

PHỤ LỤC

THÔNG TIN BÁO CÁO

Báo cáo môn học: Kĩ Thuật Đồ Hoạ
Giáo viên bộ môn: Nguyễn Đình Cường
Họ và tên sinh viên: Nguyễn Văn Hải Long
MSSV: 60136035

CHI TIẾT BÁO CÁO

1. Các thuật toán cơ bản và các phép biến đổi hình học cơ bản

1.1. Thuật toán DDA - Digital Differential Analyzer

Thuật toán DDA - Digital Differential Analyzer (bộ phân tích vi sai) có thể tóm tắt qua các bước một cách tổng quan:

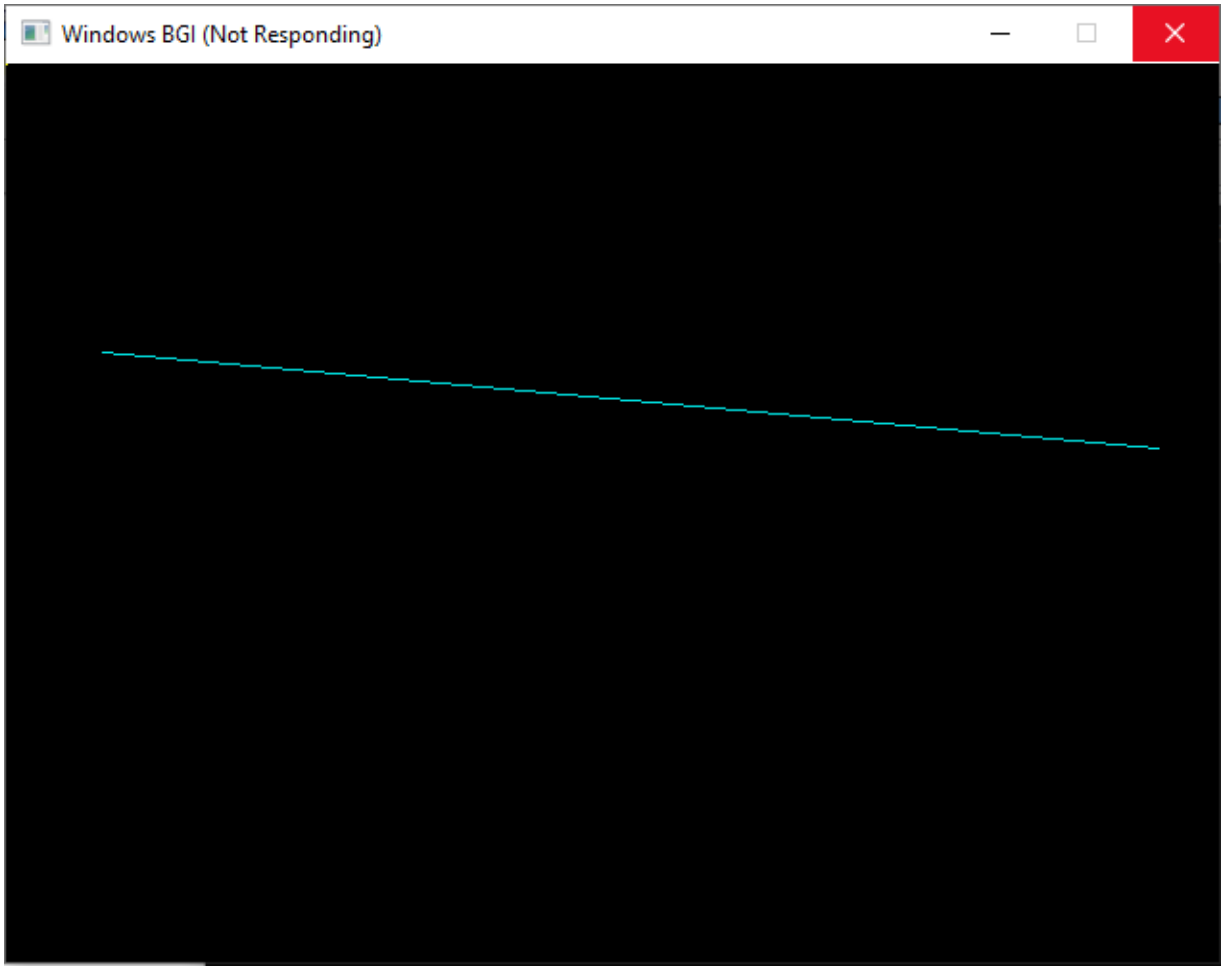
- Bước 1: Giả sử tọa độ của hai điểm A(x_A, y_A), B(x_B, y_B) không trùng nhau với điều kiện là x_A, y_A, x_B, y_B đều là số nguyên.
- Bước 2: Tính số điểm ảnh của đường thẳng được vẽ thêm trên màn hình.
  - So sánh trị tuyệt đối của $d_x = x_B - x_A$, $d_y = y_B - y_A$ và lấy trị tuyệt đối lớn nhất vì xác định càng nhiều số giao điểm thì đường thẳng càng rõ và mịn.
  - Gọi là steps là số điểm ảnh được vẽ thêm, khi đó steps = max($|d_x|$, $|d_y|$).
- Bước 3: Sau khi xác định được số giao điểm steps sẽ xác định được giá trị cộng vào cho x và y bắt đầu từ tọa độ A(x_A, y_A) cho tới tọa độ điểm B(x_B, y_B). Các giá trị này sẽ được tính như sau: x_inc = d_x / steps, y_inc = d_y / steps, giá trị là số thực.
- Bước 4: Sau khi có tất cả các thông số cần thiết bao gồm A(x_A, y_A), B(x_B, y_B), steps, x_inc, y_inc, tiến hành bước cuối cùng là tìm các tọa độ cần vẽ. Chỉ cần cộng x_A, y_A với x_inc, y_inc theo công thức x_i+1 = x_i + x_inc, y_i+1 = y_i +y_inc. Sau đó làm tròn kết quả về số nguyên để ra các tọa độ tiếp theo.

a.Vẽ đường thẳng

Triển khai chương trình:

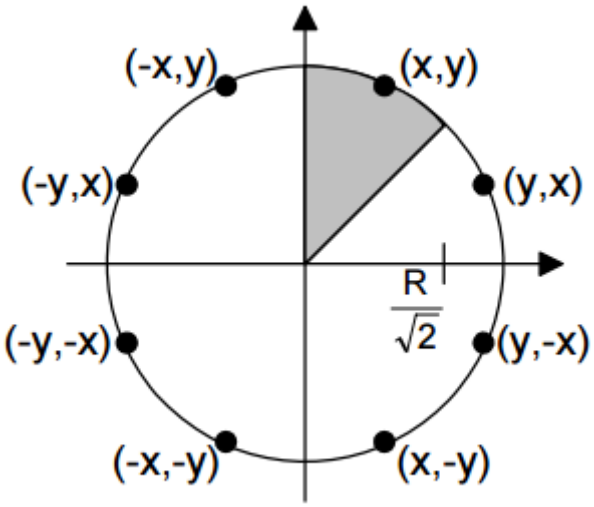
```
#include <graphics.h>
#define Round(a) (int)(a+0.5)   // lam tron so
int color = 3;//Mau ve tu 0-15

void lineDDA(int x1, int y1, int x2, int y2){       // thuat toan DDA
    int x_unit = 1, Dx = x2 - x1, Dy = y2 - y1;     // Khoi tao cac gia tri ban dau
    int x = x1;
    float y = y1;
    float m = (float)Dy/Dx;     // he so goc m
    putpixel(x, Round(y), color);

    while(x < x2){
        delay(10);  // thoi gian tre khi ve 1 diem anh
        x += x_unit;
        y += m;
        putpixel(x,Round(y), color);
    }
}
int main(){
    int gm,gd=DETECT;
    initgraph(&gd,&gm,NULL);      // khoi tao cua so do hoa
    lineDDA(50,150, 300, 200);      // ve duong thang
    getchar();
    return 0;
}
```

Kết quả chương trình:



b.Vẽ đường tròn

Sử dụng thuật toán DDA xác định 8 điểm của đường tròn và góc giữa hai đường thẳng liền kề:



Xác định 8 điểm:

```
putpixel(x + xc, y + yc, color);
putpixel( y + xc, x + yc, color);
```

```
putpixel(-y + xc, x + yc, color);
putpixel(-x + xc, y + yc, color);
putpixel(-x + xc, -y + yc, color);
putpixel(-y + xc, -x + yc, color);
putpixel(y + xc, -x + yc, color);
putpixel(x + xc, -y + yc, color);
```

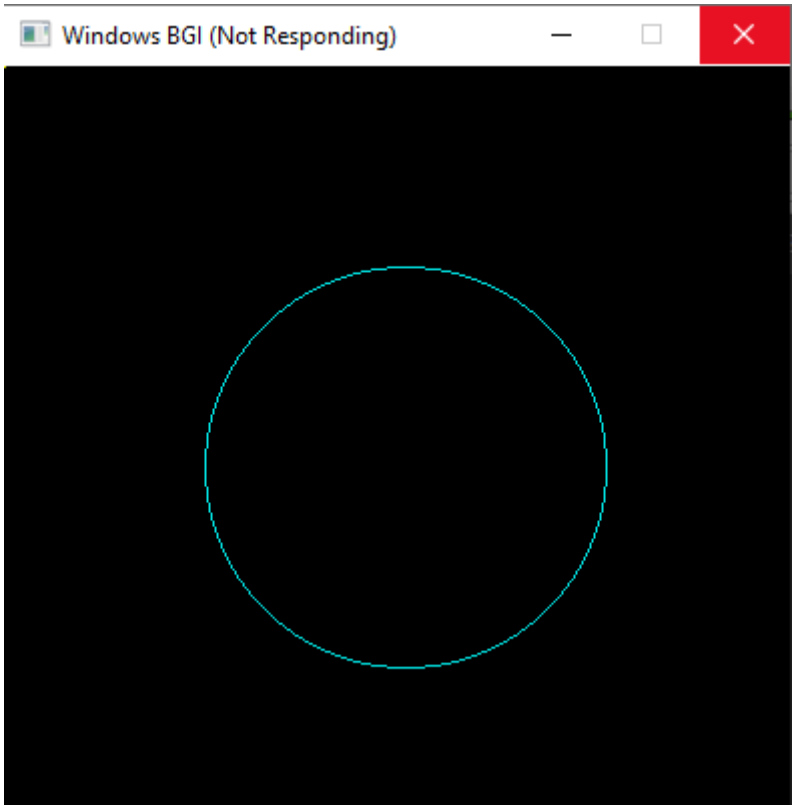
Triển khai thuật toán:

```
void DDA(int xc, int yc, int r)
{
	int x = 0, y = r;
	drawcircle(x, y, xc, yc);
	while(x<=y)
	{
		x++;
		y = ROUND(sqrt(pow(r,2)-pow(x,2)));
		drawcircle(x, y, xc, yc);
	}
}
```

Triển khai chương trình:

```
#include<graphics.h>
#include<math.h>
#define color 3
#define ROUND(a) ((int)(a+0.5))
void drawcircle(int x, int y, int xc, int yc)
{
    putpixel(x + xc, y + yc, color);
    putpixel( y + xc, x + yc, color);
    putpixel(-y + xc, x + yc, color);
    putpixel(-x + xc, y + yc, color);
    putpixel(-x + xc, -y + yc, color);
    putpixel(-y + xc, -x + yc, color);
    putpixel(y + xc, -x + yc, color);
    putpixel(x + xc, -y + yc, color);
}
void DDA(int xc, int yc, int r)
{
    int x = 0, y = r;
    drawcircle(x, y, xc, yc);
    while(x<=y)
    {
        x++;
        y = ROUND(sqrt(pow(r,2)-pow(x,2)));
        drawcircle(x, y, xc, yc);
    }
}
int main()
{
    initwindow(400,400);
    DDA(200,200,100);
    getchar();
    return 0;
}
```

Kết quả chương trình:

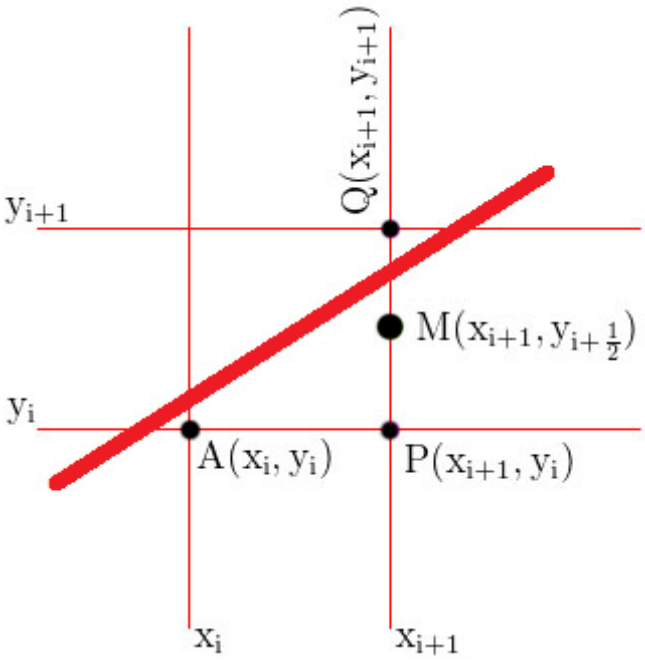


1.2. Thuật toán Midpoint

Thuật toán Midpoint (trung điểm) có thể mô tả như sau:

Đối với việc lựa chọn điểm vẽ tiếp theo sau A, có hai lựa chọn đó là điểm P và Q với M là trung điểm giữa chúng. Việc quyết định chọn điểm nào để vẽ tiếp theo phụ thuộc vào vị trí của M so với đường thẳng đang vẽ. Nếu M nằm phía dưới đường thẳng, chọn điểm Q. Ngược lại, nếu M nằm phía trên đường thẳng ta chọn điểm P. Các trường hợp xét dưới đây với $0 < m < 1$.

a.Vẽ đường thẳng



Phương trình đường thẳng là:
$y = m * x + b, m = d_y / d_x$
$\Leftrightarrow y = d_x / d_x * x + b$
$\Leftrightarrow d_y * x + (-d_x * y) + b * d_x = 0$ (1)

Mà:
$F(x, y) = a * x + b * y + c = 0$ (2)
(1), (2) $\Rightarrow a = d_y, b = -d_x, c = b * d_x$

Tìm giá trị d
$F(M) = F(x_i + 1, y_i + 1 / 2)$
$= F(x_i, y_i) + a + b / 2 = 0 + d_y + d_x / 2 = d_y + d_x / 2$

Tìm giá trị d_{max}
Đặt d = F(M)

- Nếu d > 0, chọn điểm Q để vẽ.
- Nếu d < 0, chọn điểm P để vẽ.

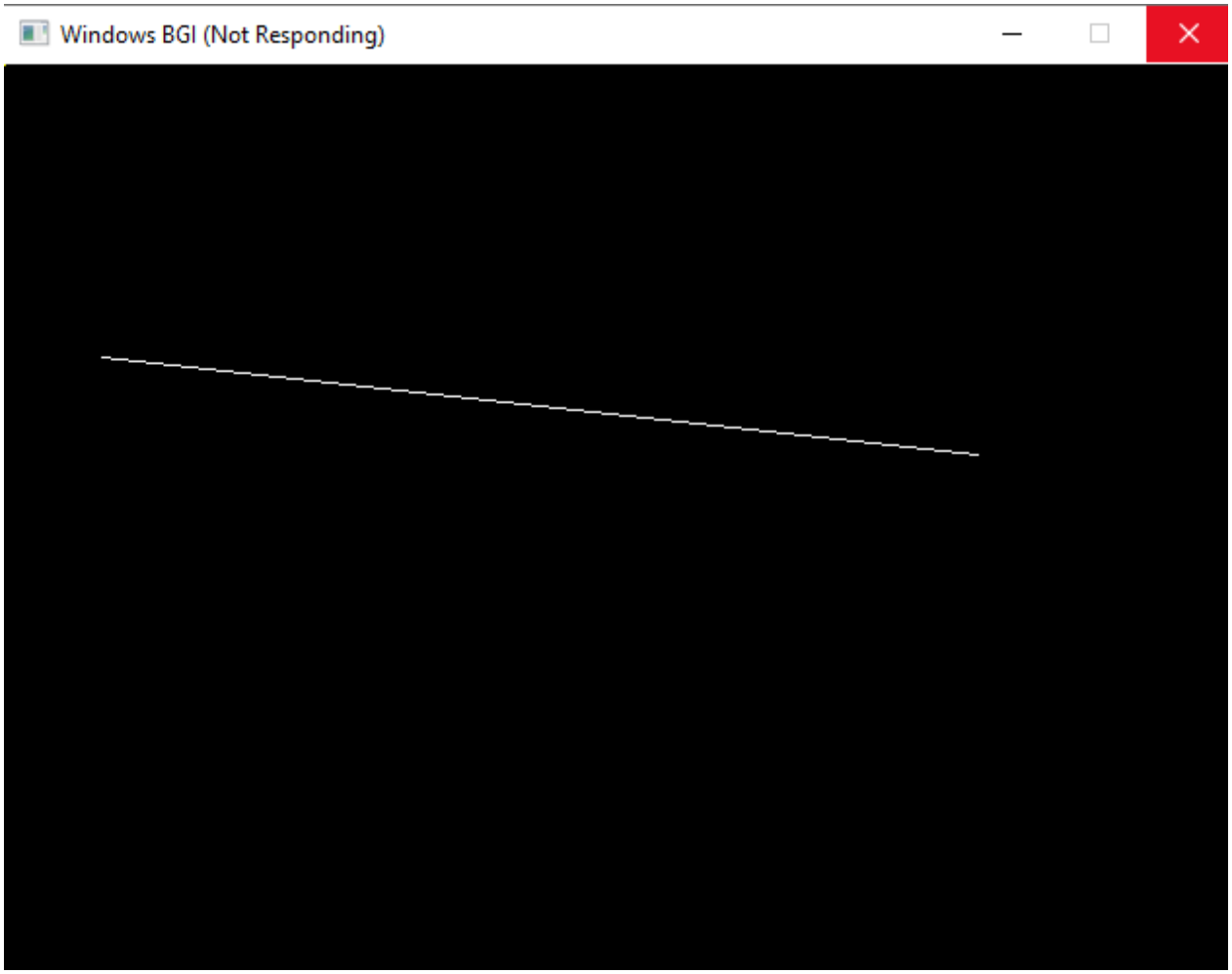
Trường hợp 1 - chọn Q
$M_{new} * (x_i + 2, y_i + 1 / 2) \Rightarrow d_{new} = F(x_i + 2, y_i + 1 / 2)$
$(\Delta d)Q = d_{new} - d_{old}$
$= F(x_i + 1, y_i + 1 / 2) - F(x_i + 2, y_i + 1 / 2) = a$
$= d_y$
$\Rightarrow d_{new} = d_{old} + d_y$

Trường hợp 2 - chọn P
$M_{new} * (x_i + 2, y_i + 3 / 2) \Rightarrow d_{new} = F(x_i + 2, y_i + 3 / 2)$
$(\Delta d)P = d_{new} - d_{old}$
$= F(x_i + 1, y_i + 1 / 2) - F(x_i + 2, y_i + 3 / 2) = a + b$
$= d_y - d_x$
$\Rightarrow d_{new} = d_{old} + d_y - d_x$

Triển khai chương trình:

```c
#include <graphics.h>

void Midpoint(int x1, int y1, int x2, int y2,int c){
    int x, y, dx, dy, d;
    y=y1;
    dx=x2-x1;
    dy=y2-y1;
    d=dy-dx/2;
    for (x=x1;x<=x2;x++){
    putpixel(x,y,c);
    if(d<=0)
        d=d+dy;
    else
        {
        y++;
        d=d+dy-dx;
        }
    delay(10);
    }
}
int main(){
    int gm,gd=DETECT;
    initgraph(&gd,&gm,NULL);
    Midpoint(50,150, 500, 200,15);        // ve duong thang voi c la mau tu 0-15
    getchar();
    return 0;
}
```

Kết quả chương trình:



b.Vẽ đường tròn

Đường tròn có tâm O(xc, yc) = (0, 0), bán kính r có phương trình:

$$x^2 + y^2 = r^2 => x^2 + y^2 - r^2 = 0$$

Đặt $f(x, y) = x^2 + y^2 - r^2$

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

- 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

Sau đó vẽ 8 điểm đối xứng. Giả sử đã vẽ được (Xi, Yi) ở bước thứ i, cần xác định (X_{i+1}, Y_{i+1}) ở bước thứ i + 1.

$X_{i+1} = X_i + 1$

$Y_{i+1} \in \{Y_i, Y_i - 1\}$

Tính Fi

Đặt $F_i = F(X, Y - 1/2)$:

$F(X_i + 1, Y_i - 1/2) = (X_i + 1)^2 + (Y_i - 1/2)^2 - R^2$

$F_i = X_i^2 + 2X_i + Y_i^2 - Y_i + 5/4 - R^2$

Nếu $F_i < 0 \Leftrightarrow (X_{i+1}, Y)$ gần với $Y_i \Rightarrow Y_{i+1} = Y_i$

Nếu $F_i \geq 0 \Leftrightarrow (X_{i+1}, Y)$ gần với $Y_{i-1} \Rightarrow Y_{i+1} = Y_{i-1}$

Tính Fi +1 theo Fi

$F_{i+1} - F_i = 2X_i + 3 + (Y_i + 12 - Y_i^2) + (Y_{i+1} - Y_i)$ (*)

Nếu $F_i < 0$ thì $F_{i+1} = F_i + 2X_i + 3$, do thay thế $Y_{i+1} = Y_i$ vào (*)

Nếu $F_i \geq 0$ thì $F_{i+1} = F_i + 2(X_i - Y_i) + 5$, do thay thế $Y_{i+1} = Y_{i-1}$ vào (*)

Tính giá trị F đầu tiên

$F(X_i + 1, Y_i - 1/2) = (X_i + 1)^2 + (Y_i - 1/2)^2 - R^2$

$F_i = X_i^2 + 2X_i + Y_i^2 - Y_i + 5/4 - R^2$

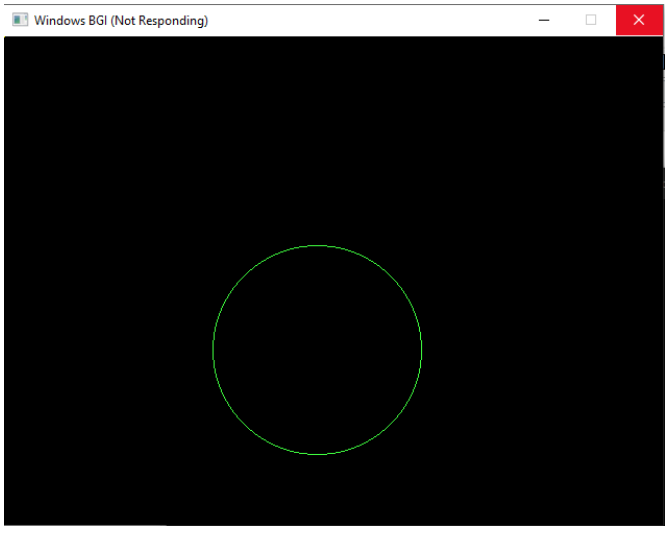
Thay $X_i = 0$ và $Y_i = R$ trong công thức trên:

$F = 5/4 - R$

Triển khai chương trình:

```c
#include<stdio.h>
#include<graphics.h>
#include<math.h>
int color=10;
void ve8diem(int xc, int yc, int x, int y, int color)
{
    putpixel(x + xc, y + yc, color);
    putpixel(-x + xc, y + yc, color);
    putpixel(x + xc, -y + yc, color);
    putpixel(-x + xc, -y + yc, color);
    putpixel( y + xc, x + yc, color);
    putpixel(-y + xc, x + yc, color);
    putpixel(y + xc, -x + yc, color);
    putpixel(-y + xc, -x + yc, color);
}
void Midpoint(int xc, int yc, int r, int color)
{
    int x = 0, y = r;
    int f = 1 - r;
    ve8diem(xc, yc, x, y, color);
    while (x < y)
    {
        if (f < 0) f += (x << 1) + 3;
        else
        {
            y--;
            f += ((x - y) << 1) + 5;
        }
        x++;
        ve8diem(xc, yc, x, y, color);
        delay(100);
    }
}
int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "c:\\tc\\bgi");
    Midpoint(300, 300, 100, color);
    getchar();
    return 0;
}
```

Kết quả chương trình:



1.3. Thuật toán Bresenham

Mô tả thuật toán Bresenham: thuật toán đưa ra cách chọn y_{i+1} là y_i hay y_{i+1} theo một hướng khác (giảm thời gian thực hiện hơn so với DDA). 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 hợp 0<m<1

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

$y = m(x_i + 1)+b$

Gọi d_1 là khoảng cách từ y đến y_i.

Gọi d_2 là khoảng cách từ y đến y_{i+1}.

Ta có:

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

$d_2 = y_i + 1 - y = y_i + 1 - [m(x_i + 1) + b]$

Ta xét $(d_1 - d_2)$:

$d_1 - d_2 > 0 \Rightarrow d_1 > d_2 \Rightarrow y_i + 1$

Ngược lại: $d_1 - d_2 \leq 0 \Rightarrow d_1 \leq d_2 \Rightarrow y_i+1 = y_i$

$\Rightarrow$ $d_1 - d_2 = [m(x_i +1) + b - y_i] - [y_i +1 - m(x_i +1) -b]$

$= m(x_i +1) +b - y_i - y_i -1 + m(x_i +1) +b$

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

Dễ thấy $d_1 - d_2$ 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:

Mặt khác $dx \geq 0$ với mọi trường hợp

$\Rightarrow$ dấu của P_i cùng dấu với $d_1 - d_2$

$\Rightarrow P_i > 0 \Rightarrow y_{i+1} = y_i + 1$

$P_i \leq 0 \Rightarrow y_{i+1} = y_i$

Ta lại có: $P_{i+1} = 2dyx_{i+1} - 2dxy_{i+1} + c$

$P_i = 2dyx_i - 2dxy_i + c$

$\Rightarrow P_{i+1} - P_i = 2dy(x_{i+1} - x_i) - 2dx(y_{i+1} - y_y)$

$= 2dy - 2dx(y_{i+1} - y_y)$ với $(x_{i+1} - x_i) = 1$

$P_{i+1} = P_i + 2dy - 2dx(y_{i+1} - y_i)$

Nếu $P_i < 0 \Rightarrow y_{i+1} = y_i \Rightarrow P_{i+1} = P_i + 2dy$

Ngược lại $P_i > 0 \Rightarrow y_{i+1} = y_i + 1 \Rightarrow P_{i+1} = P_i + 2(dy - dx)$

Ta có : $P_1 = 2x_1dy - 2y_1dx + c$

$= 2x_1dy - 2\left(\frac{dy}{dx}x_1 + b\right)dx + 2dy + (2b - 1)dx$

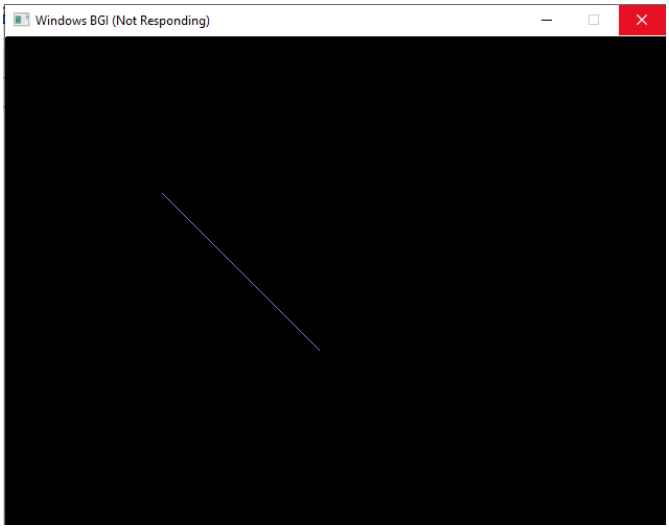
$= 2dy - dx$

a.Vẽ đường thẳng

Phát triển chương trình:

```
#include <graphics.h>
#define DELAY 10
int color = 9;
void Bresenham(int x1, int y1, int x2, int y2){
    int x, y, Dx, Dy, p, c1, c2;
    Dx = abs(x2 - x1);
    Dy = abs(y2 - y1);
    p = 2*Dy - Dx;
    c1 = 2*Dy;
    c2 = 2*(Dy-Dx);
    x = x1;
    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 gm,gd=DETECT;
    initgraph(&gd,&gm,NULL);
    Bresenham(150,150, 300, 600);        // ve duong thang
    getchar();
    return 0;
}
```

Kết quả chương trình:



b.Vẽ đường tròn

Bước 1:

- Chọn điểm đầu cần vẽ (x,y) = (0,R).
- Tính p đầu tiên: p = 3 - 2R.
- Vẽ 8 điểm ứng với (x,y).

Bước 2:

- Tăng x lên 1 pixel: x = x + 1.
- Nếu p < 0: p = p + 4x + 6.

Ngược lại: p = p + 4(x - y) + 10 và y = y – 1.

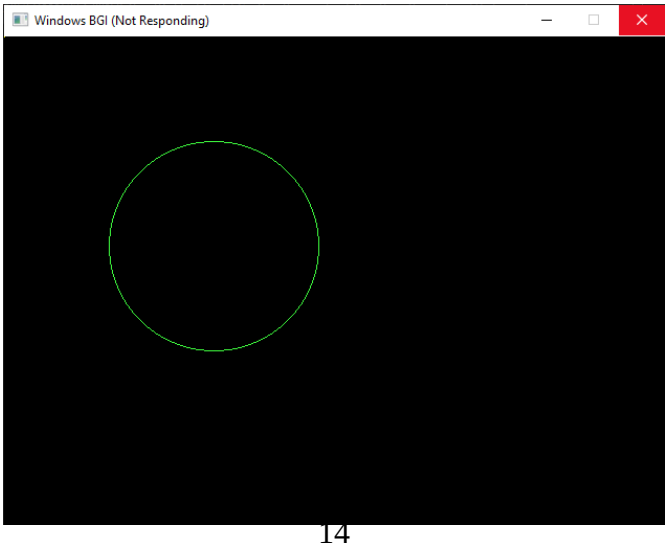
- Vẽ điểm 8 điểm ứng với (x,y) mới.

Bước 3: Lặp lại bước 2 cho đến khi x = y.

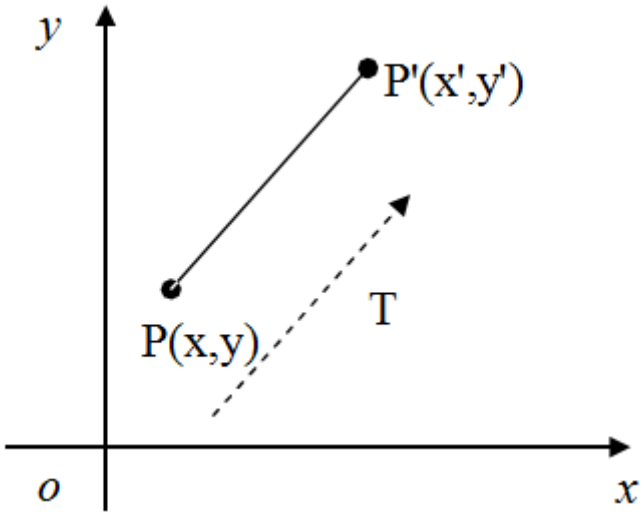
Triển khai chương trình:

```
#include<stdio.h>
#include<graphics.h>
#include<conio.h>
#include<math.h>
int color=10;

void ve8diem(int xc, int yc, int x, int y, int color)
{
    putpixel(x + xc, y + yc, color);
    putpixel(-x + xc, y + yc, color);
    putpixel(x + xc, -y + yc, color);
    putpixel(-x + xc, -y + yc, color);
    putpixel( y + xc, x + yc, color);
    putpixel(-y + xc, x + yc, color);
    putpixel(y + xc, -x + yc, color);
    putpixel(-y + xc, -x + yc, color);
}
void Bresenham(int xc, int yc, int r, int color)
{
    int x=0,y=r;
    int p=3-2*r;
    while (x <= y)
    {
        ve8diem(xc, yc, x, y, color);
        if (p < 0) p = p+4*x+6;
        else
        {
            p = p+4*(x-y)+10;
            y--;
        }
        x++;
        delay(100);
    }
}
int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, "c:\\tc\\bgi");
    Bresenham(200, 200, 100, color);
    getchar();
    return 0;
}
```

Kết quả chương trình:



1.4 Phép tịnh tiến



Nếu ta kí hiệu tọa độ các điểm và véc tơ như sau:

$$P=\begin{bmatrix} x \\ y \end{bmatrix},\ P'=\begin{bmatrix} x' \\ y' \end{bmatrix},\ T=\begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

khi đó phép tịnh tiến được mô tả bởi phương trình:

$$P'=\begin{bmatrix} x' \\ y' \end{bmatrix}=P+T=\begin{bmatrix} x \\ y \end{bmatrix}+\begin{bmatrix} t_x \\ t_y \end{bmatrix}$$

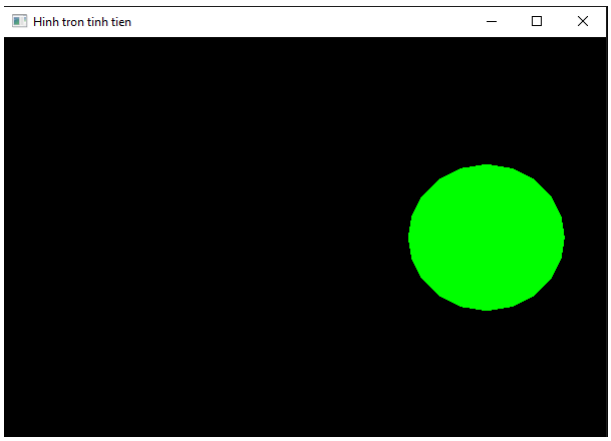
Phép tịnh tiến không làm thay đổi hình dạng của vật thể.

Triển khai chương trình:

```
#include<iostream>
#include<stdlib.h>
#ifdef __APPLE__
#else
#include<GL/glut.h>
#endif
using namespace std;
float ballX = -0.5f;
float ballY = 0.0f;
float ballZ = -1.0f;
static int flag=1;
void drawBall(void) {
        glColor3f(0.0, 1.0, 0.0); //dat mau hinh tron
        glTranslatef(ballX,ballY,ballZ);
        glTranslatef(ballX+1.5,ballY,ballZ);
        glutSolidSphere (0.3, 20, 20); //
        }
void keyPress(unsigned char key,int x,int y)
{
    switch(key)
    {
       case 27:
           exit(0);
    }
}
```

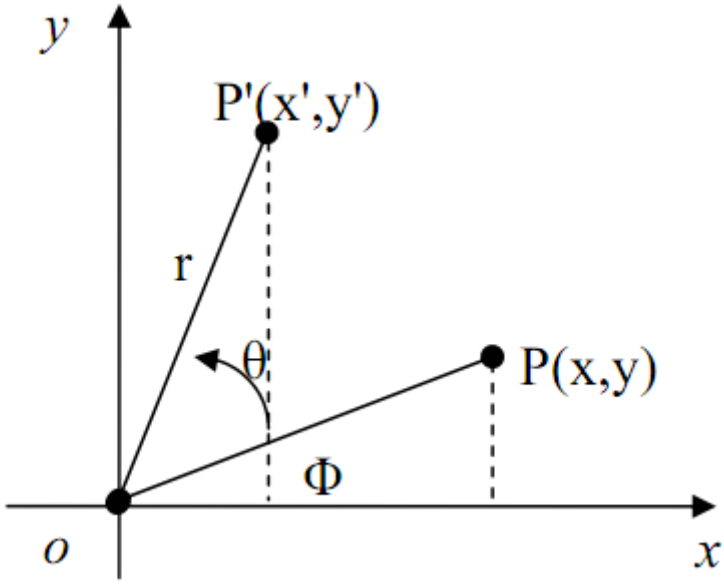
```
void initRendering()
{
    glEnable(GL_DEPTH_TEST);
}
void handleResize(int w, int h) {
    glViewport(0, 0, w, h);
    glColor3f(0.0, 1.0, 0.0);//Mau sac doi tuong
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0,(double)w / (double)h,1.0,200.0);
}
float _angle = 30.0f;
float _cameraAngle = 0.0f;
//hinh tron
void drawScene()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    drawBall();
    //drawBall2();
    glutSwapBuffers();
}
//float _angle = 30.0f;
void update(int value) {
    if(flag)
    {
    ballX += 0.001f;
       if(ballX>0.3)
       {
          flag=0;
       }
    }
    if (!flag)
    {
        ballX -= 0.001f;
        if(ballX<-0.3)
       {
          flag=1;
       }
    }
    glutPostRedisplay();
    glutTimerFunc(25, update, 0);
}

int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(600,400);
    glutCreateWindow("Hinh tron tinh tien");
    initRendering();
    glutDisplayFunc(drawScene);
    glutKeyboardFunc(keyPress);
    glutReshapeFunc(handleResize);
    glutTimerFunc(15, update, 0); //thoi gian xoay
    glutMainLoop();
    return(0);
}
```

Kết quả chương trình:



1.5. Phép xoay

Trong không gian hai chiều ta xét phép quay vật thể quanh tâm quay $I(x_r,y_r)$ với góc quay θ (θ > 0 nếu chiều quay ngược chiều kim đồng hồ và θ < 0 nếu chiều quay cùng chiều kim đồng hồ).



Ta có các phương trình biến đổi sau:

x'= r cos(Φ+θ)= r cosΦcosθ- r sinΦsinθ

y'= r sin(Φ+θ)= r cosΦsinθ+ r sinΦcosθ

Mặt khác ta lại có x=r cosΦ và y=r sinΦ. Thay vào hai phương trình trên ta có:

x'= x cosθ- y sinθ

y'= x sinθ+ y cosθ

Do đó phép quay được mô tả bởi phương trình:

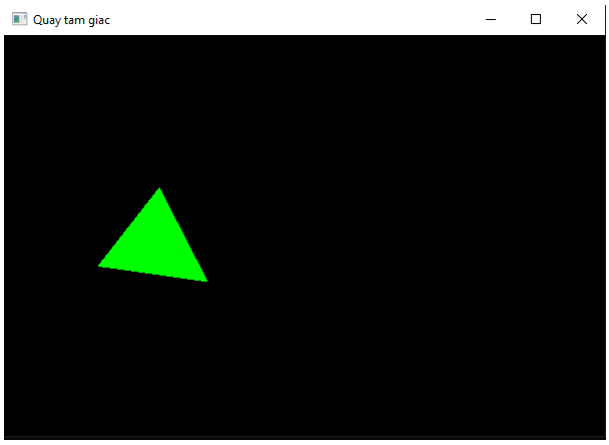
$$P'=\begin{bmatrix} x' \\ y' \end{bmatrix} = R.P= \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix}$$

Phép quay cũng giống như phép tịnh tiến không làm thay đổi hình dáng của vật thể.

Triển khai chương trình:

```
#include<iostream>
#include<stdlib.h>
#ifdef __APPLE__
#else
#include<GL/glut.h>
#endif
using namespace std;
void keyPress(unsigned char key,int x,int y)
{
    switch(key)
    {
       case 27:
           exit(0);
    }
}
void initRendering()
{
    glEnable(GL_DEPTH_TEST);
}
void handleResize(int w, int h) {
    glViewport(0, 0, w, h);
    glColor3f(0.0, 1.0, 0.0);//Mau sac doi tuong
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0,(double)w / (double)h,1.0,200.0);
}
float _angle = 30.0f;
float _cameraAngle = 0.0f;
//hinh tam giac
void drawScene()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glPushMatrix();
    glRotatef(_angle, -1.5f, 0.5f, -5.0f);//Xoay quanh truc z
    glBegin(GL_TRIANGLES);
        glVertex3f(-0.5f, 0.5f, -5.0f);
        glVertex3f(-1.0f, 1.5f, -5.0f);
        glVertex3f(-1.5f, 0.5f, -5.0f);
    glEnd();
    glPopMatrix();
    glutSwapBuffers();
}
void update(int value) {
    _angle += 2.0f;// toc do quay
    if (_angle > 360) {
        _angle -= 360;
    }
    glutPostRedisplay();
    glutTimerFunc(25, update, 0);
}
int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(600,400);
    glutCreateWindow("Quay tam giac");
    initRendering();
    glutDisplayFunc(drawScene);
    glutKeyboardFunc(keyPress);
    glutReshapeFunc(handleResize);
    glutTimerFunc(15, update, 0); //thoi gian xoay
    glutMainLoop();
    return(0);
}
```

Kết quả chương trình:



1.6. Phép tỉ lệ

Phép tỉ lệ sẽ nhân hoành độ và tung độ ban đầu với hệ số tỉ lệ s_x và s_y tương ứng, cụ thể ta sẽ có x'= x.s_x và y'= y.s_y. Phương trình của phép biến đổi tỉ lệ có thể được mô tả như sau:

$$P'=\begin{bmatrix} x' \\ y' \end{bmatrix} = S.P= \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} . \begin{bmatrix} x \\ y \end{bmatrix}$$

Khi (sx, sy)≠(1, 1) phép biến đổi tỉ lệ sẽ làm thay đổi hình dáng của vật thể

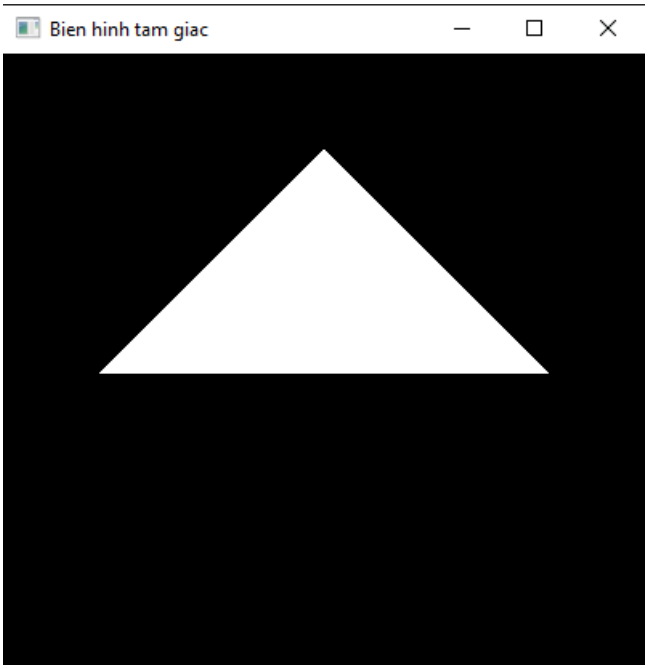
Triển khai chương trình:

```
#include<iostream>
#include<stdlib.h>
#ifdef __APPLE__
#include<openGL/openGL.h>
#include<GLUT/glut.h>
#else
#include<GL/glut.h>
#endif
using namespace std;
double rotate_by_key=0;
double rotate_x=0.5;
//Bien minh qua phim mui ten len/xuong
void keyPress(int key,int x,int y)
{
    if(key==27)
        exit(0);
    if (key == GLUT_KEY_UP)
        rotate_x += .05;
    if (key == GLUT_KEY_DOWN)
        rotate_x -= .05;
    glutPostRedisplay();
}
void initRendering()
{
    glEnable(GL_DEPTH_TEST);
}
//Goi cua so window
void handleResize(int w, int h) {
    glViewport(0, 0, w, h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();//Lam moi man hinh
```

```
    gluPerspective(45.0,(double)w / (double)h,1.0,200.0);
}
void drawScene()
{
    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glScalef( rotate_x,rotate_x,1.0f );
    glRotatef(  rotate_by_key,-1.0f, 1.5f, -5.0f );
    glBegin(GL_TRIANGLES);
        glVertex3f(1.0f, 0.0f, -5.0f);
        glVertex3f(0.0f, 1.0f, -5.0f);
        glVertex3f(-1.0f, 0.0f, -5.0f);
    glEnd();
    glutSwapBuffers();
}
int main(int argc,char** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(400,400);
    glutCreateWindow("Bien hinh tam giac");
    initRendering();
    glutDisplayFunc(drawScene);
    glutSpecialFunc(keyPress);
    glutReshapeFunc(handleResize);
    glutMainLoop();
    return(0);
}
```

Kết quả chương trình:

Ghi chú: Sử dụng phím mũi tên lên – xuống để điều chỉnh tỉ lệ hình, khi hình thu nhỏ hết mức sẽ đổi chiều hình ảnh (tương ứng là phép đối xứng)



2. Chạy các chương trình demo đồ họa bằng OpenGL

OpenGL (Thư viện đồ họa mở) là một API tiêu chuẩn đa nền tảng, tăng tốc phần cứng, không phụ thuộc vào ngôn ngữ để kết xuất đồ họa 3D (bao gồm cả 2D). Máy tính hiện đại có GPU chuyên dụng (Bộ xử lý đồ họa) với bộ nhớ riêng để tăng tốc độ kết xuất đồ họa. OpenGL là giao diện phần mềm với phần cứng đồ họa. Nói

cách khác, các lệnh kết xuất đồ họa OpenGL do các ứng dụng của bạn đưa ra có thể được chuyển hướng đến phần cứng đồ họa và được tăng tốc.

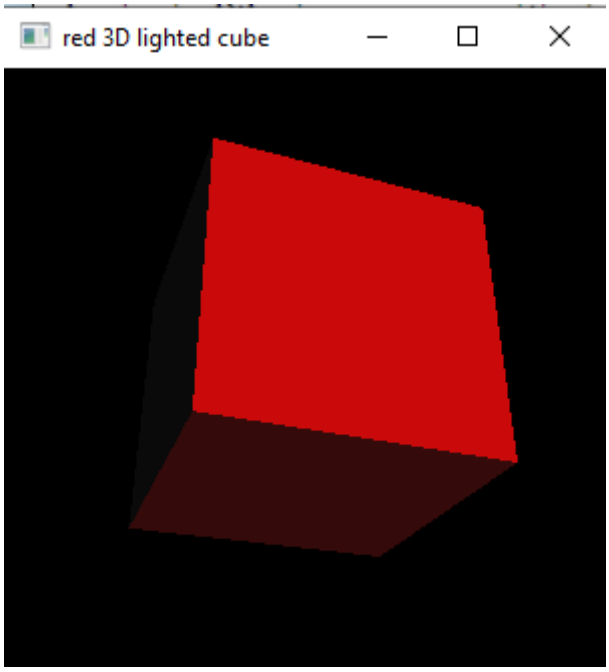
2.1. Vẽ hình học 3D với OpenGL

Triển khai chương trình:

```
#include <GL/glut.h>

GLfloat light_diffuse[] = {1.0, 0.0, 0.0, 1.0};
GLfloat light_position[] = {1.0, 1.0, 1.0, 0.0};
GLfloat n[6][3] = {
  {-1.0, 0.0, 0.0}, {0.0, 1.0, 0.0}, {1.0, 0.0, 0.0},
  {0.0, -1.0, 0.0}, {0.0, 0.0, 1.0}, {0.0, 0.0, -1.0} };
GLint faces[6][4] = {
  {0, 1, 2, 3}, {3, 2, 6, 7}, {7, 6, 5, 4},
  {4, 5, 1, 0}, {5, 6, 2, 1}, {7, 4, 0, 3} };
GLfloat v[8][3];

void
drawBox(void)
{
  int i;
  for (i = 0; i < 6; i++) {
    glBegin(GL_QUADS);
    glNormal3fv(&n[i][0]);
    glVertex3fv(&v[faces[i][0]][0]);
    glVertex3fv(&v[faces[i][1]][0]);
    glVertex3fv(&v[faces[i][2]][0]);
    glVertex3fv(&v[faces[i][3]][0]);
    glEnd();
  }
}

void
display(void)
{
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  drawBox();
  glutSwapBuffers();
}
void
init(void)
{
  v[0][0] = v[1][0] = v[2][0] = v[3][0] = -1;
  v[4][0] = v[5][0] = v[6][0] = v[7][0] = 1;
  v[0][1] = v[1][1] = v[4][1] = v[5][1] = -1;
  v[2][1] = v[3][1] = v[6][1] = v[7][1] = 1;
  v[0][2] = v[3][2] = v[4][2] = v[7][2] = 1;
  v[1][2] = v[2][2] = v[5][2] = v[6][2] = -1;
  glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diffuse);
  glLightfv(GL_LIGHT0, GL_POSITION, light_position);
  glEnable(GL_LIGHT0);
  glEnable(GL_LIGHTING);
  glEnable(GL_DEPTH_TEST);
  glMatrixMode(GL_PROJECTION);
  gluPerspective(40.0,1.0,1.0,10.0);
  glMatrixMode(GL_MODELVIEW);
  gluLookAt(0.0, 0.0, 5.0,0.0, 0.0, 0.0,0.0, 1.0, 0.);
  glTranslatef(0.0, 0.0, -1.0);
  glRotatef(60, 1.0, 0.0, 0.0);
  glRotatef(-20, 0.0, 0.0, 1.0);
}
```

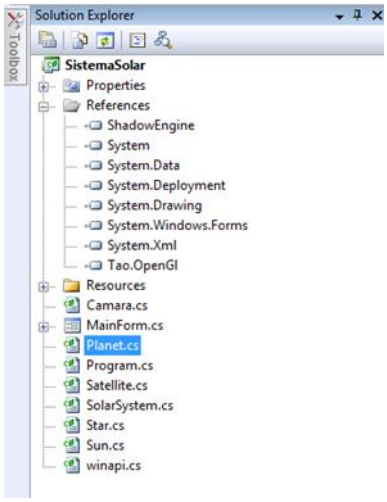
```
int
main(int argc, char **argv)
{
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutCreateWindow("Lap Phuong 3D");
  glutDisplayFunc(display);
  init();
  glutMainLoop();
  return 0;
}
```

Kết quả chương trình:



2.2. Mô phỏng hệ Mặt Trời với OpenGL

Sử dụng thư viện OpenGL kết hợp .Net Framework và ngôn ngữ C# mô phỏng hệ Mặt Trời (soure code tham khảo). Ứng dụng dạng Winform với các chức năng thu phóng, cùng những class C# sau:



Camara.cs

Đây là một máy ảnh dạng FPS (góc nhìn thứ nhất) cổ điển. Giải thích về cách hoạt động của FPS vượt ra ngoài phạm vi của bài viết này. Chúng hoạt động theo cách sau:

- Con trỏ chuột được ghim chính giữa màn hình.
- Khi người dùng di chuyển chuột, delta X và delta Y được tính từ điểm đầu.
- Delta X và delta Y được chuyển thành các góc và cách quay của máy ảnh.
- Khi người dùng muốn tiến hoặc lùi, máy ảnh sẽ di chuyển theo hướng là góc trỏ.
- Người đọc có thể xem qua public void Update(int pressedButton) tại lớp camera để hiểu rõ hơn.

MainForm.cs

Lớp này là code chính và duy nhất của dự án. Nó chứa lệnh gọi tải kết cấu, khởi tạo bối cảnh 3D, bản vẽ nội dung 3D, v.v. Nó cũng xử lý phím nhập vào từ người dùng và đầu vào của chuột. Vì nội dung 3D yêu cầu vẽ ít nhất 30 khung hình / giây nên tác giả đã sử dụng bộ đếm thời gian và đặt tất cả câu lệnh của bản vẽ bên trong lệnh đó.

Planet.cs

Một hành tinh chứa các biến sau:

- Toạ độ
- Kết cấu (lưu dạng ảnh trong \bin\Debug\texturas\)
- Quỹ đạo quay (khoảng cách hiện tại tính từ mặt trời)
- Góc quay hiện tại
- Tốc độ di chuyển trên quỹ đạo hiện tại

Tác giả đã sử dụng OpenGL mặt cong bậc 2 để vẽ hình cầu mô phỏng hành tinh. Quadrics(mặt cong bậc 2) là các hình dạng được xác định trước OpenGL để trợ giúp trong các tác vụ vẽ nhỏ. Trong mỗi khung, hành tinh chuyển động trên quỹ đạo của nó theo tốc độ di chuyển. Ngoài ra, có một biến bool được gọi là hasMoon để chỉ định xem người dùng có muốn xem bản vẽ mặt trăng cho hành tinh đó hay không. Tác giả chỉ có mặt trăng của Trái Đất nhưng nếu người dùng thích, ví dụ, để vẽ mặt trăng sao Hỏa là Phobos và Deimos, họ có thể sử dụng mã đó. Một chức năng thú vị khác có chứa trong lớp hành tinh là chức năng được sử dụng để vẽ quỹ đạo của nó. Đầu tiên, tác giả tạo các điểm bằng hàm sin và sau đó kết nối chúng bằng GL_LINE_STRIP. Đây là code:

```
public void DrawOrbit()
{
    Gl.glBegin(Gl.GL_LINE_STRIP);

    for (int i = 0; i < 361; i++)
    {
        Gl.glVertex3f(p.x * (float)Math.Sin(i * Math.PI / 180),
         0, p.x * (float)Math.Cos(i * Math.PI / 180));
    }
    Gl.glEnd();
 }
```

Satellite.cs

Lớp vệ tinh chứa mọi biến giống với lớp khai báo hành tinh. Điểm khác biệt duy nhất là tâm quay của nó không phải là mặt trời mà là hành tinh chủ của nó.

SolarSystem.cs

Đây là lớp chứa danh sách các hành tinh, các ngôi sao và vệ tinh. Lớp này để khai báo và vẽ các đối tượng đồ hoạ đó.

Star.cs

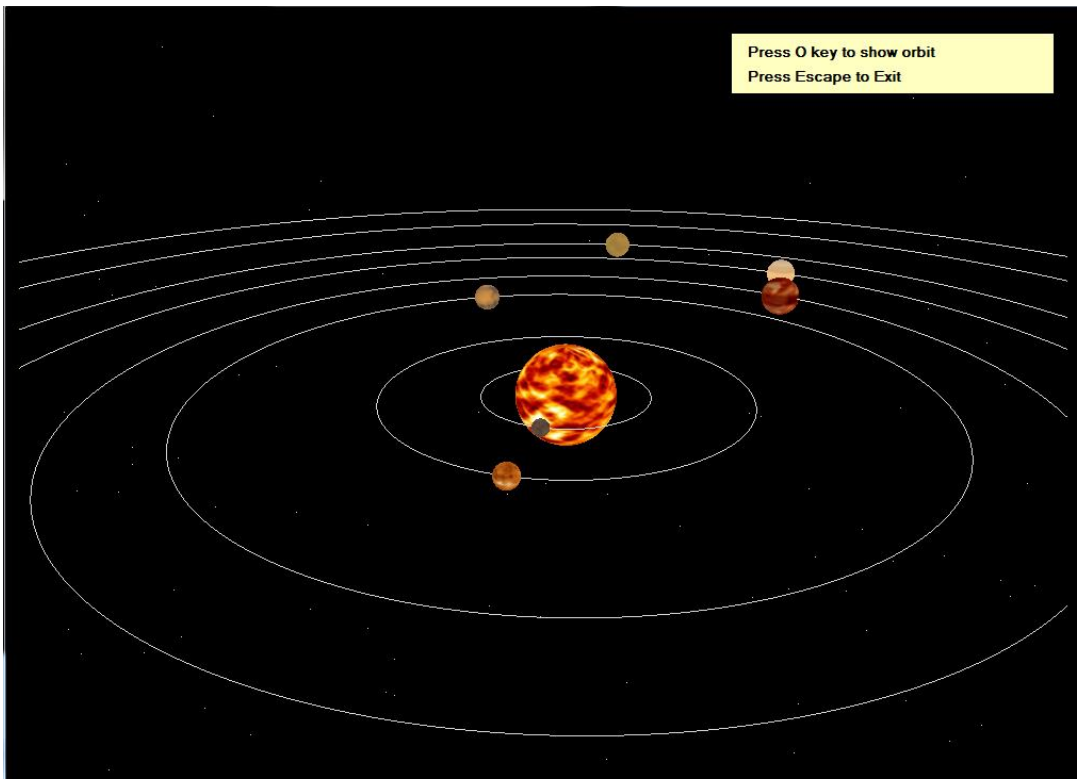
Đây là lớp vẽ các ngôi sao. Các ngôi sao vẽ bằng GL_POINTS đơn được vẽ ở các vị trí ngẫu nhiên (vùng ngoài). Đây là mã lệnh để vẽ chúng:

```
public void CreateStars(int amount)
{
    Random r = new Random();
    int count = 0;

    while (count != amount)
    {
       Position p = default(Position);
       p.x = (r.Next(110)) * (float)Math.Pow(-1, r.Next());
       p.z = (r.Next(110)) * (float)Math.Pow(-1, r.Next());
       p.y = (r.Next(110)) * (float)Math.Pow(-1, r.Next());
       if (Math.Pow(Math.Pow(p.x, 2) + Math.Pow(p.y, 2) +
                             Math.Pow(p.z, 2), 1 / 3f) > 15)
         {
             stars.Add(p);
             count++;
         }
      }
 }
```

Sun.cs

Lớp mặt trời là lớp đơn giản nhất, nó giống như lớp hành tinh chỉ là nó không có quỹ đạo. Nó chỉ có một chuyển động quay quanh trục của nó.

Thông tin các lớp được dịch sang Tiếng Việt bởi Nguyễn Văn Hải Long. Blog cá nhân của tác giả có kèm nhiều chương trình 3D: http://vasilydev.blogspot.com.
Kết quả chương trình(Sử dụng Visual Studio):



2.3 Đọc file OBJ sử dụng OpenGL

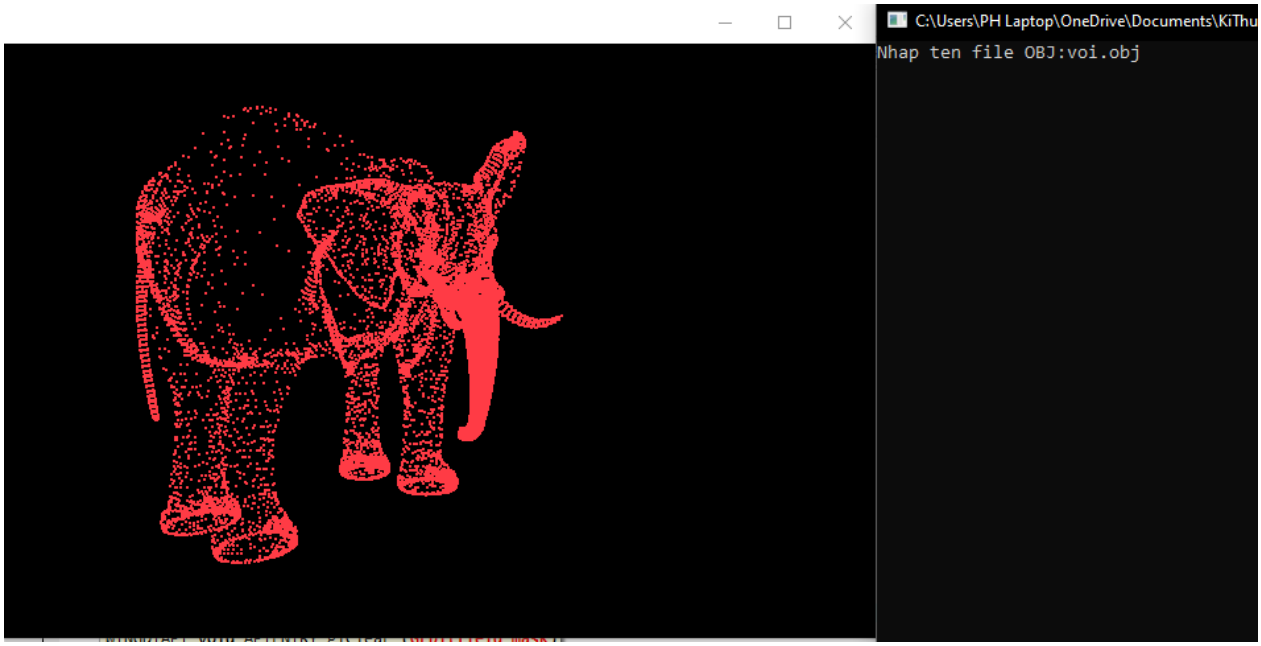
Tệp OBJ là một định dạng tệp mô hình 3D. Một tiêu chuẩn được phát triển bởi Alias Wavefront cho phần mềm hoạt hình và mô hình 3D "Advanced Visualizer", phù hợp để tương tác lẫn nhau giữa các mô hình và phần mềm 3D, nó cũng có thể được đọc và lập trình thông qua Maya.

Triển khai chương trình:

```
#include<GL/gl.h>
#include<GL/glut.h>
#include<stdio.h>
GLuint elephant;
float elephantrot;
char ch='1';
void loadObj()
{
    char fname[65],tenfile[24],duongdan[]="data/";
    st:printf("Nhap ten file OBJ:");scanf("%s",tenfile);
    strcpy(fname,duongdan);// Sao chep chuoi duongdan vao fname
    strcat(fname,tenfile);// Noi chuoi dang vao tenfile vua sao chep
    FILE *fp;
    int read;
    GLfloat x, y, z;
    char ch;
    elephant=glGenLists(1);
    fp=fopen(fname,"r");
    if (!fp)
    {
        printf("Khong the mo file: %s\n", fname);
        goto st;
    }
    glPointSize(2.0);
    glNewList(elephant, GL_COMPILE);
    {
        glPushMatrix();
        glBegin(GL_POINTS);
```

```
        while(!(feof(fp)))
        {
            read=fscanf(fp,"%c %f %f %f",&ch,&x,&y,&z);
            if(read==4&&ch=='v')
            {
                glVertex3f(x,y,z);
            }
        }
        glEnd();
    }
    glPopMatrix();
    glEndList();
    fclose(fp);
}
void reshape(int w,int h)
{
    glViewport(0,0,w,h);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective (60, (GLfloat)w / (GLfloat)h, 0.1, 1000.0);
    glMatrixMode(GL_MODELVIEW);
}
void drawElephant()
{
    glPushMatrix();
    glTranslatef(0,-30.00,-100.00);//Toa do chieu obj
    glColor3f(1.0,0.23,0.27);
    glScalef(0.1,0.1,0.1);
    glRotatef(elephantrot,0,1,0);
    glCallList(elephant);
    glPopMatrix();
    elephantrot=elephantrot+0.03;// Toc do quay vat the
    if(elephantrot>360)elephantrot=elephantrot-360;
}
void display(void)
{
    glClearColor (0.0,0.0,0.0,1.0);
    glClear (GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    drawElephant();
    glutSwapBuffers(); //swap the buffers
}
int main(int argc,char **argv)
{

    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutInitWindowSize(800,450);
    glutInitWindowPosition(20,20);
    glutCreateWindow("Obj Loader");
    glutReshapeFunc(reshape);
    glutDisplayFunc(display);
    glutIdleFunc(display);
    loadObj();
```

Kết quả chương trình:



3. Đường cong Bezier và mặt cong Bezier

3.1. Đường cong Bezier

Đường cong Bezier là một đường cong tham số thường được sử dụng trong đồ họa máy tính và một số lĩnh vực khác. Dạng tổng quát hóa của đường cong Bezier trong không gian nhiều chiều được gọi là mặt phẳng Bezier, trong đó tam giác Bezier là một trường hợp đặc biệt.

Đường cong Bezier được công bố lần đầu vào năm 1962 bởi một kỹ sư người Pháp Pierre Bezier, người sử dụng nó để thiết kế thân ôtô. Nhưng việc nghiên cứu những đường cong này thực tế đã bắt đầu từ năm 1959 bởi nhà toán học Paul de Casteljau, ông sử dụng giải thuật De Casteljau để đánh giá các đường cong đó.

Về mặt ứng dụng, đường cong Bezier thường được sử dụng trong đồ họa vector để mô hình hóa các đường cong mượt (smooth curves) và những đường cong đó có thể được phóng to hoặc thu nhỏ theo một tỉ lệ không giới hạn. "Đường dẫn" (path), một khái niệm được sử dụng trong các chương trình xử lý ảnh, được tạo ra bằng cách liên kết các đường cong Bezier với nhau. Đường cong Bezier còn thường được sử dụng như là một công cụ để điều khiển sự chuyển động (animation).

Mô tả thuật toán vẽ đường cong Bezier lập phương hay bậc ba (cubic). Với 4 điểm P_0, P_1, P_2 và P_3 trên mặt phẳng hoặc trong không gian nhiều chiều có thể định nghĩa một đường cong Bezier bậc 3. Đường cong này bắt đầu từ điểm P_0, đi theo hướng của điểm P_1 và P_2 trước khi kết thúc tại p_3. Đường cong được hình thành thường không trực tiếp đi qua điểm P_1 và P_2 và 2 điểm này chỉ mang tính định hướng cho đường cong.

Phương trình đường cong Bezier bậc 3 có thể được định nghĩa bằng cách kết hợp 2 đường cong Bezier bậc 2 với nhau, với phương trình cho đường cong Bezier bậc 2 là $B_{Pi,Pj,Pk}$, trong đó P_i, P_j, P_k là các điểm của đường cong đó:

$$\mathbf{B}(t) = (1-t)\mathbf{B}_{\mathbf{P}_0,\mathbf{P}_1,\mathbf{P}_2}(t) + t\mathbf{B}_{\mathbf{P}_1,\mathbf{P}_2,\mathbf{P}_3}(t),\ t \in [0,1].$$

Dạng tường minh cho phương trình này là:

$$\mathbf{B}(t) = (1-t)^3\mathbf{P}_0 + 3(1-t)^2 t\mathbf{P}_1 + 3(1-t)t^2\mathbf{P}_2 + t^3\mathbf{P}_3,\ t \in [0,1].$$

Triển khai chương trình vẽ Bezier bậc 3:

```
#include<graphics.h>
#include<math.h>
#include<conio.h>
#include<stdio.h>
int main()
{
//Khai bao toa do x,y duoi dang mang
int x[4]={100,300,300,150},y[4]={200,250,300,400};
double put_x,put_y,t;
int gr=DETECT,gm;
initgraph(&gr,&gm,"C:\\TURBOC3\\BGI");
printf("\n****** Duong cong Bezier bac 3 **********");
for(int i=0;i<4;i++)
{
putpixel(x[i],y[i],BLUE);
setcolor(BLUE);//Chon mau ve khung
    if (i<3)
        line(x[i],y[i],x[i+1],y[i+1]);//Ve khung
}

for(t=0.0;t<=1.0;t=t+0.001)
{
put_x = pow(1-t,3)*x[0] + 3*t*pow(1-t,2)*x[1] + 3*t*t*(1-t)*x[2] + pow(t,3)*x[3];
put_y =  pow(1-t,3)*y[0] + 3*t*pow(1-t,2)*y[1] + 3*t*t*(1-t)*y[2] + pow(t,3)*y[3];
putpixel(put_x,put_y, RED);
}
getch();
closegraph();
}
```

Kết quả chương trình:

3.2. Mặt cong Bezier

Một bề mặt Bezier cho trước có bậc (n , m) được xác định bởi tập hợp (n + 1) (m + 1) điểm điều khiển $k_{i,j}$ trong đó i = 0, ..., n và j = 0, .. ., m . Nó ánh xạ theo đơn vị vuông (là một hình vuông có độ dài các cạnh là 1) thành một bề mặt liên tục được đặt trong không gian chứa $k_{i,j}$.

Ví dụ, nếu $k_{i,j}$ là tất cả các điểm trong không gian bốn chiều, thì bề mặt đó sẽ nằm trong 1 không gian bốn chiều.

Một bề mặt Bezier hai chiều có thể được định nghĩa là một bề mặt tham số trong đó vị trí của một điểm p là một hàm của các tọa độ tham số u , v được cho bởi:

$$\mathbf{p}(u,v) = \sum_{i=0}^{n} \sum_{j=0}^{m} B_i^n(u)\, B_j^m(v)\, \mathbf{k}_{i,j}$$

được xét trên bình phương đơn vị, trong đó

$$B_i^n(u) = \binom{n}{i} u^i (1-u)^{n-i}$$

là một đa thức Bernstein cơ sở , và

$$\binom{n}{i} = \frac{n!}{i!(n-i)!}$$

là một hệ số nhị thức.

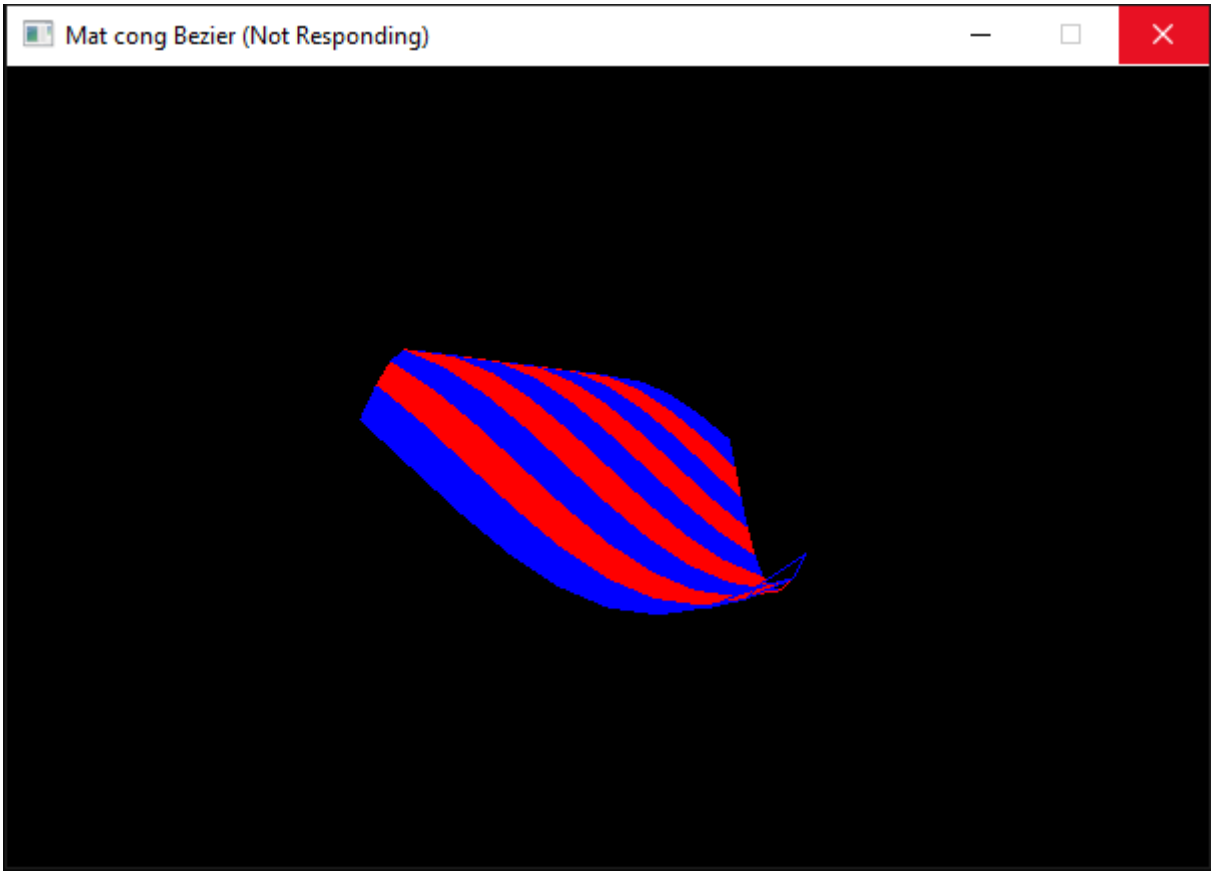
Triển khai chương trình:

```c
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <stdarg.h>
#if defined (__APPLE__) || defined (MACOSX)
#include <GLUT/glut.h>
#else
#include <GL/glut.h>
#endif
#ifdef _WIN32
#define M_PI 3.1415926
#endif

#define MAX_MESH 10
GLfloat angleCube = 0.0f;
GLfloat mesh[MAX_MESH][MAX_MESH];

GLfloat old_x, old_y, spin_x = 0, spin_y = 0;

void
reshape(int width, int height)
{
    glViewport(0, 0, width, height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(60, (GLfloat)width/height, 0.1, 1000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    gluLookAt(0, 0, 3, 0, 0, 0, 0, 1, 0);
    glShadeModel(GL_FLAT);
    glEnable(GL_DEPTH_TEST);
    glPolygonMode(GL_BACK, GL_LINE);
}

void
display(void)
{
    int i, k, swap = 0;

    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    glTranslatef(-MAX_MESH/2, 0, -MAX_MESH*2);
    glRotatef(spin_x, 0, 1, 0);
    glRotatef(spin_y, 1, 0, 0);
    glColor3ub(120, 255, 255);
    glBegin(GL_TRIANGLE_STRIP);
    for (k = 0; k < MAX_MESH-1; k++) {
        if (swap) {
            for (i = MAX_MESH-1; i >= 0; i--) {
                glColor3ub(255, 0, 0);
                glVertex3f(i, mesh[k][i], k);
                glVertex3f(i, mesh[k+1][i], k+1);
                if (i == 0)
                    glVertex3f(i, mesh[k+1][i], k+1);
            }
        } else {
            for (i = 0; i < MAX_MESH; i++) {
                glColor3ub(0, 0, 255);//
```

```
                glVertex3f(i, mesh[k][i], k);
                glVertex3f(i, mesh[k+1][i], k+1);
                if (i == MAX_MESH-1)
                    glVertex3f(i, mesh[k+1][i], k+1);
            }
        }
        swap ^= 1;
    }
    glEnd();

    glPopMatrix();

    glutSwapBuffers();
}

void keyboard(unsigned char key, int x, int y)
{
    static int wireframe = 0;

    if (key == 27) {
        exit(0);
    } else if (key == 'w') {
        wireframe ^= 1;
        if (wireframe)
            glPolygonMode(GL_FRONT, GL_LINE);
        else
            glPolygonMode(GL_FRONT, GL_FILL);
    } else if (key == 'c') {
        if (glIsEnabled(GL_CULL_FACE))
            glDisable(GL_CULL_FACE);
        else
            glEnable(GL_CULL_FACE);
    }

    glutPostRedisplay();
}

void mouse(int button, int state, int x, int y)
{
    old_x = x;
    old_y = y;

    glutPostRedisplay();
}

void motion(int x, int y)
{
    spin_x = x - old_x;
    spin_y = y - old_y;

    glutPostRedisplay();
}

void timer(int value) {
   glutPostRedisplay();
   glutTimerFunc(refreshMills, timer, 0);
}
void reshape(GLsizei width, GLsizei height) {
   if (height == 0) height = 1;
   GLfloat aspect = (GLfloat)width / (GLfloat)height
   glViewport(0, 0, width, height);
```

```
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0f, aspect, 0.1f, 100.0f);
}
int
main(int argc, char** argv)
{
    int i, k;
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(600, 400);
    glutInit(&argc, argv);

    for (k = 0; k < MAX_MESH; k++) {
        for (i = 0; i < MAX_MESH; i++) {
            mesh[k][i] = sin((float)(i+k)/MAX_MESH*M_PI)*3;
        }
    }
    glutCreateWindow("Mat cong Bezier");
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    glutMotionFunc(motion);
    glutMouseFunc(mouse);
    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    initGL();
    glutTimerFunc(0, timer, 0);
    glutMainLoop();
    return 0;
}
```

Kết quả chương trình:



4. Demo đồ hoạ với Java

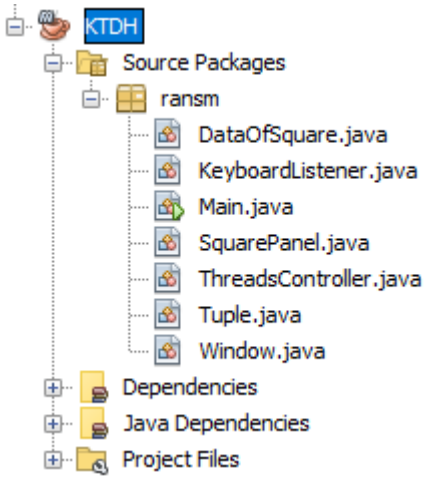
Demo những chương trình đồ hoạ đơn giản trên kênh Youtube:

https:/www.youtube.com/channel/UC4SVo0Ue36XCfOyb5Lh1viQ

Chạy code trên môi trường Apache NetBeans IDE:

Tên chương trình: Rắn săn mồi

Cấu trúc:



DataOfSquare.java

Khai báo màu sắc giao diện

```
/**
 *
 * @author Nguyễn Văn Hải Long
 */
import java.util.ArrayList;
import java.awt.Color;
public class DataOfSquare {
        ArrayList<Color> C =new ArrayList<>();
        int color;
        SquarePanel square;
        public DataOfSquare(int col){
                C.add(Color.darkGray);//0
                C.add(Color.BLUE);    //1
                C.add(Color.white);   //2
                color=col;
                square = new SquarePanel(C.get(color));
        }
        public void lightMeUp(int c){
                square.ChangeColor(C.get(c));
        }
}
```

KeyboardListener.java

Gán phím sự kiện:

```
*
* @author Nguyễn Văn Hải Long
*/
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;

public class KeyboardListener extends KeyAdapter{
        public void keyPressed(KeyEvent e){
            switch(e.getKeyCode()){
                case 39:        // -> Right
                                //if it's not the opposite direction
                                if(ThreadsController.directionSnake!=2)
                                        ThreadsController.directionSnake=1;
                                break;
                case 38:        // -> Top
                                        if(ThreadsController.directionSnake!=4)
                                                ThreadsController.directionSnake=3;
                                break;
                case 37:        // -> Left
                                        if(ThreadsController.directionSnake!=1)
                                                ThreadsController.directionSnake=2;
                                break;
                case 40:        // -> Bottom
                                        if(ThreadsController.directionSnake!=3)
                                                ThreadsController.directionSnake=4;
                                break;
                default:        break;
            }
        }

}
```

SquarePanel.java

```
package ransm;

/**
 *
 * @author Nguyễn Văn Hải Long
 */
import java.awt.Color;
import javax.swing.JPanel;

public class SquarePanel extends JPanel{

        private static final long serialVersionUID = 1L;

        public SquarePanel(Color d){
                this.setBackground(d);
        }

        public void ChangeColor(Color d){
                this.setBackground(d);
                this.repaint();
        }

}
```

Tuple.java

```
 * @author Nguyễn Văn Hải Long
 */
public class Tuple {
        public  int x;
        public  int y;
        public int xf;
        public int yf;

        public Tuple(int x, int y) {
          this.x = x;
          this.y = y;
        }
        public void ChangeData(int x, int y){
                this.x = x;
                this.y = y;
        }
        public int getX(){
                return x;
        }
        public int getY(){
                return y;
        }
        public int getXf(){
                return xf;
        }
        public int getYf(){
                return yf;
        }


}
```

Kết quả chương trình: