Practical File - CG

Name: Dhruv Dhingra

Roll no: 20078570026

Semester: VI

Subject: Computer Graphics

Course: B.Sc Hons Computer Science

Q1 :- Write a program to implement Bresenham’s line drawing algorithm

```
#include <iostream>
#include <graphics.h>
using namespace std;

// Function to draw a line using Bresenham's line drawing algorithm
void drawLine(int x1, int y1, int x2, int y2)
{
    int dx = abs(x2 - x1);
    int dy = abs(y2 - y1);
    int x, y, p;
    int incx = 1, incy = 1;
    if (x2 < x1)
        incx = -1;
    if (y2 < y1)
        incy = -1;
    x = x1;
    y = y1;
    if (dx > dy)
    {
        p = 2 * dy - dx;
        while (x != x2)
        {
            putpixel(x, y, WHITE);
            x += incx;
            if (p < 0)
```

```
                p += 2 * dy;
            else
            {
                p += 2 * (dy - dx);
                y += incy;
            }
        }
    }
    else
    {
        p = 2 * dx - dy;
        while (y != y2)
        {
            putpixel(x, y, WHITE);
            y += incy;
            if (p < 0)
                p += 2 * dx;
            else
            {
                p += 2 * (dx - dy);
                x += incx;
            }
        }
    }
    putpixel(x, y, WHITE); // draw the last pixel of the line
}

int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, ""); // initialize graphics mode
    int x1 = 100, y1 = 100, x2 = 300, y2 = 200;
    drawLine(x1, y1, x2, y2); // call function to draw line
    getch(); // wait for user input
    closegraph(); // close graphics mode
    return 0;
}
```

Q2 :- Write a program to implement mid-point circle drawing algorithm

```
#include <iostream>
#include <graphics.h>
using namespace std;

// Function to draw a circle using Mid-Point Circle Drawing Algorithm
void drawCircle(int x0, int y0, int radius)
{
    int x = radius;
    int y = 0;
    int decisionParam = 1 - radius;

    while (x >= y)
    {
        putpixel(x0 + x, y0 + y, WHITE);
        putpixel(x0 + y, y0 + x, WHITE);
        putpixel(x0 - y, y0 + x, WHITE);
        putpixel(x0 - x, y0 + y, WHITE);
        putpixel(x0 - x, y0 - y, WHITE);
        putpixel(x0 - y, y0 - x, WHITE);
        putpixel(x0 + y, y0 - x, WHITE);
        putpixel(x0 + x, y0 - y, WHITE);

        y++;

        if (decisionParam <= 0)
        {
            decisionParam += 2 * y + 1;
        }
        else
        {
            x--;
            decisionParam += 2 * (y - x) + 1;
        }
    }
}

int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, ""); // Initialize graphics mode
    int x0 = 320, y0 = 240, radius = 100;
    drawCircle(x0, y0, radius); // Call function to draw circle
    getch(); // Wait for user input
    closegraph(); // Close graphics mode
    return 0;
}
```

Q3 :- Write a program to clip a line using Cohen and Sutherland line clipping algorithm

```
#include <iostream>
#include <graphics.h>
using namespace std;

// Define the region codes for the endpoints of the line
const int INSIDE = 0; // 0000
const int LEFT = 1;   // 0001
const int RIGHT = 2;  // 0010
const int BOTTOM = 4; // 0100
const int TOP = 8;    // 1000

// Define the minimum and maximum coordinates of the clipping window
const int xmin = 100;
const int ymin = 100;
const int xmax = 500;
const int ymax = 400;

// Function to get the region code of a point
int getRegionCode(int x, int y)
{
    int code = INSIDE;
    if (x < xmin)
        code |= LEFT;
```

```
    else if (x > xmax)
        code |= RIGHT;
    if (y < ymin)
        code |= BOTTOM;
    else if (y > ymax)
        code |= TOP;
    return code;
}

// Function to clip a line using the Cohen-Sutherland algorithm
void clipLine(int x1, int y1, int x2, int y2)
{
    int code1 = getRegionCode(x1, y1);
    int code2 = getRegionCode(x2, y2);
    bool accept = false;

    while (true)
    {
        if ((code1 == 0) && (code2 == 0)) // both endpoints are inside the clipping window
        {
            accept = true;
            break;
        }
        else if (code1 & code2) // both endpoints are outside the clipping window on the same side
        {
            break;
        }
        else // at least one endpoint is outside the clipping window
        {
            int x, y;
            int outcode = (code1 != 0) ? code1 : code2;
            if (outcode & TOP)
            {
                x = x1 + (x2 - x1) * (ymax - y1) / (y2 - y1);
                y = ymax;
            }
            else if (outcode & BOTTOM)
            {
                x = x1 + (x2 - x1) * (ymin - y1) / (y2 - y1);
                y = ymin;
            }
            else if (outcode & RIGHT)
            {
                y = y1 + (y2 - y1) * (xmax - x1) / (x2 - x1);
                x = xmax;
            }
            else if (outcode & LEFT)
            {
                y = y1 + (y2 - y1) * (xmin - x1) / (x2 - x1);
                x = xmin;
            }
            if (outcode == code1)
            {
                x1 = x;
                y1 = y;
                code1 = getRegionCode(x1, y1);
            }
```

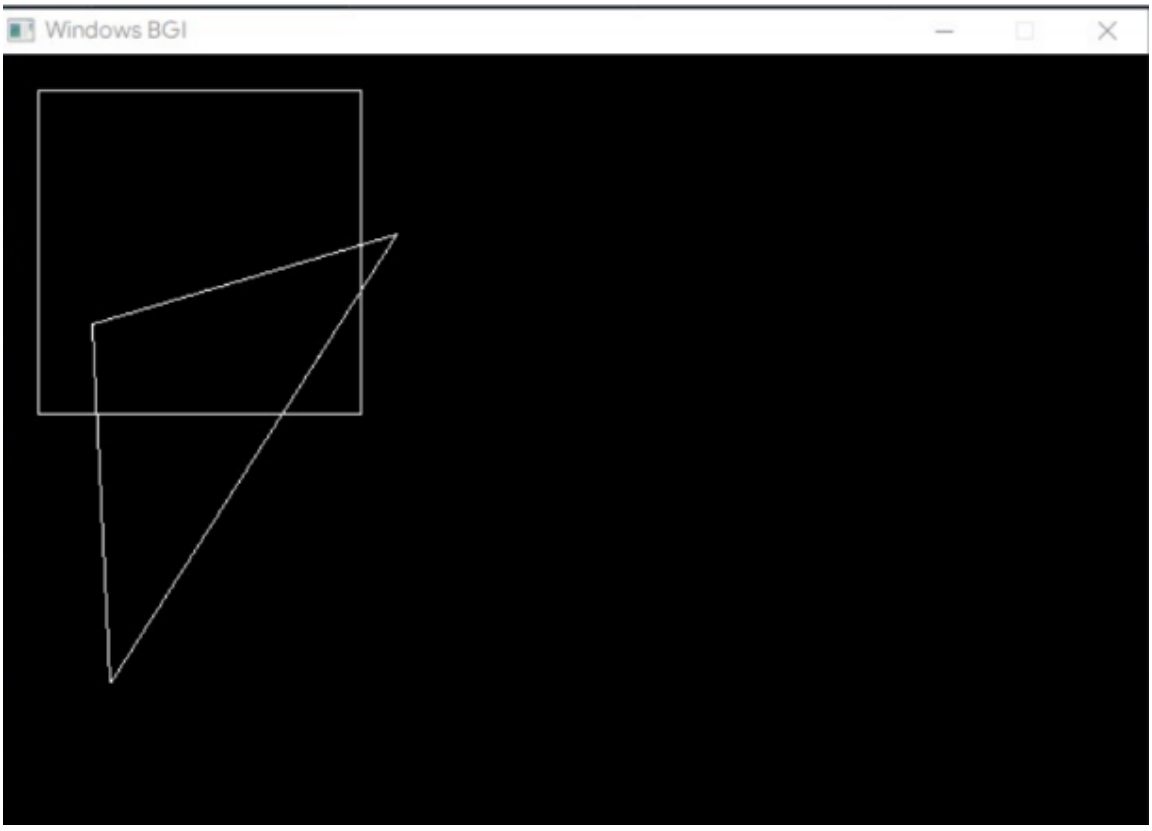
```
            else
            {
                x2 = x;
                y2 = y;
                code2 = getRegionCode(x2, y2);
            }
        }
    }

    if (accept)
    {
        line(x1, y1, x2, y2);
    }
}

int main()
{
    int gd = DETECT, gm;
    initgraph(&gd, &gm, ""); // initialize graphics mode

    // draw the clipping window
    rectangle(xmin, ymin, xmax, ymax);

    // draw the line to be clipped
    int x1 = 50, y1 = 200, x
```

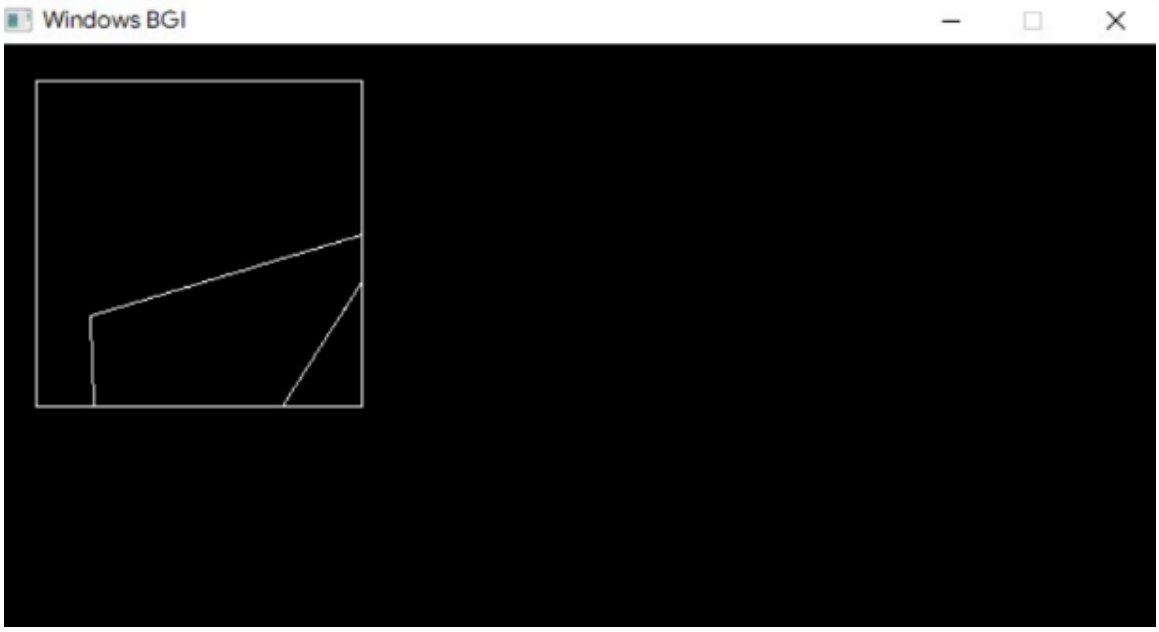
Q4 :- Write a program to clip a polygon using Sutherland Hodgeman algorithm.

```
#include <conio.h>
#include <graphics.h>
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

typedef unsigned int outcode;

outcode compOutcode(double x, double y);
enum
{
TOP = 0x1,
BOTTOM = 0x2,
RIGHT = 0x4,
LEFT = 0x8
};

double xmin, xmax, ymin, ymax;

outcode compOutcode(double x, double y)
{
outcode code = 0;
if (y > ymax)
code |= TOP;
else if (y < ymin)
code |= BOTTOM;
if (x > xmax)
code |= RIGHT;
else if (x < xmin)
code |= LEFT;
return code;
}

void clipPolygon(int x0, int y0, int x1, int y1)

{
int accept = 0, done = 0;
outcode outcode0, outcode1, outcodeOut;

outcode0 = compOutcode(x0, y0);
outcode1 = compOutcode(x1, y1);

do
{
if (!(outcode0 | outcode1))
{
accept = 1;
done = 1;
}
else if (outcode0 & outcode1)
done = 1;
else
{
```

```
double x, y;
outcodeOut = outcode0 ? outcode0 : outcode1;
if (outcodeOut & TOP)
{
x = x0 + (x1 - x0) * (ymax - y0) / (y1 - y0);
y = ymax;
}
else if (outcodeOut & BOTTOM)

{
x = x0 + (x1 - x0) * (ymin - y0) / (y1 - y0);
y = ymin;
}
else if (outcodeOut & RIGHT)
{
y = y0 + (y1 - y0) * (xmax - x0) / (x1 - x0);
x = xmax;
}
else
{
y = y0 + (y1 - y0) * (xmin - x0) / (x1 - x0);
x = xmin;
}

if (outcodeOut == outcode0)
{
x0 = x;
y0 = y;
outcode0 = compOutcode(x0, y0);
}
else
{
x1 = x;
y1 = y;
outcode1 = compOutcode(x1, y1);
}
}
} while (done == 0);

if (accept)
line(x0, y0, x1, y1);
}

int main()
{
int i, n;
int gd = DETECT, gm;
int poly[24];

initgraph(&gd, &gm, (char*)"");

cout << "Enter Bounds of Clipping Rectangle: ";
cout << "\n\txmin: ";
cin >> xmin;
cout << "\tymin: ";
cin >> ymin;
cout << "\txmax: ";
```

```
cin >> xmax;
cout << "\tymax: ";
cin >> ymax;

cout << "Enter Number of Edges in Polygon: ";
cin >> n;

cout << "Enter Coordinates of the Polygon: ";
for (i = 0; i < 2 * n; i++)
cin >> poly[i];

poly[2 * n] = poly[0];
poly[2 * n + 1] = poly[1];

rectangle(xmin, ymin, xmax, ymax);

for (i = 0; i < n; i++)
clipPolygon(poly[2 * i], poly[(2 * i) + 1], poly[(2 * i) + 2], poly[(2 *
i) + 3]);

getch();
closegraph();

return 0;
}
```





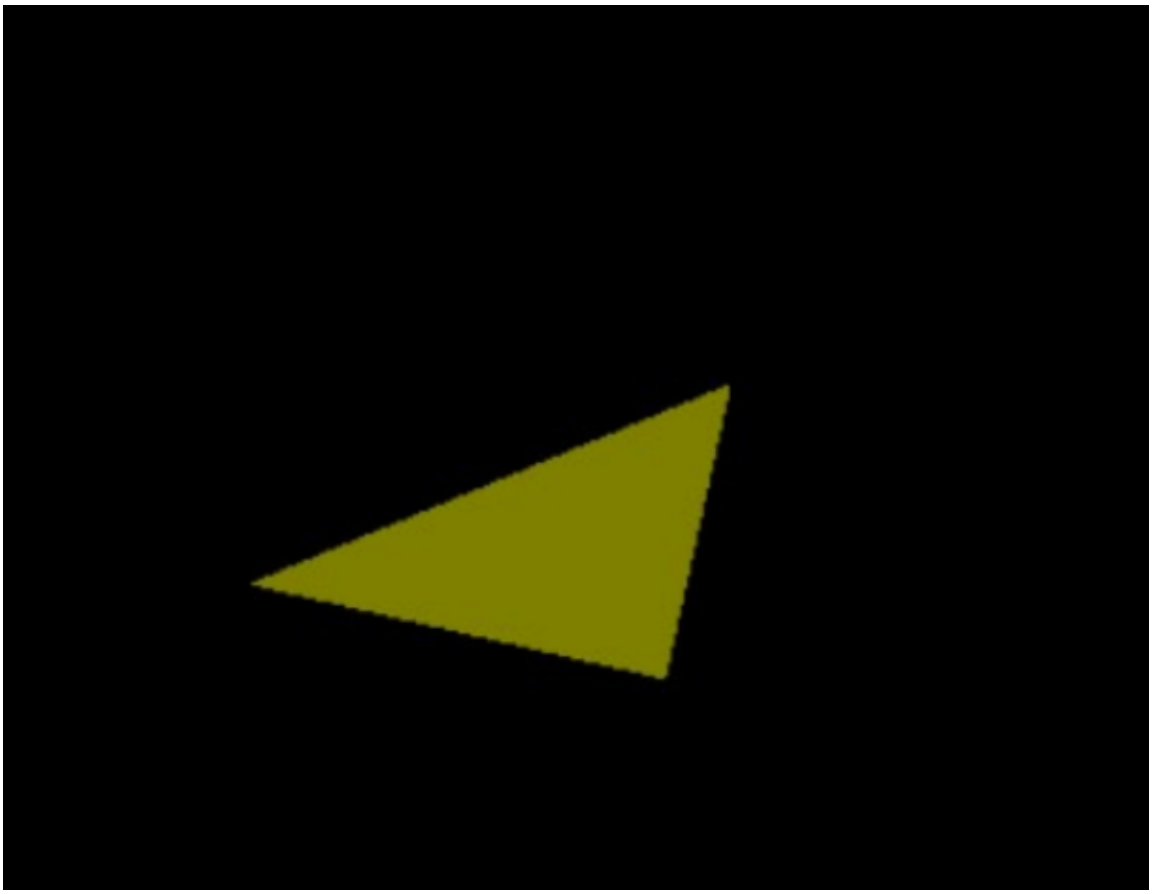
Q5 :- Write a Program to fill a Polygon using Scan Fill Algorithm.

```
#include <conio.h>
#include <iostream>
#include <graphics.h>
#include <stdlib.h>
using namespace std;

//Declaration of class point
class point
{
public:
int x,y;
};

class poly
{
private:
point p[20];
int inter[20],x,y;
int v,xmin,ymin,xmax,ymax;
public:
int c;
void read();
void calcs();
void display();
void ints(float);
void sort(int);
};
```

```
void poly::read()
{
int i;
cout<<"\n\t SCAN_FILL ALGORITHM";
cout<<"\n Enter the no of vertices of polygon:";
cin>>v;
if(v>2)
{
for(i=0;i<v; i++) //ACCEPT THE VERTICES
{
cout<<"\nEnter the co-ordinate no.- "<<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;

xmin=xmax=p[0].x;
ymin=ymax=p[0].y;

}
else
cout<<"\n Enter valid no. of vertices.";
}

void poly::calcs()
{ //MAX,MIN
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;
  }
}

//DISPLAY FUNCTION
void poly::display() {
int ch1;
char ch='y';
float s,s2;
do
{
cout<<"\n\nMENU:";
cout<<"\n\n\t1 . Scan line Fill ";
cout<<"\n\n\t2 . Exit ";
cout<<"\n\nEnter your choice:";
```

```
cin>>ch1;
switch(ch1)
{
case1:s=ymin+0.01;
delay
(100);
cleardevice();
while(s<=ymax)
{
ints(s);
sort(s);
s++;
}
break;

case 2:
exit(0);

}

cout<<"Do you want to continue?: ";
cin>>ch;
}while(ch=='y' || ch=='Y');

}

void poly::ints(float z) //DEFINE FUNCTION INTS
{
int x1,x2,y1,y2,temp;

c=0;
for(int i=0;i<v;i++)
{
x1=p[i].x;
y1=p[i].y;
x2=p[i+1].x;
y2=p[i+1].y;
if(y2<y1)
{
temp=x1;
x1=x2;
x2=temp;
temp=y1;
y1=y2;
y2=temp;
}

if(z<=y2&&z>=y1)
{
if((y1-y2)==0)
x=x1;
else // used to make changes in x. so that we can fill our polygon
after cerain distance
{
x=((x2-x1)*(z-y1))/(y2-y1);
x=x+x1;
}
```

```
if(x<=xmax && x>=xmin)
inter[c++]=x;
    }
  }
}

void poly::sort(int z) //SORT FUNCTION
{
int temp,j,i;

for(i=0;i<v;i++)
{
line(p[i].x,p[i].y,p[i+1].x,p[i+1].y); // used to make hollow
outlines of a polygon
}
delay(100);
for(i=0; i<c;i+=2)
{
delay(100);
line(inter[i],z,inter[i+1],z); // Used to fill the polygon ....
}
}

int main() //START OF MAIN
{
int cl;
initwindow(500,600);
cleardevice();
poly x;
x.read();
x.calcs();
cleardevice();
cout<<"\n\tEnter the colour u want:(0-15)->"; //Selecting colour
cin>>cl;
setcolor(cl);
x.display();
closegraph(); //CLOSE OF GRAPH
getch();
return 0;
}
```



Q6 : - Write a Program to Implement various 2D Transformations like translating, reflection,scaling, shearing, etc.

```
#define _USE_MATH_DEFINES
#include <cmath>
#include <cstdlib>
#include <graphics.h>
#include <iostream>
#define COORD_SHIFT 100

using namespace std;

void clrscr()
{
#ifdef _WIN32
system("cls");
#elif __unix__
system("clear");
#endif
}
```

```
double **inputFigure(int n)
{
cout << "Enter the matrix for the 2-D shape (homogeneous):\n";

double **figure = NULL;
figure = new double *[n];

for (int i = 0; i < n; i++)
{
figure[i] = new double[3];
for (int j = 0; j < 3; j++)
{
cin >> figure[i][j];
}
}

return figure;

}

void drawFigure(double **points, int n)
{
setcolor(WHITE);
for (int i = 0; i < n; i++)
{
line(COORD_SHIFT + points[i][0],
COORD_SHIFT + points[i][1],
COORD_SHIFT + points[(i + 1) % n][0],
COORD_SHIFT + points[(i + 1) % n][1]);
}

delay(5e3);
cleardevice();
}

double **translate(double **figure, int dim, int m, int n)
{
double **_figure = NULL;
int T[dim][3] = {{1, 0, 0}, {0, 1, 0}, {m, n, 1}};

_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{
_figure[i] = new double[3];
for (int j = 0; j < 3; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];
}
}
}

return _figure;
}
```

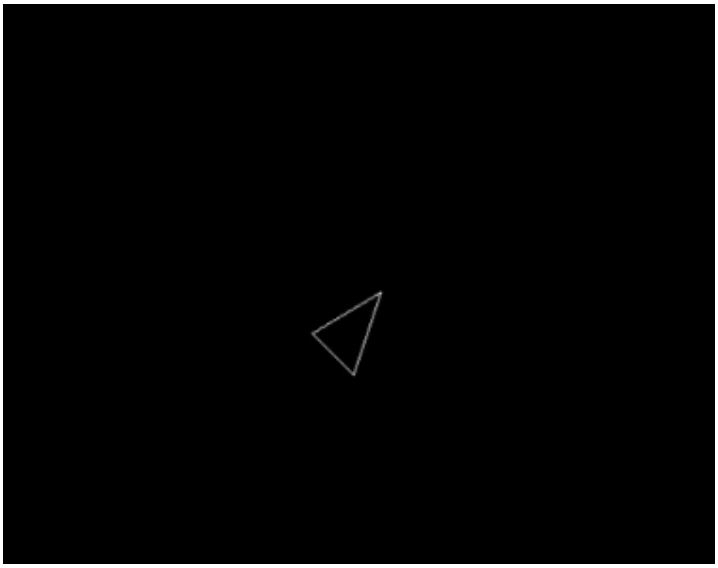
```
double **rotate(double **figure, int dim, double theta)
{
double **_figure = NULL;
double T[dim][3] = {{cos(theta * M_PI / 180.0), sin(theta * M_PI / 180.0),
0},
{-sin(theta * M_PI / 180.0), cos(theta * M_PI / 180.0),
0},
{0, 0, 1}};

_figure = new double *[dim];

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

{

_figure[i] = new double[3];
for (int j = 0; j < 2; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];
}
}
}

return _figure;
}

double **scale(double **figure, int dim, int m, int n)
{
double **_figure = NULL;
int T[dim][3] = {{m, 0, 0}, {0, n, 0}, {0, 0, 1}};

_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{
_figure[i] = new double[3];
for (int j = 0; j < 3; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];
}
}
}

return _figure;
}

double **reflect(double **figure, int dim, int c)
{
double **_figure = NULL;
int T[dim][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

switch (c)
{
```

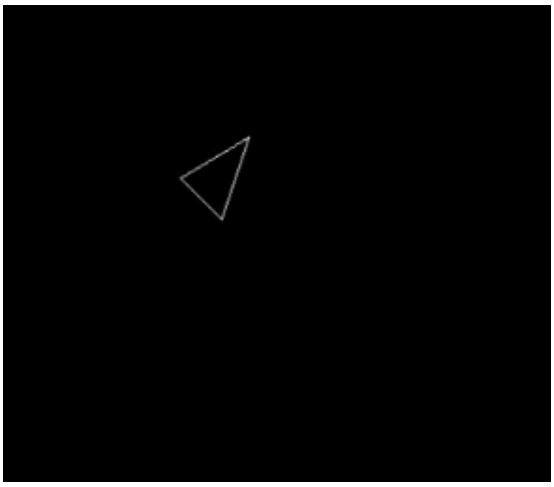
```
case 1:
T[1][1] = -1;
break;
case 2:
T[0][0] = -1;
break;
case 3:
T[0][0] = 0;
T[0][1] = 1;
T[1][0] = 1;

T[1][1] = 0;
break;
case 4:
T[0][0] = -1;
T[1][1] = -1;
break;
default:
return NULL;
break;
}

_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{
_figure[i] = new double[3];
for (int j = 0; j < 3; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];
}
}
}

return _figure;
}


double **shear(double **figure, int dim, int m, int n)
{
double **_figure = NULL;
int T[dim][3] = {{1, n, 0}, {m, 1, 0}, {0, 0, 1}};

_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{
_figure[i] = new double[3];
for (int j = 0; j < 3; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];
}
}
```

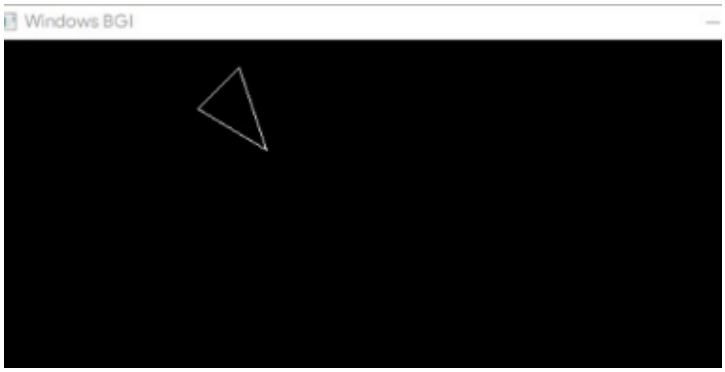
```
}

return _figure;
}

void menu(double **figure, int dim)
{

int ch = 0;
double **_figure;

do
{
clrscr();
cout << "\nMenu\n-------\n(1) Translation\n(2) Rotation";
cout << "\n(3) Scaling\n(4) Reflection\n(5) Shearing";
cout << "\n(6) View Figure\n(7) Exit\n\nEnter Choice: ";
cin >> ch;
cout << endl;
switch (ch)
{
case 1:
int m, n;

cout << "Enter translation in x-axis: ";
cin >> m;
cout << "Enter translation in y-axis: ";
cin >> n;

_figure = translate(figure, dim, m, n);

cout << "Drawing Original Figure...\n";
drawFigure(figure, dim);

cout << "Drawing Transformed Figure...\n";
drawFigure(_figure, dim);
break;
case 2:
double theta;

cout << "Enter rotation angle (degrees): ";
cin >> theta;

_figure = rotate(figure, dim, theta);

cout << "Drawing Original Figure...\n";
drawFigure(figure, dim);

cout << "Drawing Transformed Figure...\n";
drawFigure(_figure, dim);
break;
case 3:
cout << "Enter scaling in x-axis: ";
cin >> m;
cout << "Enter scaling in y-axis: ";
cin >> n;
```

```
_figure = scale(figure, dim, m, n);

cout << "Drawing Original Figure...\n";

drawFigure(figure, dim);

cout << "Drawing Transformed Figure...\n";
drawFigure(_figure, dim);
break;
case 4:
cout << "Reflect along\n(1) x-axis\n(2) y-axis\n(3) y = x\n(4) y = -x\n"
<< "\nEnter Choice: ";
cin >> m;

_figure = reflect(figure, dim, m);

cout << "Drawing Original Figure...\n";
drawFigure(figure, dim);

cout << "Drawing Transformed Figure...\n";
drawFigure(_figure, dim);
break;
case 5:
cout << "Enter shearing in x-axis: ";
cin >> m;
cout << "Enter shearing in y-axis: ";
cin >> n;

_figure = shear(figure, dim, m, n);

cout << "Drawing Original Figure...\n";
drawFigure(figure, dim);

cout << "Drawing Transformed Figure...\n";
drawFigure(_figure, dim);
break;
case 6:
cout << "Drawing Original Figure...\n";
drawFigure(figure, dim);
break;
case 7:
default:
break;
}

delete _figure;

cout << endl
<< "Finished..."
<< endl;

if (ch != 7)
{
cout << "\nPress Enter to continue ...\n";
cin.ignore();
cin.get();
}
```

```
} while (ch != 7);
};

int main(void)
{
int n;
double **fig;
int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

cout << "Enter number of points in the figure: ";
cin >> n;

fig = inputFigure(n);

menu(fig, n);

delete fig;
closegraph();

return 0;
}
```







Q7 :- Write a Program to Implement various 3D Transformation

```
#define _USE_MATH_DEFINES
#include <conio.h>
#include <graphics.h>
#include <iostream>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
#define COORD_SHIFT 100

using namespace std;

double **inputFigure(int n)
{
cout << "Enter the matrix for the 3-D shape (homogeneous):\n";

double **figure = NULL;
figure = new double *[n];
```

```
for (int i = 0; i < n; i++)
{
figure[i] = new double[4];
for (int j = 0; j < 4; j++)
{
cin >> figure[i][j];
}
}

return figure;
}

void drawFigure(double **points, int n, int p)
{
int a, b;
switch (p)
{
case 1:
a = 0;
b = 1;
break;
case 2:
a = 0;

b = 2;
break;
case 3:
a = 1;
b = 2;
break;
}

setcolor(WHITE);

for (int i = 0; i < n; i++)
{
line(COORD_SHIFT + points[i][a],
COORD_SHIFT + points[i][b],
COORD_SHIFT + points[(i + 1) % n][a],
COORD_SHIFT + points[(i + 1) % n][b]);

cout << points[i][0] << "\t"
<< points[i][1] << "\t"
<< points[i][2] << "\t"
<< points[i][3] << " "
<< ":: (" << points[i][a] << ", " << points[i][b] << ") "
<< "-> (" << points[(i + 1) % n][a] << ", " << points[(i + 1) % n][b]
<< ")"
<< endl;
}

delay(5e3);
cleardevice();
}

double **translate(double **figure, int dim, int l, int m, int n)
{
```

```
double **_figure = NULL;
int T[dim][4] = {{1, 0, 0, 0},
{0, 1, 0, 0},
{0, 0, 1, 0},
{l, m, n, 1}};

_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{
_figure[i] = new double[4];
for (int j = 0; j < 4; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];
}
}
}

return _figure;
}

double **rotate(double **figure, int dim, double theta)
{
double **_figure = NULL;
double T[dim][3] = {{cos(theta * M_PI / 180.0), sin(theta * M_PI / 180.0),
0},
{-sin(theta * M_PI / 180.0), cos(theta * M_PI / 180.0),
0},
{0, 0, 1}};

_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{

_figure[i] = new double[3];
for (int j = 0; j < 2; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];
}
}
}

return _figure;
}

double **scale(double **figure, int dim, double l, double m, double n)
{
double **_figure = NULL;
double T[dim][4] = {{l, 0, 0, 0},
{0, m, 0, 0},
{0, 0, n, 0},
{0, 0, 0, 1}};
```

```
_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{
_figure[i] = new double[4];
for (int j = 0; j < 4; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];
}
}
}

return _figure;
}

double **scale(double **figure, int dim, double s)
{

double **_figure = NULL;
double T[dim][4] = {{1, 0, 0, 0},
{0, 1, 0, 0},
{0, 0, 1, 0},
{0, 0, 0, s}};

_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{
_figure[i] = new double[4];
for (int j = 0; j < 4; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];
}
}
}

return _figure;
}

double **reflect(double **figure, int dim, int c)
{
double **_figure = NULL;
int T[dim][3] = {{1, 0, 0}, {0, 1, 0}, {0, 0, 1}};

switch (c)
{
case 1:
T[1][1] = -1;
break;
case 2:
T[0][0] = -1;
break;
```

```
case 3:
T[0][0] = 0;
T[0][1] = 1;
T[1][0] = 1;
T[1][1] = 0;
break;
case 4:

T[0][0] = -1;
T[1][1] = -1;
break;
default:

return NULL;
break;
}

_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{
_figure[i] = new double[3];
for (int j = 0; j < 3; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];
}
}
}

return _figure;
}

double **shear(double **figure, int dim, int m, int n)
{
double **_figure = NULL;
int T[dim][3] = {{1, n, 0}, {m, 1, 0}, {0, 0, 1}};

_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{
_figure[i] = new double[3];
for (int j = 0; j < 3; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * T[k][j];

}
}
}

return _figure;
}
```

```
double **project(double **figure, int dim, int p)
{
double **_figure = NULL;
int P[dim][4] = {{1, 0, 0, 0},
{0, 1, 0, 0},
{0, 0, 1, 0},
{0, 0, 0, 1}};

switch (p)

{
case 1:
P[2][2] = 0;
break;
case 2:
P[1][1] = 0;
break;
case 3:
P[0][0] = 0;
break;
}

_figure = new double *[dim];

for (int i = 0; i < dim; i++)
{
_figure[i] = new double[4];
for (int j = 0; j < 4; j++)
{
for (int k = 0; k < dim; k++)
{
_figure[i][j] += figure[i][k] * P[k][j];
}
}
}

return _figure;
}

void menu(double **figure, int dim)
{
int ch = 0;
double l, m, n, p;
double **_figure, **_projected;

do
{
//clrscr();
cout << "\nMenu\n-------\n(1) Translation\n(2) Rotation";
cout << "\n(3) Scaling\n(4) Reflection\n(5) Shearing";
cout << "\n(6) View Figure\n(7) Exit\n\nEnter Choice: ";
cin >> ch;
cout << endl;
switch (ch)
{
case 1:
```

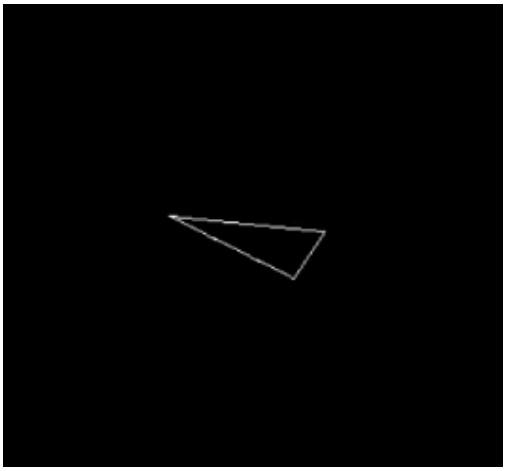
```
cout << "Enter translation in x-axis: ";
cin >> l;
cout << "Enter translation in y-axis: ";
cin >> m;
cout << "Enter translation in z-axis: ";

cin >> n;

_figure = translate(figure, dim, l, m, n);

cout << "\nChoose Projection:\n(1) xy-plane\n(2) xz-plane\n(3)
yz-plane\n"
<< "\nEnter Choice: ";
cin >> p;

if (p > 3 || p < 1)
{
cout << "\nInvalid Projection!";

cin.ignore();
cin.get();
continue;
}

cout << "Drawing Original Figure...\n";
drawFigure(project(figure, dim, p), dim, p);

cout << "Drawing Transformed Figure...\n";
drawFigure(project(_figure, dim, p), dim, p);
break;

case 3:
int scalingCh;
cout << "Scaling:\n(1) Overall Scaling\n(2) Local Scaling\n\nEnter
Choice: ";
cin >> scalingCh;

switch (scalingCh)
{
case 1:
cout << "Enter scaling factor: ";
cin >> l;
_figure = scale(figure, dim, l);
break;

case 2:

cout << "Enter scaling in x-axis: ";
cin >> l;
cout << "Enter scaling in y-axis: ";
cin >> m;
cout << "Enter scaling in z-axis: ";
cin >> n;
_figure = scale(figure, dim, l, m, n);
break;
}
```

```
cout << "Drawing Original Figure...\n";
drawFigure(project(figure, dim, p), dim, p);

cout << "Drawing Transformed Figure...\n";
drawFigure(project(_figure, dim, p), dim, p);
break;

case 6

cout << "\nChoose Projection:\n(1) xy-plane\n(2) xz-plane\n(3)
yz-plane\n"
<< "\nEnter Choice: ";
cin >> p;

if (p > 3 || p < 1)

{
cout << "\nInvalid Projection!";
cin.ignore();
cin.get();
continue;
}

cout << "Drawing Original Figure...\n";
drawFigure(project(figure, dim, p), dim, p);
case 7:
default:
break;
}

if (ch != 6)
delete _figure;

cout << endl
<< "Finished..."
<< endl;

if (ch != 7)
{
cout << "\nPress Enter to continue ...\n";
cin.ignore();

cin.get();
}
} while (ch != 7);
};

int main(void)
{
int n;
double **fig;
int gd = DETECT, gm;

initgraph(&gd, &gm, NULL);

cout << "Enter number of points in the figure: ";
cin >> n;
```

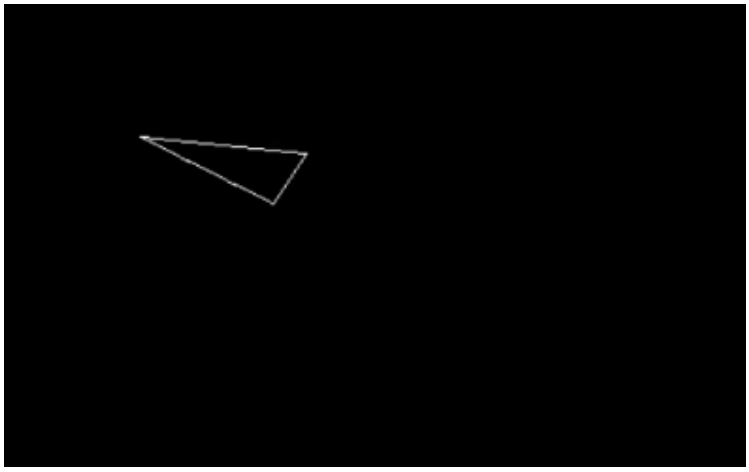
```
fig = inputFigure(n);

menu(fig, n);

delete fig;
closegraph();

return 0;
}
```

XZ Plane

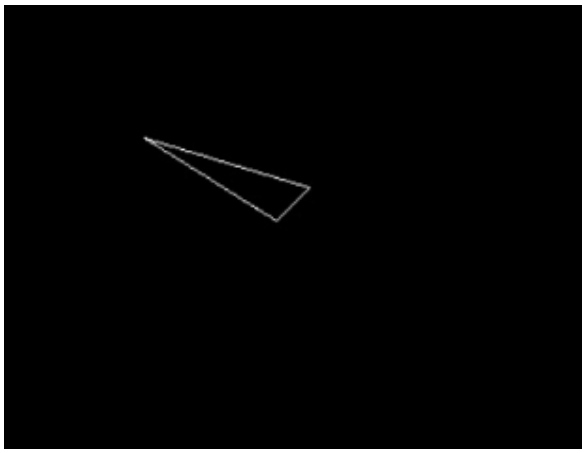
XY Plane



Translation XY



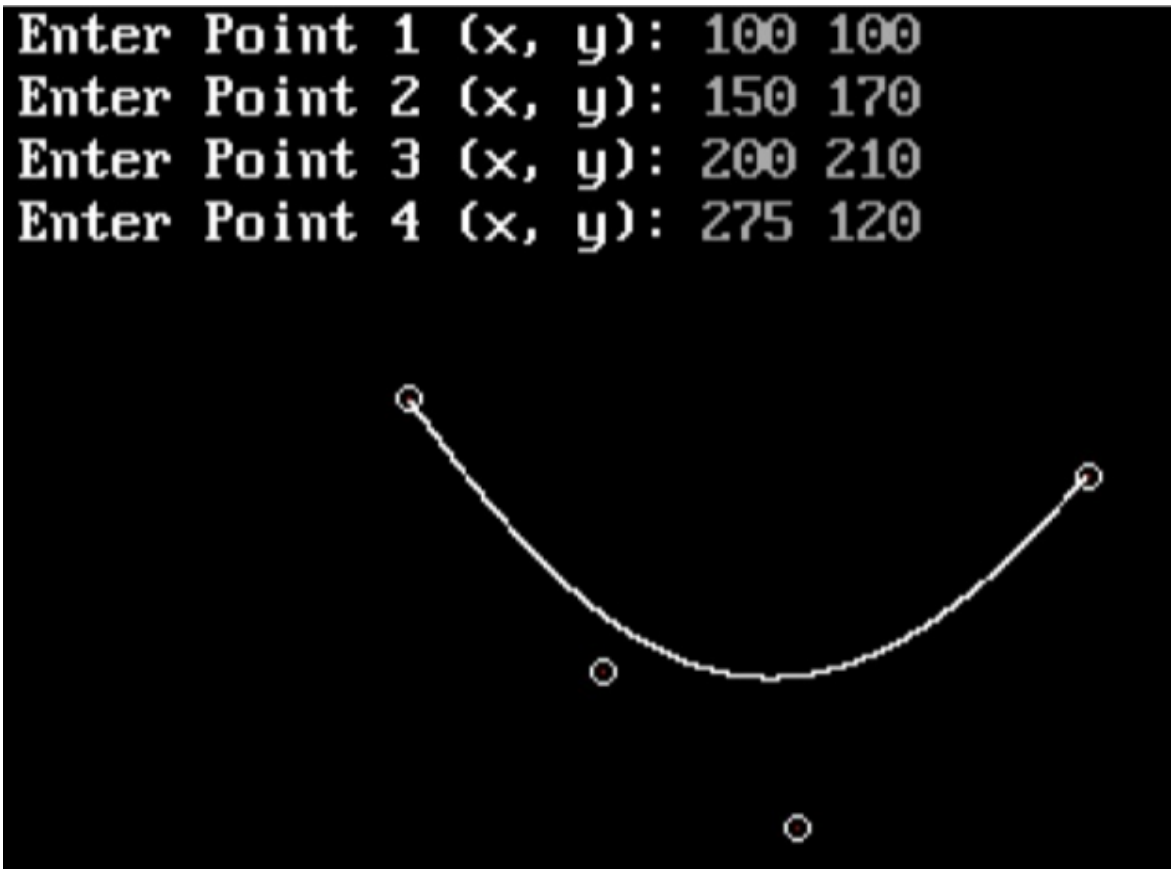
YZ Plane



Q8 :- Write a Program to Draw Hermit/ Bezier Curve.

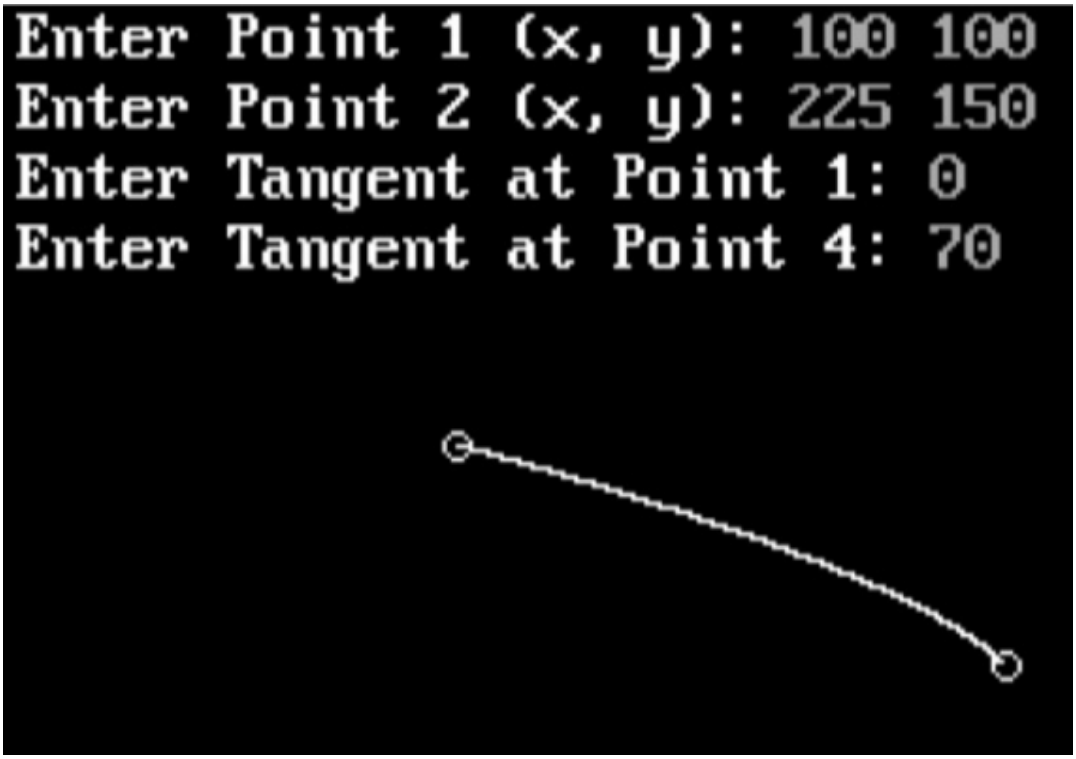
```
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

void bezier(int x[4], int y[4])
{
for (double t = 0.0; t < 1.0; t += 0.00005)
{
double xt = pow(1 - t, 3) * x[0] + 3 * t * pow(1 - t, 2) * x[1] + 3 *
pow(t, 2) * (1 - t) * x[2] + pow(t, 3) * x[3];
double yt = pow(1 - t, 3) * y[0] + 3 * t * pow(1 - t, 2) * y[1] + 3 *
```

```
pow(t, 2) * (1 - t) * y[2] + pow(t, 3) * y[3];
putpixel(xt, yt, WHITE);
}

for (int i = 0; i < 4; i++)
{
circle(x[i], y[i], 3);
}

getch();
closegraph();
return;
}

void main()
{
int i;
int x[4], y[4];
int gd = DETECT, gm, errorcode;

initgraph(&gd, &gm, "..\\bgi");

for (i = 0; i < 4; i++)
{
cout << "Enter Point " << i + 1 << " (x, y): ";
cin >> x[i] >> y[i];
}

bezier(x, y);
return;
}
```




```
#include <conio.h>
#include <graphics.h>
#include <iostream.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

using namespace std;

struct point
{
int x, y;
};

void hermite(point p1, point p4, double r1, double r4)
{
float x, y, t;
for (t = 0.0; t <= 1.0; t += 0.00005)
{
x = (2 * pow(t, 3) - 3 * pow(t, 2) + 1) * p1.x +
(-2 * pow(t, 3) + 3 * pow(t, 2)) * p4.x +
(pow(t, 3) - 2 * pow(t, 2) + t) * r1 +
(pow(t, 3) - pow(t, 2)) * r4;

y = (2 * pow(t, 3) - 3 * pow(t, 2) + 1) * p1.y +
```

```
(-2 * pow(t, 3) + 3 * pow(t, 2)) * p4.y +
(pow(t, 3) - 2 * pow(t, 2) + 1) * r1 +
(pow(t, 3) - pow(t, 2)) * r4;
putpixel(x, y, WHITE);
}

circle(p1.x, p1.y, 3);
circle(p4.x, p4.y, 3);
}

void main()
{
point p1, p4;
double r1, r4;

int gd = DETECT, gm;
initgraph(&gd, &gm, "..\\BGI");

cout << "Enter Point 1 (x, y): ";
cin >> p1.x >> p1.y;
cout << "Enter Point 2 (x, y): ";
cin >> p4.x >> p4.y;
cout << "Enter Tangent at Point 1: ";
cin >> r1;
cout << "Enter Tangent at Point 4: ";
cin >> r4;

hermite(p1, p4, r1, r4);

getch();
closegraph();
```


Enter Point 1 (x, y): 100 100
Enter Point 2 (x, y): 225 150
Enter Tangent at Point 1: 0
Enter Tangent at Point 4: 70