# CGL All Asssignments

## Assignment No – 2

**Title:- DDA Algorithm**

```
#include<iostream>
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
#define ROUND(x)((int)(x+0.5))
using namespace std;
int x1,x2,z1,z2;

void draw_pixel(int x, int y)
{
glColor3f(1.0,0.0,0.0);
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}


void drawline(int X1, int Y1, int X2, int Y2)
{
float x,y,dx,dy,length; int i;
dx=abs(X2-X1);
dy=abs(Y2-Y1);
if(dx>=dy)
   length=dx;
else length=dy;
dx=(X2-X1)/length;
dy=(Y2-Y1)/length;
x=X1;
y=Y1;
i=1;
while(i<=length)
{
draw_pixel(ROUND(x),ROUND(y));
x=x+dx;
y=y+dy;
i=i+1;
}
glFlush();
}

void drawpatt(int ax, int ay, int bx, int by, int cx, int cy, int dx, int dy, int n)
{
int m1x, m1y, m2x, m2y, m3x, m3y, m4x, m4y;
drawline(ax,ay,bx,by);
drawline(bx,by,cx,cy);
drawline(cx,cy,dx,dy);
drawline(dx,dy,ax,ay);

//midpoint calculations;

m1x=(ax+bx)/2;
m1y=(ay+by)/2;
m2x=(bx+cx)/2;
m2y=(by+cy)/2;
m3x=(cx+dx)/2;
m3y=(cy+dy)/2;
m4x=(dx+ax)/2;
m4y=(dy+ay)/2;
n--; if(n!=0)
{
        drawpatt(m1x, m1y, m2x, m2y, m3x, m3y, m4x, m4y,n);
}

}

void display(void)
{
float x,y,dx,dy,length;
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2i(-200,0);
glVertex2i(200,0);
```

```
glVertex2i(0,-200);
glVertex2i(0,200);
glEnd();
drawpatt(x1,z1,x1,z2,x2,z2,x2,x1,5);
}

void Init(void)
{
glClearColor(1,1,1,0);
gluOrtho2D(-200.0,200.0,-200.0,200.0);
}

int main(int argc, char **argv)
{
/* cout<<"\n Enter the value of left bottom x1::"; cin<<x1;
cout<<"\n  Enter the value of left bottom y1:"; cin<<y1;
cout<<"\n Enter the value of right top x2:"; cin<<x2;
cout<<"\n Enter the value of right top y2:"; cin<<y2; */

printf("Enter the value of left bottom x1:");
scanf("%d",&x1);
printf("Enter the value of left bottom y1:");
scanf("%d",&z1);
printf("Enter the value of right top x2:");
scanf("%d",&x2);
printf("Enter the value of right top y2:");
scanf("%d",&z2);

glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(100,100);
glutCreateWindow("Pattern Drawing");
Init();
glClear(GL_COLOR_BUFFER_BIT);
glutDisplayFunc(display);
glFlush();
glutMainLoop();
return 0;
}
```

# Assignment No – 3

## Title:- Bresenham’s Algorithm

```
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
int ww=1200,wh=800;
int xi,yi,xf,yf,r;
float theta=2.0933;
void putpixel(int x,int y)
{
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
glFlush();
}
double round(double n)
{
return(n>=0)?(int)(n+0.5):(int)(n-0.5);
}
void Bresenham_circle(int xc,int yc,int xr)
{
int x=0,y=xr;
int d=3-2*y;
while(x<=y)
{
putpixel(xc+x,yc+y);
putpixel(xc+y,yc+x);
putpixel(xc-x,yc+y);
putpixel(xc-x,yc-y);
putpixel(xc-y,yc+x);
putpixel(xc-y,yc-x);
putpixel(xc+y,yc-x);
putpixel(xc+x,yc-y);
if(d<0)
d=d+(4*x)+6;
else
{
d=d+(4*(x-y))+10;
```

```
y--;
}
x++;
}
}
void bresenhamAlg(int x0,int y0,int x1,int y1)
{
int dx=abs(x1-x0);
int dy=abs(y1-y0);
int x,y;
if(dx>=dy)
{
int d=2*dy-dx;
int ds=2*dy;
int dt=2*(dy-dx);
if(x0<x1)
{
x=x0;
y=y0;
}
else
{
x=x1;
y=y1;
x1=x0;
y1=y0;
}
putpixel(x,y);
while(x<x1)
{
if(d<0)
d+=ds;
else
{
if(y<y1)
{
y++;
d+=dt;
}
else
```

```
{
y--;
d+=dt;
}
}
x++;
putpixel(x,y);
}
}
else
{
int d=2*dx-dy;
int ds=2*dx;
int dt=2*(dx-dy);
if(y0<y1)
{
x=x0;
y=y0;
}
else
{
x=x1;
y=y1;
y1=y0;
x1=x0;

}
putpixel(x,y);
while(y<y1)
{
if(d<0)
d+=ds;
else
{
if(x>x1)
{
x--;
d+=dt;
}
else
```

```
{
x++;
d+=dt;
}
}
y++;
putpixel(x,y);
}
}
}
void triangle(int ix,int iy)
{
int x=ix,y=iy,x1,x2,y1,y2;
x1=xi+(x-xi)*cos(theta)-(y-yi)*sin(theta);
y1=yi-r/2;
x2=xi+(x-xi)*cos(theta)+(y-yi)*sin(theta);
y2=yi-r/2;
bresenhamAlg(x,y,x1,y1);
bresenhamAlg(x,y,x2,y2);
bresenhamAlg(x1,y1,x2,y2);
}
void display()
{
glClearColor(1.0,1.0,1.0,1.0);
glColor3f(0.0,0.0,0.0);
glClear(GL_COLOR_BUFFER_BIT);
glutSwapBuffers();
Bresenham_circle(xi,yi,r);
Bresenham_circle(xi,yi,r/2);
triangle(xi,(yi+r));
glFlush();
}
void myinit()
{
glViewport(0,0,ww,wh);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,(GLdouble)ww,0.0,(GLdouble)wh);
glMatrixMode(GL_MODELVIEW);
}
```

```
int main(int argc,char **argv)
{

printf("Enter centre of the circle");
scanf("%d%d",&xi,&yi);
printf("\n Enter radius of the circle");
scanf("%d",&r);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(ww,wh);
glutCreateWindow("Bresenham-Circle");
myinit();
glutDisplayFunc(display);
glutMainLoop();
return 0;
}
```

# Assignment No:- 4

**Title:- Implement the following polygon filling methods :**

1) **Flood fill / Seed fill**
2) **Boundary fill; using mouse click, keyboard Interface and menu driven programming.**

```
#include<iostream>
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>
using namespace std;
int option = 1;
struct Point
{
GLint x;
GLint y;
};
struct Color
{
GLfloat r;
GLfloat g;
GLfloat b;
};
void init()
{
glClearColor(0.0, 0.0, 0.0, 0.0);
glColor3f(1.0, 1.0, 1.0);
glPointSize(1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,500.0,0,500.0);
}
Color getPixelColor(GLint x, GLint y)
{
Color color;
glReadPixels(x, y, 1, 1, GL_RGB, GL_FLOAT, &color);
return color;
}
void setPixelColor(GLint x, GLint y, Color color)
{
glColor3f(color.r,color.g,color.b);
glBegin(GL_POINTS);
glVertex2i(x, y);
glEnd();
glFlush();
}
void floodFill(GLint x, GLint y, Color oldColor, Color newColor)
{
Color color;
color = getPixelColor(x, y);
if(color.r == oldColor.r && color.g == oldColor.g && color.b == oldColor.b)
{
setPixelColor(x, y, newColor);
floodFill(x+1, y, oldColor, newColor);
floodFill(x, y+1, oldColor, newColor);
floodFill(x-1, y, oldColor, newColor);
floodFill(x, y-1, oldColor, newColor);
}
return;
}
void boundryFill(GLint x, GLint y, Color fColor, Color bColor)
{
Color color;
color = getPixelColor(x, y);
if((color.r !=fColor.r || color.g != fColor.g || color.b != fColor.b) &&
(color.r != bColor.r || color.g !=bColor.g || color.b !=bColor.b))
{
setPixelColor(x, y, fColor);
boundryFill(x+1, y, fColor, bColor);
boundryFill(x, y+1, fColor, bColor);
boundryFill(x-1, y, fColor, bColor);
boundryFill(x, y-1, fColor, bColor);
}
return;
}
void onMouseClick(int button, int state, int x, int y)
{
Color newColor = {0.0f, 1.0f, 0.0f};
Color oldColor = {0.0f, 0.0f, 0.0f};
Color bColor = {1.0f, 1.0, 1.0f};
Color fColor = {0.0f, 0.0f, 1.0f};
if(option == 1)
floodFill(x, 500-y, oldColor, newColor);
else
boundryFill(x, 500-y, fColor, bColor);
}
```

```
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_LINE_LOOP); // Draw Polygon
glVertex2i(250, 250);
glVertex2i(250, 300);
glVertex2i(300, 300);
glVertex2i(300, 250);
glEnd();
glFlush();
}
int main(int argc, char** argv)
{
cout<<"1: Flood Fill"<<endl;
cout<<"2: Boundary Fill"<<endl;
cout<<"Enter Option :";
cin>>option;
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500, 500);
glutInitWindowPosition(0, 0);
glutCreateWindow("Polygon Filling");
init();
glutDisplayFunc(display),
glutMouseFunc(onMouseClick);
glutMainLoop();
return 0;
}
```

# Assignment No:- 5

**Title:- Implement Cohen Sutherland polygon clipping method to clip the Polygon with respect the viewport and window. Use mouse click, Keyboard interface.**

```
#include<GL/glut.h>
#define outcode int
double xmin = 100, ymin = 100, xmax = 200, ymax = 200; // clipping window
double xvmin=300, yvmin=300,xvmax=400, yvmax=400; // view port
double x0, y0, x1, y1;
const int RIGHT=8;
const int LEFT=2;
const int TOP=4;
const int BOTTOM=1;
outcode ComputeOutCode(double x, double y);
void CohenSutherland(double x0, double y0, double x1, double y1)
{
outcode outcode0, outcode1, outcodeOut;
bool accept=false, done=false;
outcode0=ComputeOutCode(x0,y0);
outcode1=ComputeOutCode(x1,y1);
do
{
if(!(outcode0|outcode1))
{
accept=true; // Line is completely visible
done=true;
}
else if(outcode0&outcode1)
done=true; // Line is completely invisible
else // Line is partially visible
{
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=ComputeOutCode(x0,y0);
}
else
{
x1=x;
y1=y;
outcode1=ComputeOutCode(x1,y1);
}
}
}while(!done);
if(accept)
{
double sx=(xvmax-xvmin)/(xmax-xmin);
double sy=(yvmax-yvmin)/(ymax-ymin);
double vx0=xvmin+(x0-xmin)*sx;
double vy0=yvmin+(y0-ymin)*sy;
double vx1=xvmin+(x1-xmin)*sx;
double vy1=yvmin+(y1-ymin)*sy;
glColor3f(1.0, 1.0, 1.0);
glBegin(GL_LINE_LOOP);
glVertex2f(xvmin, yvmin);
glVertex2f(xvmax, yvmin);
glVertex2f(xvmax, yvmax);
glVertex2f(xvmin, yvmax);
glEnd();
glColor3f(1.0,1.0,0.0);
glBegin(GL_LINES);
glVertex2d(vx0, vy0);
```

```
glVertex2d(vx1,vy1);
glEnd();
}
}
outcode ComputeOutCode(double x, double y)
{
outcode code=0; // Assign region code
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 display()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,1.0,0.0);
glBegin(GL_LINE_LOOP); // Draw clipping window
glVertex2f(xmin, ymin);
glVertex2f(xmax, ymin);
glVertex2f(xmax, ymax);
glVertex2f(xmin, ymax);
glEnd();
glFlush();
}
void myinit()
{
glClearColor(0.0,0.0,0.0,1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,500.0,0.0,500.0);
}
void myKeyboard(unsigned char key, int mouseX, int mouseY)
{
switch (key)
{
  case 27: // Press ESC key to exit
  exit(0);
}
```

```
}
void myMouse(int button, int state, int x, int y)
{
static int pt = 0;
if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
{
glColor3f(1.0, 1.0,1.0);
if (pt == 0) // Get the start point of the line
{
   x0 = x;
   y0= 500-y;
   pt++;
}
else if (pt == 1) // Get the end point of the line
{
   x1 = x;
   y1 = 500-y;
   glBegin(GL_LINES); // Draw initial line
   glVertex2f(x0,y0);
   glVertex2f(x1,y1);
   glEnd();
   pt = 0;
}
}
CohenSutherland(x0,y0,x1,y1);
glFlush(); // Send output to display
}
int main(int argc, char** argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Cohen-Sutherland Line Clipping");
glutMouseFunc(myMouse); // Register mouse function
glutKeyboardFunc(myKeyboard); // Register keyboard function
glutDisplayFunc(display);
myinit();
glutMainLoop();
return 0;
}
```

# Assignment No.:- 6

## Title:- 2D Transformation

```
#include<iostream>
#include<math.h>
#include<GL/glut.h>
using namespace std;
int choice;
int x1,x2,x3,x4,yy1,y2,y3,y4,nx1,nx2,nx3,nx4,ny1,ny2,ny3,ny4,c,shx,shy;
float sx,sy,xt,yt,r;
double t;
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0,1.0,1.0);
glBegin(GL_LINES);
glVertex2i(-500,0);
glVertex2i(500,0);
glVertex2i(0,-500);
glVertex2i(0,500);
glEnd();
glColor3f(1.0,1.0,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(x1,yy1);
glVertex2f(x2,y2);
glVertex2f(x3,y3);
glVertex2f(x4,y4);
glEnd();
glColor3f(1.0,0.0,0.0);
glBegin(GL_LINE_LOOP);
glVertex2f(nx1,ny1);
glVertex2f(nx2,ny2);
glVertex2f(nx3,ny3);
glVertex2f(nx4,ny4);
glEnd();
glFlush();
}
void myinit()
{
glClearColor(0.0,0.0,0.0,1.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-500.0,500.0,-500.0,500.0);
```

```
}
int main(int argc,char **argv)
{
x1=15;
yy1=15;
x2=75;
y2=45;
x3=105;
y3=105;
x4=45;
y4=75;
cout<<"\n 1. Translation \n 2. Rotation \n 3. Scaling \n 4. Shear \n 5.Exit \n
Enter Your Choice";
cin>>c;
switch(c)
{
case 1:
cout<<"Enter the translation factor x";
cin>>xt;
cout<<"Enter the translation factor y";
cin>>yt;
nx1=x1+xt;
ny1=yy1+yt;
nx2=x2+xt;
ny2=y2+yt;
nx3=x3+xt;
ny3=y3+yt;
nx4=x4+xt;
ny4=y4+yt;
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Translation");
glutDisplayFunc(display);
myinit();
glutMainLoop();
break;

case 2:
cout<<"Enter the angle of rotation";
cin>>r;
t=3.14*r/180;
```

```
nx1=(x1*cos(t)-yy1*sin(t));
ny1=(x1*sin(t)+yy1*cos(t));
nx2=(x2*cos(t)-y2*sin(t));
ny2=(x2*sin(t)+y2*cos(t));
nx3=(x3*cos(t)-y3*sin(t));
ny3=(x3*sin(t)+y3*cos(t));
nx4=(x4*cos(t)-y4*sin(t));
ny4=(x4*sin(t)+y4*cos(t));
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Rotation");
glutDisplayFunc(display);
myinit();
glutMainLoop();
break;

case 3:
cout<<"Enter the scaling factor x";
cin>>sx;
cout<<"Enter the scaling factor y";
cin>>sy;
nx1=x1*sx;
ny1=yy1*sy;
nx2=x2*sx;
ny2=y2*sy;
nx3=x3*sx;
ny3=y3*sy;
nx4=x4*sx;
ny4=y4*sy;
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Scaling");
glutDisplayFunc(display);
myinit();
glutMainLoop();
break;
case 4:
cout<<"Enter the shear factor x";
cin>>shx;
```

```
cout<<"Enter the shear factor y";
cin>>shy;
nx1=(x1+shx*yy1);
nx2=(x2+shx*y2);
nx3=(x3+shx*y3);
nx4=(x4+shx*y4);
ny1=(yy1+shy*x1);
ny2=(y2+shy*x2);
ny3=(y3+shy*x3);
ny4=(y4+shy*x4);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("Shear");
glutDisplayFunc(display);
myinit();
glutMainLoop();
break;
case 5:
break;
default:
cout<<"Enter the correct choice";

}
return 0;
}
```

# Assignment No:- 7

**Title:- Generate fractal patterns using i) Bezier ii) Koch Curve**

## Bezier Curve

```
#include <iostream>
#include <stdlib.h>
#include <GL/glut.h>
#include <math.h>
using namespace std;
class Point //Point class for taking the points
{
public:
float x, y;
void setxy(float x2, float y2)
{
x = x2; y = y2;
}
const Point & operator=(const Point &rPoint) //operator overloading for '=' sign
{
x = rPoint.x;
y = rPoint.y;
return *this;
}
};
int factorial(int n)
{
if (n<=1)
return(1);
else
n=n*factorial(n-1);
return n;
}
float binomial_coff(float n,float k)
{
float ans;
ans = factorial(n)/(factorial(k)*factorial(n-k));
return ans;
}
Point abc[20];
int SCREEN_HEIGHT = 500;
int points = 0;
int clicks = 4;
void myInit()
```

```
{
glClearColor(0.0,0.0,0.0,0.0);
glColor3f(1.0,1.0,1.0);
glPointSize(3);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,500.0,0.0,500.0);
}
void drawDot(int x, int y)
{
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
glFlush();
}
void drawLine(Point p1, Point p2)
{
glBegin(GL_LINES);
glVertex2f(p1.x, p1.y);
glVertex2f(p2.x, p2.y);
glEnd();
glFlush();
}
Point drawBezier(Point PT[], double t)//Calculate the bezier point
{
Point P;
P.x=pow((1-t),3)*PT[0].x + 3*t*pow((1-t),2)*PT[1].x + 3*(1-
t)*pow(t,2)*PT[2].x+pow(t,3)*PT[3].x;
P.y=pow((1-t),3)*PT[0].y + 3*t*pow((1-t),2)*PT[1].y + 3*(1-
t)*pow(t,2)*PT[2].y+pow(t,3)*PT[3].y;
return P;
}
Point drawBezierGeneralized(Point PT[], double t) //Calculate the bezier point
[generalized]
{
Point P;
P.x = 0; P.y = 0;
for(int i=0;i<clicks;i++)
{
P.x = P.x+binomial_coff((float)(clicks-1),(float)i)*pow(t,(double)i)*pow((1-
t),(clicks-1-i))*PT[i].x;
P.y = P.y+binomial_coff((float)(clicks-1),(float)i)*pow(t,(double)i)*pow((1-
t),(clicks-1-i))*PT[i].y;
}
return P;
}
void myMouse(int button, int state, int x, int y) // If left button was clicked
{
if(button == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
{
abc[points].setxy((float)x,(float)(SCREEN_HEIGHT-y));
// Store where mouse was clicked, Y is backwards.
points++;
drawDot(x,SCREEN_HEIGHT-y); // Draw the red dot.
if(points == clicks) // If (click-amout) points are drawn do the curve.
{
glColor3f(1.0,0.0,0.0);
for(int k=0;k<clicks-1;k++) // Drawing the control lines
drawLine(abc[k], abc[k+1]);
Point p1 = abc[0];
glColor3f(1.0,1.0,1.0);
// Draw each segment of the curve.
// Make t increment in smaller amounts for a more detailed curve.
for(double t = 0.0;t <= 1.0; t += 0.02)
{
Point p2= drawBezierGeneralized(abc,t);
drawLine(p1, p2);
p1 = p2;
}
glColor3f(0.0,0.0,0.0);
points = 0;
}
}
}
void myDisplay()
{
glClear(GL_COLOR_BUFFER_BIT);
glFlush();
}
int main(int argc, char *argv[])
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(100,150);
glutCreateWindow("Bezier Curve");
```

```
glutMouseFunc(myMouse);
glutDisplayFunc(myDisplay);
myInit();
glutMainLoop();
return 0;
}
```

## Koch Curve

```
#include<iostream>
#include<stdio.h>
#include<GL/glut.h>
#include<stdlib.h>
using namespace std;
#define SIN 0.86602540 // oin(60 dogreen)
int n;

int x1 = 0, x2 = 550, y1=0, y2 = 0;
void koch(int x1, int y1, int x2, int y2, int m)
{
int xx, yy, x[5], y[5], lx, ly, offx=50, offy = 300;
lx = (x2-x1)/3;
ly = (y2-y1)/3;
x[0] =x1;                    // Store point po
y[0] = y1;
x[4] = x2;                   // Store point p4
y[4] = y2;
x[1] =x[0] + lx;             // Store point p1
y[1] = y[0] + ly;
x[3] = x[0] + 2*lx;          // Store point p3
y[3] = y[0] + 2*ly;
xx = x[3] -x[1];             // Translate point p2 to origin
yy = y[3]- y[1];
x[2] = xx*(0.5) + yy*(SIN);    // Perform rotation for point p3
y[2] = -xx*(SIN) + yy*(0.5);
x[2] = x[2] + x[1];          // Perform inverse translation
y[2] = y[2] + y[1];
if(m>0)
{
koch(x[0], y[0], x[1], y[1], m-1);            // Recursive call to Draw part1
koch(x[1], y[1], x[2], y[2], m-1);            // Recursive call to Draw part2
koch(x[2], y[2], x[3], y[3], m-1);            // Recursive call to Draw part3
koch(x[3], y[3], x[4], y[4], m-1);          // Recursive call to Draw part4
}
else
```

```
{
glBegin(GL_LINES);
glVertex2d(offx + x[0],650-(offy + y[0]));
glVertex2d(offx + x[1],650-(offy + y[1]));
glEnd();
glBegin(GL_LINES);
glVertex2d(offx + x[1],650-(offy + y[1]));
glVertex2d(offx + x[2],650-(offy + y[2]));
glEnd();
glBegin(GL_LINES);
glVertex2d(offx + x[2],650-(offy + y[2]));
glVertex2d(offx + x[3],650-(offy + y[3]));
glEnd();
glBegin(GL_LINES);
glVertex2d(offx + x[3],650-(offy + y[3]));
glVertex2d(offx + x[4],660-(offy + y[4]));
glEnd();
}
}
void display(void)
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
koch(x1, y1, x2, y2, n);
glFlush();              // send all output to display
}
void myinit() {
glClearColor(0.0, 0.0, 0.0, 1.0);             // set background as black
glColor3f(1.0, 1.0, 0.0);                     // Draw in Yellow
glMatrixMode(GL_PROJECTION);        // Establish the coordinate system
glLoadIdentity();
gluOrtho2D(0.0, 650.0, 0.0, 650.0);
}
int main(int argc, char **argv)
{
/* Initialise graphics mode
-----------------------------------*/
cout<<"\n Enter the level of curve generation : ";
cin>>n;
glutInit(&argc, argv);        // Initialize the toolkit
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);       // Set display mode
glutInitWindowSize(650, 650);                     // Set window size
glutInitWindowPosition(0,0);                // Set window position on the screen
```

```
// Open the screen window
glutCreateWindow("Koch Curve");
glutDisplayFunc(display); // Register redraw function
myinit();
glutMainLoop();
return 0;
}
```

# Assignment No:- 8

**Title:- Implement animation principles for any object.**

## Kite Animation

```
#include <GL/glut.h>
#include<math.h>
GLsizei wh=500,ww=500;
void myinit()
{
glClearColor(1.0,1.0,1.0,0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0,(GLdouble)ww,0.0,(GLdouble)wh);
glMatrixMode(GL_MODELVIEW);
//glPointSize(4);
}
void drawkite(float x,float y)
{
glColor3f(1.0,1.0,0.0);
glBegin(GL_POLYGON);
glVertex2f(x-20,y+10);
glVertex2f(x-20,y-40);
glVertex2f(x+20,y-10);
glVertex2f(x+20,y+40);
glEnd();
glColor3f(0.0,0.0,0.0);
glBegin(GL_LINES);
glVertex2f(x-20,y+10);
glVertex2f(x+20,y-10);
glVertex2f(x-20,y-40);
glVertex2f(x+20,y+40);
glEnd();
glFlush();
}
void drawstring(float x,float y)
{
glColor3f(1.0,0.0,1.0);
glBegin(GL_LINES);
glVertex2f(x-20,y-40);
glVertex2f(x-30,y-80);
glVertex2f(x-30,y-80);
glVertex2f(x-50,y-120);
glVertex2f(x-50,y-120);
```

```
glVertex2f(x-80,y-150);
glEnd();
glFlush();
}
void delay()
{
int i,j,r;
for(i=0;i<1000;i++)
for(j=0;j<60000;j++)
r=i*j*10;
}
void myDisplay()
{
glClear(GL_COLOR_BUFFER_BIT);
float i=50.0,j=50.0;
while(j<=450.0)
{
i=50.0;
while(i<400.0)
{
drawkite(i,j);
drawstring(i,j);
glClear(GL_COLOR_BUFFER_BIT);
delay();
i=i+5.0;
j=j+1.0;
}
while(i>100.0)
{
drawkite(i,j);
drawstring(i,j);
glClear(GL_COLOR_BUFFER_BIT);
delay();
i=i-5.0;
j=j+1.0; }
//j=j+30.0;
} }
int main(int argc, char** argv) {
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(50,50);
glutCreateWindow("Click");
```

```
glutDisplayFunc(myDisplay);
myinit();
glutMainLoop();
}
```

## Car Animation

```
#include <GL/glut.h>
float rt = 0.0f;
void init(int Width, int Height)
{
 glClearColor(1.1, 1.1, 1.1, 1.1);
 glMatrixMode(GL_PROJECTION);
 gluPerspective(45.0f,(GLfloat)Width/(GLfloat)Height,0.1f,50.0f);
glMatrixMode(GL_MODELVIEW);
}
float ballX = -0.5f;
float ballY = 0.0f;
float ballZ = 0.0f;
void Draw()
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
 glTranslatef(rt,0.0f,-6.0f);
 glBegin(GL_POLYGON);
 glColor3f(0.0,0.0,0.0);
 glVertex3f(-1.0f, 1.0f, 0.0f);
 glVertex3f(0.4f, 1.0f, 0.0f);
 glVertex3f(1.0f, 0.4f, 0.0f);
 glColor3f(0.0,0.0,0.0);
 glVertex3f( 1.0f,0.0f, 0.0f);
 glColor3f(0.0,0.0,0.0);
 glVertex3f(-1.0f,0.0f, 0.0f);
 glEnd();
 glColor3f(0.0, 0.0, 0.0);
 glTranslatef(ballX,ballY,ballZ);
 glutSolidSphere (0.3, 20, 20);
 glTranslatef(ballX+1.5,ballY,ballZ);
 glutSolidSphere (0.3, 20, 20);
 rt+=0.005f;
 if(rt>2)
 rt=-2.0f;
 glutSwapBuffers();
}
int main(int argc, char **argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE );
 glutInitWindowSize(640, 480);
 glutInitWindowPosition(0, 0);
 glutCreateWindow("Moving Car");
 glutDisplayFunc(Draw);
 glutIdleFunc(Draw);
 init(640,480);
 glutMainLoop();
 return 0;
}
```