UPES

Computer Graphics Lab

B.Tech CSE – Artificial Intelligence & Machine Learning

Submitted By:	***Submitted To:***
Rahul Dhanola	*Dr Niharika Singh*
***Sap ID**: 500075154*	***Batch**: IV*
***Roll No**: R177219139*	***Semester**: V*

EXPERIMENT – 01

EXPERIMENT: INTRODUCTION TO OPENGL:

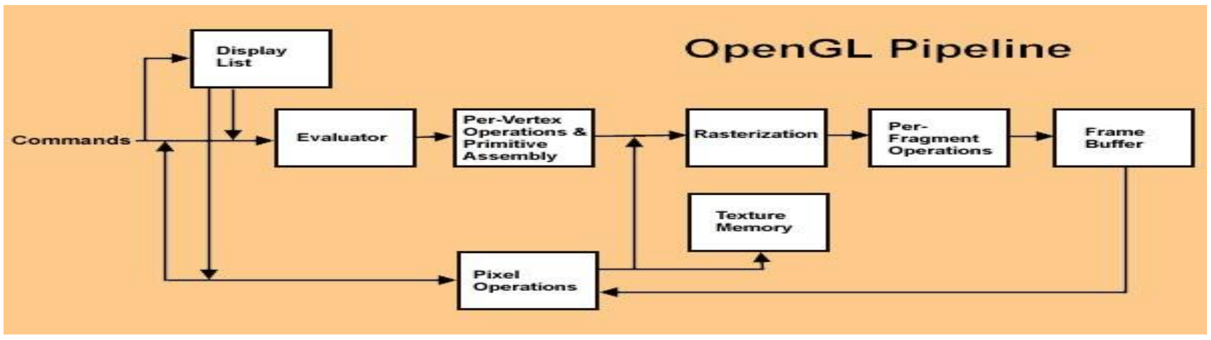
[VIRTUAL LAB ENVIRONMENT SETUP]

- ***What Is OpenGL?***

- ***Open Graphics Library (OpenGL)*** *Is A Cross-Language (Language Independent), Cross-Platform (Platform-Independent) API for Rendering 2D and 3D Vector Graphics (Use of Polygons to Represent Image). OpenGL API Is Designed Mostly in Hardware.*

- ***What Is GLU/GLUT?***
- ***GLU:***
  - ✓ *The OpenGL Utility Library (GLU) Is A Computer Graphics Library for OpenGL.*
  - ✓ *It Consists of a Number of Functions That Use the Base OpenGL Library to Provide Higher-Level Drawing Routines from The More Primitive Routines That OpenGL Provides.*
  - ✓ *It Is Usually Distributed with The Base OpenGL Package*
- ***GLUT:***
  - ✓ *GLUT Is the OpenGL Utility Toolkit, A Window System Independent Toolkit for Writing OpenGL Programs.*
  - ✓ *It Implements a Simple Windowing Application Programming Interface (API) For OpenGL.*
  - ✓ *Glut Makes It Considerably Easier to Learn About and Explore OpenGL Programming.*

- ***What Is OpenGL Architecture?***
- *The OpenGL Architecture Is Structured as A State-Based Pipeline. Below Is a Simplified Diagram of This Pipeline. Commands Enter the Pipeline from The Left.*
- *Commands May Either Be Accumulated in Display Lists, Or Processed Immediately Through the Pipeline. Display Lists Allow for Greater Optimization and Command Reuse, But Not All Commands Can Be Put in Display Lists.*
- *The First Stage in The Pipeline Is the Evaluator. This Stage Effectively Takes Any Polynomial Evaluator Commands and Evaluates Them into Their Corresponding Vertex and Attribute Commands.*



- ***The Second Stage Is the Per-Vertex Operations, Including Transformations, Lighting, Primitive Assembly, Clipping, Projection, And Viewport Mapping.***
- ***The Third Stage Is Rasterization. This Stage Produces Fragments, Which Are Series of Framebuffer Addresses and Values, From The Viewport-Mapped Primitives as Well as Bitmaps and Pixel Rectangles.***
- ***The Fourth Stage Is the Per-Fragment Operations. Before Fragments Go to The Framebuffer, They May Be Subjected to A Series of Conditional Tests and Modifications, such as Blending or Z-Buffering.***
- ***Parts of The Framebuffer May Be Fed Back into The Pipeline as Pixel Rectangles. Texture Memory May Be Used in The Rasterization Process When Texture Mapping Is Enabled.***

❖ *First OpenGL Program: Initializes A Window of Green Colour?*

```
#include<stdio.h>

#include<GL/glut.h>

#include<GL/glu.h>

#include<math.h>

#define pi 3.142857


// function to initialize


void myInit (void) {


 // making background color black as first


 // 3 arguments all are 0.0

 glClearColor(0.0, 1.0, 0.0, 0.0);


 // making picture color green (in RGB mode), as middle argument is 1.0


 glColor3f(0.0, 1.0, 0.0);


 // breadth of picture boundary is 1 pixel
 glPointSize (1.0);

 glMatrixMode(GL_PROJECTION);

 glLoadIdentity();

  // setting window dimension in X- and Y- direction

 gluOrtho2D (-780, 780, -420, 420);
}

void display (void) {

 glClear(GL_COLOR_BUFFER_BIT);

 glBegin(GL_POINTS);

 float x, y;

glVertex2i (x, y);

glEnd ();

 glFlush ();
}
int main (int argc, char** argv) {
 glutInit (&argc, argv);

 glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

 // giving window size in X- and Y- direction

 glutInitWindowSize (1366, 768);

 glutInitWindowPosition (0, 0);

 // Giving name to window

 glutCreateWindow ("Window Drawing");

 myInit ();

 glutDisplayFunc(display);
```

glutMainLoop ();

}

EXPERIMENT – 02

EXPERIMENT: Drawing a line

[Usage of Open GL on Linux Environment for Virtual Environment]

- ***take the input from user for all the three scenarios I.E. Value of (x1, y1) and (x2, y2).***

- ***draw a line using equation of line y=m*x+c.***

*// Draw a line using equation of line Y=m*X+C.*

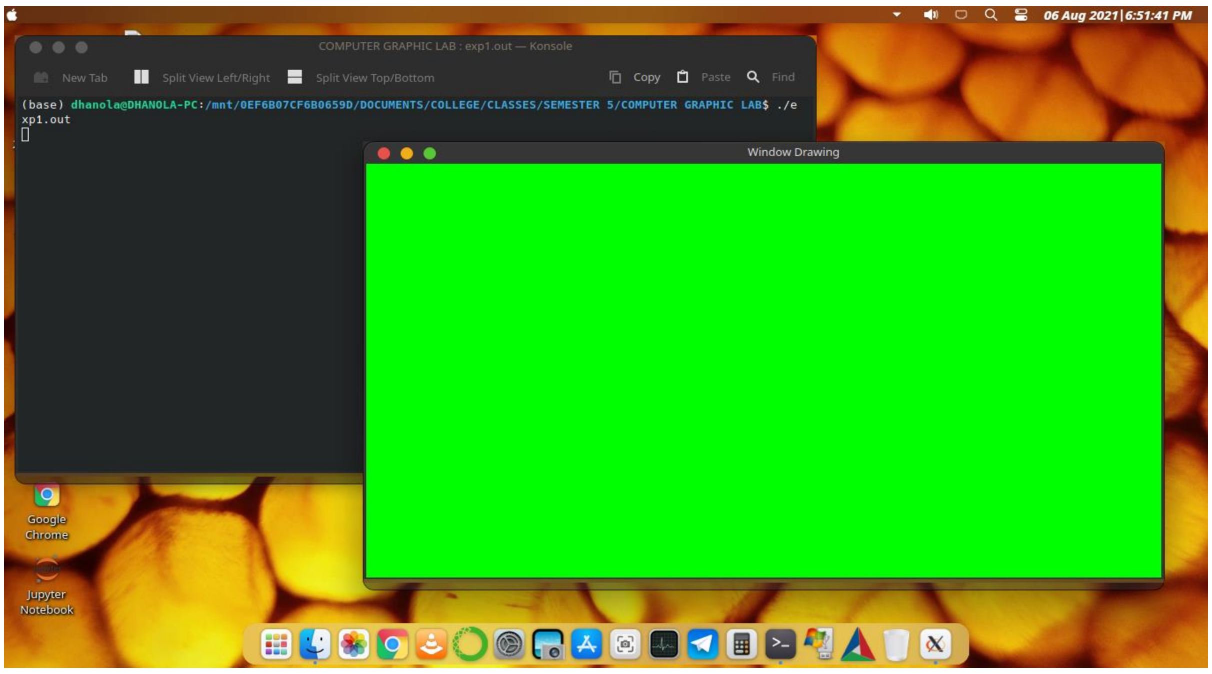
#include<stdio.h>

#include<GL/glut.h>

#include<GL/gl.h>

#include<GL/freeglut.h>

#include<math.h>



```
(base) dhanola@DHANOLA-PC:/mnt/0EF6B07CF6B0659D/DOCUMENTS/COLLEGE/CLASSES/SEMESTER 5/COMPUTER GRAPHIC LAB$ ./e
xp1.out
```

void init ();

void display ();

float x_1, y_1, x_2, y_2, start, end, y, m, c;

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

{

```
  printf ("Window size - 500x500 i.e. range of x and y is 0 -> 500\n");
  printf ("Enter the two end-points of the line: -\n");
  printf ("x1: ");

  scanf ("%f", &x_1);
   printf ("y1: ");
  scanf ("%f", &y_1);
  printf ("x2: ");
  scanf ("%f", &x_2);
  printf ("y2: ");
  scanf ("%f", &y_2);

 glutInit (&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutCreateWindow ("Line using y = Mx + C");

  init ();
glutDisplayFunc(display);
glutMainLoop ();
return 0;
}

void init ()
{
glClearColor (0.8, 0, 0, 1);
glMatrixMode(GL_PROJECTION);
gluOrtho2D (0, 500, 0, 500);
}

void display ()
{
glClear(GL_COLOR_BUFFER_BIT);
glColor3f (0, 0, 1);
glPointSize (5);
glBegin(GL_POINTS);
if((int)x_1 == (int)x_2) //Vertical line, the slope is undefined
  {
    start = y_1 < y_2? y_1: y_2;
    end = y_1 < y_2? y_2: y_1;
    while (start <= end)
    {
      glVertex2f((int)x_1, (int)start);
      start++;
    }
  }

  else // if slope < 1 || slope == 1 || slope > 1
  {
    m = (y_2 - y_1)/ (x_2 - x_1);
    c = y_1 - m * x_1;
    start = x_1 < x_2? x_1: x_2;
    end = x_1 < x_2? x_2: x_1;

    while (start <= end)
    {
      y = m * start + c;
      y = round(y);
      glVertex2f((int)start, (int)y);
      start++;
    }
  }

glEnd ();
glFlush ();
 }
```

- ***draw a line using dda algorithm for slope m<1 and m>1.***

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

float x1, x2, y1, y2;



void display(void)
{
float dy, dx, step, x, y, k, Xin, Yin;
dx=x2-x1;
dy=y2-y1;
if(abs(dx)> abs(dy))

{

step = abs(dx);

}

else


step = abs(dy);



Xin = dx/step;


Yin = dy/step;


x= x1;


y=y1;


glBegin(GL_POINTS);


glVertex2i (x, y);


glEnd ();


for (k=1; k<=step; k++)

{

x= x + Xin;

y= y + Yin;


glBegin(GL_POINTS);


glVertex2i (x, y);


glEnd ();

}



glFlush ();

}


void init(void)
```

```
{
glClearColor (0.7,0.7,0.7,0.7);
glMatrixMode(GL_PROJECTION);
glLoadIdentity ();
gluOrtho2D (-100,100, -100,100);
}

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

printf ("Enter the value of x1: ");

scanf ("%f", &x1);

printf ("Enter the value of y1: ");

scanf ("%f", &y1);

printf ("Enter the value of x2: ");

scanf ("%f", &x2);

printf ("Enter the value of y2: ");

scanf ("%f", &y2);



glutInit (&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (100,100);
glutCreateWindow ("DDA Line Algo");
init ();
glutDisplayFunc(display);
glutMainLoop ();

return 0;

}
```

- ***draw a line using bresenham algorithm for slope m<1 and m>1.***

```
// Bresenham's Line Drawing
```

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

int x1, y1, x2, y2;

void myInit () {

glClear(GL_COLOR_BUFFER_BIT);
glClearColor (0.0, 0.0, 0.0, 1.0);
glMatrixMode(GL_PROJECTION);
gluOrtho2D (0, 500, 0, 500);
}

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

void draw_line (int x1, int x2, int y1, int y2) {
int dx, dy, i, e;
int incx, incy, inc1, inc2;
int x, y;
 dx = x2-x1;
dy = y2-y1;

if (dx < 0) dx = -dx;
if (dy < 0) dy = -dy;
incx = 1;

if (x2 < x1) incx = -1;
incy = 1;
if (y2 < y1) incy = -1;
x = x1; y = y1;
if (dx > dy) {
draw_pixel (x, y);
e = 2 * dy-dx;
inc1 = 2*(dy-dx);
inc2 = 2*dy;
```

```
for (i=0; i<dx; i++) {
if (e >= 0) {
y += incy;
e += inc1;
}

else
e += inc2;
x += incx;
draw_pixel (x, y);
}}

 else {
draw_pixel (x, y);
e = 2*dx-dy;
inc1 = 2*(dx-dy);
inc2 = 2*dx;
for (i=0; i<dy; i++) {
if (e >= 0) {
x += incx;
e += inc1;
}
Else {
e += inc2;
y += incy;
draw_pixel (x, y);
}
}
}

void myDisplay () {
draw_line (x1, x2, y1, y2);
glFlush ();
}

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

printf (“Enter (x1, y1, x2, y2) \n");
scanf ("%d %d %d %d", &x1, &y1, &x2, &y2);
```

```
glutInit (&argc, argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize (500, 500);
glutInitWindowPosition (0, 0);

glutCreateWindow ("Bresenham's Line Drawing");
myInit ();
glutDisplayFunc(myDisplay);
glutMainLoop ();
 return 0;
}
```

<u>EXPERIMENT – 03</u>

<u>EXPERIMENT: Drawing a Circle and an Ellipse</u>

[Done on OpenGL Environment]

- ***Take the value of radius, major axis and minor axis as input from the user.***
- ***Draw the circle with the help of polar equations***

```
// C program to demonstrate circle
// drawing using polar equation

#include <GL/glut.h>
#include <math.h>
#include <stdio.h>
#include <stdlib.h>
float xo, yo, rad;

// Function to display the circle
void display ()
{
glClear(GL_COLOR_BUFFER_BIT);

// Color of printing object
glColor3f (1, 1, 1);

float angle;

// Start to drawing the circle
glBegin(GL_POLYGON);
 for (int i = 0; i < 100; i++) {

// Change the angle
angle = i * 2 * (M_PI / 100);
glVertex2f (xo + (cos(angle) * rad),
yo + (sin(angle) * rad));
}

glEnd ();
// Its empties all the buffer
// causing the issue
glFlush ();
}

// Driver Code
int main (int argc, char** argv)
{
glutInit (&argc, argv);
printf ("Enter x, y, radius resp. ");
scanf ("%f %f %f", &xo, &yo, &rad);
 glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

// Assigning the size of window
glutInitWindowSize (500, 500);

// Assign the position of window to be appeared
glutInitWindowPosition (200, 200);

// Defining the heading of the window
glutCreateWindow("window");

// Backgronnd Color
glClearColor (0, 1, 0, 1);

// limit of the coordinate points
gluOrtho2D (-500, 500, -500, 500);

// Calling the function
glutDisplayFunc(display);
 glutMainLoop ();
return 0;
}
```

❖ *Draw the circle with the help of mid-point method.*

```
// Draw the circle with the help of mid-point method.

#include<stdio.h>
#include<GL/glut.h>
#include<GL/gl.h>
#include<GL/freeglut.h>
```

```
void init ();
void display ();
void drawCircle (int, int, int, int);
float x, y, cx, cy, r, h;

int main (int argc, char** argv)
{
  printf ("Window size - 500x500 i.e. range of x and y is 0 -> 500\n");
  printf ("Enter the co-ordinates of center: -\n");
   printf ("x: ");
  scanf ("%f", &cx);

  printf ("y: ");
  scanf ("%f", &cy);
  printf ("Enter the radius of the circle: ");
  scanf ("%f", &r);

  glutInit (&argc, argv);
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize (500, 500);
glutCreateWindow ("Circle using Mid-Point algorithm");

  init ();
glutDisplayFunc(display);
glutMainLoop ();

return 0;
}

void init ()
{
  glClearColor (0.8, 0, 0, 1);
glMatrixMode(GL_PROJECTION);
gluOrtho2D (0, 500, 0, 500);
}

void display ()
{
glClear(GL_COLOR_BUFFER_BIT);
 glColor3f (0, 1, 0);
glPointSize (5);
glBegin(GL_POINTS);

  //Mid-point algorithm
  x = 0;
  y = r;

  drawCircle (cx, cy, x, y);
   h = 1 - r;

  while (x < y)
  {
    if (h < 0)
       h += 2*x + 3;
    else
    {
      h += 2*(x - y) + 5;
      y--;
    }
    x++;
    drawCircle (cx, cy, x, y);
  }

glEnd ();
glFlush ();
}

void drawCircle (int h, int k, int x, int y)
{
  glVertex2f (h+x, k+y);
  glVertex2f (h-x, k+y);
  glVertex2f (h+x, k-y);
  glVertex2f (h-x, k-y);
  glVertex2f (h+y, k+x);
  glVertex2f (h-y, k+x);
  glVertex2f (h+y, k-x);
  glVertex2f (h-y, k-x);
}
```

❖ ***Draw the Ellipse with the mid-point method.***

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

```
#include<iostream>

using namespace std;

int rx, ry; //semi-Major axis & semi Minor Axis
int xCenter, yCenter; //center of ellipse

void myinit(void)
{
glClearColor (1.0,1.0,1.0,0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity ();
gluOrtho2D (0.0,640.0,0.0,480.0);
}
void setPixel (GLint x, GLint y)
{
  glBegin(GL_POINTS);
 glVertex2i (x, y);
  glEnd ();
}

void ellipseMidPoint ()
{
int xCenter = 250;
int yCenter = 300;
int rx = 200;
int ry = 200;

 //plotting for 1st region of 1st quardant and the slope will be < -1
 //----------------------Region-1-----------------------//
   float x = 0;
   float y = ry;// (0, ry) ---
   float p1 = ry * ry - (rx * rx) * ry + (rx * rx) * (0.25);
   //slope
   float dx = 2 * (ry * ry) * x;
   float dy = 2 * (rx * rx) * y;
   while (dx < dy)
   {
      //plot (x, y)
       setPixel (xCenter + x, yCenter+y);
       setPixel (xCenter - x, yCenter + y);
```

```
     setPixel (xCenter + x, yCenter - y);
     setPixel (xCenter - x, yCenter - y);

    if (p1 < 0)
    {
       x = x + 1;
       dx = 2 * (ry * ry) * x;
       p1 = p1 + dx + (ry * ry);
    }
    else
    {
       x = x + 1;
       y = y - 1;
       dx = 2 * (ry * ry) * x;
       dy = 2 * (rx * rx) * y;
        p1 = p1 + dx - dy + (ry * ry);
    }
}
//ploting for 2nd region of 1st quardant and the slope will be > -1
 //----------------------Region-2------------------------//
float p2 = (ry * ry) * (x + 0.5) * (x + 0.5) + (rx * rx) * (y - 1) * (y - 1) - (rx * rx) * (ry * ry);

while (y > 0)
{
    //plot (x, y)
     setPixel (xCenter + x, yCenter+y);
     setPixel (xCenter - x, yCenter + y);
     setPixel (xCenter + x, yCenter - y);
     setPixel (xCenter - x, yCenter - y); //glEnd ();
    if (p2 > 0)
    {
       x = x;
       y = y - 1;
       dy = 2 * (rx * rx) * y;
       //dy = 2 * rx * rx *y;
       p2 = p2 - dy + (rx * rx);
    }
    else
    {
       x = x + 1;
       y = y - 1;
```

```
          dy = dy - 2 * (rx * rx);

          dx = dx + 2 * (ry * ry);

          p2 = p2 + dx -

          dy + (rx * rx);

      }

}

 }

void display ()

{

    glClear(GL_COLOR_BUFFER_BIT); // clear the screen

    glColor3f (1.0,0.0,0.0); // red foreground

    glPointSize (2.0); // size of points to be drawin (in pixel)


    //establish a coordinate system for the image

     ellipseMidPoint ();

    glFlush (); // send all output to the display

}


int main (int argc, char** argv)

{

   cout<<"\n\nEnter Center of Ellipse \n\n";

   cout<<"\n x = ";

   cin>>xCenter;

    cout<<"\n y = ";

   cin>>yCenter;

    cout<<" Enter a Semi Major Axix: ";

   cin>>rx;

   cout<<" \nEnter a Semi Minor Axis: ";

   cin>>ry;


glutInit (&argc, argv);

glutInitWindowSize (640,480); // set the size of the window

glutInitWindowPosition (10,10); // the position of the top-left of window

glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);

glutCreateWindow ("Midpoint Ellipse Drawing Algorithm");


myinit ();


glutDisplayFunc(display); // set the gl display callback function

glutMainLoop (); // enter the GL event loop
```

```
return 0;
 }
```

EXPERIMENT – 04

EXPERIMENT: Filling –Area

[Small Project will be given for demonstration]

- ***Take the value of seed point, intensity of new color as input from user.***

- ***WAP to fill the polygon using scan lines.***

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

int le [500], re [500], flag=0, m;

void init () {
gluOrtho2D (0, 500, 0, 500);
}

void edge (int x0, int y0, int x1, int y1)
{
if (y1<y0)
{
int tmp;
tmp = y1;
y1 = y0;
y0 = tmp;
tmp = x1;
x1 = x0;
x0 = tmp;
}
int x = x0;
m = (y1 - y0) / (x1 - x0);
for (int i = y0; i<y1; i++)
{
if (x<le[i])
le[i] = x;
if (x>re[i])
re[i] = x;
x += (1 / m);
}
}
void display ()
{
glClearColor (1, 1, 1, 1);
glClear(GL_COLOR_BUFFER_BIT);

glColor3f (0, 0, 1);
glBegin(GL_LINE_LOOP);
glVertex2f (200, 100);
glVertex2f (100, 200);
glVertex2f (200, 300);
glVertex2f (300, 200);
glEnd ();

for (int i = 0; i<500; i++)
{
le[i] = 500;
re[i] = 0;
}
edge (200, 100, 100, 200);
edge (100, 200, 200, 300);
edge (200, 300, 300, 200);
edge (300, 200, 200, 100);

if (flag == 1)
{
for (int i = 0; i < 500; i++)
{
if (le[i] < re[i])
{
for (int j = le[i]; j < re[i]; j++)
{
glColor3f (1, 0, 0);
glBegin(GL_POINTS);
glVertex2f (j, i);
glEnd ();
}  } }
}
glFlush ();
}
```

```
void ScanMenu (int id) {
if (id == 1)
flag = 1;
else if (id == 2)
flag = 0;
else
{
 exit (0);
}
glutPostRedisplay ();
}
 int main (int argc, char **argv)
{
glutInit (&argc, argv);
glutInitWindowPosition (100, 100);
glutInitWindowSize (500, 500);
glutCreateWindow ("scan line");
init ();

glutDisplayFunc(display);
glutCreateMenu(ScanMenu);
glutAddMenuEntry ("scanfill", 1);
glutAddMenuEntry ("clear", 2);
glutAddMenuEntry ("exit", 3);
glutAttachMenu(GLUT_RIGHT_BUTTON);
glutMainLoop ();
return 0;
}
```

- ***WAP to fill a region using boundary fill algorithm using 4 or 8 connected approaches.***

```
//Boundary fill algorithm
#include<stdio.h>
#include<GL/glut.h>
#include<GL/freeglut.h>

int xmin, ymin, xmax, ymax; //Polygon boundaries
float FillColor [3] = {1.0, 0.0, 0.0}; //Color to be filled - red
float BorderColor [3] = {0.0, 0.0, 0.0}; // Border color of polygon - black

void setPixel (int x, int y)
{
```

```
glBegin(GL_POINTS);
glColor3fv(FillColor);
glVertex2i (x, y);
glEnd ();
glFlush ();
}
 void display ()
{
glClear(GL_COLOR_BUFFER_BIT);

 //Drawing polygon
 glColor3fv(BorderColor);
 glLineWidth (6);
 glBegin(GL_LINES);
glVertex2i (xmin, ymin);
glVertex2i (xmin, ymax);
glEnd ();

glBegin(GL_LINES);
glVertex2i (xmax, ymin);
glVertex2i (xmax, ymax);
glEnd ();

glBegin(GL_LINES);
glVertex2i (xmin, ymin);
glVertex2i (xmax, ymin);
   glEnd ();
   glBegin(GL_LINES);
      glVertex2i (xmin, ymax);
      glVertex2i (xmax, ymax);

   glEnd ();
   glFlush ();
}
void BoundaryFill (int x, int y)
{
   float CurrentColor [3];
   glReadPixels (x, y, 1.0, 1.0, GL_RGB, GL_FLOAT, CurrentColor);

 if CurrentColor! = BorderColor and CurrentColor! = FillColor
   if ((CurrentColor [0]! = BorderColor [0] && (CurrentColor [1])! = BorderColor [1] &&
     (CurrentColor [2])! = BorderColor [2]) && (CurrentColor [0]! = FillColor [0] &&
     (CurrentColor [1])! = FillColor [1] && (CurrentColor [2])! = FillColor [2]))
   {
      setPixel (x, y);
      BoundaryFill (x+1, y);
      BoundaryFill (x-1, y);
      BoundaryFill (x, y+1);
      BoundaryFill (x, y-1);
   }
}
void mouse (int btn, int state, int x, int y)
{
   if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
   {
      printf ("%d, %d\n", x, y);
      BoundaryFill (x, 500-y);
   }
}
void init ()
{
   glClearColor (0.101, 1.0, 0.980, 1.0); //Background color - cyan
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D (0, 500, 0, 500);
}
 int main (int argc, char** argv)
{
   printf ("Window size - 500x500 i.e. range of x and y is 0 -> 500\n");
   printf ("\nEnter polygon boundaries: -\n");
   printf ("Enter xmin: ");
   scanf ("%d", &xmin);
   printf ("Enter ymin: ");
   scanf ("%d", &ymin);
   printf ("Enter xmax: ");
   scanf ("%d", &xmax);
   printf ("Enter ymax: ");
   scanf ("%d", &ymax);

   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
   glutCreateWindow ("Boundary-Fill Algorithm");
   init ();
   glutDisplayFunc(display);
   glutMouseFunc(mouse);
   glutMainLoop ();
   return 0;
}
```

- **WAP to fill a region using flood fill algorithm using 4 or 8 connected approaches.**

```
//Flood fill algorithm
#include<stdio.h>
#include<GL/glut.h>
#include<GL/freeglut.h>

int xmin, ymin, xmax, ymax; //Polygon boundaries
float OldColor [3] = {0.0, 0.0, 0.0}; //Color of the polygon - black
float NewColor [3] = {1.0, 0.0, 0.0}; //Color to be filled - red

void setPixel (int x, int y)
{
   glBegin(GL_POINTS);
      glColor3fv(NewColor);
      glVertex2i (x, y);
   glEnd ();
   glFlush ();}
void display ()
{
   glClear(GL_COLOR_BUFFER_BIT);

   //Drawing polygon
   glColor3fv(OldColor);
   glBegin(GL_POLYGON);
      glVertex2i (xmin, ymin);
      glVertex2i (xmin, ymax);
      glVertex2i (xmax, ymax);
      glVertex2i (xmax, ymin);

   glEnd ();
   glFlush ();}

void FloodFill (int x, int y)
```

```
{
   float CurrentColor [3];
   glReadPixels (x, y, 1.0, 1.0, GL_RGB, GL_FLOAT, CurrentColor);

   // if CurrentColor == OldColor
   if (CurrentColor [0] == OldColor [0] && (CurrentColor [1]) == OldColor [1] && (CurrentColor [2]) == OldColor [2])
   {
      setPixel (x, y);
      FloodFill (x+1, y);
      FloodFill (x-1, y);
      FloodFill (x, y+1);
      FloodFill (x, y-1);
   }
}
void mouse (int btn, int state, int x, int y)
{
   if (btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN)
 FloodFill (x, 500-y);
}
 void init ()
{
   glClearColor (0.101, 1.0, 0.980, 1.0); //Background color - cyan
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D (0, 500, 0, 500);
}
 int main (int argc, char** argv)
{
   printf ("Window size - 500x500 i.e. range of x and y is 0 -> 500\n");
   printf ("\nEnter polygon boundaries: -\n");
   printf ("Enter xmin: ");
   scanf ("%d", &xmin);
   printf ("Enter ymin: ");
   scanf ("%d", &ymin);
   printf ("Enter xmax: ");
   scanf ("%d", &xmax);
   printf ("Enter ymax: ");
   scanf ("%d", &ymax);

   glutInit (&argc, argv);
   glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize (500, 500);
```

```
    glutCreateWindow ("Flood-Fill Algorithm");


    init ();
    glutDisplayFunc(display);
    glutMouseFunc(mouse);
    glutMainLoop ();
    return 0;
}
```

EXPERIMENT – 05 & 06

EXPERIMENT: Viewing and Clipping

[Geographical Animation for demonstration]

❖ ***Write an interactive program for line clipping using Cohen Sutherland line clipping algorithm.***

```
#include<stdio.h>
#include<GL/glut.h>
#include<GL/freeglut.h>
#include<math.h>

#define TOP 8     // TBRL - 1000
#define BOTTOM 4 // TBRL - 0100
#define RIGHT 2   // TBRL - 0010
#define LEFT 1    // TBRL - 0001

int xmin, ymin, xmax, ymax; // Windows boundaries
int x_0, y_0, x_1, y_1; // Line end-points

int ComputeTBRLCode (int x, int y)
{
  int code = 0; // Calculating TBLR code for an end-point
  if (y > ymax) //above the window
    code = code | TOP;
  if (y < ymin) //below the window
    code = code | BOTTOM;
  if (x > xmax) //to the right of window
    code = code | RIGHT;
  if (x < xmin) //to the left of window
    code = code | LEFT;
  return code;
}
 void CohenSutherland ()
{
  int code_0, code_1, code_outside;
  int inside = 0, loop = 0;

  //Calculating TBLR code for end-points
```

```
code_0 = ComputeTBRLCode (x_0, y_0);
code_1 = ComputeTBRLCode (x_1, y_1);

while (! loop)
{
 if (! (code_0 | code_1)) // Check if logical OR is 0 or not
 {
  inside = 1; loop = 1;
 }
 else
  if (code_0 & code_1) // Check if logical AND is 0
   loop = 1;
  else
  {
   int x, y;
    code_outside = code_0? code_0: code_1;

   //Finding intersection point
   if (code_outside & TOP) //point is above the window
   {
    x = x_0 + (x_1 - x_0) * (ymax - y_0) / (y_1 - y_0);
    y = ymax;}
   else if (code_outside & BOTTOM) //point is below the window
   {
    x = x_0 + (x_1 - x_0) * (ymin - y_0) / (y_1 - y_0);
    y = ymin;
   }
   else if (code_outside & RIGHT) //point is to the right of window
   {
    y = y_0 + (y_1 - y_0) * (xmax - x_0) / (x_1 - x_0);
    x = xmax;
   }
   else //point is to the left of the window
   {
    y = y_0 + (y_1 - y_0) * (xmin - x_0) / (x_1 - x_0);
    x = xmin;
   }

   if (code_outside == code_0)
   {
    x_0 = x;
```

```
      y_0 = y;

      code_0 = ComputeTBRLCode (x_0, y_0);

    }

    else

    {

      x_1 = x;

      y_1 = y;

      code_1 = ComputeTBRLCode (x_1, y_1);

    }     }   }

 if(inside) // If line is completely inside or partially inside

{

  //Drawing window

  glColor3f (0, 1, 0);

  glLineWidth (10);

  glBegin(GL_LINE_LOOP);

    glVertex2f (xmin, ymin);

    glVertex2f (xmax, ymin);

    glVertex2f (xmax, ymax);

    glVertex2f (xmin, ymax);

  glEnd ();

  glFlush ();


  //Drawing line

  glColor3f (0.211, 0.211, 0.211);

  glLineWidth (10);

  glBegin(GL_LINES);

    glVertex2d (x_0, y_0);

    glVertex2d (x_1, y_1);

  glEnd ();

  glFlush ();

 }

 else //line is completely outside

 {

  //Drawing window

  glColor3f (0, 1, 0);

  glLineWidth (10);

  glBegin(GL_LINE_LOOP);

    glVertex2f (xmin, ymin);

    glVertex2f (xmax, ymin);

    glVertex2f (xmax, ymax);
```

```
        glVertex2f (xmin, ymax);
      glEnd ();
      glFlush ();
    }
}

void Keyboard (unsigned char key, int x, int y)
{
if (key == 'c')
  {
    //Drawing again with clipped part
    glClearColor (0.8, 0, 0, 1);
    glClear(GL_COLOR_BUFFER_BIT);
  CohenSutherland ();
  } }
void display ()
{
  //Drawing without clipping
  glClear(GL_COLOR_BUFFER_BIT);

  //Drawing window
  glColor3f (0, 1, 0);
  glLineWidth (10);
  glBegin(GL_LINE_LOOP);
    glVertex2f (xmin, ymin);
    glVertex2f (xmax, ymin);
    glVertex2f (xmax, ymax);
    glVertex2f (xmin, ymax);
  glEnd ();

  //Drawing line
  glColor3f (0.211, 0.211, 0.211);
  glLineWidth (10);
  glBegin(GL_LINES);
    glVertex2d (x_0, y_0);
    glVertex2d (x_1, y_1);
  glEnd ();

  glFlush ();}
void init ()
{
   glClearColor (0.8, 0, 0, 1);
   glMatrixMode(GL_PROJECTION);
   gluOrtho2D (0, 500, 0, 500);
}
 int main (int argc, char** argv)
{
  printf ("Window size - 500x500 i.e. range of x and y is 0 -> 500\n");
   printf ("\nEnter window boundaries: -\n");
  printf ("Enter xmin: ");
  scanf ("%d", &xmin);
  printf ("Enter ymin: ");
  scanf ("%d", &ymin);
  printf ("Enter xmax: ");
  scanf ("%d", &xmax);
  printf ("Enter ymax: ");
  scanf ("%d", &ymax);

  printf ("\nEnter end-points of line: -\n");
  printf ("Enter x0: ");
  scanf ("%d", &x_0);
  printf ("Enter y0: ");
  scanf ("%d", &y_0);
  printf ("Enter x1: ");
  scanf ("%d", &x_1);
  printf ("Enter y1: ");
  scanf ("%d", &y_1);

  glutInit (&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize (500, 500);
  glutCreateWindow ("Cohen Sutherland Line Clipping Algorithm");

  init ();
  glutDisplayFunc(display);
  glutKeyboardFunc(Keyboard);
  glutMainLoop ();
   return 0;
}
```

- ***Write an interactive program for line clipping using Liang-Barsky line clipping algorithm.***

```
#include <stdio.h>
```

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


int x1=-80, x2=0, y3=-80, y2=0;

float u1=0, u2=1;

int xmin=-50, ymin=-50, xmax=50, ymax=50;

double p [4], q [4]; // changed from int to double thats it


void init ()

{

   glClearColor (1.0,1.0,1.0,1.0);

   glMatrixMode(GL_PROJECTION);

   gluOrtho2D (-320,320, -240,240);

}

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

{

   int dx=x2-x1, dy=y2-y1, i;

   double t;

   p [0] =-dx; q [0] =x1-xmin;

   p [1] =dx; q [1] =xmax-x1;

   p [2] =-dy;q[2]=y1-ymin;

   p[3]=dy;q[3]=ymax-y1;


   for(i=0;i<4;i++)  {

      if(p[i]==0 && q[i]<0)

         return;

      if(p[i]<0)

      {

         t=(q[i])/(p[i]);  // This calculation was returning a zero because both q and p were int

         if(t>u1 && t<u2)

            {u1=t;}

      }

      else if(p[i]>0)

      {

         t=(q[i])/(p[i]);  // This calculation was returning a zero because both q and p were int

         if(t>u1 && t<u2)

            {u2=t;}

      }   }

   if(u1<u2)

   {

      x1=x1+u1*(x2-x1);

      y1=y1+u1*(y2-y1);
```

```
        x2=x1+u2*(x2-x1);
        y2=y1+u2*(y2-y1);
        glBegin(GL_LINES);
           glVertex2i(x1,y1);
           glVertex2i(x2,y2);
        glEnd();
        glFlush();
    } }
void display()
{
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(0.0,0.0,0.0);
    glBegin(GL_LINES);
        glVertex2i(x1,y3);
        glVertex2i(x2,y2);
    glEnd();
    glFlush();
}
void myKey(unsigned char key,int x,int y)
{
glClear(GL_COLOR_BUFFER_BIT);
    if(key=='c')
    {
        glClear(GL_COLOR_BUFFER_BIT);
        glFlush();
        glColor3f(0.0,0.0,0.0);
        glBegin(GL_LINES);
           glVertex2i(-50,-50);
           glVertex2i(-50,50);
           glVertex2i(-50,50);
           glVertex2i(50,50);
           glVertex2i(50,50);
           glVertex2i(50,-50);
           glVertex2i(50,-50);
           glVertex2i(-50,-50);
        glEnd();
        glFlush();
        clip(::x1,y3,x2,y2);
    } }
int main(int argc,char ** argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(640,480);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Clip");
    glutDisplayFunc(display);
    glutKeyboardFunc(myKey);
    init();
    glutMainLoop();
    return 0;
}
```

❖ Write an interactive program for polygon clipping using Sutherland – Hodgeman polygon clipping algorithm.

```
#include<stdio.h>
#include<GL/glut.h>
#include<GL/freeglut.h>
#include<math.h>
#include<float.h>

#define MAX 10

int xmin, ymin, xmax, ymax; // Windows boundaries
int n, result, points[MAX][2]; //Storing end-points of lines of polygon
int aux[MAX][2]; //Auxillary array to store new end-points

int leftClip(int total)
{
int x1, y1, x2, y2, iterator = 0, i;
float m;
for (i = 0; i < total; i++)
{
x1 = points[i][0];
y1 = points[i][1];
x2 = points[(i + 1) % total][0];
y2 = points[(i + 1) % total][1];

if (x2 - x1)
m = (y2 - y1) * 1.0 / (x2 - x1);
else
m = FLT_MAX;
if (x1 < xmin && x2 < xmin)
continue;
if (x1 >= xmin && x2 >= xmin)
{
aux[iterator][0] = x2;
aux[iterator++][1] = y2;
continue;
}
if (x1 >= xmin && x2 < xmin)
{
aux[iterator][0] = xmin;
aux[iterator++][1] = y1 + m * (xmin - x1);
continue;
}
if (x1 < xmin && x2 >= xmin)
{
aux[iterator][0] = xmin;
aux[iterator++][1] = y1 + m * (xmin - x1);
aux[iterator][0] = x2;
aux[iterator++][1] = y2;
} }
for (i = 0; i < iterator; i++)
{
points[i][0] = aux[i][0];
points[i][1] = aux[i][1];
aux[i][0] = 0;
aux[i][1] = 0;
}

if (iterator < total)
while (i < total)
{
points[i][0] = 0;
points[i][1] = 0;
i++; }
return iterator;
}
int topClip(int total)
{
int i, iterator = 0, x1, y1, x2, y2;
float m;
```

```
for (i = 0; i < total; i++)
{
x1 = points[i][0];
y1 = points[i][1];
x2 = points[(i + 1) % total][0];
y2 = points[(i + 1) % total][1];
if (x2 - x1)
m = (y2 - y1) * 1.0 / (x2 - x1);
else
m = FLT_MAX;

if (y1 > ymax && y2 > ymax)
continue;
if (y1 <= ymax && y2 <= ymax)
{
aux[iterator][0] = x2;
aux[iterator++][1] = y2;
continue;
}
if (y1 <= ymax && y2 > ymax)
{
aux[iterator][0] = x1 + (ymax - y1) / m;
aux[iterator++][1] = ymax;
continue;
}
if (y1 > ymax && y2 <= ymax)
{
aux[iterator][0] = x1 + (ymax - y1) / m;
aux[iterator++][1] = ymax;
aux[iterator][0] = x2;
aux[iterator++][1] = y2;
} }
for (i = 0; i < iterator; i++)
{
points[i][0] = aux[i][0];
points[i][1] = aux[i][1];
aux[i][0] = 0;
aux[i][1] = 0;
}

if (iterator < total)
```

```
while (i < total)
{
points[i][0] = 0;
points[i][1] = 0;
i++; }
return iterator;
}
 int rightClip(int total)
{
int i, iterator = 0, x1, y1, x2, y2;
float m;
for (i = 0; i < total; i++)
{
x1 = points[i][0];
y1 = points[i][1];
x2 = points[(i + 1) % total][0];
y2 = points[(i + 1) % total][1];
if (x2 - x1)
m = (y2 - y1) * 1.0 / (x2 - x1);
else
m = FLT_MAX;
if (x1 > xmax && x2 > xmax)
continue;
if (x1 <= xmax && x2 <= xmax)
{
aux[iterator][0] = x2;
aux[iterator++][1] = y2;
continue;
}
if (x1 <= xmax && x2 > xmax)
{
aux[iterator][0] = xmax;
aux[iterator++][1] = y1 + m * (xmax - x1);
continue;
}
if (x1 > xmax && x2 <= xmax)
{
aux[iterator][0] = xmax;
aux[iterator++][1] = y1 + m * (xmax - x1);
aux[iterator][0] = x2;
aux[iterator++][1] = y2;
} }
for (i = 0; i < iterator; i++)
{
points[i][0] = aux[i][0];
points[i][1] = aux[i][1];
aux[i][0] = 0;
aux[i][1] = 0;
}

if (iterator < total)
while (i < total)
{
points[i][0] = 0;
points[i][1] = 0;
i++; }
return iterator;
}
 int bottomClip(int total)
{
int i, iterator = 0, x1, y1, x2, y2;
float m;
for (i = 0; i < total; i++)
{
x1 = points[i][0];
y1 = points[i][1];
x2 = points[(i + 1) % total][0];
y2 = points[(i + 1) % total][1];
if (x2 - x1)
m = (y2 - y1) * 1.0 / (x2 - x1);
else
m = FLT_MAX;

if (y1 < ymin && y2 < ymin)
continue;
if (y1 >= ymin && y2 >= ymin)
{
aux[iterator][0] = x2;
aux[iterator++][1] = y2;
continue;
}
if (y1 >= ymin && y2 < ymin)
```

```
{
aux[iterator][0] = x1 + (ymin - y1) / m;
aux[iterator++][1] = ymin;
continue;
}
if (y1 < ymin && y2 >= ymin)
{
aux[iterator][0] = x1 + (ymin - y1) / m;
aux[iterator++][1] = ymin;
aux[iterator][0] = x2;
aux[iterator++][1] = y2;
} }
 for (i = 0; i < iterator; i++)
{
points[i][0] = aux[i][0];
points[i][1] = aux[i][1];
aux[i][0] = 0;
aux[i][1] = 0;
}

 if (iterator < total)
while (i < total)
{
points[i][0] = 0;
points[i][1] = 0;
i++; }
return iterator;
}
 void Keyboard(unsigned char key, int x, int y)
{
if(key == 'c')  {
   //Drawing again with clipped part
   glClearColor(0.8, 0, 0, 1);
   glClear(GL_COLOR_BUFFER_BIT);

   //Sutherland-Hodgeman Algorithm
   result = leftClip(n);
   result = topClip(result);
   result = rightClip(result);
   result = bottomClip(result);
   //Drawing window again
   glColor3f(0, 1, 0);
   glLineWidth(10);
   glBegin(GL_LINE_LOOP);
     glVertex2f(xmin, ymin);
     glVertex2f(xmin, ymin);
     glVertex2f(xmax, ymin);
     glVertex2f(xmax, ymax);
     glVertex2f(xmin, ymax);
   glEnd();

   //Drawing clipped polygon
   glColor3f(0, 0, 1);
   glLineWidth(10);
   glBegin(GL_LINE_STRIP);
     for(int i = 0; i < result; i++)
       glVertex2d(points[i][0], points[i][1]);
     glVertex2d(points[0][0], points[0][1]);
   glEnd();
    glFlush();
  } }
void display()
{
  //Drawing without clipping
  glClear(GL_COLOR_BUFFER_BIT);

  //Drawing window
  glColor3f(0, 1, 0);
  glLineWidth(10);
  glBegin(GL_LINE_LOOP);
    glVertex2f(xmin, ymin);
    glVertex2f(xmin, ymin);
    glVertex2f(xmax, ymin);
    glVertex2f(xmax, ymax);
    glVertex2f(xmin, ymax);
  glEnd();

  //Drawing polygon
  glColor3f(0, 0, 1);
  glLineWidth(10);
  glBegin(GL_LINE_STRIP);
  for(int i = 0; i < n; i++)
 glVertex2d(points[i][0], points[i][1]);
glVertex2d(points[0][0], points[0][1]);
  glEnd();
  glFlush();
}
 void init()
{
    glClearColor(0.8, 0, 0, 1);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(0, 500, 0, 500);
}
 int main(int argc, char** argv)
{
  printf("Window size - 500x500 i.e. range of x and y is 0 -> 500\n");
   printf("\nEnter window boundaries:-\n");
  printf("Enter xmin: ");
  scanf("%d", &xmin);
  printf("Enter ymin: ");
  scanf("%d", &ymin);
  printf("Enter xmax: ");
  scanf("%d", &xmax);
  printf("Enter ymax: ");
  scanf("%d", &ymax);
   printf("\nEnter number of points: ");
  scanf("%d", &n);

  if(n < 3)
  {
    printf("Number of points should be equal to or greater than 3!\n");
    exit(0);
  }
  else
    printf("Enter points in clock-wise/anti-clock-wise order!\n");

  for(int i = 0; i < n; i++)  {
    printf("Enter point-%d: ", i+1);
    scanf("%d %d", &points[i][0], &points[i][1]);
  }
  glutInit(&argc,argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutCreateWindow("Sutherland Hodgeman Polygon Clipping");

  init();
  glutDisplayFunc(display);
  glutKeyboardFunc(Keyboard);
  glutMainLoop();
  return 0;
}
```

EXPERIMENT – 07 & 08

EXPERIMENT: Basic Two & Three Dimensional Transformations

- ***Write an interactive program for following basic transformation.***

  - ***Translation***
  - ***Rotation***
  - ***Scaling***
  - ***Reflection about axis.***
  - ***Reflection about a line Y=mX+c and aX+bY+c=0.***
  - ***Shear about an edge and about a vertex.***

```
#include<stdio.h>

#include<GL/glut.h>

#include<GL/freeglut.h>

#include<GL/glu.h>

#include<GL/gl.h>

#include<math.h>


int choice, choice1, choice2, choice3, xaux[8], yaux[8], zaux[8], i;

int x[8] = {20, 120, 120, 20, 0, 100, 100, 0};

int y[8] = {70, 70, 20, 20, 50, 50, 0, 0};

int z[8] = {0, 0, 0, 0, 150, 150, 150, 150};


void init()

{

glClearColor(0, 0, 0, 1);

glMatrixMode(GL_PROJECTION);

glOrtho(-500, 500,-500, 500, -500, 500);

}

void display() {

  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);

  glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);

  glEnable(GL_DEPTH_TEST);

glutInitWindowPosition(500,100);

glutInitWindowSize(1000,750);

glutCreateWindow("3D Transformations");

init();
```

```
// Drawing initial polygon
glColor3f(0, 0, 1);
glLineWidth(2.0);

// TOP
glBegin(GL_LINE_LOOP);
    glVertex3f(x[1], y[1], z[1]);
    glVertex3f(x[0], y[0], z[0]);
    glVertex3f(x[4], y[4], z[4]);
    glVertex3f(x[5], y[5], z[5]);
glEnd();

// BOTTOM
glBegin(GL_LINE_LOOP);
    glVertex3f(x[6], y[6], z[6]);
    glVertex3f(x[7], y[7], z[7]);
    glVertex3f(x[3], y[3], z[3]);
    glVertex3f(x[2], y[2], z[2]);
glEnd();

// FRONT FEHG
glBegin(GL_LINE_LOOP);
    glVertex3f(x[5], y[5], z[5]);
    glVertex3f(x[4], y[4], z[4]);
    glVertex3f(x[7], y[7], z[7]);
    glVertex3f(x[6], y[6], z[6]);
glEnd();

// BACK
glBegin(GL_LINE_LOOP);
    glVertex3f(x[2], y[2], z[2]);
    glVertex3f(x[3], y[3], z[3]);
    glVertex3f(x[0], y[0], z[0]);
    glVertex3f(x[1], y[1], z[1]);
glEnd();

// LEFT
glBegin(GL_LINE_LOOP);
    glVertex3f(x[4], y[4], z[4]);
    glVertex3f(x[0], y[0], z[0]);
    glVertex3f(x[3], y[3], z[3]);
    glVertex3f(x[7], y[7], z[7]);
glEnd();

// RIGHT
glBegin(GL_LINE_LOOP);
    glVertex3f(x[1], y[1], z[1]);
    glVertex3f(x[5], y[5], z[5]);
    glVertex3f(x[6], y[6], z[6]);
    glVertex3f(x[2], y[2], z[2]);
glEnd();

// Drawing polygon after transformation
glColor3f(1, 0, 0);
glLineWidth(2.0);
// TOP BAEF
glBegin(GL_LINE_LOOP);
    glVertex3f(xaux[1], yaux[1], zaux[1]);
    glVertex3f(xaux[0], yaux[0], zaux[0]);
    glVertex3f(xaux[4], yaux[4], zaux[4]);
    glVertex3f(xaux[5], yaux[5], zaux[5]);
glEnd();

// BOTTOM
glBegin(GL_LINE_LOOP);
    glVertex3f(xaux[6], yaux[6], zaux[6]);
    glVertex3f(xaux[7], yaux[7], zaux[7]);
    glVertex3f(xaux[3], yaux[3], zaux[3]);
    glVertex3f(xaux[2], yaux[2], zaux[2]);
glEnd();

// FRONT
glBegin(GL_LINE_LOOP);
    glVertex3f(xaux[5], yaux[5], zaux[5]);
    glVertex3f(xaux[4], yaux[4], zaux[4]);
    glVertex3f(xaux[7], yaux[7], zaux[7]);
    glVertex3f(xaux[6], yaux[6], zaux[6]);
glEnd();

// BACK
glBegin(GL_LINE_LOOP);
    glVertex3f(xaux[2], yaux[2], zaux[2]);
```

```
        glVertex3f(xaux[3], yaux[3], zaux[3]);
        glVertex3f(xaux[0], yaux[0], zaux[0]);
        glVertex3f(xaux[1], yaux[1], zaux[1]);
    glEnd();

    // LEFT
    glBegin(GL_LINE_LOOP);
        glVertex3f(xaux[4], yaux[4], zaux[4]);
        glVertex3f(xaux[0], yaux[0], zaux[0]);
        glVertex3f(xaux[3], yaux[3], zaux[3]);
        glVertex3f(xaux[7], yaux[7], zaux[7]);
    glEnd();

    // RIGHT
    glBegin(GL_LINE_LOOP);
        glVertex3f(xaux[1], yaux[1], zaux[1]);
        glVertex3f(xaux[5], yaux[5], zaux[5]);
        glVertex3f(xaux[6], yaux[6], zaux[6]);
        glVertex3f(xaux[2], yaux[2], zaux[2]);
    glEnd();
     glFlush();
    glutMainLoop();
}
 void translation()
{
    printf("\n\nTRANSLATION\n");
    int T[3];
    printf("Enter translation factor (dx dy dz): ");
    scanf("%d%d%d",&T[0],&T[1],&T[2]);
    for(i=0; i<8; i++)
    {
        xaux[i] = x[i] + T[0];
        yaux[i] = y[i] + T[1];
        zaux[i] = z[i] + T[2];
    }
    display();
}
void rotation()
{
    int angle;
```

```
   printf("\n\nOPERATIONS FOR ROTATION\n\n1. About X-AXIS\n2. About Y-AXIS\n3. About Z-AXIS\n\nEnter 
your choice: ");
   scanf("%d",&choice1);
   printf("Enter angle of rotation (in degree): ");
   scanf("%d",&angle);
   switch(choice1)
   {
      case 1:
      {
         printf("\n\nROTATION ABOUT X-AXIS\n");
         for(i=0; i<8; i++)
         {
            xaux[i] = x[i];
            yaux[i] = (y[i] * cos(angle * 3.14/180)) - (z[i] * sin(angle * 3.14/180));
            zaux[i] = (y[i] * sin(angle * 3.14/180)) + (z[i] * cos(angle * 3.14/180));
         }
         display();
      }
      case 2:
      {
         printf("\n\nROTATION ABOUT Y-AXIS\n");
         for(i=0; i<8; i++)
         {
            xaux[i] = (z[i] * sin(angle * 3.14/180)) + (x[i] * cos(angle * 3.14/180));
            yaux[i] = y[i];
            zaux[i] = (y[i] * cos(angle * 3.14/180)) - (x[i] * sin(angle * 3.14/180));
         }
         display();
      }
      case 3:
      {
         printf("\n\nROTATION ABOUT Z-AXIS\n");
         for(i=0; i<8; i++)
         {
            xaux[i] = (x[i] * cos(angle * 3.14/180)) - (y[i] * sin(angle * 3.14/180));
            yaux[i] = (x[i] * sin(angle * 3.14/180)) + (y[i] * cos(angle * 3.14/180));
            zaux[i] = z[i];
         }
         display();
      }
      default: printf("Wrong Choice\n"); break;
   } }

 void scaling()
{
   printf("\n\nSCALING\n");
   int SCALE[3];
   printf("Enter the scaling factor (dx dy dz): ");
   scanf("%d%d%d",&SCALE[0],&SCALE[1],&SCALE[2]);
   for(i=0; i<8; i++)
   {
      xaux[i] = x[i] * SCALE[0];
      yaux[i] = y[i] * SCALE[1];
      zaux[i] = z[i] * SCALE[2];
   }
   display();
}
 void reflection()
{
   printf("\n\nOPERATIONS FOR REFLECTION\n\n1. About XY-Plane\n2. About YZ-Plane\n3. About ZX-
Plane\n\nEnter your choice: ");
   scanf("%d",&choice2);
   switch(choice2)
   {
      case 1:
      {
         printf("\n\nREFLECTION ABOUT XY-PLANE\n");
         for(i=0; i<8; i++)
         {
            xaux[i] = x[i];
            yaux[i] = y[i];
            zaux[i] = -z[i];
         }
         display();
      }
      case 2:
      {
         printf("\n\nREFLECTION ABOUT YZ-PLANE\n");
         for(i=0; i<8; i++)
         {
            xaux[i] = -x[i];
            yaux[i] = y[i];
            zaux[i] = z[i];
        }
        display();
    }
    case 3:
    {
        printf("\n\nREFLECTION ABOUT ZX-PLANE\n");
        for(i=0; i<8; i++)
        {
            xaux[i] = x[i];
            yaux[i] = -y[i];
            zaux[i] = z[i];
        }
        display();
    }
    default: printf("Wrong Choice\n"); break;
  } }
 void shearing()
{
  int SHEAR[3];
  printf("\n\nOPERATIONS FOR SHEARING\n\n1. About X-Axis\n2. About Y-Axis\n3. About Z-Axis\n\nEnter your
choice: ");
  scanf("%d",&choice3);
  printf("Enter the shearing factor (sx sy sz): ");
  scanf("%d%d%d",&SHEAR[0],&SHEAR[1],&SHEAR[2]);

  switch(choice3)
  {
    case 1:
    {
        printf("\n\nSHEARING ABOUT X-AXIS\n");
        for(i=0; i<8; i++)
        {
            xaux[i] = x[i];
            yaux[i] = y[i]  + SHEAR[1] * x[i];
            zaux[i] = z[i]  + SHEAR[2] * x[i];
        }
        display();
    }
    case 2:
    {
           printf("\n\nSHEARING ABOUT Y-AXIS\n");
           for(i=0; i<8; i++)
           {
              xaux[i] = x[i]  + SHEAR[0] * y[i];
              yaux[i] = y[i];
              zaux[i] = z[i]  + SHEAR[2] * y[i];
           }
           display();
        }
        case 3:
        {
           printf("\n\nSHEARING ABOUT Z-AXIS\n");
           for(i=0; i<8; i++)
           {
              xaux[i] = x[i]  + SHEAR[0] * z[i];
              yaux[i] = y[i]  + SHEAR[1] * z[i];
              zaux[i] = z[i];
           }
           display();
        }
        default: printf("Wrong Choice\n"); break;
     } }
  int main(int argc, char** argv)
  {
  glutInit(&argc, argv);

     printf("Before transformation color - Blue\n");
     printf("After transformation color - Red\n");
     printf("Range of x and y is -500 to 500\n\n");

  printf("Enter the coordinates in clockwise/anti-clockwise order:-\n");
  for(i=0; i<8; i++)
   {
   printf("Vertex %d: ",i+1);
   scanf("%d%d%d",&x[i],&y[i],&z[i]);
   }
   printf("\n3D TRANSFORMATION\n\n1. Translation\n2. Rotation\n3. Scaling\n4. Reflection\n5.
  Shearing\n\nEnter your choice: ");
  scanf("%d",&choice);

     switch(choice)
  {
    case 1:
      translation();
      break;
    case 2:
      rotation();
      break;
    case 3:
      scaling();
      break;
    case 4:
      reflection();
      break;
    case 5:
      shearing();
      break;
    default:
      printf("Wrong Choice\n");
      break;
  }   return 0;
}
```

EXPERIMENT – 09

EXPERIMENT: Drawing Bezier curves.

[Virtual GLUT based demonstration]

❖ ***Write a program to draw a cubic spline.***

```
#include <iostream>

#  include <GLUT/glut.h>
# include <iostream>

using namespace std;

// Begin globals.
static int selectedKnot = 0; // Selected knot number.
static int splineOrder = 1; // Order of spline.
static long font = (long)GLUT_BITMAP_8_BY_13; // Font selection.

// Knot values scaled by a factor of 10 to avoid floating point error when comparing knot values.
static float knots[9] = {0.0, 10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0};
// End globals.

// Routine to draw a bitmap character string.
void writeBitmapString(void *font, char *string)
{
  char *c;

  for (c = string; *c != '\0'; c++) glutBitmapCharacter(font, *c);
}

// Initialization routine.
void setup(void)
{
  glClearColor(1.0, 1.0, 1.0, 0.0);
}

// Function to increase value of a knot.
void increaseKnot(int i)
{
  if ( (i < 8) )
  {
    if (knots[i] < knots[i+1]) knots[i] += 1.0;
		else
		{
		  increaseKnot(i+1);
			  if (knots[i] < knots[i+1]) knots[i] += 1.0;
		}
  }
  if ( (i == 8 )  && ( knots[i] < 80) ) knots[i] += 1.0;
}

// Function to decrease value of a knot.
void decreaseKnot(int i)
{
  if ( (i > 0) )
  {
    if (knots[i] > knots[i-1]) knots[i] -= 1.0;
		else
		{
		  decreaseKnot(i-1);
			  if (knots[i] > knots[i-1]) knots[i] -= 1.0;
		}
  }
  if ( (i == 0 )  && ( knots[i] > 0) ) knots[i] -= 1.0;
}

// Reset knot values.
void resetKnots(void)
{
  int i;
  for (i = 0; i < 9; i++) knots[i] = 10.0*i;
  selectedKnot = 0;
}

// Recursive computation of B-spline functions.
float Bspline(int index, int order, float u)
{
  float coef1, coef2;
  if ( order == 1 )
  {
		if ( index == 0 ) if ( ( knots[index] <= u ) && ( u <= knots[index+1] ) ) return 1.0;
    if ( ( knots[index] < u ) && ( u <= knots[index+1] ) ) return 1.0;
		else return 0.0;
  }
  else
  {
    if ( knots[index + order - 1] == knots[index] )
		{
		  if ( u == knots[index] ) coef1 = 1;
			else coef1 = 0;
		}
		else coef1 = (u - knots[index])/(knots[index + order - 1] - knots[index]);

    if ( knots[index + order] == knots[index+1] )
		{
			if ( u == knots[index + order] ) coef2 = 1;
			else coef2 = 0;
		}
		else coef2 = (knots[index + order] - u)/(knots[index + order] - knots[index+1]);

    return ( coef1 * Bspline(index, order-1, u) + coef2 * Bspline(index+1,order-1 ,u) );
  }
}

// Draw a B-spline function graph as line strip and joints as points.
void drawSpline(int index, int order)
{
  float x;
  int j;

  // Drawing are scaled by factor of 3 in the y-direction.
  // Special case to handle order 1 to avoid vertical edges.
  if (order == 1)
  {
		// Spline curve.
    glBegin(GL_LINE_STRIP);
		for ( x = knots[index]; x < knots[index+1]; x+=0.05 )
      glVertex3f( -40.0 + x, 10.0, 0.0 );
		glEnd();
		glPointSize(3.0);

		// Joints.
		glBegin(GL_POINTS);
		glVertex3f( -40.0 + knots[index], 10.0, 0.0);
		glVertex3f( -40.0 + knots[index+1], 10.0, 0.0);
		glEnd();
  }
  else
  {
		// Spline curve.
		glBegin(GL_LINE_STRIP);
		for ( x = knots[index]; x <= knots[index + order]; x += 0.005 )
		  glVertex3f( -40.0 + x, 30*Bspline(index, order, x) - 20.0, 0.0 );
    glEnd();

		// Joints.
		glColor3f(0.0, 0.0, 0.0);
		glBegin(GL_POINTS);
		for (j = index; j <= index + order; j++)
		  glVertex3f( -40.0 + knots[j], 30*Bspline(index, order, knots[j]) - 20.0, 0.0 );
		glEnd();
  }
}

// Drawing routine.
void drawScene(void)
{
  int i;
```

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

glPushMatrix();

glTranslatef(0.0, 0.5, 0.0);

// Write spline order.
glColor3f(0.0, 0.0, 0.0);
switch (splineOrder)
{
  case 1:
     glRasterPos3f(-10.5, 35.0, 0.0);
     writeBitmapString((void*)font, "First-order B-splines");
              break;
  case 2:
     glRasterPos3f(-10.5, 35.0, 0.0);
     writeBitmapString((void*)font, "Linear B-splines");
              break;
  case 3:
     glRasterPos3f(-10.5, 35.0, 0.0);
     writeBitmapString((void*)font, "Quadratic B-splines");
              break;
  case 4:
     glRasterPos3f(-10.5, 35.0, 0.0);
     writeBitmapString((void*)font, "Cubic B-splines");
              break;
  default:
  break;
}

// Draw successive B-spline functions for the chosen order.
glEnable (GL_LINE_SMOOTH);
glEnable (GL_BLEND);
glBlendFunc (GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
glHint (GL_LINE_SMOOTH_HINT, GL_NICEST);
for (i = 0; i < 9 - splineOrder; i++ )
{
              switch (i)
              {
    case 0:
                 glColor3f(1.0, 0.0, 0.0);
                break;
    case 1:
                 glColor3f(0.0, 1.0, 0.0);
                break;
    case 2:
                 glColor3f(0.0, 0.0, 1.0);
                break;
    case 3:
                 glColor3f(1.0, 0.0, 1.0);
                break;
    case 4:
                 glColor3f(0.0, 1.0, 1.0);
                break;
    case 5:
                 glColor3f(1.0, 1.0, 0.0);
                break;
    case 6:
                 glColor3f(0.0, 0.0, 0.0);
                break;
    case 7:
                 glColor3f(1.0, 0.0, 0.0);
                break;
    default:
    break;
              }
   drawSpline(i, splineOrder);
}

// Draw the x-axis.
glColor3f(0.0, 0.0, 0.0);
glBegin(GL_LINES);
  glVertex3f(-40.0, -20.0, 0.0);
  glVertex3f( 40.0, -20.0, 0.0);
glEnd();

// Draw points on the x-axis.
glPointSize(5.0);
```

```
   glBegin(GL_POINTS);
     for (i = 0; i < 9; i++)
        glVertex3f(-40.0 + (float)i*10.0, -20.0, 0.0);
   glEnd();

   // Label the points on the x-axis.
   glRasterPos3f(-40.5, -23.0, 0.0);
   writeBitmapString((void*)font, "0");
   glRasterPos3f(-30.5, -23.0, 0.0);
   writeBitmapString((void*)font, "1");
   glRasterPos3f(-20.5, -23.0, 0.0);
   writeBitmapString((void*)font, "2");
   glRasterPos3f(-10.5, -23.0, 0.0);
   writeBitmapString((void*)font, "3");
   glRasterPos3f(-0.5, -23.0, 0.0);
   writeBitmapString((void*)font, "4");
   glRasterPos3f(9.5, -23.0, 0.0);
   writeBitmapString((void*)font, "5");
   glRasterPos3f(19.5, -23.0, 0.0);
   writeBitmapString((void*)font, "6");
   glRasterPos3f(29.5, -23.0, 0.0);
   writeBitmapString((void*)font, "7");
   glRasterPos3f(39.5, -23.0, 0.0);
   writeBitmapString((void*)font, "8");

   // Draw the y-axis.
   glBegin(GL_LINES);
     glVertex3f(-40.0, -20.0, 0.0);
     glVertex3f(-40.0, 40.0, 0.0);
   glEnd();

   // Draw points on the y-axis.
   glBegin(GL_POINTS);
        glVertex3f(-40.0, 10.0, 0.0);
        glVertex3f(-40.0, 40.0, 0.0);
   glEnd();

   // Label the points on the y-axis.
   glRasterPos3f(-42.5, -20.5, 0.0);
   writeBitmapString((void*)font, "0");
   glRasterPos3f(-42.5, 9.5, 0.0);
   writeBitmapString((void*)font, "1");
   glRasterPos3f(-42.5, 39.5, 0.0);
   writeBitmapString((void*)font, "2");

   // Draw horizontal bars corresponding to knot points.
   glBegin(GL_LINES);
     for (i = 0; i < 9; i++)
                 {
        glVertex3f(-40.0, -35.0 + (float)i, 0.0);
        glVertex3f( 40.0, -35.0 + (float)i, 0.0);
                 }
   glEnd();

   // Draw the knot points as dots on their respective bars.
   glColor3f(0.0, 1.0, 0.0);
   glBegin(GL_POINTS);
     for (i = 0; i < 9; i++)
        glVertex3f( -40.0 + knots[i], -35.0 + (float)i, 0.0);
   glEnd();

   // Highlight the selected knot point.
   glColor3f(1.0, 0.0, 0.0);
   glBegin(GL_POINTS);
     glVertex3f( -40.0 + knots[selectedKnot],
                                     -35.0 + (float)selectedKnot, 0.0);
   glEnd();

   // Label the knot bars.
   glColor3f(0.0, 0.0, 0.0);
   glRasterPos3f(-7.0, -40.0, 0.0);
   writeBitmapString((void*)font, "Knot Values");

   glPopMatrix();
   glutSwapBuffers();
}

// OpenGL window reshape routine.
```

```
void resize(int w, int h)
{
  glViewport(0, 0, (GLsizei)w, (GLsizei)h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-50.0, 50.0, -50.0, 50.0, -1.0, 1.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
}

// Keyboard input processing routine.
void keyInput(unsigned char key, int x, int y)
{
  switch (key)
  {
    case 27:
      exit(0);
      break;
    case ' ':
      if (selectedKnot < 8) selectedKnot++;
                              else selectedKnot = 0;
      glutPostRedisplay();
                              break;
    case 127:
      resetKnots();
                              glutPostRedisplay();
      break;
    default:
      break;
 }
}

// Callback routine for non-ASCII key entry.
void specialKeyInput(int key, int x, int y)
{
  if(key == GLUT_KEY_LEFT) decreaseKnot(selectedKnot);
  if(key == GLUT_KEY_RIGHT) increaseKnot(selectedKnot);
  if(key == GLUT_KEY_UP)
  {
                if (splineOrder < 4) splineOrder++; else splineOrder = 1;
  }
  if(key == GLUT_KEY_DOWN)
  {
                if (splineOrder > 1) splineOrder--; else splineOrder = 4;
  }
  glutPostRedisplay();
}

// Routine to output interaction instructions to the C++ window.
void printInteraction(void)
{
  cout << "Interaction:" << endl;
  cout << "Press the up/down arrow keys to cycle between order 1 through 4." << endl
     << "Press space to select a knot points." << endl
     << "Press the left/right arrow keys to move the selected knot point." << endl
     << "Press delete to reset." << endl;
}

// Main routine.
int main(int argc, char **argv)
{
  printInteraction();
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(600, 600);
  glutInitWindowPosition(100, 100);
  glutCreateWindow("bSplines.cpp");
  setup();
  glutDisplayFunc(drawScene);
  glutReshapeFunc(resize);
  glutKeyboardFunc(keyInput);
  glutSpecialFunc(specialKeyInput);
  glutMainLoop();

  return 0;
}
```

❖ *WAP to draw a Bezier curve.*

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

#include <GL/glu.h>

#include <GL/glut.h>

#include <stdlib.h>


int i;


// Function to initialize the Beizer

// curve pointer

void init(void)

{

        glClearColor(1.0, 1.0, 1.0, 1.0);

}


// Function to draw the Bitmap Text

void drawBitmapText(char* string, float x,

                                        float y, float z)

{

        char* c;

        glRasterPos2f(x, y);


        // Traverse the string

        for (c = string; *c != '\0'; c++) {

                glutBitmapCharacter(

                        GLUT_BITMAP_TIMES_ROMAN_24, *c);

        }

}


// Function to draw the shapes

void draw(GLfloat ctrlpoints[4][3])

{

        glShadeModel(GL_FLAT);

        glMap1f(GL_MAP1_VERTEX_3, 0.0, 1.0, 3, 4,

                        &ctrlpoints[0][0]);


        glEnable(GL_MAP1_VERTEX_3);
        // Fill the color
        glColor3f(1.0, 1.0, 1.0);
        glBegin(GL_LINE_STRIP);

        // Find the coordinates
        for (i = 0; i <= 30; i++)
                glEvalCoord1f((GLfloat)i / 30.0);

        glEnd();
        glFlush();
}

// Function to display the curved
// drawn using the Beizer Curve
void display(void)
{
        int i;

        // Specifying all the control
        // points through which the
        // curve will pass
        GLfloat ctrlpoints[4][3]
                = { { -0.00, 2.00, 0.0 },
                        { -2.00, 2.00, 0.0 },
                        { -2.00, -1.00, 0.0 },
                        { -0.00, -1.00, 0.0 } };
        draw(ctrlpoints);

        GLfloat ctrlpoints2[4][3]
                = { { 0.0, -1.00, 0.0 },
                        { 0.55, -0.65, 0.0 },
                        { 0.65, -0.25, 0.0 },
                        { 0.00, 0.70, 0.0 } };

        draw(ctrlpoints2);
        GLfloat ctrlpoints3[4][3]
                = { { 0.0, 0.70, 0.0 },
                        { 0.15, 0.70, 0.0 },
                        { 0.25, 0.70, 0.0 },
                        { 0.65, 0.700, 0.0 } };

        draw(ctrlpoints3);
        GLfloat ctrlpoints4[4][3]
                = { { 0.65, 0.70, 0.0 },
                        { 0.65, -0.90, 0.0 },
                        { 0.65, -0.70, 0.0 },
                        { 0.65, -1.00, 0.0 } };

        draw(ctrlpoints4);


        GLfloat ctrlpoints5[4][3]
                = { { 1.00, -1.00, 0.0 },
                        { 1.00, -0.5, 0.0 },
                        { 1.00, -0.20, 0.0 },
                        { 1.00, 1.35, 0.0 } };

        draw(ctrlpoints5);
        GLfloat ctrlpoints6[4][3]
                = { { 1.00, 1.35 },
                        { 1.10, 1.35, 0.0 },
                        { 1.10, 1.35, 0.0 },
                        { 1.90, 1.35, 0.0 } };

        draw(ctrlpoints6);
        GLfloat ctrlpoints7[4][3]
                = { { 1.00, 0.50, 0.0 },
                        { 1.10, 0.5, 0.0 },
                        { 1.10, 0.5, 0.0 },
                        { 1.90, 0.5, 0.0 } };

        draw(ctrlpoints7);
        GLfloat ctrlpoints8[4][3]
                = { { 3.50, 2.00, 0.0 },
                        { 1.50, 2.00, 0.0 },
                        { 1.50, -1.00, 0.0 },
                        { 3.50, -1.00, 0.0 } };
        draw(ctrlpoints8);
```

```
GLfloat ctrlpoints9[4][3]
        = { { 3.50, -1.00, 0.0 },
                { 4.05, -0.65, 0.0 },
                { 4.15, -0.25, 0.0 },
                { 3.50, 0.70, 0.0 } };
draw(ctrlpoints9);

GLfloat ctrlpoints10[4][3]
        = { { 3.50, 0.70, 0.0 },
                { 3.65, 0.70, 0.0 },
                { 3.75, 0.70, 0.0 },
                { 4.15, 0.700, 0.0 } };

draw(ctrlpoints10);
GLfloat ctrlpoints11[4][3]
        = { { 4.15, 0.70, 0.0 },
                { 4.15, -0.90, 0.0 },
                { 4.15, -0.70, 0.0 },
                { 4.15, -1.00, 0.0 } };

draw(ctrlpoints11);

GLfloat ctrlpoints12[4][3]
        = { { -2.0, 2.50, 0.0 },
                { 2.05, 2.50, 0.0 },
                { 3.15, 2.50, 0.0 },
                { 4.65, 2.50, 0.0 } };

draw(ctrlpoints12);

GLfloat ctrlpoints13[4][3]
        = { { -2.0, -1.80, 0.0 },
                { 2.05, -1.80, 0.0 },
                { 3.15, -1.80, 0.0 },
                { 4.65, -1.80, 0.0 } };

draw(ctrlpoints13);

GLfloat ctrlpoints14[4][3]
```

```
        = { { -2.0, -1.80, 0.0 },
            { -2.0, 1.80, 0.0 },
            { -2.0, 1.90, 0.0 },
            { -2.0, 2.50, 0.0 } };

    draw(ctrlpoints14);
    GLfloat ctrlpoints15[4][3]
        = { { 4.650, -1.80, 0.0 },
            { 4.65, 1.80, 0.0 },
            { 4.65, 1.90, 0.0 },
            { 4.65, 2.50, 0.0 } };

    draw(ctrlpoints15);

    // Specifying the colour of
    // text to be displayed
    glColor3f(1, 0, 0);
    drawBitmapText("Bezier Curves "
                   "Implementation",
                   -1.00, -3.0, 0);
    glFlush();
}

// Function perform the reshaping
// of the curve
void reshape(int w, int h)
{
    glViewport(0, 0, (GLsizei)w,
               (GLsizei)h);

    // Matrix mode
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();

    if (w <= h)
        glOrtho(-5.0, 5.0, -5.0
                                * (GLfloat)h / (GLfloat)w,
                5.0 * (GLfloat)h / (GLfloat)w, -5.0, 5.0);
    else
```

```
                glOrtho(-5.0 * (GLfloat)w / (GLfloat)h,
                                5.0 * (GLfloat)w / (GLfloat)h,
                                -5.0, 5.0,
                                -5.0, 5.0);

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();
}

// Driver Code
int main(int argc, char** argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(
                GLUT_SINGLE | GLUT_RGB);

        // Specifies the window size
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 100);

        // Creates the window as
        // specified by the user
        glutCreateWindow(argv[0]);
        init();

        // Links display event with the
        // display event handler(display)
        glutDisplayFunc(display);
        glutReshapeFunc(reshape);

        // Loops the current event
        glutMainLoop();

        return 0;
}
```