Lab Program-5

Draw a color cube and allow the user to move the camera suitably to experiment with perspective viewing.

/* Rotating cube with viewer movement */

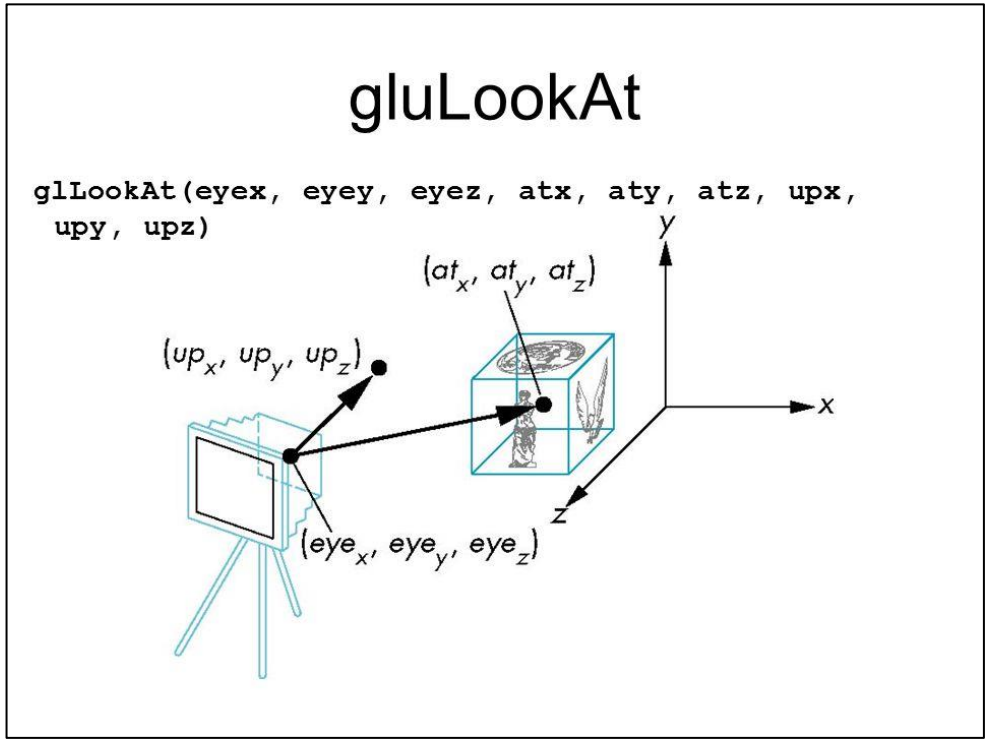
/* We use the Look at function in the display callback to point the viewer, whose position can be altered by the x,X,y,Y,z,&Z keys. The perspective view is set in the reshape callback */

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

    GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
    {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
    {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

    GLfloat normals[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
    {1.0,1.0,-1.0}, {-1.0,1.0,-1.0}, {-1.0,-1.0,1.0},
    {1.0,-1.0,1.0}, {1.0,1.0,1.0}, {-1.0,1.0,1.0}};

    GLfloat colors[][3] = {{0.0,0.0,0.0},{1.0,0.0,0.0},
    {1.0,1.0,0.0}, {0.0,1.0,0.0}, {0.0,0.0,1.0},
    {1.0,0.0,1.0}, {1.0,1.0,1.0}, {0.0,1.0,1.0}};
```

```
void polygon(int a, int b, int c , int d)
{
        glBegin(GL_POLYGON);
                glColor3fv(colors[a]);
                glNormal3fv(normals[a]);
                glVertex3fv(vertices[a]);
                glColor3fv(colors[b]);
                glNormal3fv(normals[b]);
                glVertex3fv(vertices[b]);
                glColor3fv(colors[c]);
                glNormal3fv(normals[c]);
                glVertex3fv(vertices[c]);
                glColor3fv(colors[d]);
                glNormal3fv(normals[d]);
                glVertex3fv(vertices[d]);
        glEnd();
                        }
```

```
void colorcube()
{
        polygon(0,3,2,1);
        polygon(2,3,7,6);
        polygon(0,4,7,3);
        polygon(1,2,6,5);
        polygon(4,5,6,7);
        polygon(0,1,5,4);
}

static GLfloat theta[] = {0.0,0.0,0.0};
static GLint axis = 2;
static GLdouble viewer[]= {0.0, 0.0, 5.0}; /* initial viewer location */
```

```
void display(void)
{
 glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

/* Update viewer position in modelview matrix */
        glLoadIdentity();
        gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

/* rotate cube */
        glRotatef(theta[0], 1.0, 0.0, 0.0);
        glRotatef(theta[1], 0.0, 1.0, 0.0);
        glRotatef(theta[2], 0.0, 0.0, 1.0);

 colorcube();

 glFlush();
        glutSwapBuffers();
}
```



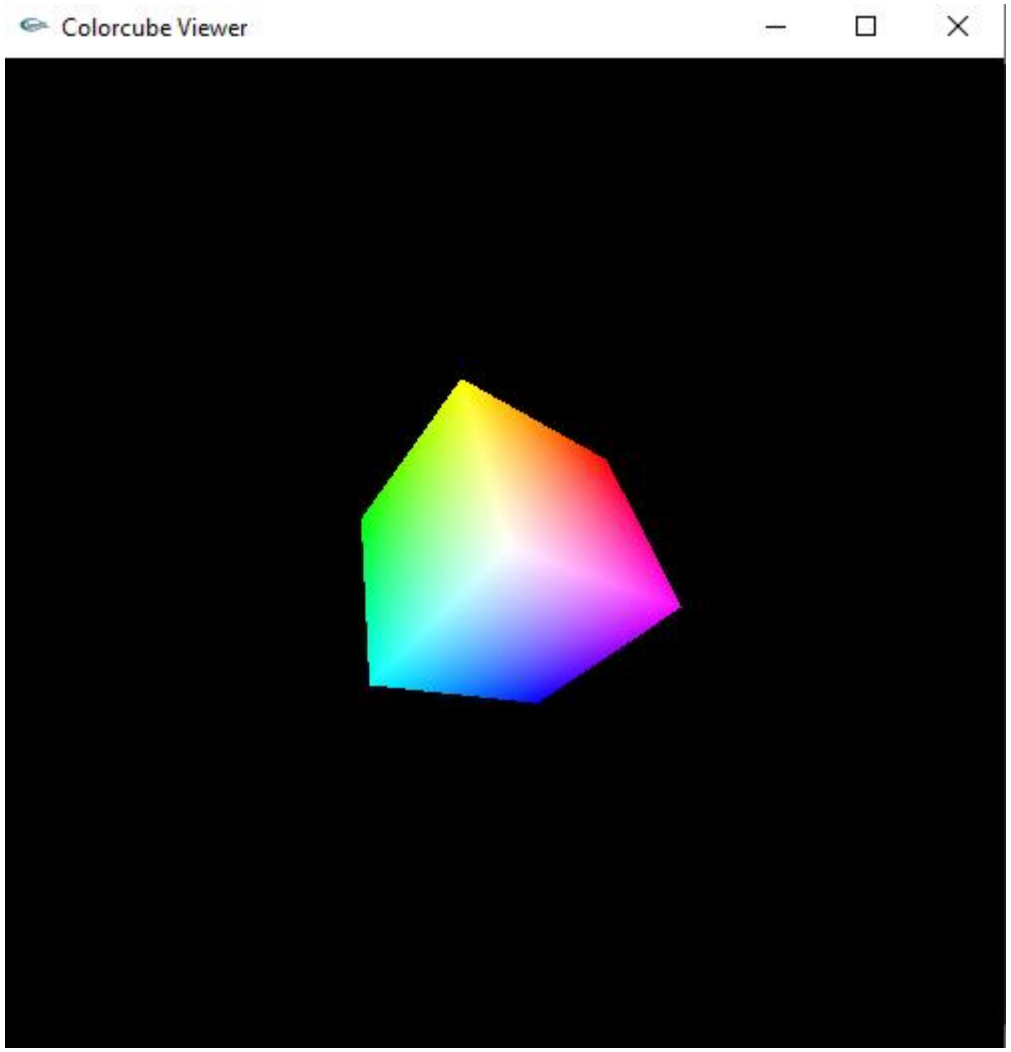
```
void mouse(int btn, int state, int x, int y)
{
        if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
        if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis = 1;
        if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
        theta[axis] += 2.0;
        if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
        display();
}
```

```
void keys(unsigned char key, int x, int y)
{

/* Use x, X, y, Y, z, and Z keys to move viewer */

   if(key == 'x') viewer[0]-= 1.0;
   if(key == 'X') viewer[0]+= 1.0;
   if(key == 'y') viewer[1]-= 1.0;
   if(key == 'Y') viewer[1]+= 1.0;
   if(key == 'z') viewer[2]-= 1.0;
   if(key == 'Z') viewer[2]+= 1.0;
   display();
}
```

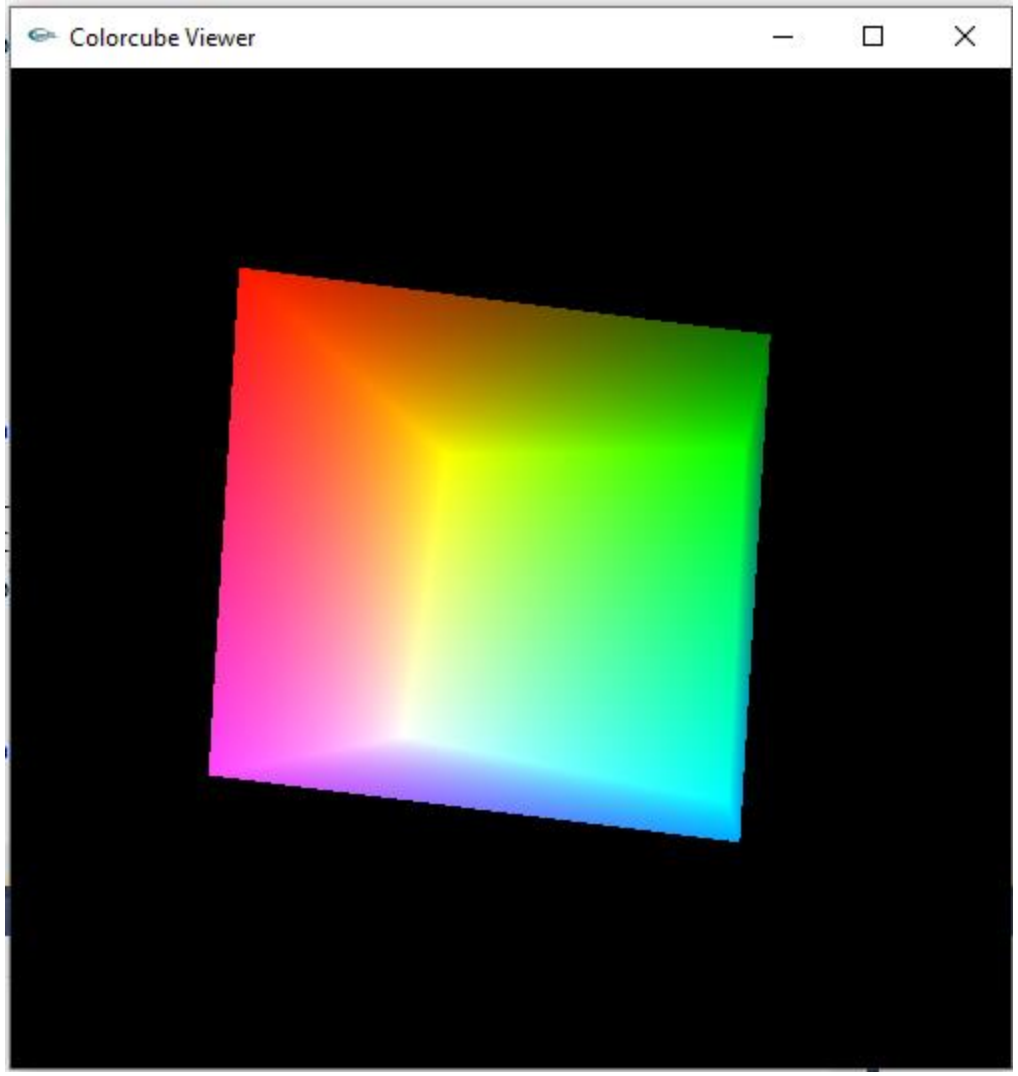
```
void myReshape(int w, int h)
{
 glViewport(0, 0, w, h);

/* Use a perspective view */
 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
          if(w<=h)
glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w,
      2.0* (GLfloat) h / (GLfloat) w, 2.0, 20.0);
          else glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h,
      2.0* (GLfloat) w / (GLfloat) h, 2.0, 20.0);

/* Or we can use gluPerspective */

 /* gluPerspective(45.0, w/h, -10.0, 10.0); */
 glMatrixMode(GL_MODELVIEW);
}
```

```
void  main(int argc, char **argv)
{
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
 glutInitWindowSize(500, 500);
 glutCreateWindow("Colorcube Viewer");
 glutReshapeFunc(myReshape);
 glutDisplayFunc(display);
        glutMouseFunc(mouse);
        glutKeyboardFunc(keys);
        glEnable(GL_DEPTH_TEST);
 glutMainLoop();
}
```

OUTPUT



OUTPUT



Lab Program 6

Develop a program to animate a flag using Bezier Curve algorithm.

```
#include<GL/glut.h>
#include<stdio.h>
#include<math.h>
#define PI 3.1416
GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;
typedef struct wcPt3D
{
GLfloat x, y, z;
};
```

```
void bino(GLint n, GLint *C)
{
GLint k, j;
for(k=0;k<=n;k++)
{
C[k]=1;
for(j=n;j>=k+1; j--)
C[k]*=j;
for(j=n-k;j>=2;j--)
C[k]/=j;
}
}
```

$$P(u) = \sum_{k=0}^{n} p_k \, BEZ_{k,n}(u), \qquad 0 \le u \le 1$$

```
void computeBezPt(GLfloat u, wcPt3D *bezPt, GLint nCtrlPts, wcPt3D *ctrlPts, GLint*C)
{
GLint k, n=nCtrlPts-1;
GLfloat bezBlendFcn;
bezPt ->x =bezPt ->y = bezPt->z=0.0;
for(k=0; k< nCtrlPts; k++)
{
bezBlendFcn = C[k] * pow(u, k) * pow( 1-u, n-k);
bezPt ->x += ctrlPts[k].x * bezBlendFcn;
bezPt ->y += ctrlPts[k].y * bezBlendFcn;
bezPt ->z += ctrlPts[k].z * bezBlendFcn;
}
}
```

The Bezier blending functions BEZk,n (u) are the Bernstein polynomials

$$BEZ_{k,n}(u) = C(n,k)u^k(1-u)^{n-k}$$

where parameters *C(n, k)* are the binomial coefficients

$$C(n,k) = \frac{n!}{k!(n-k)!}$$

```
void bezier(wcPt3D *ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
wcPt3D bezCurvePt;
GLfloat u;
GLint *C, k;
C= new GLint[nCtrlPts];
bino(nCtrlPts-1, C);
glBegin(GL_LINE_STRIP);
for(k=0; k<=nBezCurvePts; k++)
```

Computing set of three parametric equations for the individual curve coordinates (x,y,z)

$$x(u) = \sum_{k=0}^{n} x_k \, \mathrm{BEZ}_{k,n}(u)$$

$$y(u) = \sum_{k=0}^{n} y_k \, \mathrm{BEZ}_{k,n}(u)$$

$$z(u) = \sum_{k=0}^{n} z_k \, \mathrm{BEZ}_{k,n}(u)$$

Bezier curve is a polynomial of a degree that is one less than the designated number of control points

```
{
u=GLfloat(k)/GLfloat(nBezCurvePts);
computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);
glVertex2f(bezCurvePt.x, bezCurvePt.y);
}
glEnd();
delete[]C;
}
```

```
void displayFcn()
{
GLint nCtrlPts = 4, nBezCurvePts =20;
static float theta = 0;
wcPt3D ctrlPts[4] = { {20, 100, 0},{30, 110, 0},{50, 90, 0},{60, 100, 0}};
ctrlPts[1].x +=10*sin(theta * PI/180.0);
ctrlPts[1].y +=5*sin(theta * PI/180.0);
ctrlPts[2].x -= 10*sin((theta+30) * PI/180.0);
ctrlPts[2].y -= 10*sin((theta+30) * PI/180.0);
ctrlPts[3].x-= 4*sin((theta) * PI/180.0);
ctrlPts[3].y += sin((theta-30) * PI/180.0);
theta+=0.1;
```

```
glClear(GL_COLOR_BUFFER_BIT);
glColor3f(1.0, 1.0, 1.0);
glPointSize(5);
glPushMatrix();
glLineWidth(5);
glColor3f(255/255, 153/255.0, 51/255.0); //Indian flag: saffron colour code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
```

Colors Rendered as

Saffron- RGB (255,153,51)

White –RGB (255, 255, 255)

Green –RGB (18,136,7)

```
glColor3f(1, 1, 1); //Indian flag: white colour code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
glColor3f(19/255.0, 136/255.0, 8/255.0); //Indian flag: green colour code
for(int i=0;i<8;i++)
{
glTranslatef(0, -0.8, 0);
bezier(ctrlPts, nCtrlPts, nBezCurvePts);
}
```

```
glPopMatrix();
glColor3f(0.7, 0.5,0.3);
glLineWidth(5);
glBegin(GL_LINES);
glVertex2f(20,100);
glVertex2f(20,40);
glEnd();
glFlush();
glutPostRedisplay();
glutSwapBuffers();
}
```

/* **glPushMatrix** - Copies the top matrix in the stack and store copy in the second stack position.

/* **glPopMatrix** - Erases the top matrix in the stack and moves the second matrix to the top of the stack.

```
void winReshapeFun(GLint newWidth, GLint newHeight)
{
glViewport(0, 0, newWidth, newHeight);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
glClear(GL_COLOR_BUFFER_BIT);
}
```

```
void main(int argc, char **argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
glutInitWindowPosition(50, 50);
glutInitWindowSize(winWidth, winHeight);
glutCreateWindow("Bezier Curve-Animate Indian Flag");
glutDisplayFunc(displayFcn);
glutReshapeFunc(winReshapeFun);
glutMainLoop();
}
```

OUTPUT: