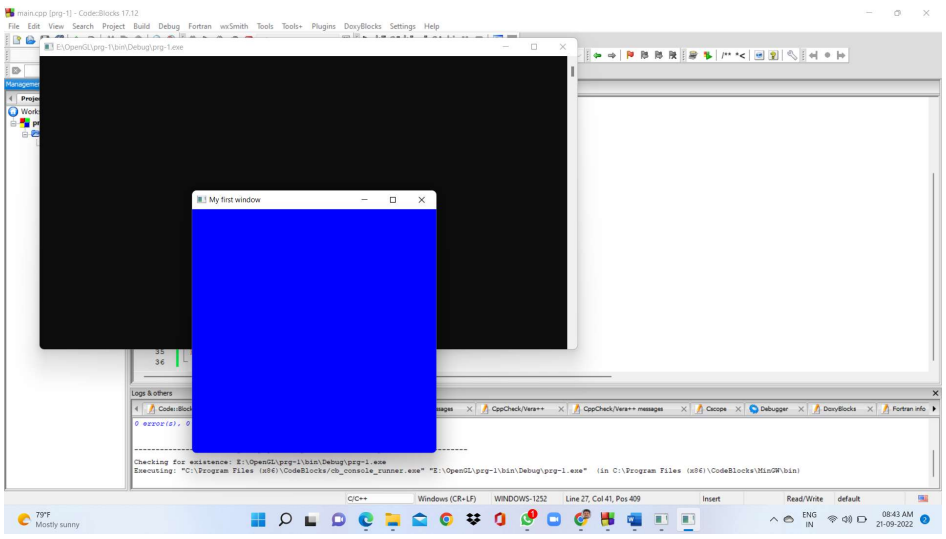
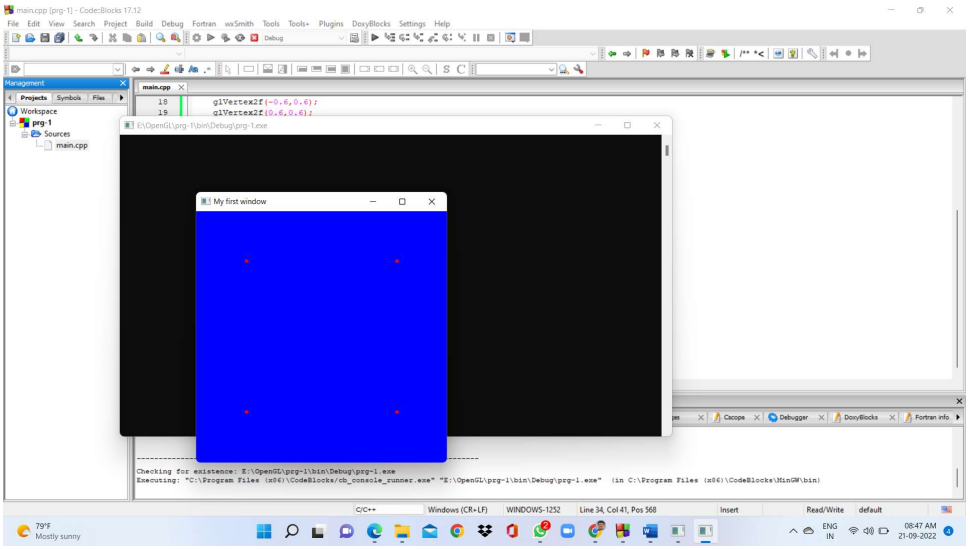
1. Program to create a window with different colours using OpenGL.

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


void MyInit()
{
  glClearColor(0,0,1,1);
}

void draw()
{
 glClear(GL_COLOR_BUFFER_BIT);

 glBegin(0);
 glEnd();

  glFlush();
}
int main(int c, char *v[])
{
   glutInit(&c,v);
   glutInitWindowPosition(300,300);
   glutInitWindowSize(400,400);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutCreateWindow("My first window");

   MyInit();

   glutDisplayFunc(draw);
   glutMainLoop();

   return 0;
}
```



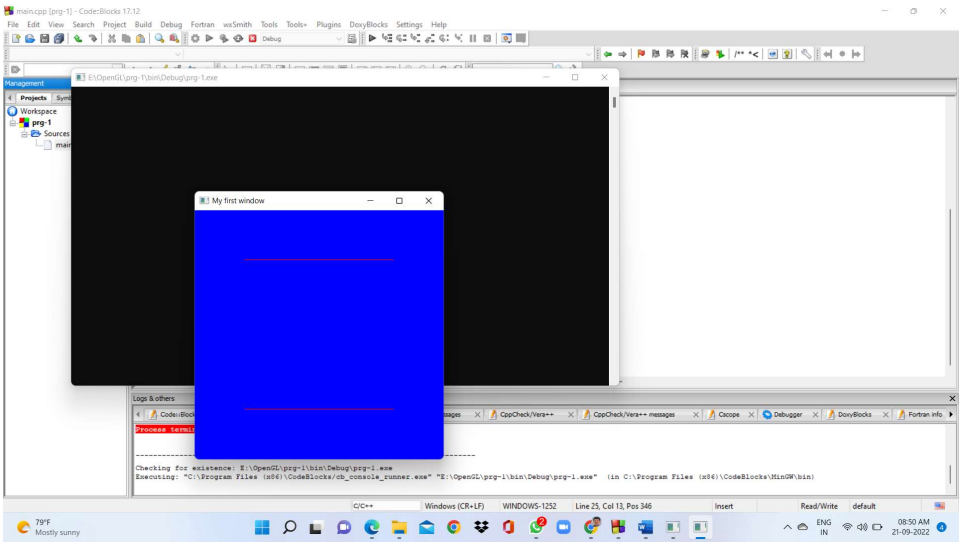
2. Program to create a window with four vertices using OpenGL.

```
#include<GL/glu.h>
#include <GL/glut.h>
void MyInit()
{
  glClearColor(0,0,1,1);
  glColor3f(1,0,0);
}
void draw()
{
 glClear(GL_COLOR_BUFFER_BIT);
 glPointSize(5);

glBegin(GL_POINTS);
   glVertex2f(-0.6,0.6);
   glVertex2f(0.6,0.6);
   glVertex2f(0.6,-0.6);
   glVertex2f(-0.6,-0.6);
glEnd();
 glFlush();
}
int main(int c, char *v[])
{
   glutInit(&c,v);
   glutInitWindowPosition(300,300);
   glutInitWindowSize(400,400);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutCreateWindow("My first window");
   MyInit();
   glutDisplayFunc(draw);
   glutMainLoop();
   return 0;
}
```



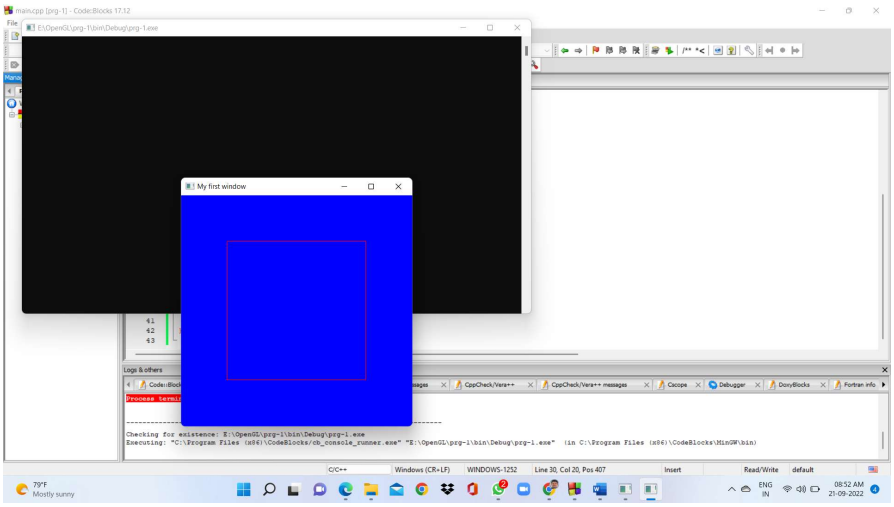
3. Program to draw two parallel lines in a window using OpenGL.

```
#include<GL/glu.h>
#include <GL/glut.h>
void MyInit()
{
  glClearColor(0,0,1,1);
  glColor3f(1,0,0);
}
void draw()
{
 glClear(GL_COLOR_BUFFER_BIT);
 glPointSize(5);

 glBegin(GL_LINES);
   glVertex2f(-0.6,0.6);
   glVertex2f(0.6,0.6);
   glVertex2f(0.6,-0.6);
   glVertex2f(-0.6,-0.6);
 glEnd();

 glFlush();
}
int main(int c, char *v[])
{
   glutInit(&c,v);
   glutInitWindowPosition(300,300);
   glutInitWindowSize(400,400);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutCreateWindow("My first window");
   MyInit();
   glutDisplayFunc(draw);
   glutMainLoop();
   return 0;
}
```



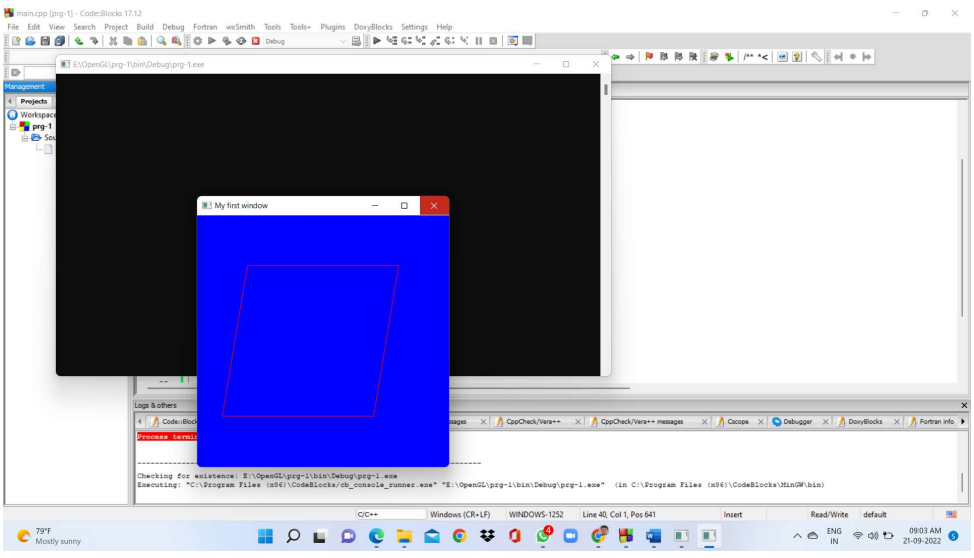
4. Program to draw a square in a window using OpenGL.

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

void MyInit()
{
  glClearColor(0,0,1,1);
  glColor3f(1,0,0);
}

void draw()
{
 glClear(GL_COLOR_BUFFER_BIT);
 glPointSize(5);

 glBegin(GL_LINE_LOOP);
   glVertex2f(-0.6,0.6);
   glVertex2f(0.6,0.6);
   glVertex2f(0.6,-0.6);
   glVertex2f(-0.6,-0.6);
 glEnd();
 glFlush();
}
int main(int c, char *v[])
{
   glutInit(&c,v);
   glutInitWindowPosition(300,300);
   glutInitWindowSize(400,400);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutCreateWindow("My first window");

   MyInit();
   glutDisplayFunc(draw);
   glutMainLoop();
   return 0;
}
```



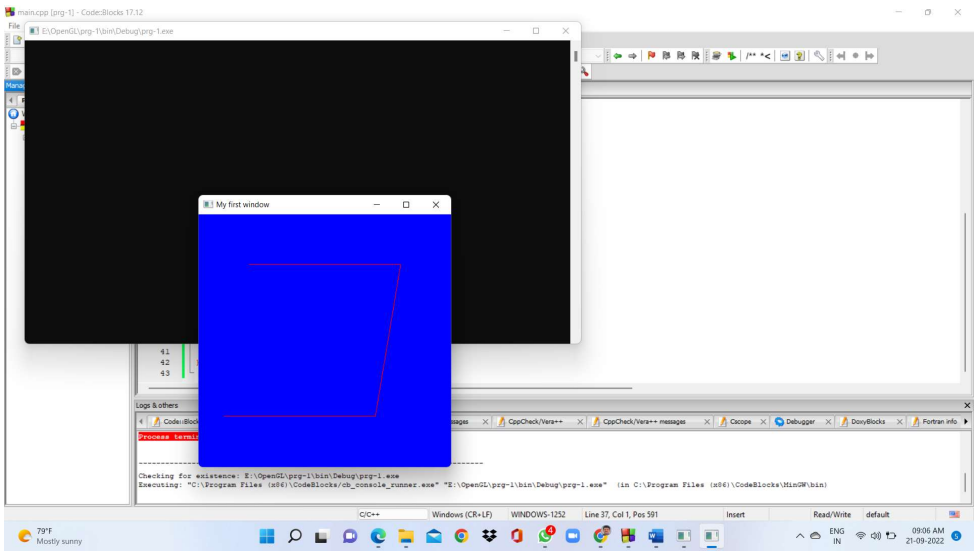
5. Program to draw a rectangle in a window using OpenGL.

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

void MyInit()
{
  glClearColor(0,0,1,1);
  glColor3f(1,0,0);
}
void draw()
{
 glClear(GL_COLOR_BUFFER_BIT);
 glPointSize(5);
glBegin(GL_LINE_LOOP);
   glVertex2f(-0.6,0.6);
   glVertex2f(0.6,0.6);
   glVertex2f(0.4,-0.6);
   glVertex2f(-0.8,-0.6);
glEnd();
 glFlush();
}
int main(int c, char *v[])
{
   glutInit(&c,v);
   glutInitWindowPosition(300,300);
   glutInitWindowSize(400,400);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutCreateWindow("My first window");
   MyInit();
   glutDisplayFunc(draw);
   glutMainLoop();
   return 0;
}
```



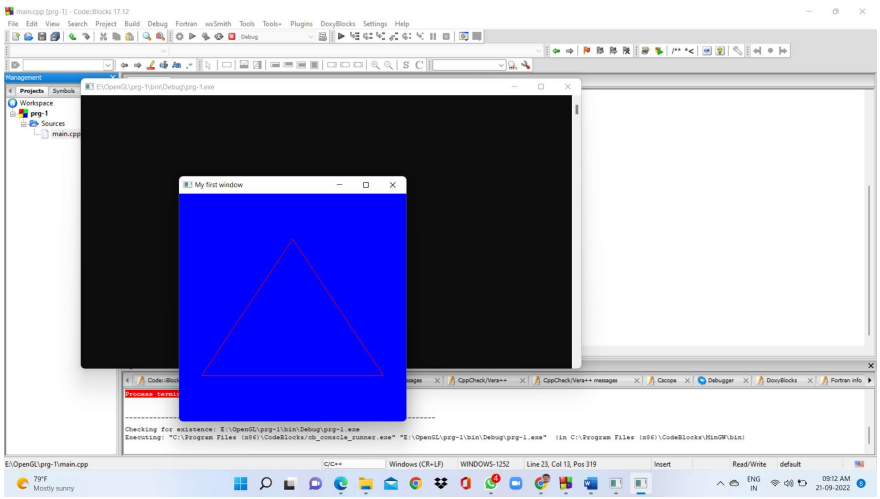
6. Program to draw a connected group of line segments with RED colour from first vertex to last.

```
#include<GL/glu.h>
#include <GL/glut.h>
void MyInit()
{
  glClearColor(0,0,1,1);
  glColor3f(1,0,0);
}
void draw()
{
 glClear(GL_COLOR_BUFFER_BIT);
 glPointSize(5);

 glBegin(GL_LINE_STRIP);
   glVertex2f(-0.6,0.6);
   glVertex2f(0.6,0.6);
   glVertex2f(0.4,-0.6);
   glVertex2f(-0.8,-0.6);
 glEnd();
 glFlush();
}
int main(int c, char *v[])
{
   glutInit(&c,v);
   glutInitWindowPosition(300,300);
   glutInitWindowSize(400,400);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutCreateWindow("My first window");
   MyInit();
   glutDisplayFunc(draw);
   glutMainLoop();
   return 0;
}
```



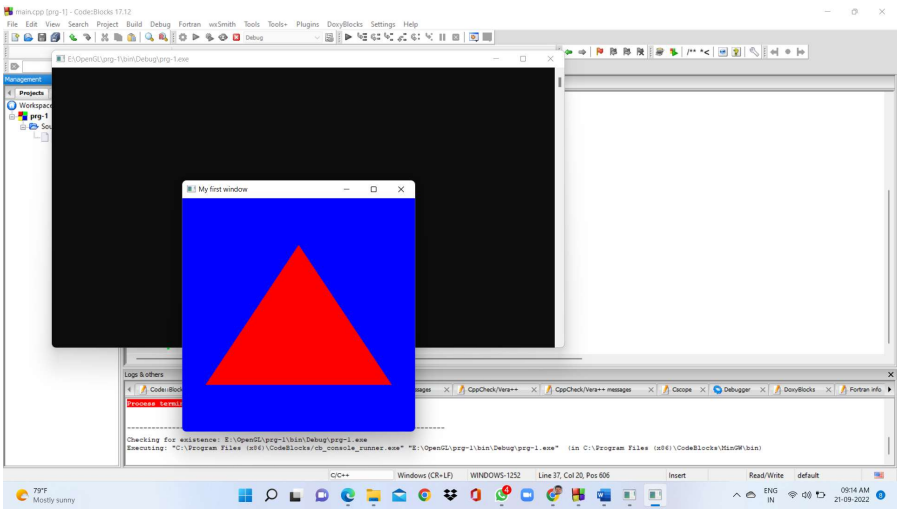
7. Program to draw a triangle with GREEN colour in a window using OpenGL.

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

void MyInit()
{
  glClearColor(0,0,1,1);
  glColor3f(1,0,0);
}
void draw()
{
 glClear(GL_COLOR_BUFFER_BIT);
 glPointSize(5);

 glBegin(GL_LINE_LOOP);
   glVertex2f(0.0,0.6);
   glVertex2f(0.8,-0.6);
   glVertex2f(-.8,-0.6);
glEnd();

 glFlush();
}
int main(int c, char *v[])
{
   glutInit(&c,v);
   glutInitWindowPosition(300,300);
   glutInitWindowSize(400,400);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutCreateWindow("My first window");
   MyInit();

   glutDisplayFunc(draw);
   glutMainLoop();

   return 0;
}
```



8. Program to draw a polygon in a newly created window using OpenGL.

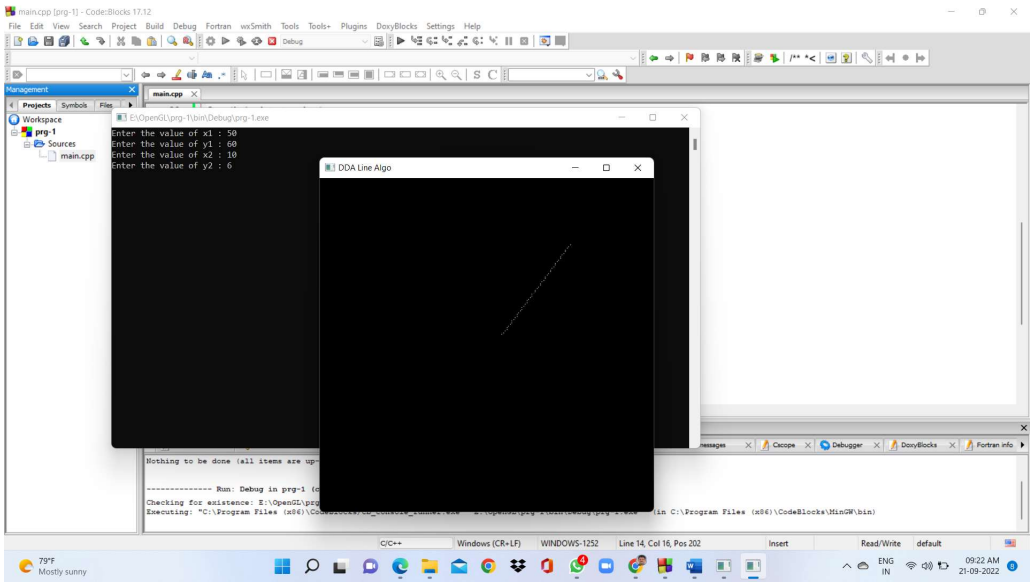
```
#include<GL/glu.h>
#include <GL/glut.h>

void MyInit()
{
  glClearColor(0,0,1,1);
  glColor3f(1,0,0);
}
void draw()
{
 glClear(GL_COLOR_BUFFER_BIT);
 glPointSize(5);

 glBegin(GL_POLYGON);
   glVertex2f(0.0,0.6);
   glVertex2f(0.8,-0.6);
   glVertex2f(-.8,-0.6);
glEnd();

 glFlush();
}
int main(int c, char *v[])
{
   glutInit(&c,v);
   glutInitWindowPosition(300,300);
   glutInitWindowSize(400,400);
   glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
   glutCreateWindow("My first window");

   MyInit();
   glutDisplayFunc(draw);
   glutMainLoop();
   return 0;
}
```



9. Program to draw a line with DDA algorithm.

```
#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);
}
```

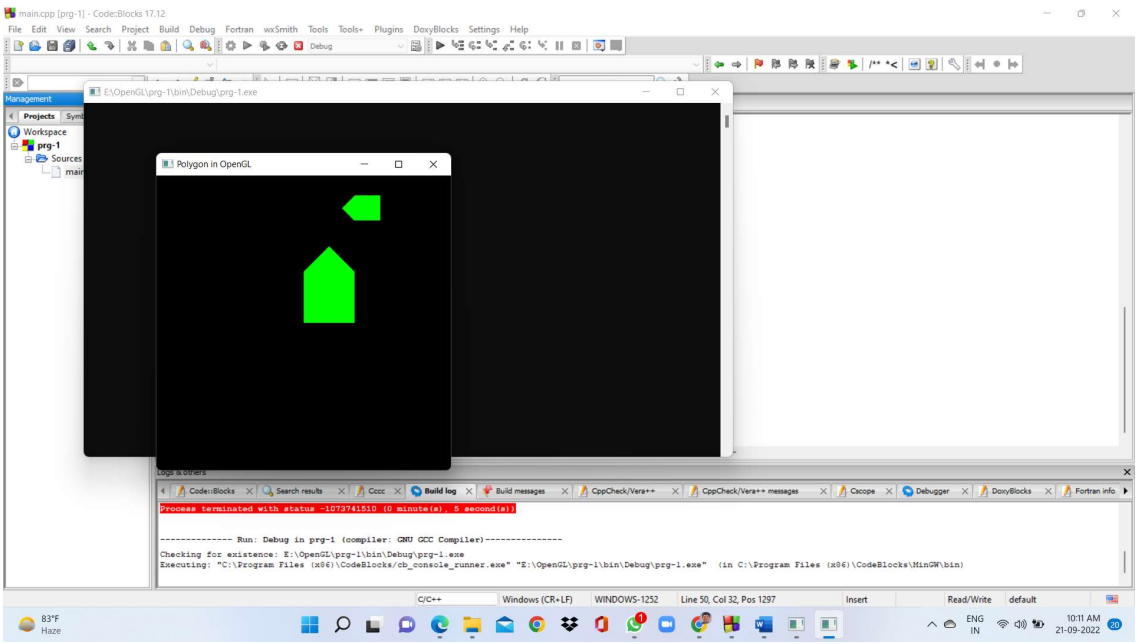
```
int main(int argc, char** argv) {
printf("Enter the value of x1 : ");
scanf("%f",&x1);
printf("Enter the value of y1 : ");
scanf("%f",&y1);
printf("Enter the value of x2 : ");
scanf("%f",&x2);
printf("Enter the value of y2 : ");
scanf("%f",&y2);

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

return 0;
}
```



10. Program to draw a polygon and transform from its original shape to another using OpenGL.

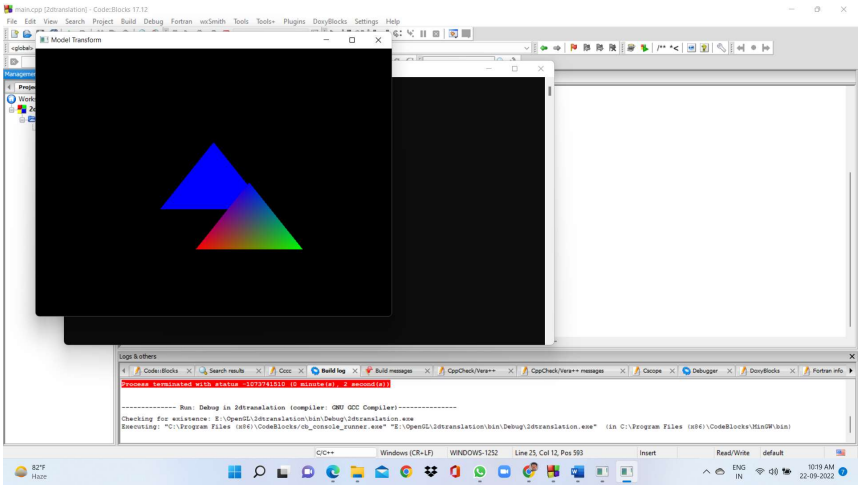
```
#include "gl/glut.h"
#include <gl/gl.h>

void Display(void)
{
	glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
	glLoadIdentity();
	gluLookAt(0.0, 0.0, 5.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
	 glColor3f(0.0, 1.0, 0.0);
	glBegin(GL_POLYGON);
		glVertex3f( 0.0, 0.0, 0.0); // V0 ( 0, 0, 0)
		glVertex3f( 1.0f, 0.0, 0.0); // V1 ( 1, 0, 0)
		 glVertex3f( 1.0f, 1.0f, 0.0); // V2 ( 1, 1, 0)
		 glVertex3f( 0.5f, 1.5f, 0.0); // V3 (0.5, 1.5, 0)
		glVertex3f( 0.0, 1.0f, 0.0); // V4 ( 0, 1, 0)
	glEnd();

	glPushMatrix();
	glTranslatef(1.5, 2.0, 0.0);
	glRotatef(90.0, 0.0, 0.0, 1.0);
	glScalef(0.5, 0.5, 0.5);

	glBegin(GL_POLYGON);
		 glVertex3f( 0.0, 0.0, 0.0); // V0 ( 0, 0, 0)
		 glVertex3f( 1.0f, 0.0, 0.0); // V1 ( 1, 0, 0)
		glVertex3f( 1.0f, 1.0f, 0.0); // V2 ( 1, 1, 0)
		 glVertex3f( 0.5f, 1.5f, 0.0); // V3 (0.5, 1.5, 0)
		 glVertex3f( 0.0, 1.0f, 0.0); // V4 ( 0, 1, 0)
	glEnd();
	glPopMatrix();
	glFlush();
	glutSwapBuffers();
}

void Init(void)
{
	glClearColor(0.0, 0.0, 0.0, 0.0);
}

void Resize(int width, int height)
{
	glViewport(0, 0, width, height);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluPerspective(60.0, width/height, 0.1, 1000.0);
	glMatrixMode(GL_MODELVIEW);
	glLoadIdentity();
}

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

```
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
        glutInitWindowSize(400, 400);
        glutInitWindowPosition(200, 200);
        glutCreateWindow("Polygon in OpenGL");
         Init();
        glutDisplayFunc(Display);
        glutReshapeFunc(Resize);
        glutMainLoop();
         return 0;
}
```



11. Program to perform 2D translation in a window using OpenGL

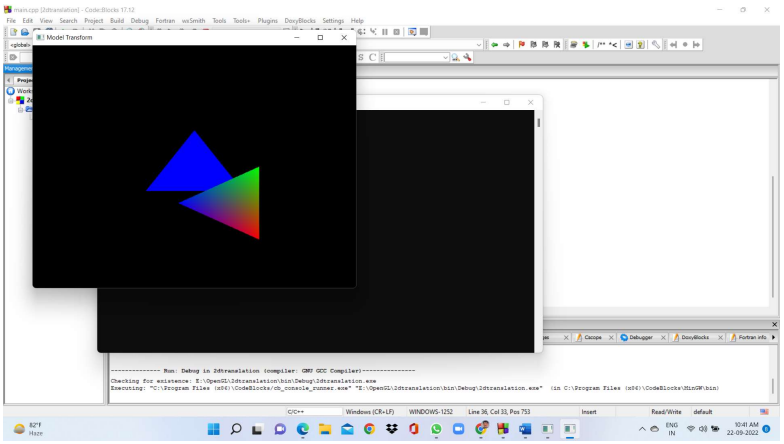
```
#include <windows.h>
#include <GL/glut.h>

void initGL() {
  glClearColor(0.0, 0.0, 0.0, 1.0);
}
void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_TRIANGLES);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f(-0.3, -0.2);
    glVertex2f( 0.3, -0.2);
    glVertex2f( 0.0,  0.3);
  glEnd();

  glTranslatef(0.2, -0.3, 0.0);
    glBegin(GL_TRIANGLES);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2f(-0.3, -0.2);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2f( 0.3, -0.2);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f( 0.0,  0.3);
  glEnd();
  glFlush();
}
int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(50, 50);
  glutCreateWindow("Model Transform");
  glutDisplayFunc(display);
  initGL();
  glutMainLoop();
  return 0;
}
```



12. Program to perform 2D rotation in a window using OpenGL.

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

void initGL() {

  glClearColor(0.0, 0.0, 0.0, 1.0);
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);

  glBegin(GL_TRIANGLES);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f(-0.3, -0.2);
    glVertex2f( 0.3, -0.2);
    glVertex2f( 0.0,  0.3);
  glEnd();

  glTranslatef(0.2, -0.3, 0.0);
  glRotatef(90.0, 0.0, 0.0, 1.0);
    glBegin(GL_TRIANGLES);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2f(-0.3, -0.2);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2f( 0.3, -0.2);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f( 0.0,  0.3);
  glEnd();

  glFlush();
}


int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitWindowSize(640, 480);
```

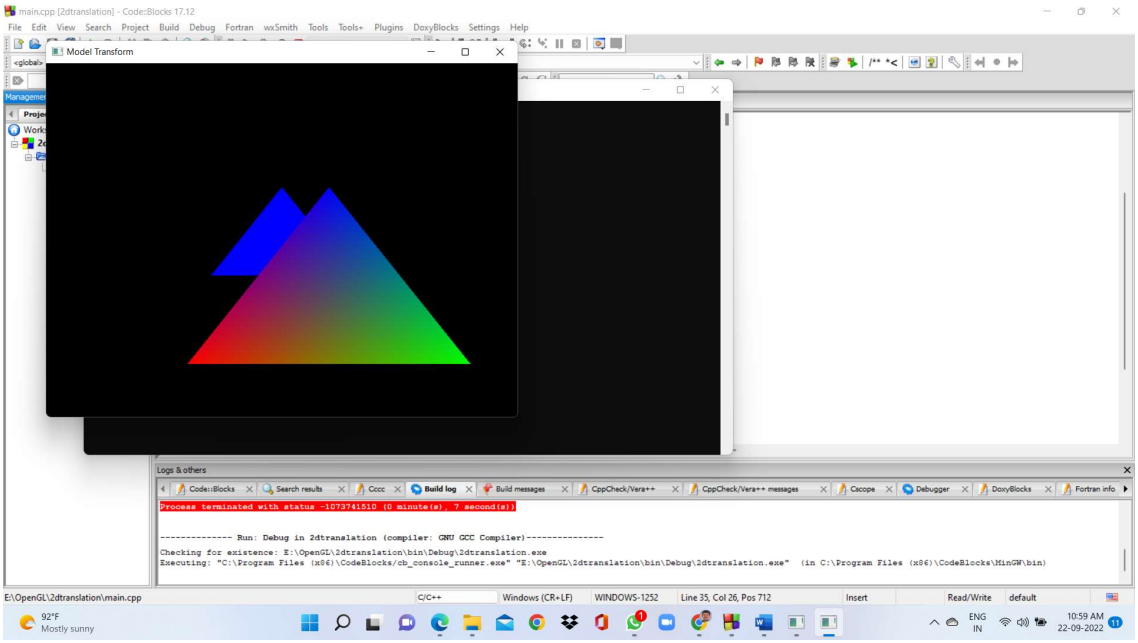
```
  glutInitWindowPosition(50, 50);
  glutCreateWindow("Model Transform");
  glutDisplayFunc(display);

  initGL();
  glutMainLoop();
  return 0;
}
```



13. Program to perform 2D scaling in a window using OpenGL.

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

void initGL() {

  glClearColor(0.0, 0.0, 0.0, 1.0);
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);

  glBegin(GL_TRIANGLES);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f(-0.3, -0.2);
    glVertex2f( 0.3, -0.2);
    glVertex2f( 0.0,  0.3);
  glEnd();

  glTranslatef(0.2, -0.3, 0.0);
  glScalef(2.0, 2.0, 2.0);
    glBegin(GL_TRIANGLES);
    glColor3f(1.0, 0.0, 0.0);
    glVertex2f(-0.3, -0.2);
    glColor3f(0.0, 1.0, 0.0);
    glVertex2f( 0.3, -0.2);
    glColor3f(0.0, 0.0, 1.0);
    glVertex2f( 0.0,  0.3);
```

```
  glEnd();

  glFlush();
}


int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitWindowSize(640, 480);
  glutInitWindowPosition(50, 50);
  glutCreateWindow("Model Transform");
  glutDisplayFunc(display);

  initGL();
  glutMainLoop();
  return 0;
}
```

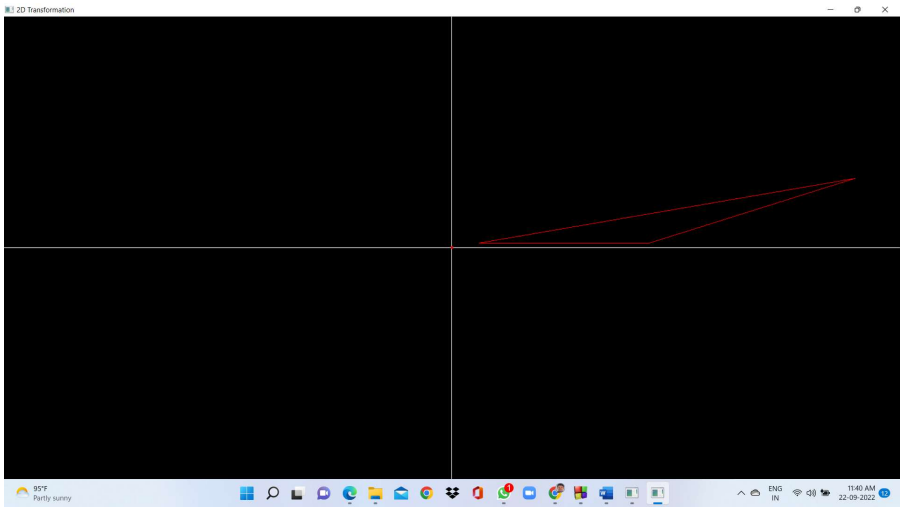


14. Program to perform 2D shearing in a window using OpenGL

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

double parr[8];

void init()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glClearColor(0,0,0,1);
   glColor3f(1,0,1);
   gluOrtho2D(-500,500,-500,500); // Left,right,bottom,top

   // Polygon Default
   parr[0] = 10;  //x
   parr[1] = 10;  //y

   parr[2] = 200;
   parr[3] = 10;

   parr[4] = 150;
   parr[5] = 150;
}

void polygon()
{
   glColor3f(1,0,0);
   glBegin(GL_LINE_LOOP);
      glVertex2f(parr[0],parr[1]);
      glVertex2f(parr[2],parr[3]);

      glVertex2f(parr[4],parr[5]);

   glEnd();

   glFlush();
}

void drawCorodinates()
{
   glClear(GL_COLOR_BUFFER_BIT);

   glColor3f(1,1,1);
   glPointSize(4);

   glBegin(GL_LINES);
      glVertex2f(-500,0);
      glVertex2f(500,0);

      glVertex2f(0,500);
      glVertex2f(0,-500);
   glEnd();
```

```
   glColor3f(1,0,0);

   glBegin(GL_POINTS);
      glVertex2f(0,0);
   glEnd();

   glFlush();
}

// Shearing About X-Axis
void shearing_x()
{
    int shx = 2;

   for(int i=0;i<6;i=i+2)
   {
      parr[i] = parr[i] + shx*parr[i+1];
   }

   polygon();
}


// Shearing About Y-Axis
void shearing_y()
{
    int shy = 2;

   for(int i=1;i<6;i=i+2)
   {
      parr[i] = parr[i] + shy*parr[i-1];
   }

   polygon();
}

void menu(int ch)
{
      drawCorodinates();
   switch(ch)
   {
      case 1: polygon();
         break;

      case 6: shearing_x();
         break;
      case 7: shearing_y();
         break;
   }
}

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

```
{
   glutInit(&argc,argv);
   glutInitWindowSize(500,500);
   glutInitWindowPosition(100,100);

   glutCreateWindow("2D Transformation");
   init();
   glutDisplayFunc(drawCorodinates);

   glutCreateMenu(menu);
      glutAddMenuEntry("1 Display Polygon",1);
      glutAddMenuEntry("6 Shearing About X-Axis",6);
      glutAddMenuEntry("7 Shearing About Y-Axiis",7);
   glutAttachMenu(GLUT_RIGHT_BUTTON);

   glutMainLoop();
   return 0;
}
```



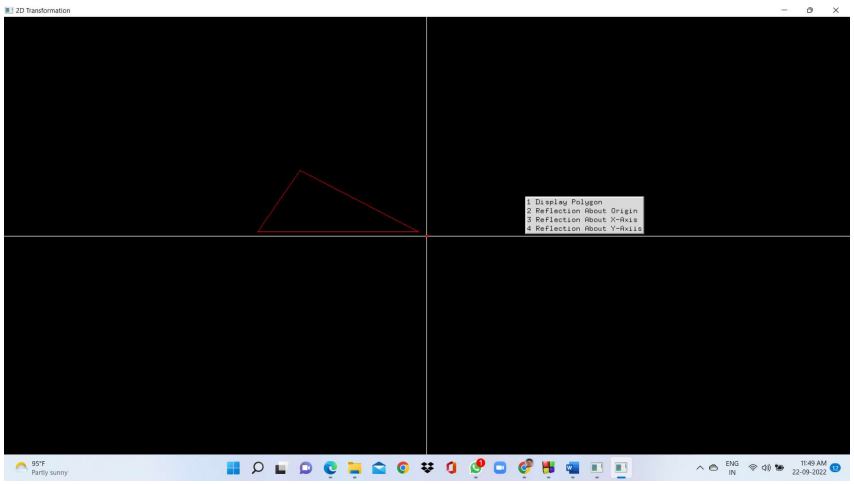
15. Program to perform 2D reflection in a window using OpenGL

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

double parr[8];

void init()
{
   glClear(GL_COLOR_BUFFER_BIT);
   glClearColor(0,0,0,1);
   glColor3f(1,0,1);
   gluOrtho2D(-500,500,-500,500); // Left,right,bottom,top

   // Polygon Defaut
```

```
   parr[0] = 10;  //x
   parr[1] = 10;  //y

   parr[2] = 200;
   parr[3] = 10;

   parr[4] = 150;
   parr[5] = 150;
}


void polygon()
{
   glColor3f(1,0,0);
   glBegin(GL_LINE_LOOP);
      glVertex2f(parr[0],parr[1]);
      glVertex2f(parr[2],parr[3]);

      glVertex2f(parr[4],parr[5]);

   glEnd();

   glFlush();
}


void drawCorodinates()
{
   glClear(GL_COLOR_BUFFER_BIT);

   glColor3f(1,1,1);
   glPointSize(4);

   glBegin(GL_LINES);
      glVertex2f(-500,0);
      glVertex2f(500,0);

      glVertex2f(0,500);
      glVertex2f(0,-500);
   glEnd();

   glColor3f(1,0,0);

   glBegin(GL_POINTS);
      glVertex2f(0,0);
   glEnd();

   glFlush();
}


// Reflection About Origin
void reflection_origin()
```

```
{
   for(int i=0;i<6;i++)
   {
      parr[i] = -parr[i];
   }

   polygon();
}


// Reflection About X-axis
void reflection_x()
{
   for(int i=1;i<6;i=i+2)
   {
      parr[i] = -parr[i];
   }

   polygon();
}


// Reflection About Y-axis
void reflection_y()
{
   for(int i=0;i<6;i=i+2)
   {
      parr[i] = -parr[i];
   }

   polygon();
}

void menu(int ch)
{
      drawCorodinates();
   switch(ch)
   {
      case 1: polygon();
         break;

      case 2: reflection_origin();
         break;
      case 3: reflection_x();
         break;
      case 4: reflection_y();
         break;
   }
}


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

```
   glutInit(&argc,argv);
   glutInitWindowSize(500,500);
   glutInitWindowPosition(100,100);

   glutCreateWindow("2D Transformation");
   init();
   glutDisplayFunc(drawCorodinates);

   glutCreateMenu(menu);
      glutAddMenuEntry("1 Display Polygon",1);

      glutAddMenuEntry("2 Reflection About Origin",2);
      glutAddMenuEntry("3 Reflection About X-Axis",3);
      glutAddMenuEntry("4 Reflection About Y-Axiis",4);

   glutAttachMenu(GLUT_RIGHT_BUTTON);

   glutMainLoop();
   return 0;
}
```



3D Transformation

```
#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 BAEF
  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 GHDC
  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 CDAB
  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 EADH
  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 BFGC
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 GHDC
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 FEHG
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 CDAB
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 EADH
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 BFGC
   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();
      }
```

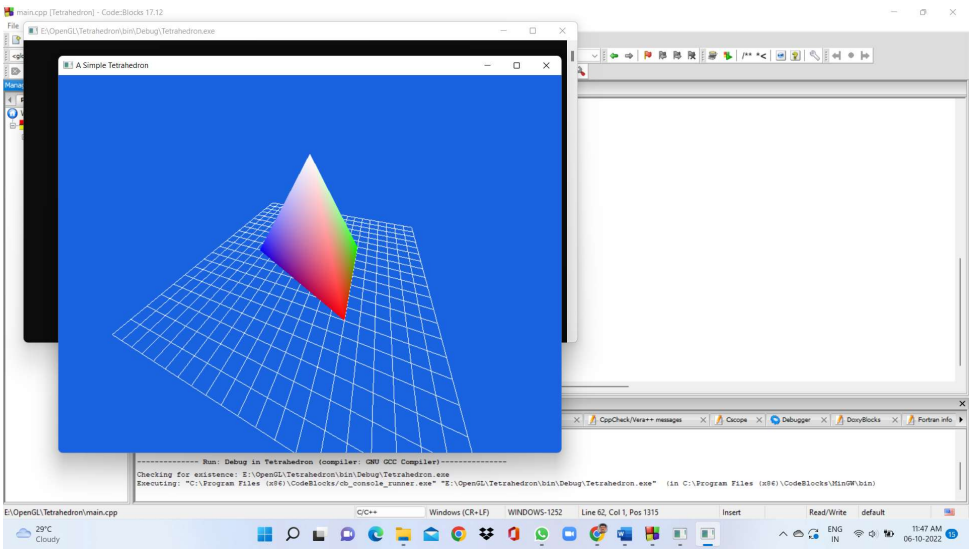
```
      case 2:
      {
         printf("\n\nSHEARING ABOUT Y-AXIS\n");
         for(i=0; i<8; i++)
         {
            xaux[i] = x[i]  + SHEAR[0] * y[i];
            yaux[i] = y[i];
            zaux[i] = z[i]  + SHEAR[2] * y[i];
         }
         display();
      }
      case 3:
      {
         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;
}
```

16. Program to displays a static picture of a tetrahedron sitting on a grid.

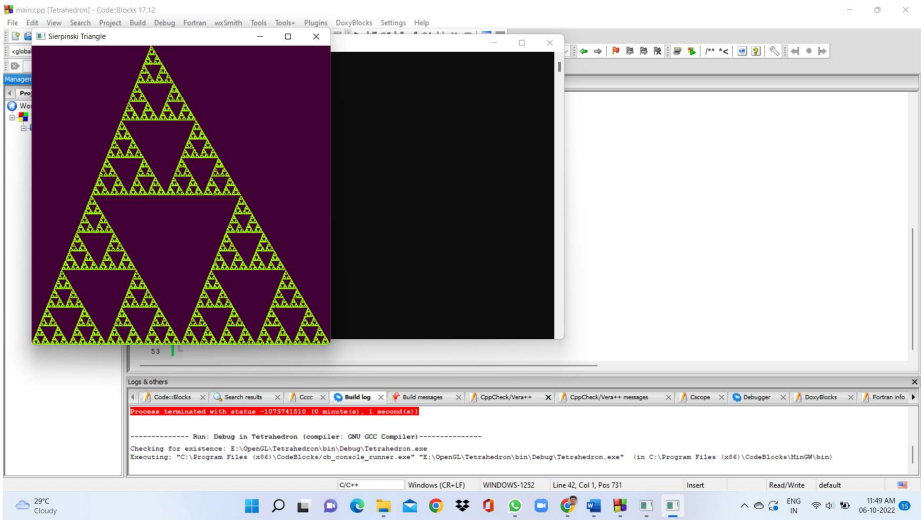
```
#include <GL/glut.h>

void display() {
 glClear(GL_COLOR_BUFFER_BIT);


 glColor3f(1.0, 1.0, 1.0);
 glBegin(GL_LINES);
 for (GLfloat i = -2.5; i <= 2.5; i += 0.25) {
  glVertex3f(i, 0, 2.5); glVertex3f(i, 0, -2.5);
  glVertex3f(2.5, 0, i); glVertex3f(-2.5, 0, i);
 }
 glEnd();

 glBegin(GL_TRIANGLE_STRIP);
  glColor3f(1, 1, 1); glVertex3f(0, 2, 0);
  glColor3f(1, 0, 0); glVertex3f(-1, 0, 1);
  glColor3f(0, 1, 0); glVertex3f(1, 0, 1);
  glColor3f(0, 0, 1); glVertex3f(0, 0, -1.4);
  glColor3f(1, 1, 1); glVertex3f(0, 2, 0);
  glColor3f(1, 0, 0); glVertex3f(-1, 0, 1);
 glEnd();

 glFlush();
}

void init() {


 glClearColor(0.1, 0.39, 0.88, 1.0);
 glColor3f(1.0, 1.0, 1.0);


 glEnable(GL_CULL_FACE);
```

```
 glCullFace(GL_BACK);


 glMatrixMode(GL_PROJECTION);
 glLoadIdentity();
 glFrustum(-2, 2, -1.5, 1.5, 1, 40);


 glMatrixMode(GL_MODELVIEW);
 glLoadIdentity();
 glTranslatef(0, 0, -3);
 glRotatef(50, 1, 0, 0);
 glRotatef(70, 0, 1, 0);
}


int main(int argc, char** argv) {
 glutInit(&argc, argv);
 glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
 glutInitWindowPosition(80, 80);
 glutInitWindowSize(800, 600);
 glutCreateWindow("A Simple Tetrahedron");
 glutDisplayFunc(display);
 init();
 glutMainLoop();
}
```



17. Program to displays a Fractals in 2D.

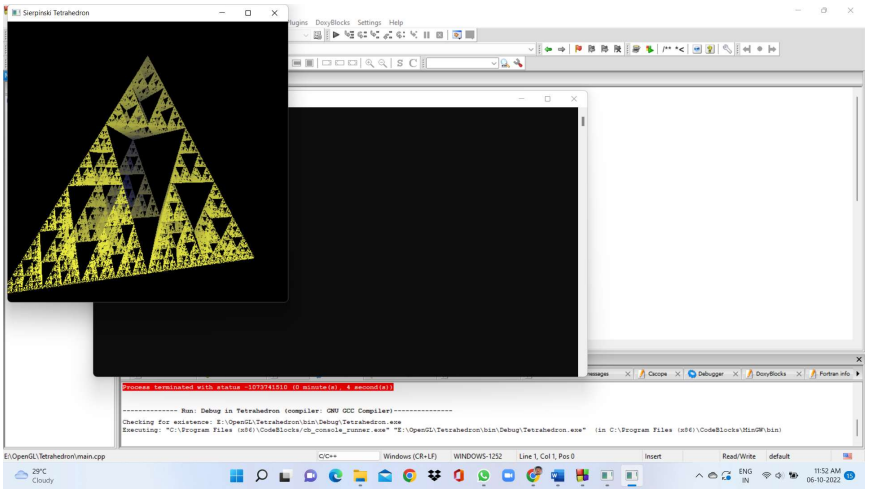
```
#include <GL/glut.h>

struct Point {
  GLfloat x, y;
  Point(GLfloat x = 0, GLfloat y = 0): x(x), y(y) {}
  Point midpoint(Point p) {return Point((x + p.x) / 2.0, (y + p.y) / 2.0);}
};


void display() {

  glClear(GL_COLOR_BUFFER_BIT);

  static Point vertices[] = {Point(0, 0), Point(200, 500), Point(500, 0)};


  static Point p = vertices[0];
  glBegin(GL_POINTS);
  for (int k = 0; k < 100000; k++) {
    p = p.midpoint(vertices[rand() % 3]);
    glVertex2f(p.x, p.y);
  }
  glEnd();
  glFlush();
}

void init() {

  glClearColor(0.25, 0.0, 0.2, 1.0);
  glColor3f(0.6, 1.0, 0.0);


  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(0.0, 500.0, 0.0, 500.0, 0.0, 1.0);
}


int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(40, 40);
  glutCreateWindow("Sierpinski Triangle");
  glutDisplayFunc(display);
  init();
  glutMainLoop();
}
```



18. Program to displays a Fractals in 3D.

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

struct Point {
  GLfloat x, y, z;
  Point(GLfloat x, GLfloat y, GLfloat z): x(x), y(y), z(z) {}
  Point midpoint(Point p) {return Point((x+p.x)/2, (y+p.y)/2, (z+p.z)/2);}
};

void reshape(GLint w, GLint h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(100.0, GLfloat(w)/GLfloat(h), 10.0, 1500.0);
}

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

void generateMorePoints() {

   static Point vertices[4] = {
    Point(-250, -225, -200),
    Point(-150, -225, -700),
    Point(250, -225, -275),
    Point(0, 450, -500)
  };
  static Point lastPoint = vertices[0];

  glBegin(GL_POINTS);
  for (int i = 0; i <= 500; i++) {
    lastPoint = lastPoint.midpoint(vertices[rand() % 4]);
    GLfloat intensity = (700 + lastPoint.z) / 500.0;
```

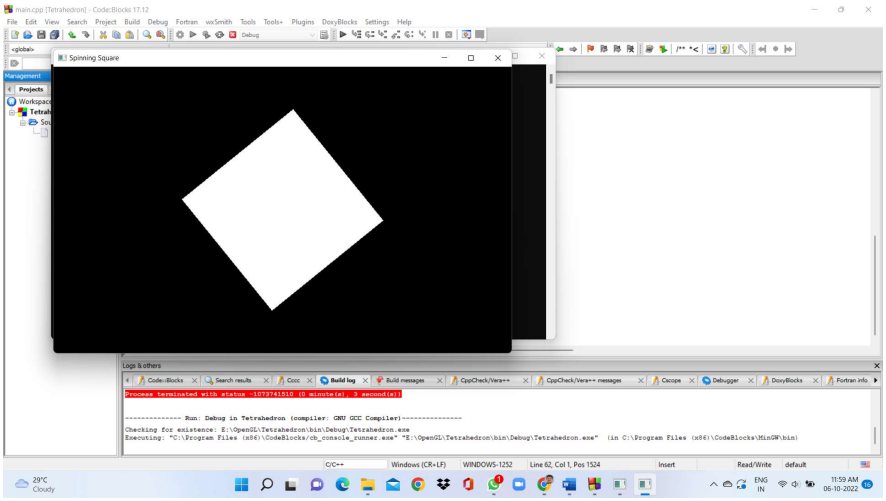
```
   glColor3f(intensity, intensity, 0.25);
   glVertex3f(lastPoint.x, lastPoint.y, lastPoint.z);
  }
  glEnd();
  glFlush();
}

void init() {
  glEnable(GL_DEPTH_TEST);
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(500, 500);
  glutInitWindowPosition(0, 0);
  glutCreateWindow("Sierpinski Tetrahedron");
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutIdleFunc(generateMorePoints);
  init();
  glutMainLoop();
}
```



19. Program to create a spinning square in a newly created window.

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

static bool spinning = true;
static const int FPS = 60;
static GLfloat currentAngleOfRotation = 0.0;

void reshape(GLint w, GLint h) {
  glViewport(0, 0, w, h);
  GLfloat aspect = (GLfloat)w / (GLfloat)h;
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  if (w <= h) {
```

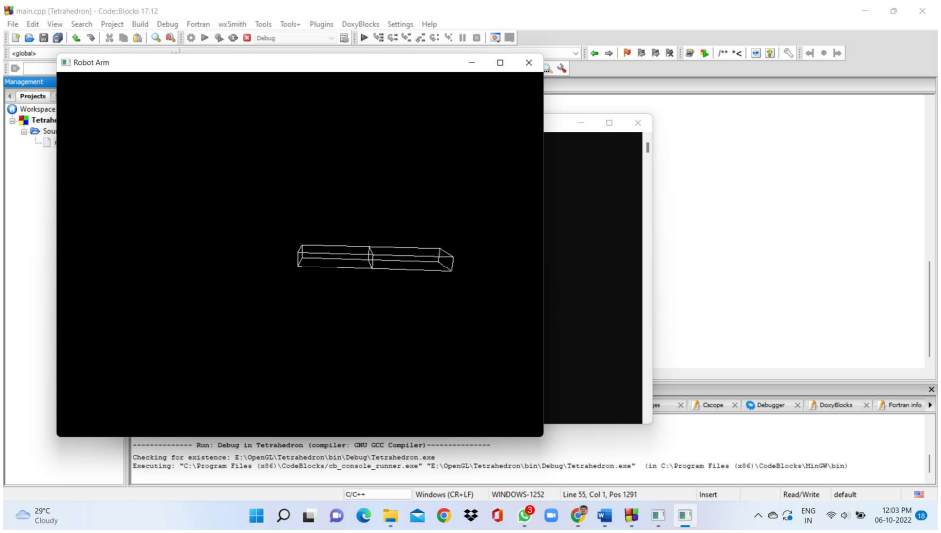
```
    glOrtho(-50.0, 50.0, -50.0/aspect, 50.0/aspect, -1.0, 1.0);
  } else {

    glOrtho(-50.0*aspect, 50.0*aspect, -50.0, 50.0, -1.0, 1.0);
  }
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glRotatef(currentAngleOfRotation, 0.0, 0.0, 1.0);
  glRectf(-25.0, -25.0, 25.0, 25.0);
  glFlush();
  glutSwapBuffers();
}

void timer(int v) {
  if (spinning) {
    currentAngleOfRotation += 1.0;
    if (currentAngleOfRotation > 360.0) {
      currentAngleOfRotation -= 360.0;
    }
    glutPostRedisplay();
  }
  glutTimerFunc(1000/FPS, timer, v);
}

void mouse(int button, int state, int x, int y) {
  if (button == GLUT_LEFT_BUTTON && state == GLUT_DOWN) {
    spinning = true;
  } else if (button == GLUT_RIGHT_BUTTON && state == GLUT_DOWN) {
    spinning = false;
  }
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowPosition(80, 80);
  glutInitWindowSize(800, 500);
  glutCreateWindow("Spinning Square");
  glutReshapeFunc(reshape);
  glutDisplayFunc(display);
  glutTimerFunc(100, timer, 0);
  glutMouseFunc(mouse);
  glutMainLoop();
}
```



20. Program to create interactive robot arm using OpenGL.

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

static int shoulderAngle = 0, elbowAngle = 0;

void special(int key, int, int) {
  switch (key) {
    case GLUT_KEY_LEFT: (elbowAngle += 5) %= 360; break;
    case GLUT_KEY_RIGHT: (elbowAngle -= 5) %= 360; break;
    case GLUT_KEY_UP: (shoulderAngle += 5) %= 360; break;
    case GLUT_KEY_DOWN: (shoulderAngle -= 5) %= 360; break;
    default: return;
  }
  glutPostRedisplay();
}

void wireBox(GLdouble width, GLdouble height, GLdouble depth) {
  glPushMatrix();
  glScalef(width, height, depth);
  glutWireCube(1.0);
  glPopMatrix();
}

void display() {

  glClear(GL_COLOR_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();

  glRotatef((GLfloat)shoulderAngle, 0.0, 0.0, 1.0);
  glTranslatef(1.0, 0.0, 0.0);
  wireBox(2.0, 0.4, 1.0);

  glTranslatef(1.0, 0.0, 0.0);
  glRotatef((GLfloat)elbowAngle, 0.0, 0.0, 1.0);
```

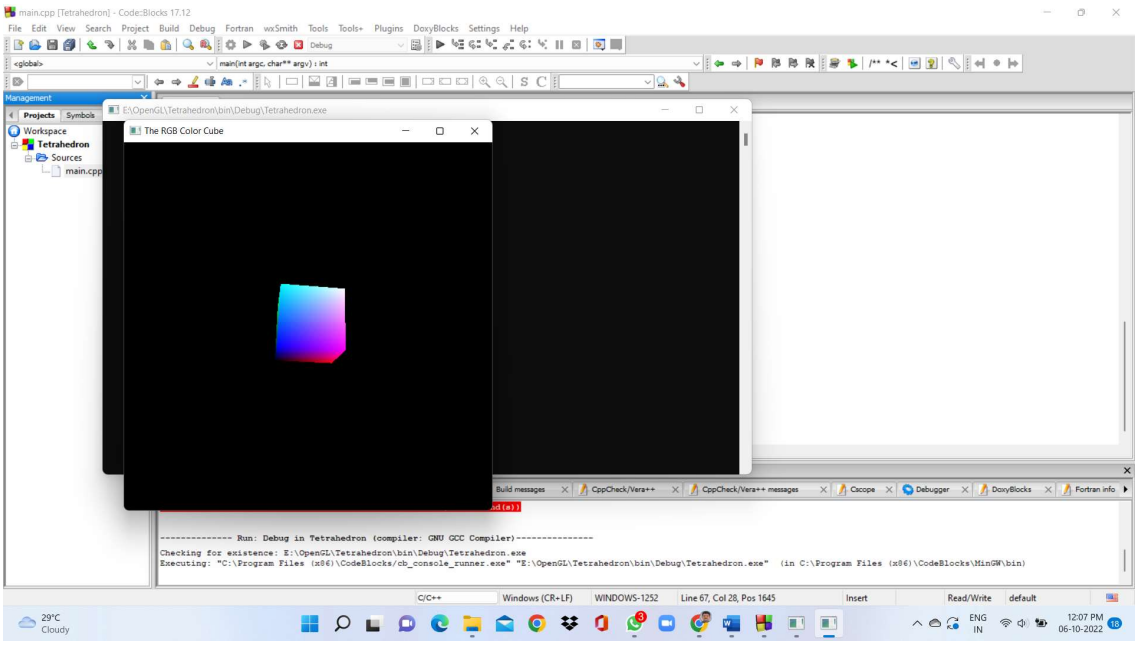
```
  glTranslatef(1.0, 0.0, 0.0);
  wireBox(2.0, 0.4, 1.0);

  glPopMatrix();
  glFlush();
}

void reshape(GLint w, GLint h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(65.0, GLfloat(w)/GLfloat(h), 1.0, 20.0);
}

void init() {
  glShadeModel(GL_FLAT);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(1,2,8, 0,0,0, 0,1,0);
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowPosition(80, 80);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Robot Arm");
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutSpecialFunc(special);
  init();
  glutMainLoop();
}
```



21. Program to create a flying cube using OpenGL

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

namespace Cube {

const int NUM_VERTICES = 8;
const int NUM_FACES = 6;

GLint vertices[NUM_VERTICES][3] = {
  {0, 0, 0}, {0, 0, 1}, {0, 1, 0}, {0, 1, 1},
  {1, 0, 0}, {1, 0, 1}, {1, 1, 0}, {1, 1, 1}};

GLint faces[NUM_FACES][4] = {
  {1, 5, 7, 3}, {5, 4, 6, 7}, {4, 0, 2, 6},
  {3, 7, 6, 2}, {0, 1, 3, 2}, {0, 4, 5, 1}};

GLfloat vertexColors[NUM_VERTICES][3] = {
  {0.0, 0.0, 0.0}, {0.0, 0.0, 1.0}, {0.0, 1.0, 0.0}, {0.0, 1.0, 1.0},
  {1.0, 0.0, 0.0}, {1.0, 0.0, 1.0}, {1.0, 1.0, 0.0}, {1.0, 1.0, 1.0}};

void draw() {
  glBegin(GL_QUADS);
  for (int i = 0; i < NUM_FACES; i++) {
    for (int j = 0; j < 4; j++) {
      glColor3fv((GLfloat*)&vertexColors[faces[i][j]]);
      glVertex3iv((GLint*)&vertices[faces[i][j]]);
    }
  }
  glEnd();
}
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  Cube::draw();
  glFlush();
  glutSwapBuffers();
}

void timer(int v) {
  static GLfloat u = 0.0;
  u += 0.01;
  glLoadIdentity();
  gluLookAt(8*cos(u), 7*cos(u)-1, 4*cos(u/3)+2, .5, .5, .5, cos(u), 1, 0);
  glutPostRedisplay();
  glutTimerFunc(1000/60.0, timer, v);
}

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

```
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0, GLfloat(w) / GLfloat(h), 0.5, 40.0);
  glMatrixMode(GL_MODELVIEW);
}

void init() {
  glEnable(GL_CULL_FACE);
  glCullFace(GL_BACK);
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
  glutInitWindowSize(500, 500);
  glutCreateWindow("The RGB Color Cube");
  glutReshapeFunc(reshape);
  glutTimerFunc(100, timer, 0);
  glutDisplayFunc(display);
  init();
  glutMainLoop();
}
```

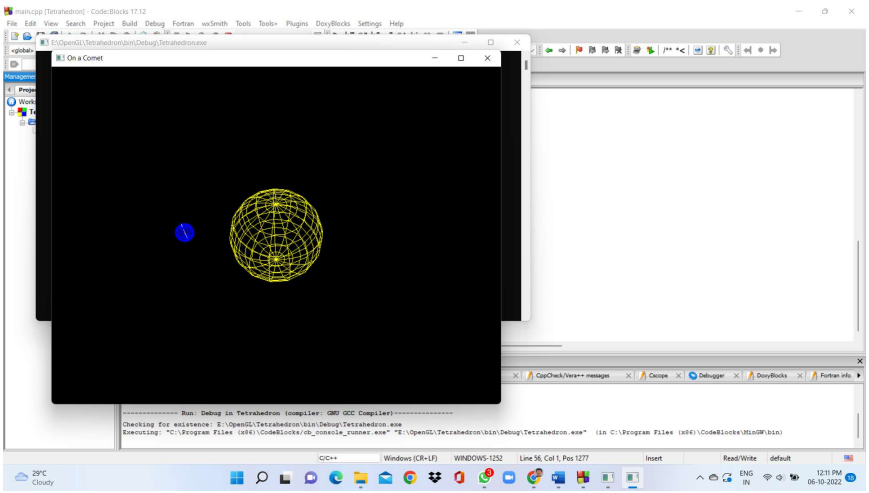


22. Program to create a solar system simulation from the point of view.

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

void myWireSphere(GLfloat radius, int slices, int stacks) {
  glPushMatrix();
  glRotatef(-90.0, 1.0, 0.0, 0.0);
  glutWireSphere(radius, slices, stacks);
  glPopMatrix();
```

```
}

static int year = 0, day = 0;

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

  glColor3f(1.0, 1.0, 0.0);
  myWireSphere(1.0, 15, 15);

  glRotatef((GLfloat)year, 0.0, 1.0, 0.0);
  glTranslatef (2.0, 0.0, 0.0);
  glRotatef((GLfloat)day, 0.0, 1.0, 0.0);
  glColor3f(0.0, 0.0, 1.0);
  myWireSphere(0.2, 15, 15);
  glColor3f(1, 1, 1);
  glBegin(GL_LINES);
    glVertex3f(0, -0.3, 0);
    glVertex3f(0, 0.3, 0);
  glEnd();

  glPopMatrix();
  glFlush();
  glutSwapBuffers();
}

static GLfloat u = 0.0;
static GLfloat du = 0.1;

void timer(int v) {
  u += du;
  day = (day + 1) % 360;
  year = (year + 2) % 360;
  glLoadIdentity();
  gluLookAt(20*cos(u/8.0)+12,5*sin(u/8.0)+1,10*cos(u/8.0)+2, 0,0,0, 0,1,0);
  glutPostRedisplay();
  glutTimerFunc(1000/60, timer, v);
}

void reshape(GLint w, GLint h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0, (GLfloat)w/(GLfloat)h, 1.0, 40.0);
  glMatrixMode(GL_MODELVIEW);
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode (GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowSize(800, 600);
  glutCreateWindow("On a Comet");
```

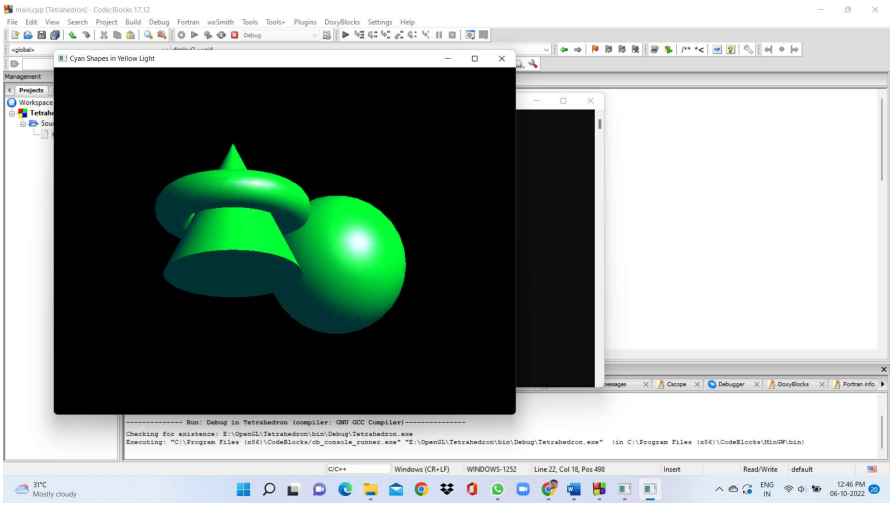
```
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutTimerFunc(100, timer, 0);
  glEnable(GL_DEPTH_TEST);
  glutMainLoop();
}
```



23. Program to create three cyan objects illuminated with yellow color.

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

void display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();

  glRotatef(-20.0, 1.0, 0.0, 0.0);

  glPushMatrix();
  glTranslatef(-0.75, 0.5, 0.0);
  glRotatef(90.0, 1.0, 0.0, 0.0);
  glutSolidTorus(0.275, 0.85, 16, 40);
  glPopMatrix();

  glPushMatrix();
  glTranslatef(-0.75, -0.5, 0.0);
  glRotatef(270.0, 1.0, 0.0, 0.0);
  glutSolidCone(1.0, 2.0, 70, 12);
  glPopMatrix();

  glPushMatrix();
  glTranslatef(0.75, 0.0, -1.0);
  glutSolidSphere(1.0, 30, 30);
  glPopMatrix();

  glPopMatrix();
  glFlush();
}
```

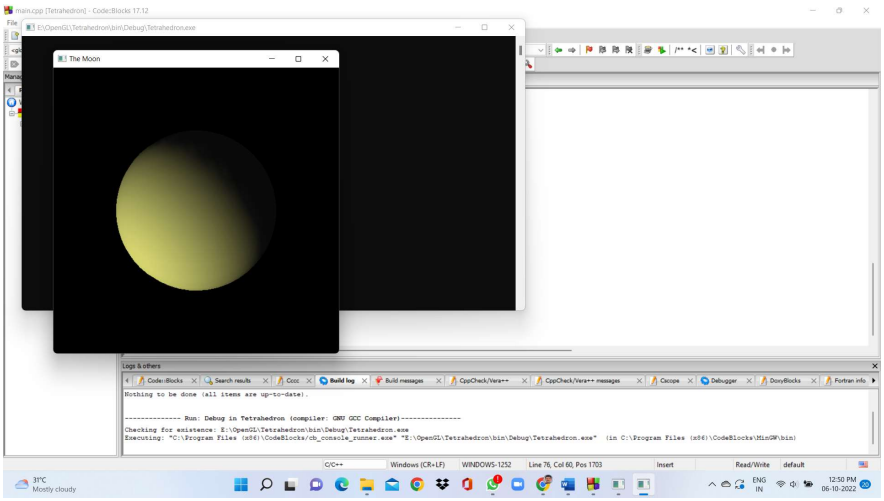
```
void reshape(GLint w, GLint h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  GLfloat aspect = GLfloat(w) / GLfloat(h);
  glLoadIdentity();
  if (w <= h) {
    // width is smaller, so stretch out the height
    glOrtho(-2.5, 2.5, -2.5/aspect, 2.5/aspect, -10.0, 10.0);
  } else {
    // height is smaller, so stretch out the width
    glOrtho(-2.5*aspect, 2.5*aspect, -2.5, 2.5, -10.0, 10.0);
  }
}

void init() {
  GLfloat black[] = { 0.0, 0.0, 0.0, 1.0 };
  GLfloat yellow[] = { 1.0, 1.0, 0.0, 1.0 };
  GLfloat cyan[] = { 0.0, 1.0, 1.0, 1.0 };
  GLfloat white[] = { 1.0, 1.0, 1.0, 1.0 };
  GLfloat direction[] = { 1.0, 1.0, 1.0, 0.0 };

  glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, cyan);
  glMaterialfv(GL_FRONT, GL_SPECULAR, white);
  glMaterialf(GL_FRONT, GL_SHININESS, 30);

  glLightfv(GL_LIGHT0, GL_AMBIENT, black);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, yellow);
  glLightfv(GL_LIGHT0, GL_SPECULAR, white);
  glLightfv(GL_LIGHT0, GL_POSITION, direction);

  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  glEnable(GL_DEPTH_TEST);
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowPosition(80, 80);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Cyan Shapes in Yellow Light");
  glutReshapeFunc(reshape);
  glutDisplayFunc(display);
  init();
  glutMainLoop();
}
```



24. Program to simulate the phases of moon.

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

class Moon {
  int displayListId;
public:
  void create() {
    displayListId = glGenLists(1);
    glNewList(displayListId, GL_COMPILE);
    GLfloat direction[] = {-1.0, -1.0, -1.0, 0.0};
    glLightfv(GL_LIGHT0, GL_POSITION, direction);
    glutSolidSphere(1.0, 25, 25);
    glEndList();
  }
  void draw() {
    glCallList(displayListId);
  }
};

static Moon moon;

class Orbiter {
  double radius;
  double u;
public:
  Orbiter(double radius): radius(radius), u(0.0) {}
  void advance(double delta) {u += delta;}
  void getPosition(double& x, double& y, double& z) {
    x = radius * cos(u);
    y = 0;
    z = radius * sin(u);
  }
```

```
};

// The one and only orbiter.
static Orbiter orbiter(5.0);

void display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glMatrixMode(GL_MODELVIEW);
  glPushMatrix();
  glLoadIdentity();
  double x, y, z;
  orbiter.getPosition(x, y, z);
  gluLookAt(x, y, z, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
  moon.draw();
  glPopMatrix();
  glutSwapBuffers();
}

void timer(int v) {
  orbiter.advance(0.01);
  glutPostRedisplay();
  glutTimerFunc(1000/60, timer, v);
}

void reshape(GLint w, GLint h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0, GLfloat(w) / GLfloat(h), 1.0, 10.0);
}

void init() {
  glEnable(GL_DEPTH_TEST);
  GLfloat yellow[] = {1.0, 1.0, 0.5, 1.0};
  glLightfv(GL_LIGHT0, GL_DIFFUSE, yellow);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  moon.create();
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowPosition(80, 80);
  glutInitWindowSize(500, 500);
  glutCreateWindow("The Moon");
  glutDisplayFunc(display);
  glutTimerFunc(100, timer, 0);
  glutReshapeFunc(reshape);
  init();
  glutMainLoop();
}
```



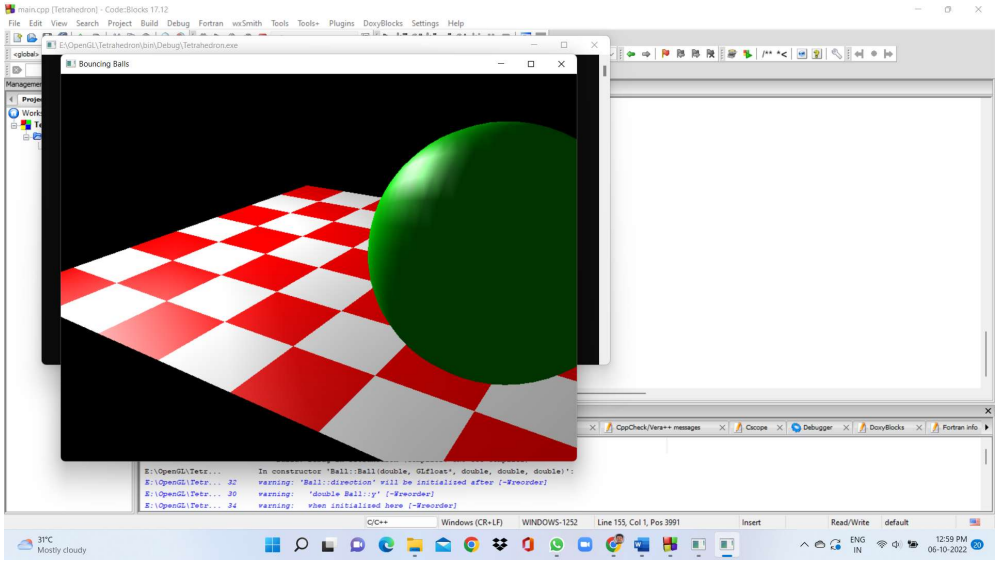
25. Program to draw a bouncing ball and perform animations using OpenGL.

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

GLfloat WHITE[] = {1, 1, 1};
GLfloat RED[] = {1, 0, 0};
GLfloat GREEN[] = {0, 1, 0};
GLfloat MAGENTA[] = {1, 0, 1};

class Camera {
  double theta;
  double y;
  double dTheta;
  double dy;
public:
  Camera(): theta(0), y(3), dTheta(0.04), dy(0.2) {}
  double getX() {return 10 * cos(theta);}
  double getY() {return y;}
  double getZ() {return 10 * sin(theta);}
  void moveRight() {theta += dTheta;}
  void moveLeft() {theta -= dTheta;}
  void moveUp() {y += dy;}
  void moveDown() {if (y > dy) y -= dy;}
};

class Ball {
  double radius;
  GLfloat* color;
  double maximumHeight;
  double x;
  double y;
  double z;
  int direction;
public:
  Ball(double r, GLfloat* c, double h, double x, double z):
```

```
    radius(r), color(c), maximumHeight(h), direction(-1),
    y(h), x(x), z(z) {
  }
  void update() {
    y += direction * 0.05;
    if (y > maximumHeight) {
      y = maximumHeight; direction = -1;
    } else if (y < radius) {
      y = radius; direction = 1;
    }
    glPushMatrix();
    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, color);
    glTranslated(x, y, z);
    glutSolidSphere(radius, 30, 30);
    glPopMatrix();
  }
};

class Checkerboard {
  int displayListId;
  int width;
  int depth;
public:
  Checkerboard(int width, int depth): width(width), depth(depth) {}
  double centerx() {return width / 2;}
  double centerz() {return depth / 2;}
  void create() {
    displayListId = glGenLists(1);
    glNewList(displayListId, GL_COMPILE);
    GLfloat lightPosition[] = {4, 3, 7, 1};
    glLightfv(GL_LIGHT0, GL_POSITION, lightPosition);
    glBegin(GL_QUADS);
    glNormal3d(0, 1, 0);
    for (int x = 0; x < width - 1; x++) {
      for (int z = 0; z < depth - 1; z++) {
        glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE,
                (x + z) % 2 == 0 ? RED : WHITE);
        glVertex3d(x, 0, z);
        glVertex3d(x+1, 0, z);
        glVertex3d(x+1, 0, z+1);
        glVertex3d(x, 0, z+1);
      }
    }
    glEnd();
    glEndList();
  }
  void draw() {
    glCallList(displayListId);
  }
};

Checkerboard checkerboard(8, 8);
```

```
Camera camera;
Ball balls[] = {
  Ball(1, GREEN, 7, 6, 1),
  Ball(1.5, MAGENTA, 6, 3, 4),
  Ball(0.4, WHITE, 5, 1, 7)
};

void init() {
  glEnable(GL_DEPTH_TEST);
  glLightfv(GL_LIGHT0, GL_DIFFUSE, WHITE);
  glLightfv(GL_LIGHT0, GL_SPECULAR, WHITE);
  glMaterialfv(GL_FRONT, GL_SPECULAR, WHITE);
  glMaterialf(GL_FRONT, GL_SHININESS, 30);
  glEnable(GL_LIGHTING);
  glEnable(GL_LIGHT0);
  checkerboard.create();
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  gluLookAt(camera.getX(), camera.getY(), camera.getZ(),
          checkerboard.centerx(), 0.0, checkerboard.centerz(),
          0.0, 1.0, 0.0);
  checkerboard.draw();
  for (int i = 0; i < sizeof balls / sizeof(Ball); i++) {
    balls[i].update();
  }
  glFlush();
  glutSwapBuffers();
}

void reshape(GLint w, GLint h) {
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(40.0, GLfloat(w) / GLfloat(h), 1.0, 150.0);
  glMatrixMode(GL_MODELVIEW);
}

void timer(int v) {
  glutPostRedisplay();
  glutTimerFunc(1000/60, timer, v);
}

void special(int key, int, int) {
  switch (key) {
    case GLUT_KEY_LEFT: camera.moveLeft(); break;
    case GLUT_KEY_RIGHT: camera.moveRight(); break;
    case GLUT_KEY_UP: camera.moveUp(); break;
    case GLUT_KEY_DOWN: camera.moveDown(); break;
  }
```

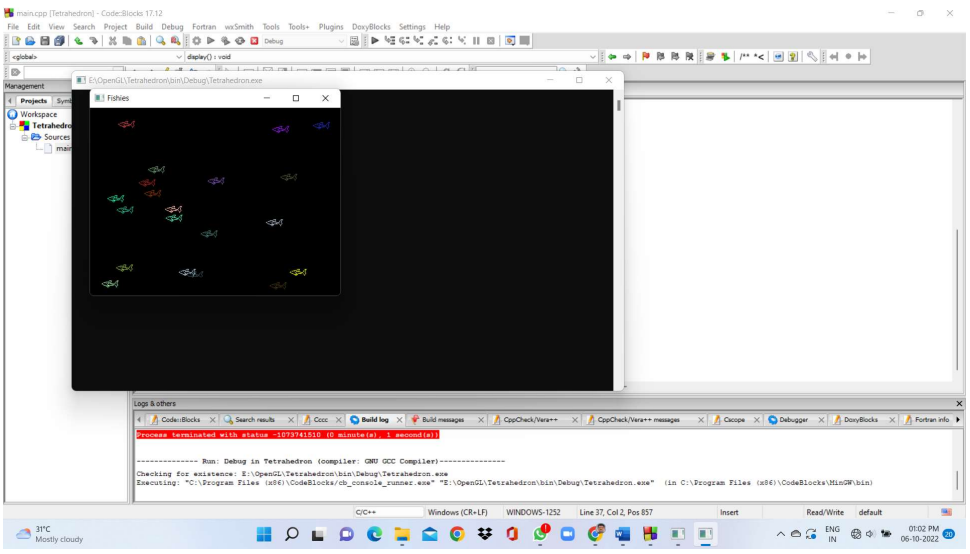
```
  glutPostRedisplay();
}

int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
  glutInitWindowPosition(80, 80);
  glutInitWindowSize(800, 600);
  glutCreateWindow("Bouncing Balls");
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutSpecialFunc(special);
  glutTimerFunc(100, timer, 0);
  init();
  glutMainLoop();
}
```



26. Program to create fish bitmaps using OpenGL.

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

GLubyte fish[] = {
  0x00, 0x60, 0x01, 0x00,
  0x00, 0x90, 0x01, 0x00,
  0x03, 0xf8, 0x02, 0x80,
  0x1c, 0x37, 0xe4, 0x40,
  0x20, 0x40, 0x90, 0x40,
  0xc0, 0x40, 0x78, 0x80,
  0x41, 0x37, 0x84, 0x80,
  0x1c, 0x1a, 0x04, 0x80,
  0x03, 0xe2, 0x02, 0x40,
  0x00, 0x11, 0x01, 0x40,
  0x00, 0x0f, 0x00, 0xe0,
```

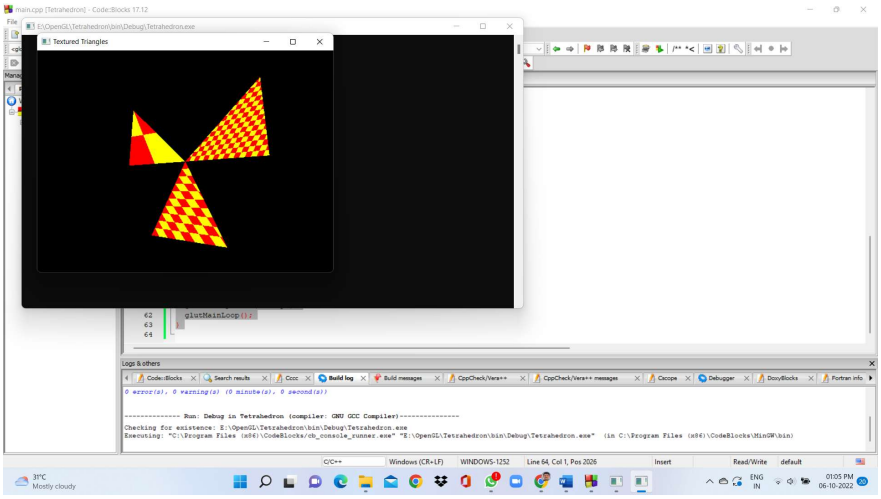
```
};

GLfloat randomFloat() {
  return (GLfloat)rand() / RAND_MAX;
}

void reshape(int width, int height) {
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0, 1, 0, 1);
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  for (int i = 0; i < 20; i++) {
    glColor3f(randomFloat(), randomFloat(), randomFloat());
    glRasterPos3f(randomFloat(), randomFloat(), 0.0);
    glBitmap(27, 11, 0, 0, 0, 0, fish);
  }
  glFlush();
}

int main(int argc, char **argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_RGB | GLUT_SINGLE);
  glutInitWindowSize(400, 300);
  glutCreateWindow("Fishies");
  glutReshapeFunc(reshape);
  glutDisplayFunc(display);
  glutMainLoop();
}
```



27. Program to create trivial illustration of checked triangles for texture mapping.

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

#define red {0xff, 0x00, 0x00}
#define yellow {0xff, 0xff, 0x00}
#define magenta {0xff, 0, 0xff}
GLubyte texture[][3] = {
   red, yellow,
   yellow, red,
};


void reshape(int width, int height) {
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(80, GLfloat(width)/height, 1, 40);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  gluLookAt(2, -1, 5, 0, 0, 0, 0, 1, 0);
  glEnable(GL_TEXTURE_2D);
  glPixelStorei(GL_UNPACK_ALIGNMENT, 1);
  glTexImage2D(GL_TEXTURE_2D,
            0,                // level 0
            3,                // use only R, G, and B components
            2, 2,              // texture has 2x2 texels
            0,                // no border
            GL_RGB,             // texels are in RGB format
            GL_UNSIGNED_BYTE,     // color components are unsigned bytes
            texture);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
  glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
}

void display() {
  glClear(GL_COLOR_BUFFER_BIT);
  glBegin(GL_TRIANGLES);
    glTexCoord2f(0.5, 1.0);    glVertex2f(-3, 3);
    glTexCoord2f(0.0, 0.0);    glVertex2f(-3, 0);
    glTexCoord2f(1.0, 0.0);    glVertex2f(0, 0);

    glTexCoord2f(4, 8);        glVertex2f(3, 3);
    glTexCoord2f(0.0, 0.0);    glVertex2f(0, 0);
    glTexCoord2f(8, 0.0);      glVertex2f(3, 0);

    glTexCoord2f(5, 5);        glVertex2f(0, 0);
    glTexCoord2f(0.0, 0.0);    glVertex2f(-1.5, -3);
    glTexCoord2f(4, 0.0);      glVertex2f(1.5, -3);
  glEnd();
  glFlush();
```

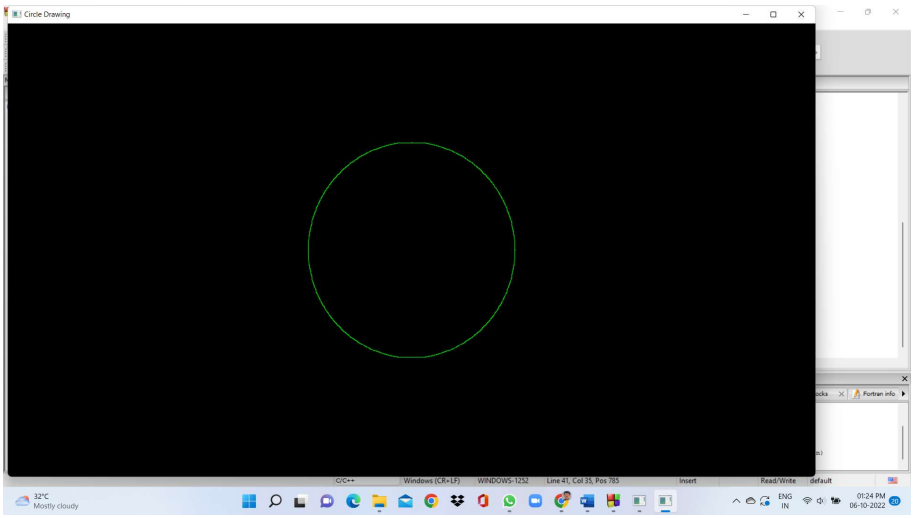
```
}

// Initializes GLUT and enters the main loop.
int main(int argc, char** argv) {
  glutInit(&argc, argv);
  glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(520, 390);
  glutCreateWindow("Textured Triangles");
  glutDisplayFunc(display);
  glutReshapeFunc(reshape);
  glutMainLoop();
}
```



28. Program to draw a circle in a window using OpenGL.

```
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
#define pi 3.142857


void myInit (void)
{
   glClearColor(0.0, 0.0, 0.0, 1.0);
   glColor3f(0.0, 1.0, 0.0);
   glPointSize(1.0);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();

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

void display (void)
{
   glClear(GL_COLOR_BUFFER_BIT);
   glBegin(GL_POINTS);
   float x, y, i;
```

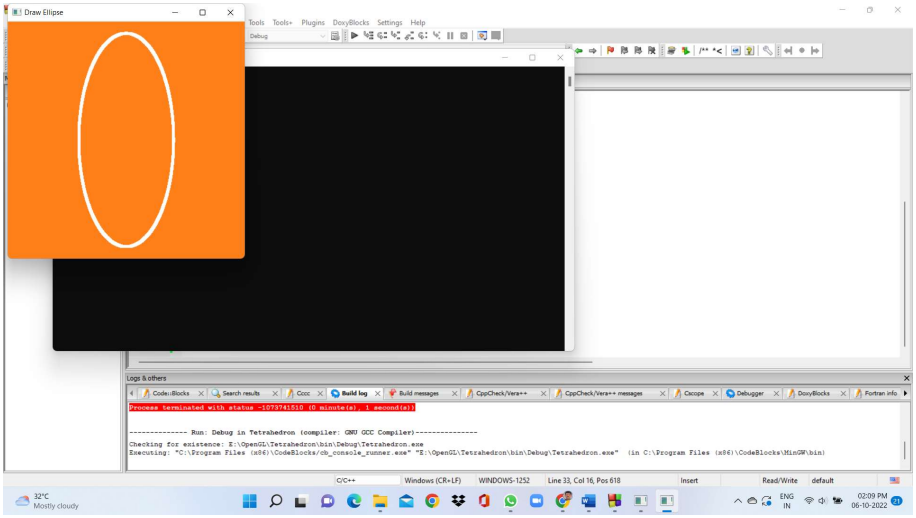
```
    for ( i = 0; i < (2 * pi); i += 0.001)
    {
       x = 200 * cos(i);
       y = 200 * sin(i);

       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("Circle Drawing");
    myInit();

    glutDisplayFunc(display);
    glutMainLoop();
}
```



29. Program to draw an ellipse in a newly created window.

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

void Ellipse() {
GLfloat xi, yi, theta = 0;
```

```
GLfloat x_c = 0, y_c = 0, r_x = 0.8, r_y = 1.8;
int COUNT;
glClear(GL_COLOR_BUFFER_BIT);
for (COUNT = 1 ; COUNT <= 10000 ; COUNT++) {
theta = theta + 0.001;
xi = x_c + r_x*cos(theta);
yi = y_c + r_y*sin(theta);

glBegin(GL_POINTS);
glVertex2f(xi, yi);
glEnd();
}
glFlush();
}

void Initial() {
glClearColor(1.0, 0.5, 0.1, 0);
glColor3f(1,1,1);
glPointSize(5.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(-2, +2, -2, +2);
}

int main(int c, char *v[])
{
glutInit(&c,v);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(400, 400);
glutInitWindowPosition(0, 0);
glutCreateWindow("Draw Ellipse");
Initial();
glutDisplayFunc(Ellipse);
glutMainLoop();
return 0;
}
```



30. Program to create a hut with front facing view.

```
#include <GL\glut.h>
#include <iostream>
#include <windows.h>
using namespace std;

// Function to initialize the drivers
void myInit(void)
{
   // Clear all the screen color
   glClearColor(1.0, 0.5, 0.0, 1.0);

   // Sets background color to orang
   glMatrixMode(GL_PROJECTION);

   glLoadIdentity();

   // Specify the display area
   gluOrtho2D(0.0, 400.0, 0.0, 400.0);
}

// Function to display the Hut like
// structure on the console
void myDisplay(void)
{
   // Clear the screen buffer
   glClear(GL_COLOR_BUFFER_BIT);

   glPointSize(4.0);

   // Rectangular part of hut
   glColor3f(0.5f, 0.5f, 0.5f);

   // Begin the polygon
   glBegin(GL_POLYGON);

   // Create the polygon
   glVertex2i(40, 40);
   glVertex2i(320, 40);

   glVertex2i(40, 200);
   glVertex2i(320, 200);

   glVertex2i(40, 200);
   glVertex2i(40, 40);

   glVertex2i(320, 200);
   glVertex2i(320, 40);

   glEnd();

   // Right Window Update
```

```
glColor3f(1.0f, 0.0f, 0.0f);

// Begin the polygon
glBegin(GL_POLYGON);

// Create the polygon
glVertex2i(220, 60);
glVertex2i(300, 60);

glVertex2i(220, 150);
glVertex2i(300, 150);

glVertex2i(220, 60);
glVertex2i(220, 150);

glVertex2i(300, 150);
glVertex2i(300, 60);

glEnd();

// Right Window Update part 2
glColor3f(1.0f, 0.0f, 0.0f);

// Begin the polygon
glBegin(GL_POLYGON);

// Create the polygon
glVertex2i(220, 170);
glVertex2i(300, 170);

glVertex2i(220, 190);
glVertex2i(300, 190);

glVertex2i(220, 170);
glVertex2i(220, 190);

glVertex2i(300, 190);
glVertex2i(300, 170);

glEnd();

// Door
glColor3f(0.60f, 0.42f, 0.16f);

// Begin the polygon
glBegin(GL_POLYGON);

// Create the polygon
glVertex2i(130, 40);
glVertex2i(130, 160);

glVertex2i(130, 160);
```

```
    glVertex2i(180, 160);

    glVertex2i(180, 100);
    glVertex2i(180, 40);

    glVertex2i(120, 40);
    glVertex2i(170, 40);

    glEnd();

    // Create Door Part 2
    glColor3f(0.60f, 0.42f, 0.16f);

    // Begin the polygon
    glBegin(GL_POLYGON);

    // Create the polygon
    glVertex2i(130, 170);
    glVertex2i(130, 180);

    glVertex2i(130, 180);
    glVertex2i(180, 180);

    glVertex2i(180, 170);
    glVertex2i(180, 180);

    glVertex2i(130, 170);
    glVertex2i(180, 170);

    glEnd();

    // Hut's top triangle part
    glColor3f(1.0f, 0.0f, 1.0f);

    // Begin the polygon
    glBegin(GL_POLYGON);

    // Create the polygon
    glVertex2i(10, 200);
    glVertex2i(340, 200);

    glVertex2i(200, 390);

    glVertex2i(10, 200);
    glVertex2i(200, 390);

    glEnd();

    // Sends all output to display
    glFlush();
}
```

```
// Driver Code
int main(int argc, char** argv)
{
    // Initialize the init function
    glutInit(&argc, argv);

    // Initialize the toolkit;
    glutInitDisplayMode(
        GLUT_SINGLE | GLUT_RGB);

    // Sets the display mode and
    // specify the colour scheme
    glutInitWindowSize(1200, 740);

    // Specify the window size
    glutInitWindowPosition(0, 0);

    // Sets the starting position
    // for the window
    glutCreateWindow("Basic hut like"
                     " structure");

    // Creates the window and
    // sets the title
    glutDisplayFunc(myDisplay);
    myInit();

    // Additional initializations
    glutMainLoop();

    // Go into a loop until event
    // occurs
    return 0;
}
```