LABORATORY MANUAL

21CSL66

COMPUTER GRAPHICS & IMAGE PROCESSING LABORATORY

2024-2025

PREPARED BY

Prof. Keerthi P

COMPUTER SCIENCE AND ENGINEERING

IMPACT COLLEGE OF ENGINEERING & APPLIED SCIENCES

Bangalore – 560 092

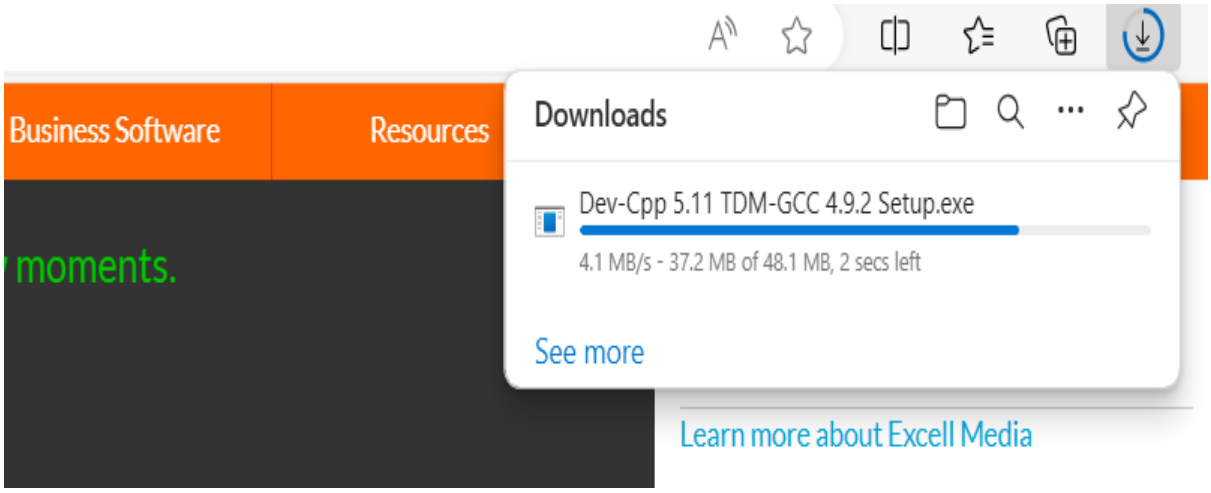
DEV C++ AND FREEGLUT INSTALLATION

A sample program execution

INSTALLATION OF DEV C++:

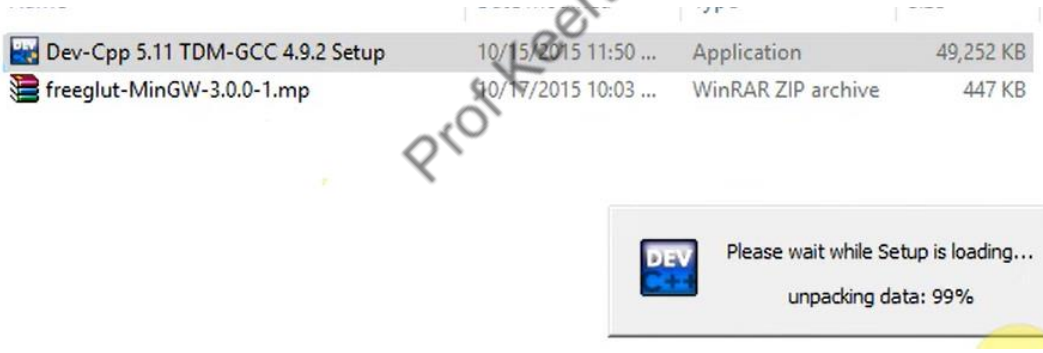
Visit the source forge Site check the below link

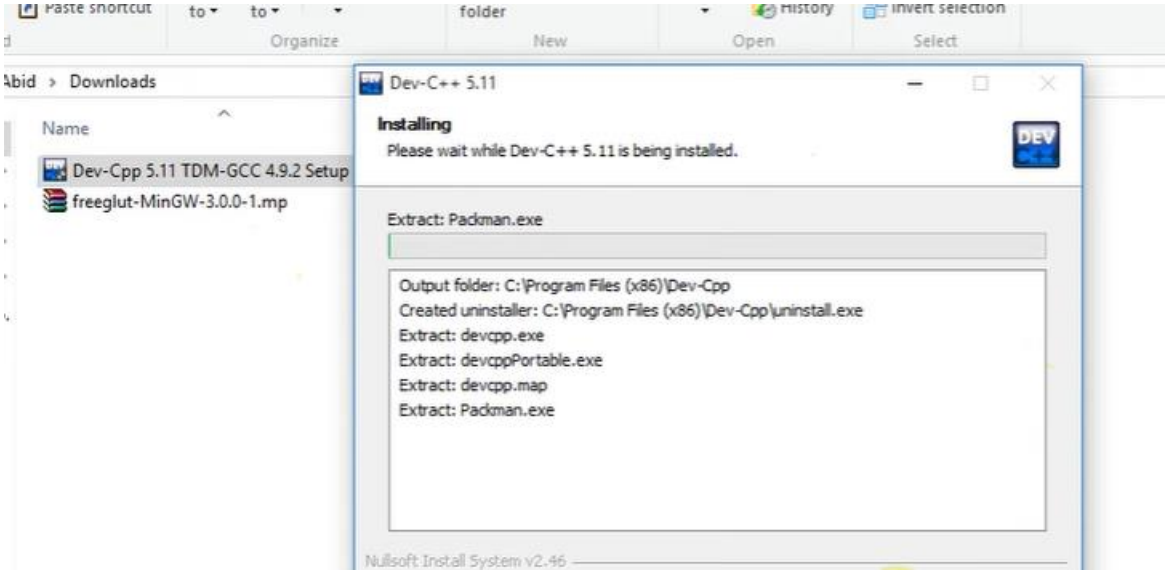
Click the download button….

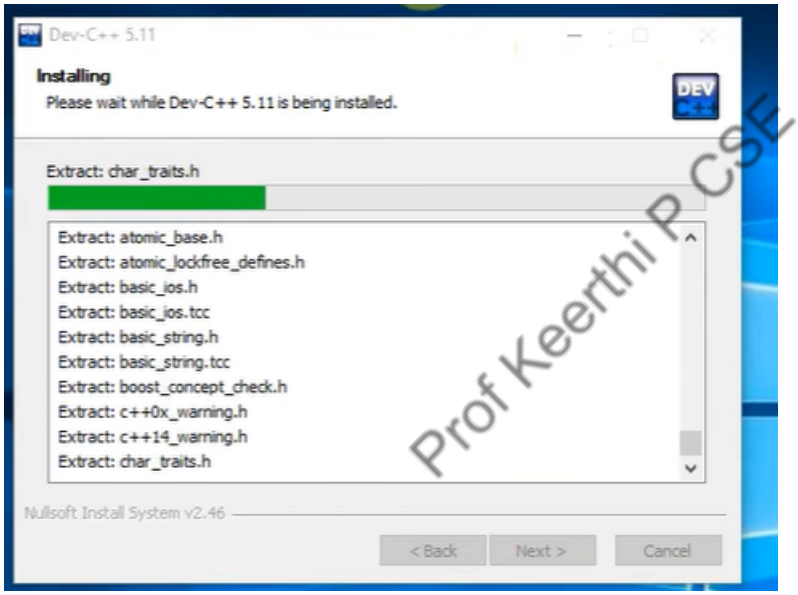


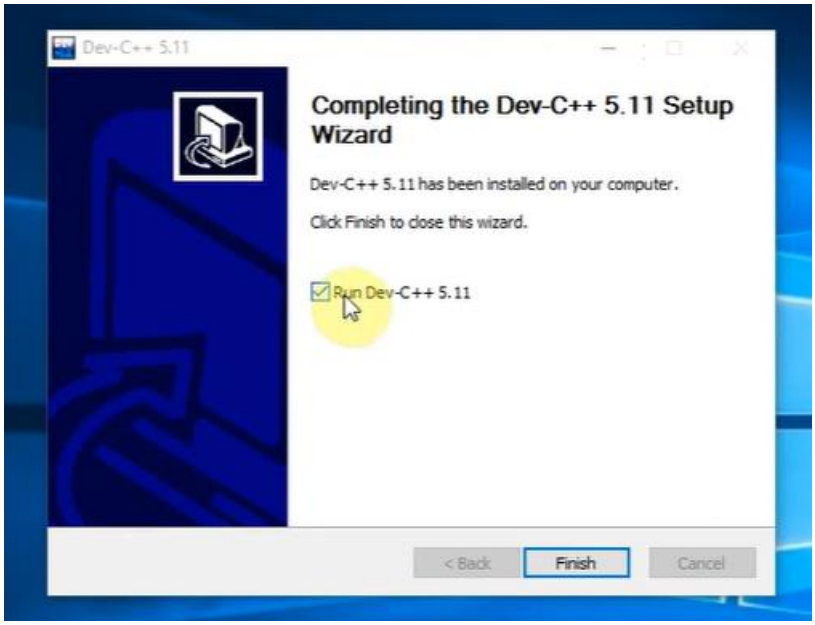
X

Once downloaded install



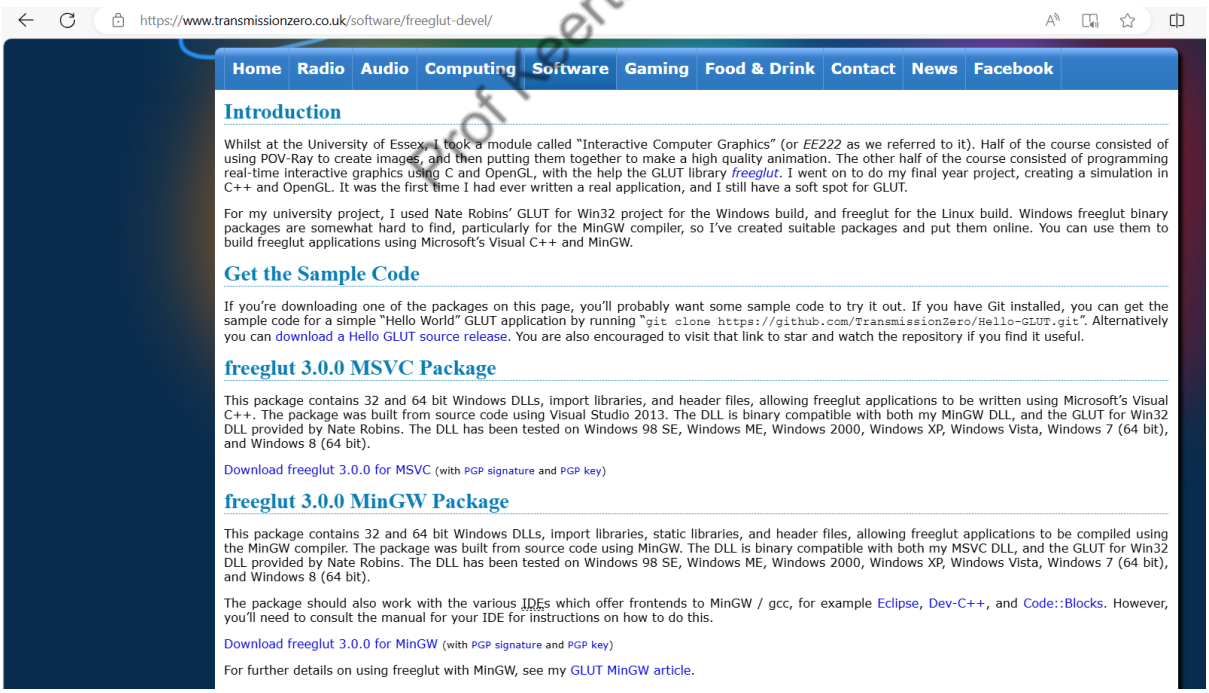

Paste shortcut
to
to
Organize
folder
New
History
Invert selection
Open
Select
Abid › Downloads
Name
Dev-Cpp 5.11 TDM-GCC 4.9.2 Setup
freeglut-MinGW-3.0.0-1.mp
Dev-C++ 5.11
Installing
Please wait while Dev-C++ 5.11 is being installed.
Extract: Packman.exe
Output folder: C:\Program Files (x86)\Dev-Cpp
Created uninstaller: C:\Program Files (x86)\Dev-Cpp\uninstall.exe
Extract: devcpp.exe
Extract: devcppPortable.exe
Extract: devcpp.map
Extract: Packman.exe
Nullsoft Install System v2.46


Dev-C++ 5.11
Installing
Please wait while Dev-C++ 5.11 is being installed.
Extract: char_traits.h
Extract: atomic_base.h
Extract: atomic_lockfree_defines.h
Extract: basic_ios.h
Extract: basic_ios.tcc
Extract: basic_string.h
Extract: basic_string.tcc
Extract: boost_concept_check.h
Extract: c++0x_warning.h
Extract: c++14_warning.h
Extract: char_traits.h
Nullsoft Install System v2.46
< Back
Next >
Cancel
Prof Keerthi P CSE



FREEGLUT INSTALLATION:

VISIT THE BELOW SITE



CLICK download free glut 3.0.0 package

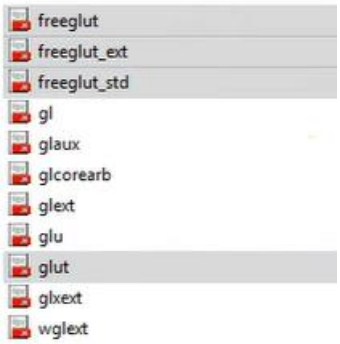
Once downloaded…

1.copy the files

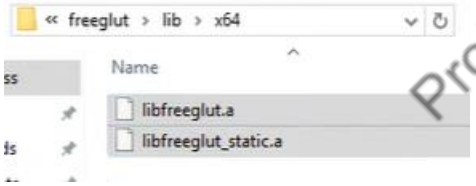
from C:\Users\keerthi\Downloads\freeglut-3.4.0.tar.gz\freeglut-3.4.0\include\GL

Paste in

C:\Program Files (x86)\Dev-Cpp\MinGW64\x86_64-w64-mingw32\include\GL



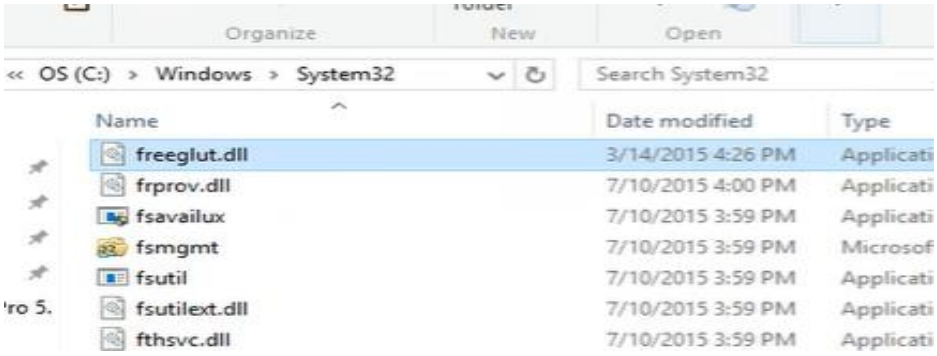
2.In free glut path copy the files



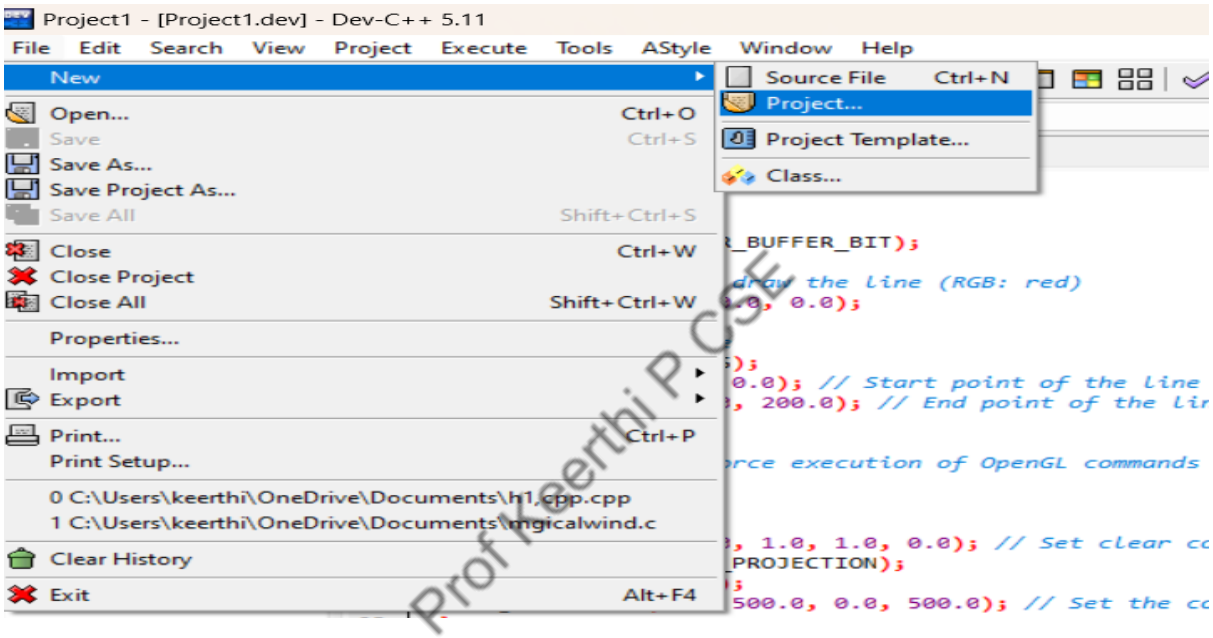
Paste in C:\Program Files (x86)\Dev-Cpp\MinGW64\x86_64-w64-mingw32\lib

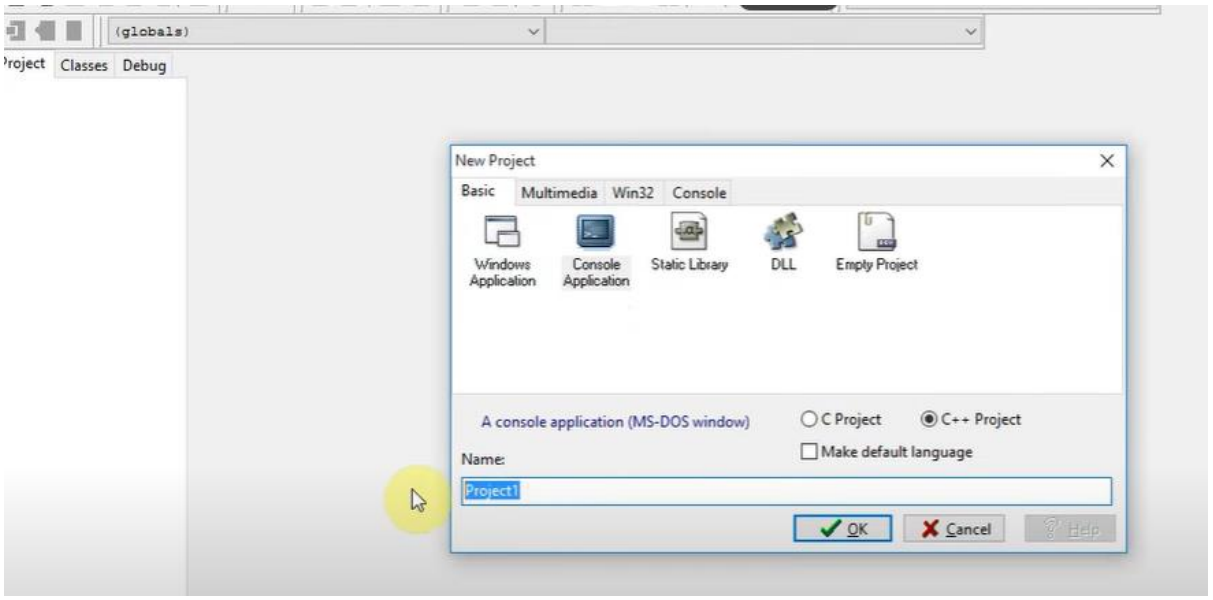
3.copy freeglut.dll from freeglut/bin path

Paste in C:\Windows\System32 path



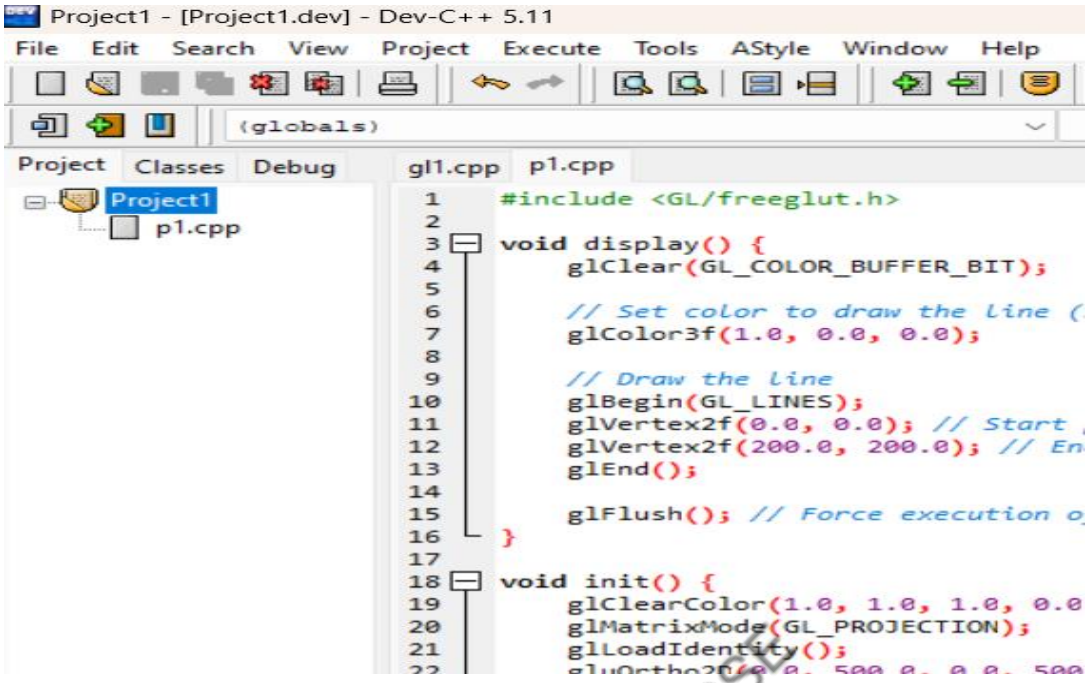
4.open dev c++ create new project



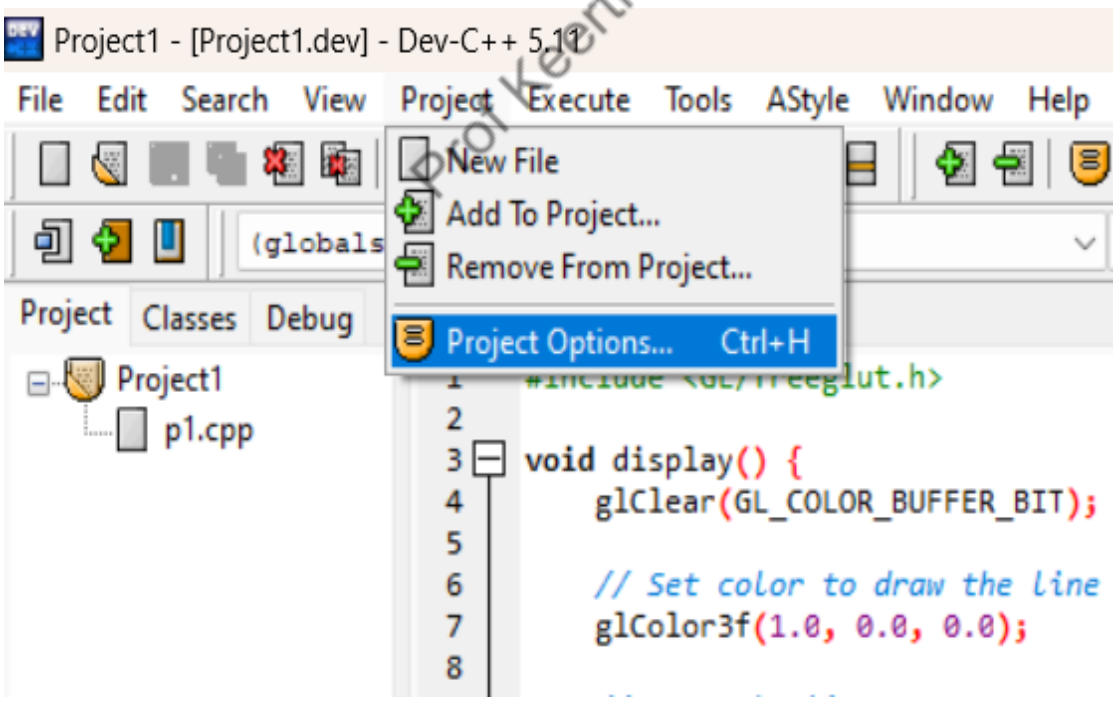


5.In console application type the name of your project click ok

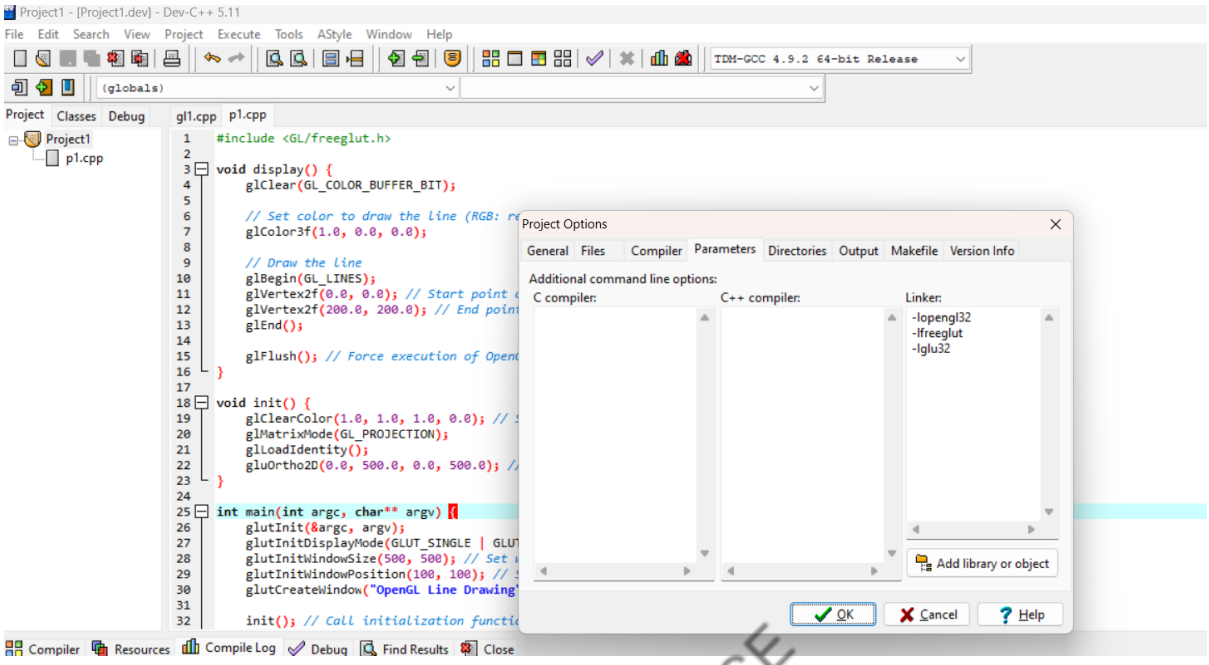
Create cpp program->**p1.cpp** under project1 package



Go to project options



6.In linker option type the following commands



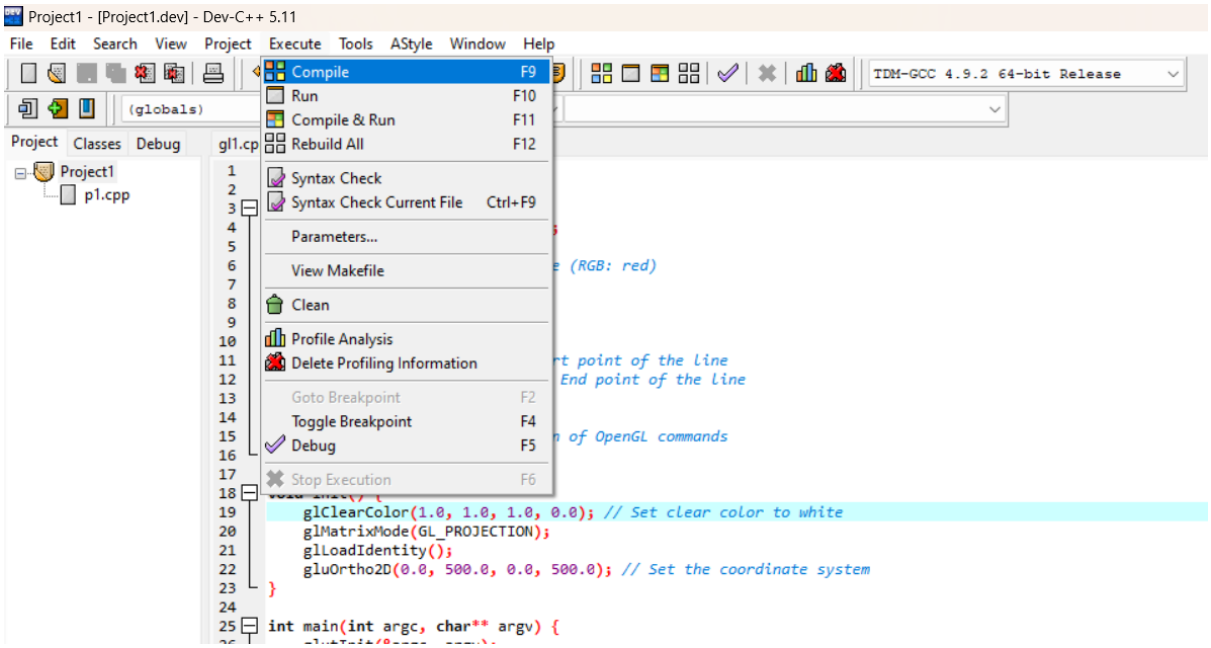
Click ok

7.Copy/type the program and save in the following path

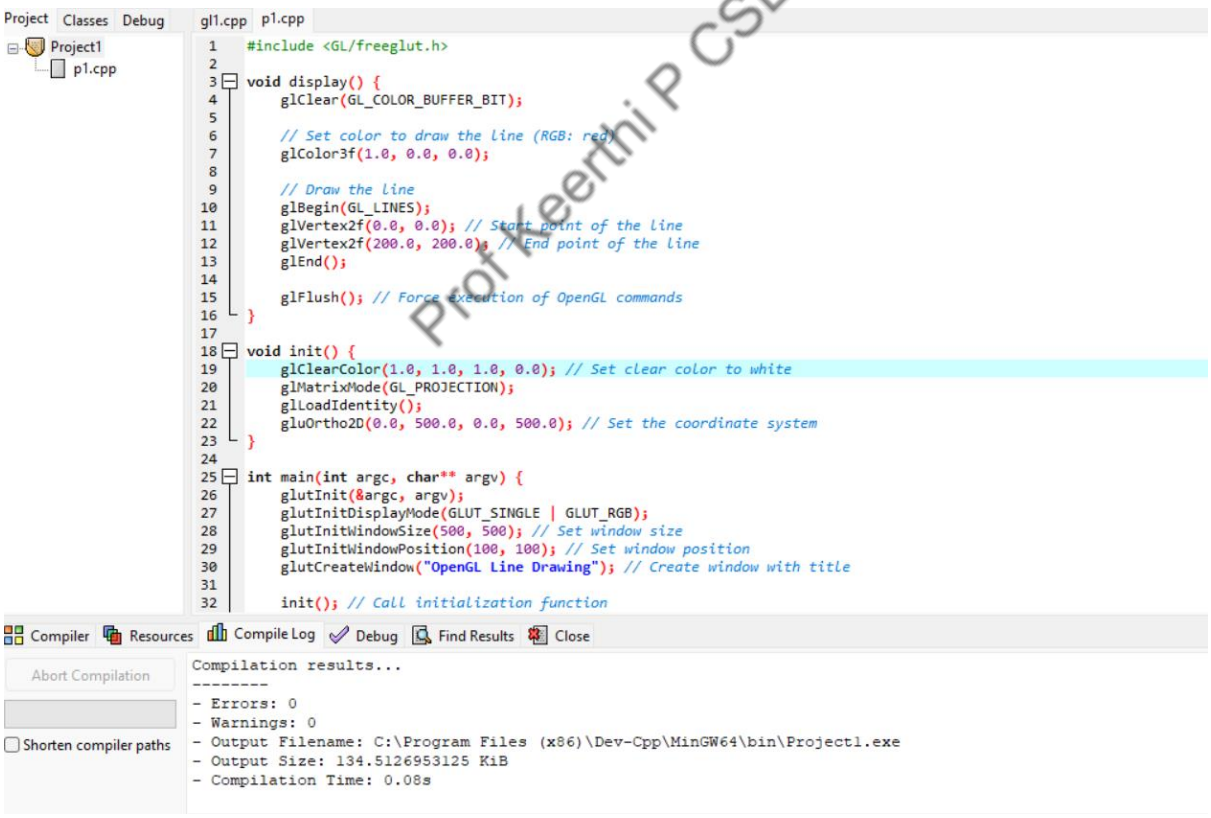
C:\Program Files (x86)\Dev-Cpp\MinGW64\bin

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

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    // Set color to draw the line (RGB: red)
    glColor3f(1.0, 0.0, 0.0);

    // Draw the line
    glBegin(GL_LINES);
    glVertex2f(100.0, 100.0); // Start point of the line
    glVertex2f(400.0, 400.0); // End point of the line
    glEnd();

    glFlush(); // Force execution of OpenGL commands
}

void init() {
    glClearColor(1.0, 1.0, 1.0, 0.0); // Set clear color to white
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 500.0, 0.0, 500.0); // Set the coordinate system
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(500, 500); // Set window size
    glutInitWindowPosition(100, 100); // Set window position
    glutCreateWindow("OpenGL Line Drawing"); // Create window with title

    init(); // Call initialization function

    glutDisplayFunc(display); // Register display callback function

    glutMainLoop(); // Enter the event-processing loop

    return 0;
}
```

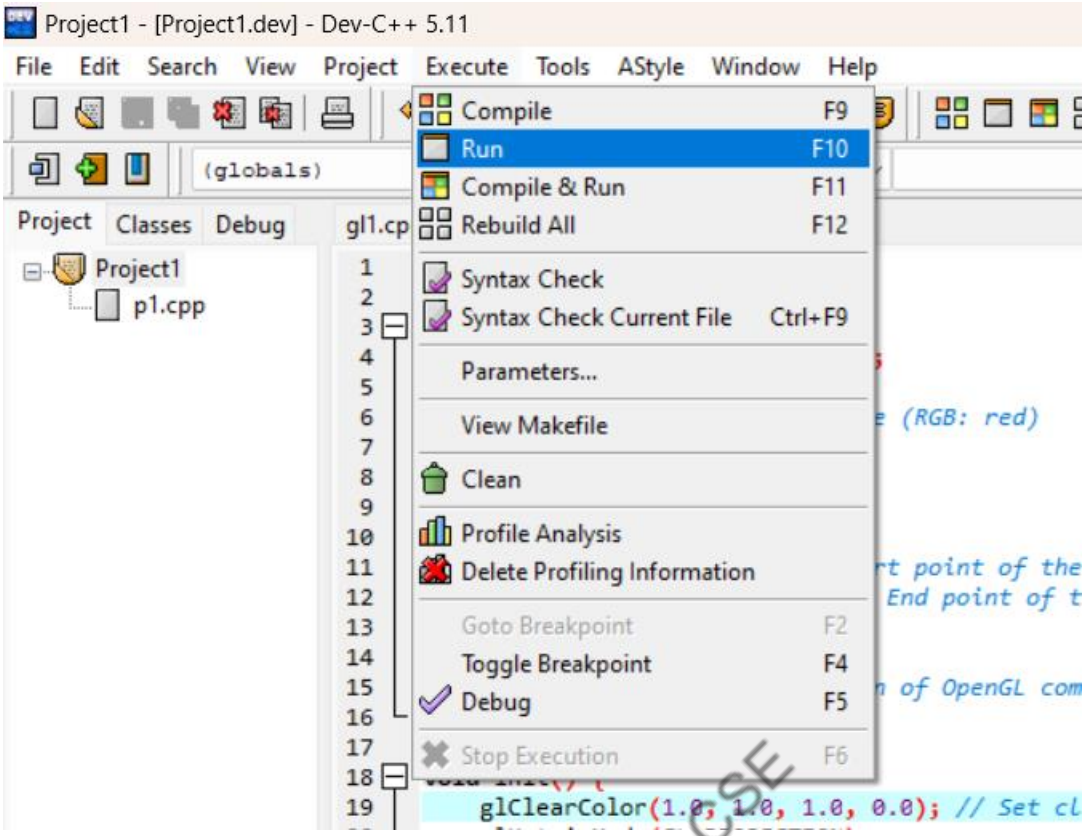

Compile



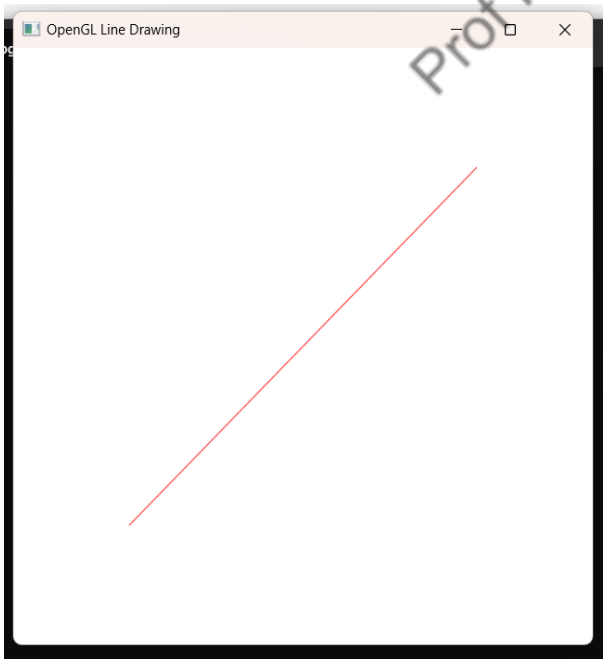
Program compiles successfully



Run the program



Output:



VI Semester

COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY			
Course Code	21CSL66	CIE Marks	50
Teaching Hours/Week (L:T:P: S)	0:0:2:0	SEE Marks	50
Total Hours of Pedagogy	24	Total Marks	100
Credits	1	Exam Hours	03

Course Objectives:

CLO 1: Demonstrate the use of Open GL.
CLO 2: Demonstrate the different geometric object drawing using openGL
CLO 3: Demonstration of 2D/3D transformation on simple objects.
CLO 4: Demonstration of lighting effects on the created objects.
CLO 5: Demonstration of Image processing operations on image/s.

Sl. No.	Practise Programs
	• Installation of OpenGL /OpenCV/ Python and required headers • Simple programs using OpenGL (Drawing simple geometric object like line, circle, rectangle, square) • Simple programs using OpenCV (operation on an image/s)

	PART A ***List of problems for which student should develop program and execute in the Laboratory using openGL/openCV/ Python***
1.	Develop a program to draw a line using Bresenham's line drawing technique
2.	Develop a program to demonstrate basic geometric operations on the 2D object
3.	Develop a program to demonstrate basic geometric operations on the 3D object
4.	Develop a program to demonstrate 2D transformation on basic objects
5.	Develop a program to demonstrate 3D transformation on 3D objects
6.	Develop a program to demonstrate Animation effects on simple objects.
7.	Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.
8.	Write a program to show rotation, scaling, and translation on an image.
9.	Read an image and extract and display low-level features such as edges, textures using filtering techniques.
10.	Write a program to blur and smoothing an image.
11.	Write a program to contour an image.
12.	Write a program to detect a face/s in an image.

PART-A

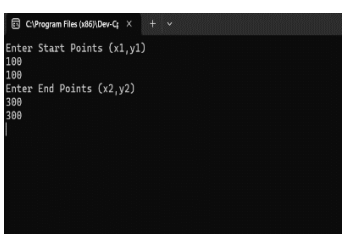
CG LAB PROGRAMS

1. Develop a program to draw a line using Bresenham's line drawing technique

```
#include <GL/freeglut.h>
#include<stdio.h>
int x1, y1, x2, y2;
void draw_pixel(int x, int y)
{
  glColor3f(1.0,0.0,0.0);
  glBegin(GL_POINTS);
  glVertex2i(x, y);
  glEnd();
}
void bresenhams_line_draw(int x1, int y1, int x2, int y2)
{
  float dx = x2 - x1;
  float dy = y2 - y1;
  float m = dy/dx;
  if(m < 1)
  {
    int decision_parameter = 2*dy - dx;
    int x = x1;              // initial x
    int y = y1;              // initial y
    if(dx < 0)              //  decide the first point and second point
     {
      x = x2;
      y = y2;
      x2 = x1;
    }
    draw_pixel(x, y);   // plot a point
    while(x < x2)         // from 1st point to 2nd point
    {
      if(decision_parameter >= 0)
      {
        x = x+1;
        y = y+1;
        decision_parameter=decision_parameter + 2*dy - 2*dx * (y+1 - y);
      }
      else
      {
        x = x+1;
        y = y;
        decision_parameter = decision_parameter + 2*dy - 2*dx * (y- y);
      }
      draw_pixel(x, y);
    }
  }

  else if(m > 1)
  {
    int decision_parameter = 2*dx - dy;
    int x = x1; // initial x
    int y = y1; // initial y
    if(dy < 0)
    {
      x = x2;
      y = y2;
      y2 = y1;
    }
    draw_pixel(x, y);
    while(y < y2)
    {
```

```
            if(decision_parameter >= 0)
            {
              x = x+1;
              y = y+1;
              decision_parameter=decision_parameter + 2*dx - 2*dy * (x+1 - x);
            }
            else
            {
              y = y+1;
              x = x;
              decision_parameter = decision_parameter + 2*dx - 2*dy * (x- x);
            }
            draw_pixel(x, y);
          }
        }
        else if (m == 1)
        {
          int x = x1;
          int y = y1;
          draw_pixel(x, y);
          while(x < x2)
          {
            x = x+1;
            y = y+1;
            draw_pixel(x, y);
          }
        }
}
void init()
{
  glClearColor(1,1,1,1);
  gluOrtho2D(0.0, 500.0, 0.0, 500.0);      // left ->0, right ->500, bottom ->0, top ->500
}
void display()
{
  glClear(GL_COLOR_BUFFER_BIT);
  bresenhams_line_draw(x1, y1, x2, y2);
  glFlush();
}
int main(int argc, char **argv)
{
  printf( "Enter Start Points (x1,y1)\n");
  scanf("%d %d", &x1, &y1);                    // 1st point from user
  printf( "Enter End Points (x2,y2)\n");
  scanf("%d %d", &x2, &y2);                    // 2nd point from user
  glutInit(&argc, argv);                       // initialize graphics system
  glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);  // single buffered mode with RGB colour variants
  glutInitWindowSize(500, 500);                // 500 by 500 window size
  glutInitWindowPosition(220, 200);            // where do you wanna see your window
  glutCreateWindow("Bresenham's Line Drawing - FVBIE");  // the title of your window
  init();                                      // initialize the canvas
  glutDisplayFunc(display);                    // call display function
  glutMainLoop();                              // run forever
}
```

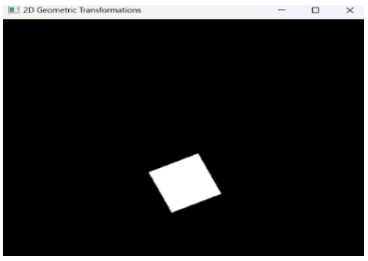

OUTPUT:

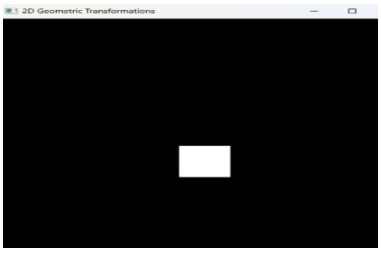


2. Develop a program to demonstrate basic geometric operations on the 2D object

```
#include <GL/glut.h>
// Initial position and size
float tx = 0.0f, ty = 0.0f; // Translation parameters
float angle = 0.0f;        // Rotation angle
float sx = 1.0f, sy = 1.0f; // Scaling factors
void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   // Apply transformations
   glPushMatrix();
   glTranslatef(tx, ty, 0.0f);
   glRotatef(angle, 0.0f, 0.0f, 1.0f);
   glScalef(sx, sy, 1.0f);
   // Draw a rectangle
   glBegin(GL_POLYGON);
   glVertex2f(-0.5f, -0.5f);
   glVertex2f(0.5f, -0.5f);
   glVertex2f(0.5f, 0.5f);
   glVertex2f(-0.5f, 0.5f);
   glEnd();
   glPopMatrix();
   glFlush();
}
void keyboard(unsigned char key, int x, int y) {
   switch (key) {
      case 'w': ty += 0.1f; break; // Translate up
      case 's': ty -= 0.1f; break; // Translate down
      case 'a': tx -= 0.1f; break; // Translate left
      case 'd': tx += 0.1f; break; // Translate right
      case 'q': angle += 5.0f; break; // Rotate counterclockwise
      case 'e': angle -= 5.0f; break; // Rotate clockwise
      case 'z': sx += 0.1f; sy += 0.1f; break; // Scale up
      case 'x': sx -= 0.1f; sy -= 0.1f; break; // Scale down
      case 27: exit(0); // Exit on 'ESC'
   }
   glutPostRedisplay();
}
void init() {
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set clear color to black
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-2.0, 2.0, -2.0, 2.0); // Set orthographic projection
}
int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("2D Geometric Transformations");
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```


Output:

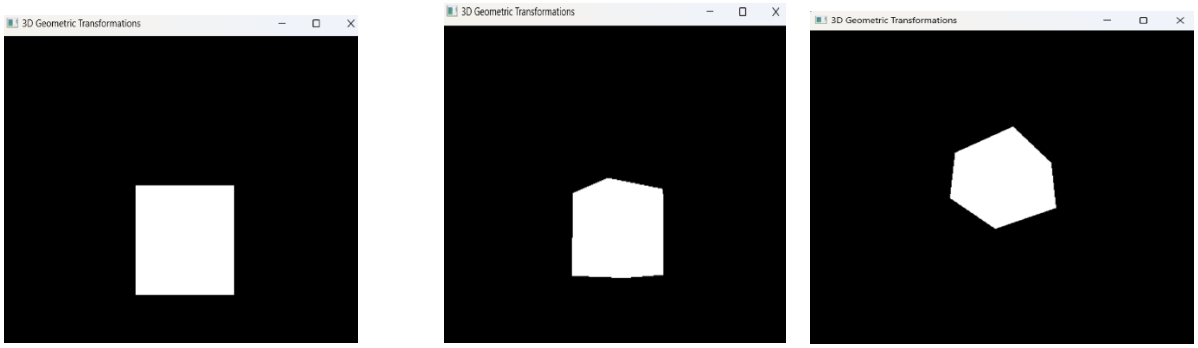




3. Develop a program to demonstrate basic geometric operations on the 3D object

```
#include <GL/glut.h>
// Initial position and size
float tx = 0.0f, ty = 0.0f, tz = 0.0f; // Translation parameters
float angleX = 0.0f, angleY = 0.0f, angleZ = 0.0f; // Rotation angles
float sx = 1.0f, sy = 1.0f, sz = 1.0f; // Scaling factors
void display() {
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   // Apply transformations
   glPushMatrix();
   glTranslatef(tx, ty, tz);
   glRotatef(angleX, 1.0f, 0.0f, 0.0f);
   glRotatef(angleY, 0.0f, 1.0f, 0.0f);
   glRotatef(angleZ, 0.0f, 0.0f, 1.0f);
   glScalef(sx, sy, sz);
   // Draw a cube
   glutSolidCube(1.0);
   glPopMatrix();
   glutSwapBuffers();
}
void keyboard(unsigned char key, int x, int y) {
   switch (key) {
      case 'w': ty += 0.1f; break; // Translate up
      case 's': ty -= 0.1f; break; // Translate down
      case 'a': tx -= 0.1f; break; // Translate left
      case 'd': tx += 0.1f; break; // Translate right
      case 'q': tz -= 0.1f; break; // Translate forward
      case 'e': tz += 0.1f; break; // Translate backward
      case 'i': angleX += 5.0f; break; // Rotate around X axis
      case 'k': angleX -= 5.0f; break; // Rotate around X axis
      case 'j': angleY += 5.0f; break; // Rotate around Y axis
      case 'l': angleY -= 5.0f; break; // Rotate around Y axis
      case 'u': angleZ += 5.0f; break; // Rotate around Z axis
      case 'o': angleZ -= 5.0f; break; // Rotate around Z axis
      case 'z': sx += 0.1f; sy += 0.1f; sz += 0.1f; break; // Scale up
      case 'x': sx -= 0.1f; sy -= 0.1f; sz -= 0.1f; break; // Scale down
      case 27: exit(0); // Exit on 'ESC'
   }
   glutPostRedisplay();
}
void init() {
   glClearColor(0.0f, 0.0f, 0.0f, 1.0f); // Set clear color to black
   glEnable(GL_DEPTH_TEST); // Enable depth testing
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluPerspective(45.0, 1.0, 1.0, 100.0);
   glMatrixMode(GL_MODELVIEW);
   glLoadIdentity();
   gluLookAt(0.0, 0.0, 5.0,  // Eye position
         0.0, 0.0, 0.0,  // Look-at position
         0.0, 1.0, 0.0); // Up direction
}
int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(500, 500);
   glutInitWindowPosition(100, 100);
   glutCreateWindow("3D Geometric Transformations");
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc(keyboard);
   glutMainLoop();
   return 0;
}
```

Output:



4. Develop a program to demonstrate 2D transformation on basic objects

```
#include <GL/freeglut.h>
// Variables for geometric transformations
float angle = 0.0f;
float scaleX = 1.0f, scaleY = 1.0f;
float translateX = 0.0f, translateY = 0.0f;
void init() {
  glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white
  glColor3f(0.0, 0.0, 0.0);       // Set drawing color to black
  glMatrixMode(GL_PROJECTION);    // Set the projection matrix
  gluOrtho2D(-500, 500, -500, 500); // Define the 2D orthographic projection
}
void drawRectangle() {
  glBegin(GL_QUADS);
  glVertex2f(-100.0f, -50.0f);
  glVertex2f(100.0f, -50.0f);
  glVertex2f(100.0f, 50.0f);
  glVertex2f(-100.0f, 50.0f);
  glEnd();
}
void drawTriangle() {
  glBegin(GL_TRIANGLES);
  glVertex2f(-50.0f, -50.0f);
  glVertex2f(50.0f, -50.0f);
  glVertex2f(0.0f, 50.0f);
  glEnd();
}
void display() {
  glClear(GL_COLOR_BUFFER_BIT);

  // Draw and transform rectangle
  glPushMatrix();
  glTranslatef(translateX, translateY, 0.0f);
  glRotatef(angle, 0.0f, 0.0f, 1.0f);
  glScalef(scaleX, scaleY, 1.0f);
  glColor3f(0.0, 0.0, 1.0); // Blue color
  drawRectangle();
  glPopMatrix();
  // Draw and transform triangle
  glPushMatrix();
  glTranslatef(-translateX, -translateY, 0.0f);
  glRotatef(-angle, 0.0f, 0.0f, 1.0f);
  glScalef(scaleX, scaleY, 1.0f);
  glColor3f(1.0, 0.0, 0.0); // Red color
  drawTriangle();
  glPopMatrix();
  glutSwapBuffers();
}
void timer(int value) {
  angle += 1.0f;
  if (angle > 360.0f) angle -= 360.0f;
```

```
    translateX += 1.0f;
    if (translateX > 500.0f) translateX = -500.0f;
    glutPostRedisplay(); // Redraw the scene
    glutTimerFunc(16, timer, 0); // 16 ms for ~60 FPS
}
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 27: // ESC key
            exit(0);
        case 'w':
            translateY += 10.0f;
            break;
        case 's':
            translateY -= 10.0f;
            break;
        case 'a':
            translateX -= 10.0f;
            break;
        case 'd':
            translateX += 10.0f;
            break;
        case '+':
            scaleX += 0.1f;
            scaleY += 0.1f;
            break;
        case '-':
            scaleX -= 0.1f;
            scaleY -= 0.1f;
            break;
    }
    glutPostRedisplay();
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowSize(800, 600);
    glutCreateWindow("2D Geometric Operations using OpenGL and GLUT");
    init();
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
    glutTimerFunc(0, timer, 0);
    glutMainLoop();
    return 0;
}
```

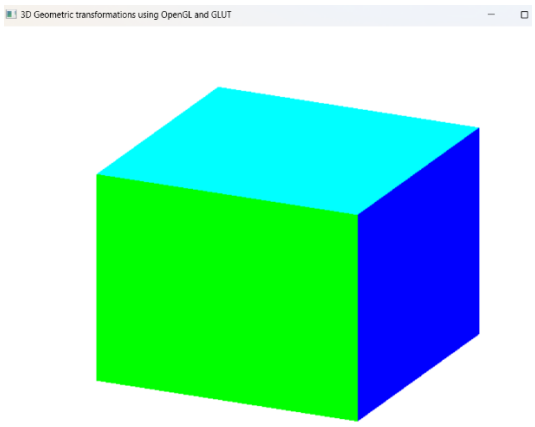

Output:

5. Develop a program to demonstrate 3D transformation on 3D objects

```
#include <GL/glut.h>
// Variables for geometric transformations
float angleX = 0.0f;
float angleY = 0.0f;
float scaleX = 1.0f, scaleY = 1.0f, scaleZ = 1.0f;
float translateX = 0.0f, translateY = 0.0f, translateZ = 0.0f;
void init() {
  glClearColor(1.0, 1.0, 1.0, 1.0); // Set background color to white
  glEnable(GL_DEPTH_TEST);        // Enable depth testing
}
void drawCube() {
  glBegin(GL_QUADS);
  // Front face
  glColor3f(1.0, 0.0, 0.0); // Red
  glVertex3f(-0.5, -0.5,  0.5);
  glVertex3f( 0.5, -0.5,  0.5);
  glVertex3f( 0.5,  0.5,  0.5);
  glVertex3f(-0.5,  0.5,  0.5);
  // Back face
  glColor3f(0.0, 1.0, 0.0); // Green
  glVertex3f(-0.5, -0.5, -0.5);
  glVertex3f(-0.5,  0.5, -0.5);
  glVertex3f( 0.5,  0.5, -0.5);
  glVertex3f( 0.5, -0.5, -0.5);
  // Left face
  glColor3f(0.0, 0.0, 1.0); // Blue
  glVertex3f(-0.5, -0.5,  0.5);
  glVertex3f(-0.5,  0.5,  0.5);
  glVertex3f(-0.5,  0.5, -0.5);
  glVertex3f(-0.5, -0.5, -0.5);
  // Right face
  glColor3f(1.0, 1.0, 0.0); // Yellow
  glVertex3f(0.5, -0.5,  0.5);
  glVertex3f(0.5, -0.5, -0.5);
  glVertex3f(0.5,  0.5, -0.5);
  glVertex3f(0.5,  0.5,  0.5);
  // Top face
  glColor3f(1.0, 0.0, 1.0); // Magenta
  glVertex3f(-0.5,  0.5,  0.5);
  glVertex3f( 0.5,  0.5,  0.5);
  glVertex3f( 0.5,  0.5, -0.5);
  glVertex3f(-0.5,  0.5, -0.5);
  // Bottom face
  glColor3f(0.0, 1.0, 1.0); // Cyan
  glVertex3f(-0.5, -0.5,  0.5);
  glVertex3f(-0.5, -0.5, -0.5);
  glVertex3f( 0.5, -0.5, -0.5);
  glVertex3f( 0.5, -0.5,  0.5);
  glEnd();
}
void display() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glPushMatrix();
  // Apply transformations
  glTranslatef(translateX, translateY, translateZ);
  glRotatef(angleX, 1.0, 0.0, 0.0);
  glRotatef(angleY, 0.0, 1.0, 0.0);
  glScalef(scaleX, scaleY, scaleZ);
  // Draw cube
  drawCube();
  glPopMatrix();
  glutSwapBuffers();
```

```
}

void timer(int value) {
   angleX += 1.0f;
   angleY += 1.0f;
   if (angleX > 360.0f) angleX -= 360.0f;
   if (angleY > 360.0f) angleY -= 360.0f;
   glutPostRedisplay(); // Redraw the scene
   glutTimerFunc(16, timer, 0); // 16 ms for ~60 FPS
}
void keyboard(unsigned char key, int x, int y) {
   switch (key) {
      case 27: // ESC key
         exit(0);
      case 'w':
         translateY += 0.1f;
         break;
      case 's':
         translateY -= 0.1f;
         break;
      case 'a':
         translateX -= 0.1f;
         break;
      case 'd':
         translateX += 0.1f;
         break;
      case 'q':
         translateZ -= 0.1f;
         break;
      case 'e':
         translateZ += 0.1f;
         break;
      case 'i':
         scaleY += 0.1f;
         break;
      case 'k':
         scaleY -= 0.1f;
         break;
      case 'j':
         scaleX -= 0.1f;
         break;
      case 'l':
         scaleX += 0.1f;
         break;
      case 'u':
         scaleZ -= 0.1f;
         break;
      case 'o':
         scaleZ += 0.1f;
         break;
   }
}
int main(int argc, char** argv) {
   glutInit(&argc, argv);
   glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
   glutInitWindowSize(800, 600);
   glutCreateWindow("3D Geometric transformations using OpenGL and GLUT");
   init();
   glutDisplayFunc(display);
   glutKeyboardFunc(keyboard);
   glutTimerFunc(0, timer, 0);
   glutMainLoop();
   return 0;
}
```

Output:

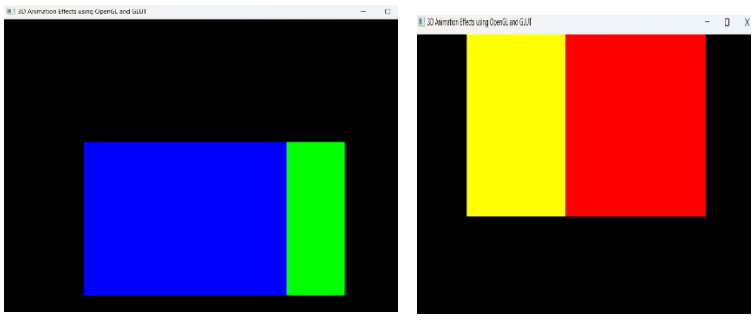


6. Develop a program to demonstrate Animation effects on simple objects.

```
#include <GL/glut.h>
// Variables for animation
float angle = 0.0f;
float bounce = 0.0f;
float bounceSpeed = 0.05f;
bool goingUp = true;
void init() {
   glClearColor(0.0, 0.0, 0.0, 1.0); // Set background color to black
   glEnable(GL_DEPTH_TEST);         // Enable depth testing
}
void drawCube() {
   glBegin(GL_QUADS);
   // Front face
   glColor3f(1.0, 0.0, 0.0); // Red
   glVertex3f(-0.5, -0.5,  0.5);
   glVertex3f( 0.5, -0.5,  0.5);
   glVertex3f( 0.5,  0.5,  0.5);
   glVertex3f(-0.5,  0.5,  0.5);
   // Back face
   glColor3f(0.0, 1.0, 0.0); // Green
   glVertex3f(-0.5, -0.5, -0.5);
   glVertex3f(-0.5,  0.5, -0.5);
   glVertex3f( 0.5,  0.5, -0.5);
   glVertex3f( 0.5, -0.5, -0.5);
   // Left face
   glColor3f(0.0, 0.0, 1.0); // Blue
   glVertex3f(-0.5, -0.5,  0.5);
   glVertex3f(-0.5,  0.5,  0.5);
   glVertex3f(-0.5,  0.5, -0.5);
   glVertex3f(-0.5, -0.5, -0.5);
   // Right face
   glColor3f(1.0, 1.0, 0.0); // Yellow
   glVertex3f(0.5, -0.5,  0.5);
   glVertex3f(0.5, -0.5, -0.5);
   glVertex3f(0.5,  0.5, -0.5);
   glVertex3f(0.5,  0.5,  0.5);
   // Top face
   glColor3f(1.0, 0.0, 1.0); // Magenta
   glVertex3f(-0.5,  0.5,  0.5);
   glVertex3f( 0.5,  0.5,  0.5);
   glVertex3f( 0.5,  0.5, -0.5);
   glVertex3f(-0.5,  0.5, -0.5);
```

```
    // Bottom face
    glColor3f(0.0, 1.0, 1.0); // Cyan
    glVertex3f(-0.5, -0.5,  0.5);
    glVertex3f(-0.5, -0.5, -0.5);
    glVertex3f( 0.5, -0.5, -0.5);
    glVertex3f( 0.5, -0.5,  0.5);
    glEnd();
}
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glPushMatrix();
    // Apply rotation and translation for bounce
    glTranslatef(0.0f, bounce, 0.0f);
    glRotatef(angle, 0.0, 1.0, 0.0);
    // Draw cube
    drawCube();
    glPopMatrix();
    glutSwapBuffers();
}

void timer(int value) {
    // Update rotation angle
    angle += 1.0f;
    if (angle > 360) angle -= 360;
    // Update bounce position
    if (goingUp) {
        bounce += bounceSpeed;
        if (bounce > 1.0f) goingUp = false;
    } else {
        bounce -= bounceSpeed;
        if (bounce < -1.0f) goingUp = true;
    }
    glutPostRedisplay(); // Redraw the scene
    glutTimerFunc(16, timer, 0); // 16 ms for ~60 FPS
}
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(800, 600);
    glutCreateWindow("3D Animation Effects using OpenGL and GLUT");
    init();
    glutDisplayFunc(display);
    glutTimerFunc(0, timer, 0);
    glutMainLoop();
    return 0;
}
```


Output:



INSTALLATION OF PYTHON AND OPENCV

1.Install vscode/pycharm/spyder IDE to work with python programs

2.Once installed, install opencv with the below command in python console

Install OpenCV

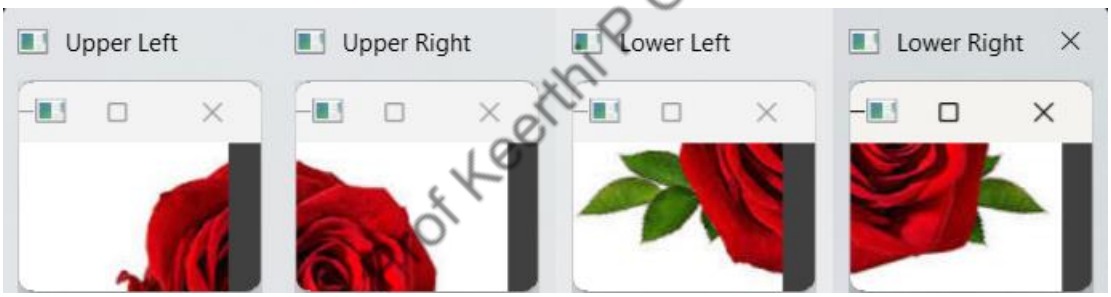
Make sure you have OpenCV installed. You can install it using pip:

```
pip install opencv-python
```

7. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.

```
import cv2
import numpy as np
# Read the image
image = cv2.imread('rose.jpg')
# Get the dimensions of the image
height, width = image.shape[:2]
# Calculate the center point
center_y, center_x = height // 2, width // 2
# Split the image into four quadrants
upper_left = image[:center_y, :center_x]
upper_right = image[:center_y, center_x:]
lower_left = image[center_y:, :center_x]
lower_right = image[center_y:, center_x:]
#Display the quadrants
cv2.imshow('Upper Left', upper_left)
cv2.imshow('Upper Right', upper_right)
cv2.imshow('Lower Left', lower_left)
cv2.imshow('Lower Right', lower_right)
# Wait until a key is pressed and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

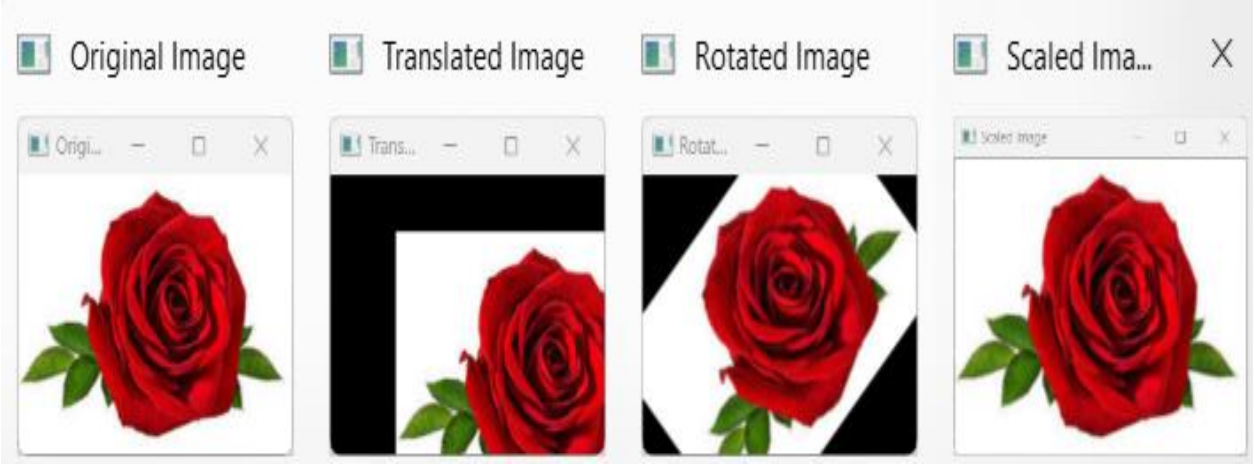
output:



8. Write a program to show rotation, scaling, and translation on an image.

```
import cv2
import numpy as np
def translate_image(image, tx, ty):
    # Define the translation matrix
    M = np.float32([[1, 0, tx], [0, 1, ty]])
    # Perform the translation
    translated = cv2.warpAffine(image, M, (image.shape[1], image.shape[0]))
    return translated
def rotate_image(image, angle, scale=1.0):
    # Get the image dimensions
    (h, w) = image.shape[:2]
    # Calculate the center of the image
    center = (w // 2, h // 2)
    # Define the rotation matrix
    M = cv2.getRotationMatrix2D(center, angle, scale)
    # Perform the rotation
    rotated = cv2.warpAffine(image, M, (w, h))
    return rotated
def scale_image(image, scale_x, scale_y):
    # Perform the scaling
    scaled = cv2.resize(image, None, fx=scale_x, fy=scale_y, interpolation=cv2.INTER_LINEAR)
    return scaled
# Read the image
image = cv2.imread('rose.jpg')
# Translation parameters
tx, ty = 50, 30
# Rotation parameters
angle = 45  # degrees
# Scaling parameters
scale_x, scale_y = 1.5, 1.5
# Apply transformations
translated_image = translate_image(image, tx, ty)
rotated_image = rotate_image(image, angle)
scaled_image = scale_image(image, scale_x, scale_y)
# Display the original and transformed images
cv2.imshow('Original Image', image)
cv2.imshow('Translated Image', translated_image)
cv2.imshow('Rotated Image', rotated_image)
cv2.imshow('Scaled Image', scaled_image)
# Wait until a key is pressed and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

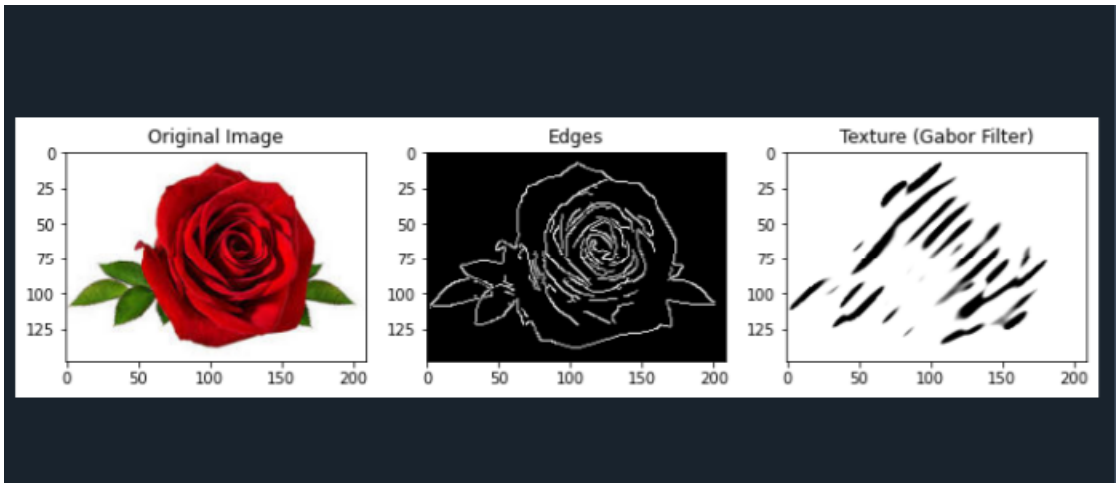

output:



9.Read an image and extract and display low-level features such as edges, textures using filtering techniques.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
def apply_gabor_filters(image):
    # Define Gabor filter parameters
    ksize = 31  # Size of the filter
    sigma = 4.0  # Standard deviation of the gaussian envelope
    theta = np.pi / 4  # Orientation of the normal to the parallel stripes
    lambd = 10.0  # Wavelength of the sinusoidal factor
    gamma = 0.5  # Spatial aspect ratio
    phi = 0  # Phase offset
    # Create Gabor kernel
    gabor_kernel = cv2.getGaborKernel((ksize, ksize), sigma, theta, lambd, gamma, phi, ktype=cv2.CV_32F)
        # Apply the Gabor filter
    filtered_img = cv2.filter2D(image, cv2.CV_8UC3, gabor_kernel)
        return filtered_img
# Read the image
image = cv2.imread('rose.jpg')
gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Edge detection using Canny
edges = cv2.Canny(gray_image, 100, 200)
# Texture extraction using Gabor filters
texture = apply_gabor_filters(gray_image)
# Display the original image, edges, and textures
plt.figure(figsize=(10, 7))
plt.subplot(1, 3, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(1, 3, 2)
plt.title("Edges")
plt.imshow(edges, cmap='gray')
plt.subplot(1, 3, 3)
plt.title("Texture (Gabor Filter)")
plt.imshow(texture, cmap='gray')
plt.tight_layout()
plt.show()
```

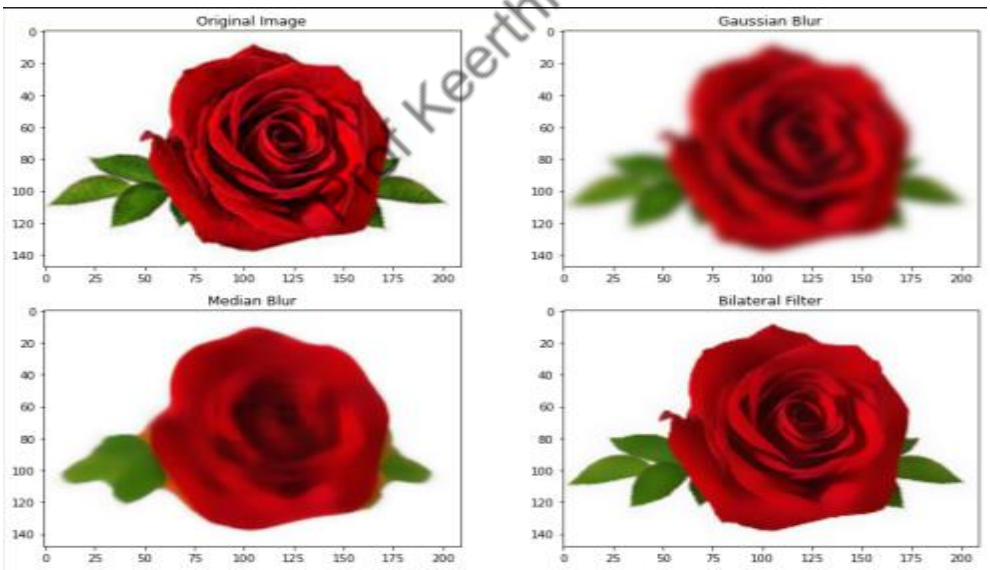

output:



10.Write a program to blur and smoothing an image.

```
import cv2
import numpy as np
import matplotlib.pyplot as plt
# Read the image
image = cv2.imread('rose.jpg')
# Apply Gaussian Blur
gaussian_blur = cv2.GaussianBlur(image, (15, 15), 0)
# Apply Median Blur
median_blur = cv2.medianBlur(image, 15)
# Apply Bilateral Filter
bilateral_blur = cv2.bilateralFilter(image, 15, 75, 75)
# Display the original and blurred images
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.title("Original Image")
plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
plt.subplot(2, 2, 2)
plt.title("Gaussian Blur")
plt.imshow(cv2.cvtColor(gaussian_blur, cv2.COLOR_BGR2RGB))
plt.subplot(2, 2, 3)
plt.title("Median Blur")
plt.imshow(cv2.cvtColor(median_blur, cv2.COLOR_BGR2RGB))
plt.subplot(2, 2, 4)
plt.title("Bilateral Filter")
plt.imshow(cv2.cvtColor(bilateral_blur, cv2.COLOR_BGR2RGB))
plt.tight_layout()
plt.show()
```

output:



11.Write a program to contour an image.

```
import cv2
import numpy as np
# Read the image
image = cv2.imread('rose.jpg')
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Apply a binary threshold to the image
_, binary = cv2.threshold(gray, 150, 255, cv2.THRESH_BINARY_INV)
# Find contours in the binary image
contours, _ = cv2.findContours(binary, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
# Draw the contours on the original image
contoured_image = image.copy()
cv2.drawContours(contoured_image, contours, -1, (0, 255, 0), 2)
# Display the images
cv2.imshow('Original Image', image)
cv2.imshow('Binary Image', binary)
cv2.imshow('Contoured Image', contoured_image)
# Wait until a key is pressed and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

output:

12.Write a program to detect a face/s in an image.

```
import cv2
# Load the pre-trained Haar Cascade classifier for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')
# Read the image
image = cv2.imread('einstein.jpg')
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))
# Draw rectangles around the detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)
# Display the output
cv2.imshow('Detected Faces', image)
# Wait until a key is pressed and close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

output: