DAYANANDA SAGAR ACADEMY OF TECHNOLOGY & MANAGEMENT
Opp. Art of Living, Udayapura, Kanakapura Road, Bangalore- 560082
(Affiliated to Visvesvaraya Technological University, Belagavi and Approved by AICTE, New Delhi)
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
(CE, CSE, ECE, EEE, ISE, ME Courses Accredited by NBA, New Delhi Accredited by NAAC, A+)

COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY

VI Semester
Course Code: 21CSL66

Editorial Committee
Computer Graphics Lab Faculty, Dept. of CSE

Approved by
H.O.D, Department of CSE

Document Log

Name of the document	Computer Graphics and Image Processing Laboratory
Scheme	2021
Subject code	21CSL66
Editorial Committee	Dr. Shalini S Dr. Nagaraj M Lutimath Prof. Manasa Sandeep
Computer Programmer	Mr. Vinay Kumar
Approved by	HOD, Dept. of CSE

Table of Contents

Vision of the Department

Epitomize CSE graduate to carve a niche globally in the field of computer science to excel in the world of information technology and automation by imparting knowledge to sustain skills for the changing trends in the society and industry.

Mission of the Department

M1: To educate students to become excellent engineers in a confident and creative environment through world-class pedagogy.

M2: Enhancing the knowledge in the changing technology trends by giving hands-on experience through continuous education and by making them to organize & participate in various events.

M3: Impart skills in the field of IT and its related areas with a focus on developing the required competencies and virtues to meet the industry expectations.

M4: Ensure quality research and innovations to fulfill industry, government & social needs.

M5: Impart entrepreneurship and consultancy skills to students to develop self-sustaining life skills in multi-disciplinary areas.

Program Educational Objectives (PEOs)

After the course completion, CSE graduates will be able to:

PEO1: Engage in professional practice to promote the development of innovative systems and optimized solutions for Computer Science and Engineering.

PEO2: Adapt to different roles and responsibilities in multidisciplinary working environment by respecting professionalism and ethical practices within organization and society at national and international level.

PEO3: Graduates will engage in life-long learning and professional development to acclimate the rapidly changing work environment and develop entrepreneurship skills.

Program Specific Outcomes (PSO)

PSO1: Foundation of Mathematical Concepts: Ability to use mathematical methodologies to solve the problem using suitable mathematical analysis, data structure and suitable algorithm.

PSO2: Foundation of Computer System: Ability to interpret the fundamental concepts and methodology of computer systems. Students can understand the functionality of hardware and software aspects of computer systems.

PSO3: Foundations of Software Development: Ability to grasp the software development lifecycle and methodologies of software systems. Possess competent skills and knowledge of software design process. Familiarity and practical proficiency with a broad area of programming concepts and provide new ideas and innovations towards research.

PSO4: Foundations of Multi-Disciplinary Work: Ability to acquire leadership skills to perform professional activities with social responsibilities, through excellent flexibility to function in multi-disciplinary work environment with self-learning skills.

Program Outcomes (POs)

Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. **Problem analysis**: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. **Design/development of solutions**: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. **Modern tool usage**: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. **The engineer and society**: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. **Environment and sustainability**: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics**: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. **Individual and team work**: Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.

10. **Communication**: Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.

11. **Project management and finance**: Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.

12. **Life-long learning**: Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

Course Details

Syllabus

Subject Code	**: 21CSL66**	**IA Marks**	**: 50**
No. of Practical Hrs/ Week	**: 0:0:2:0**	**Exam Hours**	**: 03**
Total No. of Pedagogy	**: 24**	**Exam Marks**	**: 50**

Laboratory Experiments:

Practice 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 – PROGRAMS

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 –B
Practical Based Learning

Student should develop a mini project and it should be demonstrate in the laboratory examination, Some of the projects are listed and it is not limited to:

¬ Recognition of License Plate through Image Processing

¬ Recognition of Face Emotion in Real-Time

¬ Detection of Drowsy Driver in Real-Time

¬ Recognition of Handwriting by Image Processing

¬ Detection of Kidney Stone

¬ Verification of Signature

¬ Compression of Color Image

¬ Classification of Image Category

¬ Detection of Skin Cancer

¬ Marking System of Attendance using Image Processing

¬ Detection of Liver Tumor

¬ IRIS Segmentation

¬ Detection of Skin Disease and / or Plant Disease

¬ Biometric Sensing System

¬ Projects which helps to farmers to understand the present developments in agriculture.

¬ Projects which helps high school/college students to understand the scientific problems

¬ Simulation projects which helps to understand innovations in science and technology

Course Outcomes

Upon successful completion of this course, students are able to:

COs	**COURSE OUTCOMES**
21CSL66.1	Use openGL /OpenCV for the development of mini Projects.
21CSL66.2	Analyze the necessity mathematics and design required to demonstrate basic geometric transformation techniques
21CSL66.3	Demonstrate the ability to design and develop input interactive techniques.
21CSL66.4	Apply the concepts to Develop user friendly applications using Graphics and IP concepts

CO-PO-PSO MAPPING

Course	**PO1**	**PO2**	**PO3**	**PO4**	**PO5**	**PO6**	**PO7**	**PO8**	**PO9**	**PO10**	**PO11**	**PO12**	**PSO1**	**PSO2**
21CSL66.1	**3**	**3**	**3**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**3**	**-**
21CSL66.2	**3**	**3**	**3**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**3**	**-**
21CSL66.3	**3**	**3**	**3**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**3**	**-**
21CSL66.4	**3**	**3**	**3**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**3**	**-**
Avg	**3**	**3**	**3**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**-**	**3**	**-**

Assessment Details (both CIE and SEE)

The weightage of Continuous Internal Evaluation (CIE) is 50% and for Semester End Exam (SEE) is 50%. The minimum passing mark for the CIE is 40% of the maximum marks (20 marks). A student shall be deemed to have satisfied the academic requirements and earned the credits allotted to each course. The student has to secure not less than 35% (18 Marks out of 50) in the semester-end examination (SEE).

Continuous Internal Evaluation (CIE):

CIE marks for the practical course is 50 Marks.

The split-up of CIE marks for record/ journal and test are in the ratio 60:40.

- Each experiment to be evaluated for conduction with observation sheet and record write-up. Rubrics for the evaluation of the journal/write-up for hardware/software experiments designed by the faculty who is handling the laboratory session and is made known to students at the beginning of the practical session.
- Record should contain all the specified experiments in the syllabus and each experiment writeup will be evaluated for 10 marks.
- Total marks scored by the students are scaled downed to 30 marks (60% of maximum marks).
- Weightage to be given for neatness and submission of record/write-up on time.
- Department shall conduct 02 tests for 100 marks, the first test shall be conducted after the 8th week of the semester and the second test shall be conducted after the 14th week of the semester.
- In each test, test write-up, conduction of experiment, acceptable result, and procedural knowledge will carry a weightage of 60% and the rest 40% for viva-voce.
- The suitable rubrics can be designed to evaluate each student's performance and learning ability. Rubrics suggested in Annexure-II of Regulation book
- The average of 02 tests is scaled down to 20 marks (40% of the maximum marks). The Sum of scaled-down marks scored in the report write-up/journal and average marks of two tests is the total CIE marks scored by the student.

Semester End Evaluation (SEE):

SEE marks for the practical course is 50 Marks.

- SEE shall be conducted jointly by the two examiners of the same institute, examiners are appointed by the University

- All laboratory experiments are to be included for practical examination.
- (Rubrics) Breakup of marks and the instructions printed on the cover page of the answer script to be strictly adhered to by the examiners. OR based on the course requirement evaluation rubrics shall be decided jointly by examiners.
- Students can pick one question (experiment) from the questions lot prepared by the internal /external examiners jointly.
- Evaluation of test write-up/ conduction procedure and result/viva will be conducted jointly by examiners.

- General rubrics suggested for SEE are mentioned here, writeup-20%, Conduction procedure and result in -60%, Viva-voce 20% of maximum marks. SEE for practical shall be evaluated for 100 marks and scored marks shall be scaled down to 50 marks (however, based on course type, rubrics shall be decided by the examiners)

- Students can pick one experiment from the questions lot of PART A with equal choice to all the students in a batch.
- PART B : Student should develop a mini project and it should be demonstrated in the laboratory examination (with report and presentation).
- Weightage of marks for PART A is 60% and for PART B is 40%. General rubrics suggested to be followed for part A and part B.
- Change of experiment is allowed only once (in part A) and marks allotted to the procedure part to be made zero.
- The duration of SEE is 03 hours.

INTRODUCTION

What Is OpenGL?

OpenGL is a software interface to graphics hardware. This interface consists of about 150 distinct commands that you use to specify the objects and operations needed to produce interactive three-dimensional applications.

OpenGL is designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. With OpenGL, you must build up your desired model from a small set of *geometric primitives* - points, lines, and polygons. The OpenGL Utility Library (GLU) provides many of the modeling features, such as quadric surfaces and NURBS curves and surfaces. GLU is a standard part of every OpenGL implementation. Also, there is a higher-level, object-oriented toolkit,

Construct shapes from geometric primitives, thereby creating mathematical descriptions of objects. (OpenGL considers points, lines, polygons, images, and bitmaps to be primitives.)

OpenGL programs can work across a network even if the client and server are different kinds of computers. If an OpenGL program isn't running across a network, then there's only one computer, and it is both the client and the server.

Advantages of OpenGL

1. Uniform approach to writing graphics applications.
2. Some code can be compiled and run on a variety of graphics environments.
3. OpenGL is portable
4. Cross platform (Windows, Linux, Mac, even some hand held devices).

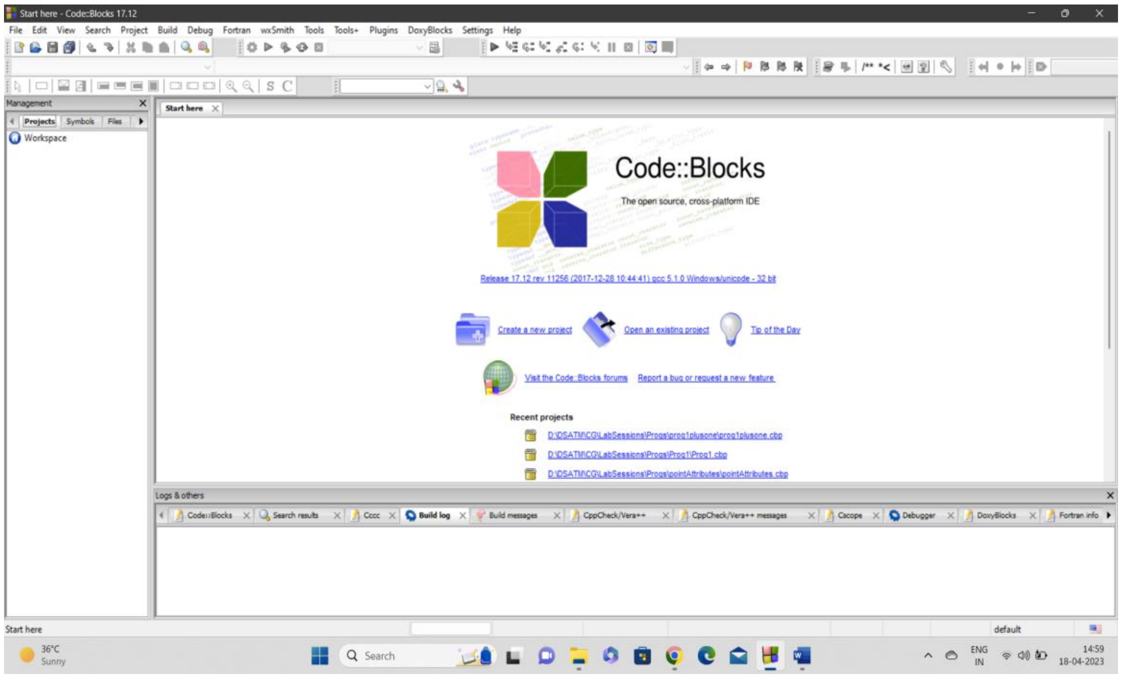
Software's required:

1. Codeblocks
2. The important files and libraries for running the OpenGL application are:
    - gl.h glu.h opengl32.libglu32.lib

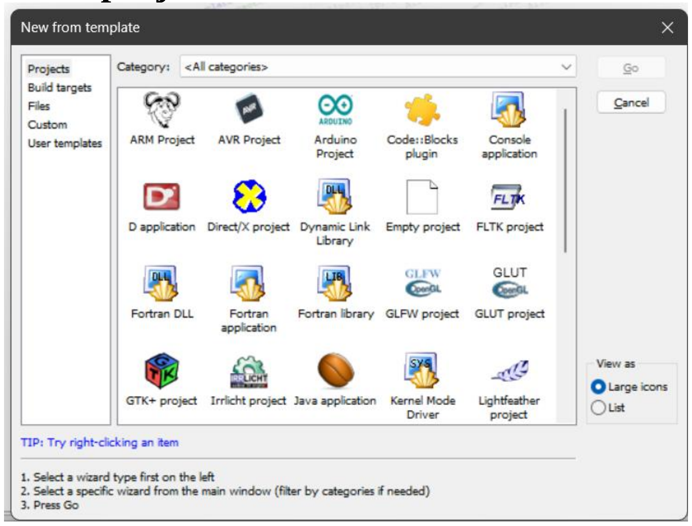
PROJECT PROCEDURE

PROCEDURE TO START A NEW PROJECT

1. Click *start*
2. Choose All Programs
3. Select Codeblocks
4. Choose File → New → Project. You get the following Dialogbox


Start here - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Management
Projects Symbols Files
Workspace
Start here
Code::Blocks
The open source, cross-platform IDE
Release 17.12 rev 11256 (2017-12-28 10:44:41) gcc 5.1.0 Windows/unicode - 32 bit
Create a new project
Open an existing project
Tip of the Day
Visit the Code::Blocks forums
Report a bug or request a new feature
Recent projects
D:\DSATM\CG\LabSessions\Progs\prog1plusone\prog1plusone.cbp
D:\DSATM\CG\LabSessions\Progs\Prog1\Prog1.cbp
D:\DSATM\CG\LabSessions\Progs\pointAttributes\pointAttributes.cbp
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck/Vera++ CppCheck/Vera++ messages Cscope Debugger DoxyBlocks Fortran info
Start here
default
36°C Sunny
Search
ENG IN
14:59 18-04-2023

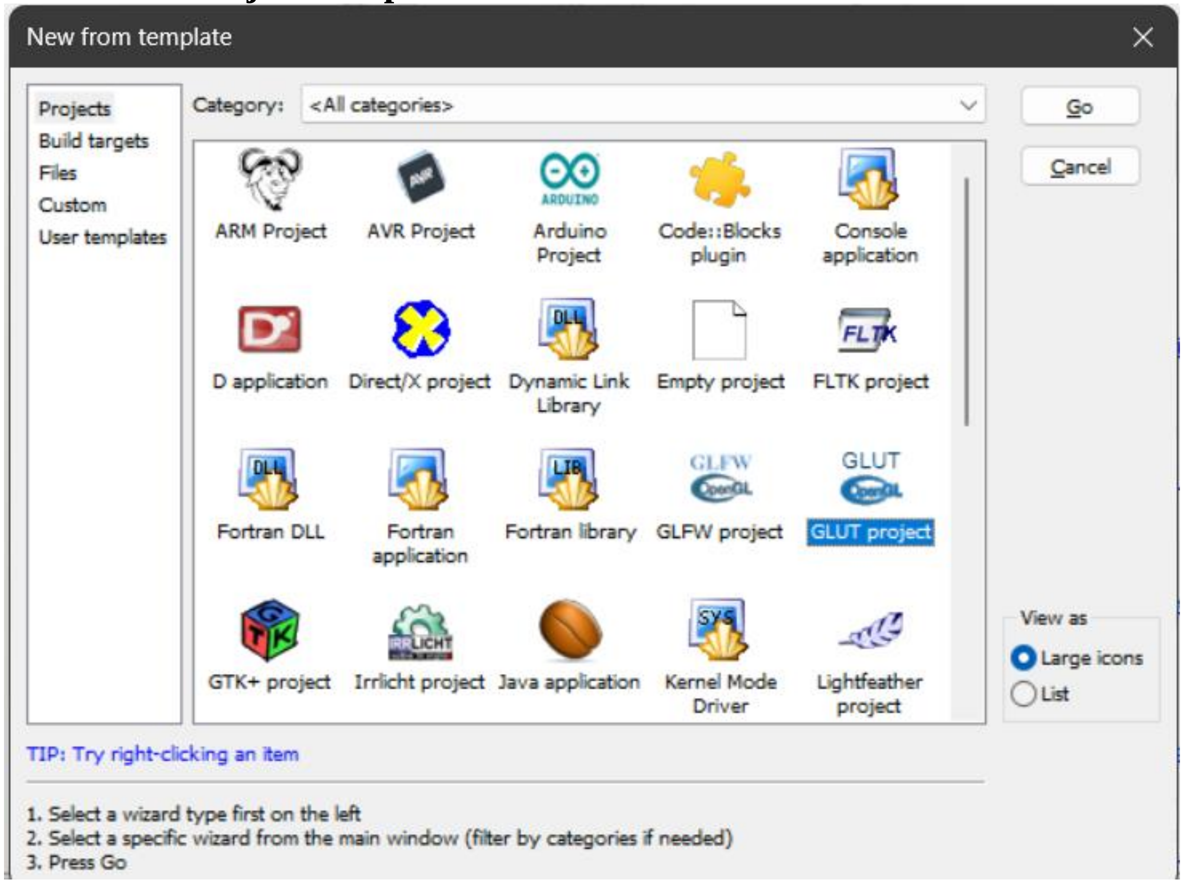
Select create new project



TIP: Try right-clicking an item

1. Select a wizard type first on the left
2. Select a specific wizard from the main window (filter by categories if needed)
3. Press Go

Select Glut Project and press Go



New from template

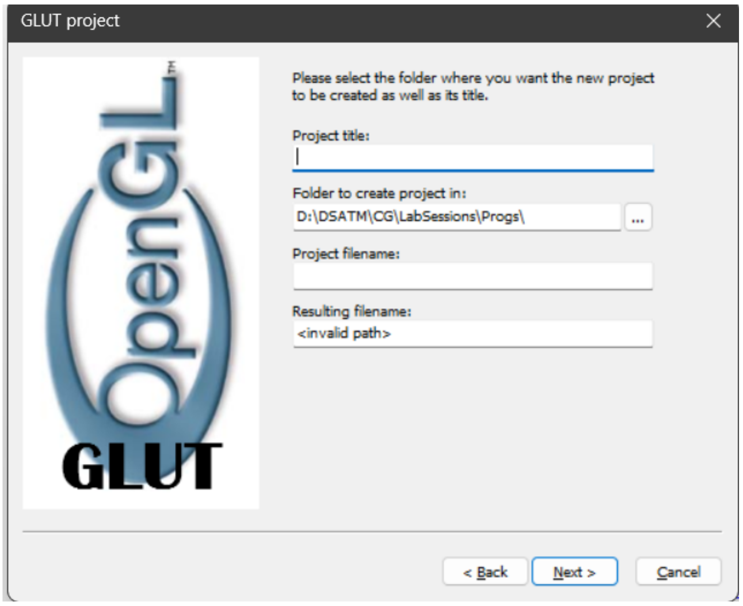
Category: <All categories>

Projects
Build targets
Files
Custom
User templates

TIP: Try right-clicking an item

1. Select a wizard type first on the left
2. Select a specific wizard from the main window (filter by categories if needed)
3. Press Go

Select the title and Finish



GLUT project

Please select the folder where you want the new project to be created as well as its title.

Project title:

Folder to create project in: D:\DSATM\CG\LabSessions\Progs\

Project filename:

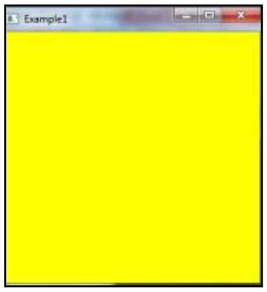
Resulting filename: <invalid path>

SAMPLE PROGRAMS

1. Creating a Window

***Note:* By default, the window has a size of (300, 300) pixels, and its position is up to the window manager to choose.**

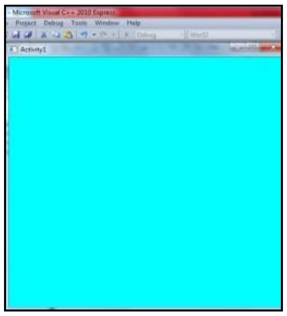
```
#include<glut.h>
void display ()  /* callback function which is called when OpenGL needs to update the display */
{
glClearColor (1.0,1.0,0.0,0.0);      /*defaultcolor–black… ....................Now set to YELLOW*/
glClear (GL_COLOR_BUFFER_BIT);             /*Clear the window-set the color of  pixels in buffer*/
glFlush();  /* Force update of screen */
}

void main (int argc, char **argv)
{
glutInit (&argc, argv);                              /* Initialise OpenGL */
glutCreateWindow ("Example1");            /* Create the window */
glutDisplayFunc (display);             /* Register the "display" function */
glutMainLoop ();              /* Enter the OpenGL main loop */
}
```



Make necessary change in program to get the following output

Change the window size to 500, 500
Change the window position to 100, 100
Change the window color to CYAN

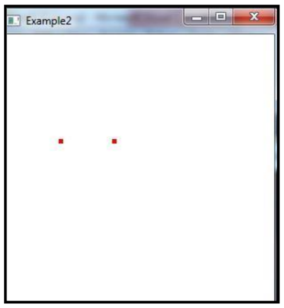
```
#include<glut.h>
void display (void)
{
      glClearColor (0.0,1.0,1.0,0.0);
      glClear(GL_COLOR_BUFFER_BIT);
      glFlush();
}
void main (int argc, char **argv)
{
      glutInit (&argc, argv);           /* Initialise OpenGL */
      glutInitWindowSize(500,500);
      glutInitWindowPosition(100,100);
```



```
glutCreateWindow ("Activity1");          /* Create the window */
glutDisplayFunc (display);               /* Register the "display" function */
glutMainLoop ();                         /* Enter the OpenGL main loop *
```

2. Drawing pixels/points

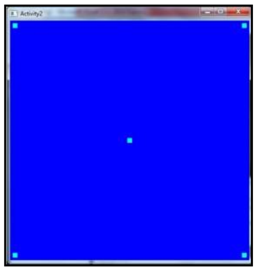
```
#include<glut.h>
void display()
{
glClear(GL_COLOR_BUFFER_BIT);
glBegin(GL_POINTS);
      glVertex2i(100,300);
      glVertex2i(201,300);
glEnd();
glFlush();
}

void myinit()
{
      glClearColor(1.0,1.0,1.0,1.0);          // set the window color to white
      glColor3f(1.0,0.0,0.0);                 // set the point color to red (RGB)
      glPointSize(5.0);                       // set the pixel size
      gluOrtho2D(0.0,500.0,0.0,500.0);        // coordinates to be used with the
                                              //viewport(left,right,bottom,top)
}
void main(int argc, char** argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);  // sets the initial display mode, GLUT single-default
      glutInitWindowSize(300,300);
      glutInitWindowPosition(0,0);
      glutCreateWindow("Example2");
      glutDisplayFunc(display);
      myinit();
      glutMainLoop();
}
```



Make necessary change in program to get the following output

Change the window color to BLUE
Change the point color to CYAN
Change the point width to 10

Draw FIVE points: at 4 corners of the window and one more at the Centre of the window.

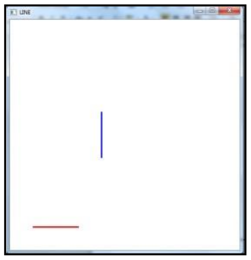


```
#include<GL/glut.h>
void display()
{
      glClear(GL_COLOR_BUFFER_BIT);
      glBegin(GL_POINTS);
            glVertex2i(10,10);
            glVertex2i(250,250);
            glVertex2i(10,490);
            glVertex2i(490,490);
            glVertex2i(490,10);
      glEnd();
      glFlush();
}
voidmyinit()
{
      glClearColor(0.0,0.0,1.0,0.0);      // set the window color to blue
      glColor3f(0.0,1.0,1.0);             // set the point color to cyan (RGB)
      glPointSize(10.0);
      gluOrtho2D(0.0,500.0,0.0,500.0);    // coordinates to be used with the viewport left, right,
                                             bottom, top)
}
void main(int argc, char** argv)
{
      glutInit(&argc,argv);
      glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
      glutInitWindowSize(500,500);
      glutInitWindowPosition(0,0);
      glutCreateWindow("Activity2");
      glutDisplayFunc(display);
      myinit();
      glutMainLoop();
}
```

3. Drawing lines

```
#include<glut.h>
void display()
{
      glClear(GL_COLOR_BUFFER_BIT);
```

```
        glColor3f(1.0,0.0,0.0);          //draw the line with red color
        glLineWidth(3.0);                // Thickness of line
        glBegin(GL_LINES);
                glVertex2d(50,50);               // to draw horizontal line in red color
                glVertex2d(150,50);
                glColor3f(0.0,0.0,1.0);          //draw the line with blue color
                glVertex2d(200,200);             // to draw vertical line in blue color
                glVertex2d(200,300);
        glEnd();
        glFlush();
}
void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,0.0,0.0);
        glPointSize(1.0);
        gluOrtho2D(0.0,500.0,0.0,500.0);
}
void main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(100,100);
        glutCreateWindow("LINE");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

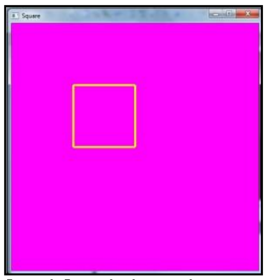


Make necessary change in program to get the following output

Change the window color to MAGENTA

Change the line color to YELLOW

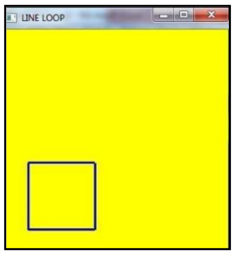
Change the line width to width to 4 Draw a square using 4lines



```
#include<glut.h>
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0,1.0,0.0);
        glLineWidth(4.0);
        glBegin(GL_LINES);
                glVertex2d (50, 100);
                glVertex2d (100,100);
                glVertex2d (100,100);
                glVertex2d (100,150);
                glVertex2d (100,150);
                glVertex2d (50,150);
                glVertex2d (50,150);
                glVertex2d (50,100);
        glEnd();
        glFlush();
    }
    void myinit()
    {
        glClearColor(1.0,0.0,1.0,1.0);
        gluOrtho2D(0.0,200.0,0.0,200.0);
    }
    void main(int argc, char** argv)
    {
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(10,100);
        glutCreateWindow("Square");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
    }
```

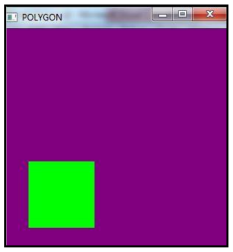
4. Drawing a square usingLINE_LOOP

```
#include<GL/glut.h>
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0, 0.0, 1.0);
        glLineWidth(3.0);
        glBegin(GL_LINE_LOOP);          // If you put GL_LINE_LOOP, it is only boundary.
                glVertex2f(50, 50);
                glVertex2f(200, 50);
                glVertex2f(200, 200);
                glVertex2f(50, 200);
        glEnd();
        glFlush();
}
void myinit()
{
        glClearColor(1.0,1.0,0.0,1.0);
        gluOrtho2D(0.0,499.0,0.0,499.0);
}

void main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(300,300);
        glutInitWindowPosition(0,0);
        glutCreateWindow("LINE LOOP");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```



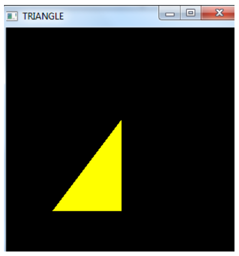
Make necessary change in program to get the following output

Change the window color to PURPLE
Change the line color to GREEN
Change the line width to width to 3
Draw a square using GL_POLYGON



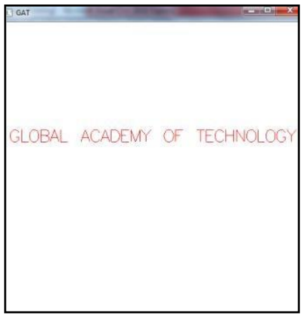
```
#include<glut.h>
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0, 1.0, 0.0);               // set line color to green
        glLineWidth(3.0);
        glBegin(GL_POLYGON);
                glVertex2f(50, 50);
                glVertex2f(200, 50);
                glVertex2f(200, 200);
                glVertex2f(50, 200);
        glEnd();
        glFlush();
}
void myinit()
{
        glClearColor(0.5,0.0,0.5,1.0);          // set window color to purple
        gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(300,300);
        glutInitWindowPosition(0,0);
        glutCreateWindow("POLYGON");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

5. Drawing a right angled triangle usingGL_TRINGLES



```
#include<glut.h>
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0, 1.0, 0.0);
        glLineWidth(3.0);
        glBegin(GL_TRIANGLES);
                glVertex2i(100,100);
                glVertex2i(250,100);
                glVertex2i(250,300);
        glEnd();
        glFlush();
}

void myinit()
{
        glClearColor(0.0,0.0,0.0,0.0);
        gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc, char** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(300,300);
        glutInitWindowPosition(0,0);
        glutCreateWindow("TRIANGLE");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

6. Writing Text



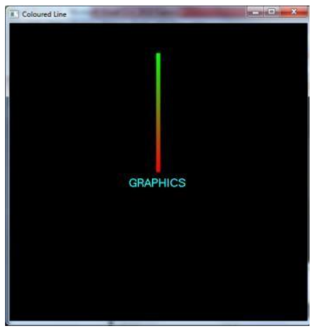
```
GLOBAL ACADEMY OF TECHNOLOGY
```

```
#include<glut.h>
void output(GLfloat x,GLfloat y,char *text)
{
        char*p;
        glPushMatrix();
        glTranslatef(x,y,0);
        glScaled(0.2,0.2,0);
        for(p=text;*p;p++)
        glutStrokeCharacter(GLUT_STROKE_ROMAN,*p);
        glPopMatrix();
}
void display
{
        glClear(GL_COLOR_BUFFER_BIT);
        output(10,300,"GLOBAL ACADEMY OF TECHNOLOGY");
        glFlush();
}
void myinit()
{
        glClearColor(1.0,1.0,1.0,1.0);
        glColor3f(1.0,0.0,0.0);
        gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc,char ** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("DSATM");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

7. Drawing colored line and writing Text

```
#include<glut.h>
#include<string.h>
```

```
char*str= "GRAPHICS";
void display()
{
        int i;
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(1.0,0.0,0.0);
        glLineWidth(10.0);
        glBegin(GL_LINES);
                glVertex2f(0.0,0.0);
                glColor3f(0.0,1.0,0.0);
                glVertex2f(0.0,0.8);
        glEnd();
        glColor3f(0.0,1.0,1.0);
        glRasterPos2f(-0.2,-0.1); //font type character to be displayed
        for(i=0;i<strlen(str);i++)
        glutBitmapCharacter(GLUT_BITMAP_HELVETICA_18,str[i]);
        glFlush();
}
void myinit()
{
        glClearColor(0.0,0.0,0.0,0.0);
        gluOrtho2D(-1.0,1.0,-1.0,1.0);
}

void main(int argc, char **argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_RGB|GLUT_SINGLE);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("Coloured Line");
        myinit();
        glutDisplayFunc(display);
        glutMainLoop();
}
```

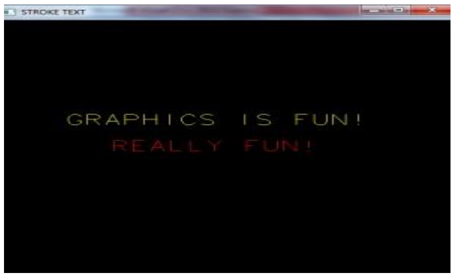


Make necessary change in program to get the following output

Change the window color to BLACK
Set the font to GLUT_STROKE_MONO_ROMAN
Display “GRAPICS IS FUN!” in YELLOW color
Display “REALLY FUN!” in RED color in the next line

```
#include<glut.h>
void output(GLfloat x,GLfloat y,char *text)
```

```
{
        char*p;
        glPushMatrix();
        glTranslatef(x,y,0);
        glScaled(0.2,0.2,0);
        for(p=text;*p;p++)
        glutStrokeCharacter( GLUT_STROKE_MONO_ROMAN,*p);
        glPopMatrix();
}
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        output(70,300,"GRAPHICS IS FUN!");
        glColor3f(1.0,0.0,0.0);
        output(120,250,"REALLY FUN!");
        glFlush();
 }
void myinit
{
        glClearColor(0.0,0.0,0.0,0.0);
        glColor3f(1.0,1.0,0.0);
        gluOrtho2D(0.0,499.0,0.0,499.0);
}

void main(int argc,char ** argv)
{
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
        glutInitWindowSize(500,500);
        glutInitWindowPosition(0,0);
        glutCreateWindow("STROKE TEXT");
        glutDisplayFunc(display);
        myinit();
        glutMainLoop();
}
```

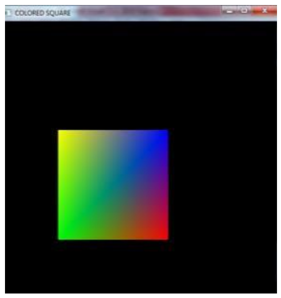


```
GRAPHICS IS FUN!
REALLY FUN!
```

8. Drawing a colored square

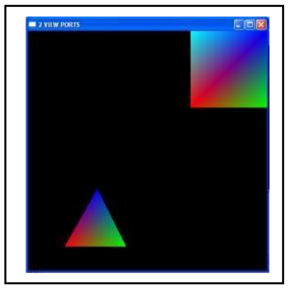
```
#include<glut.h>
void display()
{
        glClear(GL_COLOR_BUFFER_BIT);
        glColor3f(0.0, 1.0, 0.0); // Green
        glBegin(GL_POLYGON);
                glVertex2f(100, 100);
```

```
                    glColor3f(1.0,0.0,0.0); // Red
                    glVertex2f(300, 100);
                    glColor3f(0.0,0.0,1.0); // Blue
                    glVertex2f(300, 300);
                    glColor3f(1.0,1.0,0.0); // Yellow
                    glVertex2f(100, 300);
          glEnd();
          glFlush();
}
void myinit()
{
          glClearColor(0.0,0.0,0.0,1.0);
          glColor3f(1.0,0.0,0.0);// Red
          gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc, char** argv)
{
          glutInit(&argc,argv);
          glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
          glutInitWindowSize(500,500);
          glutInitWindowPosition(0,0);
          glutCreateWindow("COLORED SQUARE");
          glutDisplayFunc(display);
          myinit();
          glutMainLoop();
}
```



9. Creating 2 view ports

```
#include<glut.h>
void display()
{
          glClear(GL_COLOR_BUFFER_BIT);
          glViewport (5,-150,400,400);
          glBegin(GL_POLYGON);
                    glColor3f(1.0,0.0,0.0);
                    glVertex2f(90,250);
                    glColor3f(0.0,1.0,0.0);
                    glVertex2f(250,250);
                    glColor3f(0.0,0.0,1.0);
                    glVertex2f(175,400);
          glEnd();
          glViewport (300,300,400,400);
          glBegin(GL_POLYGON);
                    glColor3f(1.0,0.0,0.0);
```

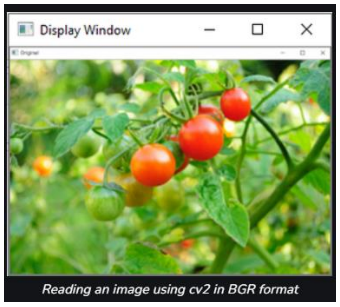


```
			glVertex2f(50,50);
			glColor3f(0.0,1.0,0.0);
			glVertex2f(250,50);
			glColor3f(0.0,0.0,1.0);
			glVertex2f(250,250);
			glColor3f(0.0,1.0,1.0);
	glVertex2f(50,250);
	glEnd();
	glFlush();
}
void myinit()
{
	glClearColor(0.0,0.0,0.0,1.0);
	gluOrtho2D(0.0,499.0,0.0,499.0);
}
void main(int argc, char** argv)
{
	glutInit(&argc,argv);
	glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
	glutInitWindowSize(500,500);
	glutInitWindowPosition(0,0);
	glutCreateWindow("2 VIEW PORTS");
	glutDisplayFunc(display);
	myinit();
	glutMainLoop();
}
```

10. Program to read and display an image using opencv

```
# Program 1
import cv2

img = cv2.imread("PASTE THE PATH YOU WANT TO READ", cv2.IMREAD_COLOR)
cv2.imshow("image", img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

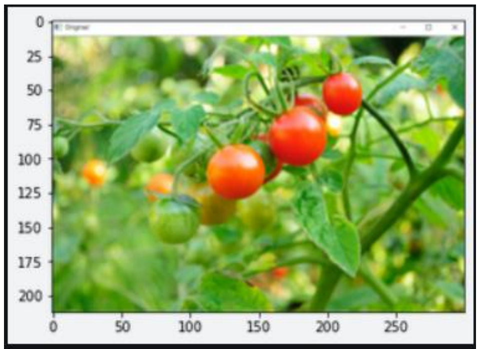
Output:



11. **Program to read image in RGB**

```
import cv2

import numpy as np

import matplotlib.pyplot as plt

img=cv2.imread("geeks.png")


RGB_img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

plt.imshow(RGB_img)

plt.waitforbuttonpress()

plt.close('all')
```

Output:



BGR color format changes to RGB color format

VTU SYLLABUS PROGRAMS

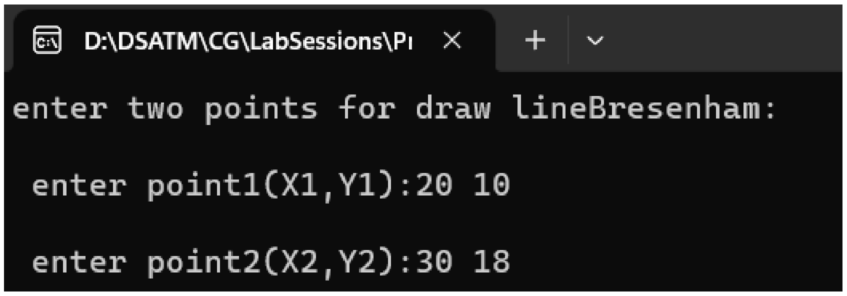
Program 1

Develop a program to draw a line using Bresenham’s line drawing technique

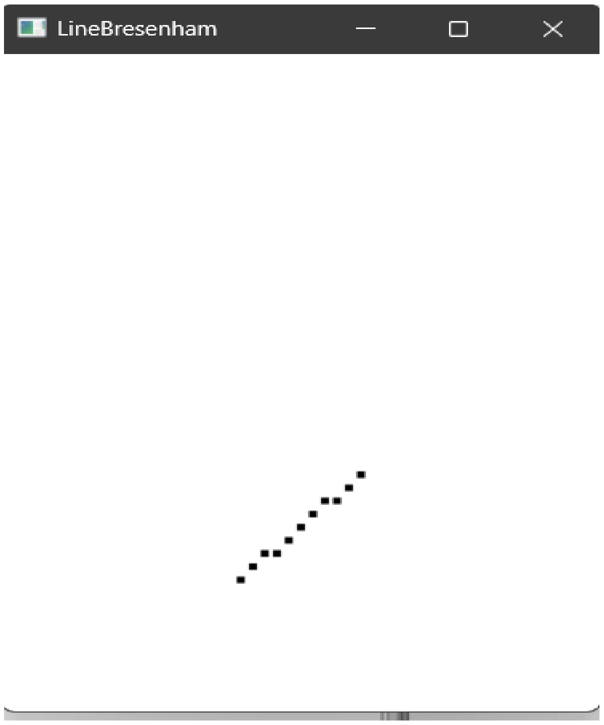
Bresenham's line algorithm is named after Jack Elton Bresenham who developed it in 1962 at IBM. It is commonly used to draw line primitives in a bitmap image (e.g. on a computer screen), as it uses only integer addition, subtraction and bit shifting, all of which are very cheap operations in standard computer architectures. It is an incremental error algorithm.

```
#include<windows.h>
#include<stdio.h>
#include<math.h>
#include<gl/glut.h>

GLint X1,Y1,X2,Y2;

void LineBres(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	int dx=abs(X2-X1),dy=abs(Y2-Y1);
	int p=2*dy-dx;
	int twoDy=2*dy,twoDyDx=2*(dy-dx);
	int x,y;

	if(X1>X2)
	{
		x=X2;
		y=Y2;
		X2=X1;
	}
	else
	{
		x=X1;
		y=Y1;
		X2=X2;
	}
	glBegin(GL_POINTS);
	glVertex2i(x,y);
	while(x<X2)
	{
		x++;
		if(p<0)
```

```
                p+=twoDy;
            else
            {
                y++;
                p+=twoDyDx;
            }
            glVertex2i(x,y);
        }
glEnd();
glFlush();
}

void Init()
{
        glClearColor(1.0,1.0,1.0,0);
        glColor3f(0.0,0.0,0.0);
        glPointSize(4.0);
        glViewport(0,0,50,50);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0,50,0,50);
}
int main(int argc,char **argv)
{
        printf("enter two points for draw lineBresenham:\n");
        printf("\n enter point1(X1,Y1):");
        scanf("%d%d",&X1,&Y1);
        printf("\n enter point2(X2,Y2):");
        scanf("%d%d",&X2,&Y2);
        glutInit(&argc,argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(300,400);
        glutInitWindowPosition(0,0);
        glutCreateWindow("LineBresenham");
        Init();
        glutDisplayFunc(LineBres);
        glutMainLoop();
}
```



```
enter two points for draw lineBresenham:

 enter point1(X1,Y1):20 10

 enter point2(X2,Y2):30 18
```

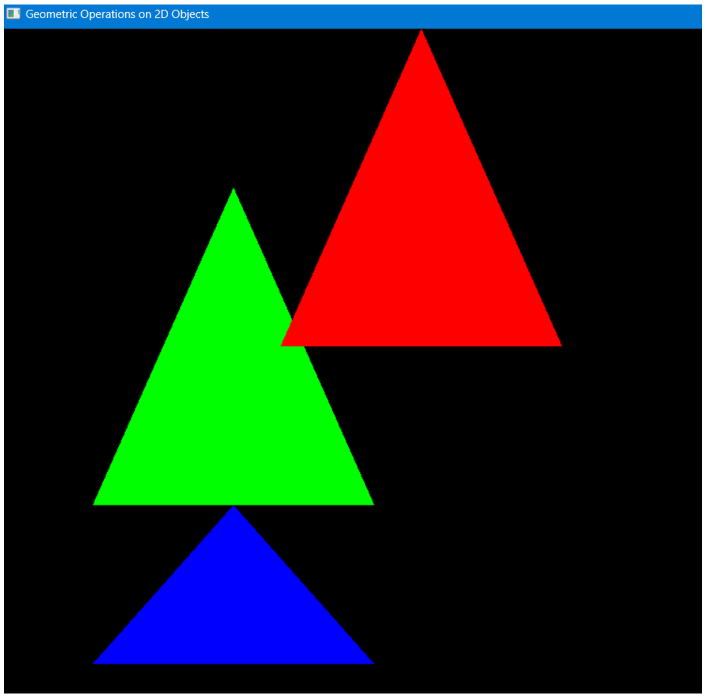


Program 2

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

```
#include<windows.h>
#include<stdio.h>
#include<math.h>
#include<gl/glut.h>

void display()
{

        glClear(GL_COLOR_BUFFER_BIT);
        glClearColor(1.0, 1.0, 1.0, 1);
        glMatrixMode(GL_MODELVIEW);

        int p1[] = {50, 100};
        int p2[] = {200, 100};
        int p3[] = {125, 200};

        glColor3f(0.0,0.0,1.0);
        glBegin(GL_TRIANGLES);
        glVertex2iv(p1);
        glVertex2iv(p2);
        glVertex2iv(p3);
        glEnd();

        int sx=1; int sy=2;

           p1[0]= p1[0] * sx;
           p1[1]= p1[1] * sy;

           p2[0]= p2[0] * sx;
           p2[1]= p2[1] * sy;

           p3[0]= p3[0] * sx;
           p3[1]= p3[1] * sy;
        glColor3f(0.0,1.0,0.0);
        glBegin(GL_TRIANGLES);
        glVertex2iv(p1);
         glVertex2iv(p2);
         glVertex2iv(p3);
         glEnd();

        int tx=100; int ty=100;

           p1[0]= p1[0] + tx;
           p1[1]= p1[1] + ty;

           p2[0]= p2[0] + tx;
           p2[1]= p2[1] + ty;

           p3[0]= p3[0] + tx;
           p3[1]= p3[1] + ty;

              glColor3f(1.0,0.0,0.0);
              glBegin(GL_TRIANGLES);
              glVertex2iv(p1);
              glVertex2iv(p2);
              glVertex2iv(p3);
              glEnd();
        glFlush();
}

int main(int argc, char**argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
        glutInitWindowSize(1000, 1000);
        glutCreateWindow("Geometric Operations on 2D Objects");
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0, 500, 0, 500);
        glutDisplayFunc(display);
        glutMainLoop();
}
```

OUTPUT

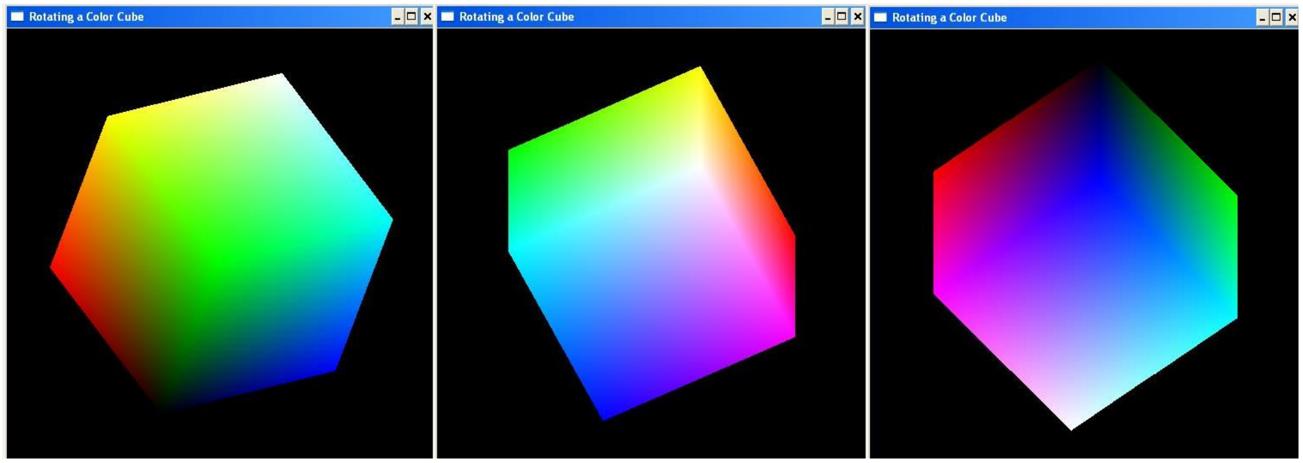


Program 3

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

```
#include<stdlib.h>
#include<stdio.h>
#include<GL/glut.h>
#include<math.h>
float v[8][3]={{-1,-1,1},{-1,1,1},{1,1,1},{1,-1,1},
                              {-1,-1,-1},{-1,1,-1},{1,1,-1},{1,-1,-1}};
float c[8][3]={{0,0,1},{0,1,1},{1,1,1},{1,0,1},
                         {0,0,0},{0,1,0},{1,1,0},{1,0,0}};
float theta[3]={0,0,0};
int flag=2;
void myinit()
{
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        glOrtho(-2,2,-2,2,-2,2);
        glMatrixMode(GL_MODELVIEW);
}

void idlefunc()
{
        theta[flag]++;
        if(theta[flag]>360)theta[flag]=0;
        glutPostRedisplay();
}
void mousefunc(int button,int status,int x,int y)
{
        if(button==GLUT_LEFT_BUTTON&&status==GLUT_DOWN)
                flag=2;
        if(button==GLUT_MIDDLE_BUTTON&&status==GLUT_DOWN)
                flag=1;
        if(button==GLUT_RIGHT_BUTTON&&status==GLUT_DOWN)
                flag=0;

}
void drawpoly(int a,int b,int c1,int d)
{
        glBegin(GL_POLYGON);
                glColor3fv(c[a]);
                glVertex3fv(v[a]);
                glColor3fv(c[b]);
                glVertex3fv(v[b]);
                glColor3fv(c[c1]);
                glVertex3fv(v[c1]);
                glColor3fv(c[d]);
                glVertex3fv(v[d]);
        glEnd();
}
void colorcube()
{
        drawpoly(0,1,2,3);
        drawpoly(0,1,5,4);
        drawpoly(1,5,6,2);
        drawpoly(4,5,6,7);
        drawpoly(3,2,6,7);
        drawpoly(0,4,7,3);
}
void display()
{
        glClearColor(1,1,1,1);
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glColor3f(1,0,0);
        glEnable(GL_DEPTH_TEST);
        glLoadIdentity();
        glRotatef(theta[0],1,0,0);//x
        glRotatef(theta[1],0,1,0);//y
        glRotatef(theta[2],0,0,1);//z
        colorcube();
        glFlush();
        glutSwapBuffers();
}
void main()
{
                glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE |GLUT_DEPTH);
                glutInitWindowPosition(100,100);
                glutInitWindowSize(500,500);
                glutCreateWindow("cube rotation");
                myinit();

                glutDisplayFunc(display);
                glutMouseFunc(mousefunc);
                glutIdleFunc(idlefunc);
                glutMainLoop();
}
```

OUTPUT

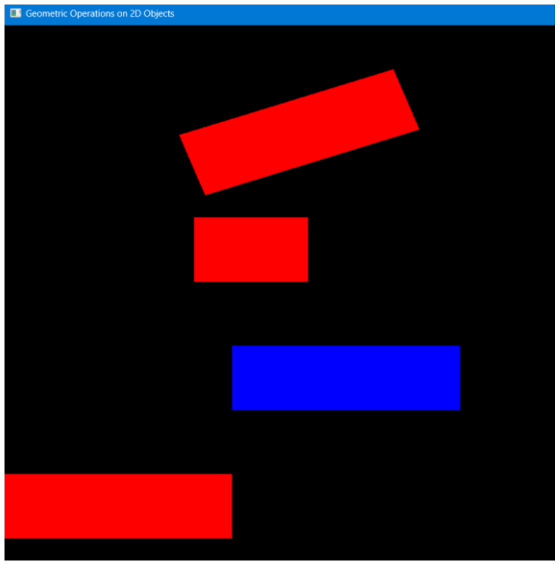


Program 4

Develop a program to demonstrate 2D transformation on basic objects

```
#include<windows.h>
#include<stdio.h>
#include<math.h>
#include<gl/glut.h>

void display()
{

glClear(GL_COLOR_BUFFER_BIT);
glClearColor(1.0, 1.0, 1.0, 1);
glMatrixMode(GL_MODELVIEW);

glColor3f(0.0,0.0,1.0);
glRecti(250,300,400,350);

glPushMatrix();
/* Translate Operation*/
glColor3f(1.0,0.0,0.0);
glTranslatef(-100.0,-50,0.0);
glRecti(250,300,400,350);
glPopMatrix();


glPushMatrix();
/* Rotate Operation*/
glRotatef(20, 0.0, 0.0, 1.0);
glRecti(250,300,400,350);
glPopMatrix();


glPushMatrix();
/* Scale Operation*/
glScalef(0.5, 1.0, 1.0);
glRecti(250,300,400,350);
glPopMatrix();


glFlush();

}
int main(int argc, char**argv)
{
glutInit(&argc, argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(1000, 1000);
glutCreateWindow("Geometric Operations on 2D Objects");
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0, 500, 0, 500);
glutDisplayFunc(display);
glutMainLoop();
}
```

OUTPUT



<u>Program 5</u>

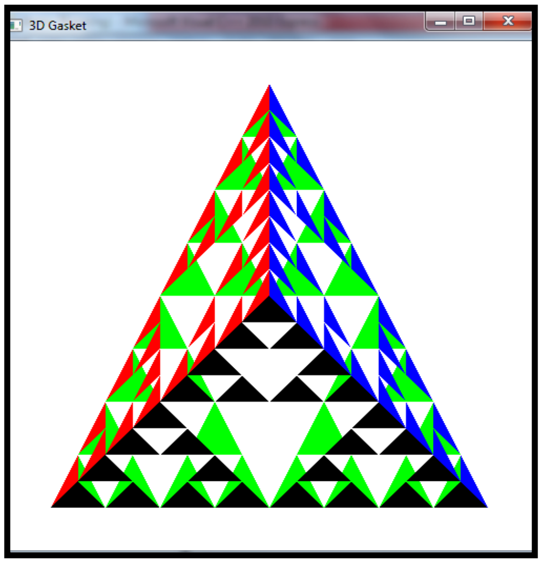
Develop a program to demonstrate 3D transformation on 3D objects

```
#include<stdio.h>
#include<GL/glut.h>
float v[4][3] = { {-1,-0.25,0},{1,-0.25,0}, {0,1,0}, {0,0,1} };
int n;
void myinit()
{
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   glOrtho(-2, 2, -2, 2, -2, 2);
   glMatrixMode(GL_MODELVIEW);
}
void draw_triangle(float *a,float *b, float * c)
{
   glBegin(GL_POLYGON);
   glVertex3fv(a);
   glVertex3fv(b);
   glVertex3fv(c);
   glEnd();
}
```

```
void draw_tetera(float *a, float * b, float * c, float * d)
{
   glColor3f(1.0, 0.0, 0.0);
   draw_triangle(a, b, c);
   glColor3f(0.0, 0.0, 1.0);
   draw_triangle(a, c, d);
   glColor3f(0.0, 1.0, 0.0);
   draw_triangle(c, b, d);
   glColor3f(0.0, 0.0, 0.0);
   draw_triangle(a, b, d);
}



void divide_tetrahedron(float *a, float * b, float * c, float * d, int m)
{
   float ab[3], ac[3], ad[3], bc[3], bd[3], cd[3];
   int j;
   if (m > 0) {
          for (j = 0; j < 3; j++)
          {
                 ab[j] = (a[j] + b[j]) / 2;
                 ac[j] = (a[j] + c[j]) / 2;
                 ad[j] = (a[j] + d[j]) / 2;
                 bc[j] = (b[j] + c[j]) / 2;
                 bd[j] = (d[j] + b[j]) / 2;
                 cd[j] = (c[j] + d[j]) / 2;

          }
          // consider midpoints as vertex and divide bigger triangle
          // to 3 parts recursively
          divide_tetrahedron(a, ab, ac, ad, m - 1);
          divide_tetrahedron(ab, b, bc, bd, m - 1);
          divide_tetrahedron(ac, bc, c, cd, m - 1);
          divide_tetrahedron(ad, bd, cd, d, m - 1);
          //note if u want the colors of phases to align just adjudt the above points
   }
   //draw the sub divided traingles
   else {
          draw_tetera(a, b, c, d);
   }
}

//this is called everytime the display is refreshed. Here it draw a tetrahedron.
```

```
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();
    divide_tetrahedron(v[0], v[1], v[2], v[3], n);
    glFlush();
}

// This function is executed when the wiindow size is changed.


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

    n = 2;
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(500, 500);
    glutCreateWindow("3DGasket"); //window with a title
    myinit();
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glClearColor(1.0, 1.0, 1.0, 1.0);
    glutMainLoop();
}
```

OUTPUT



Program 6

Develop a program to demonstrate Animation effects on simple objects.

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

#define PI 3.1416

GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;
static int window;
static int menu_id = 2;
static int submenu_id = 1;
static int value = 0;

typedef structwcPt3D
{
	GLfloat x, y,z;
};

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

voidcomputeBezPt(GLfloat u, wcPt3D *bezPt, GLint nCtrlPts, wcPt3D *ctrlPts, GLint *C)
{
	GLint k, n = nCtrlPts - 1;
	GLfloat bezBlendFcn;
	bezPt->x = bezPt->y = bezPt->z = 0.0;
	for (k = 0; k<nCtrlPts; k++)
	{
```

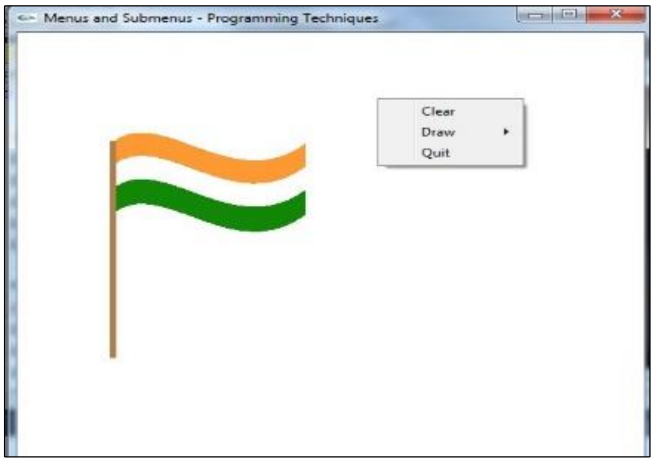
```
                bezBlendFcn = C[k] * pow(u, k) * pow(1 - u, n - k);
                bezPt->x += ctrlPts[k].x *bezBlendFcn;
                bezPt->y += ctrlPts[k].y *bezBlendFcn;
                bezPt->z += ctrlPts[k].z *bezBlendFcn;
        }
}
voidbezier(wcPt3D *ctrlPts, GLint nCtrlPts, GLint nBezCurvePts)
{
        wcPt3DbezCurvePt;
        GLfloat u;
        GLint *C, k;
        C = new GLint[nCtrlPts];
        bino(nCtrlPts - 1, C);
        glBegin(GL_LINE_STRIP)
        ;
        for(k = 0; k <= nBezCurvePts; k++)
        {
                u = GLfloat(k) / GLfloat(nBezCurvePts);
                computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);
                glVertex2f(bezCurvePt.x, bezCurvePt.y);
        }
        glEnd();
        delete[]C
        ;
}

voiddisplayFunc()
{
        GLint nCtrlPts = 4, nBezCurvePts = 20;
        static float theta =0;
        wcPt3DctrlPts[4] ={
                { 20, 100, 0},
                { 30, 110, 0},
                { 50, 90, 0 },
                { 60, 100, 0 } };
        ctrlPts[1].x += 10 * sin(theta * PI / 180.0);
        ctrlPts[1].y += 5 * sin(theta * PI / 180.0);
        ctrlPts[2].x -= 10 * sin((theta + 30) * PI / 180.0);
        ctrlPts[2].y -= 10 * sin((theta + 30) * PI / 180.0);
        ctrlPts[3].x -= 4 * sin((theta)* PI / 180.0);
        ctrlPts[3].y += sin((theta - 30) * PI / 180.0);
```

```
	theta += 0.1;
	glClear(GL_COLOR_BUFFER_BIT);
	glColor3f(1.0, 1.0, 1.0);
	glPointSize(5);
	glPushMatrix();
	glLineWidth(5);
	glColor3f(255 / 255, 153 / 255.0, 51 / 255.0);			//Indian flag: Orange color code
	for (int i = 0; i<8; i++)
	{
		glTranslatef(0, -0.8, 0);
		bezier(ctrlPts, nCtrlPts,nBezCurvePts);
	}
	glColor3f(1, 1, 1); //Indian flag: white colorcode
	for(int i = 0; i<8; i++)
	{
		glTranslatef(0, -0.8, 0);
		bezier(ctrlPts, nCtrlPts, nBezCurvePts);
	}
	glColor3f(19 / 255.0, 136 / 255.0, 8 / 255.0); //Indian flag: green color code
	for (int i = 0; i<8; i++)
	{
		glTranslatef(0, -0.8, 0);
		bezier(ctrlPts, nCtrlPts, nBezCurvePts);
	}
	glPopMatrix();
	glColor3f(0.7, 0.5, 0.3);
	glLineWidth(5);
	glBegin(GL_LINES);
		glVertex2f(20, 100);
		glVertex2f(20, 40);
	glEnd();
	glFlush();
	glutPostRedisplay();
	glutSwapBuffers();
}
voidwinReshapeFun(GLint newWidth, GLint newHeight)
{
	glViewport(0, 0, newWidth, newHeight);
	glMatrixMode(GL_PROJECTION);
	glLoadIdentity();
	gluOrtho2D(xwcMin, xwcMax, ywcMin, ywcMax);
	glClear(GL_COLOR_BUFFER_BIT);
}
```

```
voiddisplay(void)
{
        glClear(GL_COLOR_BUFFER_BIT);
        if (value == 1)
        {
                glutPostRedisplay();
        }
        else if (value == 2)
        {
                glPushMatrix();
                glColor3d(1.0, 0.0, 0.0);
                glutDisplayFunc(displayFunc);
                //glutWireSphere(0.5, 50, 50);
                glPopMatrix();
        }
        glFlush();
}

voidmenu(int num)
{
        if(num == 0)
        {
                glutDestroyWindow(window);
                exit(0);
        }
        else
        {
                value = num;
        }
        glutPostRedisplay();
}

voidcreateMenu(void)
{
        submenu_id = glutCreateMenu(menu);
        glutAddMenuEntry("draw a flag", 2);
        menu_id = glutCreateMenu(menu);
        glutAddMenuEntry("Clear", 1);
        glutAddSubMenu("Draw", submenu_id);
        glutAddMenuEntry("Quit", 0);
        glutAttachMenu(GLUT_RIGHT_BUTTON);
}
```

```
voidmyinit()
{
        glViewport(0, 0, 500, 500);
        glClearColor(1.0, 1.0, 1.0, 1.0);
        glColor3f(1.0, 0.0, 0.0);
        glMatrixMode(GL_PROJECTION);
        glLoadIdentity();
        gluOrtho2D(0.0, 499.0, 0.0, 499.0);
}

intmain(int argc, char **argv)
{
        glutInit(&argc, argv);
        glutInitDisplayMode(GLUT_RGBA | GLUT_SINGLE);
        glutInitWindowSize(500, 500);
        glutInitWindowPosition(100, 100);
        window = glutCreateWindow("Menus and Submenus - Programming Techniques");
        createMenu();
        glClearColor(0.0, 0.0, 0.0, 0.0);
        glutDisplayFunc(display);
        //glutDisplayFunc(displayFunc);
        glutReshapeFunc(winReshapeFun);
        myinit();
        glutMainLoop();
}
```

OUTPUT

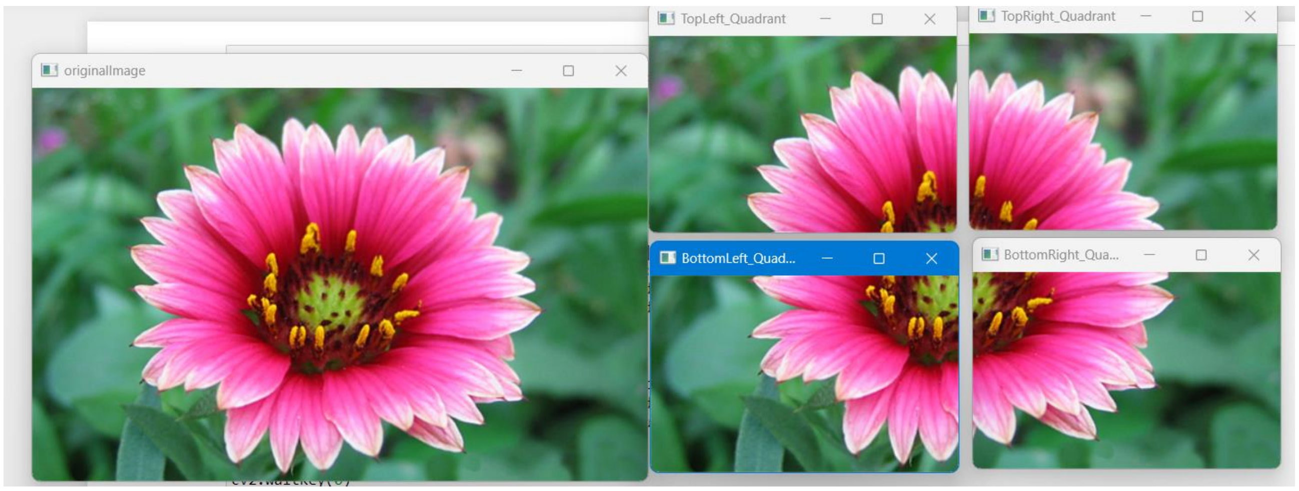


Program 7

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

```
import cv2
img = cv2.imread("D:/DSATM/CG/CG&DIP/Lab_Images/Prog7_flower.jpg")
original= img
h, w, channels = img.shape
half_width = w//2
half_height = h//2

TopLeft_quadrant = img[:half_height, :half_width]
TopRight_quadrant = img[:half_height, half_width:]
BottomLeft_quadrant = img[half_height:, :half_width]
BottomRight_quadrant = img[half_height:, half_width:]

cv2.imshow('originalImage',original)
cv2.imshow('TopLeft_Quadrant', TopLeft_quadrant)
cv2.imshow('TopRight_Quadrant', TopRight_quadrant)
cv2.imshow('BottomLeft_Quadrant', BottomLeft_quadrant)
cv2.imshow('BottomRight_Quadrant', BottomRight_quadrant)

cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:



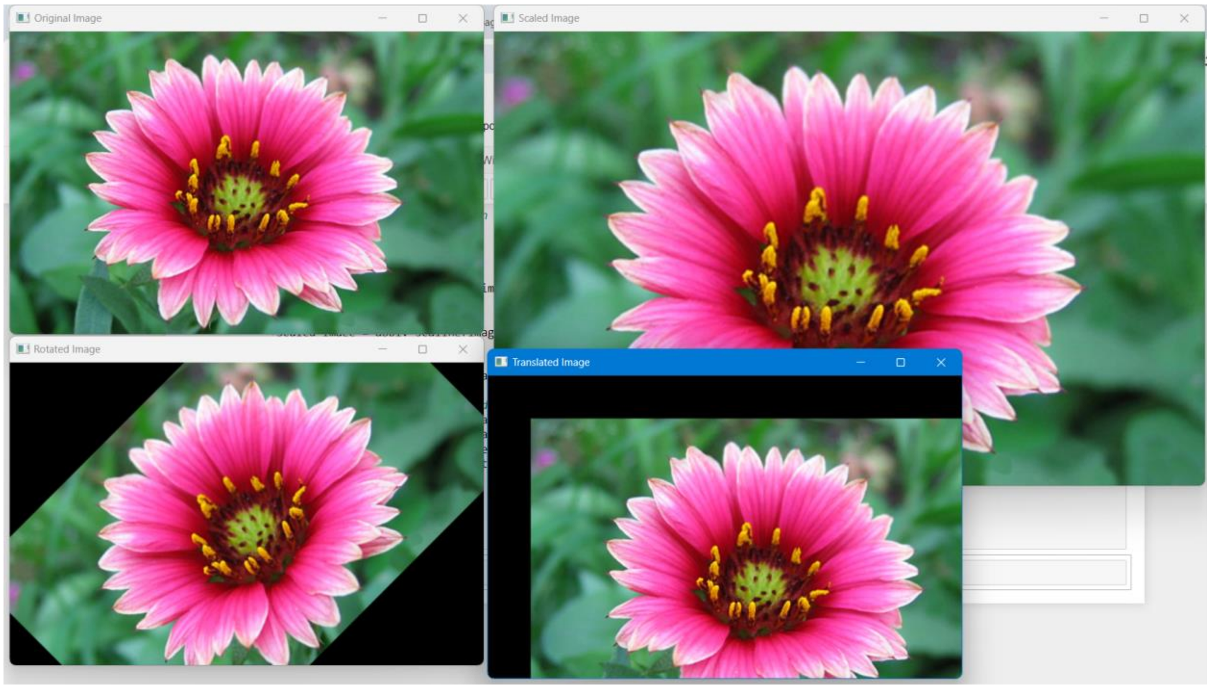
Program 8

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

```
import cv2
import numpy as np

def apply_rotation(image, angle):
    height, width = image.shape[:2]
    # Calculate the rotation matrix
    rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), angle, 1)
    rotated_image = cv2.warpAffine(image, rotation_matrix, (width, height))
    return rotated_image

def apply_scaling(image, scale_x, scale_y):
    scaled_image = cv2.resize(image, None, fx=scale_x, fy=scale_y)
    return scaled_image

def apply_translation(image, tx, ty):
    translation_matrix = np.float32([[1, 0, tx], [0, 1, ty]])
    translated_image = cv2.warpAffine(image, translation_matrix, (image.shape[1], image.shape[0]))
    return translated_image

image = cv2.imread('D:/DSATM/CG/CG&DIP/Lab_Images/Prog7_flower.jpg')

# Define the angle for rotation (in degrees)
angle = 45

# Define scaling factors
scale_x = 1.5
scale_y = 1.5

# Define translation amounts (in pixels)
tx = 50
ty = 50

# Apply rotation
rotated_image = apply_rotation(image, angle)

# Apply scaling
scaled_image = apply_scaling(image, scale_x, scale_y)

# Apply translation
translated_image = apply_translation(image, tx, ty)
```

```
# Display the original image and the modified images
cv2.imshow('Original Image', image)
cv2.imshow('Rotated Image', rotated_image)
cv2.imshow('Scaled Image', scaled_image)
cv2.imshow('Translated Image', translated_image)

# Wait for a key press and then close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT

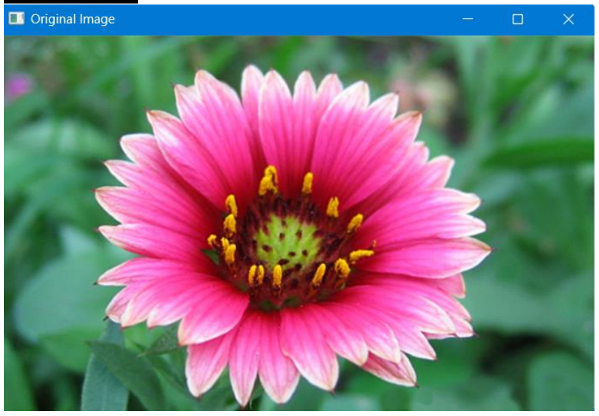


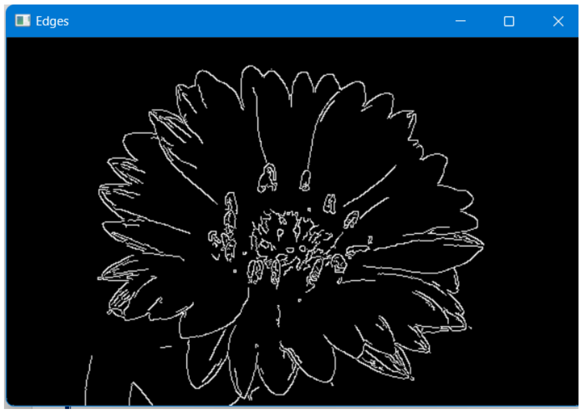
Program 9

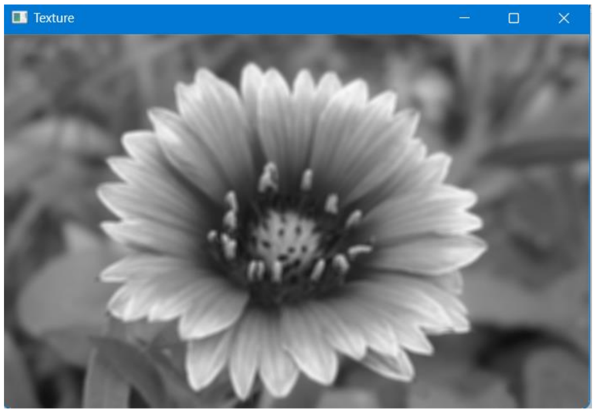
Read an image and extract and display low-level features such as edges, textures using filtering techniques.

```
import cv2
import numpy as np

# Load the image
img = cv2.imread("D:/DSATM/CG/CG&DIP/Lab_Images/Prog7_flower.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

# Edge detection
edges = cv2.Canny(gray, 100, 200)

# Texture extraction
kernel = np.ones((5, 5), np.float32) / 25 # Define a 5x5 averaging kernel
texture = cv2.filter2D(gray, -1, kernel) # Apply the averaging filter for texture extraction

# Display the original image, edges, and texture
cv2.imshow("Original Image", img)
cv2.imshow("Edges", edges)
cv2.imshow("Texture", texture)

# Wait for a key press and then close all windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT





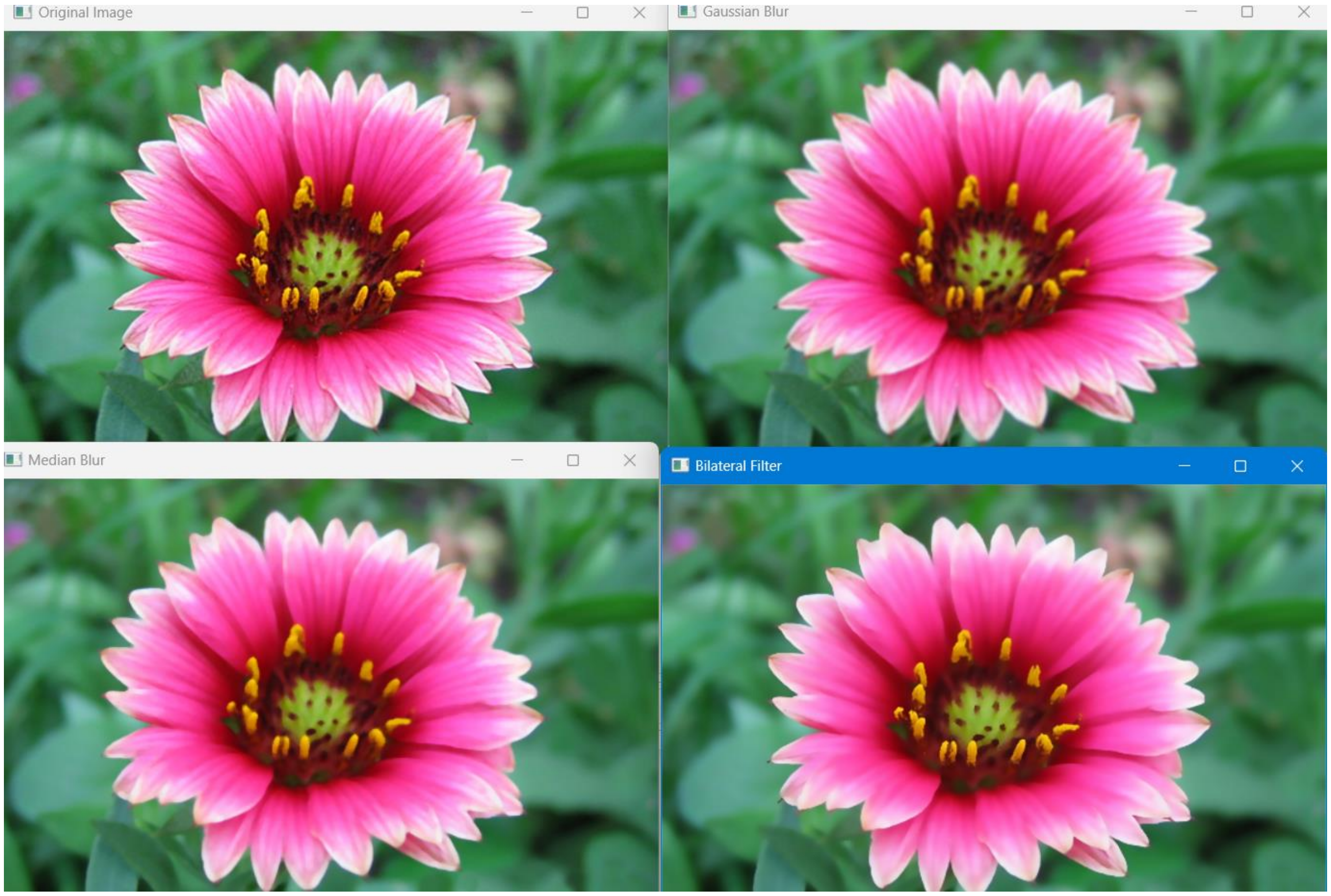


Program 10

Write a program to blur and smoothing an image

```
import cv2

# Load the image
image = cv2.imread("D:/DSATM/CG/CG&DIP/Lab_Images/Prog7_flower.jpg")


# Gaussian Blur
gaussian_blur = cv2.GaussianBlur(image, (5, 5), 0)

# Median Blur
median_blur = cv2.medianBlur(image, 5)

# Bilateral Filter  Smoothing
bilateral_filter = cv2.bilateralFilter(image, 9, 75, 75)

# Display the original and processed images
cv2.imshow('Original Image', image)
cv2.imshow('Gaussian Blur', gaussian_blur)
cv2.imshow('Median Blur', median_blur)
cv2.imshow('Bilateral Filter', bilateral_filter)

# Wait for a key press to close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT



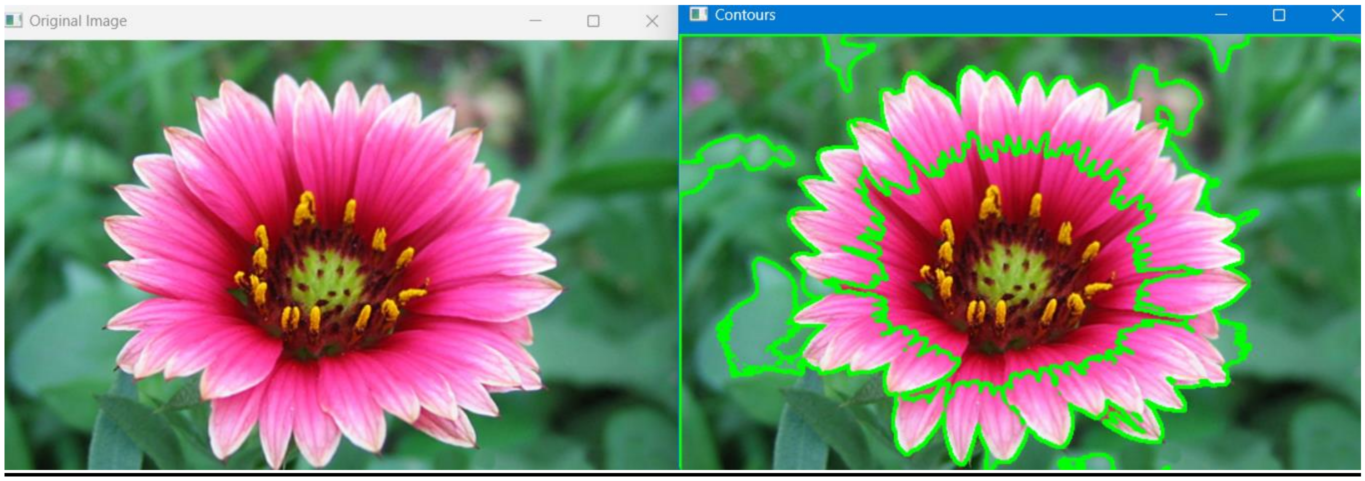
Program 11

Write a program to contour an image.

```
import cv2
import numpy as np

# Load the image
image =
cv2.imread("D:/DSATM/CG/CG&DIP/Lab_Images/Prog7_flower.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply binary thresholding
```

```
ret, thresh = cv2.threshold(gray, 0, 255, cv2.THRESH_BINARY_INV +
cv2.THRESH_OTSU)

# Find contours
contours, hierarchy = cv2.findContours(thresh, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)


# Create a copy of the original image to draw contours on
contour_image = image.copy()

# Draw contours on the image
cv2.drawContours(contour_image, contours, -1, (0, 255, 0), 2)

# Display the original and contour images
cv2.imshow('Original Image', image)
cv2.imshow('Contours', contour_image)

# Wait for a key press to close the windows
cv2.waitKey(0)
cv2.destroyAllWindows()
```

<u>OUTPUT</u>

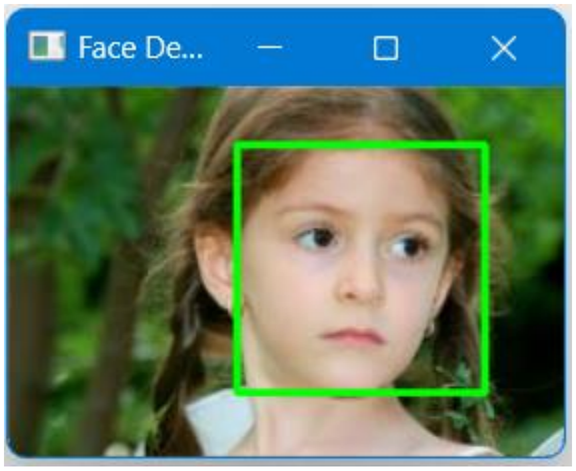


Program 12

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

```
import cv2

# Load the cascade classifier for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')


# Load the image
image = cv2.imread("D:/DSATM/CG/CG&DIP/Lab_Images/girlFace.jpg")

# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Detect faces in the grayscale 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), (0, 255, 0), 2)

# Display the image with detected faces

cv2.imshow('Face Detection', image)

# Wait for a key press to close the window
cv2.waitKey(0)
cv2.destroyAllWindows()
```

OUTPUT



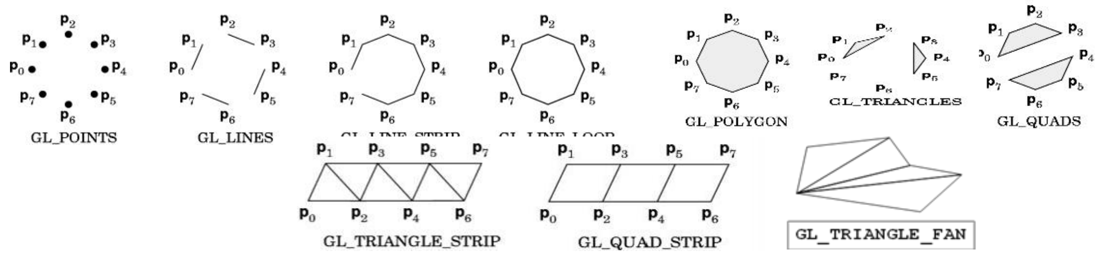
Viva Questions

1. **Specify the default values for thefollowing**

WindowPosition:	0, 0 from Top-LeftCorner
WindowSize:	300,300 from Top-LeftCorner
Background Color ofthewindow:	Black
Foreground Color ofthewindow:	White
RGBColorvalues:	Each1.0
ImagePosition:	**Bottom-Left if only one quadrant is used.** Ex : glVertex3i(70, 80), glVertex3f(7.5, 8.0) **Centre of the Screen if 4 quadrants are used.** Ex : glVertex3f(-5.0, 3.0, 0.0)

2. **What would happen to the RGB color values set to 3.0 OR-7.0?**
   If the color is set to 3.0, it is reset to 1.0 If the
   color is set to -7.0, it is reset to 0.0
   Each color takes a min value as 0.0 and max value as 1.0

3. **List differentprimitives.**
   - **GL_POINTS: Each** vertex is displayed at a size of at least one pixel.
   - **GL_LINES: Successive** pairs of vertices would be connected as aline.
   - **GL_LINE_STRIP: Connects** the successive vertices using line segments. However the final vertex would not be connected to the initialvertex.
   - **GL_LINE_LOOP: Connects** the successive vertices using line segments to form a closedpath.
   - **GL_POLYGON:** Connects the successive vertices using line segments to forma closed path. The interior is filled according to the state of the relevantattributes.
   - **GL_QUADS:** Special case of polygon where successive group of 4 vertices are interpreted asquadrilaterals.
   - **GL_TRIANGLES:** Special case of polygon where successive group of three vertices would be interpreted as atriangle.
   - **GL_TRIANGLE_STRIP:** Each additional vertex is combined with the previous two vertices to define a newtriangle.
   - **GL_QUAD_STRIP: We**combine two new vertices with the previous two verticesto design a newquadrilateral.
   - **GL_TRIANGLE_FAN:** Based on one fixed point. The next two points determine the first triangle. The subsequent triangles are formed from one new point, the previous point and the first (fixed)point.



4. **What is the use ofglVertex3fv()?**
A Vertex is used to define geometric primitives.
In OpenGL a vertex can be represented as **glVertex*()** where * can be nt OR ntv
n : no. of dimensions (2, 3 , 4)
t : data types (int, float,double)
v : specifies that variables are specified through a pointer to an array.

5. **How to set the background color of the window toCYAN?**
glClearColor(0.0, 1.0,1.0,1.0);
The first 3 arguments represent RGB values and the 4th argument represents alpha used for creating ***fog effects*** (combining the images).
0.0 – transparent (passes all kinds of light)
1.0 – opaque(does not pass any light)

6. **How to set the image color toMAGENTA?**
glColor3f(1.0,0.0,1.0)

7. **What is the use ofglEnable(GL_DEPTH_TEST)?**
This function enables hidden surface removal so that the hidden surface will not be seen.

8. **What is the use ofglutMainLoop()?**
- Causes the program to begin an event processingloop.
- If there are no events to process, then the program would enter the wait statewith the output on thescreen.
- It is similar to getch() in Cprogram.

9. **Which function is used to make the hidden surface to beviewed?**
glClear(GL_DEPTH_BUFFER_BIT);

10. **Why do we needglLoadIdentity()?**
- It replaces the current matrix with the identitymatrix.
- Serves to “reset” the coordinate system to unity before any matrix manipulations are performed.
- Use glLoadIdentity to clear a matrix stack rather than loading yourown.

11. **What are glOrtho() and gluOrtho2D() functions?**
These functions are used for parallelprojections.
glOrtho() --**Syntax:**
glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble nearVal, GLdouble farVal)
Parameters
left, right : Specify the coordinates for the left and right vertical clipping planes.
bottom, top : Specify the coordinates for the bottom and top horizontal clipping planes.
nearVal, farVal : Specify the distances to the nearer and farther depth clipping planes. These values are neDSATMive if the plane is to be behind the viewer.

gluOrtho2D()
gluOrtho2D sets up a two-dimensional orthographic viewing region. This is equivalent to calling glOrtho with near = -1 and far = 1 .
Syntax :
glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)
Parameters
left, right : Specify the coordinates for the left and right vertical clipping planes.
bottom, top : Specify the coordinates for the bottom and top horizontal clippingplanes.

12. **What are the functions for viewing perspectiveprojections?**
- glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,GLdouble nearVal, GLdoublefarVal)

Parameters
left, right : Specify the coordinates for the left and right vertical clippingplanes.
bottom, top : Specify the coordinates for the bottom and top horizontalclippingplanes.
nearVal, farVal : Specify the distances to the near and far depth clipping planes.
Both distances must be positive.

- gluPerspective(GLdouble fov, GLdouble aspect, GLdouble znear, GLdoublezfar)

Parameters
fovy : Specifies the field of view angle, in degrees, in the y direction.
aspect : Specifies the aspect ratio that determines the field of view in the x direction. Theaspect ratio is the ratio of x (width) to y (height).
zNear : Specifies the distance from the viewer to the near clipping plane(always positive).
zFar : Specifies the distance from the viewer to the far clipping plane (always positive).

13. What are window callbackfunctions?

Window callbacks indicate when to redisplay or reshape a window, when the visibility of the window changes, and when input is available for the window.

14. What is the use ofglFlush()?

glFlush empties all of the buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

15. In Sierpinski gasket, if the no. of divisions=2, how many triangles areformed?

Sierpinski gasket is a tetrahedran, hence there are 4 triangles.

If no. of divisions n=0, the no. of triangles=4, If n=1, the no. of triangles=4*3=12

If n=2, the no. of triangles=4*3*3=$4*3^2$ Hence if n is the no. of divisions, triangles formed = $4 * 3^n$

16. Name the rule which is followed for rotation of acube.

Right-handrule: The front faces are rotated in anti-clockwisedirection.
The back faces are rotated in clock-wise direction.

17. Differentiate between

a. GL_LINE_LOOP andGL_POLYGON
b. GL_QUADS andGL_POLYGON
c. GL_LINES andGL_LINE_LOOP

   a. **GL_LINE_LOOP** : forms a closed path but not a solid(filled)polygon
   GL_POLYGON: similar to GL_LINE_LOOP but the interior is filled according to the state of the relevant attributes.
   b. **GL_QUADS** : successive group of 4 vertices are interpreted asquadrilaterals.
   GL_POLYGON: any no. of vertices can be connected.
   c. **GL_LINES:** Successive pairs of vertices would be connected as a line.
   GL_LINE_LOOP : Connects the successive vertices using line segments to form a closedpath.

18. What is the use of GLUT_DOUBLE?

It is a bit mask to select a double buffered window. This overrides GLUT_SINGLE if it is also specified.

19. How do we swap thebuffers?

glClear(GL_DEPTH_BUFFER_BIT);glutSwapBuffers();

20. Name the type of graphic function of the following: glutMouseFunc(mouse),glutKeyboardFunc(keys)

Input functions.

21. What is the rotation matrix in 2D about

(i) Origin

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

(ii) fixed (pivot)point

$$R(\theta) = \begin{pmatrix} \cos\theta & -\sin\theta & m \\ \sin\theta & \cos\theta & n \\ 0 & 0 & 1 \end{pmatrix}$$

where

m =x-xcos(θ)+ysin(θ); n= y-xsin(θ)-ycos(θ);

22. In Cohen-Sutherland line clipping algorithm, what is the condition for trivial accept and trivial reject?

Trivial accept:

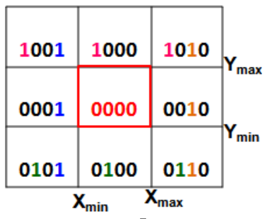
a. Both end points have to be in the region with code**0000.**

b. Trivial accept happens only if **code1 | code2 = 0000**

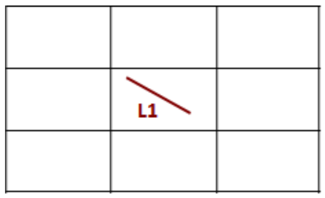
Trivial reject:

a. Codes of both end points will have **1** in the same bitposition.

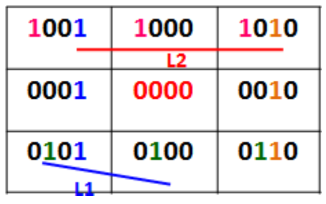
b. Trivial reject happens only if **code1 & code2 != 0000**



Outcodes



Trivial accept



Trivial reject