Visvesvaraya Technological University
"Jnana Sangama", Belagavi-590018

****

Sixth Semester B.E.

[As per Choice Based Credit System (CBCS) Scheme]

(For Internal Circulation Only)

Computer Graphics and Image Processing Laboratory (21CSL66) Lab Manual

(For Reference Only)

Name	
USN	
Section	
Lab Batch	
Day /Time	

****

Kalpataru Institute of Technology, Tiptur - 572 201

Department of Computer Science and Engineering

AY: 2023-2024

VISVESVARAYA TECHNOLOGICAL UNIVERSITY, BELAGAVI

COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY

LAB MANUAL

21CSL66

VI Semester

KALPATARU INSTITUTE OF TECHNOLOGY, TIPTUR

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

2023-24

CONTENTS

PROGRAM NO.	PROGRAM TITLE	PAGE NO.
	Introduction	1-12
1	Develop a program to draw a line using Bresenham’s line drawing technique.	13-14
2	Develop a program to demonstrate basic geometric operations on the 2D object.	15-16
3	Develop a program to demonstrate basic geometric operations on the 3D object.	17-19
4	Develop a program to demonstrate 2D transformation on basic objects.	20-23
5	Develop a program to demonstrate 3D transformation on 3D objects.	24-26
6	Develop a program to demonstrate Animation effects on simple objects.	27-31
7	Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left.	32-35
8	Write a program to show rotation, scaling, and translation on an image.	36-37
9	Read an image and extract and display low-level features such as edges, textures using filtering techniques.	38
10	Write a program to blur and smoothing an image.	39-40
11	Write a program to contour an image.	41
12	Write a program to detect a face/s in an image.	42
	Viva questions	43-46

KALPATARU INSTITUTE OF TECHNOLOGY

(Accredited by NBA, Approved by A.I.C.T.E. New Delhi, Recognized by Govt. of Karnataka & Affiliated to V.T U., Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Vision and Mission of the Institution

Vision

"To bring forth technical graduates of high caliber with a strong character and to uphold the spiritual and cultural values of our country."

Mission

"To impart quality technical and managerial education at graduate and post graduate levels through our dedicated and well qualified faculty."

Vision and Mission of the CSE Department

Vision

"To be a centre of excellence in education and research in field of computer science and engineering by empowering the students to be highly competent, technologically proficient, self motivated, innovative professionals, entrepreneurs and responsible global citizens possessing human values to meet the global challenges."

Mission

"To impart the students with strong fundamental concepts, analytical capability, problem-solving skills, thereby enhancing the employability skills required for the industries."

KALPATARU INSTITUTE OF TECHNOLOGY

(Accredited by NBA, Approved by A.I.C.T.E. New Delhi, Recognized by Govt. of Karnataka & Affiliated to V.T U., Belagavi)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Program Outcomes	
a.	**Engineering Knowledge:** Apply knowledge of mathematics, science, engineering fundamentals and an engineering specialization to the solution of complex engineering problems.
b.	**Problem Analysis: Identify,** formulate, research literature and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences and engineering sciences
c.	**Design/ Development of Solutions**: Design solutions for complex engineering problems and design system components or processes that meet specified needs with appropriate consideration for public health and safety, cultural, societal and environmental considerations.
d.	**Conduct investigations of complex problems** using research-based knowledge and research methods including design of experiments, analysis and interpretation of data and synthesis of information to provide valid conclusions.
e.	**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 under- standing of the limitations.
f.	**The Engineer and Society**: Apply reasoning informed by contextual knowledge to assess societal, health, safety, legal and cultural issues and the Consequent responsibilities relevant to professional engineering practice.
g.	**Environment and Sustainability**: Understand the impact of professional Engineering solutions in societal and environmental contexts and demonstrate knowledge of and need for sustainable development.
h.	**Ethics**: Apply ethical principles and commit to professional ethics andResponsibilities and norms of engineering practice.
i.	**Individual and Team Work:** Function effectively as an individual, and as amember or leader in diverse teams and in multi disciplinary settings.
j.	**Communication:** Communicate effectively on complex engineering activities withthe 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**.**
k.	**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.
l.	**Project Management and Finance:** Demonstrate knowledge and understandingof 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 Multi disciplinary environments.
Program Specific Outcomes	
m.	**PSO1**: Adapt, Contribute Innovate ideas in the field of Artificial Intelligence andMachine Learning
n.	**PSO2**: Enrich the abilities to qualify for Employment, Higher studies and Research in various domains of Artificial Intelligence and Machine Learning such as Data Science, Computer Vision, Natural Language Processing with ethical values
o.	**PSO3**: Acquire practical proficiency with niche technologies and open source platforms and become Entrepreneur in the domain of Artificial Intelligence and Machine Learning

List of Experiments

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 openGLCLO
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 theLaboratory 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 formers to understand the present developments inagriculture. ➢ Projects which helps high school/college students to understand the scientific

problems.
- Simulation projects which helps to understand innovations in science and technology

Course Outcome (Course Skill Set)
At the end of the course the student will be able to:

CO 1: Use OpenGL / OpenCV for the development of mini Projects.
CO 2: Analyze the necessity mathematics and design required to demonstrate basic geometrictransformation techniques.
C0 3: Demonstrate the ability to design and develop input interactive techniques.
C0 4: Apply the concepts to Develop user friendly applications using Graphics and IP concepts.

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 write- up 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 learningability. 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 thelaboratory examination (with report and presentation).
- Weightage of marks for **PART A is 60%** and for **PART B is 40%.** General rubrics suggested tobe followed for part A and part B.
- Change of experiment is allowed only once (in part A) and marks allotted to the procedurepart to be made zero.
- The duration of SEE is 03 hours.

Suggested Learning Resources:

1. Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version,3rd/4th Edition, Pearson Education,2011
2. James D Foley, Andries Van Dam, Steven K Feiner, John F Huges Computer graphics with OpenGL: Pearson education

Weblinks and Video Lectures (e-Resources):

1. https://nptel.ac.in/courses/106/106/106106090/
2. https://nptel.ac.in/courses/106/102/106102063/
3. https://nptel.ac.in/courses/106/103/106103224/
4. https://nptel.ac.in/courses/106/102/106102065/
5. https://www.tutorialspoint.com/opencv/
6. https://medium.com/analytics-vidhya/introduction-to-computer-vision-opencv-in-python-fb722e805e8b

INTRODUCTION

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer hardware and software. The development of computer graphics, or simply referred to as CG, has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized the animation and video game industry. 2D computer graphics are digital images—mostly from two-dimensional models, such as 2D geometric models, text (vector array), and 2D data. 3D computer graphics in contrast to 2D computer graphics are graphics that use a three-dimensional representation of geometric data that is stored in the computer for the purposes of performing calculations and rendering images.

OPEN GL

OpenGL is the most extensively documented 3D graphics API (Application Program Interface) to date. Information regarding OpenGL is all over the Web and in print. It is impossible to exhaustively list all sources of OpenGL information. OpenGL programs are typically written in C and C++. One can also program OpenGL from Delphi (a Pascal-like language), Basic, Fortran, Ada, and other langauges. To compile and link OpenGL programs, one will need OpenGL header files. To run OpenGL programs one may need shared or dynamically loaded OpenGL libraries, or a vendor-specific OpenGL Installable Client Driver (ICD).

GLUT

The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system. Functions performed include window definition, window control, and monitoring of keyboard and mouse input. Routines for drawing a number of geometric primitives (both in solid and wireframe mode) are also provided, including cubes, spheres, and cylinders. GLUT even has some limited support for creating pop-up menus. The two aims of GLUT are to allow the creation of rather portable code between operating systems (GLUT is cross-platform) and to make learning OpenGL easier. All GLUT functions start with the glut prefix (for example, glutPostRedisplay marks the current window as needing to be redrawn).

KEY STAGES IN THE OPENGL RENDERING PIPELINE:

- **Display Lists**

All data, whether it describes geometry or pixels, can be saved in a *display list* for current or later use. (The alternative to retaining data in a display list is processing the data immediately - also known as *immediate mode*.) When a display list is executed, the retained data is sent from the display list just as if it were sent by the application in immediate mode.

- **Evaluators**

All geometric primitives are eventually described by vertices. Parametric curves and surfaces may be initially described by control points and polynomial functions called basis functions. Evaluators provide a method to derive the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, texture coordinates, colors, and spatial coordinate values from the control points.

- **Per-Vertex Operations**

For vertex data, next is the "per-vertex operations" stage, which converts the vertices into primitives. Some vertex data (for example, spatial coordinates) are transformed by 4 x 4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen. If advanced features are enabled, this stage is even busier. If texturing is used, texture coordinates may be generated and transformed here. If lighting is enabled, the lighting calculations are performed using the transformed vertex, surface normal, light source position, material properties, and other lighting information to produce a color value.

- **Primitive Assembly**

Clipping, a major part of primitive assembly, is the elimination of portions of geometry which fall outside a half-space, defined by a plane. Point clipping simply passes or rejects vertices; line or polygon clipping can add additional vertices depending upon how the line or polygon is clipped. In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then viewport and depth (z coordinate) operations are applied. If culling is enabled and the primitive is a polygon, it then may be rejected by a culling test. Depending upon the polygon mode, a polygon may be drawn as points or lines.

The results of this stage are complete geometric primitives, which are the transformed and clipped vertices with related color, depth, and sometimes texture-coordinate values and guidelines for the rasterization step.

- **Pixel Operations**

While geometric data takes one path through the OpenGL rendering pipeline, pixel data takes a different route. Pixels from an array in system memory are first unpacked from one of a variety of formats into the proper number of components. Next the data is scaled, biased, and processed by a pixel map. The results are clamped and then either written into texture memory or sent to the rasterization step If pixel data is read from the frame buffer, pixel-transfer operations (scale, bias, mapping, and clamping) are performed. Then these results are packed into an appropriate format and returned to an array in system memory.

There are special pixel copy operations to copy data in the framebuffer to other parts of the framebuffer or to the texture memory. A single pass is made through the pixel transfer operations before the data is written to the texture memory or back to the framebuffer.

- **Texture Assembly**

  An OpenGL application may wish to apply texture images onto geometric objects to make them look more realistic. If several texture images are used, it's wise to put them into texture objects so that you can easily switch among them. Some OpenGL implementations may have special resources to accelerate texture performance. There may be specialized, high-performance texture memory. If this memory is available, the texture objects may be prioritized to control the use of this limited and valuable resource.

- **Rasterization**

  Rasterization is the conversion of both geometric and pixel data into *fragments*. Each fragment square corresponds to a pixel in the frame buffer. Line and polygon stipples, line width, point size, shading model, and coverage calculations to support anti-aliasing are taken into consideration as vertices are connected into lines or the interior pixels are calculated for a filled polygon. Color and depth values are assigned for each fragment square.

- **Fragment Operations**

  Before values are actually stored into the frame buffer, a series of operations are performed that may alter or even throw out fragments. All these operations can be enabled or disabled. The first operation which may be encountered is texturing, where a texel (texture element) is generated from texture memory for each fragment and applied to the fragment. Then fog calculations may be applied, followed by the scissor test, the alpha test, the stencil test, and the depth-buffer test (the depth buffer is for hidden-surface removal). Failing an enabled test may end the continued processing of a fragment's square. Then, blending, dithering, logical operation, and masking by a bitmask may be performed. Finally, the thoroughly processedfragment is drawn into the appropriate buffer, where it has finally advanced to be a pixel and achieved its final resting place.

- **OpenGL-Related Libraries**

  OpenGL provides a powerful but primitive set of rendering commands, and all higher-level drawing must be done in terms of these commands. Also, OpenGL programs have to use the underlying mechanisms of the windowing system. A number of libraries exist to allow you to simplify your programming tasks, including the following:

The OpenGL Utility Library (GLU) contains several routines that use lower-level OpenGL commands to perform such tasks as setting up matrices for specific viewing orientations and projections, performing polygon tessellation, and rendering surfaces. This library is provided as part of every OpenGL implementation. Portions of the GLU are described in the *OpenGL* For every window system, there is a library that extends the functionality of that window system to support OpenGL rendering. For machines that use the X Window System, the OpenGL Extension to the X Window System (GLX) is provided as an adjunct to OpenGL. GLX routines use the prefix glX. For Microsoft Windows, the WGL routines provide the Windows to OpenGL interface. All WGL routines use the prefix wgl. For IBM OS/2, the PGL is the Presentation Manager to OpenGL interface, and its routines use the prefix pgl. The OpenGL Utility Toolkit (GLUT) is a window system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window system APIs. Open Inventor is an object-oriented toolkit based on OpenGL which provides objects and methods for creating interactive three-dimensional graphics applications. Open Inventor, which is written in C++, provides prebuilt objects and a built-in event model for user interaction, high-level application components for creating and editing three-dimensional scenes, and the ability to print objects and exchange data in other graphics formats. Open Inventor is separate from OpenGL.

- **GLUT, the OpenGL Utility Toolkit**

OpenGL contains rendering commands but is designed to be independent of any window system or operating system. Consequently, it contains no commands for opening windows or reading events from the keyboard or mouse. Unfortunately, it’s impossible to write a complete graphics program without at least opening a window, and most interesting programs require a bit of user input or other services from the operating system or window system. In many cases, complete programs make the most interesting examples, so this book uses GLUT to simplify opening windows, detecting input, and so on. In addition, since OpenGL drawing commands are limited to those that generate simple geometric primitives (points, lines, and polygons), GLUT includes several routines that create more complicated three-dimensional objects such as a sphere, a torus, and a teapot. This way, snapshots of program output can be interesting to look at. (Note that the OpenGL Utility Library, GLU, also has quadrics routines that create some of the same three-dimensional objects as GLUT, such as a sphere, cylinder, or cone).

GLUT may not be satisfactory for full-featured OpenGL applications, but you may find it a useful starting point for learning OpenGL. The rest of this section briefly describes a small subset of GLUT routines so that you can follow the programming examples in the rest of this book.

OBJECTIVE AND APPLICATION OF THE LAB

The objective of this lab is to give students hands on learning exposure to understand and apply computer graphics with real world problems. The lab gives the direct experience to Visual Basic Integrated Development Environment (IDE) and GLUT toolkit. The students get a real world exposure to Windows programming API. Applications of this lab are profoundly felt in gaming industry, animation industry and Medical Image Processing Industry. The materials learned here will useful in Programming at the Software Industry.

Setting up GLUT - main()

GLUT provides high-level utilities to simplify OpenGL programming, especially in interacting with the Operating System (such as creating a window, handling key and mouse inputs). The following GLUT functions were used in the above program:

- **glutInit:** initializes GLUT, must be called before other GL/GLUT functions. It takes the same arguments as the main().

  void **glutInit**(int *argc, char **argv)
- **glutCreateWindow:** creates a window with the given title.

  int **glutCreateWindow**(char *title)
- **glutInitWindowSize:** specifies the initial window width and height, in pixels.

  void **glutInitWindowSize**(int width, int height)
- **glutInitWindowPosition:** positions the top-left corner of the initial window at (x, y). The coordinates (x, y), in term of pixels, is measured in window coordinates, i.e., origin (0, 0) is at the top-left corner of the screen; x-axis pointing right and y-axis pointing down.

  void **glutInitWindowPosition**(int x, int y)
- **glutDisplayFunc:** registers the callback function (or event handler) for handling window-paint event. The OpenGL graphic system calls back this handler when it receives a window re-paint request. In the example, we register the function display() as the handler.

  void **glutDisplayFunc**(void (*func)(void))
- **glutMainLoop:** enters the infinite event-processing loop, i.e, put the OpenGL graphics system to wait for events (such as re-paint), and trigger respective event handlers (such as display()).

  void **glutMainLoop**()

- **glutInitDisplayMode:** requests a display with the specified mode, such as color mode (GLUT_RGB, GLUT_RGBA, GLUT_INDEX), single/double buffering (GLUT_SINGLE, GLUT_DOUBLE), enable depth (GLUT_DEPTH), joined with a bit OR '|'.

  void **glutInitDisplayMode**(unsigned int displayMode)
- **void glMatrixMode (GLenum mode);**

  The **glMatrixMode** function specifies which matrix is the current matrix.
- **void glOrtho(GLdoubleleft,GLdoubleright,GLdoublebottom,GLdoubletop,GLdoublezNear,GLdoublezFar);**

  The **glOrtho** function multiplies the current matrix by an orthographic matrix.
- **void glPointSize (GLfloat size);**

  The **glPointSize** function specifies the diameter of rasterized points.
- **voidglutPostRedisplay(void);**

  glutPostRedisplay marks the current window as needing to be redisplayed.
- **void glPushMatrix (void);**

  void glPopMatrix (void);

  The **glPushMatrix** and **glPopMatrix** functions push and pop the current matrix stack.
- **GLintglRenderMode (GLenum mode);**

  The **glRenderMode** function sets the rasterization mode.
- **void glRotatef (GLfloat angle, GLfloat x, GLfloat y, GLfloat);**

  The glRotatef functions multiply the current matrix by a rotation matrix.
- **void glScalef (GLfloat x, GLfloat y, GLfloat z);**

  The **glScalef** functions multiply the current matrix by a general scaling matrix.
- **void glTranslatef (GLfloat x, GLfloat y, GLfloat z);**

  The glTranslatef functions multiply the current matrix by a translation matrix.
- **void glViewport (GLint x, GLint y, GLsizei width, GLsizei height);**

  The **glViewport** function sets the viewport.
- **void glEnable, glDisable();**

  The **glEnable** and **glDisable** functions enable or disable OpenGL capabilities.
- **glutBitmapCharacter();**

  The **glutBitmapCharacter** function used for font style.

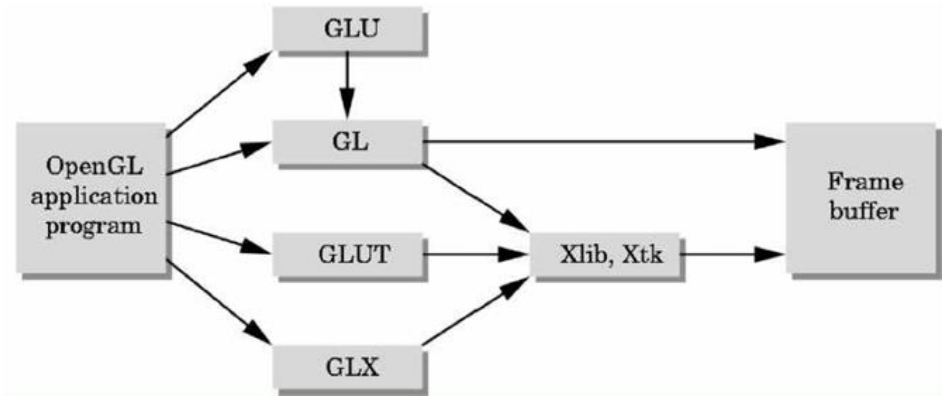
OpenGL Primitives

Value	Meaning
GL_POINTS	individual points
GL_LINES	pairs of vertices interpreted as individual line segments
GL_POLYGON	boundary of a simple, convex polygon
GL_TRIANGLES	triples of vertices interpreted as triangles
GL_QUADS	quadruples of vertices interpreted as four-sided polygons
GL_LINE_STRIP	series of connected line segments
GL_LINE_LOOP	same as above, with a segment added between last and first vertices
GL_TRIANGLE_STRIP	linked strip of triangles
GL_TRIANGLE_FAN	linked fan of triangles
GL_QUAD_STRIP	linked strip of quadrilaterals

Introduction to OpenGL

OpenGL is an API.OpenGL is nothing more than a set of functions you call from your program (think of as collection of .h files.

OpenGL Libraries:



OpenGL Hierarchy:

- Several levels of abstraction are provided
- GL
    - Lowest level: vertex, matrix manipulation
    - glVertex3f(point.x, point.y, point.z)
- GLU

  - Helper functions for shapes, transformations
  - gluPerspective(fovy, aspect, near, far)
  - gluLookAt(0, 0, 10, 0, 0, 0, 0, 1, 0);
- GLUT
  - Highest level: Window and interface management
  - glutSwapBuffers()
  - glutInitWindowSize (500, 500);

OpenGL Implementations :

- OpenGL IS an API (think of as collection of .h files):
  - #include <GL/gl.h>
  - #include <GL/glu.h>
  - #include <GL/glut.h>
- Windows, Linux, UNIX, etc. all provide a platform specific implementation.
- Windows: opengl32.lib glu32.lib glut32.lib
- Linux: -l GL -l GLU –l GLUT

Event Loop:

- OpenGL programs often run in an event loop:
  - – Start the program
  - – Run some initialization code
  - – Run an infinite loop and wait for events such as
    - • Key press
    - • Mouse move, click
    - • Reshape window
    - • Expose event

OpenGL Command Syntax (1) :

- OpenGL commands start with “gl”
- OpenGL constants start with “GL_”
- Some commands end in a number and one, two or three letters at the end (indicating number and type of arguments)
- A Number indicates number of arguments

- Characters indicate type of argument

OpenGL Command Syntax (2)

– `f' float

– `d' double float

– `s' signed short integer

– `i' signed integer

– `b' character

– `ub' unsigned character

– `us' unsigned short integer

– `ui' unsigned integer

IMAGE PROCESSING

Image processing is the process of transforming an image into a digital form and performing certain operations to get some useful information from it. The image processing system usually treats all images as 2D signals when applying certain predetermined signal processing methods.

Deep learning has had a tremendous impact on various fields of technology in the last few years. One of the hottest topics buzzing in this industry is computer vision, the ability for computers to understand images and videos on their own. Self-driving cars, biometrics and facial recognition all rely on computer vision to work. At the core of computer vision is image processing.

What Is an Image?

An image is represented by its dimensions (height and width) based on the number of pixels. For example, if the dimensions of an image are 500 x 400 (width x height), the total number of pixels in the image is 200000.

This pixel is a point on the image that takes on a specific shade, opacity or color. It is usually represented in one of the following:

- Grayscale - A pixel is an integer with a value between 0 to 255 (0 is completely black and 255 is completely white).
- RGB - A pixel is made up of 3 integers between 0 to 255 (the integers represent the intensity of red, green, and blue).
- RGBA - It is an extension of RGB with an added alpha field, which represents the opacity of the image.

Image processing requires fixed sequences of operations that are performed at each pixel of an image. The image processor performs the first sequence of operations on the image, pixel by pixel. Once this is fully done, it will begin to perform the second operation, and so on. The output value of these operations can be computed at any pixel of the image.

What Is Image Processing?

Image processing is the process of transforming an image into a digital form and performing certain operations to get some useful information from it. The image processing system usually treats all images as 2D signals when applying certain predetermined signal processing methods.

Types of Image Processing

There are five main types of image processing:

- Visualization - Find objects that are not visible in the image.
- Recognition - Distinguish or detect objects in the image.
- Sharpening and restoration - Create an enhanced image from the original image.
- Pattern recognition - Measure the various patterns around the objects in the image.
- Retrieval - Browse and search images from a large database of digital images that are similar to the original image.

Fundamental Image Processing Steps:

- **Image Acquisition**

Image acquisition is the first step in image processing. This step is also known as preprocessing in image processing. It involves retrieving the image from a source, usually a hardware-based source.

- **Image Enhancement**

Image enhancement is the process of bringing out and highlighting certain features of interest in an image that has been obscured. This can involve changing the brightness, contrast, etc.

- **Image Restoration**

Image restoration is the process of improving the appearance of an image. However, unlike image enhancement, image restoration is done using certain mathematical or probabilistic models.

- **Color Image Processing**

Color image processing includes a number of color modeling techniques in a digital domain. This step has gained prominence due to the significant use of digital images over the internet.

- **Wavelets and Multiresolution Processing**

  Wavelets are used to represent images in various degrees of resolution. The images are subdivided into wavelets or smaller regions for data compression and for pyramidal representation.

- **Compression**

  Compression is a process used to reduce the storage required to save an image or the bandwidth required to transmit it. This is done particularly when the image is for use on the Internet.

- **Morphological Processing**

  Morphological processing is a set of processing operations for morphing images based on their shapes.

- **Segmentation**

  Segmentation is one of the most difficult steps of image processing. It involves partitioning an image into its constituent parts or objects.

- **Representation and Description**

  After an image is segmented into regions in the segmentation process, each region is represented and described in a form suitable for further computer processing. Representation deals with the image’s characteristics and regional properties. Description deals with extracting quantitative information that helps differentiate one class of objects from the other.

- **Recognition**

  Recognition assigns a label to an object based on its description.

Applications of Image Processing

1. **Medical Image Retrieval**
2. **Traffic Sensing Technologies**
3. **Image Reconstruction**
4. **Face Detection**

Benefits of Image Processing

The implementation of image processing techniques has had a massive impact on many tech organizations. Here are some of the most useful benefits of image processing, regardless of the field of operation:

- The digital image can be made available in any desired format (improved image, X-Ray, photo negative, etc)
- It helps to improve images for human interpretation
- Information can be processed and extracted from images for machine interpretation
- The pixels in the image can be manipulated to any desired density and contrast
- Images can be stored and retrieved easily
- It allows for easy electronic transmission of images to third-party providers

LAB PROGRAMS

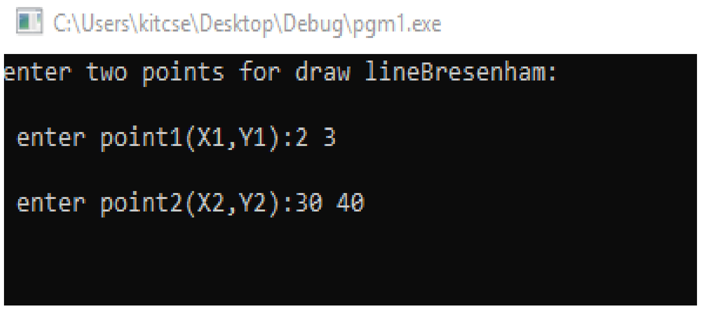
Experiment No 1

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

Program:

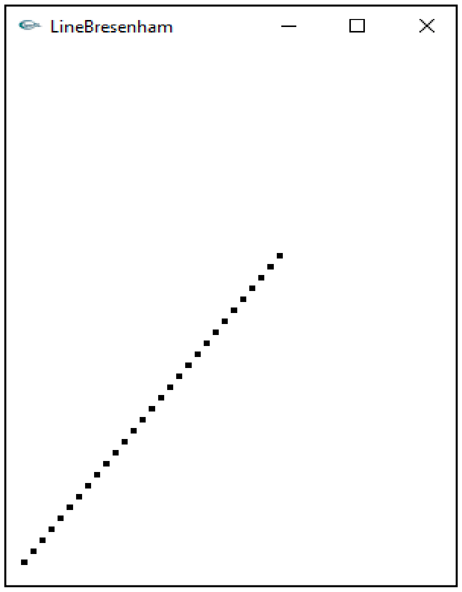
```
#include<stdio.h>
#include<math.h>
#include<gl/glut.h>
GLint X1,Y1,X2,Y2;
void LineBres(void)
{
	glClear(GL_COLOR_BUFFER_BIT);
	int dx=abs(X2-X1),dy=abs(Y2-Y1);
	int p=2*dy-dx;
	int twoDy=2*dy,twoDyDx=2*(dy-dx);
	int x,y;
	if(X1>X2)
	{
		x=X2;
		y=Y2;
		X2=X1;
	}
	else
	{
		x=X1;
		y=Y1;
		X2=X2;
	}
	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);
}
void 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();
}
```

Output:



C:\Users\kitcse\Desktop\Debug\pgm1.exe

```
enter two points for draw lineBresenham:

 enter point1(X1,Y1):2 3

 enter point2(X2,Y2):30 40
```



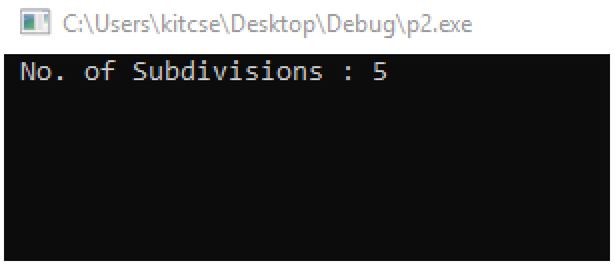
Experiment No 2

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

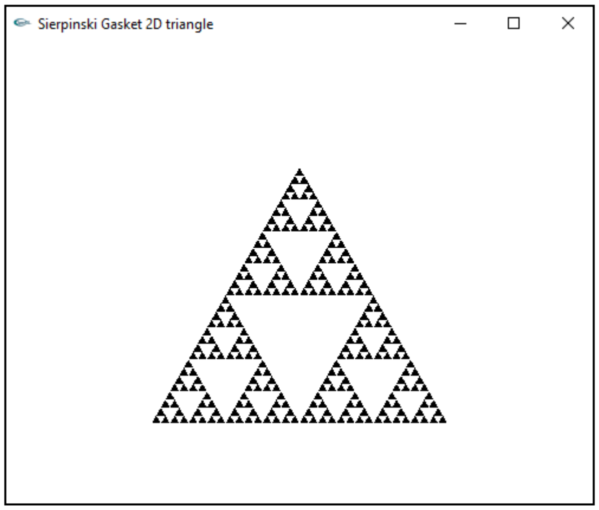
Program:

```
#include<stdio.h>
#include<GL/glut.h>
typedef float point2[2];

/* initial triangle */

point2 v[]={{-1.0, -0.58}, {1.0, -0.58}, {0.0, 1.15}};
int n;

/* display one triangle  */
void triangle( point2 a, point2 b, point2 c)
{
     glBegin(GL_TRIANGLES);
     glVertex2fv(a);
     glVertex2fv(b);
     glVertex2fv(c);
     glEnd();
}

void divide_triangle(point2 a, point2 b, point2 c, int m)
{
/* triangle subdivision using vertex numbers */

    point2 v0, v1, v2;
     int j;
     if(m>0)
    {
          for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
          for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
          for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
          divide_triangle(a, v0, v1, m-1);
          divide_triangle(c, v1, v2, m-1);
          divide_triangle(b, v2, v0, m-1);
    }
     else(triangle(a,b,c)); /* draw triangle at end of recursion */
}
void display(void)
{
     glClear(GL_COLOR_BUFFER_BIT);
     divide_triangle(v[0], v[1], v[2], n);
     glFlush();
}

void myinit()
{
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     gluOrtho2D(-2.0, 2.0, -2.0, 2.0);
     glMatrixMode(GL_MODELVIEW);
     glClearColor (1.0, 1.0, 1.0, 1.0);
     glColor3f(0.0,0.0,0.0);
}

void main(int argc, char **argv)
{
     printf(" No. of Subdivisions : ");
     scanf("%d",&n);
     glutInit(&argc, argv);
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB );
     glutInitWindowSize(500, 500);
     glutCreateWindow("Sierpinski Gasket 2D triangle");
     glutDisplayFunc(display);
     myinit();
     glutMainLoop();
}
```

Output:



```
No. of Subdivisions : 5
```



Experiment No 3

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

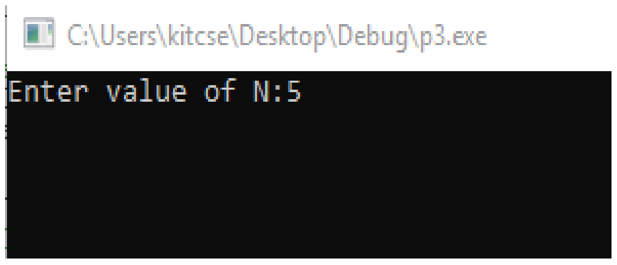
Program:

```
#include<stdio.h>
#include<stdlib.h>
#include<GL/glut.h>
typedef float point[3];

/* initial tetrahedron */
point v[]={{0.0, 0.0, 1.0}, {0.0, 0.942809, -0.33333},{-0.816497,
-0.471405, -0.333333}, {0.816497, -0.471405, -0.333333}};
static GLfloat theta[] = {0.0,0.0,0.0};
int n;

/* display one triangle using a line loop for wire frame, a single
normal for constant shading, or three normals for interpolative
shading */
void triangle( point a, point b, point c)
{
     glBegin(GL_POLYGON);
     glNormal3fv(a);
     glVertex3fv(a);
     glVertex3fv(b);
     glVertex3fv(c);
     glEnd();
}

/* triangle subdivision using vertex numbers right hand rule applied
to create outward pointing faces */
void divide_triangle(point a, point b, point c, int m)
{
     point v1, v2, v3;
     int j;
     if(m>0)
     {
          for(j=0; j<3; j++) v1[j]=(a[j]+b[j])/2;
          for(j=0; j<3; j++) v2[j]=(a[j]+c[j])/2;
          for(j=0; j<3; j++) v3[j]=(b[j]+c[j])/2;
          divide_triangle(a, v1, v2, m-1);
          divide_triangle(c, v2, v3, m-1);
          divide_triangle(b, v3, v1, m-1);
     }
     else(triangle(a,b,c)); /* draw triangle at end of recursion */
}
```

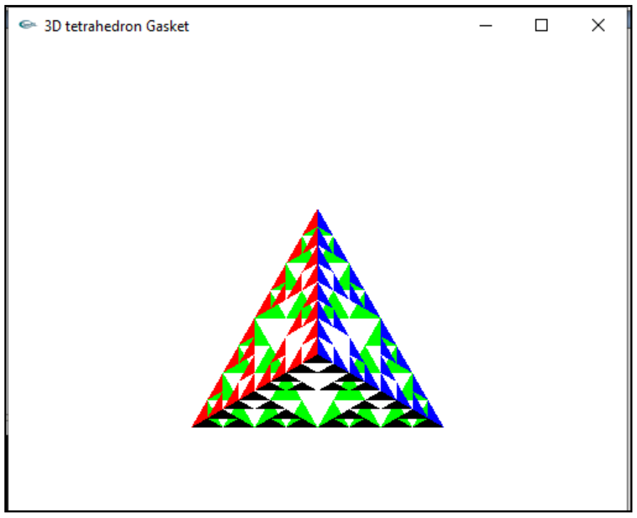
```
/* Apply triangle subdivision to faces of tetrahedron */
void tetrahedron( int m)
{
     glColor3f(1.0,0.0,0.0);
     divide_triangle(v[0], v[1], v[2], m);
     glColor3f(0.0,1.0,0.0);
     divide_triangle(v[3], v[2], v[1], m);
     glColor3f(0.0,0.0,1.0);
     divide_triangle(v[0], v[3], v[1], m);
     glColor3f(0.0,0.0,0.0);
     divide_triangle(v[0], v[2], v[3], m);
}

void display()
{
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     glLoadIdentity();
     tetrahedron(3);
     glFlush();
}

void myReshape(int w, int h)
{
     glViewport(0, 0, w, h);
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     if (w <= h)
          glOrtho(-2.0,2.0,-2.0*(GLfloat)h/(GLfloat)w,2.0 *
          (GLfloat)h/(GLfloat)w,-10.0,10.0);
     else
          glOrtho(-2.0*(GLfloat)w/(GLfloat)h,2.0*(GLfloat)w/
          (GLfloat)h,-2.0, 2.0,-10.0,10.0);
     glMatrixMode(GL_MODELVIEW);
     glutPostRedisplay();
}

void main(int argc, char **argv)
{
     int i = 0;
     printf("Enter value of N:");
     scanf("%d", &i);
     n = i;
     glutInit(&argc, argv);
     glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
     glutInitWindowSize(500, 500);
     glutCreateWindow("3D tetrahedron Gasket");
     glutReshapeFunc(myReshape);
```

```
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    glClearColor (1.0, 1.0, 1.0, 1.0);
    glutMainLoop();
}
```

Output:



```
C:\Users\kitcse\Desktop\Debug\p3.exe
Enter value of N:5
```



Experiment No 4

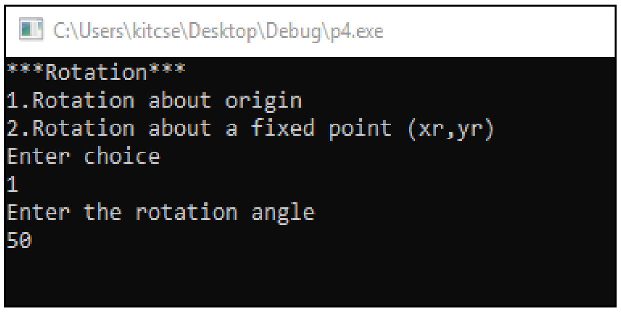
Develop a program to demonstrate 2D transformation on basic objects.

Program:

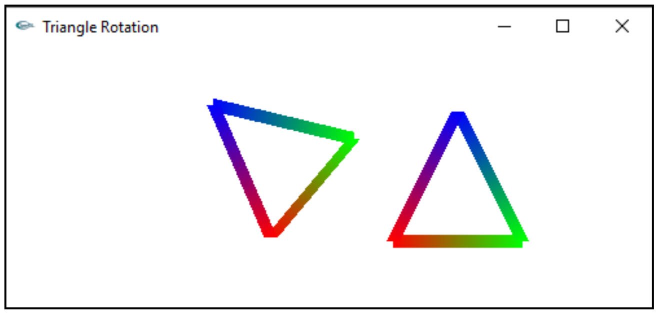
```
#include<stdio.h>
#include<math.h>
#include<GL/glut.h>

GLfloat t[3][3]={{10.0,30.0,20.0},{20.0,20.0,40.0},{1.0,1.0,1.0}};
GLfloat rotatemat[3][3]={{0},{0},{0}};
GLfloat result[3][9]={{0},{0},{0}};

GLfloat xr=10.0;
GLfloat yr=20.0;

GLfloat theta;
GLint ch;

void multiply(){
     int i,j,k;
     for(i=0;i<3;i++)
          for(j=0;j<9;j++)
          {
               result[i][j]=0;
               for(k=0;k<3;k++)
               result[i][j]=result[i][j]+rotatemat[i][k]*t[k][j];
          }
}

void rotate_about_origin(){
     rotatemat[0][0]=cos(theta);
     rotatemat[0][1]=-sin(theta);
     rotatemat[0][2]=0;
     rotatemat[1][0]=sin(theta);
     rotatemat[1][1]=cos(theta);
     rotatemat[1][2]=0;
     rotatemat[2][0]=0;
     rotatemat[2][1]=0;
     rotatemat[2][2]=1;
     multiply();
}

void rotate_about_fixed_point(){
     GLfloat m,n;
     m=xr*(1-cos(theta))+yr*sin(theta);
```

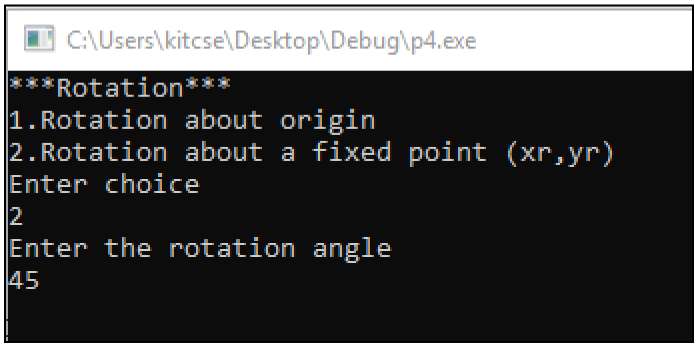
```
    n=yr*(1-cos(theta))-xr*sin(theta);
    rotatemat[0][0]=cos(theta);
    rotatemat[0][1]=-sin(theta);
    rotatemat[0][2]=m;
    rotatemat[1][0]=sin(theta);
    rotatemat[1][1]=cos(theta);
    rotatemat[1][2]=n;
    rotatemat[2][0]=0;
    rotatemat[2][1]=0;
    rotatemat[2][2]=1;
    multiply();
}

void draw_triangle(){
    glLineWidth(10);
    glBegin(GL_LINE_LOOP);
    glColor3f(1.0,0.0,0.0);
    glVertex2f(t[0][0],t[1][0]);
    glColor3f(0.0,1.0,0.0);
    glVertex2f(t[0][1],t[1][1]);
    glColor3f(0.0,0.0,1.0);
    glVertex2f(t[0][2],t[1][2]);
    glEnd();
    glFlush();
}

void draw_rotated_triangle(){
    glLineWidth(10);
    glBegin(GL_LINE_LOOP);
    glColor3f(1.0,0.0,0.0);
    glVertex2f(result[0][0],result[1][0]);
    glColor3f(0.0,1.0,0.0);
    glVertex2f(result[0][1],result[1][1]);
    glColor3f(0.0,0.0,1.0);
    glVertex2f(result[0][2],result[1][2]);
    glEnd();
    glFlush();
}

void display(){
    glClear(GL_COLOR_BUFFER_BIT);
    if(ch==1){
        draw_triangle();
        rotate_about_origin();
        draw_rotated_triangle();
        glFlush();
    }
    if(ch==2){
        draw_triangle();
        rotate_about_fixed_point();
        draw_rotated_triangle();
        glFlush();
    }
}

void myinit(){
    glClearColor(1.0,1.0,1.0,1.0);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(-50.0,50.0,-50.0,50.0);
}

int main(int argc, char** argv){
    printf("***Rotation***\n1.Rotation about origin\n2.Rotation
    about a fixed point (xr,yr)\n");
    printf("Enter choice\n");
    scanf("%d",&ch);
    printf("Enter the rotation angle\n");
    scanf("%f",&theta);
    theta=theta*(3.14/180);
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
    glutInitWindowSize(500,500);
    glutInitWindowPosition(0,0);
    glutCreateWindow("Triangle Rotation\n");
    glutDisplayFunc(display);
    myinit();
    glutMainLoop();
    return 0;
}
```

Output:



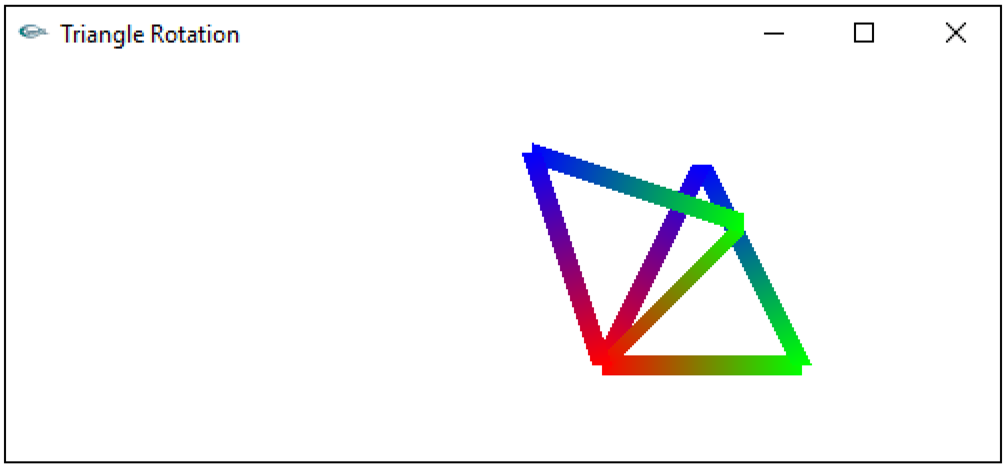
```
***Rotation***
1.Rotation about origin
2.Rotation about a fixed point (xr,yr)
Enter choice
1
Enter the rotation angle
50
```





C:\Users\kitcse\Desktop\Debug\p4.exe

```
***Rotation***
1.Rotation about origin
2.Rotation about a fixed point (xr,yr)
Enter choice
2
Enter the rotation angle
45
```



Experiment No 5

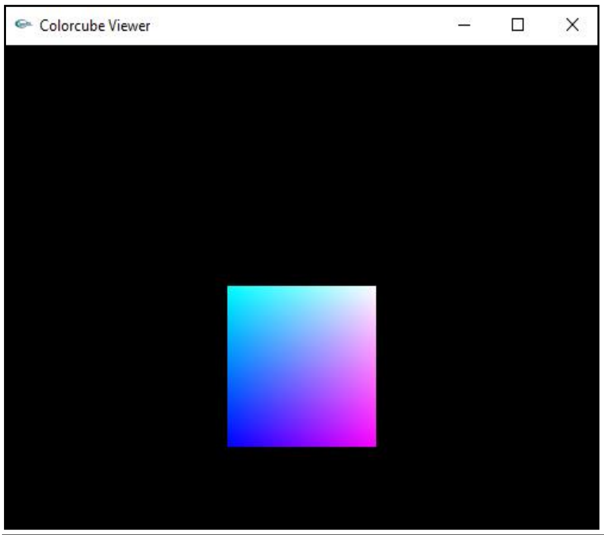
Develop a program to demonstrate 3D transformation on 3D objects

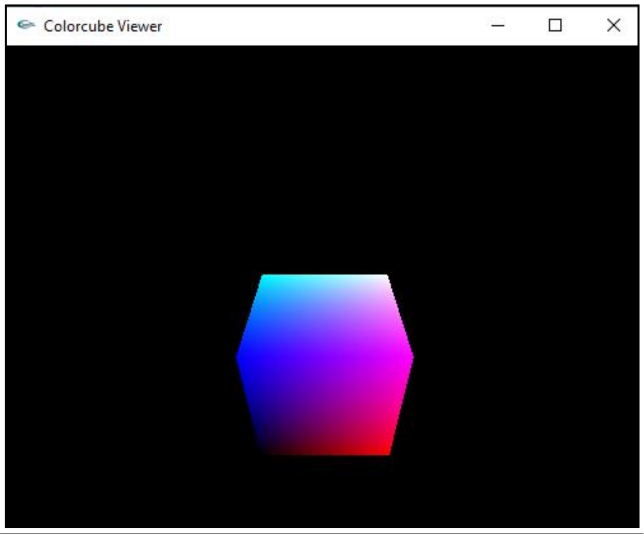
Program:

```
#include<stdlib.h>
#include<GL/glut.h>
GLfloat vertices[][3] = {{-1.0,-1.0,-1.0},{1.0,-1.0,-1.0},
                         {1.0, 1.0,-1.0}, {-1.0, 1.0,-1.0},
                         {-1.0,-1.0, 1.0}, {1.0,-1.0, 1.0},
                         {1.0,1.0, 1.0}, {-1.0,1.0,1.0}};

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

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

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

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

```
void display(void)
{
     glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
     /* Update viewer position in modelview matrix */
     glLoadIdentity();
     gluLookAt(viewer[0],viewer[1],viewer[2], 0.0, 0.0, 0.0, 0.0,
     1.0, 0.0);
     /* rotate cube */
     glRotatef(theta[0], 1.0, 0.0, 0.0);
     glRotatef(theta[1], 0.0, 1.0, 0.0);
     glRotatef(theta[2], 0.0, 0.0, 1.0);
     colorcube();      /* draw the rotated color cube */
     glFlush();
     glutSwapBuffers();
}

void mouse(int btn, int state, int x, int y)
{
     if(btn==GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 0;
     if(btn==GLUT_MIDDLE_BUTTON && state == GLUT_DOWN) axis =1;
     if(btn==GLUT_RIGHT_BUTTON && state == GLUT_DOWN) axis = 2;
     theta[axis] += 4.0;
     if( theta[axis] > 360.0 ) theta[axis] -= 360.0;
     display();
}

void keys(unsigned char key, int x, int y)
{
     /* Use x, X, y, Y, z, and Z keys to move viewer */
     if(key == 'x') viewer[0]-= 1.0;
     if(key == 'X') viewer[0]+= 1.0;
     if(key == 'y') viewer[1]-= 1.0;
     if(key == 'Y') viewer[1]+= 1.0;
     if(key == 'z') viewer[2]-= 1.0;
     if(key == 'Z') viewer[2]+= 1.0;
     display();
}

void myReshape(int w, int h)
{
     glViewport(0, 0, w, h);
     /* Use a perspective view */
     glMatrixMode(GL_PROJECTION);
     glLoadIdentity();
     if(w<=h)
          glFrustum(-2.0, 2.0, -2.0 * (GLfloat) h/ (GLfloat) w,2.0*
          (GLfloat) h / (GLfloat) w, 2.0, 20.0);
     else
          glFrustum(-2.0, 2.0, -2.0 * (GLfloat) w/ (GLfloat) h,2.0*
          (GLfloat) w / (GLfloat) h, 2.0, 20.0);
/* Or we can use gluPerspective that is gluPerspective (45.0, w/h,
-10.0, 10.0); */
     glMatrixMode(GL_MODELVIEW);
}

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

Output:





Experiment No 6

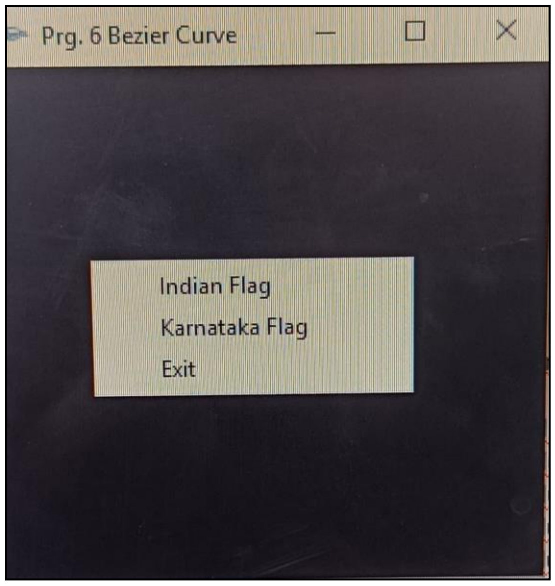
Develop a program to demonstrate Animation effects on simple objects.

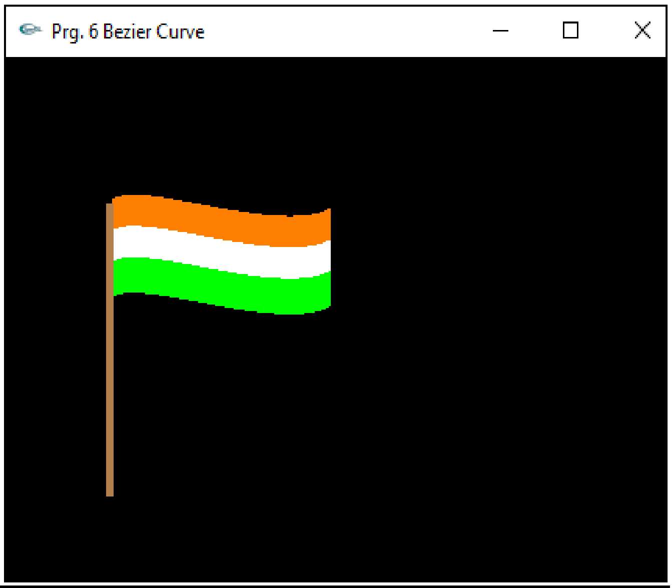
Program:

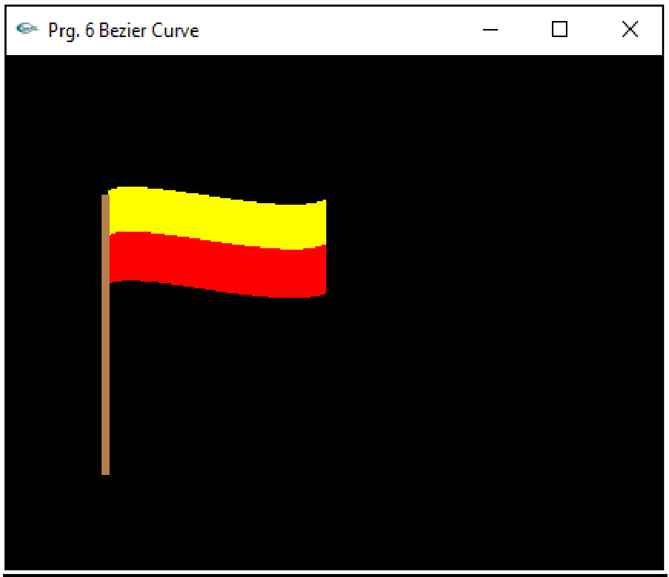
```
#include <stdio.h>
#include <math.h>
#include <GL/glut.h>

#define PI 3.1416

static int win, val = 0, CMenu;

void CreateMenu(void);
void Menu(int value);

struct wcPt3D {
    GLfloat x, y, z;
};

GLsizei winWidth = 600, winHeight = 600;
GLfloat xwcMin = 0.0, xwcMax = 130.0;
GLfloat ywcMin = 0.0, ywcMax = 130.0;

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;
    }
}

void computeBezPt(GLfloat u, struct wcPt3D *bezPt, GLint nCtrlPts,
struct 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;
    }
```

```
}

void bezier(struct wcPt3D *ctrlPts, GLint nCtrlPts, GLint
nBezCurvePts) {
    struct wcPt3D bezCurvePt;
    GLfloat u;
    GLint *C, k;
    C = new GLint[nCtrlPts];
    bino(nCtrlPts - 1, C);

    glBegin(GL_LINE_STRIP);
    for (k = 0; k <= nBezCurvePts; k++) {
        u = GLfloat(k) / GLfloat(nBezCurvePts);
        computeBezPt(u, &bezCurvePt, nCtrlPts, ctrlPts, C);
        glVertex2f(bezCurvePt.x, bezCurvePt.y);
    }
    glEnd();
    delete[] C;
}

void displayFcn() {
    GLint nCtrlPts = 4, nBezCurvePts = 20;

    static float theta = 0;
    struct wcPt3D ctrlPts[4] = { {20, 100, 0}, {30, 110, 0}, {50,
90, 0}, {60, 100, 0} };
    ctrlPts[1].x += 10 * sin(theta * PI / 180.0);
    ctrlPts[1].y += 5 * sin(theta * PI / 180.0);
    ctrlPts[2].x -= 10 * sin((theta + 30) * PI / 180.0);
    ctrlPts[2].y -= 10 * sin((theta + 30) * PI / 180.0);
    ctrlPts[3].x -= 4 * sin((theta) * PI / 180.0);
    ctrlPts[3].y += sin((theta - 30) * PI / 180.0);
    theta += 0.1;
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0, 1.0, 1.0);

    glPointSize(5);

    // Indian Flag
    if (val == 1) {
        glPushMatrix();
        glLineWidth(5);
        glColor3f(1.0, 0.5, 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 color code
        for (int j = 0; j < 8; j++) {
            glTranslatef(0, -0.8, 0);
            bezier(ctrlPts, nCtrlPts, nBezCurvePts);
        }
        glColor3f(0, 1.0, 0); // Indian flag: green color code
        for (int k = 0; k < 8; k++) {
            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();
    }

    // Karnataka Flag
    if (val == 2) {
        glPushMatrix();
        glLineWidth(5);
        glColor3f(1.0, 1.0, 0.0); // Karnataka flag: Yellow color
code
        for (int i = 0; i < 12; i++) {
            glTranslatef(0, -0.8, 0);
            bezier(ctrlPts, nCtrlPts, nBezCurvePts);
        }
        glColor3f(1, 0.0, 0.0); // Karnataka flag: Red color code
        for (int j = 0; j < 12; j++) {
            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();
}

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

void CreateMenu(void) {
    CMenu = glutCreateMenu(Menu); // Create Menu Option

    glutAddMenuEntry("Indian Flag", 1);
    glutAddMenuEntry("Karnataka Flag", 2);
    glutAddMenuEntry("Exit", 0);

    glutAttachMenu(GLUT_RIGHT_BUTTON);
}

void Menu(int value) {
    if (value == 0) {
        glutDestroyWindow(win);
        exit(0);
    } else {
        val = value;
        glutPostRedisplay(); // Redraw the window
    }
}

int main(int argc, char **argv)
{
     glutInit(&argc, argv);
     glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
     glutInitWindowPosition(50, 50);
     glutInitWindowSize(winWidth, winHeight);
     glutCreateWindow("Prg. 6 Bezier Curve");
     CreateMenu();
     glutDisplayFunc(displayFcn);
     glutReshapeFunc(winReshapeFun);
     glutMainLoop();
     return 0;
}
```

Output:


Prg. 6 Bezier Curve
Indian Flag
Karnataka Flag
Exit


Prg. 6 Bezier Curve


Prg. 6 Bezier Curve

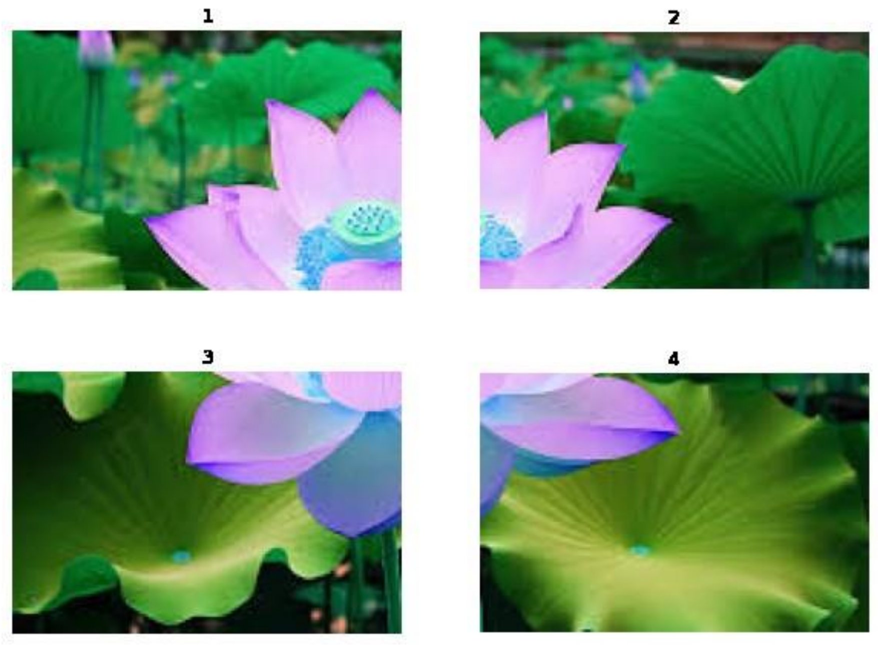
Experiment No 7

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

Program:

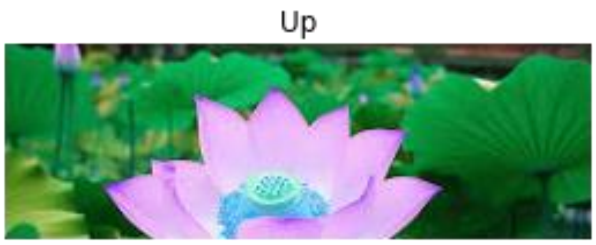
```
import cv2
import numpy as np
import matplotlib.pyplot as plt

image_path="Give your image path here”
#Example:
#image_path="E:/KIT_college/2023-24-EvenSem/CG Lab/lotus.jpg"

# Read the image

img = cv2.imread(image_path)

# Get the height and width of the image
height, width = img.shape[:2]

# Split the image into four quadrants
quad1 = img[:height//2, :width//2]
quad2 = img[:height//2, width//2:]
quad3 = img[height//2:, :width//2]
quad4 = img[height//2:, width//2:]
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(quad1)
plt.title("1")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(quad2)
plt.title("2")
plt.axis("off")

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(quad3)
plt.title("3")
plt.axis("off")
plt.subplot(1, 2, 2)
plt.imshow(quad4)
plt.title("4")
plt.axis("off")
plt.show()
```

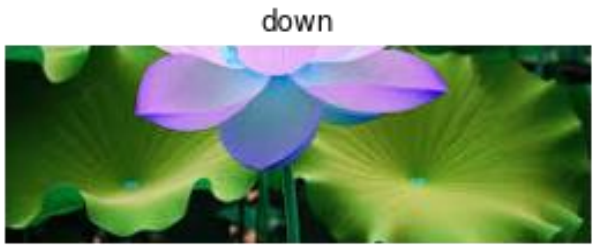
Output:



```
# Up- down
import cv2
import numpy as np

image_path="E:/KIT_college/2023-24-Even Sem/CG Lab/lotus.jpg"
# Read the image
img = cv2.imread(image_path)
# Get the height and width of the image
height, width = img.shape[:2]

up = img[:height//2,:]
down = img[height//2:,:]
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(up)
plt.title("Up")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(down)
plt.title("down")
plt.axis("off")
plt.show()
```

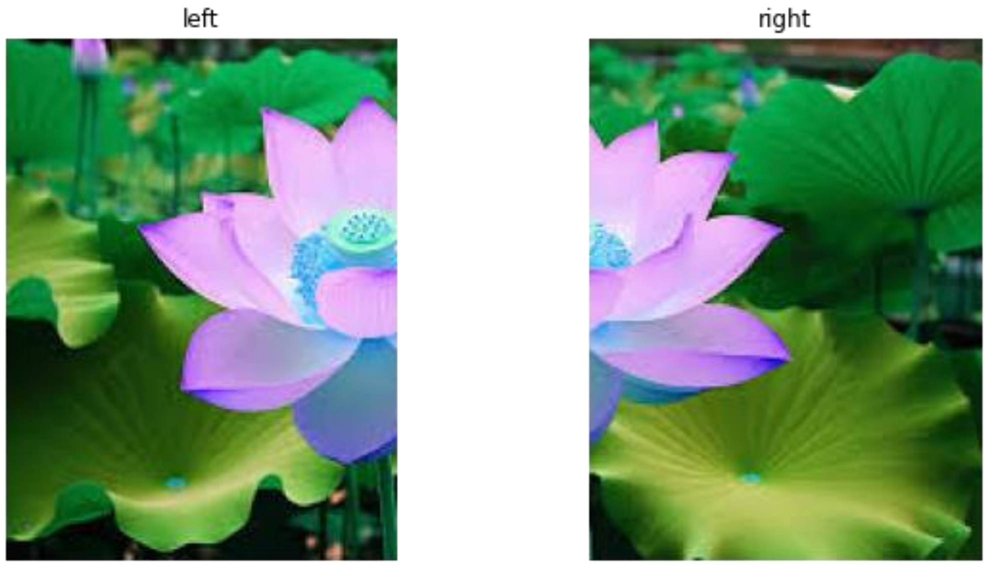
Output:





```
# left- right
import cv2
import numpy as np

image_path="E:/KIT_college/2023-24-Even Sem/CG Lab/lotus.jpg"
# Read the image
img = cv2.imread(image_path)

# Get the height and width of the image
height, width = img.shape[:2]

left = img[:, :width//2]

right = img[:, width//2:]

up = img[:height//2,:]
down = img[height//2:,:]

quad1 = img[:height//2, :width//2]
quad2 = img[:height//2, width//2:]
quad3 = img[height//2:, :width//2]
quad4 = img[height//2:, width//2:]

plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(left)
plt.title("left")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(right)
plt.title("right")
plt.axis("off")
plt.show()
```

Output:



Experiment No 8

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

Program:

```
#Rotation and scaling of image
import cv2
import numpy as np

def translate_image(image, dx, dy):
    rows, cols = image.shape[:2]
    translation_matrix = np.float32([[1, 0, dx], [0, 1, dy]])
    translated_image = cv2.warpAffine(image, translation_matrix,
     (cols, rows))
    return translated_image

# Read the image
image = cv2.imread('E:/KIT_college/2023-24-Even Sem/CGLab/
lotus.jpg')

# Get image dimensions
height, width = image.shape[:2]
# Calculate the center coordinates of the image
center = (width // 2, height // 2)
rotation_value = int(input("Enter the degree of Rotation:"))
scaling_value = int(input("Enter the zooming factor:"))
# Create the 2D rotation matrix
rotated = cv2.getRotationMatrix2D(center=center,
angle=rotation_value, scale=1)
rotated_image = cv2.warpAffine(src=image, M=rotated, dsize=(width,
height))
scaled = cv2.getRotationMatrix2D(center=center, angle=0,
scale=scaling_value)
scaled_image = cv2.warpAffine(src=rotated_image, M=scaled,
dsize=(width, height))
h = int(input("How many pixels you want the image to be translated
horizontally? "))
v = int(input("How many pixels you want the image to be translated
vertically? "))
translated_image = translate_image(scaled_image, dx=h, dy=v)
cv2.imwrite('Final_image.png', translated_image)
```

Output:

```
Image with file named "Final_image" will be generated in present working directory.
```

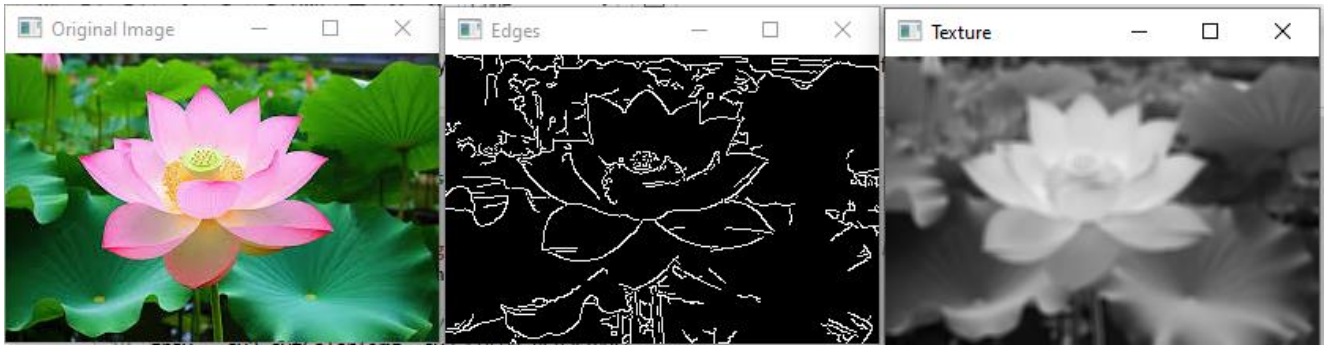
Experiment No 9

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

Program:

```
import cv2
import numpy as np
import matplotlib.pyplot as plt

# Load the image
image_path = "E:/KIT_college/2023-24-Even Sem/CG Lab/lotus.jpg"  #
Replace with the path to your image
img = cv2.imread(image_path)

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

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

# 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:



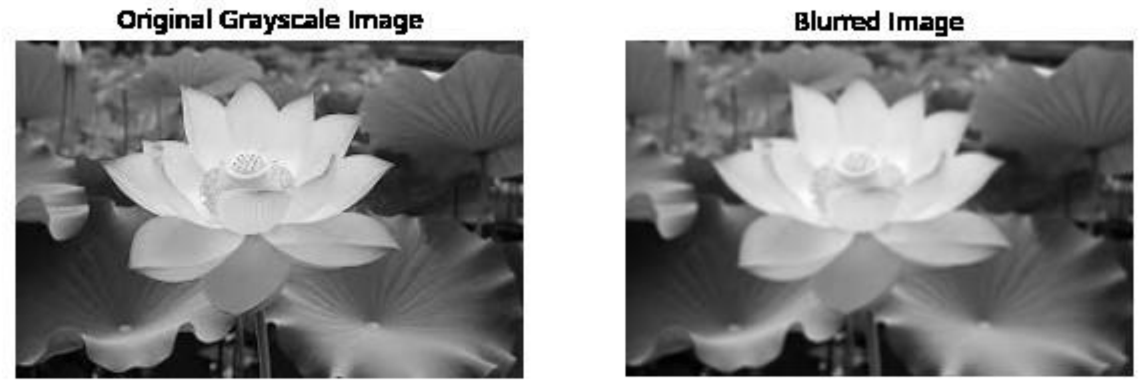
Experiment No 10

Write a program to blur and smoothing an image.

Program:

```
import numpy as np
import cv2
import matplotlib.pyplot as plt

img = cv2.imread("E:/KIT_college/2023-24-Even Sem/CG
Lab/lotus.jpg",cv2.IMREAD_GRAYSCALE)
image_array = np.array(img)
print(image_array)
def sharpen():
  return np.array([[1,1,1],[1,1,1],[1,1,1]])
def filtering(image, kernel):
    m, n = kernel.shape
    if (m == n):
        y, x = image.shape
        y = y - m + 1 # shape of image - shape of kernel + 1
        x = x - m + 1
        new_image = np.zeros((y,x))
        for i in range(y):
            for j in range(x):
                new_image[i][j] = np.sum(image[i:i+m, j:j+m]*kernel)
    return new_image
# Display the original and sharpened images
plt.figure(figsize=(10, 5))
plt.subplot(1, 2, 1)
plt.imshow(image_array,cmap='gray')
plt.title("Original Grayscale Image")
plt.axis("off")

plt.subplot(1, 2, 2)
plt.imshow(filtering(image_array, sharpen()),cmap='gray')
plt.title("Blurred Image")
plt.axis("off")
plt.show()
```

<u>Output:</u>



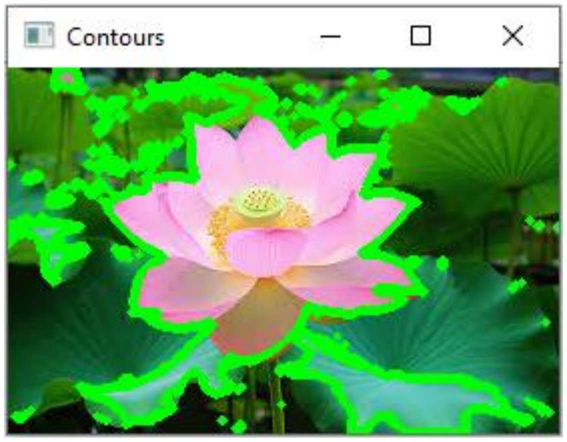
Experiment No 11

Write a program to contour an image.

Program:

```
import cv2
import numpy as np

image_path = 'E:/KIT_college/2023-24-Even Sem/CG Lab/lotus.jpg'
image = cv2.imread(image_path)

# Convert the image to grayscale (contours work best on binary
images)
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Apply thresholding (you can use other techniques like Sobel edges)
_, binary_image = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

# Find contours
contours, _ = cv2.findContours(binary_image, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

# Draw all contours on the original image
cv2.drawContours(image, contours, -1, (0, 255, 0), 3)

# Display the result
cv2.imshow('Contours', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:



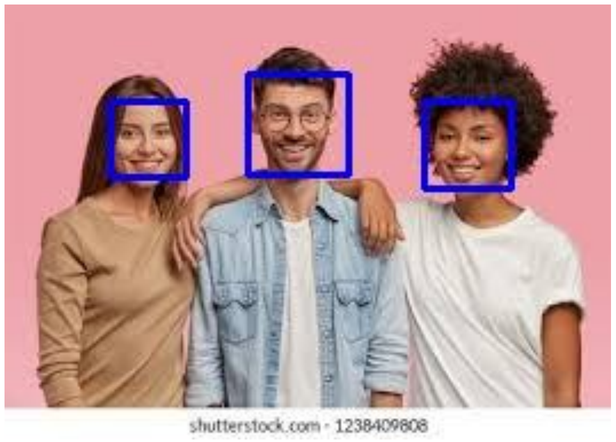
Experiment No 12

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

Program:

```
import cv2
# Load the pre-trained Haar Cascade classifier for face detection
face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_frontalface_default.xml')
eye_cascade = cv2.CascadeClassifier(cv2.data.haarcascades +
'haarcascade_eye.xml')

# Read the input image (replace 'your_image.jpg' with the actual
image path)
image_path = 'E:/KIT_college/2023-24-Even Sem/CG Lab/33.jpg'
#image_path = 'F:/2nd year sohan/1.jpg'
image = cv2.imread(image_path)
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
# Detect faces in the image
faces = face_cascade.detectMultiScale(gray, scaleFactor=1.3,
minNeighbors=5)
# Draw rectangles around detected faces
for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)
# Save or display the result
cv2.imwrite('detected_faces.jpg', image)  # Save the result
cv2.imshow('Detected Faces', image)  # Display the result
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Output:



VIVA QUESTIONS AND ANSWERS

1. What is Computer Graphics?

Computer graphics are graphics created using computers and, more generally, the representation and manipulation of image data by a computer.

2. What is OpenGL?

OpenGL is the most extensively documented 3D graphics API (Application Program Interface) to date. It is used to create Graphics.

3. What is GLUT?

The OpenGL Utility Toolkit (GLUT) is a library of utilities for OpenGL programs, which primarily perform system-level I/O with the host operating system.

4. What are the applications of Computer Graphics?

Gaming Industry, Animation Industry and Medical Image Processing Industries. The sum total of these industries is a Multi Billion Dollar Market. Jobs will continue to increase in this arena in the future.

5. Explain in brief 3D Sierpinski gasket?

The Sierpinski triangle (also with the original orthography Sierpinski), also called the Sierpinski gasket or the Sierpinski Sieve, is a fractal named after the Polish mathemati-cian Waclaw Sierpinski who described it in 1915. Originally constructed as a curve, this is one of the basic examples of self-similar sets, i.e. it is a mathematically generated pattern that can be reproducible at any magnification or reduction.

6. What is Liang-Barsky line clipping algorithm?

In computer graphics, the Liang-Barsky algorithm is a line clipping algorithm. The Liang-Barsky algorithm uses the parametric equation of a line and inequalities describing the range of the clipping box to determine the intersections between the line and the clipping box. With these intersections it knows which portion of the line should be drawn.

7. Explain in brief Cohen-Sutherland line-clipping algorithm?

The Cohen-Sutherland line clipping algorithm quickly detects and dispenses with two common and trivial cases. To clip a line, we need to consider only its endpoints. If both endpoints of a line lie inside the window, the entire line lies inside the window. It is trivially accepted and needs no clipping.

On the other hand, if both endpoints of a line lie entirely to one side of the window, the line must lie entirely outside of the window. It is trivially rejected and needs to be neither clipped nor displayed.

8. Explain in brief scan-line area filling algorithm?

The scanline fll algorithm is an ingenious way of filling in irregular polygons. The algorithm begins with a set of points. Each point is connected to the next, and the line between them is considered to be an edge of the polygon. The points of each edge are adjusted to ensure that the point with the smaller y value appears frst. Next, a data structure is created that contains a list of edges that begin on each scanline of the image. The program progresses from the first scanline upward. For each line, any pixels that contain an intersection between this scanline and an edge of the polygon are filled in. Then, the algorithm progresses along the scanline, turning on when it reaches a polygon pixel and turning of when it reaches another one, all the way across the scanline.

9. Explain Midpoint Line algorithm.

The Midpoint line algorithm is an algorithm which determines which points in an n-dimensional raster should be plotted in order to form a close approximation to a straight line between two given points. It is commonly used to draw lines 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.

10. What is a Pixel?

In digital imaging, a pixel (or picture element) is a single point in a raster image. The Pixel is the smallest addressable screen element; it is the smallest unit of picture which can be controlled. Each Pixel has its address. The address of Pixels corresponds to its coordinate. Pixels are normally arranged in a 2-dimensional grid, and are often represented using dots or squares.

11. What is Graphical User Interface?

A graphical user interface (GUI) is a type of user interface item that allows people to interact with programs in more ways than typing such as computers; hand-held devices such as MP3 Players, Portable Media Players or Gaming devices; household appliances and office equipment with images rather than text commands.

12. What is the general form of an OpenGL program?

There are no hard and fast rules. The following pseudocode is generally recognized as good OpenGL form. program entrypoint { // Determine which depth or pixel format should be used. // Create a window with the desired format. // Create a rendering context and make it current with the window. // Set up initial OpenGL state. // Set up callback routines for window resize and window refresh. } handle resize { glViewport(...); glMatrixMode(GL PROJECTION); glLoadIdentity(); // Set projection transform with glOrtho, glFrustum, gluOrtho2D, gluPerspective, etc. } handle refresh { glClear(...); glMatrixMode(GL MODELVIEW); glLoadIdentity(); // Set view transform with gluLookAt or equivalent // For each object (i) in the scene that needs to be rendered: // Push relevant stacks, e.g., glPushMatrix, glPushAttrib. // Set OpenGL state specifc to object (i). // Set model transform for object (i) using glTranslatef, glScalef, glRotatef, and/or equivalent. // Issue rendering commands for object (i). // Pop relevant stacks, (e.g., glPopMatrix, glPopAttrib.) // End for loop. // Swap bufers. }

13. What support for OpenGL does Open, Net, FreeBSD or Linux provide?

The X Windows implementation, XFree86 4.0, includes support for OpenGL using Mesa or the OpenGL Sample Implementation. XFree86 is released under the XFree86 license. http://www.xfree86.org/ .

14. What is the AUX library?

The AUX library was developed by SGI early in OpenGL's life to ease creation of small OpenGL demonstration programs. It's currently neither supported nor maintained. Developing OpenGL programs using AUX is strongly discouraged. Use the GLUT instead. It's more flexible and powerful and is available on a wide range of platforms. Very important: Don't use AUX. Use GLUT instead.

15. How does the camera work in OpenGL?

As far as OpenGL is concerned, there is no camera. More specifcally, the camera is always located at the eye space coordinate (0., 0., 0.). To give the appearance of moving the camera, your OpenGL application must move the scene with the inverse of the camera transformation.

16. How do I implement a zoom operation? Answer: A simple method for zooming is to use a uniform scale on the ModelView matrix. However, this often results in clipping by the zNear and zFar clipping planes if the model is scaled too large. A better method is to restrict the width and height of the view volume in the Projection matrix.

17. What is Microsoft Visual Studio?

Microsoft Visual Studio is an integrated development environment (IDE) for developing windows applications. It is the most popular IDE for developing windows applications or windows based software.

18. What does the .gl or .GL file format have to do with OpenGL?

.gl files have nothing to do with OpenGL, but are sometimes confused with it. .gl is a file format for images, which has no relationship to OpenGL.

19. Who needs to license OpenGL? Who doesn't? Is OpenGL free software?

Companies which will be creating or selling binaries of the OpenGL library will need to license OpenGL. Typical examples of licensees include hardware vendors, such as Digital Equipment, and IBM who would distribute OpenGL with the system software on their workstations or PCs. Also, some software vendors, such as Portable Graphics and Template Graphics, have a business in creating and distributing versions of OpenGL, and they need to license OpenGL. Applications developers do NOT need to license OpenGL. If a developer wants to use OpenGL that developer needs to obtain copies of a linkable OpenGL library for a particular machine. Those OpenGL libraries may be bundled in with the development and/or run-time options or may be purchased from a third-party software vendor, without licensing the source code or use of the OpenGLtrademark.

20. How do we make shadows in OpenGL?

There are no individual routines to control neither shadows nor an OpenGL state for shadows. However, code can be written to render shadows.

21. What is the use of GlutInit?

void glutInit(int *argcp, char **argv); glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInit may cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized.

22. Describe the usage of glutInitWindowSize and glutInitWindowPosition?

void glutInitWindowSize(int width, int height); void glutInitWindowPosition(int x, int y); Windows created by glutCreateWindow will be requested to be created with the current initial window position and size. The intent of the initial window position and size values is
to provide a suggestion to the window system for a window's initial size and position. The window system is not obligated to use this information. Therefore, GLUT programs should not assume the window was created at the specified size or position. A GLUT program should use the window's reshape callback to determine the true size of the window.

23. Describe the usage of glutMainLoop?

void glutMainLoop(void); glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program. Once called, this routine will never return. It will call as necessary any callbacks that have been registered.