

# **ATME COLLEGE OF ENGINEERING**

**13<sup>th</sup> KM Stone, Bannur Road, Mysore - 560 028**



**A T M E**  
**College of Engineering**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**(ACADEMIC YEAR 2023-24)**

## **LABORATORY MANUAL**

**SUBJECT: COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY**

**SUBJECT CODE: 21CSL66**

**SEMESTER: VI**

**Outcome Based Education(OBE) and Choice Based Credit System (CBCS)**

**(Effective from the academic year 2021 - 22)**

**Composed By**

**Verified By**

**Approved By**

**Mr. Rajiv P**

**Mrs. Keerthana M M Mrs. Kavyashree E D**

**Dr. Puttegowda D**

**Programmer**

**Faculty Co-Ordinators**

**Professor & Head, CSE**

# **INSTITUTIONAL MISSION AND VISION**

## **Objectives**

- To provide quality education and groom top-notch professionals, entrepreneurs and leaders for different fields of engineering, technology and management.
- To open a Training-R & D-Design-Consultancy cell in each department, gradually introduce doctoral and postdoctoral programs, encourage basic & applied research in areas of social relevance, and develop the institute as a center of excellence.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To develop academic, professional and financial alliances with the industry as well as the academia at national and transnational levels.
- To cultivate strong community relationships and involve the students and the staff in local community service.
- To constantly enhance the value of the educational inputs with the participation of students, faculty, parents and industry.

## **Vision**

- Development of academically excellent, culturally vibrant, socially responsible and globally competent human resources.

## **Mission.**

- To keep pace with advancements in knowledge and make the students competitive and capable at the global level.
- To create an environment for the students to acquire the right physical, intellectual, emotional and moral foundations and shine as torch bearers of tomorrow's society.
- To strive to attain ever-higher benchmarks of educational excellence.

# Department of Computer Science & Engineering

## Vision of the Department

- To develop highly talented individuals in Computer Science and Engineering to deal with real world challenges in industry, education, research and society.

## Mission of the Department

- To inculcate professional behavior, strong ethical values, innovative research capabilities and leadership abilities in the young minds & to provide a teaching environment that emphasizes depth, originality and critical thinking.
- Motivate students to put their thoughts and ideas adoptable by industry or to pursue higher studies leading to research.

## Program outcomes (POs)

### Engineering Graduates will be able to:

- **PO1. Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems
- **PO2. 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.
- **PO3. 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.
- **PO4. 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.
- **PO5. 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.

- **PO6. 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
- **PO7. 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.
- **PO8. Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
- **PO9. Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
- **PO10. 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.
- **PO11. 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.
- **PO12. 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.

### **Program Educational Objectives (PEO'S):**

1. Empower students with a strong basis in the mathematical, scientific and engineering fundamentals to solve computational problems and to prepare them for employment, higher learning and R&D.
2. Gain technical knowledge, skills and awareness of current technologies of computer science engineering and to develop an ability to design and provide novel engineering solutions for software/hardware problems through entrepreneurial skills.
3. Exposure to emerging technologies and work in teams on interdisciplinary projects with effective communication skills and leadership qualities.
4. Ability to function ethically and responsibly in a rapidly changing environment by Applying innovative ideas in the latest technology, to become effective professionals in Computer Science to bear a life-long career in related areas.

## **Program Specific Outcomes (PSOs)**

1. Ability to apply skills in the field of algorithms, database design, web design, cloud computing and data analytics.
2. Apply knowledge in the field of computer networks for building network and internet based applications.

## COMPUTER GRAPHICS AND IMAGE PROCESSING LABORATORY

Subject Code	:	21CSL66	CIE Marks	:	50
Teaching Hours/Week (L:T:P: S)	:	0:0:2:0	SEE Marks	:	50
Total Hours of Pedagogy	:	24	Total Marks	:	100
Credits	:	1	Exam Hours	:	03

Course objectives:

CLO 1: Demonstrate the use of Open GL.

CLO 2: Demonstrate the different geometric object drawing using OpenGL

CLO 3: Demonstration of 2D/3D transformation on simple objects.

CLO 4: Demonstration of lighting effects on the created objects.

CLO 5: Demonstration of Image processing operations on image/s.

### Practise Programs:

- Installation of OpenGL /OpenCV/ Python and required headers
- Simple programs using OpenGL (Drawing simple geometric object like line, circle, rectangle, square)
- Simple programs using OpenCV (operation on an image/s)

## PART A

**List of problems for which student should develop program and execute in the Laboratory using OpenGL/openCV/ Python:**

1. Develop a program to draw a line using Bresenham's line drawing technique
2. Develop a program to demonstrate basic geometric operations on the 2D object
3. Develop a program to demonstrate basic geometric operations on the 3D object
4. Develop a program to demonstrate 2D transformation on basic objects
5. Develop a program to demonstrate 3D transformation on 3D objects
6. Develop a program to demonstrate Animation effects on simple objects.
7. Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left
8. Write a program to show rotation, scaling, and translation on an image
9. Read an image and extract and display low-level features such as edges, textures using filtering techniques.
10. Write a program to blur and smoothing an image.

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

**PART 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 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 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 geometric transformation 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 down 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 8<sup>th</sup> week of the semester and the second test shall be conducted after the 14<sup>th</sup> 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

**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

**Web links 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-inpythonfb722e805e8b>

## CONTENT LIST

SL.NO.	EXPERIMENT NAME	PAGE NO.
1	Introduction	1
2	Sample Programs	14
3	<b>Program 1:</b> Develop a program to draw a line using Bresenham's line drawing technique	24
4	<b>Program 2:</b> Develop a program to demonstrate basic geometric operations on the 2D object	27
5	<b>Program 3:</b> Develop a program to demonstrate basic geometric operations on the 3D object	30
6	<b>Program 4:</b> Develop a program to demonstrate 2D transformation on basic objects	33
7	<b>Program 5:</b> Develop a program to demonstrate 3D transformation on 3D objects	37
8	<b>Program 6:</b> Develop a program to demonstrate Animation effects on simple objects.	41
9	<b>Program 7:</b> Write a Program to read a digital image. Split and display image into 4 quadrants, up, down, right and left	44
10	<b>Program 8:</b> Write a program to show rotation, scaling, and translation on an image	46
11	<b>Program 9:</b> Read an image and extract and display low-level features such as edges, textures using filtering techniques.	48
12	<b>Program 10:</b> Write a program to blur and smoothing an image.	50
13	<b>Program 11:</b> Write a program to contour an image.	53
14	<b>Program 12:</b> Write a program to detect a face/s in an image.	55
15	<p style="text-align: center;"><b>PART B Practical Based Learning</b></p> <p><b>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:</b></p> <ul style="list-style-type: none"> <li>➤ Recognition of License Plate through Image Processing</li> <li>➤ Recognition of Face Emotion in Real-Time</li> <li>➤ Detection of Drowsy Driver in Real-Time</li> <li>➤ Recognition of Handwriting by Image Processing</li> <li>➤ Detection of Kidney Stone</li> <li>➤ Verification of Signature</li> <li>➤ Compression of Color Image</li> </ul>	57

	<ul style="list-style-type: none"> <li>➤ Classification of Image Category</li> <li>➤ Detection of Skin Cancer</li> <li>➤ Marking System of Attendance using Image Processing</li> <li>➤ Detection of Liver Tumor</li> <li>➤ IRIS Segmentation</li> <li>➤ Detection of Skin Disease and / or Plant Disease</li> <li>➤ Biometric Sensing System.</li> <li>➤ Projects which helps to formers to understand the present developments in agriculture</li> <li>➤ Projects which helps high school/college students to understand the scientific problems.</li> <li>➤ Simulation projects which helps to understand innovations in science and technology</li> </ul>	
16	Viva Questions and answers	60

## 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 languages. 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).

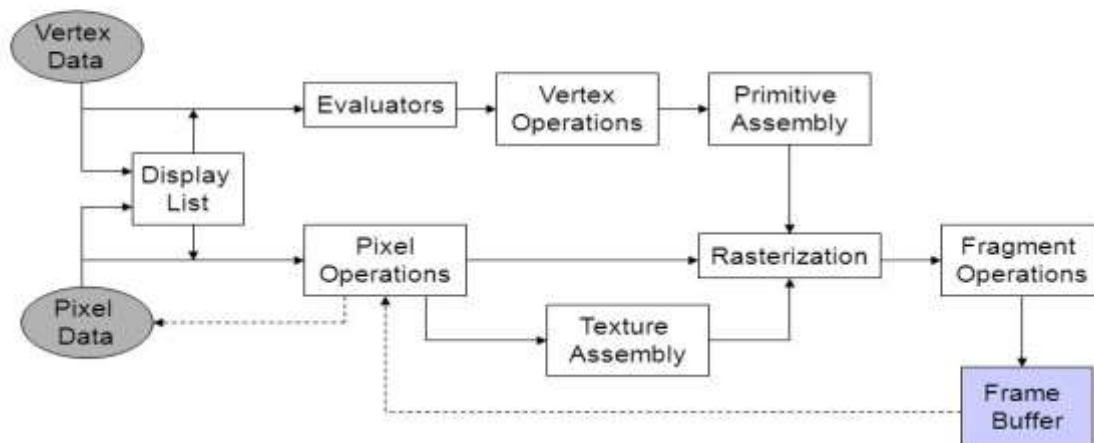
### GLUT

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).

### Rendering Pipeline

Most implementations of OpenGL have a similar order of operations, a series of processing stages called the OpenGL rendering pipeline. Although this is not a strict rule of how OpenGL is implemented, it provides a reliable guide for predicting what OpenGL will do. Geometric data (vertices, line, and polygons) follow a path through the row of boxes that includes evaluators and per-vertex operations, while pixel data (pixels, images and bitmaps) are treated differently for part of the process. Both types of data undergo the same final step (rasterization) before the final pixel data is written to the frame buffer.

## OpenGL Rendering Pipeline



OpenGL can be considered as a **state machine**. It has many states (modes) that control various aspects of the rendering pipeline. These states remain effective until you change them.

**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-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. Evaluators provide a method for deriving the vertices used to represent the surface from the control points. The method is a polynomial mapping, which can produce surface normal, colors, and spatial coordinate values from the control points.

**Per-Vertex and Primitive Assembly:** For vertex data, the next step converts the vertices into primitives. Some types of vertex data are transformed by 4x4 floating-point matrices. Spatial coordinates are projected from a position in the 3D world to a position on your screen. In some cases, this is followed by perspective division, which makes distant geometric objects appear smaller than closer objects. Then view port and depth operations are applied. The results at this point are geometric primitives, which are transformed with related color and depth 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, processed by a pixel map, and sent to the rasterization step.

**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 width, point size, shading model, and coverage calculations to support antialiasing 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. The processed fragment is then drawn into the appropriate buffer, where it has finally advanced to be a pixel and achieved its final resting place.

## Libraries

OpenGL provides a powerful but primitive set of rendering command, and all higher-level drawing must be done in terms of these commands. There are several libraries that allow you to simplify your programming tasks, including the following:

- 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 and rendering surfaces.
- OpenGL Utility Toolkit (GLUT) is a window-system-independent toolkit, written by Mark Kilgard, to hide the complexities of differing window APIs.

## Include Files

For all OpenGL applications, you want to include the `gl.h` header file in every file. Almost all OpenGL applications use GLU, the aforementioned OpenGL Utility Library, which also requires inclusion of the `glu.h` header file. So almost every OpenGL source file begins with:

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

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

If you are using the OpenGL Utility Toolkit (GLUT) for managing your window manager tasks, you should include:

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

Note that `glut.h` guarantees that `gl.h` and `glu.h` are properly included for you so including these three files is redundant. To make your GLUT programs portable, include `glut.h` and do not include `gl.h` or `glu.h` explicitly.

The first things you will need are the OpenGL and GLUT header files and libraries for your current Operating System.

## Setting up Compilers

### • Install OpenGL on Ubuntu

Here we will be looking at step by step process to install OpenGL:

Run the following commands to install OpenGL.

```
$ sudo apt-get update
$ sudo apt-get install libglu1-mesa-dev freeglut3-dev mesa-common-dev
```

To test if OpenGL libraries are working fine on our Ubuntu 18.04 LTS, we will create a C/C++ program and test it.

So create a following C/C++ Program.

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

void displayMe(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glBegin(GL_POLYGON);
        glVertex3f(0.5, 0.0, 0.5);
        glVertex3f(0.5, 0.0, 0.0);
        glVertex3f(0.0, 0.5, 0.0);
        glVertex3f(0.0, 0.0, 0.5);
    glEnd();
    glFlush();
}

int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE);
    glutInitWindowSize(400, 300);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("Hello world!");
    glutDisplayFunc(displayMe);
    glutMainLoop();
    return 0;
}
```

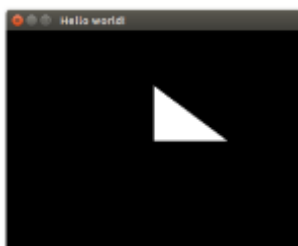
Now give the command below to compile your code.

```
$ gcc sample.c -lglut -lGLU -lGL
```

Now run your OpenGL program with following command

```
$ ./a.out
```

OUTPUT:



## 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 be useful in Programming at the Software Industry.

### Setting up GLUT - main ()

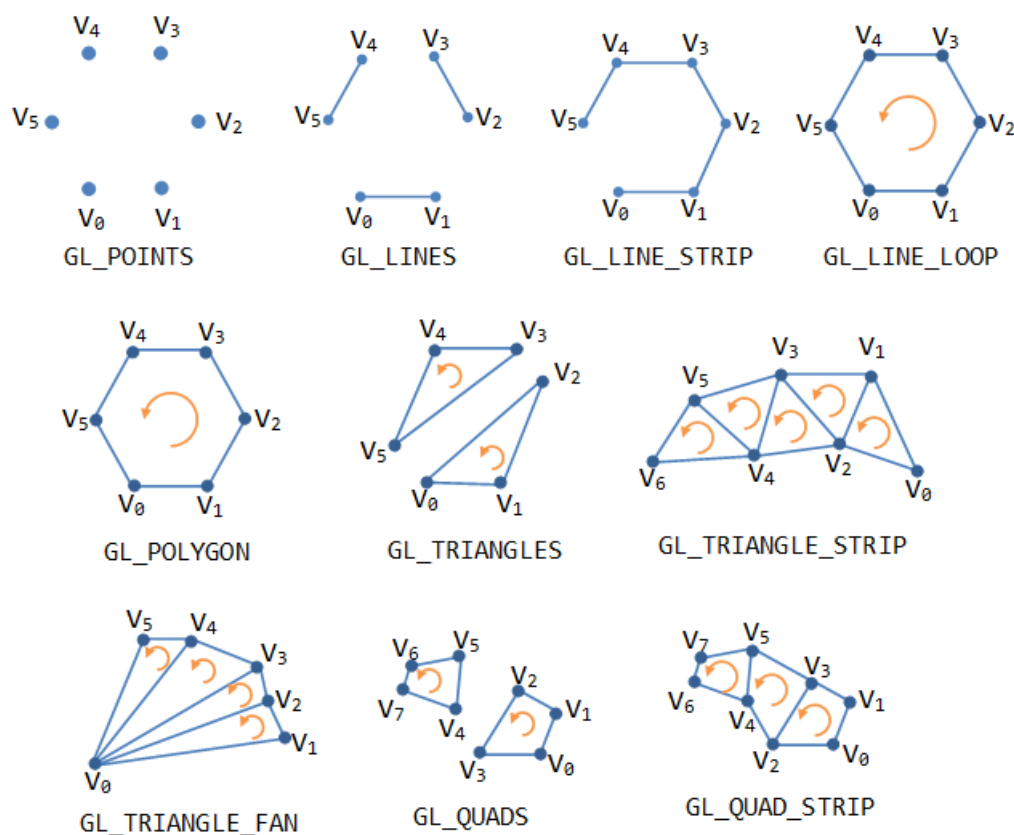
GLUT provides high-level utilities to simplify OpenGL programming, especially in interacting with the Operating System (such as creating a window, handling key and mouse inputs). The following GLUT functions were used in the above program:

- `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 terms 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(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble zNear, GLdouble zFar)` The `glOrtho` function multiplies the current matrix by an orthographic matrix.
- `void glPointSize (GLfloat size)`; The `glPointSize` function specifies the diameter of rasterized points.



- void glutPostRedisplay(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.
- GLint glRenderMode (GLenum mode); The glRenderMode function sets the rasterization mode.
- void glRotatf (GLfloat angle, GLfloat x, GLfloat y, GLfloat z); The glRotatf 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.

### Geometric Objects used in graphics



**OpenGL Primitives**

**Point, Lines and Polygons**

Each geometric object is described by a set of vertices and the type of primitive to be drawn. A vertex is no more than a point defined in three dimensional space. Whether and how these vertices are connected is determined by the primitive type. Every geometric object is ultimately described as an ordered set of vertices. We use the `glVertex*()` command to specify a vertex. The '\*' is used to indicate that there are variations to the base command `glVertex()`.

Some OpenGL command names have one, two, or three letters at the end to denote the number and type of parameters to the command. The first character indicates the number of values of the indicated type that must be presented to the command. The second character indicates the specific type of the arguments. The final character, if present, is 'v', indicating that the command takes a pointer to an array (a vector) of values rather than a series of individual agreements.

For example, in the command `glVertex3fv()`, '3' is used to indicate three arguments, 'f' is used to indicate the arguments are floating point, and 'v' indicates that the arguments are in vector format.

**Points:** A point is represented by a single vertex. Vertices specified by the user as two dimensional (only x- and y-coordinates) are assigned a z-coordinate equal to zero. To control the size of a rendered point, use `glPointSize()` and supply the desired size in pixels as the argument. The default is as 1 pixel by 1 pixel point. If the width specified is 2.0, the point will be a square of 2 by 2 pixels. `glVertex*()` is used to describe a point, but it is only effective between a `glBegin()` and a `glEnd()` pair. The argument passed to `glBegin()` determines what sort of geometric primitive is constructed from the vertices.

**Lines:** In OpenGL, the term line refers to a line segment, not the mathematician's version that extends to infinity in both directions. The easiest way to specify a line is in terms of the vertices at the endpoints. As with the points above, the argument passed to `glBegin()` tells it what to do with the vertices. The option for lines includes:

**GL\_LINES:** Draws a series of unconnected line segments drawn between each set of vertices. An extraneous vertex is ignored.

**GL\_LINE\_STRIP:** Draws a line segment from the first vertex to the last. Lines can intersect arbitrarily.

**GL\_LINE\_LOOP:** Same as `GL_STRIP`, except that a final line segment is drawn from the last vertex back to the first.

With OpenGL, the description of the shape of an object being drawn is independent of the description of its color. When a particular geometric object is drawn, it's drawn using the currently specified coloring scheme. In general, an OpenGL programmer first sets the color, using `glColor*()` and then draws the objects. Until the color is changed, all objects are drawn in that color or using that color scheme.

**Polygons:** Polygons are the areas enclosed by single closed loops of line segments, where the line segments are specified by the vertices at their endpoints. Polygons are typically drawn with the pixels in the interior filled in, but you can also draw them as outlines or a set of points. In OpenGL, there are a few restrictions on what constitutes a primitive polygon. For example, the edges of a polygon cannot intersect and they must be convex (no indentations). There are special commands for a three-sided (triangle) and four-sided (quadrilateral) polygons,

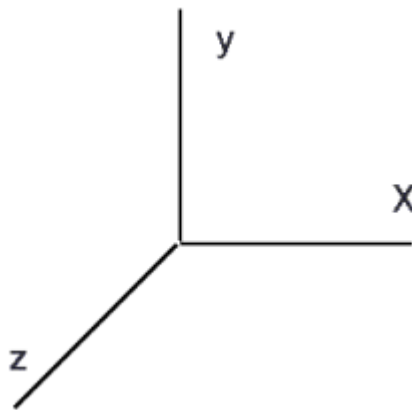
`glBegin(GL_TRIANGLES)` and `glBegin(GL_QUADS)`, respectively. However, the general case of a polygon can be defined using `glBegin(GL_POLYGON)`.

### Drawing 3-D Objects

GLUT has a series of drawing routines that are used to create three-dimensional models. This means we don't have to reproduce the code necessary to draw these models in each program. These routines render all their graphics in immediate mode. Each object comes in two flavors, wire or solid. Available objects are:

### Transformations

A modeling transformation is used to position and orient the model. For example, you can rotate, translate, or scale the model - or some combination of these operations. To make an object appear further away from the viewer, two options are available - the viewer can move closer to the object or the object can be moved further away from the viewer. Moving the viewer will be discussed later when we talk about viewing transformations. For right now, we will keep the default "camera" location at the origin, pointing toward the negative z-axis, which goes into the screen perpendicular to the viewing plane.



When transformations are performed, a series of matrix multiplications are actually performed to affect the position, orientation, and scaling of the model. You must, however, think of these matrix multiplications occurring in the opposite order from how they appear in the code. The order of transformations is critical. If you perform transformation A and then perform transformation B, you almost always get something different than if you do them in the opposite order.

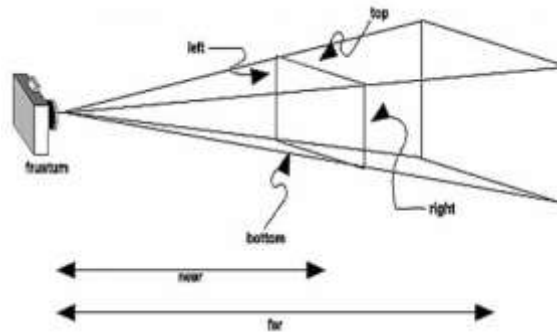
**Scaling:** The scaling command `glScale()` multiplies the current matrix by a matrix that stretches, shrinks, or reflects an object along the axes. Each x-, y-, and z-coordinate of every point in the object is multiplied by the corresponding argument x, y, or z. The `glScale*()` is the only one of the three modeling transformations that changes the apparent size of an object: scaling with values greater than 1.0 stretches an object, and using values less than 1.0 shrinks it. Scaling with a -1.0 value reflects an object across an axis.

**Translation:** The translation command `glTranslate()` multiplies the current matrix by a matrix that moves (translates) an object by the given x-, y-, and z-values.

**Rotation:** The rotation command `glRotate()` multiplies the current matrix that rotates an object in a counterclockwise direction about the ray from the origin through the point (x,y,z). The angle parameter specifies the angle of rotation in degrees. An object that lies farther from the axis of rotation is more dramatically rotated (has a larger orbit) than an object drawn near the axis.

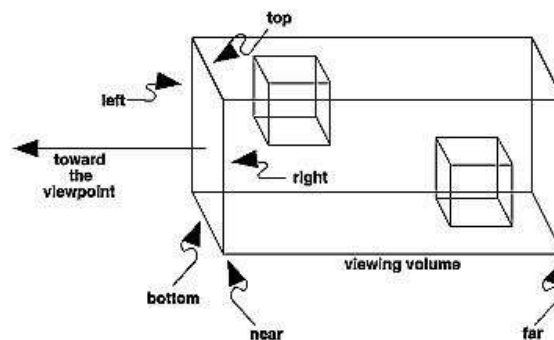
## Projections in OpenGL

### Perspective projection



```
void glFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
```

### Orthographic projection



```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far);
```

## What is Image Processing?

Image processing is the field of study and application that deals with modifying and analyzing digital images using computer algorithms. The goal of image processing is to enhance the visual quality of images, extract useful information, and make images suitable for further analysis or interpretation.

## Image Processing Using OpenCV

OpenCV (Open Source Computer Vision) is a powerful and widely-used library for image processing and computer vision tasks. It provides a comprehensive set of functions and tools that facilitate the development of applications dealing with images and videos.

While taking photographs is as simple as pressing a button, processing and improving those images sometimes takes more than a few lines of code. That's where image processing libraries like OpenCV come into play. OpenCV is a popular open-source package that covers a wide range of image processing and computer vision capabilities and methods. It supports multiple programming languages including Python, C++, and Java. OpenCV is highly tuned for real-time applications and has a wide range of capabilities.

## Image Processing with Python

We will make the following operations most commonly uses for data augmentation task which training the model in computer Vision.

- Image Resizing
- Image Rotation
- Image Translation
- Image Normalization
- Edge detection of Image
- Image Blurring
- Morphological Image Processing

## Image Resizing

Scaling operations increase or reduce the size of an image.

- **The cv2.resize() function** is used to resize a python image in OpenCV. It takes the following arguments:

```
cv2.resize(src, dsize, interpolation)
Here,
src          :The image to be resized.
dsize        :The desired width and height of the resized image.
interpolation: The interpolation method to be used.
```

- When the python image is resized, the **interpolation** method defines how the new pixels are computed. There are several interpolation techniques, each of which has its own quality vs. speed trade-offs.

- **It is important to note that resizing an image can reduce its quality.** This is because the new pixels are calculated by interpolating between the existing pixels, and this can introduce some blurring.

## Image Rotation

Images can be rotated to any degree clockwise or otherwise. We just need to define rotation matrix listing rotation point, degree of rotation and the scaling factor.

- The `cv2.getRotationMatrix2D ()` function is used to create a rotation matrix for an image. It takes the following arguments:
  - The center of rotation for the image.
  - The angle of rotation in degrees.
  - The scale factor.
- The `cv2.warpAffine()` function is used to apply a transformation matrix to an image. It takes the following arguments:
  - The python image to be transformed.
  - The transformation matrix.
  - The output image size.
- The rotation angle can be positive or negative. A positive angle rotates the image clockwise, while a negative angle rotates the image counterclockwise.
- The scale factor can be used to scale the image up or down. A scale factor of 1 will keep the image the same size, while a scale factor of 2 will double the size of the python image.

## Image Translation

Translating an image means shifting it within a given frame of reference that can be along the x-axis and y-axis.

- **To translate an image using OpenCV**, we need to create a transformation matrix. This matrix is a  $2 \times 3$  matrix that specifies the amount of translation in each direction.
- **The `cv2.warpAffine()` function** is used to apply a transformation matrix to an image. It takes the following arguments:
  - The image to be transformed.
  - The transformation matrix.
  - The output image size.
- **The translation parameters** are specified in the transformation matrix as the tx and ty elements. The tx element specifies the amount of translation in the x-axis, while the ty element specifies the amount of translation in the y-axis.

## Edge detection of Image

The process of image edge detection involves detecting sharp edges in the image. This edge detection is essential in the context of image recognition or object localization/detection. There are several algorithms for detecting edges due to its wide applicability.

In image processing and computer vision applications, Canny Edge Detection is a well-liked edge detection approach. In order to detect edges, the Canny edge detector first smoothes the image to reduce noise, then computes its gradient, and then applies a threshold to the gradient. The multi-stage Canny edge detection method includes the following steps:

- Gaussian smoothing: The image is smoothed using a Gaussian filter to remove noise.
- Gradient calculation: The gradient of the image is calculated using the Sobel operator.
- Non-maximum suppression: Non-maximum suppression is applied to the gradient image to remove spurious edges.
- Hysteresis thresholding: Hysteresis thresholding is applied to the gradient image to identify strong and weak edges.

The Canny edge detector is a powerful edge detection algorithm that can produce high-quality edge images. However, it can also be computationally expensive.

## Image Blurring

Image blurring is the technique of reducing the detail of an image by averaging the pixel values in the neighborhood. This can be done to reduce noise, soften edges, or make it harder to identify a picture. In many image processing tasks, image blurring is a common preprocessing step. It is useful in the optimization of algorithms such as image classification, object identification, and image segmentation. In OpenCV, a variety of different blurring methods are available, each with a particular trade-off between blurring strength and speed.

Some of the most common blurring techniques include:

- Gaussian blurring: This is a popular blurring technique that uses a Gaussian kernel to smooth out the image.
- Median blurring: This blurring technique uses the median of the pixel values in a neighborhood to smooth out the image.
- Bilateral blurring: This blurring technique preserves edges while blurring the image.


A bilateral filter is used for smoothening images and reducing noise, while **preserving edges**. This article explains an approach using the averaging filter, while this article provides one using a median filter. However, these convolutions often result in a loss of important edge information, since they blur out everything, irrespective of it being noise or an edge. To counter this problem, The none-linear bilateral filter was introduced.

## Gaussian Blur

Gaussian blurring can be formulated as follows:

$$GB[I]_p = \sum_{q \in S} G_{\sigma}(\|p - q\|) I_q$$

↓  
Normalized Gaussian  
Function




Here,  $GA[p]$  is the result at pixel  $p$ , and the RHS is essentially a sum over all pixels  $q$  weighted by the Gaussian function.  $I_q$  is the intensity at pixel  $q$ .

### Bilateral Filter: an Additional Edge Term


The bilateral filter can be formulated as follows:

$$BF[I]_p = \frac{1}{W_p} \sum_{q \in S} G_{\sigma_s}(\|p - q\|) G_{\sigma_r}(|I_p - I_q|) I_q$$

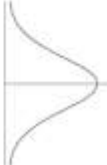
Normalization  
Factor



Space Weight



Range Weight



Here, the normalization factor and the range weight are new terms added to the previous equation.  $\partial$  denotes the spatial extent of the kernel, i.e. the size of the neighborhood, and  $\partial$  denotes the minimum amplitude of an edge. It ensures that only those pixels with intensity values similar to that of the central pixel are considered for blurring, while sharp intensity changes are maintained. The smaller the value of  $\partial$ , the sharper the edge. As  $\partial_r$  tends to infinity, the equation tends to a Gaussian blur.

OpenCV has a function called `bilateralFilter()` with the following arguments:

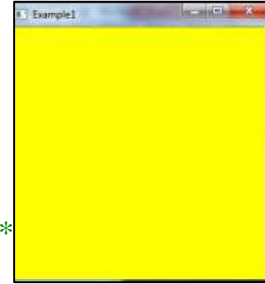
1. `d`: Diameter of each pixel neighborhood.
2. `sigmaColor`: Value of  $\partial$  in the color space. The greater the value, the colors farther to each other will start to get mixed.
3. `sigmaSpace`: Value of  $\partial$  in the coordinate space. The greater its value, the further pixels will mix together, given that their colors lie within the `sigmaColor` range.



## Sample Programs

### 1) Creating a Window:

```
#include<GL/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);/*default color –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 */
}
```



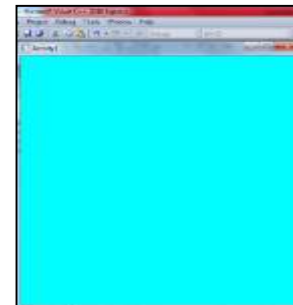
### Activity 1:

Change the window size to 500, 500

Change the window position to 100, 100

Change the window color to CYAN

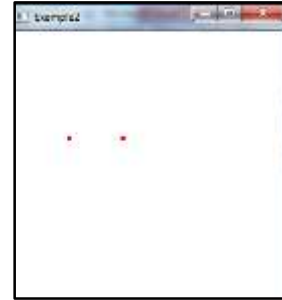
```
#include<GL/glut.h>
void display (void)
{
    glClearColor (0.0,1.0,1.0,0.0);
    glClear (GL_COLOR_BUFFER_BIT);
    glFlush();
}
void main (int argc, char **argv)
{
    glutInit (&argc, argv); /* Initialise OpenGL */
    glutInitWindowSize(500,500);
    glutInitWindowPosition(100,100);
    glutCreateWindow ("Activity1"); /* Create the window */
    glutDisplayFunc (display); /* Register the "display" function */
    glutMainLoop (); /* Enter the OpenGL main loop */
}
```



**2) Drawing pixels/points:**

```
#include<GL/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();
}
```

**3) Drawing lines :**

```
#include<GL/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();
}
```

### Activity 3

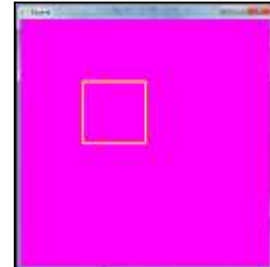
Change the window color to MAGENTA

Change the line color to YELLOW

Change the line width to width to 4

Draw a square using 4 lines

```
#include<GL/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 (100, 150);
glVertex2d (50, 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 using LINE\_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();  
}
```

**Activity 4**

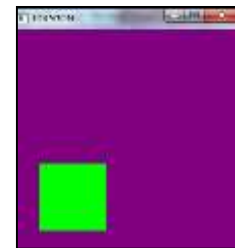
Change the window color to PURPLE

Change the line color to GREEN

Change the line width to width to 3

Draw a square using GL\_POLYGON

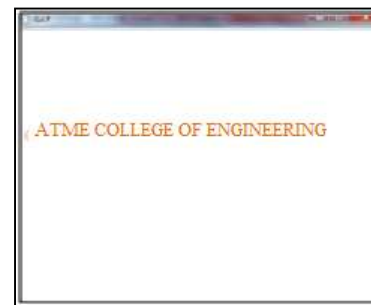
```
#include<GL/glut.h>  
void display()  
{  
    glClear(GL_COLOR_BUFFER_BIT);  
  
    glColor3f(0.0, 1.0, 0.0); // set line color to green  
  
    glLineWidth(3.0);  
  
    glBegin(GL_POLYGON);  
  
    glVertex2f(50, 50);  
  
    glVertex2f(200, 50);  
  
    glVertex2f(200, 200);  
  
    glVertex2f(50, 200);  
  
    glEnd();  
}  
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) Writing Text

```
#include<GL/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,"ATME COLLEGE OF ENGINEERING");  
    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(&2wargc,argv);  
    glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);  
    glutInitWindowSize(500,500);  
    glutInitWindowPosition(0,0);  
    glutCreateWindow("ATME");  
    glutDisplayFunc(display);
```



```
myinit();  
glutMainLoop();  
}
```

### 6) Drawing colored line and writing Text

```
#include<GL/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();  
}
```



**7) Drawing a colored square**

```
#include<GL/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)
```

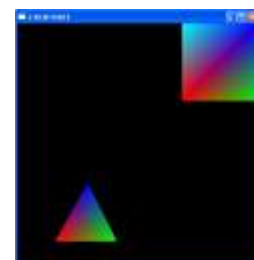
```
{
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE|GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("COLORED SQUARE");
glutDisplayFunc(display);
myinit();
glutMainLoop();
}
```

**8) Creating 2 view ports**

```
#include<GL/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();
}
```

### 9) Program to read and display an image using openCV

```
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:**



### 10) 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:**



**PART A****Program 1: Develop a program to draw a line using Bresenham's line drawing technique****Program Objective:**

- creation of 2D objects
- Implement GLU and GLUT functions
- To have the detailed knowledge of the graphics Bresenham's line algorithm.

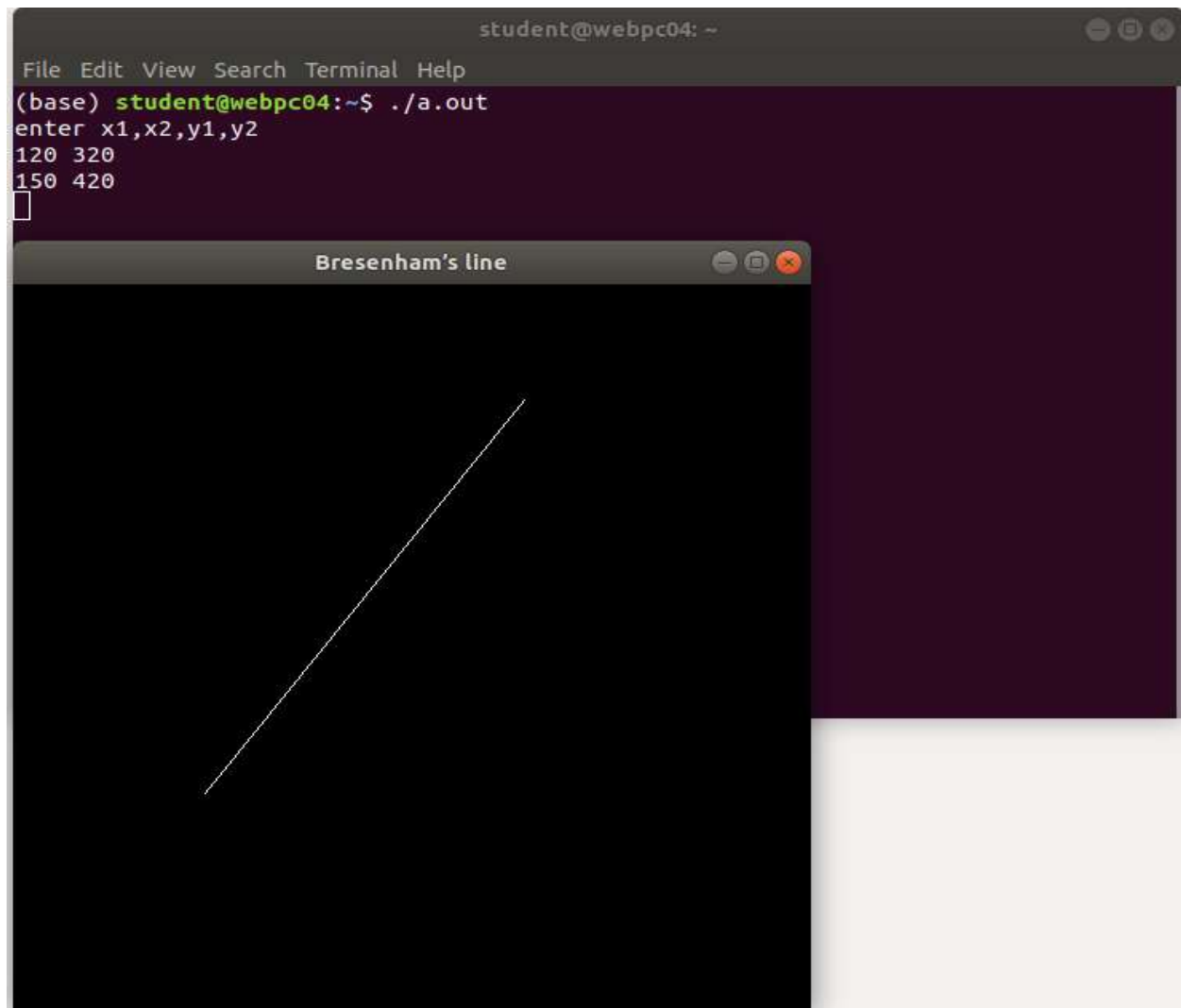
```
#include<GL/glut.h>
#include<stdio.h>
int x1,x2,y1,y2;
void myInit()
{
glClearColor(0.0,0.0,0.0,0.0);
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0,500,0,500);
}
void draw_pixel(int x,int y)
{
glBegin(GL_POINTS);
glVertex2i(x,y);
glEnd();
}
void draw_line(int x1,int x2,int y1,int y2)
{
int dx,dy,i,e;
int incx,incy,inc1,inc2;
int x,y;
dx=x2-x1;
dy=y2-y1;
if(dx<0)
dx=-dx;
if(dy<0)
dy=-dy;
incx=1;
if(x2<x1)
incx=-1;
incy=1;
if(y2<y1)
incy=-1;
x=x1;
y=y1;
if(dx>dy)
{
draw_pixel(x,y);
e=2*dy-dx;
inc1=2*(dy-dx);
inc2=2*dy;
for(i=0;i<dx;i++)
{
```

```
if(e>=0)
{
y+=incy;
e+=inc1;
}
else
e+=inc2;
x+=incx;
draw_pixel(x,y);
}
}
else
{
draw_pixel(x,y);
e=2*dx-dy;
inc1=2*(dx-dy);
inc2=2*dx;
for(i=0;i<dy;i++)
{
if(e>=0)
{
x+=incx;
e+=inc1;
}
else
e+=inc2;
y+=incy;
draw_pixel(x,y);
}
}
}
void myDisplay()
{
glClear(GL_COLOR_BUFFER_BIT);
draw_line(x1,x2,y1,y2);
glFlush();
}
int main(int argc,char **argv)
{
printf("enter x1,x2,y1,y2\n");
scanf("%d%d%d%d",&x1,&x2,&y1,&y2);
glutInit(&argc,argv);
glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
glutInitWindowSize(500,500);
glutInitWindowPosition(0,0);
glutCreateWindow("win");
glutDisplayFunc(myDisplay);
myInit();
glEnable(GL_DEPTH_TEST);
glClearColor(0.0,0.0,0.0,0.0);
glutMainLoop();
}
```

**RUN:**

```
g++ 1.cpp -lglut -lGL -lGLU
```

```
./a.out
```

**OUTPUT:**

S

**Program Outcome:**

- Ability to Design and develop 2D objects for different graphics applications.
- Use matrix algebra in computer graphics and implement Brenham's line algorithm.
- Analyze and evaluate the use of openGL methodss in practical applications of 2D representations.

**Program 2: Develop a program to demonstrate basic geometric operations on the 2D object****Program Objective:**

- creation of 2D objects
- Create 2D objects with basic transformation operations like translation, rotation, scaling.
- Use mathematical and theoretical principles i.e. transformation matrices of computer graphics to draw and translate, rotate, scaled object.

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

float rectangleWidth = 100.0f;
float rectangleHeight = 50.0f;

float translateX = 0.0f;
float translateY = 0.0f;

float rotateAngle = 0.0f;

float scaleX = 1.0f;
float scaleY = 1.0f;

void init() {
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-300.0, 300.0, -300.0, 300.0);
}

void drawRectangle() {
    glPushMatrix();
    glTranslatef(translateX, translateY, 0.0);
    glRotatef(rotateAngle, 0.0, 0.0, 1.0);
    glScalef(scaleX, scaleY, 1.0);

    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    glVertex2f(-rectangleWidth / 2, -rectangleHeight / 2);
    glVertex2f(rectangleWidth / 2, -rectangleHeight / 2);
    glVertex2f(rectangleWidth / 2, rectangleHeight / 2);
    glVertex2f(-rectangleWidth / 2, rectangleHeight / 2);
    glEnd();

    glPopMatrix();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);
    drawRectangle();
    glFlush();
}
```

```
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 't':
            translateX += 10.0f;
            translateY += 10.0f;
            break;
        case 'r':
            rotateAngle += 10.0f;
            break;
        case 's':
            scaleX += 0.1f;
            scaleY += 0.1f;
            break;
    }
    glutPostRedisplay();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("2D object Geometric Operations in OpenGL");

    init();
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);

    glutMainLoop();

    return 0;
}
```

**RUN:**

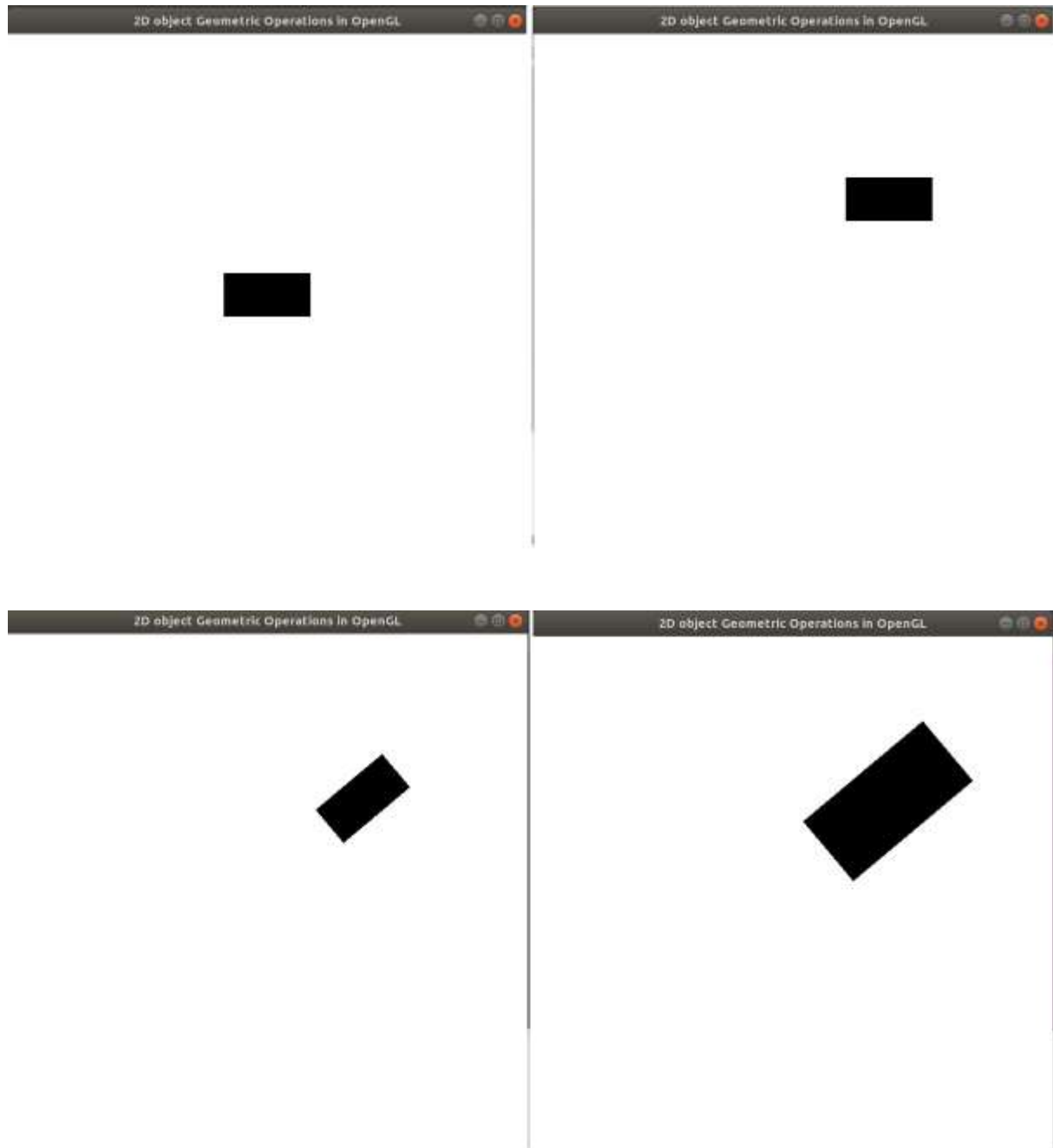
```
g++ 2.cpp -lglut -lGL -lGLU
```

```
./a.out
```

This program creates a rectangle using OpenGL and demonstrates three basic geometric operations:

1. Translation: Press 't' to translate the rectangle by 10 units in both the x and y directions.
2. Rotation: Press 'r' to rotate the rectangle by 10 degrees clockwise around its center.
3. Scaling: Press 's' to scale the rectangle by 0.1 in both the x and y directions.

Compile and run this program, then press the respective keys to perform the geometric operations. You should see the rectangle change accordingly on the screen.

**OUTPUT:****Program Outcome:**

- Ability to Design and develop 2D objects for different graphics applications.
- Use matrix algebra in computer graphics and implement fundamental transformations involved in transformation operation.
- Ability to draw objects using basic objects like points and lines.
- Analyze and evaluate the use of OpenGL methods in practical applications of 2D representations.



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

**Program Objective:**

- creation of 3D objects
- Create 3D object with basic transformation operations like translation, rotation, scaling.
- Use mathematical and theoretical principles i.e. transformation matrices of computer graphics to draw and translate, rotate, scaled object.

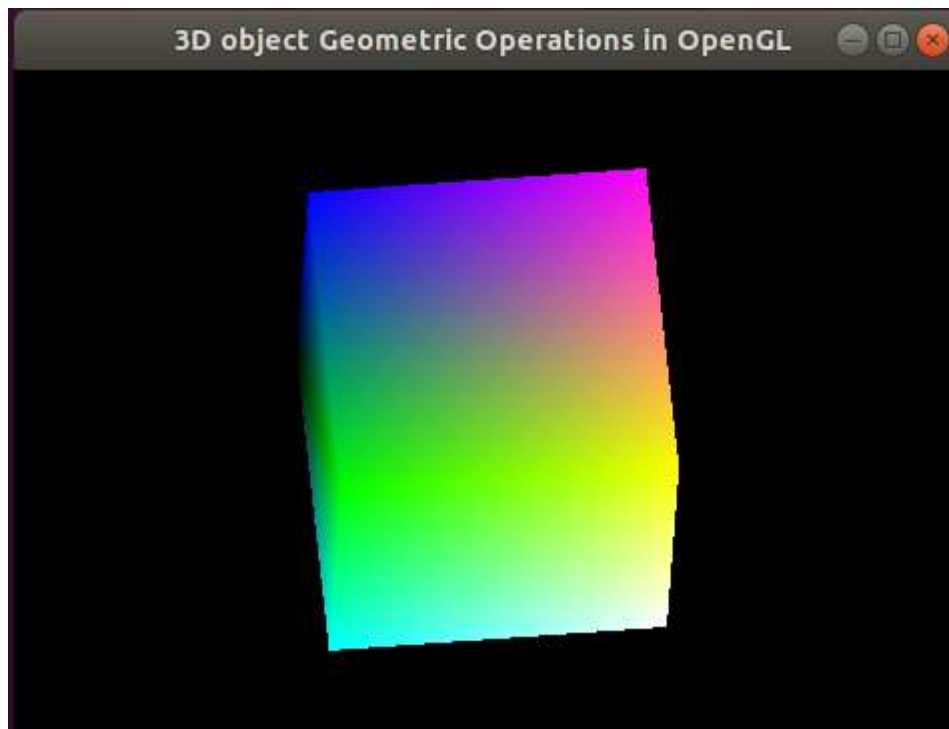
```
#include<GL/glut.h>
#include<stdio.h>
GLfloat      vertices[][3]={ {-1.0,-1.0,-1.0},{ 1.0,-1.0,-1.0},{ 1.0,1.0,-1.0},{ -1.0,1.0,-1.0},{ -1.0,-1.0,1.0},{ 1.0,-1.0,1.0},{ 1.0,1.0,1.0},{ -1.0,1.0,1.0}};
GLfloat
colors[][3]={ {0.0,0.0,0.0},{0.0,0.0,1.0},{0.0,1.0,0.0},{0.0,1.0,1.0},{ 1.0,0.0,0.0},{ 1.0,0.0,1.0},{ 1.0,1.0,0.0},{ 1.0,1.0,1.0}};
static GLfloat theta[]={0.0,0.0,0.0};
static GLint axis=2;
static GLdouble viewer[]={0,0,5};
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(1,2,6,5);
polygon(0,4,5,1);
polygon(4,5,6,7);
polygon(0,3,7,4);
}
void display()
{
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
glLoadIdentity();
glRotatef(theta[0],1.0,0.0,0.0);
glRotatef(theta[1],0.0,1.0,0.0);
glRotatef(theta[2],0.0,0.0,1.0);
colorcube();
glFlush();
glutSwapBuffers();
}
```

```
}
void spincube()
{
    theta[axis]+=2.0;
    if(theta[axis]>360.0)
        theta[axis]-=360.0;
    glutPostRedisplay();
}
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;
    spincube();
}
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();
}
int main(int argc,char**argv)
{
    glutInit(&argc,argv);
    glutInitDisplayMode(GLUT_DOUBLE|GLUT_RGB|GLUT_DEPTH);
    glutCreateWindow("3D object Geometric Operations in OpenGL ");
    glutReshapeFunc(myreshape);
    glutDisplayFunc(display);
    glutIdleFunc(spincube);
    glutMouseFunc(mouse);
    glEnable(GL_DEPTH_TEST);
    glClearColor(0.0,0.0,0.0,0.0);
    glutMainLoop();
    return 0;
}
```

**RUN:**

g++ 3.cpp -lglut -lGL -lGLU

./a.out

**OUTPUT:****Program Outcome:**

- Ability to Design and develop 3D objects for different graphics applications.
- Use matrix algebra in computer graphics and implement fundamental transformations involved in transformation operation.
- Ability to draw objects using basic objects like points and lines.
- Analyze and evaluate the use of OpenGL methods in practical applications of 3D representations.

**Program 4: Develop a program to demonstrate 2D transformation on basic objects****Program Objective:**

- creation of 2D objects
- Create 2D objects with basic transformation operations like translation, rotation, scaling.
- Use mathematical and theoretical principles i.e. transformation matrices of computer graphics to draw and rotate the color cube object.
- Use matrix algebra in computer graphics and implement fundamental algorithms and transformations involved in viewing models.

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

// Object properties
struct Object {
    float x;
    float y;
    float size;
};

// Initial positions and sizes of objects
Object rectangle = { -50.0f, 50.0f, 100.0f };
Object triangle = { 50.0f, 50.0f, 100.0f };
Object circle = { 0.0f, -50.0f, 50.0f };

// Transformation properties
float translateX = 0.0f;
float translateY = 0.0f;
float rotateAngle = 0.0f;
float scaleX = 1.0f;
float scaleY = 1.0f;

void init() {
    glClearColor(1.0, 1.0, 1.0, 0.0);
    glMatrixMode(GL_PROJECTION);
    gluOrtho2D(-300.0, 300.0, -300.0, 300.0);
}

void drawRectangle(float x, float y, float size) {
    glPushMatrix();
    glTranslatef(x, y, 0.0);
    glRotatef(rotateAngle, 0.0, 0.0, 1.0);
    glScalef(scaleX, scaleY, 1.0);

    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    glVertex2f(-size / 2, -size / 2);
    glVertex2f(size / 2, -size / 2);
    glVertex2f(size / 2, size / 2);
    glVertex2f(-size / 2, size / 2);
}
```

```
    glEnd();
    glPopMatrix();
}

void drawTriangle(float x, float y, float size) {
    glPushMatrix();
    glTranslatef(x, y, 0.0);
    glRotatef(rotateAngle, 0.0, 0.0, 1.0);
    glScalef(scaleX, scaleY, 1.0);

    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_TRIANGLES);
    glVertex2f(0.0, size / 2);
    glVertex2f(-size / 2, -size / 2);
    glVertex2f(size / 2, -size / 2);
    glEnd();

    glPopMatrix();
}

void drawCircle(float x, float y, float size) {
    const int numSegments = 50;
    const float angleIncrement = 2.0f * 3.14159f / numSegments;

    glPushMatrix();
    glTranslatef(x, y, 0.0);
    glRotatef(rotateAngle, 0.0, 0.0, 1.0);
    glScalef(scaleX, scaleY, 1.0);

    glColor3f(0.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    for (int i = 0; i < numSegments; ++i) {
        float angle = i * angleIncrement;
        glVertex2f(cos(angle) * size / 2, sin(angle) * size / 2);
    }
    glEnd();

    glPopMatrix();
}

void display() {
    glClear(GL_COLOR_BUFFER_BIT);

    drawRectangle(rectangle.x + translateX, rectangle.y + translateY, rectangle.size);
    drawTriangle(triangle.x + translateX, triangle.y + translateY, triangle.size);
    drawCircle(circle.x + translateX, circle.y + translateY, circle.size);

    glFlush();
}

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

```
case 't':
    translateX += 10.0f;
    translateY += 10.0f;
    break;
case 'r':
    rotateAngle += 10.0f;
    break;
case 's':
    scaleX += 0.1f;
    scaleY += 0.1f;
    break;
}
glutPostRedisplay();
}

int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB);
    glutInitWindowSize(600, 600);
    glutInitWindowPosition(100, 100);
    glutCreateWindow("2D Transformations in OpenGL");

    init();
    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);

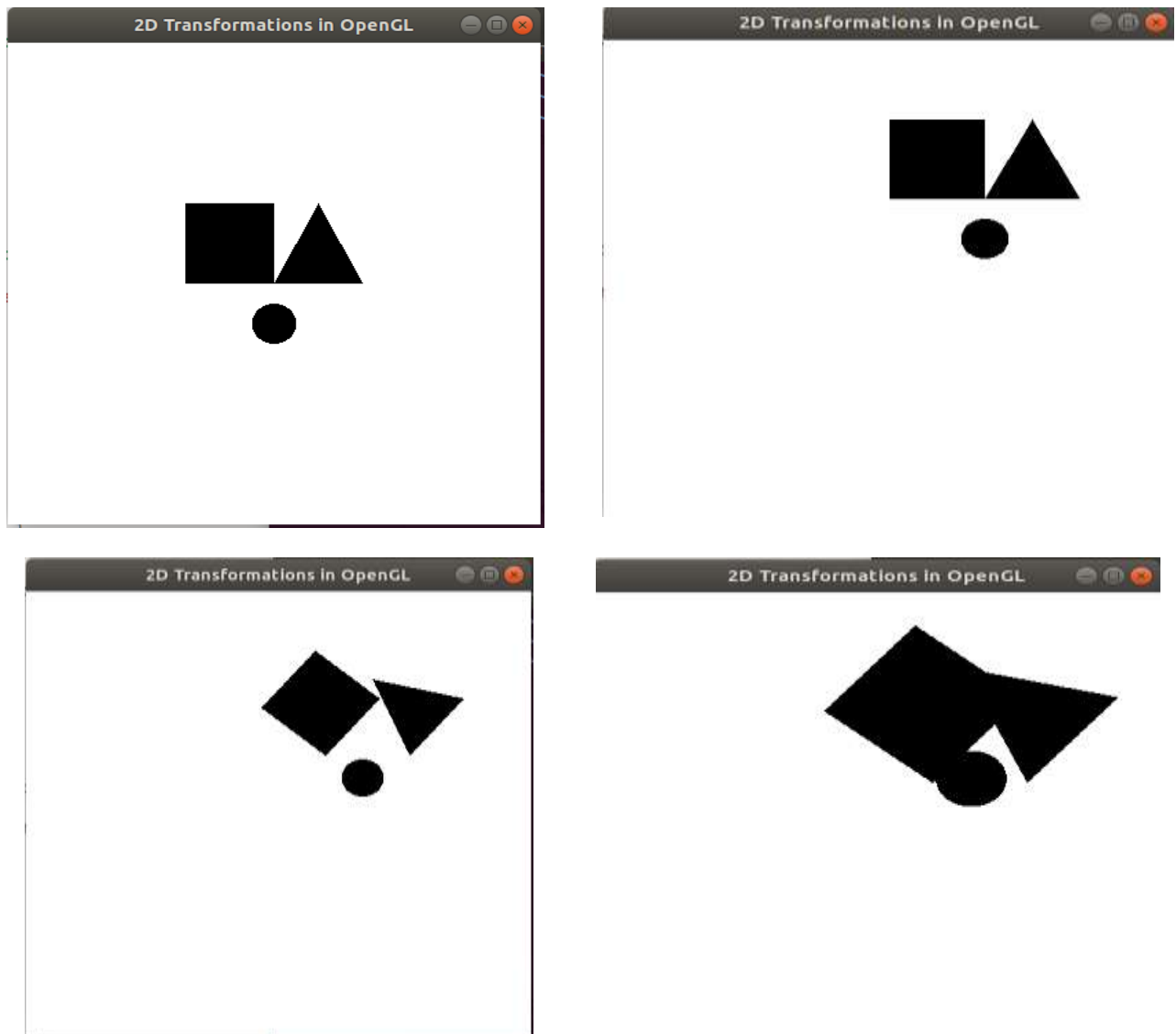
    glutMainLoop();

    return 0;
}
```

**RUN:**

```
g++ 4.cpp -lglut -lGL -lGLU
```

```
./a.out
```

**OUTPUT:****Program Outcome:**

- Ability to Design and develop 2D objects for different graphics applications.
- Use matrix algebra in computer graphics and implement fundamental transformations involved in viewing models.
- Analyze and evaluate the use of OpenGL methods in practical applications of 2D representations.

**Program 5: Develop a program to demonstrate 3D transformation on 3D objects****Program Objective:**

- creation of 3D objects
- Create 3D objects with basic transformation operations like translation, rotation, scaling.
- Use mathematical and theoretical principles i.e. transformation matrices of computer graphics to draw and rotate the color cube object.
- Use matrix algebra in computer graphics and implement fundamental algorithms and transformations involved in viewing models.

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

// Global variables
int width = 800;
int height = 600;
GLfloat translateX = 0.0f;
GLfloat translateY = 0.0f;
GLfloat rotationX = 0.0f;
GLfloat rotationY = 0.0f;
GLfloat scale = 1.0f;

// Function to draw a cube
void drawCube() {
    glBegin(GL_QUADS);

    // Front face
    glColor3f(1.0f, 0.0f, 0.0f); // Red
    glVertex3f(-0.5f, -0.5f, 0.5f);
    glVertex3f(0.5f, -0.5f, 0.5f);
    glVertex3f(0.5f, 0.5f, 0.5f);
    glVertex3f(-0.5f, 0.5f, 0.5f);

    // Back face
    glColor3f(0.0f, 1.0f, 0.0f); // Green
    glVertex3f(-0.5f, -0.5f, -0.5f);
    glVertex3f(-0.5f, 0.5f, -0.5f);
    glVertex3f(0.5f, 0.5f, -0.5f);
    glVertex3f(0.5f, -0.5f, -0.5f);

    // Top face
    glColor3f(0.0f, 0.0f, 1.0f); // Blue
    glVertex3f(-0.5f, 0.5f, -0.5f);
    glVertex3f(-0.5f, 0.5f, 0.5f);
    glVertex3f(0.5f, 0.5f, 0.5f);
    glVertex3f(0.5f, 0.5f, -0.5f);

    // Bottom face
    glColor3f(1.0f, 1.0f, 0.0f); // Yellow
    glVertex3f(-0.5f, -0.5f, -0.5f);
```



```
    glVertex3f(0.5f, -0.5f, -0.5f);
    glVertex3f(0.5f, -0.5f, 0.5f);
    glVertex3f(-0.5f, -0.5f, 0.5f);

    // Right face
    glColor3f(1.0f, 0.0f, 1.0f); // Magenta
    glVertex3f(0.5f, -0.5f, -0.5f);
    glVertex3f(0.5f, 0.5f, -0.5f);
    glVertex3f(0.5f, 0.5f, 0.5f);
    glVertex3f(0.5f, -0.5f, 0.5f);

    // Left face
    glColor3f(0.0f, 1.0f, 1.0f); // Cyan
    glVertex3f(-0.5f, -0.5f, -0.5f);
    glVertex3f(-0.5f, -0.5f, 0.5f);
    glVertex3f(-0.5f, 0.5f, 0.5f);
    glVertex3f(-0.5f, 0.5f, -0.5f);

    glEnd();
}

// Function to handle display
void display() {
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    // Apply transformations
    //glTranslatef(0.0f, 0.0f, -3.0f);
    glTranslatef(translateX, 0.0f, -3.0f);
    glTranslatef(0.0f, translateY, -3.0f);
    glRotatef(rotationX, 1.0f, 0.0f, 0.0f);
    glRotatef(rotationY, 0.0f, 1.0f, 0.0f);
    glScalef(scale, scale, scale);

    // Draw cube
    drawCube();

    glutSwapBuffers();
}

// Function to handle keyboard events
void keyboard(unsigned char key, int x, int y) {
    switch (key) {
        case 'x':
            rotationX += 5.0f;
            break;
        case 'X':
            rotationX -= 5.0f;
            break;
        case 'z':
```

```
        rotationY += 5.0f;
        break;
    case 'Z':
        rotationY -= 5.0f;
        break;
    case '+':
        scale += 0.1f;
        break;
    case '-':
        if (scale > 0.1f)
            scale -= 0.1f;
        break;
    case 't':
        translateX += 0.5f;
        break;
    case 'T':
        translateX -= 0.5f;
        break;
    case 'y':
        translateY += 0.5f;
        break;
    case 'Y':
        translateY -= 0.5f;
        break;
    case 27: // Escape key to exit
        exit(0);
        break;
    }

    glutPostRedisplay(); // Trigger a redraw
}

// Function to initialize OpenGL
void initializeOpenGL(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowSize(width, height);
    glutCreateWindow("Geometric Operations in 3D");

    glEnable(GL_DEPTH_TEST);
    glClearColor(1.0f, 1.0f, 1.0f, 1.0f); // White background

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45.0f, (float)width / (float)height, 1.0f, 100.0f);

    glutDisplayFunc(display);
    glutKeyboardFunc(keyboard);
}

// Main function
int main(int argc, char** argv) {
```

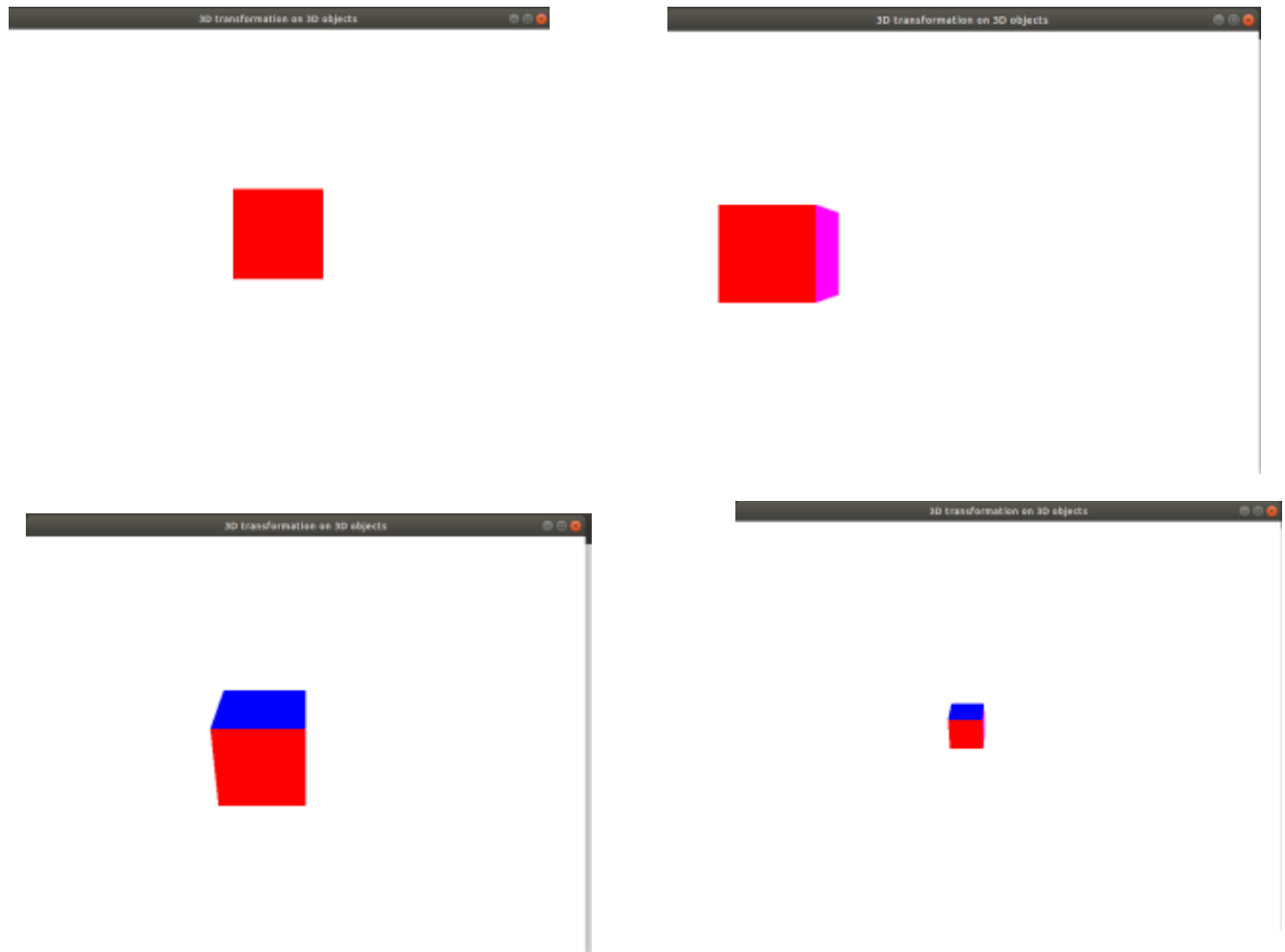
---

```
    initializeOpenGL(argc, argv);  
    glutMainLoop();  
    return 0;  
}
```

**RUN:**

```
g++ 5.cpp -lglut -lGL -lGLU
```

```
./a.out
```

**OUTPUT:****Program Outcome:**

- Ability to Design and develop 3D objects for different graphics applications.
- Use matrix algebra in computer graphics and implement fundamental transformations involved in viewing models.
- Analyze and evaluate the use of openGL methods in practical applications of 3D representations.

**Program 6: Develop a program to demonstrate Animation effects on simple objects.****Program Objective:**

- creation of animation effects on objects

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

const double TWO_PI = 6.2831853;
GLsizei winWidth = 500, winHeight = 500;
GLuint regHex;
static GLfloat rotTheta = 0.0;

// Initial display window size.
// Define name for display list.
class scrPt {
public:
    GLint x, y;
};

static void init(void)
{
    scrPt hexVertex;
    GLdouble hexTheta;
    GLint k;
    glClearColor(1.0, 1.0, 1.0, 0.0);
    /* Set up a display list for a red regular hexagon.
    * Vertices for the hexagon are six equally spaced
    * points around the circumference of a circle.
    */
    regHex = glGenLists(1);
    glNewList(regHex, GL_COMPILE);
    glColor3f(1.0, 0.0, 0.0);
    glBegin(GL_POLYGON);
    for(k = 0; k < 6; k++) {
        hexTheta = TWO_PI * k / 6;
        hexVertex.x = 150 + 100 * cos(hexTheta);
        hexVertex.y = 150 + 100 * sin(hexTheta);
        glVertex2i(hexVertex.x, hexVertex.y);
    }
    glEnd();
    glEndList();
}

void displayHex(void)
{
    glClear(GL_COLOR_BUFFER_BIT);
    glPushMatrix();
    glRotatef(rotTheta, 0.0, 0.0, 1.0);
```

```
glCallList(regHex);
glPopMatrix( );
glutSwapBuffers( );
glFlush( );
}

void rotateHex(void)
{
    rotTheta += 3.0;
    if(rotTheta > 360.0)
        rotTheta -= 360.0;
    glutPostRedisplay( );
}

void winReshapeFcn(GLint newWidth, GLint newHeight)
{
    glViewport(0, 0, (GLsizei) newWidth, (GLsizei) newHeight);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity( );
    gluOrtho2D(-320.0, 320.0, -320.0, 320.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity( );
    glClear(GL_COLOR_BUFFER_BIT);
}

void mouseFcn(GLint button, GLint action, GLint x, GLint y)
{
    switch(button) {
        case GLUT_MIDDLE_BUTTON:
            // Start the rotation.
            if(action == GLUT_DOWN)
                glutIdleFunc(rotateHex);
            break;
        case GLUT_RIGHT_BUTTON:
            // Stop the rotation.
            if(action == GLUT_DOWN)
                glutIdleFunc(NULL);
            break;
        default:
            break;
    }
}

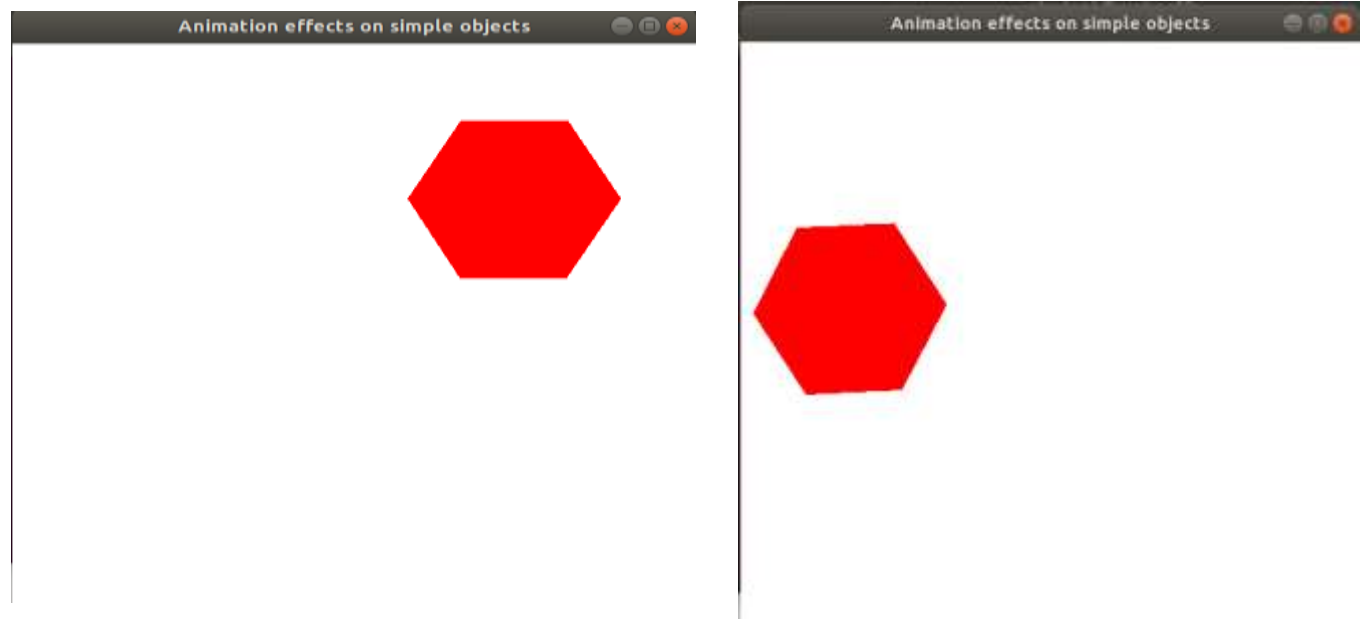
int main(int argc, char** argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB);
    glutInitWindowPosition(150, 150);
    glutInitWindowSize(winWidth, winHeight);
    glutCreateWindow("Animation Example");
    init( );
    glutDisplayFunc(displayHex);
    glutReshapeFunc(winReshapeFcn);
    glutMouseFunc(mouseFcn);
```

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

**RUN:**

```
g++ 6.cpp -lglut -lGL -lGLU
```

```
./a.out
```

**OUTPUT:****Program Outcome:**

- Applying animation effects on objects.

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

**Program Objective:**

- Implement CV2 and numpy packages

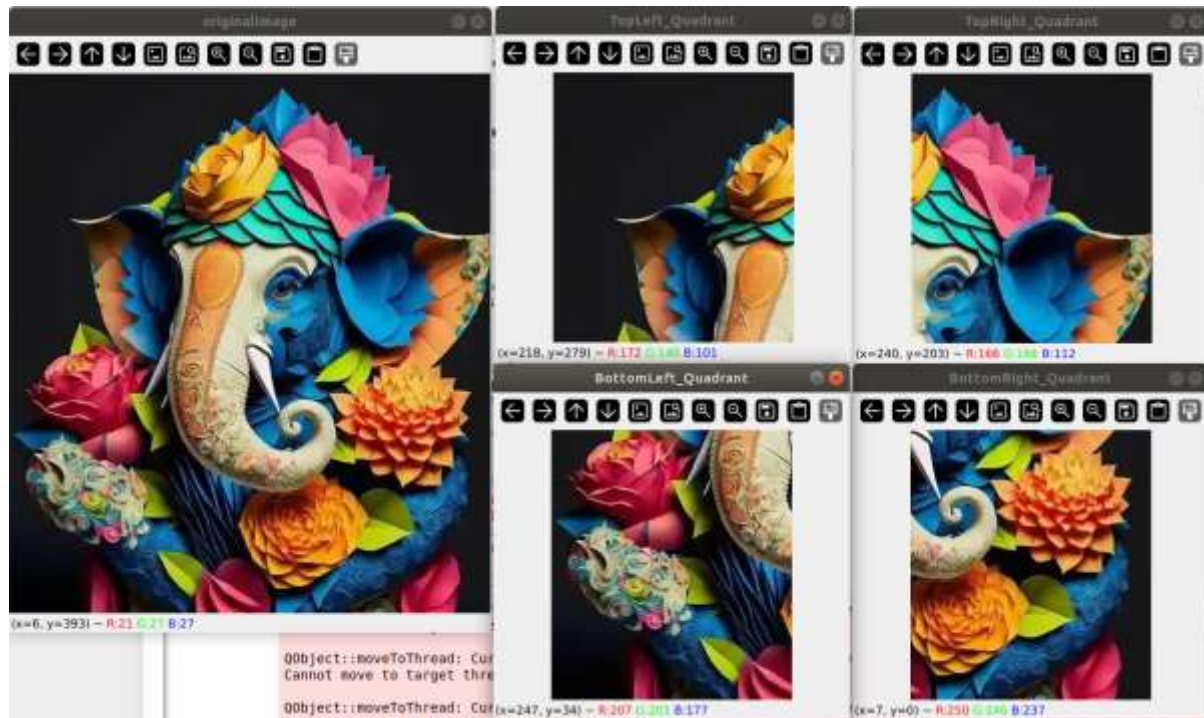
```
# Load the image
img = cv2.imread("atc.jpg") # Replace with the path to your image
original= img

# Get the height and width of the image
h, w, channels = img.shape
half_width = w//2
half_height = h//2

# Split the image into four quadrants
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:]

# Display the canvas
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 Outcome:**

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



**Program 8: Write a program to show rotation, scaling, and translation on an image****Program Objective:**

- Implement CV2 and numpy packages
- Implement rotation, scaling, and translation on an image

```
import cv2
import numpy as np

# Load the image
image_path = "atc.jpg" # Replace with the path to your image
img = cv2.imread(image_path)

# Get the image dimensions
height, width, _ = img.shape

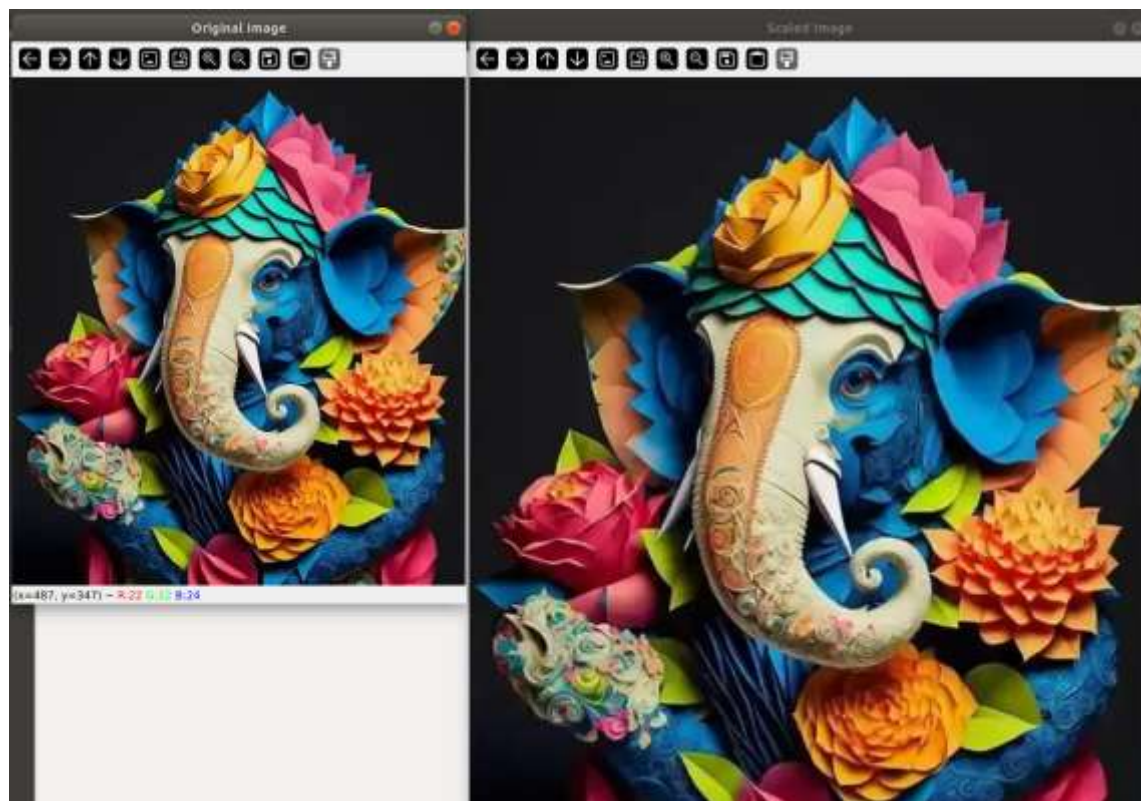
# Define the transformation matrices
rotation_matrix = cv2.getRotationMatrix2D((width/2, height/2), 45, 1)
# Rotate by 45 degrees
scaling_matrix = np.float32([[1.5, 0, 0], [0, 1.5, 0]]) # Scale by 1.5x
translation_matrix = np.float32([[1, 0, 100], [0, 1, 50]]) # Translate by (100, 50)

# Apply transformations
rotated_img = cv2.warpAffine(img, rotation_matrix, (width, height))
scaled_img = cv2.warpAffine(img, scaling_matrix, (int(width*1.5),
int(height*1.5)))
translated_img = cv2.warpAffine(img, translation_matrix, (width, height))

# Display the original and transformed images
cv2.imshow("Original Image", img)
cv2.imshow("Rotated Image", rotated_img)
cv2.imshow("Scaled Image", scaled_img)
cv2.imshow("Translated Image", translated_img)

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

## OUTPUT:

**Program Outcome:**

- Ability to show rotation, translation and scaling on an image.

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

**Program Objective:**

- Implement CV2 and numpy packages
- Implement edges, textures using filtering techniques.

```
import cv2
import numpy as np

# Load the image
image_path = "atc.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:****Program Outcome:**

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

**Program 10: Write a program to blur and smoothing an image.****Program Objective:**

- Implement CV2 and numpy packages
- Implement blur and smoothing an image

```
import cv2

# Load the image
image = cv2.imread('atc.jpg')

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

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

# Bilateral Filter
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 Outcome:**

- Able to show blur and show smoothing on an image
- Image smoothing and blurring are common preprocessing techniques used in image processing and computer vision to reduce noise and details in an image. OpenCV provides several functions for applying various types of blurring and smoothing operations.

**Program 11: Write a program to contour an image.****Program Objective:**

- Implement CV2 and numpy packages
- Implement contour an image.

```
import cv2
import numpy as np

# Load the image
image = cv2.imread('atc.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)

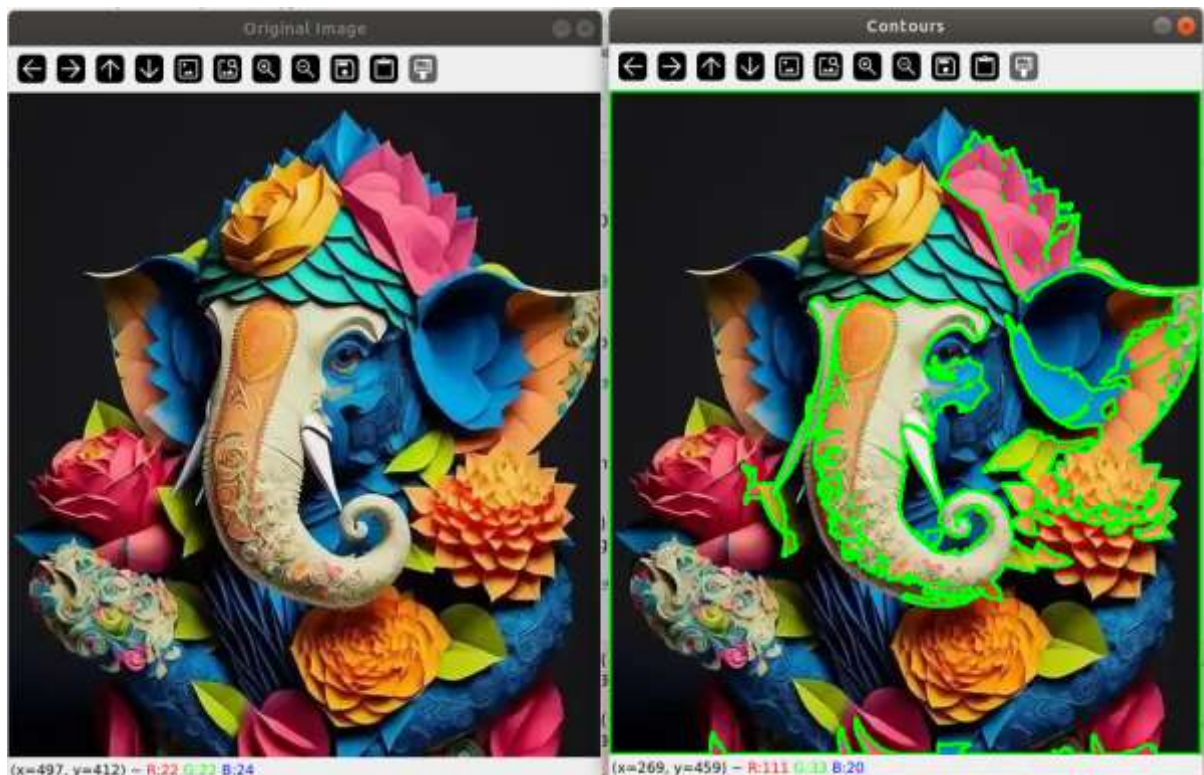
# Create a copy of the original image to draw contours on
contour_image = image.copy()

# 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()
```



**OUTPUT:****Program Outcome:**

- Able to contour an image
- Contours can be explained simply as a curve joining all the continuous points (along the boundary), having same color or intensity. The contours are a useful tool for shape analysis and object detection and recognition. For better accuracy, use binary images.

**Program 12: Write a program to detect a face/s in an image.****Program Objective:**

- Implement CV2 and numpy packages
- Implement 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('face.jpg')

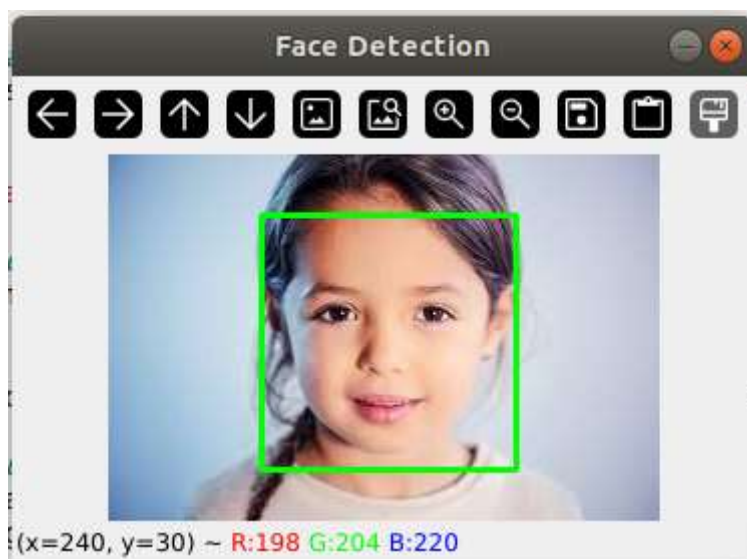
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# 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:****Program Outcome:**

- Able to detect a face in any given image
- Step 1: Import the OpenCV Package.
- Step 2: Read the Image.
- Step 3: Convert the Image to Grayscale.
- Step 4: Load the Classifier.
- Step 5: Perform the Face Detection.
- Step 6: Drawing a Bounding Box.

---

## 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 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 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 geometric transformation techniques.

CO 3: Demonstrate the ability to design and develop input interactive techniques.

CO 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 down 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 8<sup>th</sup> 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

## 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. 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.

### 6. 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.

### 7. 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/>

### 8. 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.

### 9. How does the camera work in OpenGL?

As far as OpenGL is concerned, there is no camera. More specifically, 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.

---



**10.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.

**11.What does the .gl or .GL le format have to do with OpenGL?**

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

**12.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.

**13.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.

**14.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.

**15.What is image processing in Python?**

Python enables image processing through libraries like OpenCV, PIL, and scikit-image for diverse applications.

**16.Which is the famous image processing Python?**

OpenCV is a popular and powerful image processing library widely used for computer vision applications.

**17.List some application of OpenCV?**

Some of the applications of OpenCV are object detection, face recognition, medical diagnosis, etc.

**18.Which method of opencv is used to read image?**

The `cv2.imread()` method of the `Imgcodecs` class is used in OpenCV to read an image.

**19.How many types of image filters available in OpenCV ?**

There are two types of image filters available in OpenCV: 1) Linear filter 2) Non-linear filter