Visvesvaraya Technological University
Belgaum, Karnataka-590 018

A Computer Graphics mini Project report
On

"Basketball Court 3D Viewing"

submitted in partial fulfillment of the requirement for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE & ENGINEERING

Submitted by

SANTOSH KUMAR PAITAL	**1HK20CS143**
SHREY VERMA	**1HK20CS152**

Under the Guidance of

Prof. Preetha M
Assistant Professor
Department of CSE, HKBKCE

Department of Computer Science & Engineering
HKBK College of Engineering
No.22/1, Opp., Manyata Tech Park Rd, Nagavara, Bengaluru, Karnataka 560045.
Approved by AICTE & Affiliated by VTU

2022-23

HKBK College of Engineering

No.22/1, Opp., Manyata Tech Park Rd, Nagavara, Bengaluru, Karnataka 560045.
Approved by AICTE & Affiliated by VTU

Department of Computer Science and Engineering

CERTIFICATE

Certified that the Computer Graphics Mini-Project (18CSL67) entitled **"Basketball Court 3D Viewing",** carried out by **SANTOSH KUMAR PAITAL (1HK20CS143) and SHREY VERMA (1HK20CS152)** is a bonafide student of **HKBK College of Engineering**, in partial fulfillment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** of the Visvesvaraya Technological University, Belgaum during the year **2022 – 23**. It is certified that all corrections/suggestions indicated for **Computer Graphics Mini-Project (18CSL67)** have been incorporated in the Report deposited in the departmental library. This report has been approved as it satisfies the academic requirements in respect of **Computer Graphics Mini-Project(18CSL67)** prescribed for the said Bachelor of Engineering.

Prof. Preetha M	**DR. Smitha Kurian**	**DR. Tabassum Ara**
Guide	HOD - CSE	Principal – HKBKCE

External Viva

Name of the examiners	**Signature with date**
1.____________________	__________________
2.____________________	__________________

ACKNOWLEDGEMENT

I would like to express my regards and acknowledgement to all who helped me in completing this Computer Graphics Mini-Project successfully

First of all I would take this opportunity to express my heartful gratitude to the personalities of HKBK college of engineering, **Mr. C M Ibrahim**, **Chairman, HKBKGI** and **Mr. Faiz Mohammed**, **Director, HKBKGI** for providing facilities throughout the course.

We would like to express our thanks to the Principal **DR. Tabassum Ara**, for her encouragement thatmotivated us for the successful completion of Project Work.

We wish to express our gratitude to **DR. Smitha Kurian.**, Professor and Head of Department of Computer Science & Engineering for providing such a healthy environment for the successful completion of Project work.

We express my heartfelt appreciation and gratitude to my Guide, **Prof. Preetha M**, Professor of Computer Science and Engineering, HKBK College of Engineering, Bangalore, for his/her intellectually- motivating support and invaluable encouragement during my Computer Graphics Mini-Project.

We would also like to thank all other teaching and technical staffs of Department of Computer Scienceand Engineering, who have directly or indirectly helped us in the completion of this Project Work. And lastly,we would hereby acknowledge and thank our parents who have been a source of inspiration and also instrumental in the successful completion of this project.

SANTOSH KUMAR PAITAL (1HK20CS143)

SHREY VERMA (1HK20CS152)

ABSTRACT

Main aim of this Mini Project is to illustrate the concepts of 3D Viewing of Basketball Court in OpenGL. This mini project aims to develop a computer graphics application using OpenGL to create a 3D view of a Basketball court with a ball. The project focuses on rendering a realistic representation of a Basketball court and simulating the movement of a 3D ball within the court. The implementation involves the use of OpenGL's graphics pipeline, geometric transformations, shading, and texturing techniques to achieve the desired visual effects. We have used input devices like mouse and key board to interact with program. To differentiate between objects we have used different colors for different objects. The project begins with the setup of the OpenGL development environment, ensuring the availability of the necessary libraries and tools. A window is created, and the graphics context is initialized for rendering the 3D scene. Camera controls are implemented to allow the user to navigate and view the 3D scene from different perspectives. Transformations are applied to move, rotate, and zoom the camera, giving the user a sense of control and interactivity. Lighting effects play a crucial role in enhancing the visual quality of the scene. Basic lighting models, including ambient, diffuse, and specular lighting, are applied to illuminate the football court. These lighting techniques add depth and realism to the scene, highlighting the contours of the court and the ball.

TABLE OF CONTENTS

LIST OF FIGURES:

LIST OF SNAPSHOTS:

CHAPTER 1:

INTRODUCTION

INTRODUCTION

1.1 Computer graphics:

- Graphics provides one of the most natural means of communicating with a computer, since our highly developed 2D Or 3D pattern-recognition abilities allow us to perceive and process pictorial data rapidly.
- Computers have become a powerful medium for the rapid and economical production of pictures.
- There is virtually no area in which graphical displays cannot be used to some advantage.
- Graphics provide a so natural means of communicating with the computer that they have become widespread.
- Interactive graphics is the most important means of producing pictures since the invention of photography and television .
- We can make pictures of not only the real world objects but also of abstract objects such as mathematical surfaces on 4D and of data that have no inherent geometry.
- A computer graphics system is a computer system with all the components of the general purpose computer system. There are five major elements in system: input devices, processor, memory, frame buffer, output devices.

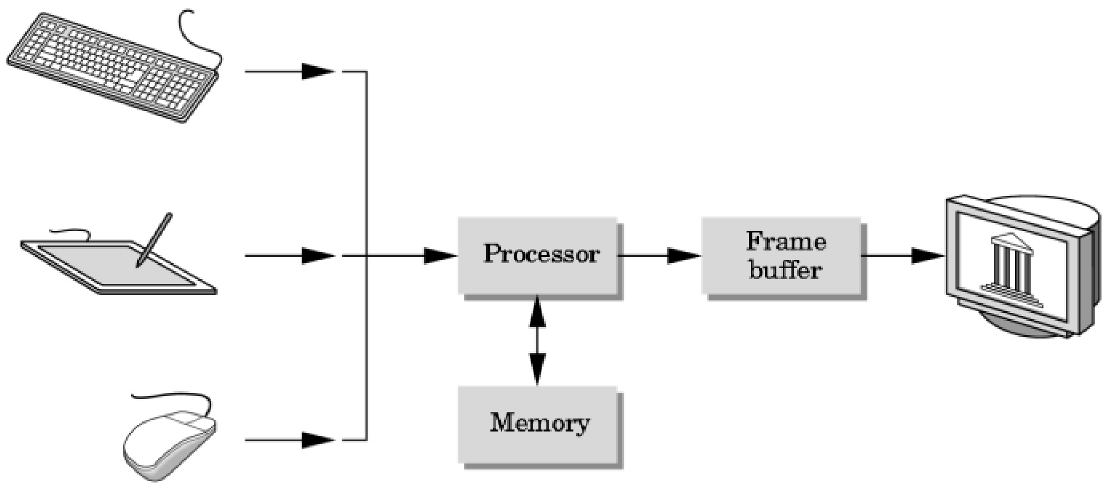

Fig 1.1 Components Of Computer Graphics System

1.2 OpenGL Technology

OpenGL is the premier environment for developing portable, interactive 2D and 3D graphics applications.
Since its introduction in 1992, OpenGL has become the industry's most widely used and supported 2D and 3D graphics application programming interface (API), bringing thousands of applications to a wide variety of computer platforms.

OpenGL fosters innovation and speeds application development by incorporating a broad set of rendering, texture mapping, special effects, and other powerful visualization functions. Developers can leverage the power of OpenGL across all popular desktop and workstation platforms, ensuring wide application deployment.

OpenGL Available Everywhere: Supported on all UNIX® workstations, and shipped standard with every Windows 95/98/2000/NT and MacOS PC, no other graphics API operates on a wider range of hardware platforms and software environments.

OpenGL runs on every major operating system including Mac OS, OS/2, UNIX, Windows 95/98, Windows 2000, Windows NT, Linux, OPENStep, and BeOS; it also works with every major windowing system, including Win32, MacOS, Presentation Manager, and X-Window System. OpenGL is callable from Ada, C, C++, Fortran, Python, Perl and Java and offers complete independence from network protocols and topologies.

The **OpenGL interface**, Our application will be designed to access OpenGL directly through functions in three libraries namely: gl,glu,glut.

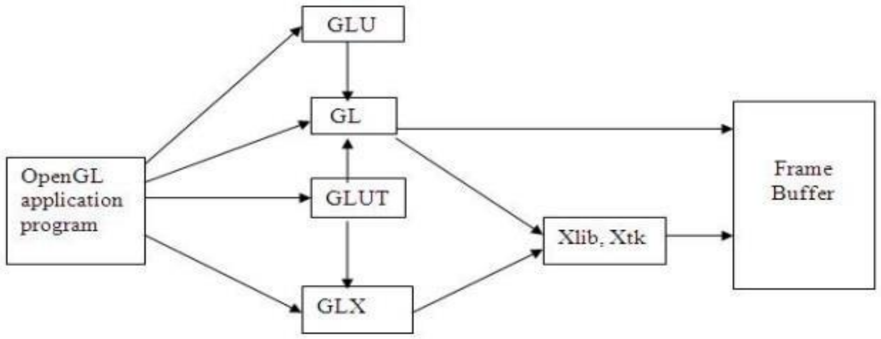

Fig 1.2 Open Gl Block Diagram

CHAPTER 2:
LITERATURE SURVEY

LITERATURE SURVEY

CG (Computer graphics) started with the display of data on hardcopy plotters and cathode ray tube screens soonafter the introduction of computer themselves. It includes the creation, storage, and manipulation of models and images of objects. These models include physical, mathematical, engineering, architectural, and even conceptualor abstract structures, natural phenomena, and so on. Computer Graphics today is largely interactive- the user controls the contents, structure, and appearance of objects and their displayed images by using input devices, suchas keyboard, mouse or touch sensitive panel on the screen. Bitmap graphics is used for user-computer interaction.A Bitmap is an ones and zeros representation of points (pixels, short for 'picture elements') on the screen. Bitmap graphics provide easy-to-use and inexpensive graphics-based applications.

The concept of 'desktop' is a popular metaphor for organizing screen space. By means of a window manager,the user can create, position, and resize rectangular screen areas, called windows, that acted as virtual graphics terminals, each running an application. This allowed users to switch among multiple activities just by pointing atthe desired window, typically with the mouse. Graphics provides one of the most natural means of communicatingwith the computer, since our highly developed 2D and 3D pattern – recognition abilities allow us to perceive andprocess pictorial data rapidly and efficiently. In many designs, implementation, and construction processes, the information pictures can give is virtually indispensable.

2.1 Interactive and Non-Interactive graphics:

Computer graphics is the creation and manipulation of pictures with the aid of computers. It is divided intotwo broad classes:

- ❖ Non-Interactive Graphics.
- ❖ Interactive Graphics.

2.1.1 Non-Interactive graphics:

This is a type of graphics where observer has no control over the pictures produced on the screen. It is alsocalled as Passive graphics

2.1.2 Interactive Graphics:

This is the type of computer graphics in which the user can control the pictures produced. It involves two- way communication between user and computer. The computer upon receiving signal from the input device can modify the displayed picture appropriately. To the user it appears that the picture changes instantaneously in response to his commands. The following fig. shows the basic graphics system:

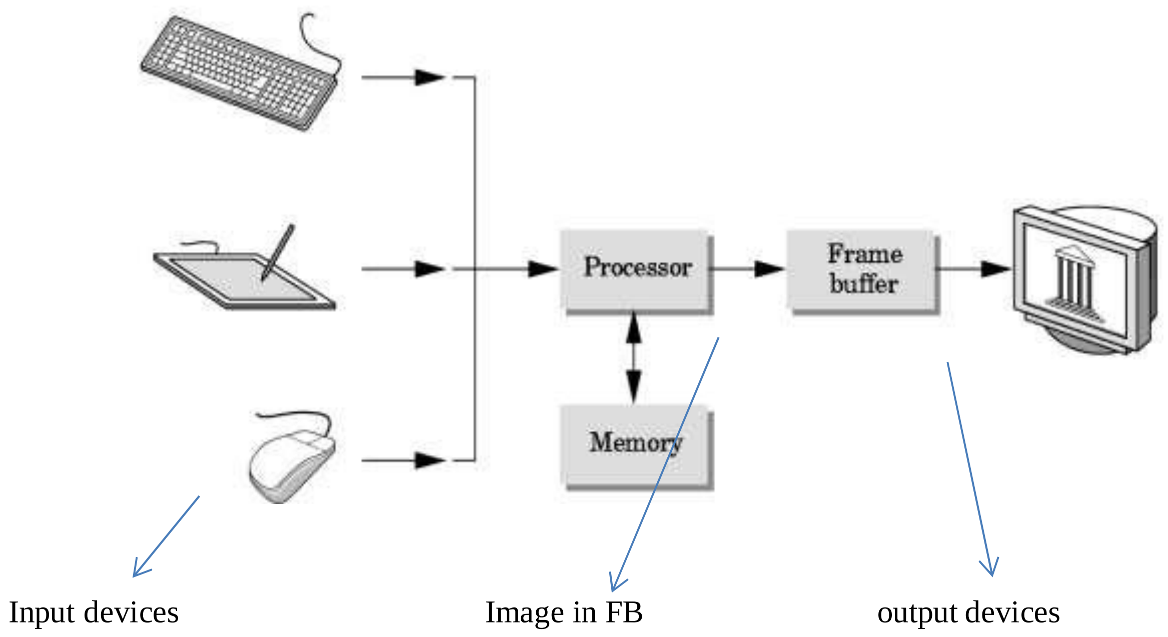

Fig 2.1: Basic Graphics System.

2.2 About OpenGL:

OpenGL is an open specification for an applications program interface for defining 2D and 3D objects. Thespecification is cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. It renders 3D objects to the screen, providing the same set of instructions on different computers and graphics adapters. Thus it allows us to write an application that can create the same effects in any operating systemusing any OpenGL-adhering graphics adapter.

In Computer graphics, a 3-dimensional primitive can be anything from a single point to an ‘n’ sided polygon. From the software standpoint, primitives utilize the basic 3-dimensional rasterization algorithms such as Bresenham's line drawing algorithm, polygon scan line fill, texture mapping and so forth.

OpenGL is a low-level, procedural API, requiring the programmer to dictate the exact steps required to render a scene. OpenGL's low-level design requires programmers to have a good knowledge of the graphics pipeline, but also gives a certain amount of freedom to implement novel rendering algorithms.

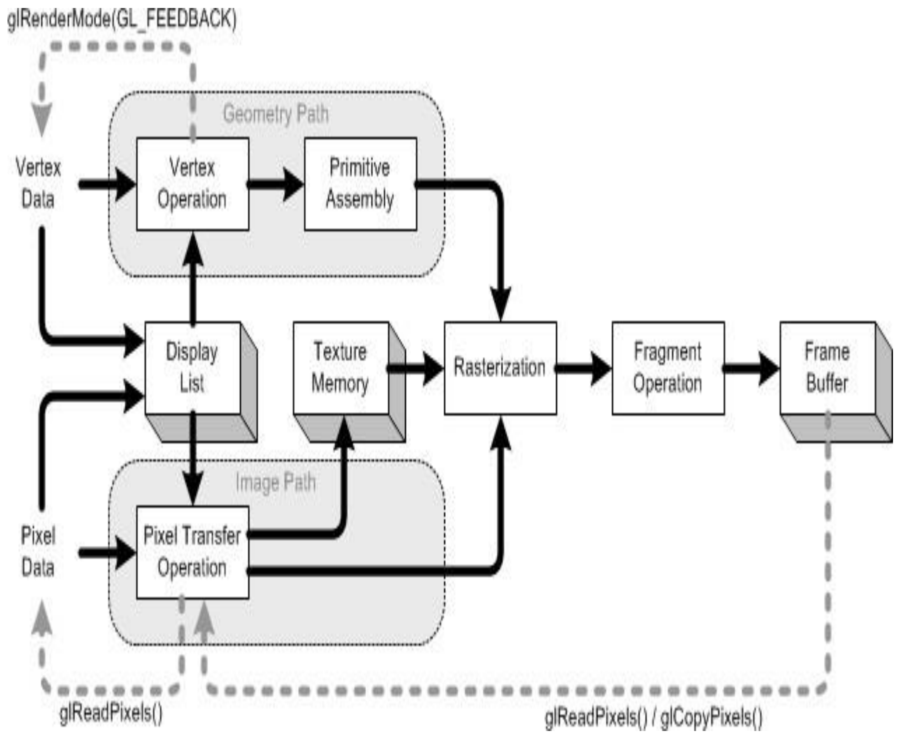

Fig 2.2: The OpenGL rendering pipeline.

2.3 Advantages of OpenGL

- OpenGL is an industry-standard guided by an independent consortium, the OpenGL Architecture Review Board oversees the OpenGL specification. It makes OpenGL the only truly open, vendor-neutral, and multi-platform graphics standard.
- OpenGL offers scalability with applications that can execute on systems running the gamut from consumer electronics to PCs, workstations, and supercomputers. Consequently, applications can scale to any class of machine that the developer chooses to target.

CHAPTER 3:
SYTEM REQUIREMENTS SPECIFICATIONS

SYSTEM REQUIREMENTS SPECIFICATION

A software requirement definition is an abstract description of the services which the system should provide, and the constraints under which the system must operate. It should only specify the external behaviour of the system.

3.1 Functional Requirements

Functional Requirements define the internal working of the software. The following conditions must be taken care of: The ability to perform correct operations when corresponding keys are pressed.

3.2 Non-functional Requirements

Non-functional requirements are requirements which specify criteria that can be used to judge the operation of the system, rather than specific behaviours. This should be contrasted with functional requirements that specify specific behaviour or functions. Typical non-functional requirements are reliability and scalability. Non- functional requirements are “constraints”, “quality attributes” and “quality of service requirements”.

Types of non-functional requirements

Volume: Volume of a system (the total space occupied) varies depending on how the component assemblies are arranged and connected.

Reliability: System reliability depends on component reliability but unexpected interactions can cause new types of failure and therefore affect the reliability of the system.

Security: The security of the system (it’s ability to resist attacks) is a complex property that cannot be easily measured. Attacks maybe devised that were not anticipated by the system designers and may use default built-in safeguards.

Repairability: This property reflects how easy it is to fix a problem with the system once it has been discovered. It depends on being able to diagnose the problem, access the components that are faulty and modify or replace these components.

Usability: This property reflects how easy it is to use the system. It depends on the technical system components, its operators and its operating environment.

3.3 Software Requirements:

Operating System	: Windows
IDE	: VS Express
Coding Language	: C++

3.4 Hardware Requirements

System	: Intel® Core™ i3 – 6006U CPU @ 2.00GHz
Hard Disk	: 30 GB or above
Monitor	: 15 VGA color
RAM	: 256 MB or above

3.5 Introduction to Environment

OpenGL is a software interface to graphics hardware. This interface consists of about 120 distinct commands, which you use to specify the objects and operations needed to produce interactive three - dimensional applications.

OpenGL is designed to work efficiently even if the computer that displays the graphics you create isn't the computer that runs your graphics program. This might be the case if you work in a networked computer environment where many computers are connected to one another by wires capable of carrying digital data. In this situation, the computer on programs can work across a network even if the client and server are different kinds of computers. If an OpenGL program isn't running which your program runs and issues OpenGL drawing commands is called the client, and the computer that receives those commands and performs the drawing is called the server. The format for transmitting OpenGL commands (called the protocol) from the client to the server is always the same, so OpenGL across a network, then there's only one computer, and it is both the client and the server.

CHAPTER 4:
ALGORITHM DESIGN AND ANALYSIS

ALGORITHM DESIGN AND ANALYSIS

4.1 Pseudo Code:

```
#include "stdafx.h"
#include<stdio.h>
#include<glut.h>
#include<math.h>
#include<time.h>
int firsttime = 0;
float x = 0 , y = 0, z = 0.0;
GLfloat oldy = 0, oldz = 0, tempz, dy = 0, dz = 0;int triggered = 0;
GLfloat courtVertices[][3] = {
        {-2.5, -1.0, -4.7}, {2.5, -1.0, -4.7},{2.5, -1.0, 4.7}, {-2.5, -1.0, 4.7}};
GLfloat firstPoleVertices[][3] = {
        {-0.1, -1.0, -5.2}, {0.1, -1.0, -5.2},{-0.1, -1.0, -5.0}, {0.1, -1.0, -5.0},
        {-0.1, 0.5, -5.2}, {0.1, 0.5, -5.2},{-0.1, 0.4, -5.0}, {0.1, 0.4, -5.0},
        {-0.1, 1.3, -4.4 }, { 0.1, 1.3 , -4.4},{-0.1, 1.7, -4.4 }, { 0.1, 1.7, -4.4 }};
GLfloat secondPoleVertices[][3] = {
        {-0.1, -1.0, 5.2}, {0.1, -1.0, 5.2},{-0.1, -1.0, 5.0}, {0.1, -1.0, 5.0},
        {-0.1, 0.5, 5.2}, {0.1, 0.5, 5.2},{-0.1, 0.4, 5.0}, {0.1, 0.4, 5.0},
        {-0.1, 1.3, 4.4 }, { 0.1, 1.3 , 4.4},{-0.1, 1.7, 4.4 }, { 0.1, 1.7, 4.4}};
GLfloat firstBoardVertices[][3] = {
        {-0.5, 1.0, -4.3}, { 0.5, 1.0, -4.3},{-0.5, 1.0, -4.4}, { 0.5, 1.0, -4.4},
        {-0.5, 2.0, -4.3}, { 0.5, 2.0, -4.3},{-0.5, 2.0, -4.4}, { 0.5, 2.0, -4.4},};
GLfloat secondBoardVertices[][3] = {
        {-0.5, 1.0, 4.3}, { 0.5, 1.0, 4.3},{-0.5, 1.0, 4.4}, { 0.5, 1.0, 4.4},
        {-0.5, 2.0, 4.3}, { 0.5, 2.0, 4.3},{-0.5, 2.0, 4.4}, { 0.5, 2.0, 4.4},};
GLfloat baseVertices[][3] = {
        {-3.0, -1.0001, -5.2}, {3.0, -1.0001, -5.2},{-3.0, -1.0001, 5.2},
  {3.0, -1.0001, 5.2},{-3.0, -1.5, -5.2}, {3.0, -1.5, -5.2},
        {-3.0, -1.5, 5.2}, {3.0, -1.5, 5.2},};
void poles(int a, int b, int c, int d)
{
        glBegin(GL_POLYGON);
        glColor3f(55.0 / 255.0, 51.0/ 255.0, 49.0/ 255.0);
        glVertex3fv(firstPoleVertices[a]);
        glVertex3fv(firstPoleVertices[b]);
        glVertex3fv(firstPoleVertices[c]);
        glVertex3fv(firstPoleVertices[d]);
        glEnd();
        glBegin(GL_LINE_LOOP);
        glColor3f(43.0 / 255.0, 39.0/ 255.0, 37.0/ 255.0);
        glVertex3fv(firstPoleVertices[a]);glVertex3fv(firstPoleVertices[b]);
        glVertex3fv(firstPoleVertices[c]);glVertex3fv(firstPoleVertices[d]);
        glEnd();
        glBegin(GL_POLYGON);     glColor3f(0.5, 0.5, 0.5);
```

```
            glVertex3fv(secondBoardVertices[a]);glVertex3fv(secondBoardVertices[b]);
            glVertex3fv(secondBoardVertices[c]);glVertex3fv(secondBoardVertices[d]);
            glEnd();
            glBegin(GL_LINE_LOOP);
            glColor3f(86.0/255.0, 86.0/255.0, 86.0/255.0);
            glVertex3fv(secondBoardVertices[a]);glVertex3fv(secondBoardVertices[b]);
            glVertex3fv(secondBoardVertices[c]);glVertex3fv(secondBoardVertices[d]);
            glEnd();
    }
    void base(int a, int b, int c, int d)
    {
            glBegin(GL_POLYGON);
            glColor3f(0.4, 0.6, 0.3);
            glVertex3fv(baseVertices[a]);glVertex3fv(baseVertices[b]);
            glVertex3fv(baseVertices[c]);glVertex3fv(baseVertices[d]);
            glEnd();

            glBegin(GL_LINE_LOOP);
            glColor3f(165.0/255.0, 0.0/255.0, 3.0/255.0);
            glVertex3fv(baseVertices[a]);glVertex3fv(baseVertices[b]);
            glVertex3fv(baseVertices[c]);glVertex3fv(baseVertices[d]);
            glEnd();
    }
    void polygon(int a, int b, int c, int d)
    {
            base(0, 1, 3, 2);base(4, 5, 7, 6);base(2, 3, 7, 6);
            base(0, 1, 5, 4);base(0, 2, 6, 4);base(1, 3, 7, 5);
            glBegin(GL_POLYGON);
            glColor3f(0.8, 0.4, 1.0);glVertex3fv(courtVertices[a]);
            glColor3f(0.8, 0.4, 1.0);glVertex3fv(courtVertices[b]);
            glColor3f(0.8, 0.4, 1.0);glVertex3fv(courtVertices[c]);
            glColor3f(0.8, 0.4, 1.0);glVertex3fv(courtVertices[d]);
            glEnd();
            poles(0, 1, 3, 2);poles(4, 5, 7, 6);poles(2, 3, 7, 6);
            poles(4, 5, 1, 0);poles(3, 1, 5, 7);poles(0, 2, 6, 4);
            poles(6, 7, 9, 8);poles(4, 5, 11, 10);
      poles(6, 4, 10 , 8);poles(7, 5, 11, 9);
      board(0, 1, 3, 2);board(4, 5, 7, 6);board(0, 2, 6, 4);
      board(1, 3, 7, 5);board(0, 1, 5, 4);board(3, 7, 6, 2);
            lines(-2.5, -0.05, 0.05, 2.5);lines(-2.50, 4.7, -4.7, -2.55);
            lines(2.50, 4.7, -4.7, 2.55);lines(-2.55, 4.70, 4.75, 2.55);
            lines(-2.55, -4.70, -4.75, 2.55);lines(-2.2, 4.7, 4.05, -2.27);
            lines( 2.2, 4.7, 4.05, 2.27);lines(-2.2, -4.7, -4.05, -2.27);
            lines( 2.2, -4.7, -4.05, 2.27);lines(-0.6, 4.7, 2.8, -0.64);
  lines(0.6, 4.7, 2.8, 0.64);lines(-0.6, -4.7, -2.8, -0.64);
  lines(0.6, -4.7, -2.8, 0.64);lines(-0.6, 2.8, 2.84, 0.6);
  lines(-0.6, -2.8, -2.84, 0.6);
  onBoardLines(-0.15, 1.2, 1.24, 0.15); //bottom
 {
            int i;
            glColor3f(1.0, 0.1, 0.8);
            glPointSize(3.0);
            glBegin(GL_POINTS);
            for(i = 0; i < 1000; i++)
            {
                    glVertex3f((r * cos(1*3.14159 * i/1000.0)), -0.9999,
     4.05 - (r * sin(1*3.14159 * i/1000.0)));
                    glVertex3f((r * cos(1*3.14159 * i/1000.0)), -0.9999,
    -4.05 + (r * sin(1*3.14159 * i/1000.0)));     }
            glEnd();
    }
    void ball(){
            if(firsttime){
                    glTranslatef(0.0, 1.2, -1.5);
            }
            else{
                    glTranslatef(0.0, 0.8, -2.8);
            }
            glColor3f(0.81176, 0.3254, 0.0);glutSolidSphere(0.15, 1000, 20);
    }
    void net(int poleChooser){
            float r = 0.15;
            int i;
            float poleDecider = 0;
            GLfloat topVertices[10][200];
            GLfloat middleVertices[10][200];
            GLfloat bottomVertices[10][200];
            if(poleChooser == 1){
                    poleDecider = 4.3 - 0.19;
            }
            else{
                    poleDecider = -4.3 + 0.19;
            }
            glColor3f(235.0/255.0, 63.0/255.0, 23.0/255.0);
            for(i = 0; i < 20; i++){
                    topVertices[0][i] = ((r) * cos(2 * 3.14159 * i/20.0));
                    topVertices[1][i] = (poleDecider + (r) * sin(2 * 3.14159 * i/20.0));
                    glBegin(GL_POINTS);
                    glVertex3f(topVertices[0][i], 1.2, topVertices[1][i]);
                    glEnd();
            }
            for(i = 0; i < 20; i++)
            {
              middleVertices[0][i] = ((r - 0.05)* cos(2 * 3.14159 * i/20.0));
              middleVertices[1][i] = ( poleDecider + (r - 0.05)* sin(2 * 3.14159 * i/20.0));
                    glBegin(GL_POINTS);
                    glVertex3f(middleVertices[0][i], 1.0, middleVertices[1][i]);
                    glEnd();
            }
            for(i = 0; i < 20; i++){
        bottomVertices[0][i] = ((r - 0.05)* cos(2 * 3.14159 * i/20.0));
            bottomVertices[1][i] = ( poleDecider + (r - 0.05)* sin(2 * 3.14159 * i/20.0));
                    glBegin(GL_POINTS);
                    glVertex3f(bottomVertices[0][i], 0.8, bottomVertices[1][i]);
                    glEnd();
            }
            for(i = 0; i < 20; i++){
                    //from top vertices to the middle vertices
                    glBegin(GL_LINES);
                    if(i == 19){
                            glVertex3f(topVertices[0][i], 1.2, topVertices[1][i]);
                            glVertex3f(middleVertices[0][0], 1.0, middleVertices[1][0]);
                    }
                    else{
                            glVertex3f(topVertices[0][i], 1.2, topVertices[1][i]);
                            glVertex3f(middleVertices[0][i + 1], 1.0, middleVertices[1][i+1]);
                    }
                    glEnd();
                    glBegin(GL_LINES);
                    if(i == 0){
                            glVertex3f(topVertices[0][i], 1.2, topVertices[1][i]);
                            glVertex3f(middleVertices[0][19], 1.0, middleVertices[1][19]);
                    }
                    else{
                            glVertex3f(topVertices[0][i], 1.2, topVertices[1][i]);
                            glVertex3f(middleVertices[0][i - 1], 1.0, middleVertices[1][i-1]);
                    }
                    glEnd();
                    glBegin(GL_LINES);
                    if(i == 19){
                            glVertex3f(middleVertices[0][i], 1.0, middleVertices[1][i]);
                            glVertex3f(bottomVertices[0][0], 0.8, bottomVertices[1][0]);
                    }
                    else{
                            glVertex3f(middleVertices[0][i], 1.0, middleVertices[1][i]);
                            glVertex3f(bottomVertices[0][i + 1], 0.8, bottomVertices[1][i+1]);
                    }
                    glEnd();
                    glBegin(GL_LINES);
                    if(i == 0){
                            glVertex3f(middleVertices[0][i], 1.0, middleVertices[1][i]);
                            glVertex3f(bottomVertices[0][19], 0.8, bottomVertices[1][19]);
                    }
                    else{
                            glVertex3f(middleVertices[0][i], 1.0, middleVertices[1][i]);
                            glVertex3f(bottomVertices[0][i - 1], 0.8, bottomVertices[1][i-1]);}
    void mouse(int btn, int state, int x, int y){
            if(btn == GLUT_LEFT_BUTTON && state == GLUT_DOWN) axis = 1;
            if(btn == GLUT_MIDDLE_BUTTON && state == GLUT_DOWN)axis = 0;
            if(btn == GLUT_RIGHT_BUTTON && state == GLUT_DOWN)axis = 2;
            theta[axis] += 2.0;
            if(theta[axis] > 360.0)
      theta[axis] -= 360.0;
            display();
    }
    void myReshape(int w, int h){
            glViewport(0, 0, w , h);
            glMatrixMode(GL_PROJECTION);
            glLoadIdentity();
            if(w <= h)
                    glFrustum(-2.0, 2.0, -2.0 * (GLfloat)h / (GLfloat)w , 2.0 * (GLfloat)h /
    (GLfloat)w, 2.0, 20.0);
            else
                    glFrustum(-2.0, 2.0, -2.0 * (GLfloat)w / (GLfloat)h , 2.0 * (GLfloat)w /
    (GLfloat)h,  2.0, 20.0);
            glMatrixMode(GL_MODELVIEW);
    }
    void keys(unsigned char key, int x, int y){
            if(key == 'x' && viewer[0] != -6) viewer[0] -= 1.0;
            if(key == 'X' && viewer[0] != 6) viewer[0] += 1.0;
            if(key == 'y' && viewer[1] != 0) viewer[1] -= 1.0;
            if(key == 'Y' && viewer[1] != 9) viewer[1] += 1.0;
            if(key == 'z'  && viewer[2] != 4) viewer[2] -= 1.0;
            if(key == 'Z'  && viewer[2] != 10) viewer[2] += 1.0;
            if(key == 's' || key == 'S')        {
                    triggered = 1;firsttime = 1;
            }
            display();
    }
    void main(int argc, char **argv){
            glutInit(&argc , argv);
            glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
            glutInitWindowSize(1000, 1000);
            glutCreateWindow("Colourable viewer");
            glutReshapeFunc(myReshape);
            glutDisplayFunc(display);
            glutMouseFunc(mouse);
            glutKeyboardFunc(keys);
            glEnable(GL_DEPTH_TEST);
            glutTimerFunc(1, update, 0);
            glClearColor(1.0, 1.0, 1.0, 0.0);
            glutMainLoop();
    }
```

4.2 Analysis:

- OpenGL Setup: Setting up the development environment with the necessary libraries and tools. Creating an OpenGL window and initializing the graphics context.
- Rendering the Basketball Court: Define the vertices for the basketball court, poles, boards, and lines using OpenGL coordinate system.
- Implement functions to draw the different components of the basketball court, such as poles, boards, lines, and the court itself. Apply appropriate colors and textures to the court, poles, and boards to enhance realism.
- Ball Model: Creating a 3D model of a football using geometric primitives like spheres.
- Applying material properties such as color and texture to the ball. Implementing physics-based motion to simulate the movement of the ball within the court.
- Creating the 3D Basketball: Define the geometry of the basketball using OpenGL primitives. Apply appropriate colors and textures to make the basketball visually appealing. Implement physics-based motion for the basketball to simulate realistic movement.
- User Interaction: Handle user input to navigate the 3D scene, allowing the user to move the camera around the court. Implement shooting mechanics to allow the user to interact with the basketball and shoot it towards the hoops.
- Camera Controls: Implementing camera controls to allow the user to navigate and view the 3D scene.
- Implementing transformations to move, rotate, and zoom the camera.
- Lighting Effects: Applying lighting techniques to illuminate the football court.
- Implementing basic lighting models such as ambient, diffuse, and specular lighting.

CHAPTER 5:

IMPLEMENTATION

IMPLEMENTATION

Implementation is an act or instance of implementing something that is, the process of making something active or effective. Implementation must follow any preliminary thinking in order for something to actually happen. In the context of information technology, implementation encompasses the process involved in getting new software or hardware to operate properly in its environment. This program is implemented using various OpenGL functions which are shown below.

4.1 Functions Used:

Various functions used in this program.

- **glutInit**() : interaction between the windowing system and OPENGL is initiated.
- **glutInitDisplayMode**() : used when double buffering and depth information is required
- **glutCreateWindow**() : opens the OPENGL window and displays the title at top of the window
- **glutInitWindowSize**() : specifies the size of the window
- **glutInitWindowPosition**() : specifies the position of the window in screen co-ordinates
- **glutKeyboardFunc**() : handles normal ascii symbols
- **glutSpecialFunc**() : handles special keyboard keys
- **glutReshapeFunc**() : sets up the callback function for reshaping the window
- **glutIdleFunc**() : this handles the processing of the background
- **glutDisplayFunc**() : this handles redrawing of the window
- **glutMainLoop**() : this starts the main loop, it never returns
- **glViewport**() : used to set up the viewport
- **glVertex3fv**() : used to set up the points or vertices in three dimensions
- **glColor3fv**() : used to render color to faces
- **glFlush**() : used to flush the pipeline
- **glutPostRedisplay**() : used to trigger an automatic redrawal of the object
- **glMatrixMode**() : used to set up the required mode of the matrix
- **glLoadIdentity**() : used to load or initialize to the identity matrix
- **glTranslatef**() : used to translate/move the rotation centre from one point to another in 3D
- **glRotatef**() : used to rotate an object through a specified rotation angle

4.2 User Defined Functions:

The user-defined functions that are used and can be identified as:

- **poles( ):** This function is responsible for drawing the poles of the basketball court. It uses these indices to draw the polygons and lines that make up the poles.
- **onBoardLines( ):** Similar to the `lines` function, this function is used to draw the lines on the basketball backboard. This function is used to draw the vertical lines on the backboard.
- **base( ):** This function is used to draw the base of the basketball court. It draws the polygons and lines that make up the base.
- **polygon( ):** This function is responsible for drawing the main polygonal structure of the basketball court It draws the polygons and lines that make up the court.
- **circle( ):** This function is used to draw a circle on the basketball court. It uses a `for` loop and `glVertex3f()` to calculate and draw the points that make up the circle.
- **Dcircle( ):** Similar to the `circle` function, this function is used to draw a semicircle on the basketball court. It uses a `for` loop and `glVertex3f()` to calculate and draw the points that make up the semicircle.
- **semicircle( ):** This function is used to draw the three-point semicircle on the basketball court. It uses a `for` loop and `glVertex3f()` to calculate and draw the points that make up the semicircle.
- **ball( ):** This function is responsible for drawing the basketball. It uses `glTranslatef()` to position the ball, and `glutSolidSphere()` to draw a solid sphere with a specified radius and number of subdivisions.
- **net( ):** This function is used to draw the net on the basketball hoop. It takes a single integer parameter `poleChooser`. It uses a combination of vertices, lines, and loops to draw the net in a rhombus pattern.
- **display( ):** This function is the display callback function that is called by GLUT to redraw the scene. It sets up the view using `gluLookAt()` and applies rotations based on the `theta` values.
- **update( ):** This function is a timer callback function to update the animation of the ball. It updates the position of the ball and triggers the rendering of the scene using `glutPostRedisplay()`.
- **myReshape( ):** This function is the reshape callback function that is called when the window resized.

CHAPTER 6:
RESULTS AND DISCUSSION

RESULTS AND DISCUSSIONS

6.1 Discussions:

Discussing the Basketball Court 3D Viewing in OpenGL with some key points:

- The implementation of the project successfully creates a 3D view of a basketball court using OpenGL.
- The rendered scene includes a basketball court with accurately positioned poles, boards, lines, and a 3D basketball.
- The user can navigate the 3D scene and interact with the basketball by shooting it towards the hoops.
- The physics-based motion of the basketball provides a realistic experience.

Overall, implementation involves the use of OpenGL's graphics pipeline, geometric transformations, shading, and texturing techniques to achieve the desired visual effects.

6.2 Snapshots:

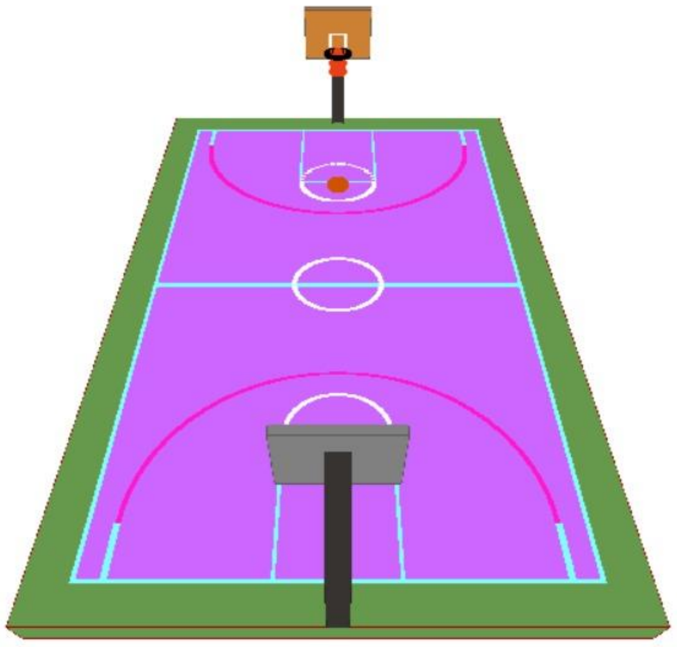

Fig 5.1 Output(1) – Top view of the Basketball Court

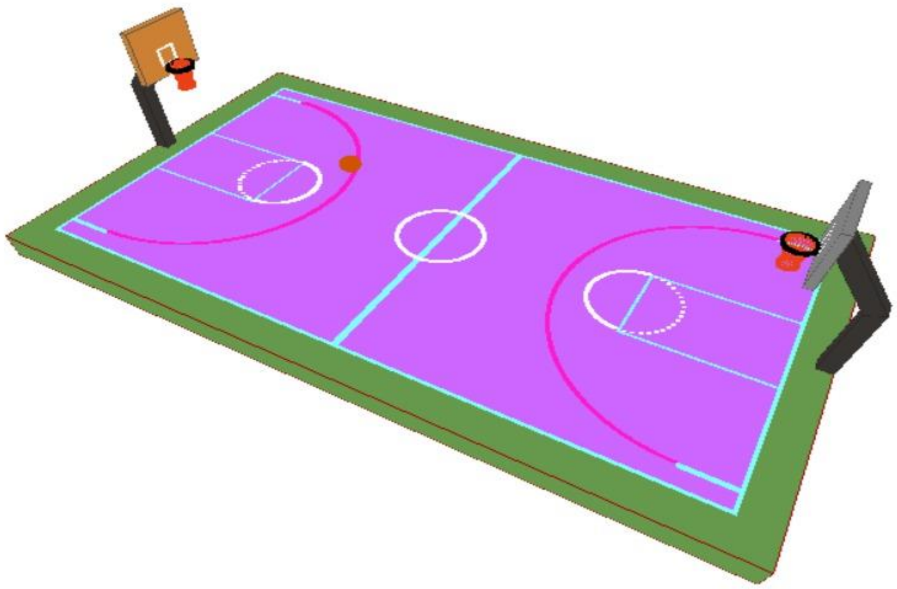

Fig 5.2 Output(2) – Rotation of the Basketball court along the axis

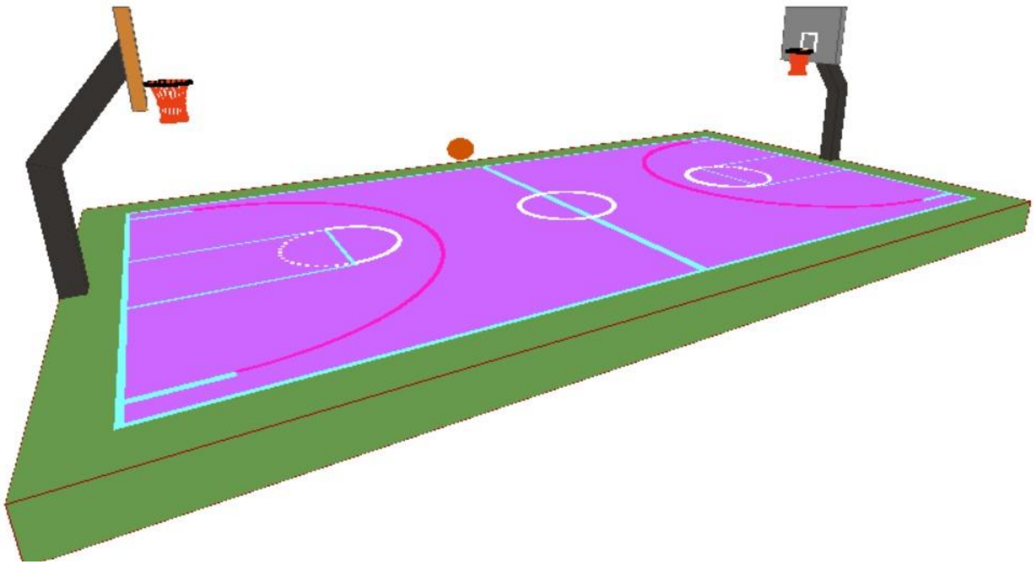

Fig 5.3 Output(3) - y-axis Rotation of the Basketball Court

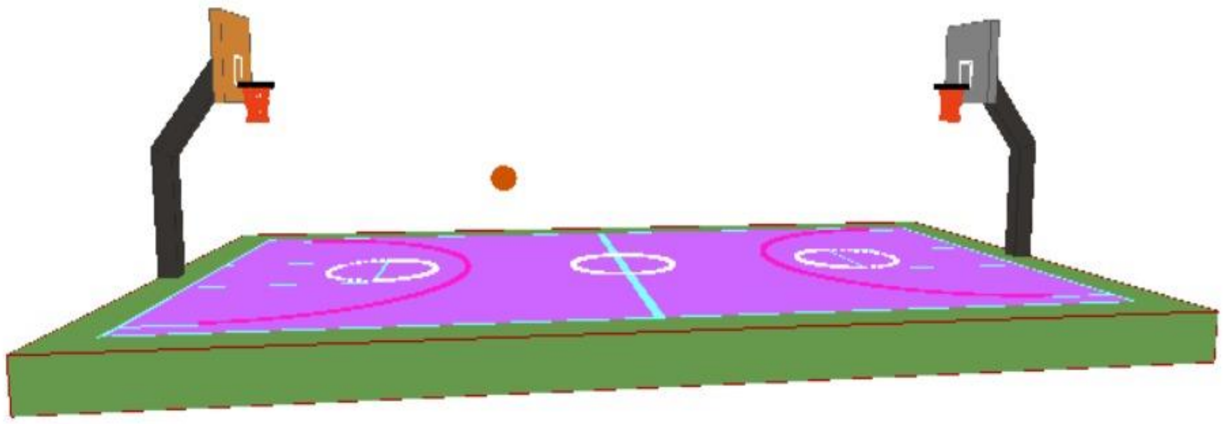

Fig 5.4 Output(4) – x-axis Rotation of the Basketball Court

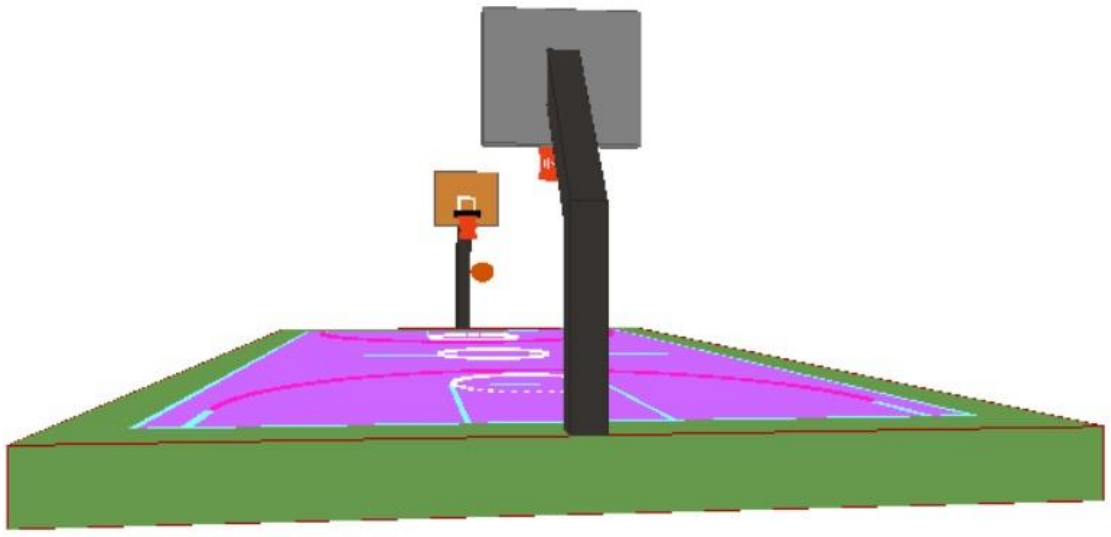

Fig 5.5 Output(5) – Rotation of the Basketball Court along negative x-axis

CONCLUSIONS AND FUTURE SCOPE

CONCLUSIONS AND FUTURE SCOPE

In this mini project, we successfully developed a computer graphics application using OpenGL to create a 3D view of a Basketball court with a ball. The implemented 3D view of a basketball court provides a realistic environment for users to explore and interact with. The project can be further extended by adding more features, such as player models, realistic lighting, and sound effects.

General Constraints:

- The code relies on the GLUT library, which is specific to the OpenGLenvironment.
- The code is designed for a console application, limiting its graphical capabilities.
- The code may not be portable across different operating systems due to itsreliance on Windows-specific headers.

Future Enhancements:

Although the current implementation is functional, there are several areas for potential improvement and future enhancements. Some possible avenues for further development include:

- Implementing more advanced lighting models and techniques to enhance the realism of the scene.
- Adding additional objects and elements to the Basketball court, such as players, Score Board, and spectators.
- Incorporating physics-based simulations for ball interactions, such as collisions and rebounds.
- Implementing user interactions, such as ball tackel and player movements, to create a more interactive experience.

BIBLIOGRAPHY:

The Resources used to design and implement the Project Successively, theFollowing are some of the resources:-

TEXTBOOKS:-

- INTERACTIVE COMPUTER GRAPHICS A TOP-DOWN APPROACH
  -By Edward Angel.
- COMPUTER GRAPHICS,PRINCIPLES & PRACTICES
  - Foley van dam
  - Feiner hughes

LINKS / WEB REFERENCES:

- http://jerome.jouvie.free.fr/OpenGl/Lessons/Lesson3.php
- The website providing OpenGL lessons and tutorials, including Lesson 3, created by Jerome Jouvie.
- http://opengl.org
- The official website of the OpenGL graphics API, providing documentation, resources, andsupport for OpenGL developers.

HKBK
College of Engineering

No.22/1, Opp., Manyata Tech Park Rd, Nagawara, Bengaluru, Karnataka 560045.
Approved by AICTE & Affiliated by VTU

Department of Computer Science & Engineering

DEPARTMENT MISSION AND VISION

VISION

To advance the intellectual capacity of the nation and the international community by imparting knowledge to graduates who are globally recognized as innovators, entrepreneur and competent professionals.

MISSION

M - 1. To provide excellent technical knowledge and computing skills to make the graduates globally competitive with professional ethics.

M - 2. To involve in research activities and be committed to lifelong learning to make positive contributions to the society. Institute

INSTITUTE MISSION AND VISION

VISION

To empower students through wholesome education and enable the students to develop into highly qualified and trained professionals with ethics and emerge as responsible citizen with broad outlook to build a vibrant nation.

MISSION

M - 1. To achieve academic excellence through in-depth knowledge in science, engineering and technology through dedication to duty, innovation in teaching and faith in human values.

M - 2. To enable our students to develop into outstanding professionals with high ethical standards to face the challenges of the 21st century

M - 3. To provide educational opportunities to the deprived and weaker section of the society, to uplift their socio-economic status.

PROGRAM OUTCOMES(PO'S)

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

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

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

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

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

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

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

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

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

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

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

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

PROGRAM SPECIFIC OUTCOMES (PSOs)

PSO-1. Problem-Solving Skills: An ability to investigate and solve a problem by analysis, interpretation of data, design and implementation through appropriate techniques, tools and skills.

PSO-2. Professional Skills: An ability to apply algorithmic principles, computing skills and computer science theory in the modelling and design of computer-based systems.

PSO-3. Entrepreneurial Ability: An ability to apply design, development principles and management skills in the construction of software product of varying complexity to become an entrepreneur.

PROGRAM EDUCATIONAL OBJECTIVES (PEO)

PEO-1. To provide students with a strong foundation in engineering fundamentals and in the computer science and engineering to work in the global scenario.

PEO-2. To provide sound knowledge of programming and computing techniques and good communication and interpersonal skills so that they will be capable of analyzing, designing and building innovative software systems.

PEO-3. To equip students in the chosen field of engineering and related fields to enable him to work in multidisciplinary teams.

PEO-4. To inculcate in students professional, personal and ethical attitude to relate engineering issues to broader social context and become responsible citizen.

PEO-5. To provide students with an environment for life-long learning which allow them to successfully adapt to the evolving technologies throughout their professional career and face the global challenges.