VISVESVARAYA TECHNOLOGICAL UNIVERSITY

BELAGAVI -590018

A Mini Project Report On

"BASKETBALL SHOOTER"

Submitted in the partial fulfillment of the requirements for the award of the Degree of

Bachelor of Engineering in Computer Science and Engineering

Submitted by

RESHMA.K **(1OX20CS104)**

SATISH.A **(1OX20CS116)**

V.LOKESH **(1OX20CS135)**

Under the guidance of

Prof. Manjula L

Project Guide

Department of Computer Science and Engineering

The Oxford College of Engineering

Hosur Road, Bommanahalli, Bengaluru-560068

2022-2023

THE OXFORD COLLEGE OF ENGINEERING

Hosur Road, Bommanahalli, Bengaluru – 560068

(Approved by AICTE, New Delhi, Accredited by NAAC Grade A, NBA New Delhi & Affiliated by VTU, Belagavi-590018)

Department of Computer Science and Engineering

CERTIFICATE

Certified that the mini project work entitled **"BASKETBALL SHOOTER"** carried out by **Reshma.K (1OX20CS104),Satish.A (1OX20CS116),V.Lokesh (1OX20CS135) ,**bonafie students of The Oxford College of Engineering, Bangalore in partial fulfillment for the award of the Degree of **Bachelor of Engineering in Computer Science and Engineering** of **Visvesvaraya Technological University, Belagavi** during the year 2022-2023. The Mini project report has been approved as it satisfies the academic requirements in respect of project work prescribed for the said degree.

Prof.Manjula L	**Dr. R.Ch.A.Naidu**	**Dr. N Kannan**
Project Guide	**Professor & H.O.D**	**Principal, TOCE**

External Viva

Name of the Examiners **Signature with Date**

1._____ _____

2._____ _____

THE OXFORD COLLEGE OF ENGINEERING

Hosur Road, Bommanahalli, Bengaluru – 560068

(Approved by AICTE, New Delhi, Accredited by NAAC Grade A, NBA New Delhi & Affiliated by VTU, Belagavi-590018)

Department of Computer Science and Engineering

Department Vision

To establish the department as a renowned Centre of excellence in the area of scientific education, research with industrial guidance, and exploration of the latest advances in the rapidly changing field of computer science.

Department Mission

M1:To produce the Best Computer Science Professionals with intellectual skills.

M2:To provide a Vibrant Ambiance that promotes Creativity, Technology Competent and Innovations for the new Era.

M3:To Pursue Professional Excellency with Ethical and Moral Values to sustain in the highly demanding world.

THE OXFORD COLLEGE OF ENGINEERING

Hosur Road, Bommanahalli, Bengaluru – 560068

(Approved by AICTE, New Delhi, Accredited by NAAC Grade A, NBA New Delhi & Affiliated by VTU, Belagavi-590018)

Department of Computer Science and Engineering

DECLARATION

We, the students of sixth semester B.E, in the Department of Computer Science and Engineering, **The Oxford College of Engineering**, Bengaluru declare that the Project work entitled **"BASKETBALL SHOOTER"** has been carried out by us and submitted in partial fulfillment of the course requirements for the award of degree in **Bachelor of Engineering in Computer Science and Engineering** discipline of **Visvesvaraya Technological University, Belagavi** during the academic year **2022-2023**. Further, the matter embodied in the dissertation has not been submitted previously by anybody for the award of any degree or diploma to any other university.

RESHMA.K

SATISH.A

V.LOKESH

Place: Bengaluru

Date:

ABSTRACT

OpenGL (Open Graphics Library) is a cross-platform, hardware-accelerated, language-independent, industrial standard API for producing 3D (including 2D) graphics. Modern computers have dedicated GPU (Graphics Processing Unit) with its own memory to speed up graphics rendering. Since OpenGL is a graphics API and not a platform of its own, it requires a language to operate in and the language of choice is C++. Basketball shooter is a 3-Dimensional project designed as a game, in which a player shoots a ball into the basket. This shooting is done with the help of specific keys. The program starts with a player with a ball in his hand in front of the basket in the basketball court. The basketball court can be viewed in different angle by the choice of the user. The view of the basketball court along with the player can be changed by the mouse click and the keyboard keys. By the mouse click the basketball court is viewed around a specific angle. Using the keys the basketball court is viewed as per users choice. The player shoots towards the basket when the key 'S' or s' is used irrespective of the case. The 'X','Y','Z' are used to zoom out the court from the x-axis, y-axis, z-axis respectively. Whereas the 'x', 'y', 'z' are used to zoom in the court from the x-axis , y-axis, z-axis respectively. This gives a closer and good view around the basketball court.

ACKNOWLEDGEMENT

The satisfaction and euphoria that accompany the successful completion of any task would be incomplete without the mention of people who made it possible whose constant guidance and encouragement crowned our effort with success.

We consider ourselves proud to be a part of The Oxford family, the institution that stood by our way in all our endeavours. We have a great pleasure in expressing our deep sense of gratitude to the founder chairman **late Sri S. Narasa Raju** and to our chairman **Sri S. N. V. L. Narasimha Raju** for providing us with a great infrastructure and well-furnished labs.

We would like to express our gratitude to **Dr. N. Kannan,** Principal, The Oxford College of Engineering for providing us a congenial environment and surrounding to work in.

Our hearty thanks to **Dr. R Ch A Naidu,** Professor & Head, Department of Computer Science and Engineering, The Oxford College of Engineering for his encouragement and support.

Guidance and deadlines play a very important role in successful completion of the project report on time. We convey our gratitude to **Ms. Manjula L ,** Assistant Professor, Department of Computer Science and Engineering for having constantly monitored the completion of the Project Report and setting up precise deadlines.

We also thank them for their immense support, guidance, specifications and ideas without which the Project Report would have been incomplete. Finally a note of thanks to the Department of Computer Science and Engineering, both teaching and non-teaching staff for their cooperation extended to us.

RESHMA.K (1OX20CS104)

SATISH.A (1OX20CS116)

V.LOKESH (1OX20CS135)

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 PREAMBLE

Computer graphics is a sub-field of computer science which studies methods for digitally synthesizing and manipulating visual content. Although the term often refers to the study of three-dimensional computer graphics it also encompasses two-dimensional computer graphics and image processing. The individuals who serve as professional designers for computers graphics are known as "Graphics Programmers", who often are computer programmers with skills in computer graphics design. The studies the anesthetic manipulation of visual and geometric information using computational techniques. It focuses on the mathematical and computational foundations of image generation and processing rather than purely aesthetic issues. Computer graphics is often differentiated from the field of visualization, although the two fields have many similarities.

1.2 OVERVIEW OF OPENGL

OpenGL (Open Graphics Library) is a standard specification defining a cross-language, cross-platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of function calls which can be used to draw complex two-dimensional and three-dimensional scenes from simple primitives. The header file <GL/glut.h> is included in order to utilize the inbuilt functions needed in the implementation. C programming language is used for the project coding. OpenGL serves two main purposes.

- To hide the complexities of interfacing with different 3D accelerators, by presenting the programmer with a single, uniform API.
- To hide the differing capabilities of hardware platforms, by requiring that all implementations support the full OpenGL feature set.

OpenGL's basic operation is to accept primitives such as points, lines and polygons, and convert them into pixels. This is done by a graphics pipeline known as the OpenGL state machine. Most OpenGL commands either issue primitives to the graphics pipeline or configure how the pipeline processes these primitives.

1.1.2 PRIMITIVES AND COMMANDS

OpenGL draws primitives—points, line segments, or polygons—subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors, normal, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its associated data are processed independently, in order, and in the same way. The only exception to this rule is if the group of vertices must be clipped so that a particular primitive fit within a specified region; in this case, vertex data may be modified and new vertices created. The type of clipping depends on which primitive the group of vertices represents.

Commands are always processed in the order in which they are received, although there may be an indeterminate delay before a command takes effect. This means that each primitive is drawn completely Before any subsequent command takes place. It also means effect that state-querying commands return data that's consistent with complete execution of all previously issued OpenGL commands.

CHAPTER 2

SYSTEM REQUIREMENTS AND SPECIFICATION

2.1 GENERAL DESCRIPTION OF THE SYSTEM

The existing system involves computer graphics. Computer graphics started with the display of data on hardcopy plotters and cathode ray tube screens soon after the introduction of computer themselves. These models include physical, mathematical, engineering, architectural 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, such as keyboard, mouse or touch-sensitive panel on the screen. Interactive computer graphics is the most important means of producing pictures since the invention of photography and television.

2.2 SYSTEM REQUIREMENTS

2.2.1 HARDWARE REQUIREMENTS

Dual Core Processor

- 2GB RAM
- 40GB Hard disk
- Mouse and other pointing devices
- Keyboard

2.2.2 SOFTWARE REQUIREMENTS

Operating System: Windows

- Programming Languages: C
- Compiler – C Compiler
- Graphics library – GL/glut.h
- OpenGL 2.0

CHAPTER 3

SYSTEM DESIGN AND MODELLING

3.1 PRELIMINARY DESIGN

Basketball shooter:

- The game designed as a basketball player
- The ball is shot into the basket by the player with the help of keys.
- The game also focuses on the view of the basketball court.
- The view around the court can be taken with axis rotation by the help of keys and mouse clicks.

The project is implemented on C platform with the help of OpenGL in-built functions. Care is taken to provide easy-to-use mouse and keyboard interface involving an icon-based interaction.

3.2 ARCHITECTURAL DESIGN

In proposed system, the OpenGL is a graphic software system designed as a gaming, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using.

OpenGL doesn't provide high-level commands for describing models of three-dimensional objects. OpenGL is merely a graphics library, window and context creation must be handled by an external library, usually provided by the operating system. But since you could be using a different operating system than me, a cross-platform library must be used, and FreeGLUT is an Open-Source library.

CHAPTER 4

IMPLEMENTATION

4.1 MODULE DESCRIPTION

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 the Utah teapot. GLUT also has some limited support for creating pop-up menus.

This project mainly uses :

GLUT

Since we are using code blocks we use some of the header files as shown below.

(a) **Stdio.h**

All functions in C are declared in header files thus programmer have to include this in order to use functions declared in it. This header in the C standard library contains macro definitions, constants, and declarations of functions and type used for various standard input and output operations. The name “stdio” stands “standard library”.

(b) **GL/glut.h**

It includes the definitions for the macro such as GL_POINTS, GL_LINES and GL_POLYGON etc., which are used by the programmer.

(c) **Math.h**

It involves the mathematical functions used in the program.

(d) **Time.h**

It contains definitions of functions to get and manipulate date and time information.

4.2 FUNCTIONS USED:

- **void poles():**

  This function is designed to view the poles of the basketball court.

- **void lines():**

  This function is designed to view the lines of the basketball court.

- **void onBoardLines():**

  This function is designed to view lines on the board of the basketball court.

- **void board():**

  This function is designed the board of the basketball court.

- **void base():**

  Using this function the base of the basketball court are designed.

- **void polygon()**

  Using this function different polygons are designed in the basketball court.

- **void circle()**

  Using this function circles are designed around the basketball court.

- **void ring():**

  Using this function circles are designed around the basketball court.

- **void semicircle():**

  In this function, semicircles on the basketball court are designed.

- **void ball():**

  In this function, the basketball is designed.

- **void net():**

  In this function, the basketball net is designed

- **void characterDesign():**

  In this function, the character on the basketball court is designed

- **void draw():**

  This function contains all the other functions used.

- **void update():**

  Designed to update the dimensions to view the basketball court.

- **void mouse():**

  The function contains the code on mouse actions.

- **void myReshape():**

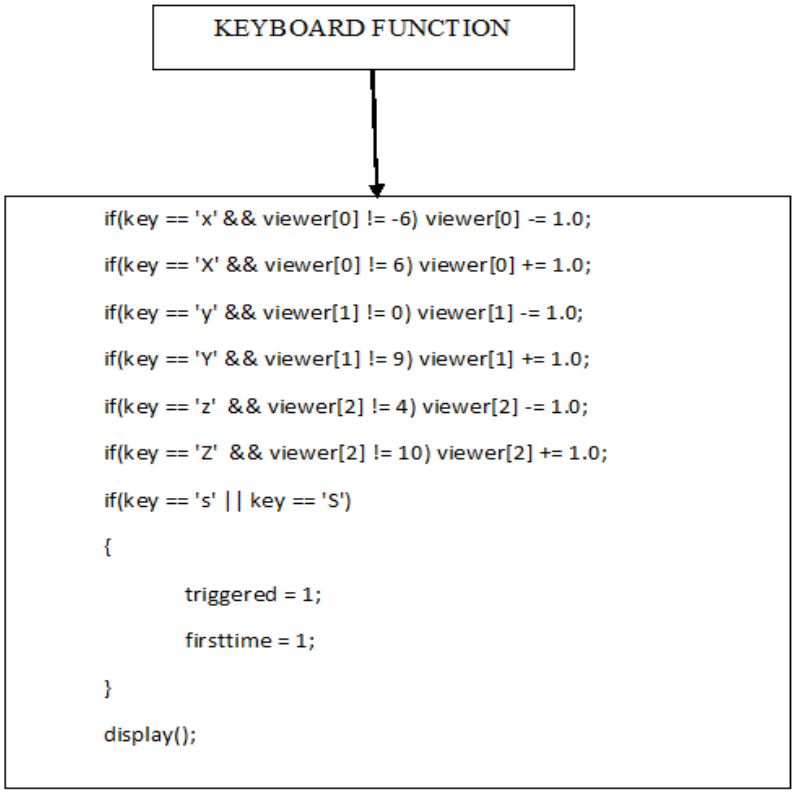
  Used for reshaping the display

- **void keys():**

  Contains the code of keys used and implemented for the game.

4.3 DISPLAYING

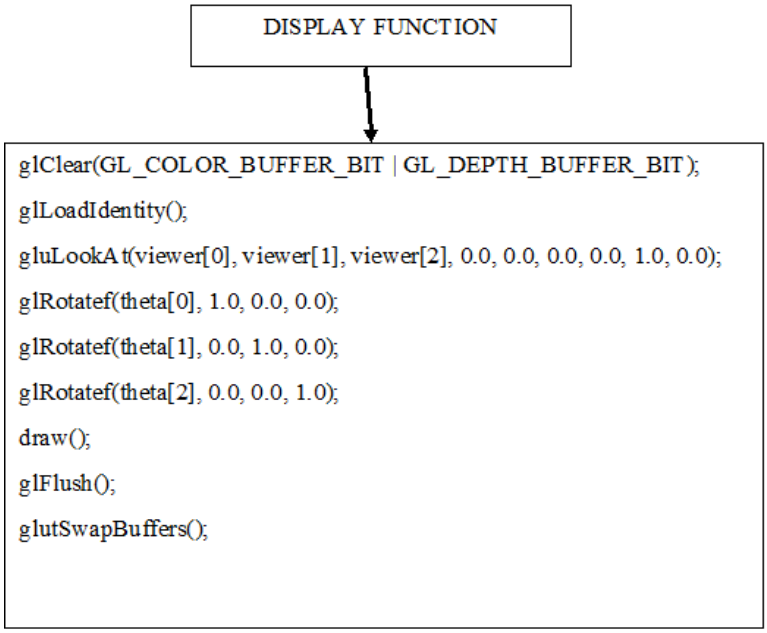
Keyboard function : Manages operations by various keys pressed on the key board .



Display function:

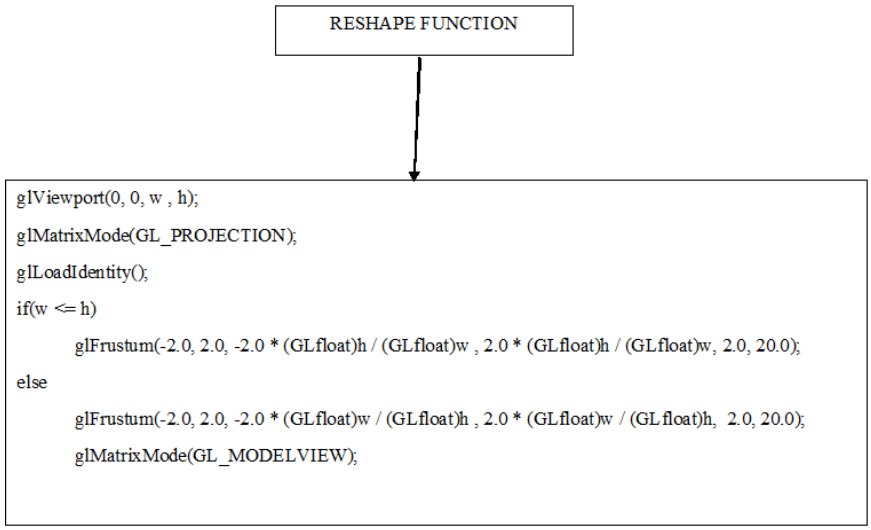
Uses the following functions:

- **glClear (GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT):**
  Indicates the buffers currently enabled for color writing and also indicates the depth buffer.
- **glPushMatrix (), glPopMatrix ()** :
  to push and pop the current matrix stack.
- **glFlush ()**:
  force execution of GL commands in finite time.
- **glutSwapBuffers ():**
  Swap the buffers makes the result of rendering visible.



Reshape function:

- **glMatrixMode (GL_PROJECTION) :**
  applies subsequent matrix operations to the projection matrix stack.
- **glMatrixMode (GL_MODELVIEW):**
  applies subsequent matrix operations to the model view matrix stack. We adjust the viewing volume. We use the whole window for rendering and adjust point size to window size.



4.4 MAIN FUNCTION

Here we specify the initial display mode, window size and position. Create a new window where the output is rendered.

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

{

glutInit(&argc , argv);

glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

glutInitWindowSize(1000, 1000);

glutCreateWindow(" 3D Basketball Shoot");

glutReshapeFunc(myReshape);

glutDisplayFunc(display);

glutMouseFunc(mouse);

glutKeyboardFunc(keys);

glEnable(GL_DEPTH_TEST);

glutTimerFunc(1, update, 0);

glClearColor(0,0,0,0);

glutMainLoop();

}
```

CHAPTER 5

TESTING

This chapter gives the outline of all the testing methods that are carried out to get a bug free application. Quality can be achieved by testing the product using different techniques at different phases of the project development.

5.1 TESTING PROCESS

Testing is an integral part of software development. Testing process, in a way certifies, whether the product, that is developed, complies with standards, that it was designed to. Testing process involves building of test cases, against which, the product has to be tested. In some cases are done based on the system requirements specified for the product/software, which is to be developed

5.2 TESTING OBJECTIVES

The main objectives of testing process are as follows:

- Testing is a process of executing a program with the intent of finding an error.
- A good test case is one that has high probability of finding an as yet undiscovered error.
- A successful test is one that uncovers an as yet undiscovered error.

5.3 LEVELS OF TESTING

Different levels of testing are used in the testing process; each level of testing aims to test different aspects of the system. The basic levels are unit testing, integration testing, system testing and acceptance testing.

5.3.1 UNIT TESTING

Unit testing focuses verification effort on the smallest unit of software design the module. The software built, is a collection of individual modules. Each module performs a unique function. In this kind of testing, exact flow of control for each module was verified. With detailed design Considerations used as a guide, important control paths are tested to uncover errors within the boundary of the module. The Unit Test can be conducted in parallel for multiple modules.

Module	Action	Expected Output
Display module	Display screen is generated based on the input.	The screen is displayed based on the mouse and keyboard function
Keyboard module	'X' is pressed. 'x' is pressed. 'Y' is pressed. 'y' is pressed. 'Z' is pressed. 'z' is pressed. 'S' or 's' is pressed.	Zoom out of x-axis. Zoom in of x-axis. Zoom out of y-axis. Zoom in of y-axis. Zoom out of z-axis. Zoom in of z-axis. The ball is shot into the basket.
Mouse Module	On clicking left button.	Rotational view of the basketball court

Table 5.1: Test Cases for Basketball Shooter.

5.3.2 INTEGRATION TESTING

The second level of testing is called integration testing. In this, many class-tested modules are combined into subsystems, which are then tested. The goal here is to see if all the modules can be integrated properly. We have integrated all the classes and have checked for compatibility. Errors obtained are identified and debugged.

5.3.3 SYSTEM TESTING

Here the application is tested. The reference document for this process is the requirements documents, and the goal is to see if the application meets its requirements.

The module and component of ethereal was thoroughly tested to remove bugs through a System testing strategy. Test cases were generated for all possible input sequences and the output was verified for its correctness.

CONCLUSION

Engineers who are game developers can build their model by looking this project. They get a good picturization by seeing this and it will be helpful for them in animation game developing. So this project will be benefited by different developers. The view of the basketball court along with the player can be changed by the mouse click and the keyboard keys. By the mouse click the basketball court is viewed around a specific angle. Using the keys the basketball court is viewed as per users choice. An attempt is made to develop an OpenGL package which means necessary requirements of the user successfully. Since it is user friendly, it enables the user to interact efficiently and easily. The development of the mini project which has been built by using point as a source, has given a good exposure towards OpenGL through which some of the techniques which help in development of animations, gaming was well understood. It helps to understand the basic implementation of functions and basics of OpenGL.

REFERENCES

Text Book: 1. Developing Simple Games with OpenGL.

2. Donald Hearn & Pauline Baker: Computer Graphics with OpenGL Version 3rd Edition, Pearson, Education, 2011

Websites

1. https://www.youtube.com/watch?v=45MIykWJ-C4
2. https://www.youtube.com/playlist?list=PLlrATfBNZ98foTJPJ_Ev03o2oq3-GGOS2
3. https://www.w3schools.com/c/
4. https://www.geeksforgeeks.org/c-programming-language/

APPENDIX A

SNAPSHOTS OF THE PROJECT

Figure 1: Initial View

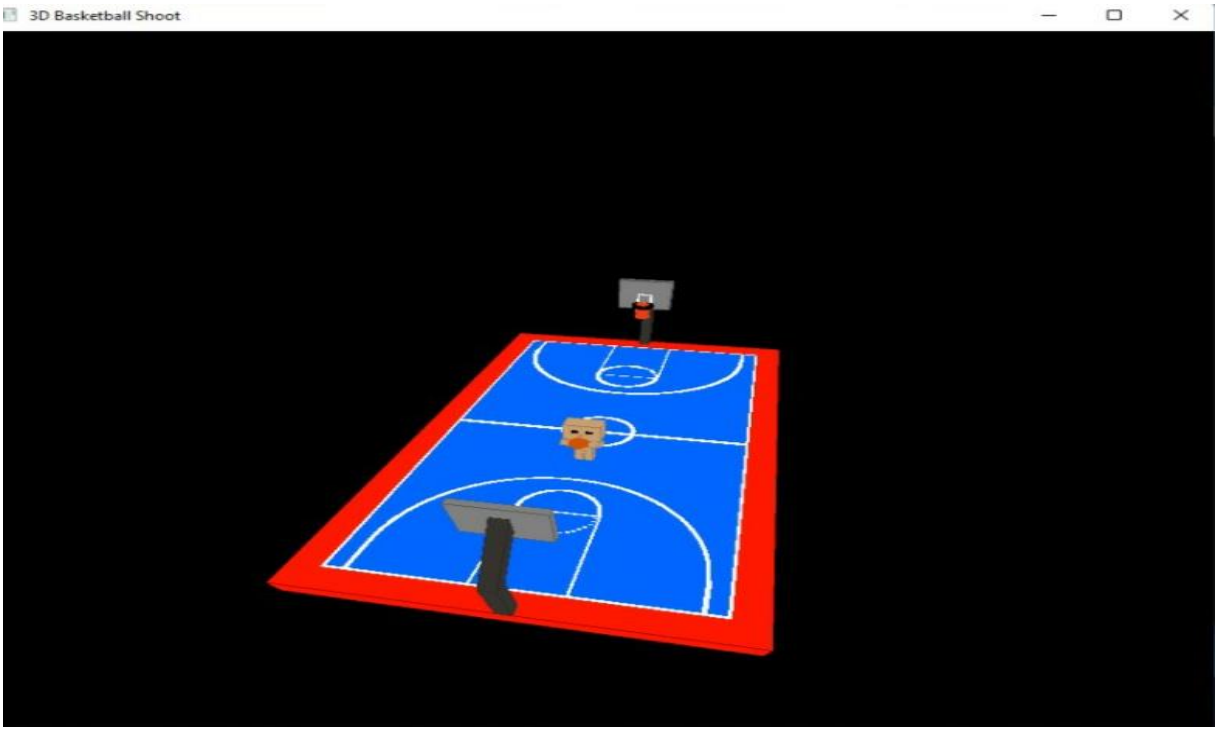


Figure 2: Top View

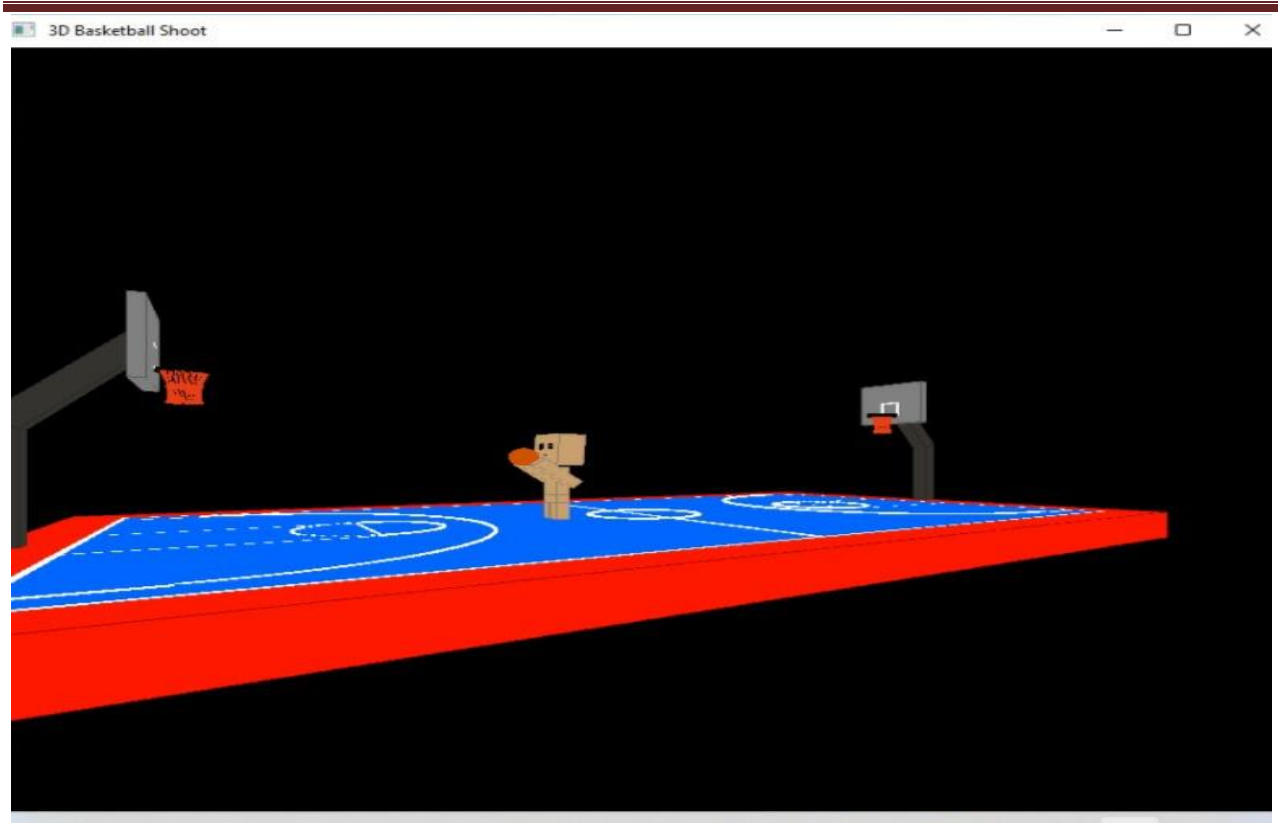


Figure 3: Left View

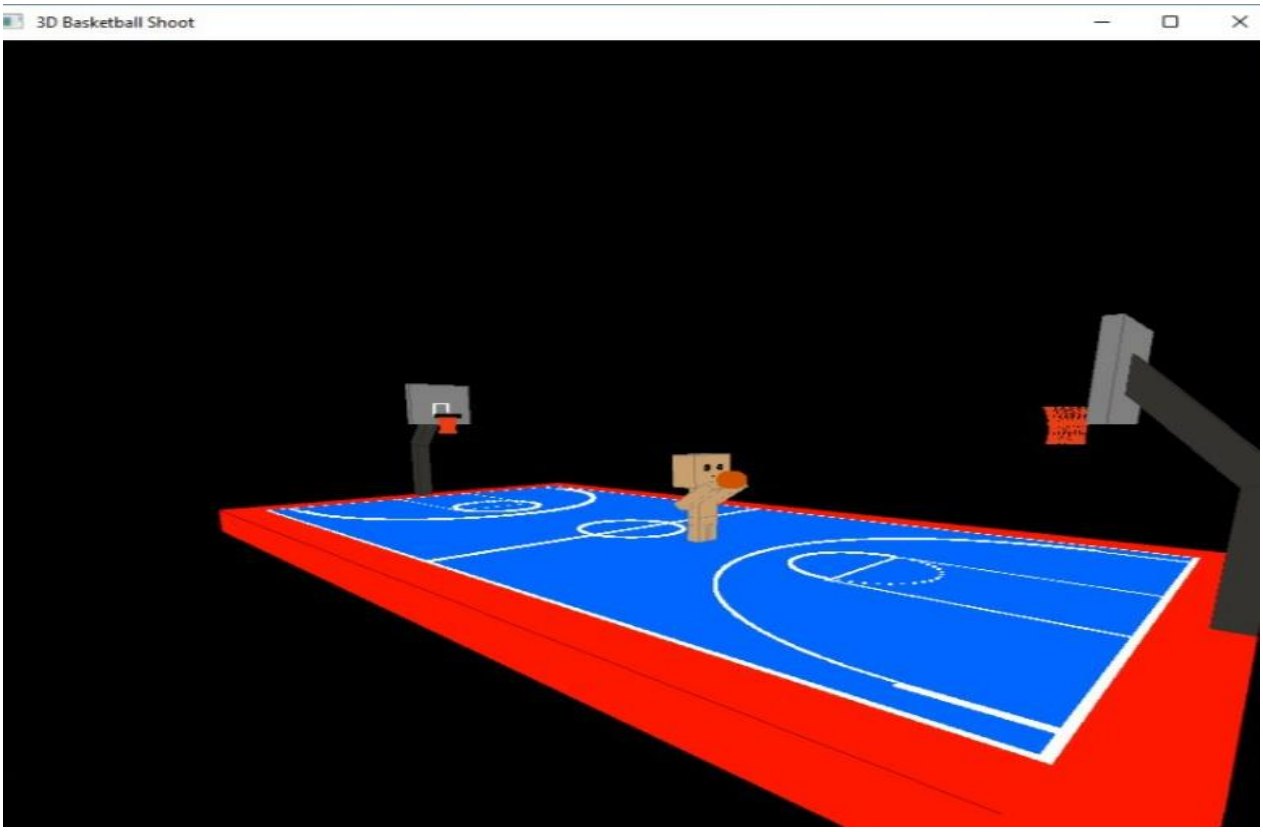


Figure 4: Right View

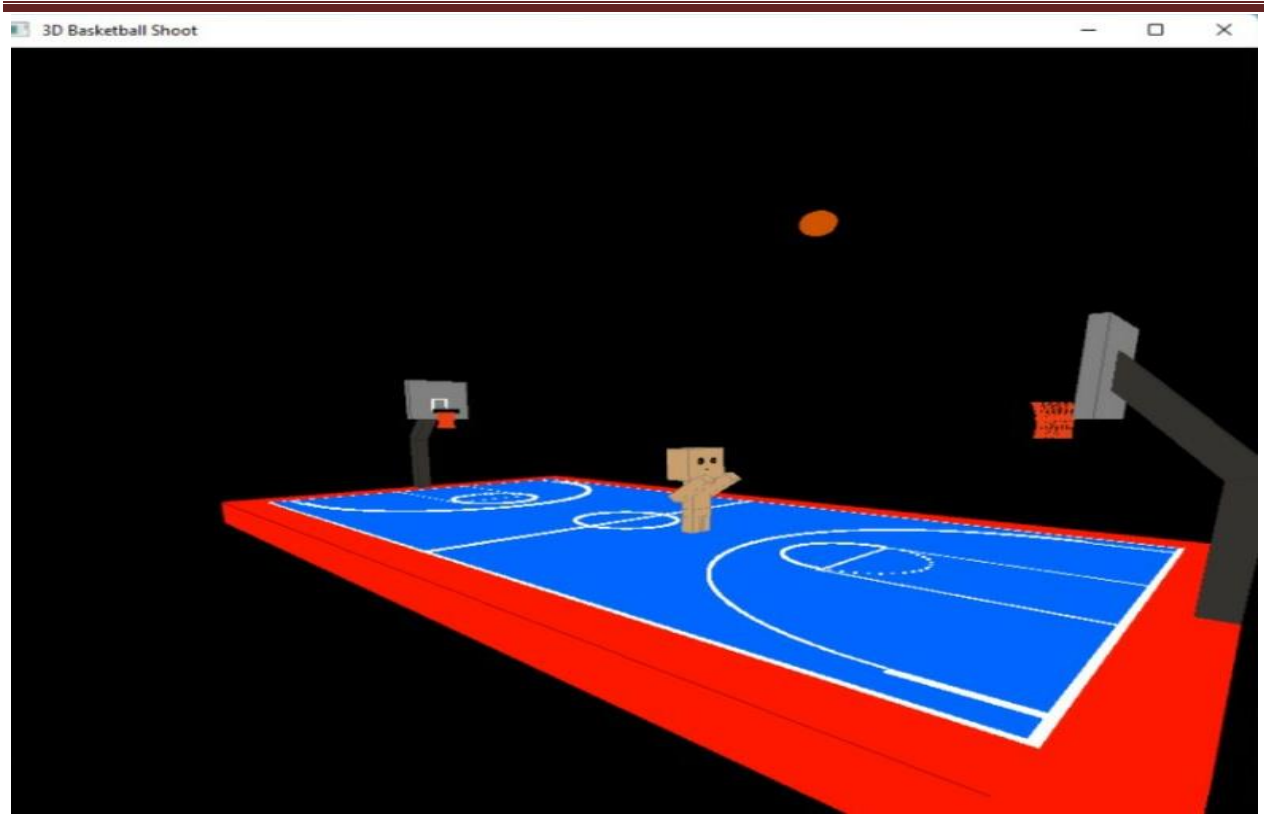


Figure 5: Shooting the Ball in the Basketiew

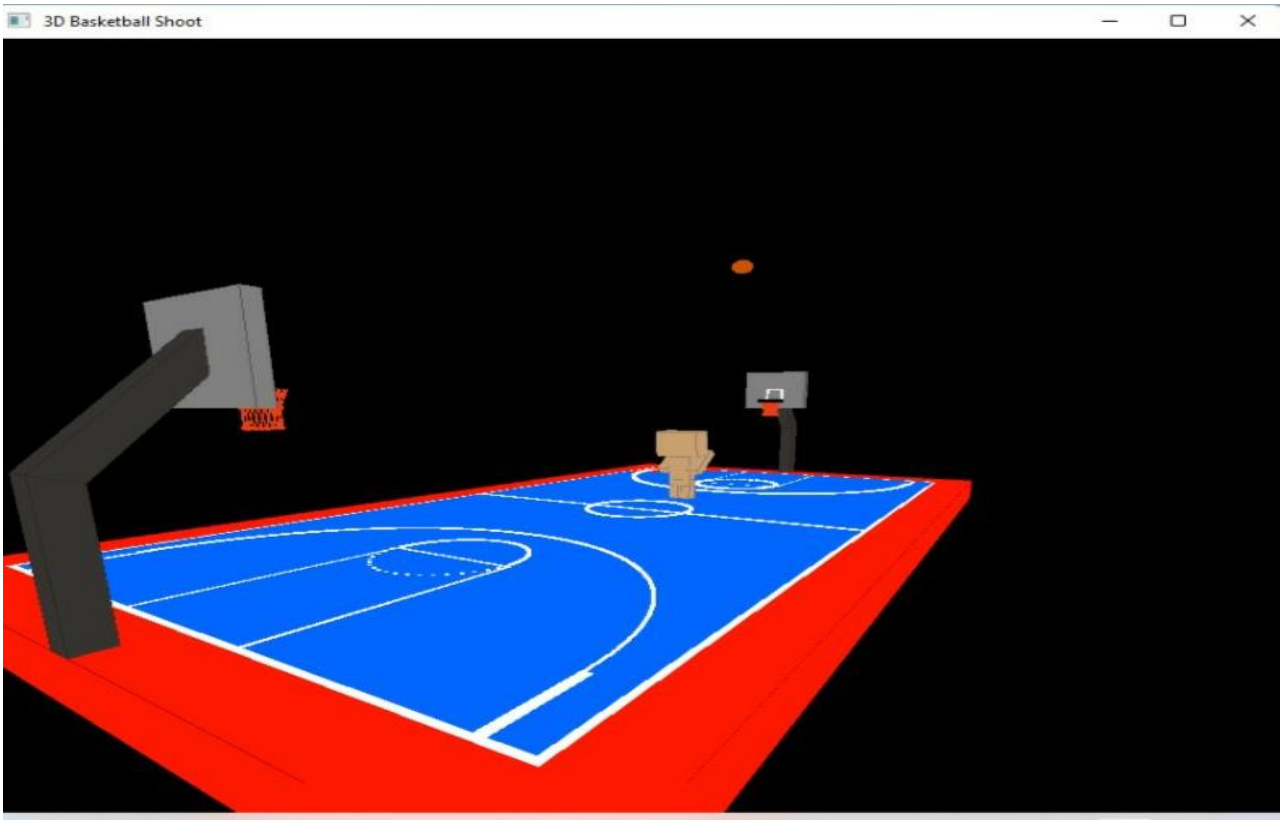


Figure 6: Back View of Shooting the Ball

APPENDIX B

CODE OF THE PROJECT

```
#include<stdio.h>

#include<GL/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(0.0, 0.0, 0.0);

        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();
        //border for the poles
        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_LINE_LOOP);
        glColor3f(43.0 / 255.0, 39.0/ 255.0, 37.0/ 255.0);
        glVertex3fv(secondPoleVertices[a]);
        glVertex3fv(secondPoleVertices[b]);
        glVertex3fv(secondPoleVertices[c]);
        glVertex3fv(secondPoleVertices[d]);
        glEnd();
}
void lines(float a, float b, float c, float d)
{       //a = -2.5, b = -0.05 c = 0.05 d = 2.5
        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, 1.0);
        glVertex3f(a , -0.9999 , b);
        glVertex3f(d , -0.9999 , b);
        glVertex3f(d , -0.9999 , c);
        glVertex3f(a , -0.9999 , c);
        glEnd();
        glBegin(GL_POLYGON);
```

```
        glColor3f(1.0, 1.0, 1.0);
        glVertex3f(a , -0.9999 , b);
        glVertex3f(d , -0.9999 , b);
        glVertex3f(d , -0.9999 , c);
        glVertex3f(a , -0.9999 , c);
        glEnd();
}
void onBoardLines(float a, float b, float c, float d)
{
        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, 1.0);
        glVertex3f(a, b, -4.29);
        glVertex3f(d, b, -4.29);
        glVertex3f(d, c, -4.29);
        glVertex3f(a, c, -4.29);
        glEnd();

        glBegin(GL_POLYGON);
        glColor3f(1.0, 1.0, 1.0);
        glVertex3f(a, b, 4.29);
        glVertex3f(d, b, 4.29);
        glVertex3f(d, c, 4.29);
        glVertex3f(a, c, 4.29);
        glEnd();
}
void board(int a, int b, int c, int d)
{               glBegin(GL_POLYGON);
        glColor3f(0.5, 0.5, 0.5);
        glVertex3fv(firstBoardVertices[a]);
        glVertex3fv(firstBoardVertices[b]);
        glVertex3fv(firstBoardVertices[c]);
```

```
        glVertex3fv(firstBoardVertices[d]);
        glEnd();
        glBegin(GL_LINE_LOOP);
        glColor3f(86.0/255.0, 86.0/255.0, 86.0/255.0);
        glVertex3fv(firstBoardVertices[a]);
        glVertex3fv(firstBoardVertices[b]);
        glVertex3fv(firstBoardVertices[c]);
        glVertex3fv(firstBoardVertices[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(1.0, 0.1, 0.0);
        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);
        //court color
        glBegin(GL_POLYGON);
        glColor3f(0.0, 0.4, 1.0);
        glVertex3fv(courtVertices[a]);
        glColor3f(0.0, 0.4, 1.0);
        glVertex3fv(courtVertices[b]);
        glColor3f(0.0, 0.4, 1.0);
        glVertex3fv(courtVertices[c]);
        glColor3f(0.0, 0.4, 1.0);
        glVertex3fv(courtVertices[d]);
                glEnd();
        // pole from bast to the top
        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 from center to board
        poles(6, 7, 9, 8);
        poles(4, 5, 11, 10);
        poles(6, 4, 10 , 8);
        poles(7, 5, 11, 9);
```

```
//drawing board
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);        // remove the comments to draw the board faces
//center line
lines(-2.5, -0.05, 0.05, 2.5);
//side lines
lines(-2.50, 4.7, -4.7, -2.55);
lines(2.50, 4.7, -4.7, 2.55);
//base lines
lines(-2.55, 4.70, 4.75, 2.55);
lines(-2.55, -4.70, -4.75, 2.55);
//three pointer lines
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);
//two pointer lines
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 joining the above two lines
lines(-0.6, 2.8, 2.84, 0.6);
lines(-0.6, -2.8, -2.84, 0.6);
//vertical lines on the board
onBoardLines(-0.15, 1.2, 1.24, 0.15); //bottom
onBoardLines(-0.15, 1.5, 1.54, 0.15); //down
```

```
        onBoardLines(-0.15, 1.2, 1.5, -0.10);
        onBoardLines(0.15, 1.2, 1.5, 0.10);
}
void circle(float r)
{       int i;
        glColor3f(1.0, 1.0, 1.0);
        glPointSize(3.0);
        glBegin(GL_POINTS);
        for(i = 0; i < 1000; i++)
        {               //x and y defines the radius
                glVertex3f( (r * cos(2*3.14159 * i/1000.0)), -0.9999, (r * sin(2*3.14159 *
i/1000.0)));
        }
        glEnd();
}
void Dcircle(float r)
{       int i;
        glColor3f(1.0, 1.0, 1.0);
        glBegin(GL_POINTS);
        for(i = 0; i < 1000; i++)
        {       //x and y defines the radius
glVertex3f( (r * cos(1*3.14159 * i/1000.0)), -0.9999, 2.8 - (r * sin(1*3.14159 * i/1000.0)));
glVertex3f( (r * cos(1*3.14159 * i/1000.0)), -0.9999, -2.8 + (r * sin(1*3.14159 *
i/1000.0)));
        }
        for(i = 0; i < 20; i++)
        {       //x and y defines the radius
                glVertex3f( (r * cos(1*3.14159 * i/20.0)), -0.9999, 2.8 + (r * sin(1*3.14159
* i/20.0)));
                glVertex3f( (r * cos(1*3.14159 * i/20.0)), -0.9999, -2.8 - (r * sin(1*3.14159
* i/20.0)));
        }
```

```
        glEnd();
}
void ring(float r)
{       int i;
        glColor3f(0.0, 0.0, 0.0);
        glBegin(GL_POINTS);
        for(i = 0; i< 1000; i++)
        {

                glVertex3f((r * cos(2*3.14159 * i/1000.0)), 1.2, 4.3 - 0.19 + (r *
sin(2*3.14159 * i/1000.0)));
                glVertex3f((r * cos(2*3.14159 * i/1000.0)), 1.2, -4.3 + 0.19 + (r *
sin(2*3.14159 * i/1000.0)));
        }
        glEnd();
}
void semicircle(float r)
{       int i;
        glColor3f(1.0, 1.0, 1.0);
        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); //calculated using the last vertex of parabola
        }
        else
        {       glTranslatef(0.0, 0.8, -2.8); //calculated using the last vertex of parabola
        }
                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];
        //choosing the pole
        if(poleChooser == 1)
        {
                poleDecider = 4.3 - 0.19;
        }
        else
        {
                poleDecider = -4.3 + 0.19;
        }
        //top vertices
        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)); //x values
                topVertices[1][i] = (poleDecider + (r) * sin(2 * 3.14159 * i/20.0)); //y values
                glBegin(GL_POINTS);
                glVertex3f(topVertices[0][i], 1.2, topVertices[1][i]);
```

```
        glEnd();
    }
    //middle vertices
    for(i = 0; i < 20; i++)
    {       middleVertices[0][i] = ((r - 0.05)* cos(2 * 3.14159 * i/20.0)); //x values
            middleVertices[1][i] = ( poleDecider + (r - 0.05)* sin(2 * 3.14159 * i/20.0));
            //y values
            glBegin(GL_POINTS);
            glVertex3f(middleVertices[0][i], 1.0, middleVertices[1][i]);
            glEnd();
    }
    //bottom vertices
    for(i = 0; i < 20; i++)
    {
            bottomVertices[0][i] = ((r - 0.05)* cos(2 * 3.14159 * i/20.0)); //x values
            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();
    }
    //drawing lines using vertices to get the rhombus pattern
    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]);
            }
            glEnd();
        }
}
float rightLeg[][3] = {{0.03, -0.9999, -0.1}, {0.03, -0.9999, 0.1}, {0.2, -0.9999, 0.1}, {0.2, -0.9999, -0.1},
        {0.03, -0.6, -0.1}, {0.03, -0.6, 0.1}, {0.2, -0.6, 0.1}, {0.2, -0.6, -0.1}};
float leftLeg[][3] = {  {-0.2, -0.9999, -0.1}, {-0.2, -0.9999, 0.1}, {-0.03, -0.9999, 0.1}, {-0.03, -0.9999, -0.1},   {-0.2, -0.6, -0.1}, {-0.2, -0.6, 0.1}, {-0.03, -0.6, 0.1}, {-0.03, -0.6, -0.1}};
float trunk[][3] = {    {-0.2, -0.6, -0.1}, {-0.2, -0.6, 0.1}, {0.2, -0.6, 0.1}, {0.2, -0.6, -0.1},
        {-0.2, -0.4, -0.1}, {-0.2, -0.4, 0.1}, {0.2, -0.4, 0.1}, {0.2, -0.4, -0.1}};
float body[][3] = {     {-0.2, -0.4, -0.1}, {-0.2, -0.4, 0.1}, {0.2, -0.4, 0.1}, {0.2, -0.4, -0.1},
        {-0.2, 0.3, -0.1}, {-0.2, 0.3, 0.1}, {0.2, 0.3, 0.1}, {0.2, 0.3, -0.1}};
float head[][3] = {     {-0.4, 0.35, -0.2}, {-0.4, 0.35, 0.2}, {0.4, 0.35, 0.2}, {0.4, 0.35, -0.2},   {-0.4, 1.1, -0.2}, {-0.4, 1.1, 0.2}, {0.4, 1.1, 0.2}, {0.4, 1.1, -0.2}};
float leftHand[][3] = {{-0.2, 0.2, -0.1}, {-0.2, 0.2, 0.1}, {-0.4, 0.2, 0.1}, {-0.4, 0.2, -0.1},
        {-0.2, -0.6, -0.1}, {-0.2, -0.6, 0.1}, {-0.4, -0.6, 0.1}, {-0.4, -0.6, -0.1}};
float rightHand[][3] = {{0.2, 0.2, -0.1}, {0.2, 0.2, 0.1}, {0.4, 0.2, 0.1}, {0.4, 0.2, -0.1},
        {0.2, -0.6, -0.1}, {0.2, -0.6, 0.1}, {0.4, -0.6, 0.1}, {0.4, -0.6, -0.1}};
void characterDesign(int a, int b, int c, int d)
{       float R = 143.0/255.0, G = 125.0 / 255.0, B = 100.0 / 225.0;
        glColor3f(202.0 / 255.0, 160.0/255.0, 100.0/225.0);
        glBegin(GL_POLYGON);
        glVertex3fv(trunk[a]);
        glVertex3fv(trunk[b]);
        glVertex3fv(trunk[c]);
        glVertex3fv(trunk[d]);
        glEnd();
        glColor3f(202.0 / 255.0, 160.0/255.0, 100.0/225.0);
        glBegin(GL_POLYGON);
        glVertex3fv(body[a]);
```

```
        glVertex3fv(body[b]);
        glVertex3fv(body[c]);
        glVertex3fv(body[d]);
        glEnd();
        glColor3f(R, G, B);
        glBegin(GL_LINE_LOOP);
        glVertex3fv(head[a]);
        glVertex3fv(head[b]);
        glVertex3fv(head[c]);
        glVertex3fv(head[d]);
        glEnd();
        int i;
        float r = 0.05;
        glColor3f(0.0, 0.0, 0.0);
        glPointSize(2.0);
        glBegin(GL_POINTS);
        for(i = 0; i< 1000; i++)
        {//x and y defines the radius
glVertex3f( 0.15 - (r * cos(2*3.14159 * i/1000.0)), 0.8 + (r * sin(2*3.14159 * i/1000.0)), -
0.2);
glVertex3f( -0.15 - (r * cos(2*3.14159 * i/1000.0)), 0.8 + (r * sin(2*3.14159 * i/1000.0)), -
0.2);
        }
        glEnd();
        glColor3f(0.0, 0.0, 0.0);
        glBegin(GL_POLYGON);
        glVertex3f(0.0 , 0.55 , -0.21);
        glVertex3f(0.05 , 0.6, -0.21);
        glVertex3f(-0.05 , 0.6 , -0.21);
        glEnd();
        glPopMatrix();
        glColor3f(202.0 / 255.0, 160.0/255.0, 100.0/225.0);
```

```
        glBegin(GL_POLYGON);
        glVertex3fv(rightLeg[a]);
        glVertex3fv(rightLeg[b]);
        glVertex3fv(rightLeg[c]);
        glVertex3fv(rightLeg[d]);
        glEnd();
        glPushMatrix();
        glRotatef(135.0, 1.0, 0.0, 0.0);
        glColor3f(202.0 / 255.0, 160.0/255.0, 100.0/225.0);
        glBegin(GL_POLYGON);
        glVertex3fv(leftHand[a]);
        glVertex3fv(leftHand[b]);
        glVertex3fv(leftHand[c]);
        glVertex3fv(leftHand[d]);
        glEnd();
        glColor3f(R, G, B);
        glBegin(GL_LINE_LOOP);
        glVertex3fv(rightHand[a]);
        glVertex3fv(rightHand[b]);
        glVertex3fv(rightHand[c]);
        glVertex3fv(rightHand[d]);
        glEnd();
        glPopMatrix();
        glColor3f(R, G, B);
        glBegin(GL_LINE_LOOP);
        glVertex3fv(body[a]);
        glVertex3fv(body[b]);
        glVertex3fv(body[c]);
        glVertex3fv(body[d]);
        glEnd();
}
```

```
void character()
{       glScalef(0.7, 0.7, 0.7);
         glTranslatef(0.0, -0.4, -1.5);
        characterDesign(0, 1, 2, 3);
        characterDesign(4, 5, 6, 7);
        characterDesign(0, 1, 5, 4);
        characterDesign(2, 3, 7, 6);
        characterDesign(1, 2, 6, 5);
        characterDesign(0, 3, 7, 4);
}
void draw(void)
{       glPushMatrix();
        if(firsttime == 0)
        {               glTranslatef(0, y, dz);
        }
        ball();
        glPopMatrix();
        polygon(0, 1, 2, 3);
        circle(0.60);
        Dcircle(0.6);
        net(1);
        net(2);
        ring(0.17);
        semicircle(2.22);
        character();

}
static GLfloat theta[] = {0.0, 0.0, 0.0};
static GLint axis = 2;
static GLdouble viewer[] =  {0.0, 7.0, 7.0};
oid display(void)
```

```
{       glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        glLoadIdentity();
        gluLookAt(viewer[0], viewer[1], viewer[2], 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        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);
        draw();
        glFlush();
        glutSwapBuffers();
}
void update(int value)
{       if(firsttime)
        {       y = 0.5;
                dz = 1.2;
                firsttime = 0;
        }
        if(triggered && y >= -2 && z >= -1.0)
        {       y = -(2 * dz * dz) + 3.6;
                dz -= 0.05;
        }
        else
        {       y = -0.7;
                dz = 1.4;
                triggered = 0;
}
        glutPostRedisplay();
        glutTimerFunc(25,update,0);
}
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)
{//test conditions to ensure that the camera always capture the obejct and does move too far
from the object
        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();
}
int main(int argc, char **argv)
{         glutInit(&argc , argv);
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);
        glutInitWindowSize(1000, 1000);
        glutCreateWindow(" 3D Basketball Shoot");
        glutReshapeFunc(myReshape);
        glutDisplayFunc(display);
        glutMouseFunc(mouse);
        glutKeyboardFunc(keys);
        glEnable(GL_DEPTH_TEST);
        glutTimerFunc(1, update, 0);
        glClearColor(0,0,0,0);
        glutMainLoop();
}
```