[**VISVESVARAYA TECHNOLOGICAL UNIVERSITY**](http://www.vtu.ac.in/)

**“JNANA SANGAMA” , BELAGAVI-590018, KARNATAKA**



**Mini Project Report**

**On**

S

**“TIC-TAK-TOE GAME SIMULATION”**

**Submitted in partial fulfillment of the requirement for completion of Computer Graphics**

**Laboratory with Miniproject of VI semester**

**BACHELOR OF ENGINEERING In**

**COMPUTER SCIENCE AND ENGINEERING**

**Submitted By**

**Harshita T 1VA17CS015**

**Jagbeer Poonia 1VA17CS018**

**Under the Guidance of**

**Dr. Sangeetha V Prof. Sunil G L Associate Professor Assistant Professor Department of IS&E Department of CS&E**



**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**SAI VIDYA INSTITUTE OF TECHNOLOGY**

**(Affiliated to Visvesvaraya Technological University, Belagavi | Recognized by Govt. of Karnataka | Approved by AICTE, New Delhi)**

**RAJANUKUNTE, BENGALURU – 560 064**

**2019-20**

S

**SAI VIDYA INSTITUTE OF TECHNOLOGY**

**(Affiliated to Visvesvaraya Technological University, Belagavi | Recognized by Govt. of Karnataka | Approved by AICTE, New Delhi)**

**Rajanukunte, Bengaluru- 560 064**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**



Certified that the Mini Project work entitled *“****TIC-TAK-TOE GAME SIMULATION****”* carried out by **Ms. Harshita T (1VA17CS015) and Mr. Jagbeer Poonia (1VA17CS018)**  bonafide students of **SAI VIDYA INSTITUTE OF TECHNOLOGY**, Bengaluru, in partial fulfillment for the award of Bachelor of Engineering in Computer Science and Engineering of **VISVESVARAYA TECHNOLOGICAL UNIVERSITY**, Belagavi during the year **2019-20.** It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the Report deposited in the departmental library. The Mini Project report has been approved as it satisfies the academic requirements in respect of Computer Graphics Laboratory with Miniproject(17CSL68) prescribed for the said Degree.

**Dr. Sangeetha.V Prof. Sunil G L**

Associate Professor Assistant Professor

Dept. of ISE, SVIT Dept. of CSE, SVIT

**Dr. K. Ananthapadmanabha Dr. H S Ramesh Babu**

Professor & Head Principal

Dept. of CSE, SVIT SVIT

**External Viva:**

**Name of the Examiner Signature with date**

**1.**

**2.**

**ACKNOWLEDGEMENT**

The completion of Mini project brings with and sense of satisfaction, but it is never completed without thanking the persons who are all responsible for its successful completion. First and foremost, I wish to express our deep sincere feelings of gratitude to my Institution, **Sai Vidya Institute of Technology**, for providing me an opportunity to do our education.

I would like to thank the **Management** and **Prof. M R Holla,** Director, Sai Vidya

Institute of Technology for providing the facilities.

I extend my deep sense of sincere gratitude to **Dr. H S Ramesh Babu**, Principal, Sai Vidya Institute of Technology, Bengaluru, for having permitted to carry out the Mini project work on **“TIC-TAK-TOE SIMULATION”** successfully**.**

I am thankful to **Prof. A M Padma Reddy**, Additional Director and Professor, Department of Computer Science and Engineering, Sai Vidya Institute of Technology, for his constant support and motivation.

I express my heartfelt sincere gratitude to **Dr. K. Ananthapadmanabha,** Professor and Head, Department of Computer Science and Engineering, Sai Vidya Institute of Technology, Bengaluru, for his valuable suggestions and support.

I express my sincere gratitude to **Dr. Sangeetha V,** Associate Professor, Department of ISE**,** and **Prof. Sunil G L,** Assistant Professor, Department of CSE, Sai Vidya Institute of Technology, Bengaluru, for their constant support in completing the Mini project.

Finally, I would like to thank all the Teaching, Technical faculty and supporting staff members of Department of Information Science and Engineering, Sai Vidya Institute of Technology, Bengaluru, for their support.

Harshita T 1VA17CS015

Jagbeer Poonia 1VA17CS018

I

**ABSTRACT**

TIC-TAC-TOE is the most popular and easiest game. It is two players (X and O)

game, where each player taking turns marks the spaces in a 3\*3 grid. This project

has been developed in Ubuntu software with interfacing keyboard and mouse. This

project is written in C and used OpenGL.

OpenGL is a standard specification defining a cross language. Open Graphics

Library is a cross-language, cross-platform application programming interface for

rendering 2D and 3D vector graphics.

The API is typically used to interact with a graphics processing unit, to achieve

hardware-accelerated rendering. The game starts with one of the players and the

game ends when one of the players has one whole row/ column/ diagonal filled with

his/her respective character. If no one wins, then the game is said to be draw.

This project has been developed in Ubuntu OS with interfacing keyboard and mouse with menu driven interface. And plans to include lighting, shading and other features in future enhancement.

This project is written in C and used OpenGL (Open Graphics Library). Open Graphics Library is a cross-language, cross-platform application programming interface for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit, to achieve hardware-accelerated rendering.

II

**Table of Contents**

**ACKNOWLEDGEMENT I ABSTRACT II**

**TABLE OF CONTENTS III LIST OF FIGURES V**

**LIST OF TABLES VI**

**CHAPTER 1**

**INTRODUCTION 1**

1.1 Aim 1

1.2 Introduction to Open GL 1

1.3 Project related concepts 2

1.4 Interface 3

**CHAPTER 2**

**REQUIREMENT SPECIFICATION 4**

2.1 Software requirements 4

2.2 Hardware requirements 4

**CHAPTER 3**

**DESIGN 5**

3.1 Window design 5

3.2 Game board Design 6

3.3 Result Window 6

3.4 Flow of Control 7

**CHAPTER 4**

**IMPLEMENTATION 8**

4.1 Functions Used 8-9

**CHAPTER 5**

**SOURCE CODE……………………………………………………………………….**10

III

**CHAPTER 6**

**TEST CASES….……………………………………………………………………….24**

**CHAPTER 7**

**SNAPSHOTS.………………………………………………………………………….26**

**CHAPTER 8**

**CONCLUSION & FUTURE WORK 28**

**REFERENCES 29**

IV

**List of Figures**

**Figure No. Figure Name Page**

**No.**

Figure 3.1 Home Screen 5

Figure 3.2 Game display Window 6

Figure 3.3 Result Window 6

Figure 3.4 Flow of control 7

V

**List of Tables**

**Table No. Table Name Page**

**No.**

Table 6.1 Test cases for Mouse interface 24

Table 6.2 Test cases for Keyboard interface 24

Table 6.3 Test cases for ArrowKeyInterface 25

VI

**CHAPTER 1**

**INTRODUTION**

**1.1Aim**

The aim of this project is to develop the simple 3-D game simulator, which contains options like selecting the position in the game. And also stopping the game simulation. The interface should be user friendly and should use mouse and keyboard interface for the interaction with the user. The main goal is to develop a simple and very easy game with three-dimensional view and allow the user for its use and take feedback from user for further improvement.

**1.2 Introduction to OpenGL**

OpenGL is an open specification for an applications program interface for defining 2D and

3D objects. The specification 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 system using any OpenGL-adhering graphics adapter.

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’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.

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.

**GLUT**

GLUT, short for OpenGL Utility Toolkit, is a set of support libraries available on every major platform. OpenGL does not directly support any form of windowing, menus, or input. That’s where GLUT comes in. It provides basic functionality in all of those areas, while remaining platform independent, so that you can easily move GLUT-based applications from, for example, Windows to UNIX with few, if any, changes.

**1.3 Project Related Concepts**

The objective is to build a tik-tak-toe game simulator which can convince the audience about conditions of this game. The coding is implemented for the two players represent “cone” and “rectangle”, that play automatically one after the other. In this game importance is given to player choice and for his next move.

The basic requirements of the game simulator are analyzed to be:

1. User Interface- This game mainly built for two players this not it’s not just a relief but also amusing. The choice is for player to be 1st player or 2nd player

2. Start/ Stop Simulation – User can use the keyboard interface for playing and mouse interface to entre and exit.

**1.4 INTERFACE**

**Mouse interface**

This works on the home screen that has two sections “Start Game” and “Credits” with each

buttons respectively.

** Left Button**

Start the game or after completion of game again starts a new.

** Right Button**

On the double click of button it shows information about developer.

**Keyboard Interface**

Eleven functionalities are implemented using the keyboard function.

 After selecting position of pawn by user, if a user presses spacebar the pawn get placed.

 After starting the game, the user needs to use 4 types of **arrow keys** that is:

* UP ARROW 🡪 to move the pawn upwards.
* DOWN ARROW 🡪 to move the pawn downwards.
* LEFT ARROW 🡪 to move the pawn towards left.
* RIGHT ARROW 🡪 to move the pawn towards right.

 **Quit Game**

If the user clicks on the ‘X’ key, game will be paused and closed and moved to the home screen, if after the end of the game when result screen appears, to go back to home screen this button is used. This button is also used to exit or quit the window as well as game.

 **3D-ROTATION**

S/s and W/w 🡪 to rotate along X-axis

D/d and A/a 🡪 to rotate along Y-axis

E/e and Q/q 🡪 to rotate along Z-axis

 **Space** “ ” button 🡪 to place the pawn on distinct place.

**CHAPTER 2**

**REQUIREMENTS SPECIFICATION**

Visual Studio 2005 delivers on Microsoft’s vision of smart client applications by letting developers quickly create connected applications that deliver the highest quality rich user experiences. This new version lets any size organization create more secure, more manageable, and more reliable applications that take advantage of Windows Vista, windows7, 2007 Office System and the Web. By building these new types of applications, organizations will find it easier than ever to capture and analyze information so that they can make effective business decisions.

**2.1 Software Requirements**

 An MS-DOS based operating system like Windows 98, Windows 2000 or Windows XP, vista, windows 7 is the platform required to develop the 2D and 3D graphics applications.

 A Visual C/C++ compiler is required for compiling the source code to make the executable file which can then be directly executed.

 A built-in graphics library like glut and glut32, and header file like GL\glut.h and also dynamic link libraries like glut and glut32 are required for creating the 3D layout.

**2.2 Hardware Requirements**

The hardware requirements are very minimal and the software can run on most of the machines.

 Processor - Intel 486/Pentium processor or above.

 Processor Speed - 500 MHz or above

 RAM - 64MB or above Storage Space - 2 MB or above, hard disk -10MB.

 Monitor resolution - A color monitor with a minimum resolution of 1000\*700

 Supports both single &double buffering.

**CHAPTER 3**

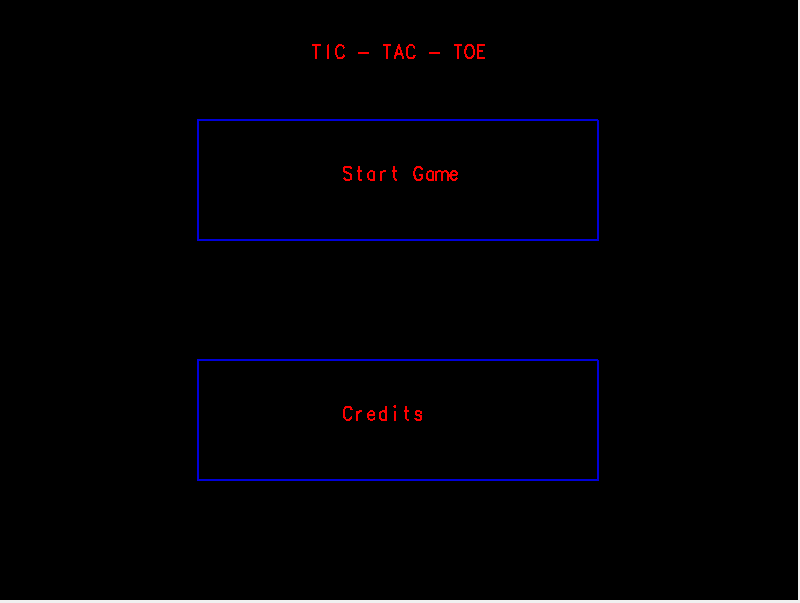
**DESIGN**

**3.1 Window design**

Game simulation has 3 windows. That is

 **Tic Tak Toe (Main window):** This window appears automatically on ./a.out and

contains “Start game” button and “Credits”. This is window used display next window that is game board or credits window that displays about the developer and user points after game is stored, on the start of new game this credits data destroys and start with new data but developer’s information remains same. And all mouse and keyboard events triggered in this window. All the labels and Information about the model will be displayed on this window.



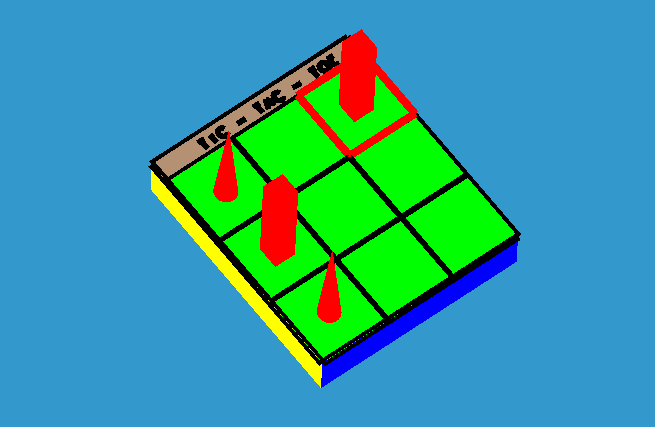
**Figure 3.2: Main Window**

**3.2 Game board display**

This displays a game board that provides three-dimensional view by providing the user buttons

to rotate in x, y, z axis and move their pawn on the board with help of arrows and place using

spacebar to come out of the game in middle of the game can be done using another button.



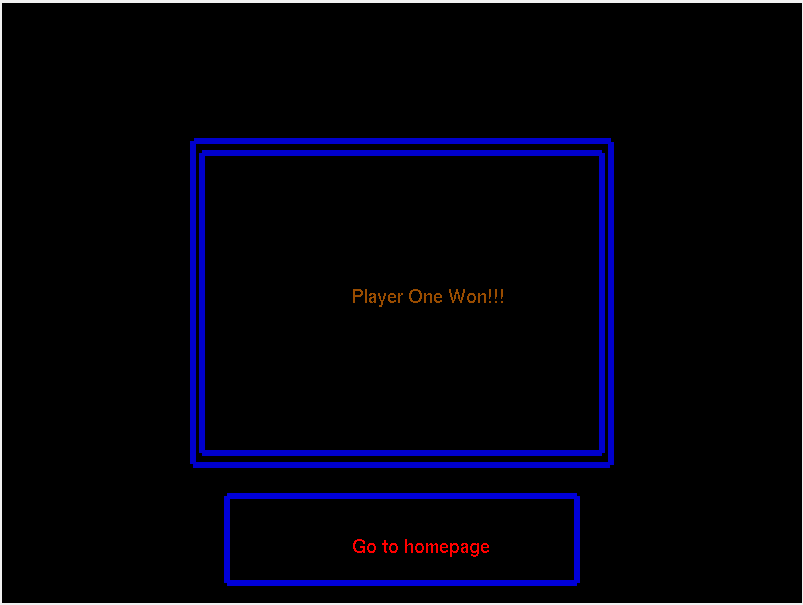
**Figure 3.1: game display window**

**3.3 Result board**

This window appears automatically once the match is draw or when of the winner is confirmed.

It displays either as “player one won” or “player two won” or “match dawn” upon the outcome

of the game. It indicates to go to homes screen if the user wants to start a new game or quit.



**Figure 3.2: Result Window**

./a.out

Home Screen

Credits

Start Game

Information about Developer

Play Game

Result

Quit

**Figure 3.4: Flow of Control**

**CHAPTER4**

**IMPLEMENTATION**

**4.1 Functions used**

**glutPostRedisplay(button):** Mark the normal plane of current window as needing to be redisplayed. The next iteration through glutMainLoop, the window's display callback will be called to redisplay the window's normal plane. Multiple calls to glutPostRedisplay before the next display callback opportunity generates only a single redisplay callback. glutPostRedisplay may be called within a window's display or overlay display callback to re-mark that window for redisplay.

**glutMouseFunc(args):** glutMouseFunc sets the mouse callback for the current window. When a user presses and releases mouse buttons in the window, each press and each release generates a mouse callback. The button parameter is one of glut\_left\_button, glut\_middle\_button, or glut\_right\_button. For systems with only two mouse buttons, it may not be possible to generate glut\_middle\_button callback. For systems with a single mouse button, it may be possible to generate only a glut\_left\_button callback. The state parameter is either glut\_up or glut\_down indicating whether the callback was due to a release or press respectively. The x and y callback parameters indicate the window relative coordinates when the mouse button state changed. If a glut\_down callback for a specific button is triggered, the program can assume a glut\_up callback for the same button will be generated (assuming the window still has a mouse callback registered) when the mouse button is released even if the mouse has moved outside the window.

**glutKeyboardFunc(args):** glutkeyboardfunc sets the keyboard callback for the current window. When a user types into the window, each key press generating an ASCII character will generate a keyboard callback. The key callback parameter is the generated ASCII character. The state of modifier keys such as Shift cannot be determined directly; their only effect will be on the returned ASCII data. The x and y callback parameters indicate the mouse location in window relative coordinates when the key was pressed.

When a new window is created, no keyboard callback is initially registered, and ASCII key strokes in the window are ignored. Passing NULL to glutkeyboardfunc disables the generation of keyboard callbacks. During a keyboard callback, glutGetModifiers may be called to determine the state of modifier keys when the keystroke generating the callback occur.

**glutDisplayFunc** (): glutDisplayFunc sets the display callback for the current window. When GLUT determines that the normal plane for the window needs to be redisplayed, the display callback for the window is called. Before the callback, the current window is set to the window needing to be redisplayed and (if no overlay display callback is registered) the layer in use is set to the normal plane. The display callback is called with no parameters. The entire normal plane region should be redisplayed in response to the callback (this includes ancillary buffers if your program depends on their state.

CHAPTER 5

SOURCE CODE

#include <GL/glut.h>

#include<string.h>

#define PLAYER\_ONE 0

#define PLAYER\_TWO 1

#define NO\_WINNER\_TILL\_NOW (-1)

void mainPage ();

int page=0;

// Points to draw board.

GLdouble boardPoints [][3] = {

{-100.0, -100.0, 10.0},

{100.0, -100.0, 10.0},

{100.0, -100.0, -10.0},

{-100.0, -100.0, -10.0},

{100.0, 100.0, 10.0},

{100.0, 100.0, -10.0},

{-100.0, 100.0, -10.0},

{-100.0, 100.0, 10.0}

};

// Points to draw lines on the board.

GLdouble lines [][3] = {

{-33.33, -100.0, 11.0},

{-33.33, 100.0, 11.0},

{33.33, -100.0, 11.0},

{33.33, 100.0, 11.0},

{-100.0, 33.33, 11.0},

{100.0, 33.33, 11.0},

{-100.0, -33.33, 11.0},

{100.0, -33.33, 11.0}

};

// Points to display heading.

GLdouble headingPoints [][3] = {

{-100.0, 100.0, 10.0},

{100.0, 100.0, 10.0},

{100.0, 100.0, -10.0},

{-100.0, 100.0, -10.0},

{100.0, 120.0, 10.0},

{100.0, 120.0, -10.0},

{-100.0, 120.0, -10.0},

{-100.0, 120.0, 10.0}

};

double boxPoints [4][3] = {

{-100, 100, 12},

{-33.33, 100, 12},

{-33.33, 33.33, 12},

{-100, 33.33, 12}

};

/\*\* 2D array to store progress of game.

\* -1 => Box is empty.

\* 0 => Cone is marked.

\* 1 => Cube is marked. \* /

int gameProgress [3][3] = {

{-1, -1, -1},

{-1, -1, -1},

{-1, -1, -1}

};

/\*

\* Variable to keep track of turn.

\* Even => Player 1.

\* Odd => Player 2.

\*/

int turn = 0;

// Player who won the game.

int winner = NO\_WINNER\_TILL\_NOW;

// Variables to identify which box is selected.

int row = 0;

int column = 0;

// Main window ID.

int mainWindow;

// Game window ID.

int gameWindow;

// Winner window ID.

int winnerWindow;

// Credits window ID.

int creditsWindow;

// Variable to indicate match drawn.

int draw = -1;

// Mouse function for home page.

void mainMouse (int button, int state, int x, int y);

// Keyboard function for home page.

void mainKeyboard (unsigned char key, int x, int y);

// Function to get input from arrow keys

void arrowKeyInput (int key, int xMouse, int yMouse);

// Function to open home page

void openMainWindow ();

void resetGame () {

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

gameProgress[i][j] = -1;

}

}

turn = 0;

winner = NO\_WINNER\_TILL\_NOW;

row = 0;

column = 0;

draw = -1;

}

void polygon (GLdouble (\*points) [3], int a, int b, int c, int d) {

glBegin (GL\_POLYGON);

glVertex3dv(points[a]);

glVertex3dv(points[b]);

glVertex3dv(points[c]);

glVertex3dv(points[d]);

glEnd ();

}

// Function to write text.

void writeStrokeText (char \*string, float x, float y, float z) {

glPushMatrix ();

glTranslatef (x, y, z + 1);

glScalef (0.09f, -0.08f, z);

glRotatef (180, 1, 0, 0);

glEnable (GL\_LINE\_SMOOTH);

for (char \*c = string; \*c != '\0'; c++) {

glutStrokeCharacter (GLUT\_STROKE\_MONO\_ROMAN, \*c);

}

glPopMatrix ();

}

void writeNormalText (char \*string, float x, float y) {

glRasterPos2f (x, y);

for (char \*c = string; \*c != '\0'; c++) {

glutBitmapCharacter (GLUT\_BITMAP\_HELVETICA\_18, \*c);

}

}

void moveBox () {

if (winner == NO\_WINNER\_TILL\_NOW && turn < 9) {

// Move row.

switch (row) {

case 1:

glTranslated (0.0, -66.66, 0.0);

break;

case 2:

glTranslated (0.0, -66.66 \* 2, 0.0);

break;

default:

break;

}

// Move column.

switch (column) {

case 1:

glTranslated(66.66, 0.0, 0.0);

break;

case 2:

glTranslated(66.66 \* 2, 0.0, 0.0);

break;

default:

break;

}

}

}

void drawCone(int row, int column) {

glPushMatrix();

glTranslatef((GLfloat) ((column - 1) \* 66.66), (GLfloat) ((row - 1) \* -66.66), 10);

glScalef(10, 10, 50);

glutSolidCone(1, 1, 36, 1);

glPopMatrix();

}

void drawCube(int row, int column) {

glPushMatrix();

glTranslatef((GLfloat) ((column - 1) \* 66.66), (GLfloat) ((row - 1) \* -66.66), 35);

glScalef(10, 10, 25);

glutSolidCube(2);

glPopMatrix();

}

// Function to select the box on the board.

void drawBoxOnSelected() {

glColor3f(1, 0, 0);

if (winner == NO\_WINNER\_TILL\_NOW && turn < 9) {

glLineWidth(7);

glPushMatrix();

moveBox();

glBegin(GL\_LINE\_LOOP);

glVertex3dv(boxPoints[0]);

glVertex3dv(boxPoints[1]);

glVertex3dv(boxPoints[2]);

glVertex3dv(boxPoints[3]);

glEnd();

glPopMatrix();

}

}

void drawBoard() {

// Board.

glColor3f(0, 0, 1);

polygon(boardPoints, 0, 1, 2, 3);

glColor3f(0, 1, 0);

polygon(boardPoints, 0, 1, 4, 7);

glColor3f(1, 0, 0);

polygon(boardPoints, 1, 2, 5, 4);

glColor3f(1, 0, 1);

polygon(boardPoints, 4, 5, 6, 7);

glColor3f(0, 1, 1);

polygon(boardPoints, 2, 3, 6, 5);

glColor3f(1, 1, 0);

polygon(boardPoints, 0, 3, 6, 7);

// Heading.

glColor3f(0, 0, 1);

polygon(headingPoints, 0, 1, 2, 3);

glColor3f(0.71, 0.57, 0.45);

polygon(headingPoints, 0, 1, 4, 7);

glColor3f(1, 0, 0);

polygon(headingPoints, 1, 2, 5, 4);

glColor3f(1, 0, 1);

polygon(headingPoints, 4, 5, 6, 7);

glColor3f(0, 1, 1);

polygon(headingPoints, 2, 3, 6, 5);

glColor3f(1, 1, 0);

polygon(headingPoints, 0, 3, 6, 7);

glColor3f(0, 0, 0);

// Border for heading.

glLineWidth(7);

glBegin(GL\_LINE\_LOOP);

glVertex3i(-100, 100, 10);

glVertex3i(100, 100, 10);

glVertex3i(100, 120, 10);

glVertex3i(-100, 120, 10);

glEnd();

// Border for board.

glBegin(GL\_LINE\_LOOP);

glVertex3i(-100, 100, 10);

glVertex3i(-100, -100, 10);

glVertex3i(100, -100, 10);

glVertex3i(100, 100, 10);

glEnd();

// Draw lines on board.

glLineWidth(5);

for (int i = 0; i < 8; i += 2) {

glBegin(GL\_LINES);

glVertex3dv(lines[i]);

glVertex3dv(lines[i + 1]);

glEnd();

}

// Heading

writeStrokeText("TIC - TAC - TOE", -60, 105, 10);

drawBoxOnSelected();

// Indicate whose turn to play.

if (turn < 9 && winner == NO\_WINNER\_TILL\_NOW) {

if (turn % 2)

drawCube(row, column);

else

drawCone(row, column);

}

// Fill boxes which are already marked.

for (int row = 0; row < 3; row++) {

for (int column = 0; column < 3; column++) {

if (gameProgress[row][column] == PLAYER\_ONE) {

drawCone(row, column);

} else if (gameProgress[row][column] == PLAYER\_TWO) {

drawCube(row, column);

}

}

}

}

int checkForWinner() {

// Check row

for (int i = 0; i < 3; i++) {

if (gameProgress[i][0] == gameProgress[i][1]

&& gameProgress[i][0] == gameProgress[i][2]

&& gameProgress[i][0] != -1) {

return gameProgress[i][0];

}

}

// Check column

for (int i = 0; i < 3; i++) {

if (gameProgress[0][i] == gameProgress[1][i]

&& gameProgress[0][i] == gameProgress[2][i]

&& gameProgress[0][i] != -1) {

return gameProgress[0][i];

}

}

// Check diagonal

if (gameProgress[0][0] == gameProgress[1][1]

&& gameProgress[0][0] == gameProgress[2][2]

&& gameProgress[0][0] != -1) {

return gameProgress[0][0];

}

// Check diagonal

if (gameProgress[0][2] == gameProgress[1][1]

&& gameProgress[0][2] == gameProgress[2][0]

&& gameProgress[0][2] != -1) {

return gameProgress[0][2];

}

// Check for draw

for (int i = 0; i < 3; i++) {

for (int j = 0; j < 3; j++) {

if (gameProgress[i][j] == -1)

return -1;

}

}

draw = 1;

return -10;

}

// Display function for game window

void gameDisplay() {

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

drawBoard();

glutSwapBuffers();

}

void gameInit() {

glClearColor(0.2, 0.6, 0.8, 0);

glOrtho(-320, 320, -180, 180, -250, 250);

glRotatef(-40, 1, 0, 0);

}

void winnerInit() {

glClearColor(0.0, 0.0, 0.0, 0.0);

glClear(GL\_COLOR\_BUFFER\_BIT);

glOrtho(0, 640, 0, 360, -50, 50);

glutSwapBuffers();

}

// Display function to display winner

void playerWinDisplay() {

glColor3f(0.0, 0.0, 0.8);

glLineWidth(6);

glBegin(GL\_LINE\_LOOP);

glVertex2i(153, 277);

glVertex2i(153, 83);

glVertex2i(487, 83);

glVertex2i(487, 277);

glEnd();

glBegin(GL\_LINE\_LOOP);

glVertex2i(160, 270);

glVertex2i(160, 90);

glVertex2i(480, 90);

glVertex2i(480, 270);

glEnd();

glColor3f(0.6, 0.3, 0.0);

if (winner == PLAYER\_ONE) {

writeNormalText("Player One Won!!!", 280, 180);

} else if (winner == PLAYER\_TWO) {

writeNormalText("Player Two Won!!!", 280, 180);

} else if (draw == 1) {

writeNormalText("Match Draw!!!", 280, 180);

}

glColor3f(0, 0, 0.85);

// Button to go to home page

glBegin(GL\_LINE\_LOOP);

glVertex2i(180, 64);

glVertex2i(460, 64);

glVertex2i(460, 12);

glVertex2i(180, 12);

glEnd();

glColor3f(1, 0, 0);

writeNormalText("Go to homepage", 280, 30);

glutSwapBuffers();

}

void playerWinKeyboard(unsigned char key, int x, int y) {

switch (key) {

case 'X':

case 'x':

glutDestroyWindow(winnerWindow);

openMainWindow();

break;

default:

break;

}

}

void mainDisplay() {

glClearColor(0, 0, 0, 0);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glColor3f(0, 0, 0.85);

if(page==0)

{

mainPage();

glLoadIdentity();

glMatrixMode(GL\_PROJECTION);

gluOrtho2D(0, 640, 0, 360);

//glFlush();

}

if(page==1){

// First button

glBegin(GL\_LINE\_LOOP);

glVertex2i(160, 288);

glVertex2i(480, 288);

glVertex2i(480, 216);

glVertex2i(160, 216);

glEnd();

// Second button

glBegin(GL\_LINE\_LOOP);

glVertex2i(160, 144);

glVertex2i(480, 144);

glVertex2i(480, 72);

glVertex2i(160, 72);

glEnd();

glColor3f(1, 0, 0);

writeStrokeText("Instruction", 275, 108, 0);

writeStrokeText("Start Game", 275, 252, 0);

writeStrokeText("TIC - TAC - TOE", 250, 325, 0);

}

glutSwapBuffers();

}

void playerWinMouse(int button, int state, int x, int y) {

float newX = (float) (640.0 / 1366) \* x;

float newY = (float) (360.0 / 768) \* y;

if (newX < 480 && newX > 160 && (360 - newY) < 64 && (360 - newY) > 12) {

mainWindow = glutCreateWindow("Tic Tac Toe - 3D");

glutDestroyWindow(winnerWindow);

glutFullScreen();

gluOrtho2D(0, 640, 0, 360);

glutDisplayFunc(mainDisplay);

glutKeyboardFunc(mainKeyboard);

glutMouseFunc(mainMouse);

}

}

void openMainWindow() {

mainWindow = glutCreateWindow("Tic Tac Toe - 3D");

glutFullScreen();

gluOrtho2D(0, 640, 0, 360);

glutDisplayFunc(mainDisplay);

glutKeyboardFunc(mainKeyboard);

glutMouseFunc(mainMouse);

}

void gameKeyboard(unsigned char key, int x, int y) {

switch (key) {

// Rotate in x axis

case 'S':

case 's':

glRotatef(5, 1, 0, 0);

break;

case 'W':

case 'w':

glRotatef(-5, 1, 0, 0);

break;

// Rotate in y axis

case 'D':

case 'd':

glRotatef(5, 0, 1, 0);

break;

case 'A':

case 'a':

glRotatef(-5, 0, 1, 0);

break;

// Rotate in z axis

case 'Q':

case 'q':

glRotatef(5, 0, 0, 1);

break;

case 'E':

case 'e':

glRotatef(-5, 0, 0, 1);

break;

case ' ':

if (turn <= 9 && gameProgress[row][column] == -1 && winner == NO\_WINNER\_TILL\_NOW) {

if ((turn % 2) == PLAYER\_ONE) {

drawCone(row, column);

} else {

drawCube(row, column);

}

if (gameProgress[row][column] == -1) {

gameProgress[row][column] = (turn % 2);

winner = checkForWinner();

// Display winner

if (winner != NO\_WINNER\_TILL\_NOW) {

winnerWindow = glutCreateWindow("Game Over!");

glutSetWindow(winnerWindow);

glutDestroyWindow(gameWindow);

glutFullScreen();

winnerInit();

glutDisplayFunc(playerWinDisplay);

glutKeyboardFunc(playerWinKeyboard);

glutMouseFunc(playerWinMouse);

}

}

turn++;

}

break;

case 'X':

case 'x':

glutDestroyWindow(gameWindow);

openMainWindow();

break;

default:

break;

}

glutPostRedisplay();

}

float height = 0;

void creditsDisplay() {

glClearColor(0, 0, 0, 0);

glClear(GL\_COLOR\_BUFFER\_BIT | GL\_DEPTH\_BUFFER\_BIT);

glColor3f(1, 1, 1);

writeNormalText("Instruction", 210, height - 20);

writeNormalText("start game --> right click on mouse", 210, height - 40);

writeNormalText("move pawn up, down, left, right --> 'use four arrow keys' ", 210, height - 60);

writeNormalText("place pawn on distict position --> 'space key' ", 210, height - 80);

writeNormalText("rotate game board x-axis --> 'W/w and S/s keys'",210, height - 100);

writeNormalText("rotate game board y-axis --> 'D/d and A/a keys'", 210, height - 120);

writeNormalText("rotate game board z-axis --> 'Q/q and E/e keys'", 210, height - 140);

writeNormalText("quit window or game --> 'X/x key'", 210, height - 160);

if (height <= 220) {

height += 1;

}

glutPostRedisplay();

glutSwapBuffers();

}

void creditsKeyboard(unsigned char key, int x, int y) {

switch (key) {

case 'X':

case 'x':

// Reset height to 0, to get scrolling animation again when it's opened.

height = 0;

glutDestroyWindow(creditsWindow);

openMainWindow();

break;

default:

break;

}

}

void mainMouse(int button, int state, int x, int y) {

float newX = (float) (640.0 / 1366) \* x;

float newY = (float) (360.0 / 768) \* y;

if (newX < 480 && newX > 160 && (360 - newY) < 288 && (360 - newY) > 216) {

// Start game button

gameWindow = glutCreateWindow("Tic Tac Toe - 3D");

glutDestroyWindow(mainWindow);

glutFullScreen();

resetGame();

glutDisplayFunc(gameDisplay);

gameInit();

glutKeyboardFunc(gameKeyboard);

glutSpecialFunc(arrowKeyInput);

glEnable(GL\_DEPTH\_TEST);

} else if (newX < 480 && newX > 160 && (360 - newY) < 144 && (360 - newY) > 72) {

// Credits button

creditsWindow = glutCreateWindow("Credits");

glutDestroyWindow(mainWindow);

glutFullScreen();

gluOrtho2D(0, 640, 0, 360);

glutDisplayFunc(creditsDisplay);

glutKeyboardFunc(creditsKeyboard);

}

}

void mainKeyboard(unsigned char key, int x, int y) {

switch (key) {

case 'X':

case 'x':

glutDestroyWindow(mainWindow);

case 'p':

page=1;

mainDisplay();

break;

default:

break;

}

}

// Function to get input from arrow keys

void arrowKeyInput(int key, int xMouse, int yMouse) {

switch (key) {

case GLUT\_KEY\_UP:

if (row > 0)

row -= 1;

break;

case GLUT\_KEY\_DOWN:

if (row < 2)

row += 1;

break;

case GLUT\_KEY\_LEFT:

if (column > 0)

column -= 1;

break;

case GLUT\_KEY\_RIGHT:

if (column < 2)

column += 1;

break;

default:

break;

}

drawBoxOnSelected();

glutPostRedisplay();

}

void print(int x,int y, unsigned char \*string, void \*font)

{

int len,k;

glRasterPos2f(x,y);

len=strlen(string);

for (int i = 0; i < len; i++) {

glutBitmapCharacter(GLUT\_BITMAP\_8\_BY\_13, string[i]);}

}

void mainPage()

{

glClear(GL\_COLOR\_BUFFER\_BIT|GL\_DEPTH\_BUFFER\_BIT); //clear the window

glColor3f(0,1,0); //text color's.

print(100,330,"SAI VIDYA INSTITUTE OF TECHNOLOGY ,Rajanukunte, Bengaluru- 560 064",GLUT\_BITMAP\_TIMES\_ROMAN\_24); //display

glColor3f(1,0,1); //text color's.

print(1,310, " (Affiliated to Visvesvaraya Technological University, Belagavi | Recognized by Govt. of Karnataka ",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

print(6,300, "| Approved by AICTE, New Delhi) ",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0,0,1);

print(150,270,"DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(1,0,0);

print(150,200,"COMPUTER GRAPHICS - TIK-TAK-TOE GAME SIMULATION",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.7,0,1);

print(350,70,"Under the guidance of :",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(1,1,1);

print(300,50," Dr.Sangeetha.V - ",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(1,1,1);

print(300,40," Sunil G L - ",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(1,1,1);

print(420,50,"Associate Professor, Dept. of ISE",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(1,1,1);

print(380,40,"Assistant Professor, Dept. of CSE",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(1,1,0);

print(10,50,"Harshita T - 1VA17CS015",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(1,1,0);

print(10,40,"Jagbeer Poonia - 1VA17CS018",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

glColor3f(0.5,1,0.1);

print(20,150,"(Press p to start)",GLUT\_BITMAP\_TIMES\_ROMAN\_24);

}

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

glutInit(&argc, argv);

glutInitDisplayMode(GLUT\_DOUBLE | GLUT\_RGB | GLUT\_DEPTH);

mainWindow = glutCreateWindow("Tic Tac Toe - 3D");

glutFullScreen();

gluOrtho2D(0, 640, 0, 360);

glutDisplayFunc(mainDisplay);

glutKeyboardFunc(mainKeyboard);

glutMouseFunc(mainMouse);

glutMainLoop();

}

**CHAPTER 6**

**TESTING**

**6.1 TEST CASES**

**Table 6.1: Test cases for Mouse interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.no** | **Functionality** | **Comments** | **Remarks** |
| 1. | Mouse Right click | It starts the game | Pass |
| 2. | Mouse Left click | Gives information about developer-Credits | Pass |

**Table 6.2: Test case for Keyboard Interface**

|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.no** | **Functionality** | **Comments** | **Remarks** |
| 1. | S/s button | Rotate along X axis | Pass |
| 2. | W/w button | Rotate along X axis | Pass |
| 3. | D/d button | Rotate along Y axis | Pass |
| 4. | A/a button | Rotate along Y axis | Pass |
| 5. | Q/q button | Rotate along Z axis | Pass |
| 6. | E/e button | Rotate along Z axis | Pass |
| 7. | X button | Exit the window or destroy the window | Pass |

**Table 6.3: Test case for ArrowKeyInput**

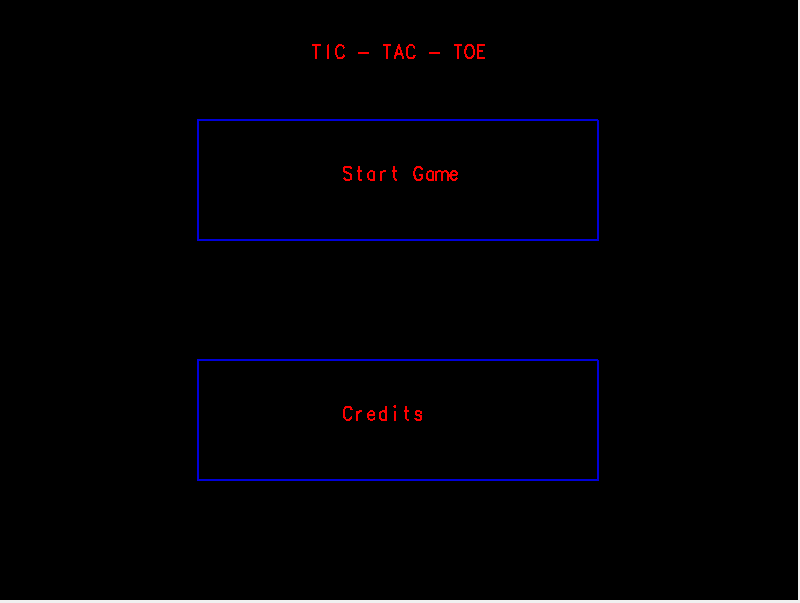
|  |  |  |  |
| --- | --- | --- | --- |
| **Sl.no** | **Functionality** | **Comments** | **Remarks** |
| 1. | Arrow Up | Move upward on the board | Pass |
| 2. | Arrow Down | Move downward on the board | Pass |
| 3. | Arrow Left | Move towards left | Pass |
| 4. | Arrow Right | Move towards right | Pass |
| 5. | Space button | To place the pawn on distinct cell | Pass |

CHAPTER 7

SNAPSHOTS

The below window displays Home Screen of tik-tak-toe game that consist of two

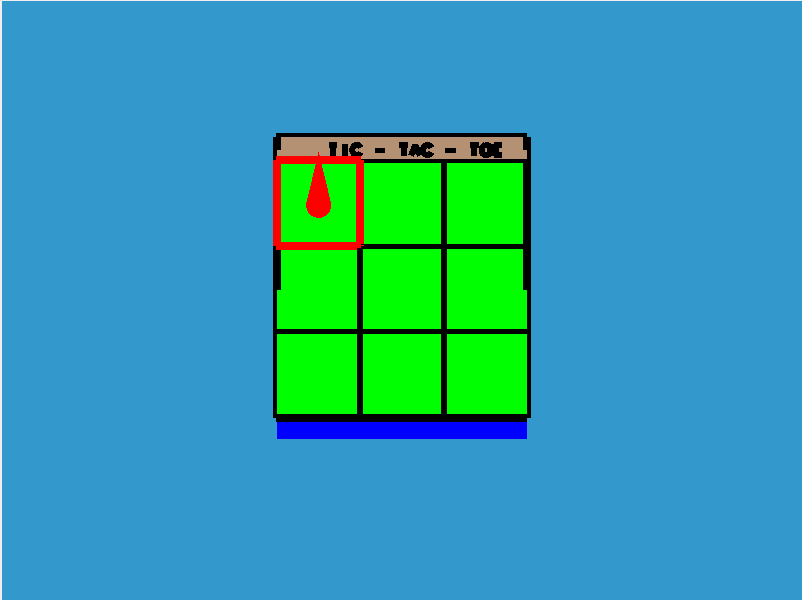
option using buttons of mouse that leads to next window.



7.1 Snapshot of home screen.

The below window is the second window that appears on the click of start game

that contains game board where players can start their game.

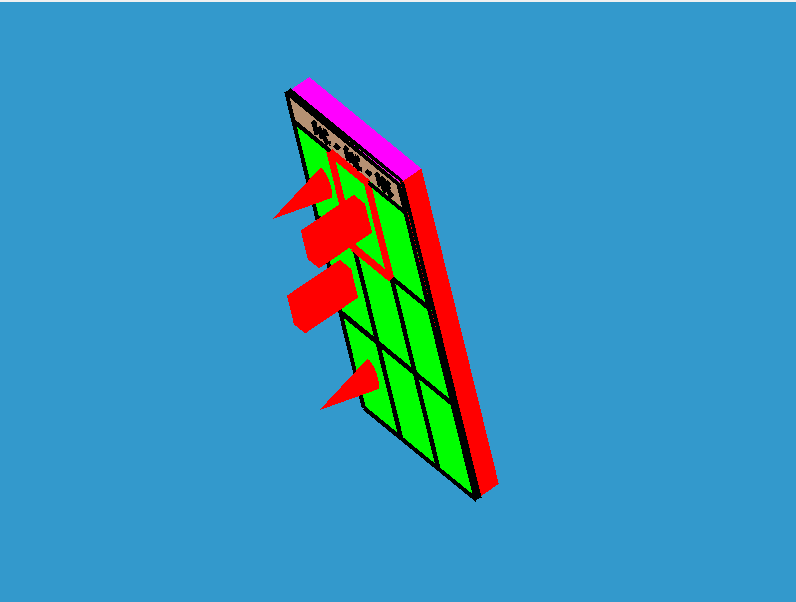


7.2 Snapshot of tik-tak-toe game board

The below window is the second window that shows three-dimensional view of

the game board which can be rotated in all three directions i.e x, y, z plane using

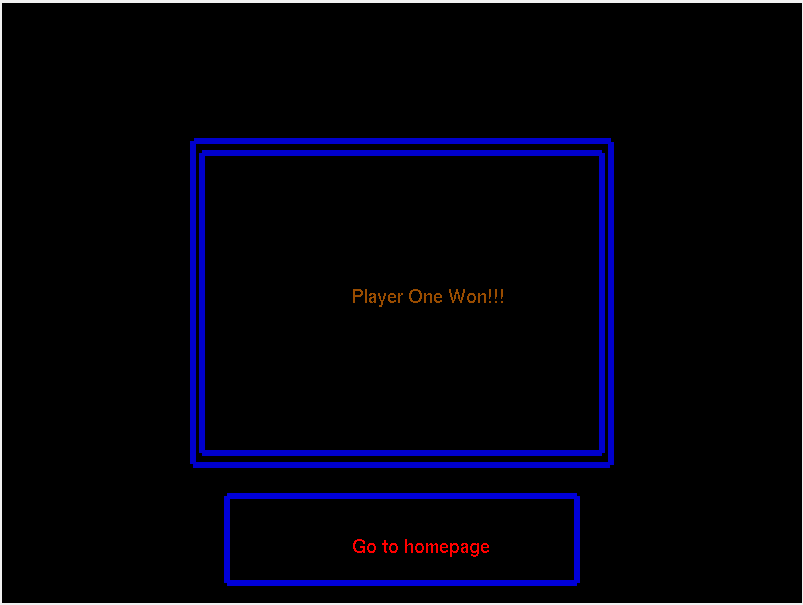
keys on keyboard.



7.3 Snapshot of 3D-game simulation

This is the last screen shown below that displays the winner and returns to home

screen either to exit or start a new game.



` 7.4 Snapshot on result of the game

**CHAPTER 8**

**CONCLUSION**

This tik-tak-toe game simulation is very good project. Users doesn’t have any complexity to understand. The interface is mouse driven and the user can select a function by clicking “start” button. And also, the interface supports keyboard interface which is mainly to move the pawn on the board of 3x3 using arrows i.e up, down, left, right and space to place on the board. We have also provided 3D effect of the board by provided to view that with four buttons on keyboard i.e “S/s & W/w” 🡪 to rotate along X-axis, “D/d & A/a” 🡪to rotate along Y-axis, “Q/q & E/e” 🡪 to rotate along Z-axis. We have tried our best to make this simulator very realistic, so that user can without any problem play the game.

**FUTURE ENHANCEMENTS**

The following are some of the features that are planned to be supported in the future versions of the atom simulator.

 In credit 🡪 adding points of the player so that wen he/she come back it remains same and add on to it.

 Features like change in color on every move of pawn.

 Adding a player choice to choose the pawn to be first or second player.

 Making the simulation in 4-D.

**REFERENCES**

[1] Interactive Computer Graphics A Top-Down Approach with OpenGL -Edward

Angel, 5th Edition, Addison-Wesley, 2008.

[2] Computer Graphics Using OpenGL – F.S. Hill Jr. 2nd Edition, Pearson Education,

2001.

[3] Computer Graphics – James D Foley, Andries Van Dam, Steven K Feiner, John F Hughes, Addison-Wesley 1997.

[4] Computer Graphics - OpenGL Version – Donald Hearn and

[5] Pauline Baker, 2nd Edition, Pearson Education, 2003. [6] [www.google.com](http://www.google.com/)

[7] [www.openglforum.org](http://www.openglforum.org/)