CHAPTER 1

# INTRODUCTION TO OPENGL

## 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.

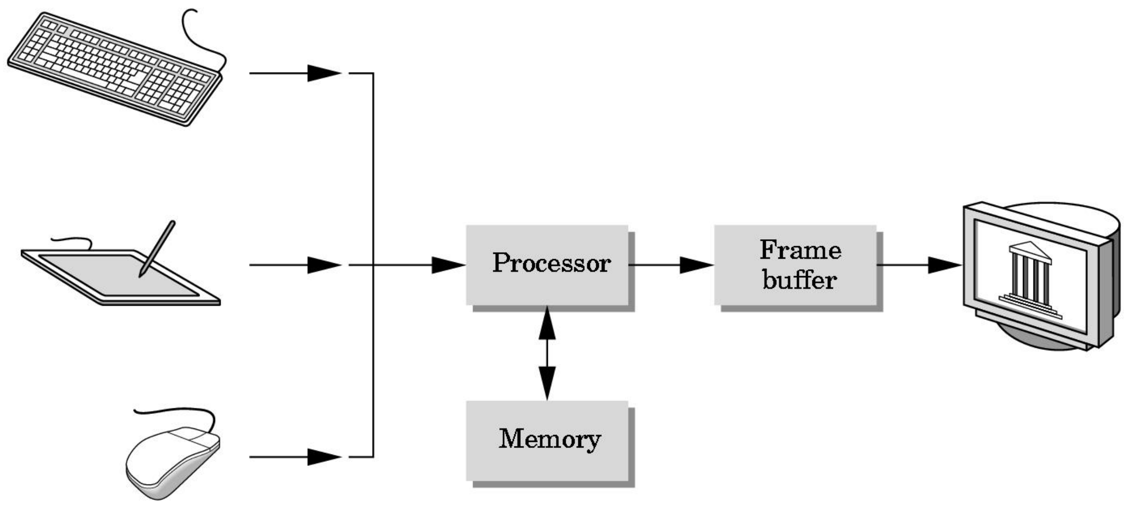


Fig 1.1: Rendering Graphics

## 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.

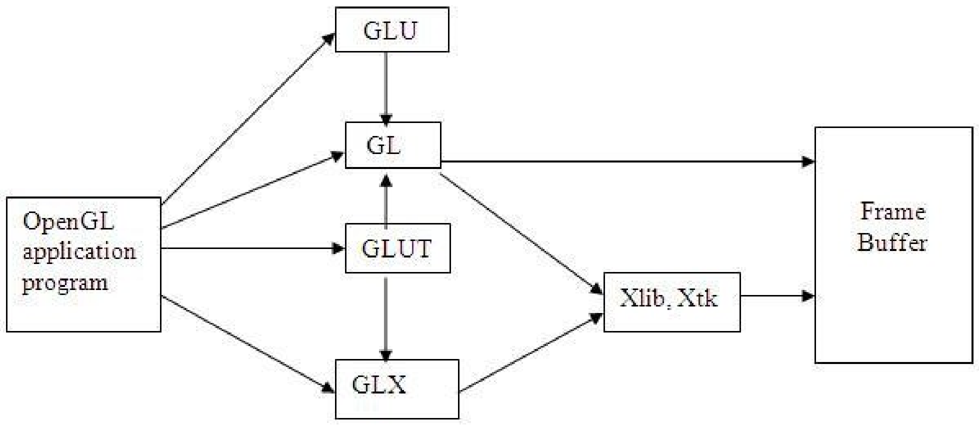


Fig 1.2: OpenGL library

**CHAPTER 2**

# INTRODUCTION TO PROJECT

The development of a Tic Tac Toe game using OpenGL presents an engaging opportunity to explore the intricacies of game design while leveraging the capabilities of OpenGL graphics rendering. Commencing with the setup of the OpenGL environment, meticulous attention is paid to ensuring the correct installation and configuration of requisite libraries on the chosen development platform. Subsequently, the focal point of the project, the game board itself, is established. Tic Tac Toe's inherent simplicity is encapsulated within a 3x3 grid, where players strategically position their marks, denoted by X or O. Utilizing OpenGL primitives, the grid is meticulously rendered, imparting a visual identity to the playing field. Facilitating player interaction is paramount to the gaming experience. Robust user input handling mechanisms are implemented, empowering players to execute moves via mouse clicks or keyboard inputs. This necessitates the translation of input coordinates into corresponding grid positions, seamlessly bridging the gap between virtual actions and physical inputs. The game logic layer assumes a pivotal role in determining the outcome of each move and advancing the game state accordingly. Whether it entails verifying win conditions or preventing stalemates, the logic layer serves as the backbone of the game's functionality. As gameplay unfolds, the prowess of OpenGL's rendering capabilities becomes apparent, as it breathes life into the game board, adorning it with X's and O's with finesse. From the graceful curves of O to the bold strokes of X, each mark contributes to the unfolding drama of the game. Upon achieving victory, a triumphant animation elegantly traverses the winning line, celebrating the player's strategic acumen. Conversely, in instances of a draw, the game gracefully acknowledges the outcome with understated poise. The journey does not culminate with the completion of the core game mechanics. Rather, it beckons towards further refinement and enhancement. Features such as a scoreboard, facilitating win-loss tracking, or the integration of artificial intelligence to offer challenging single-player experiences, serve to elevate the gameplay. With OpenGL's inherent flexibility, customization options abound, allowing for the meticulous tailoring of visual elements, from textures and colors to animations and effects. Through rigorous testing and iterative refinement, the project evolves, with each bug addressed and feature polished. The culmination of these efforts yields a polished product, poised for presentation to a broader audience.

# CHAPTER 3

# SYSTEM REQUIREMENT SPECIFICATIONS

## 3.1 Hardware Requirements

The standard output device is assumed to be a Color Monitor. It is quite essential for any graphics package to have this, as provision of color options to the user is a must. The mouse, the main input device, has to be functional i.e. used to give input in the game. A keyboard is used for controlling and inputting data in the form of characters, numbers i.e. to change the user views . Apart from these hardware requirements there should be sufficient hard disk space and primary memory available for proper working of the package to execute the program. Pentium III or higher processor, 16MB or more RAM. A functional display card. Minimum Requirements expected are cursor movement, creating objects like lines, squares, rectangles, polygons, etc. Transformations on objects/selected area should be possible. Filling of area with the specified color should be possible.

## 3.2 Software Requirements

The editor has been implemented on the OpenGL platform and mainly requires an appropriate version of eclipse compiler to be installed and functional in ubuntu. Though it is implemented in OpenGL, it is very much performed and independent with the restriction, that there is support for the execution of C and C++ files. Text Modes is recommended.

# CHAPTER 4

# SYSTEM DESIGN:

## 4.1 Existing System:

Existing system for a graphics is the C++. This system will support only the 2D graphics. 2D graphics package being designed should be easy to use and understand. It should provide various option such as free hand drawing, line drawing, polygon drawing, filled polygons, flood fill, translation, rotation, scaling, clipping etc. Even though these properties were supported, it was difficult to render 2D graphics cannot be. Very difficult to get a 3Dimensional object. Even the effects like lighting, shading cannot be provided. So, we go for Eclipse software.

## 4.2 Proposed System:

To achieve three dimensional effects, OpenGL software is proposed. It is software which provides a graphical interface. It is a interface between application program and graphics hardware. the advantages are:

1. OpenGL is designed as a streamlined.
2. It is a hardware independent interface i.e. it can be implemented on many different hardware platforms.
3. With OpenGL, we can draw a small set of geometric primitives such as points, lines and polygons etc.
4. It provides double buffering which is vital in providing transformations.
5. It is event driven software.
6. It provides call back function.

CHAPTER 5

# IMPLEMENTATION

## 5.1 Functions Used:

Void glColor3f(float red, float green, float blue);

This function is used to mention the color in which the pixel should appear. The number 3 specifies the number of arguments that the function would take. ‘f ’ gives the type that is float. The arguments are in the order RGB(Red, Green, Blue). The color of the pixel can be specified as the combination of these 3 primary colors.

Void glClearColor(int red, int green, int blue, int alpha);

This function is used to clear the color of the screen. The 4 values that are passed as arguments for this function are (RED, GREEN, BLUE, ALPHA) where the red green and blue components are taken to set the background color and alpha is a value that specifies depth of the window. It is used for 3D images.

Void glutKeyboardFunc();

void glutKeyboardFunc(void (*func)(unsigned char key, int x, int y));

where func is the new keyboard callback function. 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 keycallback parameter is the generated ASCII character. 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.

Void glFlush();

Different GL implementations buffer commands in several different locations, including network buffers and the graphics accelerator itself. glFlush empties all of these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in finite time.

Void glMatrixMode(GLenum mode);

where mode Specifies which matrix stack is the target for subsequent matrix operations. Three values are accepted: GL_MODELVIEW, GL_PROJECTION, and GL_TEXTURE. The initial value is GL_MODELVIEW. glMatrixMode sets the current matrix mode. modecan assume one of three values:

GL_MODELVIEW

Applies subsequent matrix operations to the modelview matrix stack.

GL_PROJECTION

Applies subsequent matrix operations to the projection matrix stac

void glViewport(GLint x, GLint y, GLsizei width, GLsizei height)

where x, y Specify the lower left corner of the viewport rectangle, in pixels. The initial value is (0, 0). width, height Specify the width and height of the viewport. When a GL context is first attached to a surface (e.g. window), width and height are set to the dimensions of that surface. glViewport specifies the affine transformation of x and y from normalized device coordinates to window coordinates. Let ($x_{nd}$, $y_{nd}$) be normalized device coordinates. Then the window coordinates ($x_w$, $y_w$) are computed as follows:

$x_w = ( x_{nd} + 1 ) {}^{width}/_2 + x$ $y_w = ( y_{nd} + 1 ) {}^{height}/_2 + y$

Viewport width and height are silently clamped to a range that depends on the implementation. To query this range, call glGetInteger with argument GL_MAX_VIEWPORT_DIMS.

void glutInit(int *argcp, char **argv);

glutInit will initialize the GLUT library and negotiate a session with the window system. During this process, glutInitmay cause the termination of the GLUT program with an error message to the user if GLUT cannot be properly initialized. Examples of this situation include the failure to connect to the window system, the lack of window system support for OpenGL, and invalid command line options.glutInit also processes command line options, but the specific options parse are window system dependent.

void glutReshapeFunc(void (*func)(int width, int height));

glutReshapeFunc sets the reshape callback for the current window. The reshape callback is triggered when a window is reshaped. A reshape callback is also triggered immediately before a window's first display callback after a window is created or whenever an overlay for the window is established. The width andheight parameters of the callback specify the new window size in pixels. Before the callback, the current window is set to the window that has been reshaped. If a reshape callback is not registered for a window or NULL is passed to glutReshapeFunc (to deregister a previously registered callback), the default reshape callback is used. This default callback will simply

void glutMainLoop(void);

glutMainLoop enters the GLUT event processing loop. This routine should be called at most once in a GLUT program.

## 5.2 Detailed Design:

Transformation Functions:

Matrices allow arbitrary linear transformations to be represented in a consistent format, suitable for computation. This also allows transformations to be concatenated easily (by multiplying their matrices).

Linear transformations are not the only ones that can be represented by matrices. Using homogenous coordinates, both affine transformation and perspective projection on $R^n$ can be represented as linear transformations on $RP^{n+1}$ (that is, n+1- dimensional real projective space). For this reason, 4x4 transformation matrices are widely used in 3D computer graphics.

3-by-3 or 4-by-4 transformation matrices containing homogeneous coordinates are often called, somewhat improperly, "homogeneous transformation matrices". However, the transformations they represent are, in most cases, definitely non-homogeneous and non- linear (like translation, roto-translation or perspective projection). And even the matrices themselves look rather heterogeneous, i.e. composed of different kinds of elements (see below). Because they are multi-purpose transformation matrices, capable of representing both affine and projective transformations, they might be called "general transformation matrices", or, depending on the application, "affine transformation" or "perspective projection" matrices. Moreover, since the homogeneous coordinates describe a projective vector space, they can also be called "projective space transformation matrices".

Finding the matrix of a transformation

If one has a linear transformation T(x) in functional form, it is easy to determine the transformation matrix A by simply transforming each of the vectors of the standard basis by T and then inserting the results into the columns of a matrix. In other words,

$$\mathbf{A} = \begin{bmatrix} T(\vec{e}_1) & T(\vec{e}_2) & \cdots & T(\vec{e}_n) \end{bmatrix}$$

For example, the function T(x) = 5x is a linear transformation. Applying the above process (suppose that n = 2 in this case) reveals that

$$T(\vec{x}) = 5\vec{x} = \begin{bmatrix} 5 & 0 \\ 0 & 5 \end{bmatrix} \vec{x}$$

Examples in 2D graphics:

Most common geometric transformations that keep the origin fixed are linear, including rotation, scaling, shearing, reflection, and orthogonal projection; if an affine transformation is not a pure translation it keeps some point fixed, and that point can be chosen as origin to make the transformation linear. In two dimensions, linear transformations can be represented using a 2×2 transformation matrix.

## Rotation

For rotation by an angle θ anticlockwise about the origin, the functional form is x' = xcosθ − ysinθ and y' = xsinθ + ycosθ. Written in matrix form, this becomes:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

Similarly, for a rotation clockwise about the origin, the functional form is x' = xcosθ + ysinθ and y' = − xsinθ + ycosθ and the matrix form is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

## Scaling

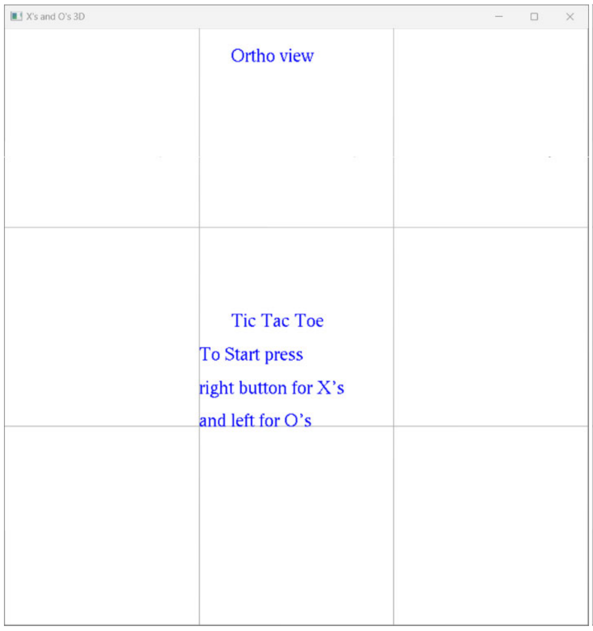
For scaling (that is, enlarging or shrinking), we have $x' = s_x \cdot x$ and $y' = s_y \cdot y$. The matrix form is:

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

When $s_x s_y = 1$,then the matrix is a squeeze mapping and preserves areas in the plane.

# CHAPTER 6

## SNAP SHOTS:



Ortho view

Tic Tac Toe
To Start press
right button for X's
and left for O's

Fig 6.1: Instructions before Starting

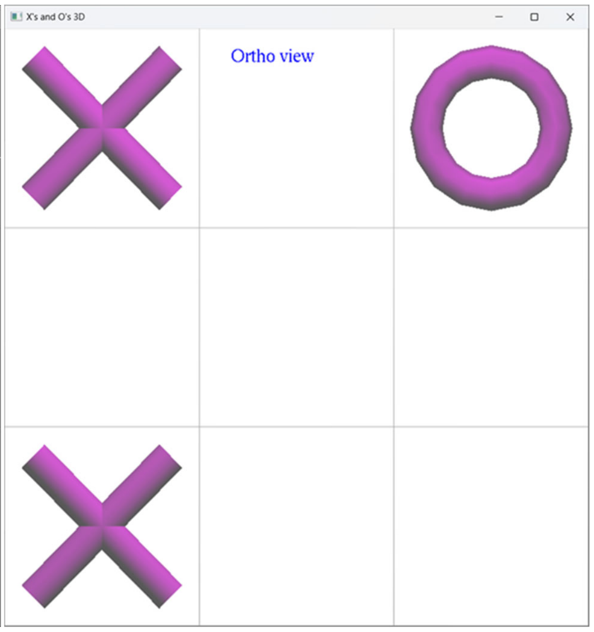
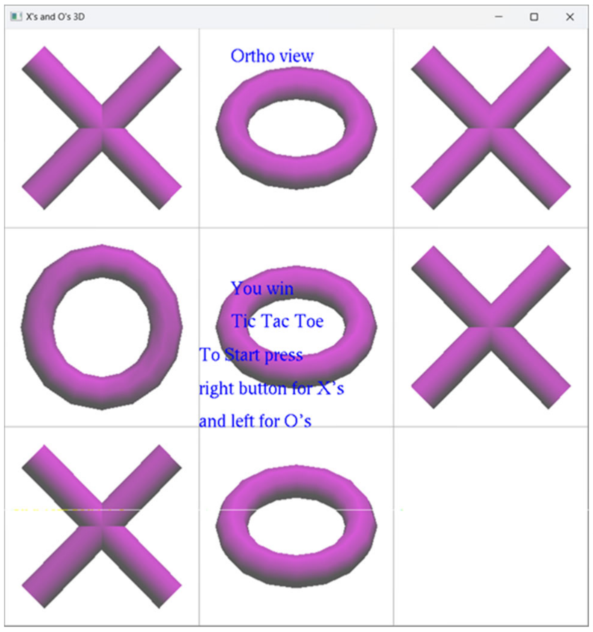


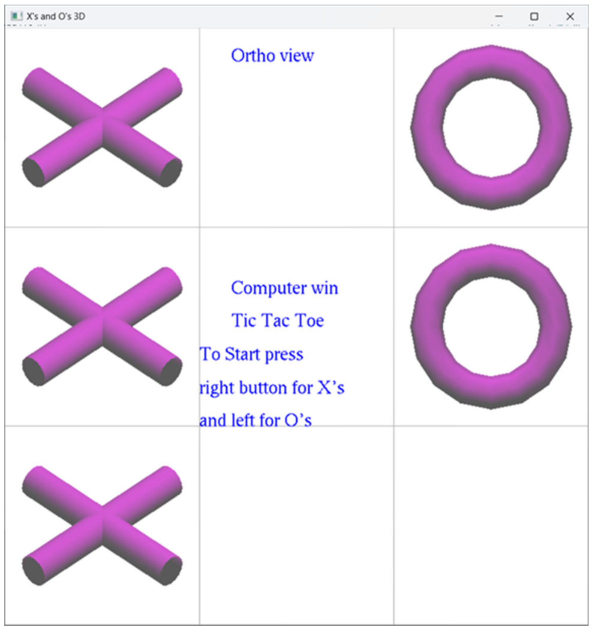
Fig 6.2: Os and Xs as Input



Ortho view

You win
Tic Tac Toe
To Start press
right button for X's
and left for O's

Fig 6.3: Player Wins Match



Ortho view

Computer win
Tic Tac Toe
To Start press
right button for X's
and left for O's

Fig 6.4: Computer Wins Match

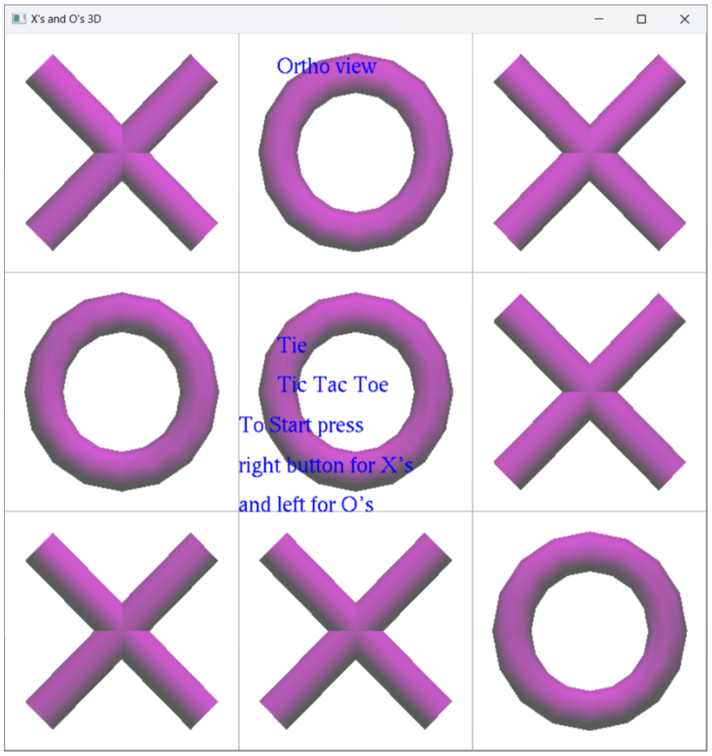


Fig 6.5: Draw Match

# CONCLUSION

Developing a Tic Tac Toe game using C++ and OpenGL within Visual Studio not only provides a platform for refining programming skills but also fosters a deeper understanding of the intricacies of graphic design. With C++ serving as the backbone for implementing the game's logic and OpenGL enhancing its visual allure, developers embark on a journey of creativity and innovation. Visual Studio's robust development environment offers an array of tools and features that simplify the coding process and streamline project management tasks. The seamless integration of OpenGL into Visual Studio further enhances the development experience, empowering developers to create captivating visuals and seamless animations that bring the game board to life. Throughout the development process, meticulous attention to detail and thorough testing ensures that every aspect of the game, from handling user inputs to rendering graphics, operates flawlessly. This meticulous approach not only ensures a polished final product but also fosters a sense of pride in the craftsmanship and technical prowess demonstrated by the development team. In essence, the creation of a Tic Tac Toe game using C++ and OpenGL in Visual Studio transcends mere game development; it represents a harmonious fusion of programming proficiency and artistic expression. It stands as a testament to the dedication, collaboration, and ingenuity of the developers involved, resulting in an engaging and immersive gaming experience that resonates with players of all ages.

# REFERENCES

[1] https://www.opengl.org/Documentation/Specs.html

[2] https://developer.nvidia.com/opengl

[3] https://en.wikipedia.org/wiki/OpenGL_ES

[4] https://www.javatpoint.com/opengl-cpp

[5] https://learnopengl.com/Introduction

[6] https://www.geeksforgeeks.org/getting-started-with-opengl/

[7] http://www.opengl-tutorial.org/