**CHAPTER 1**

# INTRODUCTION

## PREAMBLE

Computer graphics is a difficult topic, requiring associating mathematics and programming skills. When initially taught at undergraduate levels, there are several factors which discourage students. First, programming a first computer graphics program requires a substantial initial framework which can be intimidating for many of them. Second, understanding and applying mathematical concepts is very often overwhelming. To counter this intimidating feeling, a new teaching approach was proposed in 2018 to 3rd year undergraduate computer science students. The course was split into two parts, theory and practice. The theoretical concepts were seen in class, with course handouts and table exercises resembling closely to traditional computer graphics learning. The originality of the course comes from a new way of practicing 3D programming. Practical labs were built upon the Unity game engine programming platform, adapted to match the theoretical concepts seen in classroom. Conclusions are drawn over 4 years of teaching this course. When taught using an accompanying easy-to-access graphics programming platform, computer graphics becomes a more attractive course for students with lower mathematics and programming skills. It is also very satisfactory for skillfull students as it enables them to grab and master concepts quickly to reach interesting final lab achievements.

### OPENGL OPERATION

A commonly used standard for creating graphics applications is OpenGL. OpenGL is simple to grasp and has the most of the traits of other well liked graphics systems. It is a top-down strategy. A cross-language, cross platform API for creating programmes that generate 2D and 3D computer graphics is defined by the OpenGL standard definition. More than 250 distinct function calls are included in the interface, which may be utilised to create intricate three-dimensional sceneries out of basic primitives. Silicon Graphics Inc. (SGI) created OpenGL in 1992, and it is now extensively used in CAD, virtual reality, information visualisation, science visualisation, and aviation simulation. Additionally, it competes with Direct3D in video games running on Microsoft Windows systems.

Through the collection of functions included in the graphics library, the interface between the application software and the graphics system may be configured. The application programme interface is the name of the specification (API). The application programme is protected from the intricacies of the hardware and software implementation of the graphics library since it only sees the API.The majority of our apps will be made to use three libraries' functionalities to directly access openGL. Functions that are contained in the core GL library have names that start with the letter gl. The OpenGL Utility Library is the second (GLU). This library solely employs GL functions, but it also includes scripts for common object creation and viewing.

Instead of employing a new library for each system, we utilised the openGL Utility Toolkit library, which was readily accessible (GLUT). It is referred to as #include<glut.h>

A graphics editor is a piece of software that enables users to interactively create and modify images on a computer screen before saving them in one of the several well-liked "bitmap" or "raster" formats, including TIFF, JPEG, PNG, and GIF. The two main categories of graphic editors are 2D and 3D editors, respectively. Drawing 3D primitives, such as rectangles, circles, and polygons, as well as editing them using operations like cut, copy, and paste, are done in a 3D graphics editor. All facets of utilising a computer to create pictures or images are covered by computer graphics. OpenGL is a specific type of graphics software that has grown to be a widely used standard for creating graphics programmes. The following are some of the primary fields where computer graphics are used:

1) Information display

2) Design

3) Animation

4) Interfaces for users.

Our project named “Spaceship Shooter Game” uses OpenGL software interface and develops 2D images. This project uses the techniques like Translation, motion, display list, transformation techniques, etc. The graphics in OpenGL provides a wide variety of built-in function. The graphics remains one of the most exciting and rapidly growing computer fields. It has become a common element in user interface, data visualization, TV commercials, motion picture and many other applications. The current trend of computer graphics is to incorporate more physics principles into 3D graphics algorithm to better simulate the complex interactions between objects and lighting environment.

### 1.1.2 STUDY OF COMPUTER GRAPHICS

### 

The study of 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 three-dimensional computer graphics, it also encompasses two-dimensional graphics and image processing. As an academic discipline, computer graphics studies the 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.

### 1.1.3 APPLICATIONS

### 

Computer graphics deals with creation, manipulation and storage of different type of images and objects. Some of the applications of computer graphics are:

**1. Computer Art:**

Using computer graphics we can create fine and commercial art which include animation packages, paint packages. These packages provide facilities for designing object shapes and specifying object motion. Cartoon drawing, paintings, logo design can also be done.

**2.Computer Aided Drawing**:

Designing of buildings, automobile, aircraft is done with the help of computer aided drawing, this helps in providing minute details to the drawing and producing more accurate and sharp drawings with better specifications.

**3.Presentation Graphics:**

For the preparation of reports or summarising the financial, statistical, mathematical, scientific, economic data for research reports, managerial reports, moreover creation of bar, graphs, pie charts, time chart, can be done using the tools present in computer graphics.

**4.Entertainment**:

Computer graphics finds a major part of its utility in the movie industry and game industry. Used for creating motion pictures , music video, television shows, cartoon animation films. In the game industry where focus and interactivity are the key players, computer graphics helps in providing such features in the efficient way.

**5.Training**:

Specialised system for training like simulators can be used for training the candidates in a way that can be grasped in a short span of time with better understanding. Creation of training modules using computer graphics is simple and very useful.

**6.Visualisation:**

Today the need of visualise things have increased drastically, the need of visualisation can be seen in many advance technologies , data visualisation helps in finding insights of the data , to check and study the behaviour of processes around us we need appropriate visualisation which can be achieved through proper usage of computer graphics.

**7.Image Processing**:

Various kinds of photographs or images require editing in order to be used in different places. Processing of existing images into refined ones for better interpretation is one of the many applications of computer graphics.

* 1. **OVERVIEW OF OPENGL**

OpenGL (Open Graphics Library) is a cross-language, cross-platform application programming interface (API) for rendering 2D and 3D vector graphics. The API is typically used to interact with a graphics processing unit (GPU), to achieve hardwareaccelerated rendering. Silicon Graphics, Inc. (SGI) began developing OpenGL in 1991 and released it on June 30, 1992. Applications use it extensively in the fields of computer-aided design (CAD), virtual reality, scientific visualization, information visualization, flight simulation, and video games. Since 2006, OpenGL has been managed by the non-profit technology consortium Khronos Group.

OpenGL is an actively developed API. New versions of the OpenGL specifications are regularly released by the Khronos Group, each of which extends the API to support various new features. The details of each version are decided by consensus between the Group's members, including graphics card manufacturers, operating system designers, and general technology companies such as Mozilla and Google.

In addition to the features required by the core API, graphics processing unit (GPU) vendors may provide additional functionality in the form of extensions. Extensions may introduce new functions and new constants, and may relax or remove restrictions on existing OpenGL functions. Vendors can use extensions to expose custom APIs without needing support from other vendors or the Khronos Group as a whole, which greatly increases the flexibility of OpenGL. All extensions are collected in, and defined by, the OpenGL Registry. If multiple vendors agree to implement the same functionality using the same API, a shared extension may be released, using the identifier EXT. In such cases, it could also happen that the Khronos Group's Architecture Review Board gives the extension their explicit approval, in which case the identifier ARB is used.

* + 1. **PRIMITIVES AND COMMANDS**

OpenGL renders primitives, such as points, line segments, or polygons, in a variety of ways. Modes may be controlled independent of one another, therefore changing one mode does not effect the setting of any other modes (although many modes may interact to determine what eventually ends up in the frame buffer). By providing commands in the form of function calls, primitives are described, modes are established, and other OpenGL actions are documented. One or more vertices in a group are what characterise primitives. A vertex is a point, a line's terminal, or a polygonal corner where two edges converge. Each vertex and the data it contains are handled individually, sequentially, and uniformly. The data each vertex is associated with includes vertex coordinates, colours, normal, texture coordinates, and edge flags. The only situation in which this rule does not apply is when a set of vertices has to be clipped in order for a certain primitive to fit within a given area; in this situation, the vertex data may be altered and new vertices may be produced. Depending on whatever primitive the collection of vertices represents, the type of clipping will vary. There may be an arbitrary delay before a command takes effect, but commands are always handled in the order that they are received. This indicates that before any more commands are issued, each primitive is entirely rendered.

**1.2.2 BASIC BLOCK DIAGRAM**

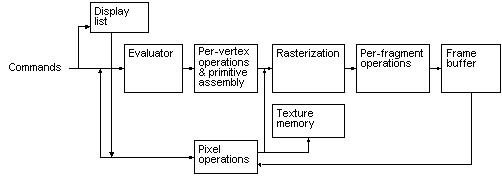


Fig 1.1 OpenGL Block Diagram

The processing stages in basic OpenGL operation are as follows:

* Display list:

Rather than having all commands proceed immediately through the pipeline, you can choose to accumulate some of them in a display list for processing later.

* Evaluator:

The evaluator stage of processing provides an efficient way to approximate curve and surface geometry by evaluating polynomial commands of input values.

* Per-vertex operations and primitive assembly:

OpenGL processes geometric primitives points, line segments, and polygons all of which are described by vertices. Vertices are transformed and lit, and primitives are clipped to the viewport in preparation for rasterization

* Rasterization:

The rasterization stage produces a series of frame-buffer addresses and associated values using a two-dimensional description of a point, line segment, or polygon. Each fragment so produced is fed into the last stage, perfragment operations.

* Per-fragment operations:

These are the final operations performed on the data before it is stored as pixels in the framebuffer. Per-fragment operations include conditional updates to the framebuffer based on incoming and previously stored z values (for z buffering) and blending of incoming pixel colors with stored colors, as well as masking and other logical operations on pixel values.

**CHAPTER 2**

## SYSTEM REQUIREMENT AND SPECIFICATION

### 2.1 GENERAL DESCRIPTON OF THE SYSTEM

The current method makes use of computer graphics. Soon after the invention of computers themselves, data was first displayed using hardcopy plotters and cathode ray tube displays. Creating, storing, and manipulating models and pictures of things are all included. These models range from the physical to the mathematical to the engineering to the architectural. The user controls the contents, structure, and look of objects and their projected pictures by utilising input devices such a keyboard, mouse, or touch-sensitive panel on the screen. Computer graphics nowadays are primarily interactive. Since the development of photography and television, interactive computer graphics have been the most significant method of making images.

### 2.2 SYSTEM REQUIREMENTS

#### 2.2.1 HARDWARE REQUIREMENTS

* Processor: Intel Core i3, i5 or i7
* RAM: 1GB Pentium II or Higher Processors

#### 2.2.2 SOFTWARE REQUIREMENTS

* Operating System: Ubuntu, Windows
* Programming Languages: C
* GLUT 32 bit file
* OpenGL

**CHAPTER 3**

## SYSTEM DESIGN AND MODELLING

### 3.1 PRELIMINARY DESIGN

Player Spaceship Control:

The game should allow the player to control their spaceship using input controls such as keyboard, mouse, or touchscreen. The spaceship should be able to move in multiple directions, such as up, down, left, and right, and also have the ability to shoot projectiles.

Enemy Spaceships:

The game should include various types of enemy spaceships that pose a threat to the player. These enemy ships may have different behaviors, such as moving in patterns, shooting projectiles, or attempting to collide with the player's spaceship.

Scoring System:

The game should implement a scoring system to track and display the player's progress. Points can be earned by shooting down enemy ships or achieving certain objectives. The game may also include a leaderboard to showcase the highest scores.

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

The OpenGL graphic software system in the proposed system is created as a simplified, hardware-independent interface to be implemented on several hardware platforms. No windowing or user input instructions are provided by OpenGL to help you accomplish these attributes; instead, you must work with the windowing system that governs the specific hardware you're using. High-level instructions for specifying models of three-dimensional objects are not provided by OpenGL. You may be able to specify pretty complex forms with such instructions, including chemicals, cars, body parts, and aeroplanes. With OpenGL, your intended model must be constructed from a limited number of geometric primitives, such as points, lines, and polygons.

### 3.3 PSEUDO CODE

**3.3.1 PSUEDO CODE TO CREATE SPACESHIP**

glColor3f(0.5, 0, 0.5); glScalef(70,20,1);

glutSolidSphere(1,50,50);

**3.3.2 PSEUDO CODE TO CREATE STEERING WHEEL**

glLineWidth(3);

glColor3f(0.20,0.,0.20);

glScalef(7,4,1);

glTranslated(-1.9,5.5,0);

glutWireSphere(1,8,8);

**3.3.3 PSEUDO CODE TO CREATE SPACESHIP DOOM**

glColor4f(0.7,1,1,0.0011); glTranslated(0,30,0);

glScalef(35,50,1); glutSolidSphere(1,50,50);

**3.3.4 PSEUDO CODE TO CREATE ALIEN**

glPushMatrix();

glRotated(-1,0,0,1);

glTranslated(-8,36,0);

glScalef(2.5,4,0);

glutSolidSphere(1,100,100);

glPopMatrix();

**CHAPTER 4**

# IMPLEMENTATION

## 4.1 MODULE DESCRIPTION

## 

The OpenGL Utility Tool Kit (GLUT) Programming Interface, which is the built-in graphics library, is used to develop the graphics package. The sketched objects are organised into built-in functions that may be employed in accordance with the specifications.

The two primary criteria for it are ease of understanding and speed of operation, both of which must be taken into consideration throughout the design and implementation phases. A header file called GL/glut.h, stdlib.h, and stdio.h are required before we can begin rendering any visuals on the screen. While graphic functions are in the OpenGL utility library, the header files include definitions and descriptions for the functions that are required. The OpenGL utility library(GLU) contains several groups of commands that compliment the core openGL by providing support for auxiliary features. Since these utility routine’s make use of core openGl commands, any openGL implementation’s is guaranteed to the support the utility routines.

In this project, first an attempt has been made to find the need of the system. To fulfil the needs, a detailed study had been conducted to find the various requirements of the system. This particular system has been designed in an attractive manner, so that even a user with minimum knowledge can be able to operate the system easily.

This Project includes:

**-GLUT**

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

• **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 constants, and declarations of functions and type used for various standard input and output operations.

• **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.

## 4.2 OPENGL FUNCTIONS USED

* **void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)**

The gluOrtho2D function sets up a two-dimensional orthographic viewing region.

* **void glutMouseFunc(mouse)**

It refers to the mouse callback function. The function to callback is defined as void mouse(int button, int state, int x, int y) { if (button == GLUT\_LEFT\_BUTTON && state == GLUT\_DOWN) s+=1; }. Here mouse interface is given to start the game by clicking on the start screen.

* **void glutSwapBuffers()**

It swaps the front and back buffers. User defined functions are used to color the curves in a standard cycle rainbow manner which becomes very easy for the user to identify the levels of recursion for the curves.

* **void glutInit(int \*argc, char\*\*argv)**

Initializes GLUT< the arguments from main are passed in and can by the application.

* **void glutCreateWindow(char \*title)**

Creates a window on the display. The string title can be used to label it. The return value provides a reference to the window that can be used when there are multiple windows.

* **void glutInitDisplaymode(unsigned int mode)**

Requests a display with the properties in mode. We use logical OR for various options.

* **void glutInitWindowSize(int width, int height)**

Specifies the initial height and width of the window in pixels.

* **void glutInitWindowPosition(int x, int y)**

Specifies the initial position of the top-left corner of the window in pixels.

* **void glViewport(int x, int y, GLsizei width, GLsizei height)**

Specifies a width \* height viewport in pixels whose lower-left corner is at (x,y) measured from the origin of the window.

* **void glutMainLoop()**

Cause the program to enter an event –processing loop.it should be the statement in main.

* **void glutPostRedisplay()**

Requests that the display callback be executed after the current callback returns.

* **glClearColor()**

glClearColor specifies the red, green, blue, and alpha values used by glClear to clear the color buffers. Values specified by glClearColor are clamped to the range [ 0, 1].

• **glClear()**

glClear sets the bit plane area of the window to values previously selected by

glClearColor, glClearIndex, glClearDepth, glClearStencil, and glClearAccum.

* **glColor3f()**

glColor3f () takes 3 arguments: the red, green and blue components of the color you want. After you use glColor3f, everything you draw will be in that color.

* **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.

**4.3 CUSTOM FUNCTIONS USED**

* **void myinit ( )**

It sets OpenGL state, program variables. We call glViewport( ), glPixelStorei( ), glMatrixMode( ), glLoadIdentity( ), gluOrtho2d( ), and glutPostRedisplay( ) functions. It unpacks the pixel matrix named rasters[ ] **.**

* **void start\_screen(int I, int j)**

It is responsible for the design of the start screen. It defines the project title as text. It prints texts like “SPACESHIP SHOOTER GAME” and “Click to start!”

* **void alien( )**

It designs the alien using glBegin(GL\_POLYGON), glVertex2f( ), glEnd( ) and glFlush( ) functions.

* **void spaceship( )**

It designs the shape of the ship using glBegin(GL\_POLYGON), glVertex3f( ) and glEnd( ) functions .

* **void score( )**

It handles the mathematical computation required for calculating the points scored by the player.

* **void mouse(int button, int state, int x, int y)**

Here, the mouse interface is given to start the game by clicking on the start screen.

**4.4 DISPLAY FUNCTION:**

It is responsible for everything that is to be displayed on the output screen. From displaying the text like- levels, and project title, to even increasing speed according to each level, it handles all basic display tasks.

It even calls the score( ) function from here. Uses the following functions:

**glClear (GL\_COLOR\_BUFFER\_BIT)**

Indicates the buffers currently enabled for color writing buffer.

**glFlush ()**

Force execution of GL commands in finite time.

**glutSwapBuffers ()**

Swap the buffers i.e, exchanges the content of the 2 buffers.

**glLoadIdentity()**

This function replaces the current matrix with the identity matrix. It sets the current matrix to the initial state, which is an identity matrix, and it effectively resets any previous transformations.

**glEnd()**

This function indicates the end of a primitive. It marks the end of a series of vertices that define a polygon or line.

**glPushMatrix()**

This function saves the current matrix onto the matrix stack. It allows you to save the current transformation state so that you can apply additional transformations without affecting the previous state.

**glPopMatrix()**

This function restores the matrix from the top of the matrix stack. It discards the current matrix and replaces it with the matrix that was previously saved using `glPushMatrix()`.

**glMatrixMode(GL\_PROJECTION)**

This function sets the current matrix mode to the projection matrix mode. It indicates that subsequent matrix operations will affect the projection matrix.

**glMatrixMode(GL\_MODELVIEW)**

This function sets the current matrix mode to the modelview matrix mode. It indicates that subsequent matrix operations will affect the model-view matrix.

**4.5 MAIN FUNCTION**

Here we specify the initial display mode, window size and position. Create a new window where the output is rendered. Create menus to start the game, get the score as well as quit the game.

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

{

glutInit(&argc, argv); glutInitDisplayMode(GLUT\_SINGLE|GLUT\_RGB);

glutInitWindowPosition(0, 0);

glutInitWindowSize(1200, 600);

glutCreateWindow("Space Shooter");

init();

//glutReshapeFunc(reshape);

glutIdleFunc(refresh);

glutKeyboardFunc(keyPressed);

glutKeyboardUpFunc(keyReleased);

glutMouseFunc(mouseClick); glutPassiveMotionFunc(passiveMotionFunc); glGetIntegerv(GL\_VIEWPORT ,m\_viewport);

glutDisplayFunc(display); glutMainLoop();

}

**CHAPTER 5**

## TESTING

This chapter provides a summary of each testing technique used to produce a bug-free application. The product may be tested for quality using a variety of methods at various project development stages.

### 5.1 TESTING PROCESS

Testing is the process of evaluating a system or its component(s) with the intent to find whether it satisfies the specified requirements or not. In simple words, testing is executing a system in order to identify any gaps, errors, or missing requirements in contrary to the actual requirements. According to ANSI/IEEE 1059 standard.

Testing can be defined as - A process of analyzing a software item to detect the differences between existing and required conditions (that is defects/errors/bugs) and to evaluate the features of the software item.

### 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 concentrates verification efforts on the module's smallest unit of software design. The developed software is made up of different components. Each module has a distinct purpose. The exact flow of control for each module was validated throughout this type of testing. Important control pathways are evaluated to detect faults within the module's border using thorough design Considerations as a guide. Multiple modules can be tested in simultaneously using the Unit Test.

|  |  |  |
| --- | --- | --- |
| Module | Action | Expected Output |
| Display module | Display screen is generated based on the input. | The screen is displayed. |
| Mouse Function | Move the mouse cursor to change the position of basket. | The basket is moving. |

5.1: Test Cases for Spaceship Shooter Game

#### 5.3.2 INTEGRATION TESTING

Integration testing is the second level of testing. This involves combining multiple classtested modules into subsystems, which are then tested. The purpose here is to test if all of the modules can be correctly merged. We

merged all of the classes and tested them for compatibility. The errors discovered are recognised 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.

|  |  |  |
| --- | --- | --- |
| Menu Item | Action | Expected Output |
| Item 1 : Start Game | Right click on screen and select first option | The game starts with the aliens opposite to each other. |
| Item 2 : Score | Right click on screen and select first option | Scores or the life of alien gets decreased when the enemy attacks. |
| Item 3: Quit | Right click on screen and select first option | Quits the game and displays score screen |

Table 5.2: Test cases of various menu driven options

**CHAPTER 6**

## CONCLUSION AND FUTURE ENHANCEMENT

### 6.1 CONCLUSION

An attempt is made to successfully construct an OpenGL package that meets the user's needs. It is user-friendly, allowing the user to engage effectively and effortlessly. The construction of the tiny project, which was developed utilising point as a source, provided strong exposure to OpenGL, and some of the approaches that aid in the production of animations and gameplay were well known. It aids in understanding the fundamental implementation of functions and the fundamentals of OpenGL.

### 6.2 FUTURE ENHANCEMENTS

1. **Interactive User Controls**:

Implement interactive user controls to allow users to interact with the animation. For example, you can add keyboard or mouse inputs to control the rotation speed of the windmill, adjust the position of the sun, or even enable users to move and explore the scene.

1. **Enhanced Visual Effects**:

Add more visually appealing effects to enhance the overall look of the animation. You can incorporate lighting effects, shadows, textures, or even particle systems to simulate elements like moving clouds or flowing water. These additions will make the animation more immersive and visually stunning.

1. **Multiple Scenes and Transitions**:

Create multiple scenes with different environments or scenarios. Implement smooth transitions between scenes to provide a seamless and dynamic experience. For example, you can have day and night cycles, changing weather conditions, or transition between different landscapes. This will add variety and depth to the animation.

# 

**REFERENCES**

1. [http://www.google.com](http://www.google.com/)
2. [http://www.youtube.com](http://www.youtube.com/)
3. <https://www.opengl.org/>
4. <http://www.codeblocks.org/>
5. Textbook: Donald Hearn & Pauline Baker:

Computer Graphics with OpenGL Version, 4th Edition, Pearson Education,2011

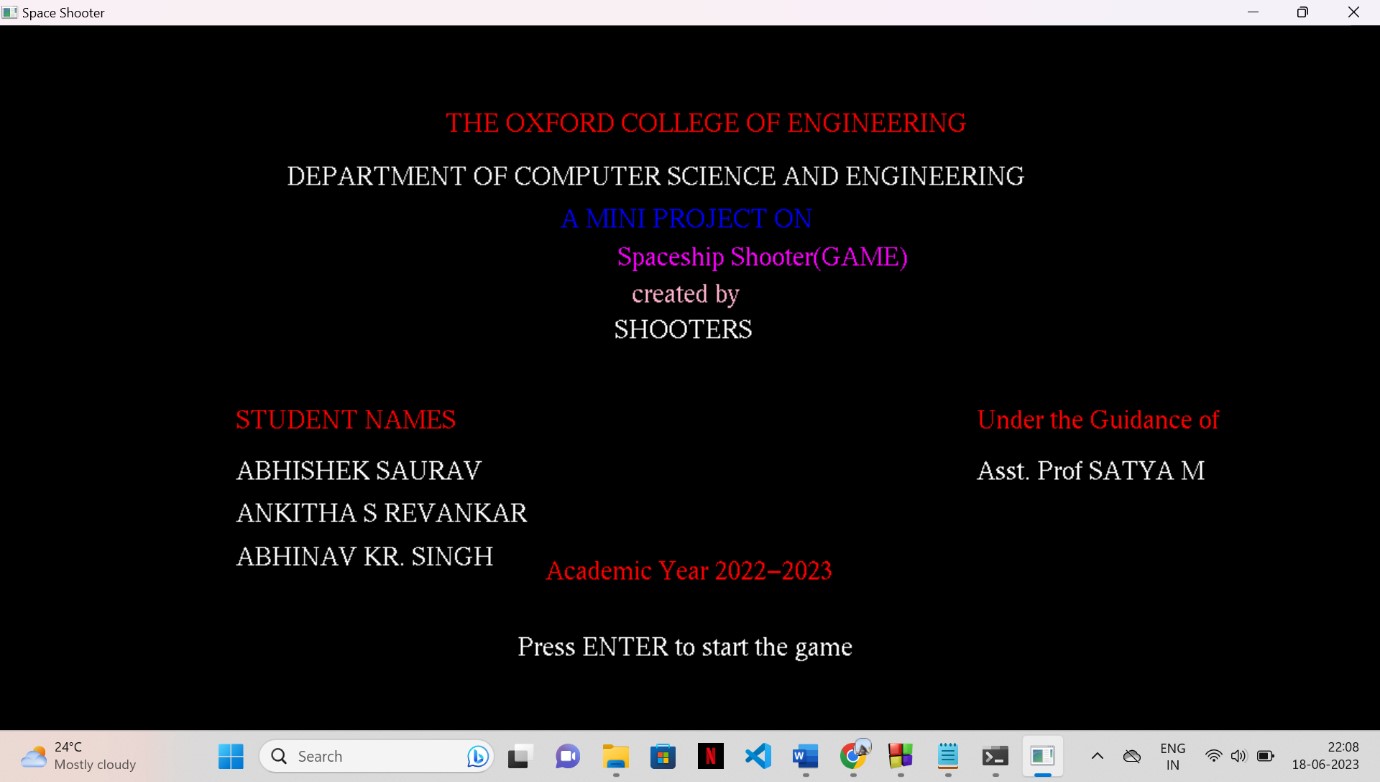
1. Textbook: Edward Angel: Interactive Computer Graphics-

A Top Down approach with OpenGL, 5th edition. Pearson

Education, 2008

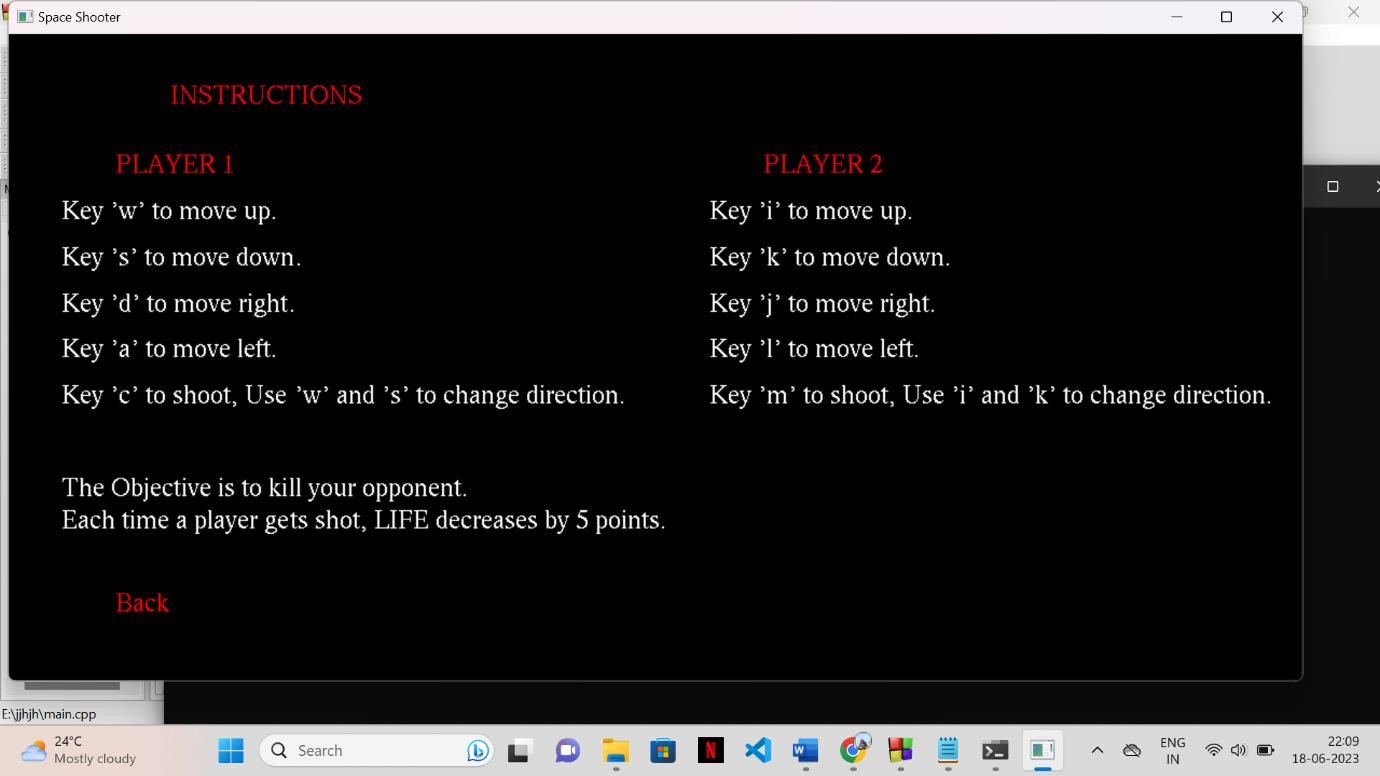
**APPENDIX**

## SNAPSHOTS OF THE PROJECT

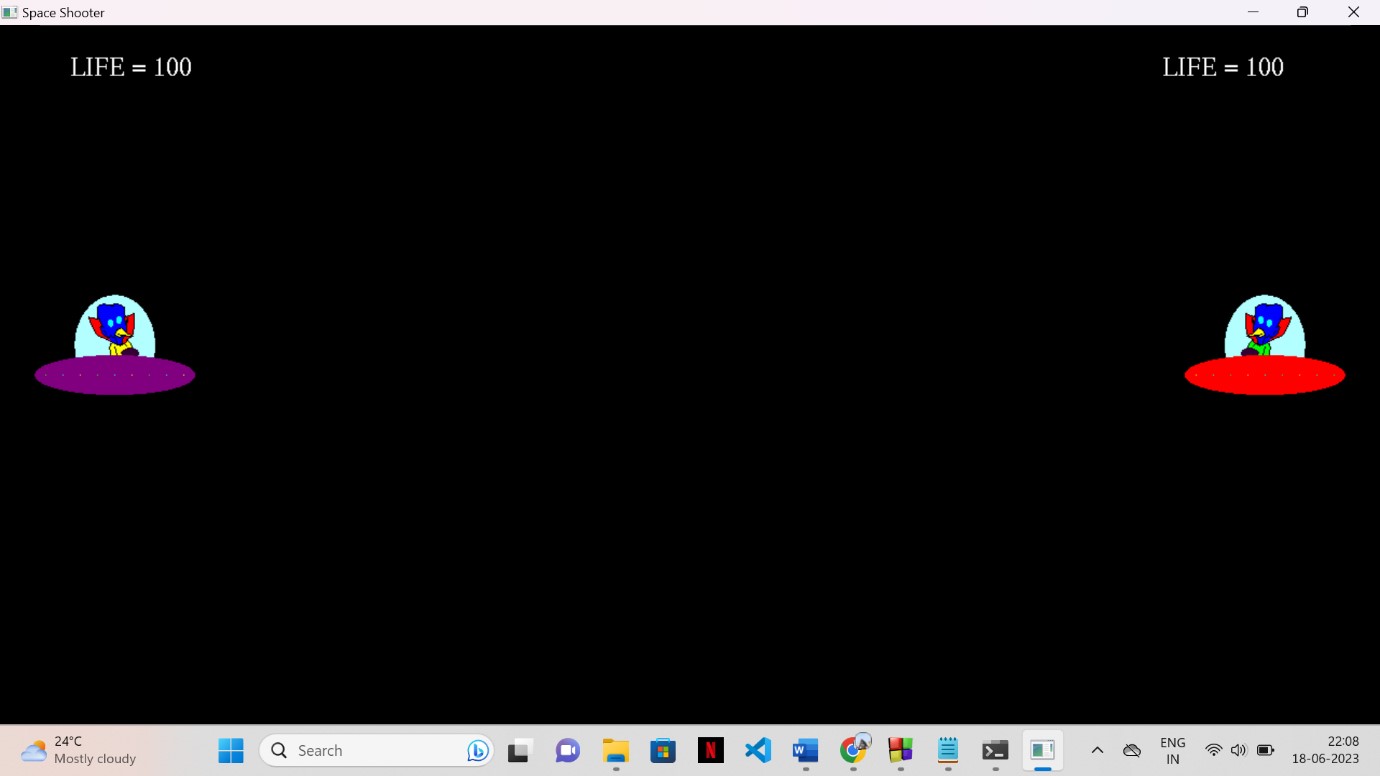
**Figure1: Display Screen**



**Figure 2: Start Game Screen**



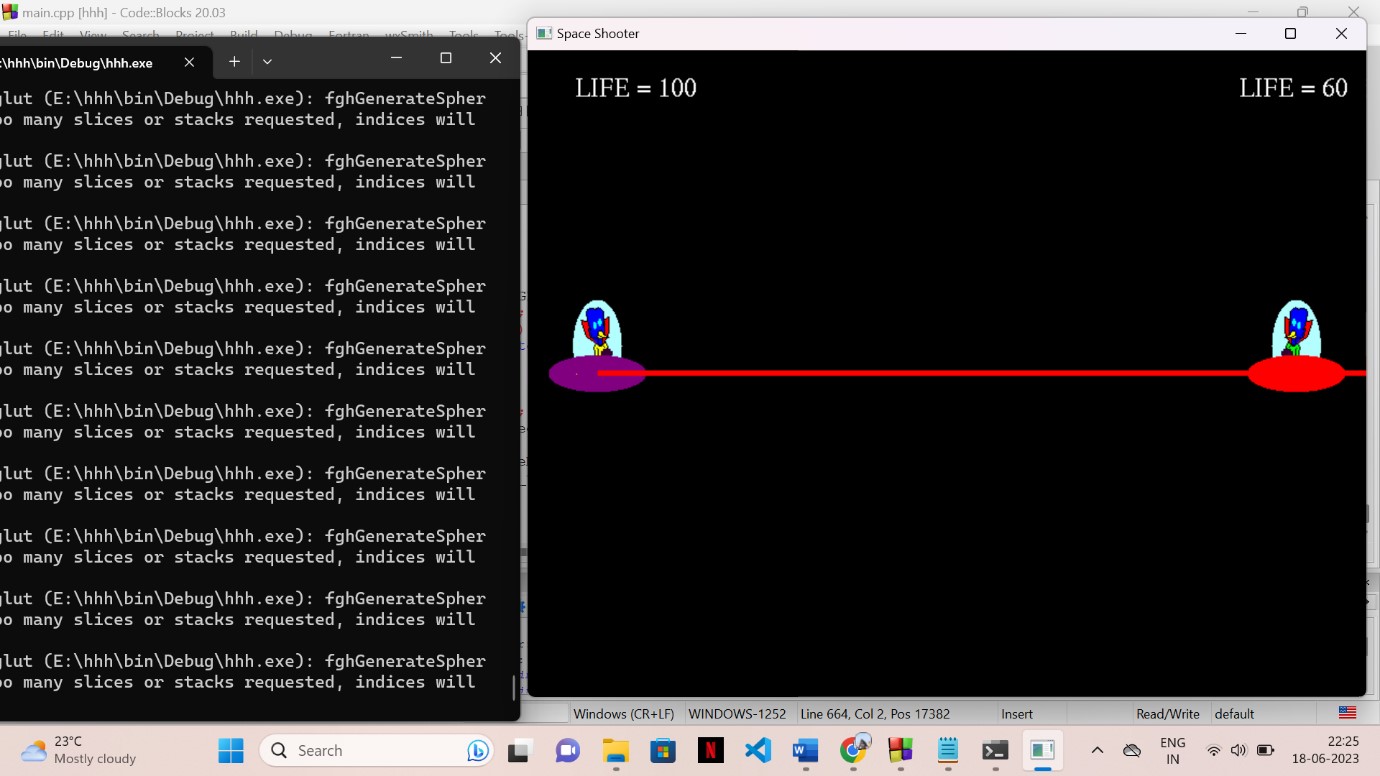
**Figure 3: Rules and Instructions Screen**



**Figure 4: Game Play Screen**



**Figure 5: Game Over Screen**



**Figure 6:Game Summary**