Visvesvaraya Technological University Belagavi,

Karnataka- 590018

A Mini Project Report

on

"BOAT SIMULATION"

Submitted in partial fulfilment of the requirements for the

Computer Graphics Laboratory with mini project (18CSL67)

Submitted by :

SRINIDHI N JOSHI
1JS20CS163

YASHASVI G M
1JS20CS187

Under the guidance of :

Dr. Sharana Basavana Gowda
Associate Professor, Dept of CSE
JSSATE, Bengaluru

Mrs. Rashmi B N
Assistant Professor, Dept of CSE
JSSATE, Bengaluru

JSS Academy of Technical Education, Bengaluru – 560060,

Department of Computer Science and Engineering

2022-2023

JSS Mahavidyapeetha, Mysuru

JSS Academy of Technical Education

JSS Campus, Uttarahalli – Kengeri Main Road, Bengaluru – 560060

Department of Computer Science and Engineering

CERTIFICATE

This is to certify that the project work entitled "**BOAT SIMULATION**" is a bona fide work carried out by **Mr. SRINIDHI N JOSHI (1JS20CS163)** and **Ms. YASHASVI GM (1JS20CS187)** in partial fulfillment of the requirements for the course Computer Graphics Laboratory of 6th semester, Bachelor of engineering in Computer Science and engineering of the Visvesvaraya Technological University, Belagavi, during the academic year 2022 – 2023. It is certified that all corrections and suggestions indicated for internal assessment have been incorporated in the report. The project report has been approved as it satisfies the academic requirements in respect of the project work prescribed for the said degree.

Dr Sharana Basavana Gowda	**Mrs Rashmi B N**	**Dr. Mallikarjuna P.B.**
Associate Professor ,	Assistant Professor,	Asso Professor and Head,
Dept. of CSE,	Dept of CSE	Dept. of CSE
JSSATE, Bengaluru	JSSATE, Bengaluru	JSSATE, Bengaluru

Name of the examiners	Signature with date
i) _____	i) _____
ii) _____	ii) _____

ABSTRACT

The boat simulation project aims to create a visually immersive experience of a boat sailing in a dynamic environment using the OpenGL graphics library. The simulation incorporates various features such as day and night modes, realistic sun and moon movements, and user-controlled boat navigation.

The simulation environment consists of meticulously designed components, including a dynamically rendered sky, sun, moon, clouds, Windmill and an animated sea. The boat itself is represented by a series of interconnected polygons, providing a visually appealing and realistic representation of a sailing vessel.

The project begins with a display function that presents project details, including the project title, author names, and the guidance of a professor. After receiving user input to proceed, a menu system is implemented, allowing the user to choose between different simulation modes and control the boat's speed.

To ensure smooth interaction, the program incorporates various keyboard input functions, enabling the user to adjust the boat's speed and direction. The boat's position is updated accordingly, providing a seamless and responsive simulation experience.

ACKNOWLEDGEMENT

I express my humble greetings to his holiness **Jagadguru Sri Shivarathri Deshikendra Mahaswamijigalavaru** who has showered their blessings on us for framing our career successfully.

The completion of any project involves the efforts of many people. I've been lucky to have received a lot of help and support quarters, during the making of this project. I take this opportunity to acknowledge all those, whose guidance and encouragement helped me emerge successful.

I am thankful to the resourceful guidance, timely assistance and graceful gesture of my guide **Dr. Sharana Basavana Gowda**, Associate professor, Department of Computer Science and engineering and **Mrs Rashmi B N,** Assistant professor, Department of Computer Science and Engineering, who helped me in every aspect of my project work.

I am also indebted to **Dr. Mallikarjuna P.B.**, Associate Professor and head of department of Computer Science and engineering for the facilities and support extended towards us.

I express our sincere thanks to our beloved principal, **Dr. Bhimasen Soragaon** for having supported us in our academic endeavors.

Last but not the least, I am pleased to express our heart full thanks to all the teaching and non-teaching staff of department of Computer and Science engineering and my friends whohave rendered their help, motivation and support.

SRINIDHI N JOSHI (1JS20CS163)
YASHASVI GM (1JS20CS187)

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

This report contains implementation of a visualizer for Singly Linked List and Doubly Linked List with several traversal functions implemented through a set of OpenGL functions. Linked list is one of the key concept in data structures.

We are mainly using separate menu to perform the necessary operations in the singly linked list and doubly linked list in the project by providing visual representation of the node insertion and deletion. This report includes details of the system requirements, overview, implementation, analysis, design and testing of the above mentioned project.

1.1.Computer Graphics

Computer graphics is concerned with all aspects of producing pictures or images using computers and, more generally, the representation and manipulation of image data by a computer. The combination of computers, networks, and the complex human visual system, through computer graphics, has led to new ways of displaying information, seeing virtual worlds, and communicating with people and machines.

Feature-length movies made entirely by computer have been successful, both critically and financially. Massive multiplayer games can involve tens of thousands of concurrent participants. The development of computer graphics has made computers easier to interact with, and better for understanding and interpreting many types of data. Developments in computer graphics have had a profound impact on many types of media and have revolutionized animation, movies and the video game industry.

OpenGL

OpenGL (for “open graphics library”) is a particular graphics software system which has become widely accepted standard for developing graphics application. It is a software interface to graphics hardware. The interface consists of a set of several hundred procedures

and functions that allow a programmer to specify the objects and operations involved in producing high quality graphical images, specifically color images of three- dimensional objects.

Most of OpenGL requires that the graphics hardware contain a frame buffer. Many OpenGL calls pertain to drawing objects such as points, lines, polygons, and bitmaps, but the way that some of this drawing occurs (such as when antialiasing or texturing is enabled) relies on the existence of a frame buffer. Further, some of OpenGL is specifically concerned with frame buffer manipulation. Our interaction with computers has become dominated by a visual paradigm that includes windows, icons, menus, and a pointing device, such as a mouse. From a user perspective, windowing OS X differ only in details. More recently, millions of people have become internet user. Their access is through graphical network browsers, such as Firefox and Internet Explorer, which uses these interface tools.

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 graphics and image processing. It is divided into two broad classes. It is also called passive graphics. Here the user has no control over the pictures produced on the screen. Interactive graphics provides extensive user-computer interaction. It provides a tool called “motion dynamics” using which the user can move objects.

A broad classification of major subfields in computer graphics might be:

1. Geometry: study of different ways to represent and process surfaces.
2. Animation: study of different ways to represent and manipulate motion.
3. Rendering: study of algorithms to reproduce light transport.
4. Imaging: study of image acquisition or image editing.
5. Topology: study of the behaviour of spaces and surfaces.

Geometry The subfield of geometry studies the representation of three-dimensional objects in a discrete digital setting. Because the appearance of an object depends largely on its exterior, boundary representations are most commonly used. Two dimensional surfaces are a good representation for most objects, though they may be non-manifold. Since surfaces are not finite, discrete digital approximations are used.

Animation The subfield of animation studies descriptions for surfaces (and other phenomena) that move or deform over time. Historically, most work in this field has focused on parametric and data driven models, but recently physical simulation has become more popular as computers have become more powerful computationally.

Rendering Rendering generates images from a model. Rendering may simulate light transport to create realistic images or it may create images that have a particular artistic style in non-photorealistic rendering. The two basic operations in realistic rendering are transport (how much light passes from one place to another) and scattering (how surfaces interact with light).

Imaging Digital imaging or digital image acquisition is the creation of digital images, such as of a physical scene or of the interior structure of an object.

Topology is the mathematical study of shapes and topological spaces.

1.2.OpenGL Interface

OpenGL is a software interface to graphics hardware. The interface consists of a set of a several hundred procedures and functions that allow a programmer to specify the objects and operation involved in producing high-quality graphical images, specifically color images of three-dimensional objects.

It is now a fact that graphical interfaces provide an alternative and easy interaction between users and computers which uses the visual control items such as buttons, menus, icons, scroll bars etc. which allows user to interact with the computer only by mouse click. This interaction is made possible by the use of libraries in OpenGL. The first library is the **GL** library. The name of the second library is **OpenGL Utility Library (GLU)** which contains GL functions but also has the code to create common objects & simplifying viewing. To interface with the windows system & to get input from the external devices into our programs we need one more library, rather than using a new library for each system, we use a readily available library called as **OpenGL Utility Toolkit (GLUT),** which provide the minimum functionality that should be expected in any modern windowing system.

Below is a diagram of Library Organization for an X Windows environment:

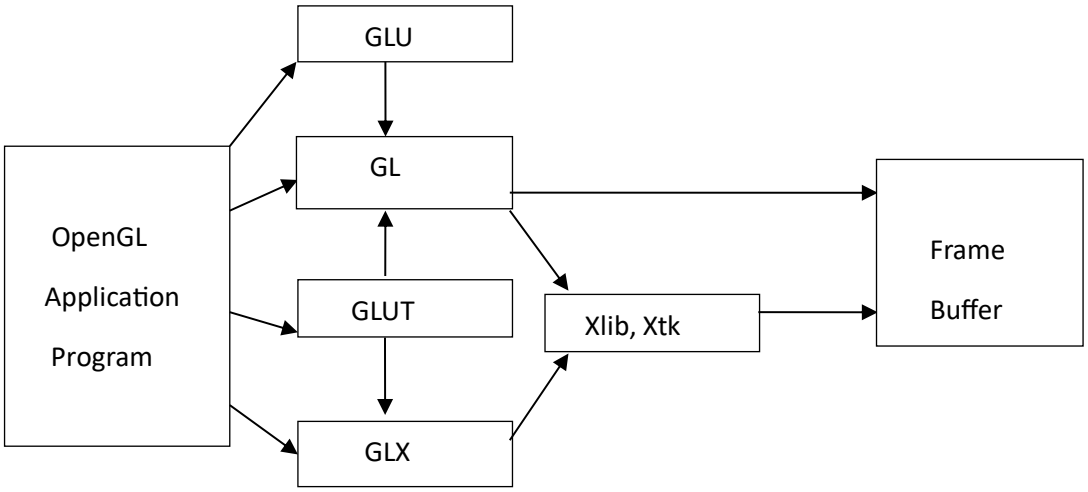

Fig.1 Library Organization

1.3. OpenGL Overview

OpenGL is a particular graphics software system which has become widely accepted standard for developing graphics application. 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.

High Visual Quality and Performance

Any visual computing application requiring maximum performance-from 3D animation to CAD to visual simulation-can exploit high-quality, high-performance OpenGL capabilities. These capabilities allow developers in diverse markets such as broadcasting, CAD/CAM/CAE, entertainment, medical imaging, and virtual reality to produce and display incredibly compelling 2D and 3D graphics.

Developer-Driven Advantages

Industrial standard

With broad industry support, OpenGL is the only truly open, vendor-neutral, multiplatform graphics standard.

Stable

OpenGL implementations have been available for more than seven years on a wide variety of platforms. Additions to the specification are well controlled, and proposed updates are announced in time for developers to adopt changes. Backward compatibility requirements ensure that existing applications do not become obsolete.

Reliable and Portable

All OpenGL applications produce consistent visual display results on any OpenGL API-compliant hardware, regardless of operating system or windowing system.

Evolving

Because of its thorough and forward-looking design, OpenGL allows new hardware innovations to be accessible through the API via the OpenGL extension mechanism. In this way, innovations appear in the API in a timely fashion, letting application developers and hardware vendors incorporate new features into their normal product release cycles.

Scalable

OpenGL API-based applications can run on systems ranging from consumer electronics to PCs, workstations, and supercomputers. As a result, applications can scale to any class of machine that the developer chooses to target.

Easy to use

OpenGL is well structured with an intuitive design and logical commands. Efficient OpenGL routines typically result in applications with fewer lines of code than those that make up programs generated using other graphics libraries or packages. In addition, OpenGL drivers encapsulate information about the underlying hardware, freeing the application developer from having to design for specific hardware features.

Well-documented

Numerous books have been published about OpenGL, and a great deal of sample code is readily available, making information about OpenGL inexpensive and easy to obtain.

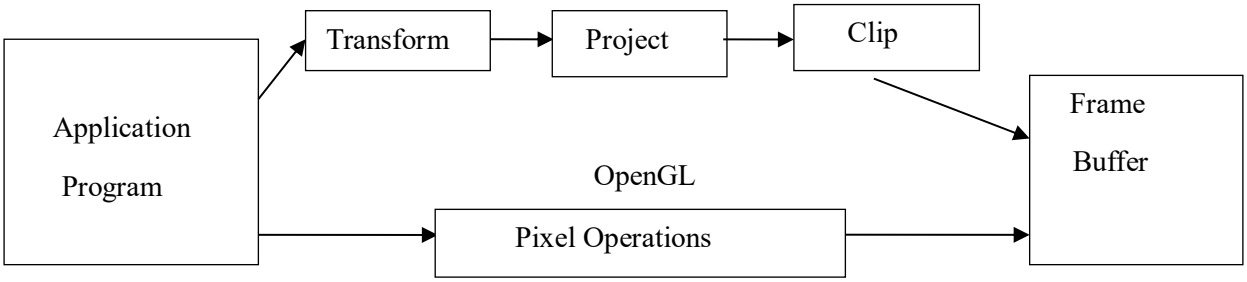

Fig.2 Simplified OpenGL pipeline

OpenGL Commands and Primitives

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 a line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors, normals, 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 fits 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 effect. It also meansthat state querying commands return data that's consistent with complete execution of all previously issued OpenGL commands.

OpenGL commands use the prefix gl and initial capital letters for each word making up the command name (glClearColor (), for example). Similarly, OpenGL defined constants begin

with GL_, use all capital letters, and use underscores to separate words (like GL_COLOR_BUFFER_BIT). Some seemingly extraneous letters appended to some command names (for example, the 3f in glColor3f () and glVertex3f ()) can also be seen. It's true that the Color part of the command name glColor3f() is enough to define the command as one that sets the current color.

However, more than one such command has been defined so that the user can use different types of arguments. In particular, the 3 parts of the suffix indicates that three arguments are given; another version of the Color command takes four arguments. The f part of the suffix indicates that the arguments are floating-point numbers. Having different formats allows OpenGL to accept the user's data in his or her own data format.

CHAPTER 2

ABOUT THE PROJECT

2.1 OVERVIEW

The boat simulation project aims to create an immersive and interactive experience of sailing a boat in a dynamic environment using the OpenGL graphics library. The project focuses on developing a visually appealing simulation that incorporates realistic rendering of various elements such as the sky, sun, moon, clouds, sea, and a detailed boat model.

The simulation provides users with the ability to navigate the boat using keyboard inputs, adjusting its speed and direction. The boat's position and orientation are updated in real-time, offering a responsive and engaging experience. Additionally, the project includes the implementation of day and night modes, with corresponding changes in the lighting and atmospheric conditions.

2.2 USER INTERFACE

The project includes a user interface with several components to enhance the user experience. Here are the details of the user interface:

Introduction Page:

The project starts by displaying an introduction page.

On the introduction page, there is a "right arrow" button.

Clicking the "right arrow" button allows the user to start the boat simulation.

Menu Page:

After the introduction page, the user is taken to the menu page.

The menu page contains instructions for the simulation.

On the menu page, there is a "<-" (left arrow) button.

Clicking the "<-" button takes the user back to the introduction page.

To proceed to the main simulation page, the user can press the "Enter" key.

Main Simulation Page:

The main simulation page provides a menu with various options.

The menu allows the user to switch between day and night scenes.

To switch to the day mode, the user can press the "d" or "D" key.

To switch to the night mode, the user can press the "n" or "N" key.

By default, the day mode is selected.

The menu also allows the user to stop all movement by pressing the "s" or "S" key.

The default speed is the normal speed the key used is r/R.

To increase the speed of the boat, the user can press the "+" key.

To decrease the speed of the boat, the user can press the "-" key.

The menu provides an option to exit the simulation by pressing the "e" or "E" key.

CHAPTER 3

SYSTEM SPECIFICATION

3.1 Software Requirements

- Operating System : Windows 11
- Language Tool : C/C++ USING OpenGL
- Compiler : C Compiler (GNU GCC Compiler)
- Libraries : 32 bit color resolution
- Documentation : MS-Word.

3.2 Hardware Requirements

- Processor: Intel core i5 onwards
- RAM : 128Mb RAM or higher
- Hard Disk : 3.1 GB or higher
- Monitor : EGVGA Compatible
- Keyboard : Standard 101 key Keyboard
- Mouse : Compatible pointing device

CHAPTER 4

IMPLEMENTATION

Header files

- **iostream:**

This header file provides basic input and output services in C++.

It includes the definitions of standard stream objects such as cin, cout, and cerr.

It is used for input/output operations in the code.

- **stdio.h:**

This is a standard header file in C that provides input and output operations.

It includes functions like printf, scanf, fprintf, fscanf, etc.

It is used for input/output operations in the code.

- **stdlib.h:**

This is a standard header file in C that provides general-purpose functions.

It includes functions such as memory allocation, process control, conversions, etc.

It is used for general-purpose functions in the code.

- **string.h:**

This is a standard header file in C that provides various string manipulation functions.

It includes functions like strcpy, strcat, strlen, strcmp, etc.

It is used for string manipulation operations in the code.

- **GL/glut.h:**

This header file is specific to the OpenGL Utility Toolkit (GLUT) library.

It includes functions and definitions for creating and managing windows, handling input events, and rendering graphics using OpenGL.

It is used for creating a graphical user interface and rendering graphics in the code.

- **math.h:**

This is a standard header file in C that provides mathematical functions and constants.

It includes functions like sin, cos, sqrt, pow, etc.

It is used for mathematical calculations in the code.

- **windows.h:**

This header file is specific to the Windows operating system.

It includes functions and definitions for working with the Windows API, handling windows, processes, threads, etc.

It is used for Windows-specific operations in the code.

4.1 Functions used in OpenGL

- **glClearColor(float red, float green, float blue, float alpha):** This function sets the clear color for the color buffer. It takes four parameters: red, green, blue, and alpha, representing the color components in the range of [0, 1].
- **glClear(GLbitfield mask):** This function clears buffers to preset values. The mask parameter specifies which buffers to clear. In the code, GL_COLOR_BUFFER_BIT is used to clear the color buffer.
- **glMatrixMode(GLenum mode):** This function sets the current matrix mode. In the code, GL_PROJECTION mode is used to set the matrix mode to operate on the projection matrix stack.
- **glLoadIdentity():** This function replaces the current matrix with the identity matrix. It is used to reset the transformation matrix to the identity matrix before applying further transformations.
- **gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top):** This function defines a 2D orthographic projection matrix. It specifies the clipping planes for left, right, bottom, and top coordinates of the viewing frustum.
- **glPushMatrix():** This function pushes the current matrix stack down by one, duplicating the current matrix. It is used to save the current transformation matrix before applying further transformations.

- **glPopMatrix():** This function pops the current matrix stack, discarding the current matrix. It is used to restore the previously saved transformation matrix.
- **glTranslatef(GLfloat x, GLfloat y, GLfloat z):** This function translates the current matrix by the specified translation vector (x, y, z). It is used to apply translation transformations.
- **glRotatef(GLfloat angle, GLfloat x, GLfloat y, GLfloat z):** This function rotates the current matrix by the specified angle around the specified axis (x, y, z). It is used to apply rotation transformations.
- **glColor3f(float red, float green, float blue):** This function sets the current color for drawing operations. It takes three parameters: red, green, and blue, representing the color components in the range of [0, 1].
- **glBegin(GLenum mode):** This function marks the beginning of a primitive specification. The mode parameter specifies the type of primitive to be drawn, such as GL_POLYGON for polygons.
- **glVertex2f(GLfloat x, GLfloat y):** This function specifies a vertex of a primitive. It takes the x and y coordinates of the vertex.
- **glEnd():** This function marks the end of a primitive specification.
- **glutCreateMenu(void (*func)(int)):** This function creates a new pop-up menu. It takes a callback function as a parameter, which will be called when a menu item is selected.
- **glutAddMenuEntry(const char *label, int value):** This function adds an entry to the current pop-up menu. It takes a label string and an integer value associated with the menu item.
- **glutAttachMenu(int button):** This function attaches the current pop-up menu to the specified button. In the code, GLUT_RIGHT_BUTTON is used to attach the menu to the right mouse button.

4.2 Main Functions

- **glutInit(int* argc, char** argv) :** Initializes the GLUT library and processes any command-line arguments. It takes the address of the argc variable and the argv array from the main function.
- **glutInitDisplayMode(unsigned int mode):** Sets the initial display mode for the GLUT window. The mode parameter specifies the display mode options, such as RGB color, single buffering, and depth buffer.
- **glutInitWindowPosition(int x, int y) :** Sets the initial position of the GLUT window on the screen. The x and y parameters specify the screen coordinates of the window's top-left corner.
- **glutInitWindowSize(int width, int height) :** Sets the initial size (width and height) of the GLUT window.
- **glutCreateWindow(const char* title)** : Creates a top-level window with the specified title.The function returns a unique identifier for the window.
- **glutDisplayFunc(display) :** Sets the display callback function for the current window. The func parameter is a function pointer to the display function that will be called whenever the window needs to be redrawn.
- **glutSpecialFunc(SpecialInput):** Sets the special keyboard callback function for the current window. The func parameter is a function pointer to the special keyboard function that will be called whenever a special key (such as the arrow keys) is pressed.
- **glutTimerFunc(20,update1,0) :** Registers a timer callback function to be called after a specified number of milliseconds.The millis parameter specifies the timer interval. The func parameter is a function pointer to the timer function that will be called. The value parameter is a user-

defined value that will be passed to the timer function.

- **glutKeyboardFunc(handleKeypress) :** Sets the keyboard callback function for the current window. The func parameter is a function pointer to the keyboard function that will be called whenever a regular key is pressed.
- **glutMouseFunc(handleMouse) :** Sets the mouse callback function for the current window. The func parameter is a function pointer to the mouse function that will be called whenever a mouse button is pressed or released.
- **glutPostRedisplay() :** Posts a redisplay request to the window system.This function notifies GLUT that the current window needs to be redrawn.
- **glutMainLoop() :** Enters the GLUT event processing loop. This function continuously checks for events (such as input events or window updates) and calls the registered callback functions to handle those events.

4.3 Source Code

Main.cpp:

```
#include <iostream>
#include<stdio.h>
#include<stdlib.h>
#include<string.h>
#include<GL/glut.h>
#include<math.h>
#include<windows.h>
#define PI 3.1416
using namespace std;
GLint i=0;
GLint mode=0,sun_mode=0,boat_mode=0,smoke_mode=0;
GLfloat sun_spin=0, sun_x=0, sun_y=0, moon_spin=0;
GLfloat ax=0,bx=0,cx=0,dx=0,str=500.0,mn=500.0,moveB=0.0,rx=60.0;
GLfloat sr=0.0,sg=0.749,sb=1.0;
float _moveA = 0.0f;
float _moveB = 0.0f;
float _moveC = 0.0f;
float _angle1 = 0.0f;
float speed = 0.02f;
void create_menu(void);
void menu(int);
GLfloat spin = 0.0;
int flag = 0;
void display(){
    glClearColor(0.3,0.1,0.3,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,1.0,1.0);
    glRasterPos3f(-0.35,0.8,-1.0);
    text("JSS ACADEMY OF TECHNICAL EDUCATION");
    glRasterPos3f(-0.25,-0.8,-1.0);
    text("Assoc. Prof., Dept of CSE");
    // Display page information is mentioned here
```

```
    glColor3f(1.0,1.0,1.0);
    glRasterPos3f(0.70,-0.9,-1.0);
    text("press -> to continue...");
   glFlush();
}
void display2(){
    glClearColor(0.3,0.1,0.3,0);
    glClear(GL_COLOR_BUFFER_BIT);
    glColor3f(1.0,1.0,1.0);
    glRasterPos3f(-0.20,0.8,-1.0);
    text("***  MENU  *** ");
   // Instruction page code is mentioned here
    glColor3f(1.0,1.0,1.0);
    glRasterPos3f(-0.95,-0.9,-1.0);
    text("press <- to go back...");
    glFlush();
}
void circle(GLdouble rad){
   GLint points = 50;
   GLdouble delTheta = (2.0 * PI) / (GLdouble)points;
   GLdouble theta = 0.0;
   glBegin(GL_POLYGON);
   {       for( i = 0; i <=50; i++, theta += delTheta )
      {           glVertex2f(rad * cos(theta),rad * sin(theta));
      }    }
   glEnd();
}
void Windmill_Stand_Model(){
   glColor3f(0.38, 0.41, 0.36);
   glBegin(GL_POLYGON);
   glVertex2i(175, 150);
   glVertex2i(180, 290);
   glVertex2i(184, 290);
   glVertex2i(190, 150);
```

```
    glEnd();
}
void Windmill_Blade(){
    ///Blade_One
    ///Blade_Two
    ///Blade_Three
    glPushMatrix();
    glRotatef(spin,0,0,90);
    glBegin(GL_POLYGON);
    glVertex2i(68, -78);
    glVertex2i(0,0);
    glVertex2i(5, 5);
    glVertex2i(70, -77);
    glEnd();
    glPopMatrix();
}   ///sky///
void sky() {
    glBegin(GL_POLYGON);// Draw a Red 1x1 Square centered at origin
    glEnd();
    }
///cloud 1///
void clouds(){
    glPushMatrix();
    glColor3d(255,255,255);
    glTranslatef(3.0, 6.5, 0);
    glutSolidSphere(0.7, 250, 250);
    glPopMatrix();
    glPopMatrix();
///cloud 2///
  ///cloud 3///
   }
void sea(){
  ///sea portion///
    glBegin(GL_POLYGON);// Draw a Red 1x1 Square centered at origin
```

```
   void left_soil(){
///left soil portion///
//polygon 1
//polygon 2
//polygon 3
//polygon 4
//polygon 5
///left sidewise border partition///
 //polygon 1
 //polygon 2
 //polygon 3
//polygon 4
//polygon 5
//polygon 6
    glBegin(GL_POLYGON);// Draw a Red 1x1 Square centered at origin
    glColor3ub(0,0,0);
    glVertex2f(22.0f,-12.0f);
    glVertex2f(22.0f,-11.5f);
    glVertex2f(-12.0f,-11.5f);
    glVertex2f(-12.0f,-12.0f);
    glEnd();
    }
void create_menu(void){
     glutCreateMenu(menu);
     glutAttachMenu(GLUT_LEFT_BUTTON);
     glutAttachMenu(GLUT_RIGHT_BUTTON);
     glutAddMenuEntry("Day", 1);
     glutAddMenuEntry("Night", 2);
     //glutAddMenuEntry("Back", 4);
     glutAddMenuEntry("Stop All Movement", 3);
     glutAddMenuEntry("Exit", 5);
}
void menu(int val){
    switch (val) {
     case 1:
```

```
                    glutDisplayFunc(drawScene);
                    glutPostRedisplay();
                    break;
        case 2:
            glutDisplayFunc(drawScene2);
            glutPostRedisplay();
                break;
        case 3:
              glutDisplayFunc(move_right);
              glutPostRedisplay();
                    break;
        case 5: exit(0);
      }   }
int main(int argc, char** argv) {
    glutInit(&argc, argv);
    glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH);
    glutInitWindowPosition(0,0);
    glutInitWindowSize(1500,800);
    glutCreateWindow ("Lake View");
    create_menu();
    glutSpecialFunc(SpecialInput);
    glutTimerFunc(20, update1, 0); //Add a timer
    glutTimerFunc(20, update2, 0); //Add a timer
    glutTimerFunc(10, update3, 0);
    glutTimerFunc(20, update4, 0);
    glutDisplayFunc(display);
    glutKeyboardFunc(handleKeypress);
    glutMouseFunc(handleMouse);
    init ();
    glDisable(GL_DEPTH_TEST);
    glutMainLoop();
}
```

CHAPTER 5

SNAPSHOTS

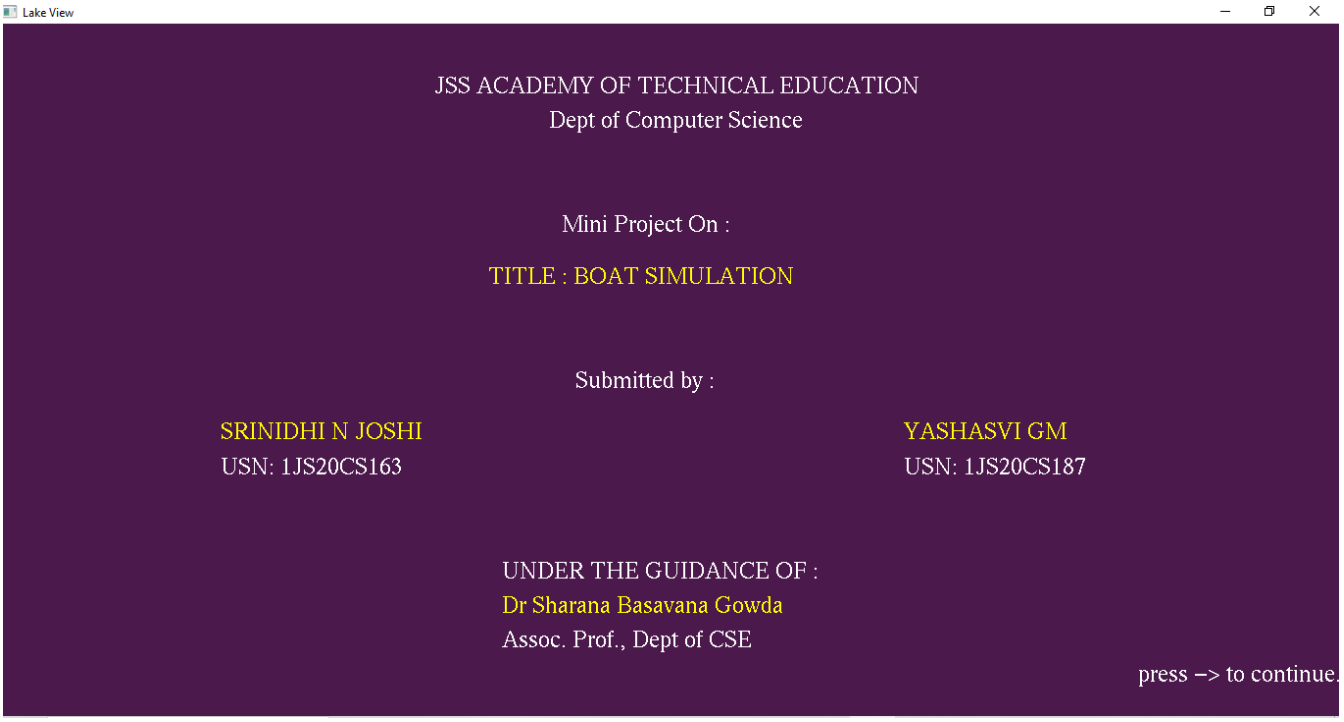

Fig.3 Introduction page

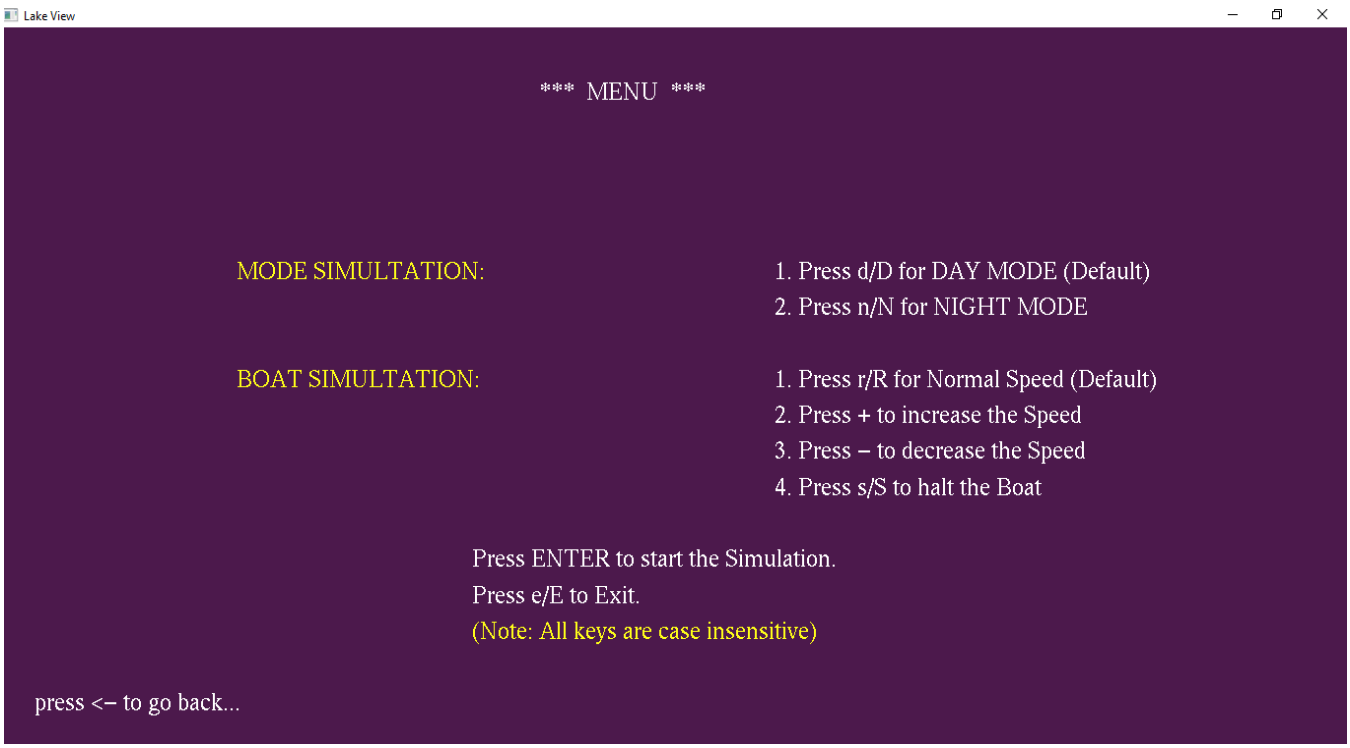

Fig.4 Instruction page

Fig.5 Day Mode

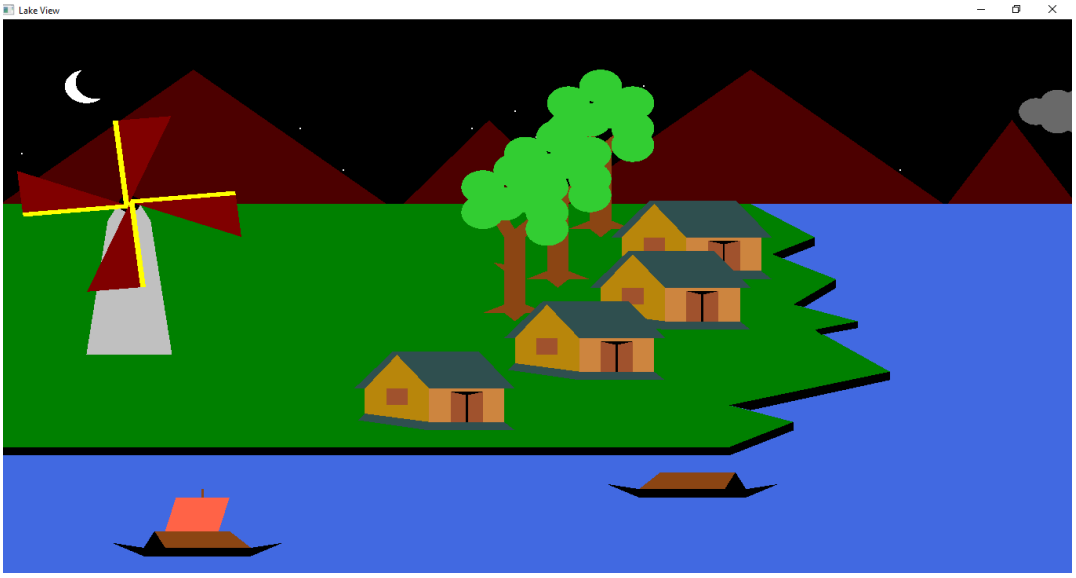

Fig.6 Night Mode

Fig.7 Menu Function

CHAPTER 6

CONCLUSION

The project showcases a scenic view of a lake with various elements, including the sun, moon, stars, clouds, hills, trees, houses, boats, and a windmill. The scene can switch between a day view and a night view based on user input.

The program features interactivity, allowing users to toggle between day and night views using the 'D' and 'N' keys, respectively. They can also control the speed of the moving objects using the '+' and '-' keys or pause the animation using the 'S' key. Moreover, the program offers a menu option to switch between day and night views, stop all movements, or exit the application.

The project provides a basic demonstration of how to create simple animations, handle user input, and organize the scene's elements using OpenGL. However, there is room for improvement and further expansion. Future enhancements could include more complex animations, interactive elements, realistic lighting and shading effects, textures, and additional objects to create a more immersive and visually appealing environment.

Overall, the "Lake View" project serves as a beginner-friendly example of using OpenGL to create a basic 2D graphical scene and lays the foundation for more complex and sophisticated OpenGL applications. It offers a glimpse into the capabilities of OpenGL as a graphics rendering library and provides a starting point for developers interested in exploring computer graphics and interactive simulations.

REFERENCES

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

[2]. Donald Hearn and Pauline Baker, Computer Graphics-OpenGL Version -2nd Edition,Pearson Education, 2003.

[3]. https://stackoverflow.com/questions/understanding-opengl

[4]. https://www.opengl.org/