CHAPTER 1:

INTRODUCTION

1.1 Introduction To Sinking Ship

The "Sinking Ship" project is an interactive computer graphics simulation developed using OpenGL (Open Graphics Library). This project aims to create a visually engaging and dynamic representation of a ship's journey, potential collision with an obstacle, and subsequent sinking process. By leveraging the capabilities of OpenGL, the simulation provides a real-time, user-controlled experience that demonstrates various concepts in computer graphics programming and interactive design.

The core of this project is a C++ program that utilizes OpenGL and GLUT (OpenGL Utility Toolkit) to render a 2D scene featuring a ship, water, and a rock obstacle. The simulation allows users to control the ship's movement, adjust its speed, and observe the consequences of a collision. Key features of the project include:

- Interactive ship control (start/stop, forward/reverse, speed adjustment)
- Dynamic water animation using Bezier curves
- Collision detection and sinking animation
- Modifiable ship structure (adjustable number of compartments)
- Menu system for instructions and controls

1.2 Objectives

The primary objectives of the Sinking Ship project are as follows:

1. To develop a visually appealing and functionally accurate simulation of a ship sinking after collision with a rock obstacle.
2. To demonstrate proficiency in using OpenGL and GLUT libraries for creating 2D graphics and animations.
3. To implement key computer graphics concepts such as geometric transformations, color manipulation, and animation timing.
4. To create a modular and well-structured codebase that allows for easy understanding and potential future enhancements.
5. To provide an interactive experience for users, allowing them to initiate and observe the simulation process.

6. To serve as a learning tool for students and enthusiasts interested in computer graphics programming and simulation techniques.

1.3 About Open GL

OpenGL (Open Graphics Library) is a standard specification defining a cross language cross platform API for writing applications that produce 2D and 3D computer graphics. The interface consists of over 250 different function calls which can be used to draw complex 3D scenes from simple primitives. OpenGL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization and flight simulation. It is also used in video games, where it competes with direct 3D on Microsoft Windows Platforms. OpenGL is managed by the non profit technology consortium, the Khronos group.INC.

Open GL serves two main purposes:

- To hide the complexities of inter facing with different3Daccelerators,By presenting programmer with a single, uniform API
- To hide the differing capabilities of hardware plat forms, by requiring that all implementations support the full OpenGL feature set.

1.4 Open GL Operation

Open GL was developed by Silicon Graphics Inc. (SGI) in 1992 and is widely used in CAD, virtual reality, scientific visualization, information visualization, and flight simulation. It is also used in video games, where it competes with Direct3D on Microsoft Windows platforms.

The interface between the application program and the graphics system can be specified through that set of function that resides in graphics library. The specification is called the APPLICATION PROGRAM INTERFACE (API).The application program sees only the API and is thus shielded from the details both the hardware and software implementation of graphics library. The software driver is responsible for interpreting the output of an API and converting these data to a form that is understood by the particular hardware.

Most of our applications will be designed to access openGL directly through functions in three libraries. Function in the main GL library have name that begin with the letter gl and stored in

the library. The second is the open GL utility Library (GLU). This library uses only GL function but contains codes for creating common object and viewing. Rather than using a different library for each system we used available library called open GL utility toolkit (GLUT). It is used as #include<glut.h>. A graphics editor is a computer program that allows users to compose and edit pictures interactively on the computer screen and save the min one of many popular“ bitmap” or “raster” a format such as TIFF, JPEG, PNG and GIF.

1.5 Primitives and Commands

Open GL draws primitives—points, line segments, or polygons-subject to several selectable modes. You can control modes independently of each other; that is, setting one mode doesn't affect whether other modes are set (although many modes may interact to determine what eventually ends up in the frame buffer). Primitives are specified, modes are set, and other OpenGL operations are described by issuing commands in the form of function calls. Primitives are defined by a group of one or more vertices. A vertex defines a point, an endpoint of line, or a corner of a polygon where two edges meet. Data (consisting of vertex coordinates, colors, normal, texture coordinates, and edge flags) is associated with a vertex, and each vertex and its associated data are processed independently.

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

CHAPTER 2

LITERATURE SURVEY

This literature survey demonstrates the rich interdisciplinary nature of the sinking ship simulation project. It draws from computer graphics fundamentals, educational technology research, marine engineering principles, and interactive design concepts. While our project implements a simplified 2D simulation, it builds upon a foundation of established research in these fields. The ongoing interest in interactive simulations for education and the continuous advancements in real-time graphics techniques suggest ample opportunities for future enhancements and extensions to this type of educational tool.

1. Ship Stability and Buoyancy

The primary factor in a ship's ability to stay afloat is its stability and buoyancy, which have been extensively studied in maritime engineering. According to Rahman et al. (2015), the stability of a ship is influenced by its hull design, weight distribution, and load conditions. The fundamental principles of Archimedes' buoyancy theory and the metacentric height concept are critical in understanding how ships float and what causes them to capsize.

2. Ship Dynamics and Sinking Simulations

Understanding ship dynamics is crucial for maritime safety and engineering. Umeda and Yamakoshi (1989) provided a comprehensive study on ship capsizing due to flooding, which forms the theoretical basis for many sinking ship simulations. Their work highlighted the importance of compartmentalization in ship design for preventing rapid sinking.

More recently, Gao et al. (2011) presented a real-time simulation of ship sinking based on SPH (Smoothed Particle Hydrodynamics). While their approach uses more advanced fluid dynamics compared to our project, it underscores the ongoing interest in realistic ship sinking simulations.

3. Water Animation Techniques

Realistic water animation has been a challenging aspect of computer graphics. Fournier and Reeves (1986) introduced one of the early techniques for simulating ocean waves using a sum of trochoids. While computationally intensive, their method laid the groundwork for future developments in water animation.

Bezier curves, which we use in our project for water animation, have been widely applied in computer graphics since their introduction by Pierre Bezier in the 1960s. Prautzsch et al. (2002) provide a thorough mathematical treatment of Bezier curves in their book "Bezier and B-Spline Techniques," explaining their properties and applications in computer-aided geometric design.

4. User Interaction in Simulations

The design of user interactions in simulations plays a crucial role in their effectiveness. Shneiderman and Plaisant (2010) in their book "Designing the User Interface" emphasize the importance of direct manipulation interfaces for interactive systems. This principle is reflected in our project's design, where users can directly control the ship's movement and properties.

5. Collision Detection in Graphics Applications

Collision detection is a fundamental problem in computer graphics and simulations. Lin and Gottschalk (1998) provide a survey of collision detection algorithms for various applications. While our project uses a simplified approach, more complex simulations often employ hierarchical bounding volume techniques or spatial partitioning methods for efficient collision detection.

6. Real-time Graphics Techniques

Real-time rendering techniques are crucial for interactive simulations. Akenine-Möller et al. (2018) provide a comprehensive overview of real-time rendering techniques in their book "Real-Time Rendering." While our project uses relatively simple 2D graphics, many of the principles discussed in their work, such as efficient update mechanisms and double buffering, are applicable to our implementation.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Hardware Requirements

The selection of hardware is very important in the existence and proper working of any software. In the selection of hardware, the size and the capacity requirements are also important. The Web Based Manufacturing System can be efficiently run on Pentium system with at least 128 MB RAM and Hard disk drive having 20 GB. Floppy disk drive of 1.44 MB and 14 inch color monitor suits the information system operation.

Processor : Intel-Corei5, i7

Processor Speed : 2.0GHz

RAM : 1GB or more

HardDisk : 40GBto 80GB

3.2 Software Requirements

Data Communication Over internet is graphically designed in such a way that the user can get the better visual experience. The Sender and the receiver are the two users of the project where the data packet and the acknowledgement would transmit from one another. This requirement has made the programming and the communication easier.

Operating System : Windows (7,10,11)

Programming Language : C++

Graphics Library : GL/glut.h

API : OpenGL2.0

CHAPTER 4

IMPLEMENTATION

4.1 Open GL Graphics Architecture

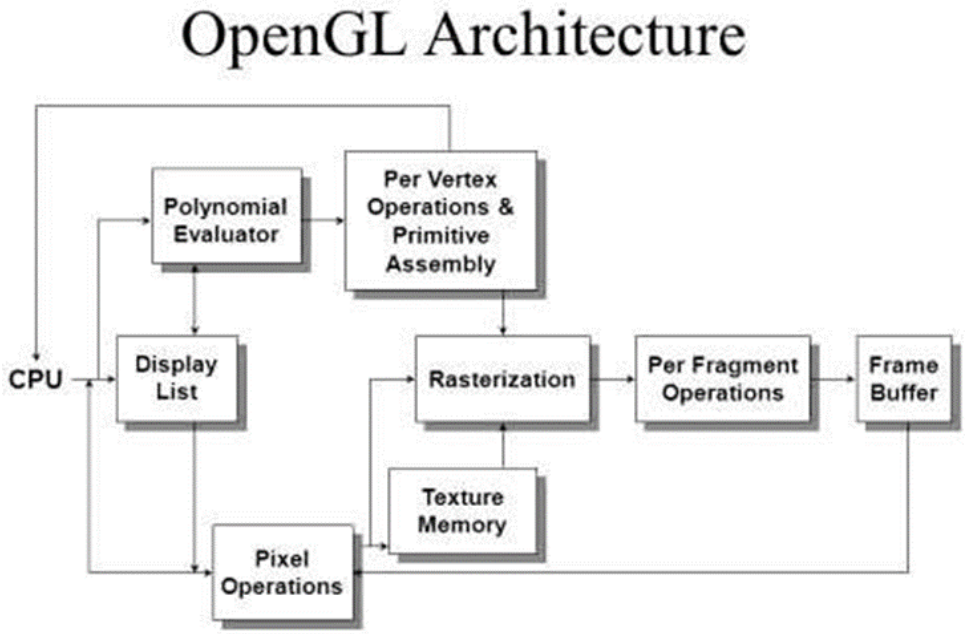

Fig 4.1: Open GL Graphics architecture

In proposed system, the OpenGL is a graphic software system designed as a streamlined, hardware-independent interface to be implemented on many different hardware platforms. To achieve these qualities, no commands for performing windowing tasks or obtaining user input are included in OpenGL; instead, you must work through whatever windowing system controls the particular hardware you're using. Open GL doesn't provide high-level commands for describing models of three-dimensional objects. Such commands might allow you to specify relatively complicated shapes such as automobiles, parts of the body, airplanes, or molecules. With OpenGL, you must build up your desired model from a small set of geometric primitives - points, lines, and polygons.

4.2 Open GL functions

This program is implemented using various Open GL functions which are shown below.

glClear(GL_COLOR_BUFFER_BIT): Clears the color buffer.

glPushMatrix(): Saves the current transformation matrix on a stack.

glPopMatrix(): Restores the previously saved transformation matrix from the stack.

glTranslated(): Applies a translation transformation.

glRotated(): Applies a rotation transformation.

glScaled(): Applies a scaling transformation.

glColor3f(): Sets the current color for drawing operations.

glBegin(GL_POLYGON): Starts defining a polygon.

glVertex2f(): Specifies a vertex for the current polygon or line.

glEnd(): Ends the definition of a polygon or line.

glPointSize(): Sets the width of points to be drawn.

glEnable(): Enables various OpenGL capabilities.

glBlendFunc(): Specifies pixel arithmetic for blending.

glClearColor(): Sets the color used when clearing the color buffer.

glMatrixMode(): Specifies which matrix stack is the target for subsequent matrix operations.

glLoadIdentity(): Replaces the current matrix with the identity matrix.

glutInit(): Initializes the GLUT library.

glutInitDisplayMode(): Sets the initial display mode.

glutInitWindowSize(): Sets the initial window size.

glutInitWindowPosition(): Sets the initial window position.

glutCreateWindow(): Creates a top-level window.

glutDisplayFunc(): Sets the display callback for the current window.

glutKeyboardFunc(): Sets the keyboard callback for the current window.

glutMouseFunc(): Sets the mouse callback for the current window.

glutTimerFunc(): Registers a timer callback to be triggered after a specified number of milliseconds.

glutMainLoop(): Enters the GLUT event processing loop.

glutPostRedisplay(): Marks the current window as needing to be redisplayed.

glutSwapBuffers(): Swaps the buffers of the current window if double buffered.

gluOrtho2D(): Defines a 2D orthographic projection matrix.

4.3 Interaction with the Program

1. Initialize:

Set initial values for ship position, speed, and state variables

Initialize cloud positions randomly

2. Main Game Loop:

Clear the screen

Draw the sky and clouds

If in main menu:

Display menu text and instructions

Else if ship has sunk completely:

Reset the game

Else:

If ship hasn't reached the rock:

Display normal ship sailing

If ship is approaching the rock:

Display ship near rock

If ship has crashed:

Display broken ship sinking

Update ship position based on current state and user input

Move clouds horizontally

Swap buffers and refresh display

User Input Handling:

Keyboard input:

Space: Start/stop ship movement

'd': Move ship forward

'a': Move ship backward

'+': Increase ship speed

'-': Decrease ship speed

',': Decrease ship compartments

'.': Increase ship compartments

ESC: Exit game

Mouse input:

Left click: Start game from main menu

4. Drawing Functions:

drawSky(): Draw background sky color and clouds

ship(): Draw the intact ship

broken_ship(): Draw the ship breaking and sinking

rock(): Draw the rock obstacle

floatingWater(): Draw animated water using Bezier curves

drawCloud(): Draw individual clouds

5. Animation and Physics:

Update ship position and rotation based on its state (sailing, crashing, sinking)

Animate water using sine waves

Move clouds continuously from right to left

6. Collision Detection:

Check if ship has collided with the rock

If collided, initiate sinking sequence

7. Game State Management:

Track game states: main menu, playing, crashed, sinking

Manage ship compartments and their effect on sinking speed

8. Rendering:

Use OpenGL functions to draw shapes and apply colors

Implement basic transformations (translation, rotation, scaling) for objects

9. Main Function:

Initialize OpenGL and GLUT

Set up window and display mode

Register callback functions for display, keyboard, mouse, and timer

Enter main event loop

This algorithm outlines the high-level structure and main components of the Sinking Ship project. It covers the game loop, user interaction, drawing routines, animation, and overall game logic.

4.4 Pseudocode

Initialize global variables and constants

Function initializeClouds():

For each cloud

Set random x position

Set random y position between 600 and 750

Function update(value):

If ship is started:

Update ship position based on direction and velocity

Move clouds to the left

If a cloud moves off-screen:

Reset its position to the right side

Schedule next update

Function display():

Clear screen

Draw sky and clouds

If in main menu:

Display menu

Else if ship has sunk completely

Reset values

Else:

If ship is before rock:

Display normal ship

Else if ship is near rock:

Display ship approaching rock

Else:

Display broken ship sinking

Swap buffers

Function drawShip():

Draw main base of ship

Draw windows and compartments

Draw chimneys

Function drawBrokenShip():

Draw main base with broken parts

Apply rotation to simulate sinking

Draw separated parts of the ship

Function drawRock():

Draw rock shape

If ship has crashed:

Apply transformations to rock parts

Function drawWater():

Use Bezier curves to create wave effect

Apply color and transparency

Function keyboard(key):

Handle user input for:

Starting/stopping ship

Changing direct

Adjusting speed

Modifying compartments

Exiting program

Function mouse(button, state, x, y):

If left click in main menu

Exit main menu

Function displayMenu():

Display game title, credits, and instructions

Main function:

Initialize OpenGL

Set up window

Initialize game value

Initialize clouds

Set up display, keyboard, and mouse callbacks

Start main loop

CHAPTER 5

SNAPSHOTS

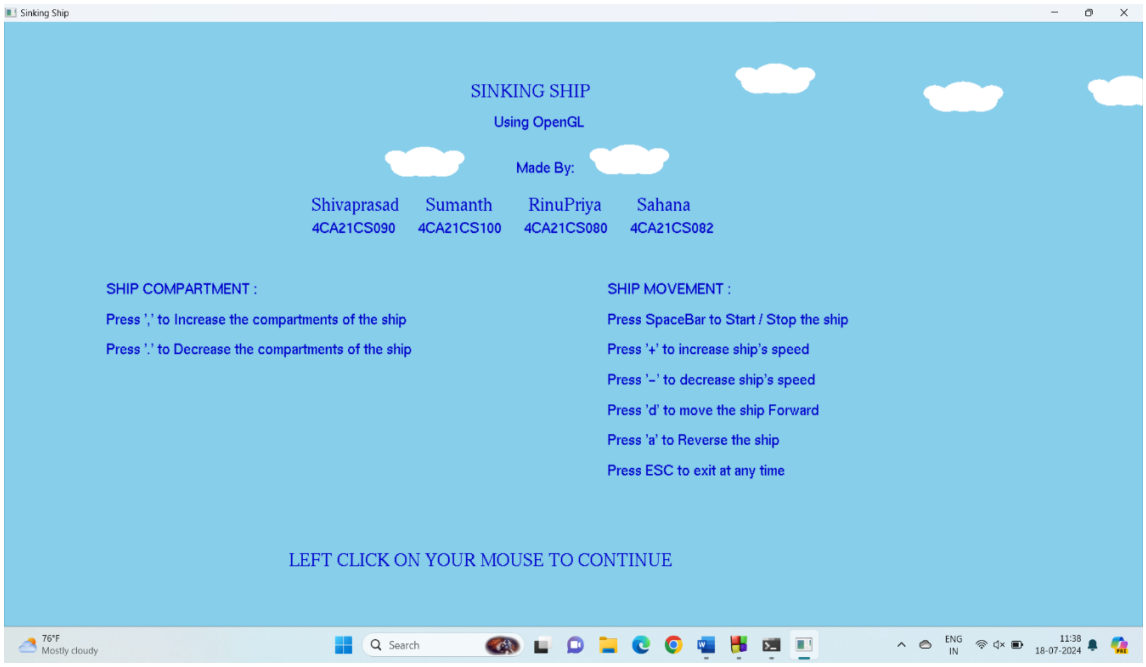

Fig 5.1: Instructions to move the ship

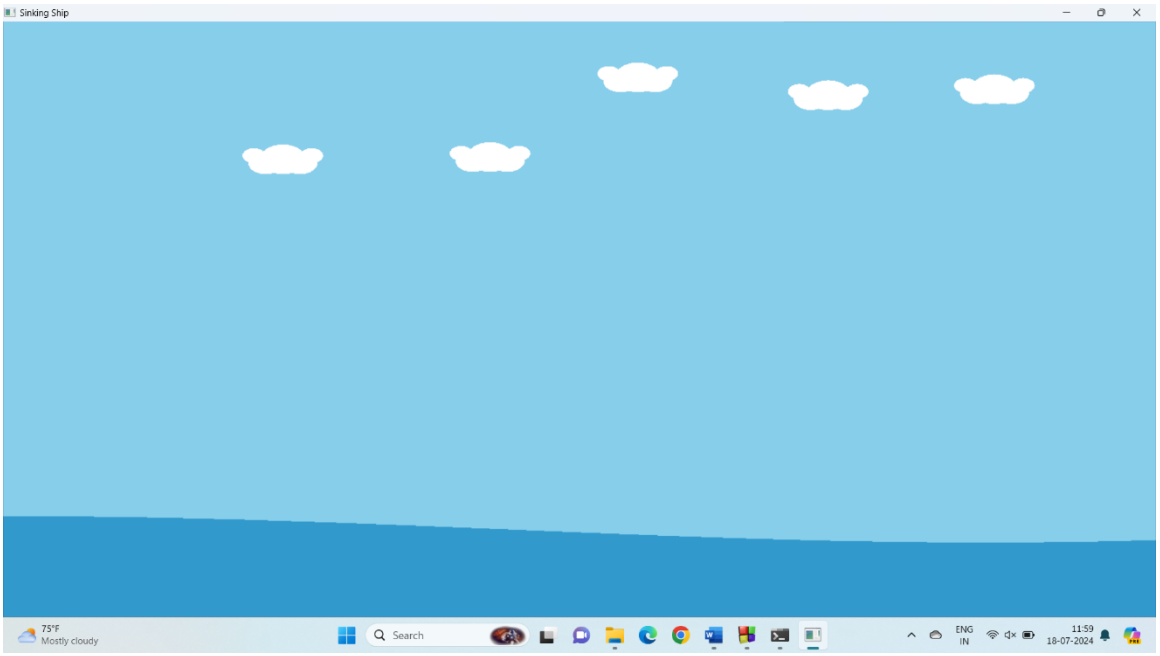

Fig 5.2: view of Sky and Clouds

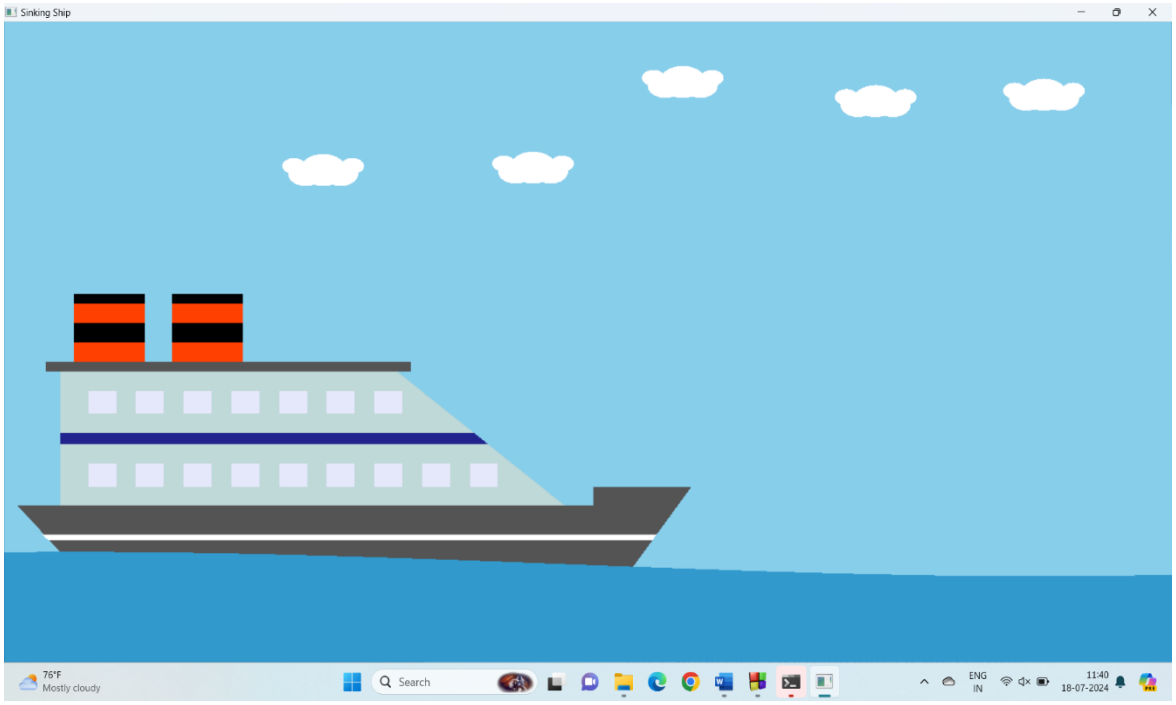

Fig 5.3: view of ship

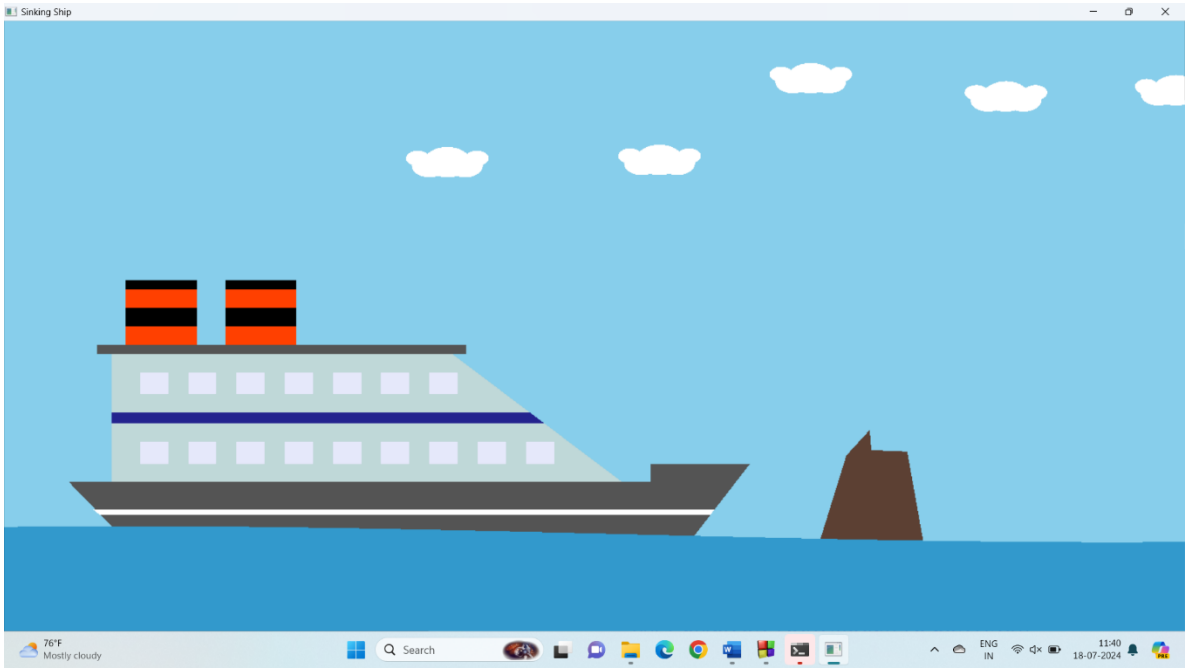

Fig 5.4: view of rock

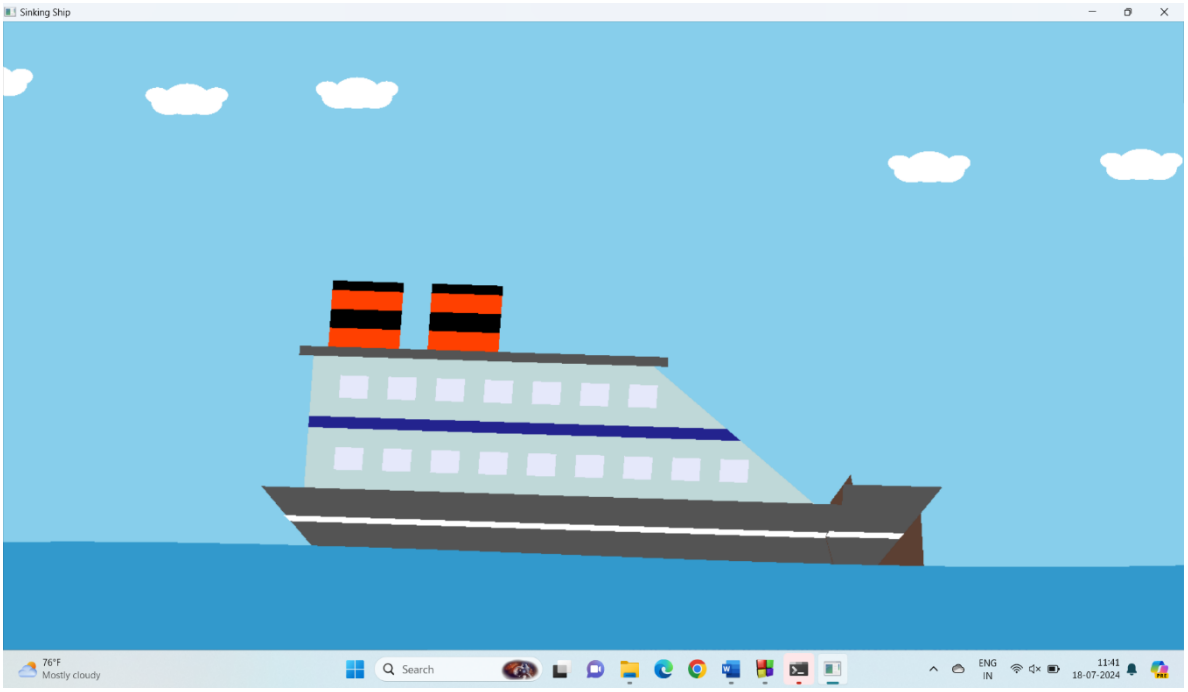

Fig 5.5: Ship while hitting the Rock

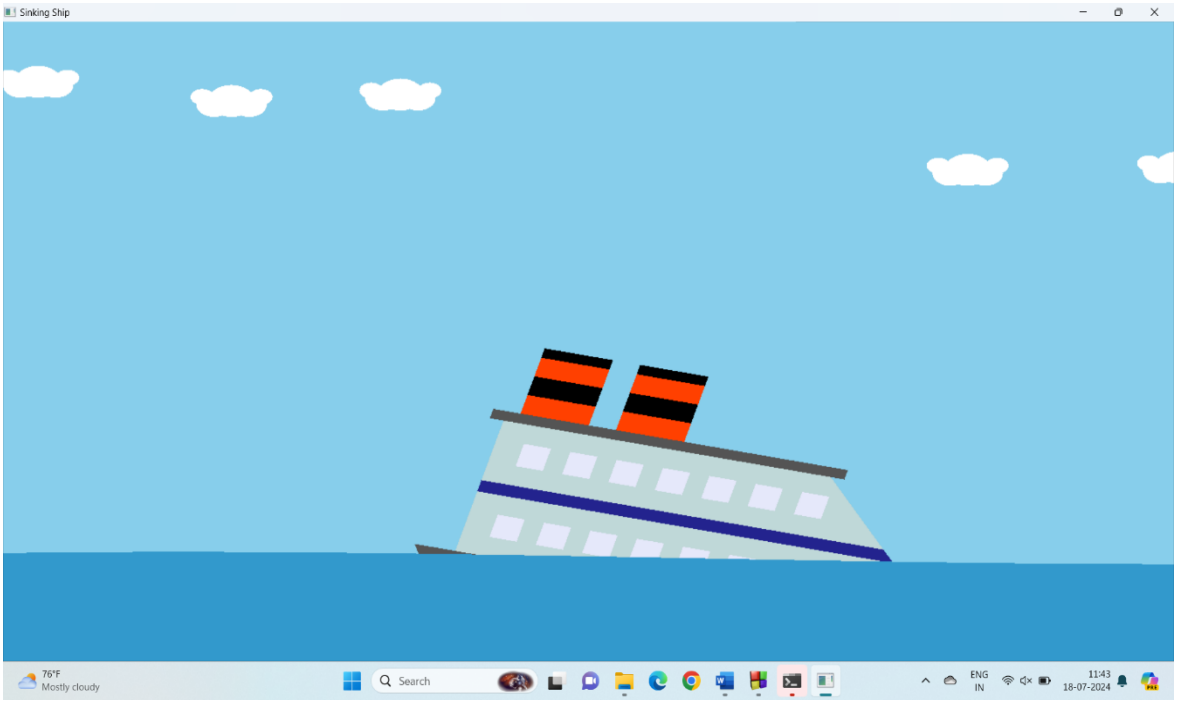

Fig 5.6: Sinking Ship