# Computer Graphics (UCS505)

# Project on

# Seaport City View

**Submitted By:**

Ridhi Thakur (102117032)

Radhika Goel (102117036)

Ira Gupta (102117057)

# Group No. 3CS2

**B.E. Third Year – COE**

**Submitted To:**

Ms. Jyoti Rani



**Computer Science and Engineering Department**

**Thapar Institute of Engineering and Technology**

**Patiala – 147001**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **S. No.** | **Description** | **Page No.** |
| 1. | Introduction to Project | 3 |
| 2. | Computer Graphics concepts used | 4 |
| 3. | User Defined Functions | 5 |
| 4. | Built-in Functions | 6 |
| 5. | Code | 7 |
| 6. | Output/ Screen shots | 7 |

**Introduction to Project**

Overview:

The “Seaport City View” project is a captivating venture in computer graphics, utilizing the OpenGL library and C++ programming language to construct a visually rich representation of a thriving urban environment centered around a bustling seaport. This project delves into the core principles of graphics programming, encompassing the rendering of 2D elements such as buildings, roads, and water bodies, alongside the implementation of algorithms for drawing geometric shapes like circles. Through the adept use of OpenGL’s capabilities, the project aspires to create a dynamic and interactive cityscape.

Technologies Used:

1. OpenGL: A versatile graphics library renowned for rendering both 2D and 3D graphics, providing tools for creating visually captivating simulations.
2. C++: A robust and efficient programming language chosen for its suitability in graphics programming and ability to handle complex simulations.

Key Features of the Simulation:

* Detailed 2D Rendering: Utilizing OpenGL to render intricate 2D elements such as buildings, roads, bridges, ships, and seaport facilities, capturing the essence of an urban environment.
* Geometric Shape Drawing: Implementing algorithms for drawing geometric shapes like circles, squares, and polygons to add depth and realism to the cityscape.
* Texturing and Lighting Effects: Incorporating textures and lighting effects to enhance visual aesthetics, creating a lifelike portrayal of the city at different times of the day.
* Dynamic Simulation: Simulating dynamic elements such as moving vehicles, changing weather conditions, and animated water bodies to add realism and interactivity.

Purpose:

The simulation serves multiple purposes:

* Virtual Exploration: Offering users a virtual experience of navigating through a bustling seaport city, exploring its architecture, infrastructure, and maritime activities.
* Educational: Providing a platform for educational institutions to study urban planning, architecture, and maritime logistics in a simulated environment.
* Entertainment: Engaging users with an immersive and interactive cityscape experience, suitable for entertainment and recreational purposes.

Usage:

The simulation can be experienced on compatible systems supporting OpenGL and a C++ development environment. Users can navigate through the virtual cityscape, interact with elements, and explore the intricacies of urban life around a seaport setting.

Further Development:

Future development of the simulation may include:

* Enhanced Visual Realism: Improving textures, lighting effects, and environmental details to create a more realistic and immersive experience.
* Expanded Interactivity: Adding interactive elements such as NPCs (non-playable characters), interactive objects, and dynamic events to enhance user engagement.
* Multiplayer Support: Introducing multiplayer functionality for collaborative exploration and social interactions within the virtual cityscape.
* Mobile Compatibility: Optimizing the simulation for mobile platforms to reach a wider audience and enable on-the-go exploration of the virtual city.

**Computer Graphics Concepts Used**

The computer graphics concepts used in the project include:

1. Rendering: Utilizing OpenGL for rendering 2D elements such as buildings, roads, bridges, ships, and seaport facilities, as well as implementing textures and lighting effects to enhance visual aesthetics.
2. Geometric Shape Drawing: Implementing algorithms for drawing geometric shapes like circles, squares, and polygons to add depth and realism to the cityscape.
3. Texturing: Applying textures to surfaces to simulate different materials such as concrete for buildings, asphalt for roads, and water textures for the seaport area.
4. Lighting: Incorporating lighting effects such as ambient lighting, directional lighting, and specular highlights to create a realistic portrayal of the cityscape under varying lighting conditions.
5. Animation: Simulating dynamic elements such as moving vehicles, changing weather conditions, and animated water bodies to add realism and interactivity to the environment.
6. Collision Detection: Implementing algorithms for detecting collisions between objects within the simulation, ensuring realistic interactions between elements such as ships, vehicles, and buildings.
7. User Interaction: Providing user interaction capabilities to navigate through the virtual cityscape, interact with objects, and explore different areas of the urban environment.
8. Optimization: Optimizing rendering performance and resource usage to ensure smooth and efficient operation of the simulation, especially when handling complex scenes with multiple elements.

These concepts collectively contribute to creating a visually immersive and interactive simulation of a bustling urban environment around a seaport, offering users a realistic and engaging experience of exploring a virtual cityscape.

**User Defined Functions**

These are the user-defined functions that are likely used in this project. Here's an explanation of each function listed:

1. ‘water’: This function likely handles the rendering or simulation of water bodies within the graphics scene, such as rivers, lakes, or oceans.
2. ‘beach\_bench’: This function could be responsible for drawing or simulating beach benches, typically found near beaches or waterfront areas.
3. ‘umbrella’: This function likely deals with rendering or animating umbrellas, commonly seen at beach resorts or outdoor locations for shading.
4. ‘wave’: This function probably handles the animation or rendering of waves in water bodies, adding realism to scenes with water elements.
5. ‘road’: This function may handle the rendering or simulation of roads within the cityscape, including road markings, lanes, and traffic elements.
6. ‘sky’: This function likely deals with rendering the sky background, including clouds, sun, moon, or stars, depending on the time of day in the scene.
7. ‘bench’: This function could handle the rendering or simulation of benches, typically found in parks, gardens, or public spaces.
8. ‘lamppost’: This function may handle rendering lampposts or streetlights, adding illumination and realism to the cityscape.
9. ‘left\_car’ and ‘right\_car’: These functions likely handle the rendering or simulation of cars moving on roads within the cityscape.
10. ‘tree’: This function could be responsible for rendering or simulating trees within the environment, adding greenery and natural elements to the scene.
11. ‘building\_1’ to ‘building\_7’: These functions may handle the rendering or simulation of different types of buildings within the cityscape, each with its design and characteristics.
12. ‘cloud’: This function likely handles the rendering or simulation of clouds in the sky background, adding dynamic and realistic cloud formations.
13. ‘sun’ and ‘moon’: These functions may handle rendering the sun or moon in the sky background, depending on the time of day in the scene.
14. ‘boat’, ‘boat\_1’, ‘ship’: These functions likely handle rendering or simulating boats and ships in water bodies such as harbors or ports.
15. ‘containers’: This function may handle rendering or simulating containers on ships or docks in the seaport area.
16. ‘first\_blade’, ‘second\_blade’: These functions could handle rendering or animating blades of a windmill or turbine.
17. ‘windmill\_stand’: This function may handle rendering or simulating the stand or structure of a windmill or turbine.
18. ‘air\_plane’: This function likely handles rendering or simulating airplanes in the sky, adding aerial elements to the scene.
19. ‘speed\_board’: This function may handle rendering or simulating speed boards or speedometers for vehicles.
20. ‘update\_blade\_1’ to ‘update\_right\_car\_1’: These functions likely handle updating the state or position of various animated elements in the scene, such as blades of a windmill, cars, boats, or airplanes, to create dynamic and interactive animations.
21. ‘identity\_axis’: This function could set up the initial coordinate system or axis orientation for the graphics scene.
22. ‘display’, ‘display1’: These functions likely contain the main drawing logic for rendering the graphics scene and may handle multiple display scenarios or views.
23. ‘handleKeypress’: This function may handle keyboard input events, such as key presses, for user interaction with the graphics scene.
24. ‘main’: This is typically the main entry point of the program, where initialization, setup, and event handling are done to run the graphics application.

These functions collectively contribute to creating a dynamic and interactive graphics scene, depicting elements of a bustling urban environment around a seaport in the "Seaport City View" project.

**Built-in Functions**

There are several standard library functions used. Here are some of them:

1. glutInit(&argc, argv);: Initializes GLUT, passing the command-line arguments argc and argv to GLUT. This is typically done at the beginning of a program to set up the GLUT environment.
2. glutCreateWindow("Seaport City View");: Creates a window with the title "Seaport City View" using GLUT. This function sets up the main window where the graphics will be displayed.
3. glutInitWindowSize(800, 400);: Sets the initial size of the window to 800 pixels wide and 400 pixels high. This determines the dimensions of the graphics viewport.
4. glutDisplayFunc(display);: Registers the display function as the callback function that GLUT will call whenever the window needs to be redrawn. This function typically contains the main drawing logic for the graphics scene.
5. glutKeyboardFunc(handleKeypress);: Registers the handleKeypress function as the callback function for handling keyboard events. This function is called when a key is pressed while the window is active.
6. identity\_axis();: This function may be a user-defined function that sets up the initial state of the graphics scene, such as setting up the coordinate system or initializing variables.
7. glutTimerFunc(...): Registers timer functions that will be called repeatedly after a specified interval. These timer functions are used for animation and updating the state of dynamic elements in the graphics scene.
8. glutMainLoop();: Enters the GLUT event processing loop, where GLUT handles window events such as keyboard input, mouse input, and window resizing. This function keeps the program running and responsive to user interactions.

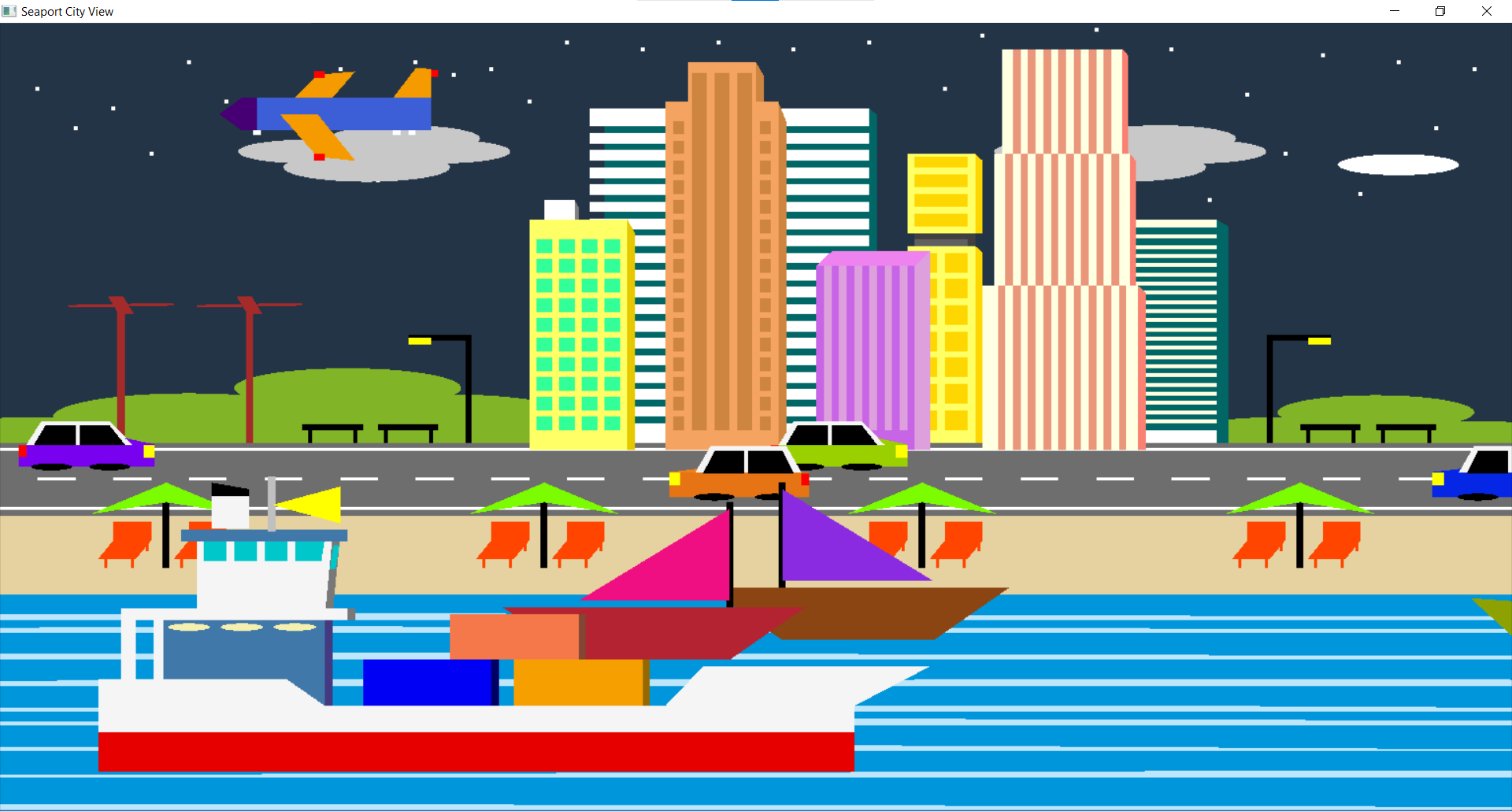
**Output/ Screenshots**

These are the snapshots of our project:

1. **DAY**

****

1. **NIGHT**

****

1. **RAIN**

****