American International University – Bangladesh

COMPUTER GRAPHICS

Spring 24-25

Section: H

Project Submission on

Changing Seasons: Urban Scenery Transformation.

Faculty Name: Dipta Justin Gomes

Group-F member List:

Name	ID	
JALAL UDDIN	22-46356-1	
NAZMUL HAQUE NOYON	22-49988-3	
ANKUR PRAMANIK	21-45452-3	
MD AREFIN ISLAM	22-48766-3	

Submission Date: 29/06/2025

Abstract

This report details the design and implementation of "Changing Seasons: Urban Scenery Transformation" a comprehensive 2D computer graphics project developed using C++ and the OpenGL Utility Toolkit (GLUT). The project's core objective is to demonstrate fundamental graphics principles through the creation of four distinct, animated scenes: a dynamic city, a tranquil sea journey, a rocket launch station, and a futuristic cityscape. A key feature of the primary city scene is its ability to transition through six different seasonal and temporal states Evening, Night, Morning, Spring, Autumn, and Winter each with unique color palettes, lighting, and weather effects like rain, snow, and falling leaves. The application employs procedural generation for all visual assets, utilizing OpenGL's fixed-function pipeline to render primitives. Animations are managed through a delta-time-based update loop for smooth, frame-rate-independent motion. User interaction via keyboard and mouse allows for scene switching and control over various animated elements, showcasing a robust event-handling system.

Keywords: Computer Graphics, OpenGL, GLUT, 2D Animation, Procedural Generation, Seasonal Transformation, Urban Scenery, User Interaction, C++

Introduction

The field of computer graphics provides a powerful medium for blending artistic creativity with technical precision. This project, "Changing Seasons: Urban Scenery Transformation," serves as a practical exploration of the foundational concepts that underpin 2D graphics and animation. The primary focus is the creation of a dynamic and interactive application that simulates various environments, with a special emphasis on the atmospheric changes brought about by different seasons and times of day.

The motivation behind this project was to develop a comprehensive application that goes beyond static imagery to create living, breathing worlds. By implementing features such as procedural weather effects (rain, snow), animated objects (vehicles, celestial bodies), and user-driven state changes, the project provides a robust demonstration of core graphics programming techniques. It was an opportunity to master 2D transformations, color theory application, state management, and event-driven programming within the C++/OpenGL ecosystem.

The principal goal was to construct four visually distinct and interactive scenes. The main city scene was designed to be the most feature-rich, with objectives including: the implementation of six unique environmental themes (Evening, Night, Morning, Spring, Autumn, Winter); the creation of smooth, time-corrected animations for elements like cars, a helicopter, and a boat; and the integration of dynamic weather systems corresponding to the selected season. Secondary goals included developing three additional scenes a sea journey, a rocket station, and a futuristic world to showcase versatility in theme and animation styles, and to implement intuitive keyboard and mouse controls for a compelling user experience.

This report outlines the entire development process. The "Related Work" section discusses the conceptual and technical inspirations for the project. The "Tools, Technologies & Libraries" section details the software and programming components used. "System Design and Implementation" provides an in-depth look at the project's architecture and key algorithms. Finally, "Results & Demonstration" showcases the final visual output, followed by a "Conclusion" that summarizes achievements, limitations, and potential future enhancements.

Related Work

This project draws inspiration from a variety of sources, primarily classic 2D animated films and early simulation video games. The aesthetic goal of creating distinct moods through color and motion is heavily influenced by the environmental storytelling seen in 2D animation, where backgrounds are not merely static backdrops but active participants in the narrative. Games like the original SimCity provided conceptual inspiration for building a miniature, observable urban environment where different elements operate on independent cycles.

Technically, the project is built upon the foundational principles of the OpenGL fixed-function pipeline, a paradigm that was central to graphics programming for many years. The core rendering loop, which clears the screen, draws all scene elements, and swaps the buffer, is a standard pattern in real-time graphics applications. The animation system is based on a time-step methodology, where object positions are updated based on the elapsed time since the last frame (delta-time). This is a well-established technique to ensure that animation speed is independent of the computer's processing power. The GLUT library is employed for window and event management, following a common model used in countless educational and introductory graphics projects for its simplicity and cross-platform nature. While no specific external codebases were used, the project leverages these well-known and documented graphics programming patterns and libraries.

Tools, Technologies & Libraries

The development of this project relied on a specific set of tools and libraries chosen for their robustness, industry prevalence, and suitability for foundational graphics programming.

Programming language used:

The entire application is written in C++. This language was selected for its high performance, low-level memory management capabilities, and its strong compatibility with the OpenGL library, making it an industry standard for graphics-intensive applications.

Graphics libraries:

OpenGL (Open Graphics Library) This is the core graphics API used for all rendering tasks. The project specifically utilizes the fixed-function pipeline of OpenGL to draw 2D geometric primitives such as GL_QUADS, GL_TRIANGLES, GL_POLYGON, and GL_LINES. All visuals, from the intricate buildings to the particle-based weather effects, are generated procedurally using these fundamental commands.

GLUT (OpenGL Utility Toolkit): GLUT serves as the supporting framework for OpenGL. It handles operating system-level tasks, including creating the window, managing display modes, and processing user input from the keyboard and mouse. Crucially, GLUT provides the glutTimerFunc, which is the cornerstone of the animation loop, enabling periodic updates to the scene.

Development environment:

The project was developed in a standard C++ environment. A text editor like Xcode, Code Block was used for writing the code, and compilation was handled by the GCC/G++ compiler (MinGW/TDM-GCC on Windows or standard GCC on macOS/Linux). The code is written to be cross-platform and can be compiled and run on Windows, macOS.

Any assets used:

No external assets were used in this project. All visual elements, including buildings, vehicles, characters, and environmental details, are procedurally generated at runtime using OpenGL drawing commands. This approach was chosen to focus on the core principles of 2D graphics programming rather than asset management.

System Design and Implementation

The project is architected as a state-driven application within a single C++ source file. The system's state is primarily managed by two enumerations: ActiveScene tracks which of the four scenes is currently being displayed, and Season controls the theme of the main city scene.

Scene Description:

The four scenes (City, Sea, Rocket, Future) are composed of procedurally drawn objects. Each object is rendered by a dedicated function (e.g., drawBuildings(), drawCars(), sea_drawShip(). The City scene is the most complex, layering multiple static and dynamic elements: a multi-layered sky that changes color, buildings with seasonal snow, trees that change foliage, a river with

animated waves, a park with benches, a road with moving cars, and a user-controllable helicopter. The other scenes use similar principles but with unique thematic elements and color palettes.

User Interface:

The user interface is entirely keyboard and mouse-driven. A central keyboard() function routes input based on the current ActiveScene. Global keys (C, S, R, F) switch between scenes. Scene-specific keys control animations and themes, such as 1-6 for seasons in the city, m/q/o/a for animations in the sea, and n/b for themes in the future city. The future scene also utilizes mouse clicks to control the main UFO's flight path.

Challenges & Solutions:

Challenge: Frame-Rate Independent Animation: A significant challenge was ensuring animations run at a consistent speed regardless of the computer's performance. The solution was to implement a delta-time calculation in the main update() function. By measuring the time elapsed since the last frame, all movement speeds (CLOUD_SPEED, CAR1_SPEED, etc.) are multiplied by this delta-time value. This ensures that objects move by the same amount over a given period, resulting in smooth, predictable motion on any hardware.

Challenge: State Management: Managing the visual and logical state for four distinct scenes and multiple sub-themes was complex. This was solved by using enums as state flags. The main display() and update() functions act as routers, using if/else or switch statements based on the currentSceneView variable to call the correct drawing and logic functions. This modular approach, though within a single file, keeps the logic for each scene separated and manageable.

Results & Demonstration

The final application successfully meets all the project objectives, resulting in a dynamic and interactive multi scene 2D graphics showcase. The following screenshots demonstrate the key features and visual outcomes.

The Urban City Scene:

The primary "City Scene" serves as the project's core, demonstrating extensive state management and procedural generation. As shown in Figures 1 through 6, the scene can be dynamically transformed through six distinct themes: Evening, Night, Morning, Spring, Autumn, and Winter. Each theme not only alters the color palette of the sky, water, and ambient lighting but also introduces unique, procedurally generated particle effects. The 'Spring' theme features gentle

rainfall (Figure 4), the 'Autumn' theme showcases falling leaves with a fluttering motion (Figure 5), and the 'Winter' theme covers the landscape and buildings in snow while a flurry falls from the sky (Figure 6).

The Rocket Station Scene:

This scene showcases the ability to create a more complex, industrial environment with multiple coordinated moving parts. The visual output (Figure 8, 9 and 10) displays a detailed rocket on its launchpad, a multi-level flyover with cars, and a detailed background of hills and houses. The key result here is the successful implementation of several concurrent animations: the rocket itself has an animated launch sequence, cars move along the flyover with rotating wheels, and a boat navigates the river in the foreground. This scene effectively demonstrates the management of numerous moving elements within a single, cohesive view.

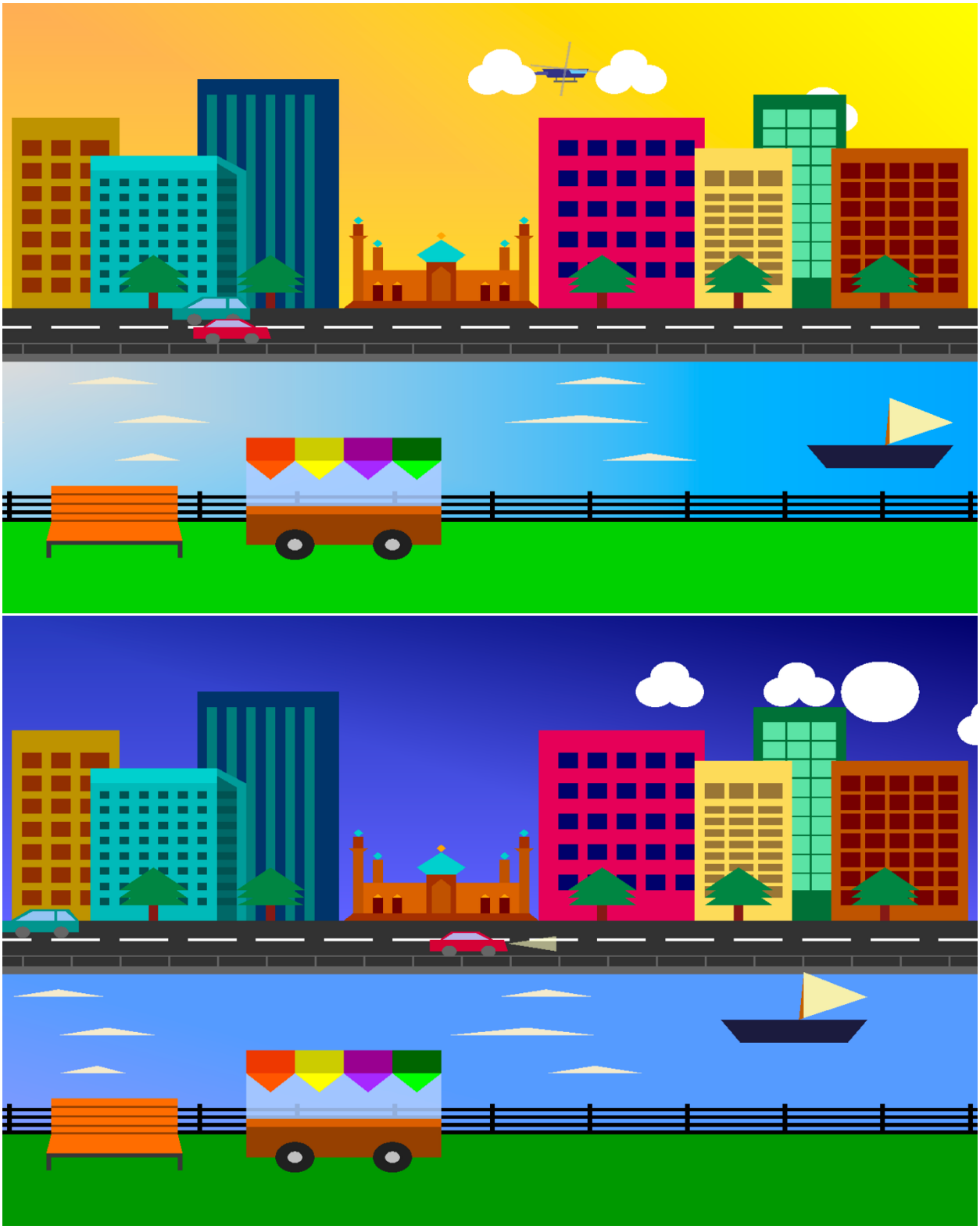
The Futuristic City Scene:

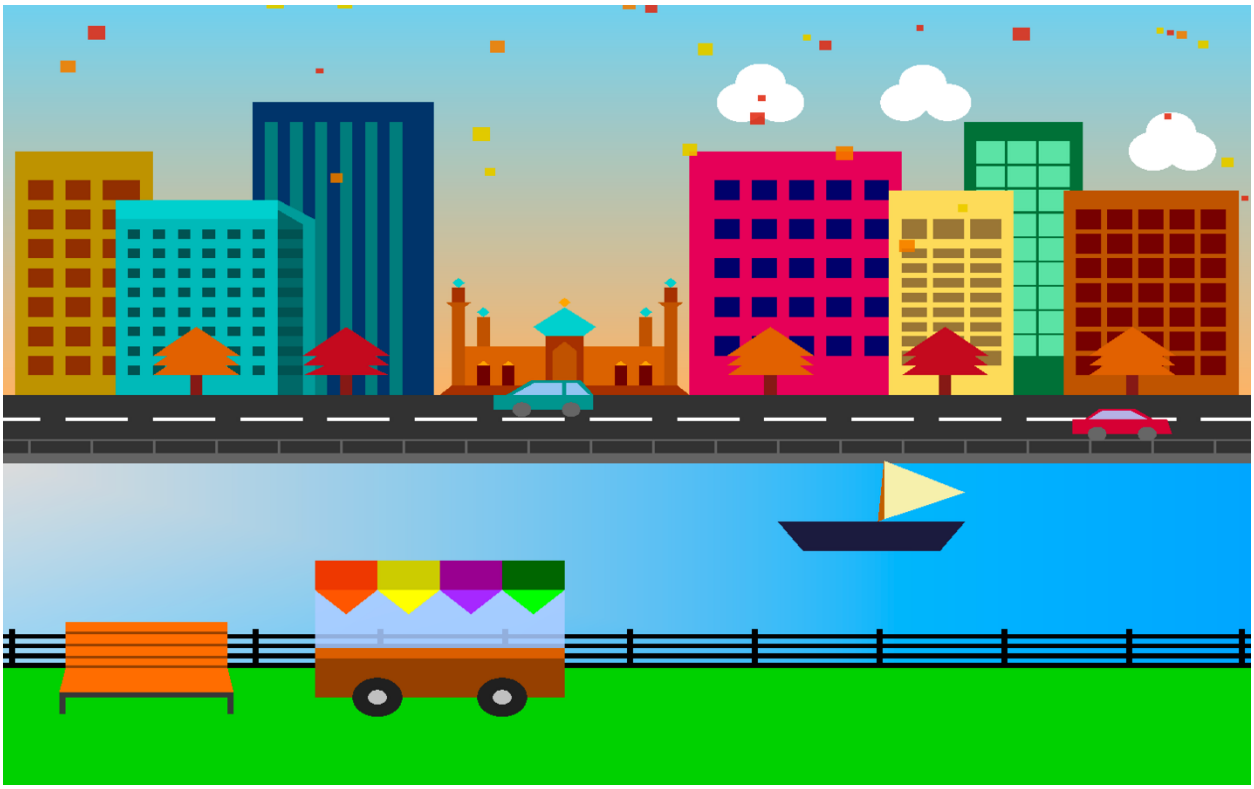
The "Future City Scene" was designed to create a stylized, science-fiction world and to integrate mouse-based interactivity. The scene features a unique visual style with glowing silhouettes and a celestial sky (Figure 11, 12 and 13). The primary animations consist of multiple UFOs moving across the screen at different speeds and on different paths. The key interactive result, shown in Figure 13, is the main UFO in the foreground, whose flight path (ascending or descending) is directly controlled by the user’s left and right mouse clicks. This successful integration of mouse input for direct object manipulation highlights another dimension of the project's event-handling capabilities.

The Sea Journey Scene:

The "Sea Journey Scene" was implemented to explore a different environmental aesthetic, focusing on a tranquil coastal landscape. The results show a successful creation of a day/night cycle, which can be toggled by the user (Figures 14, 15, 16 and 17). The scene features multiple, independent animations, including a large ship traversing the water, clouds drifting across the sky, and windmills with smoothly rotating blades. An optional rain effect can also be activated, further enhancing the scene's dynamic atmosphere. All animations are managed by user keyboard commands, demonstrating a flexible event-handling system.

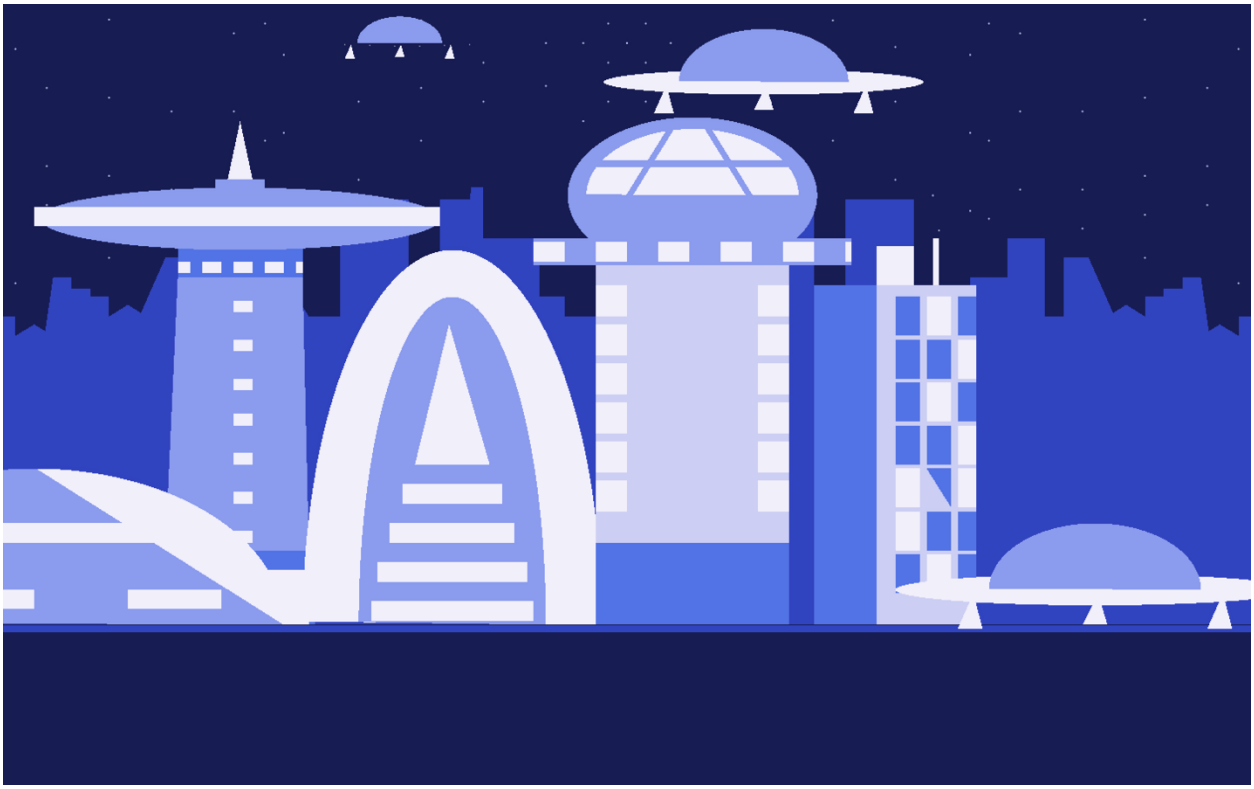
Screenshots for City Scene:

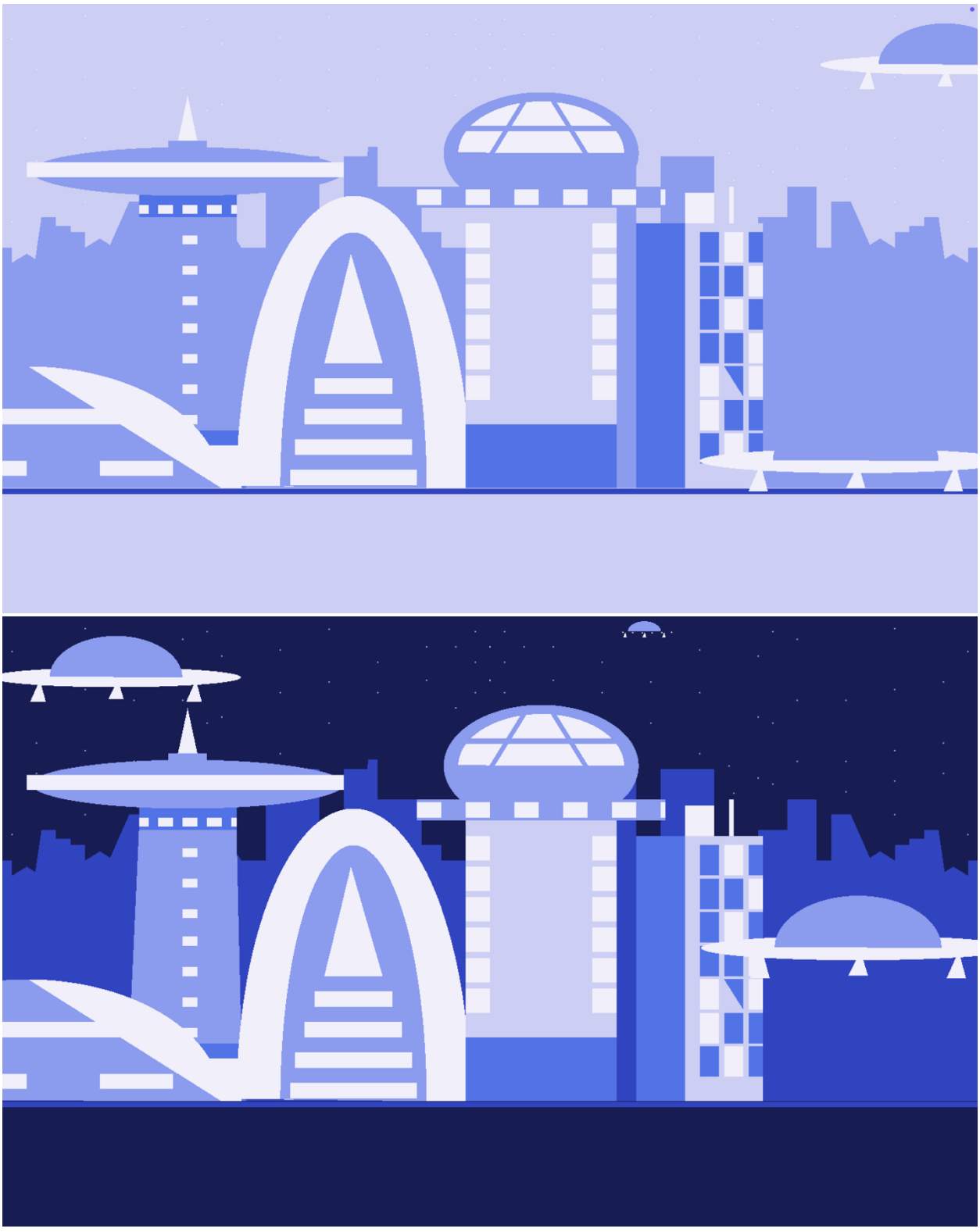




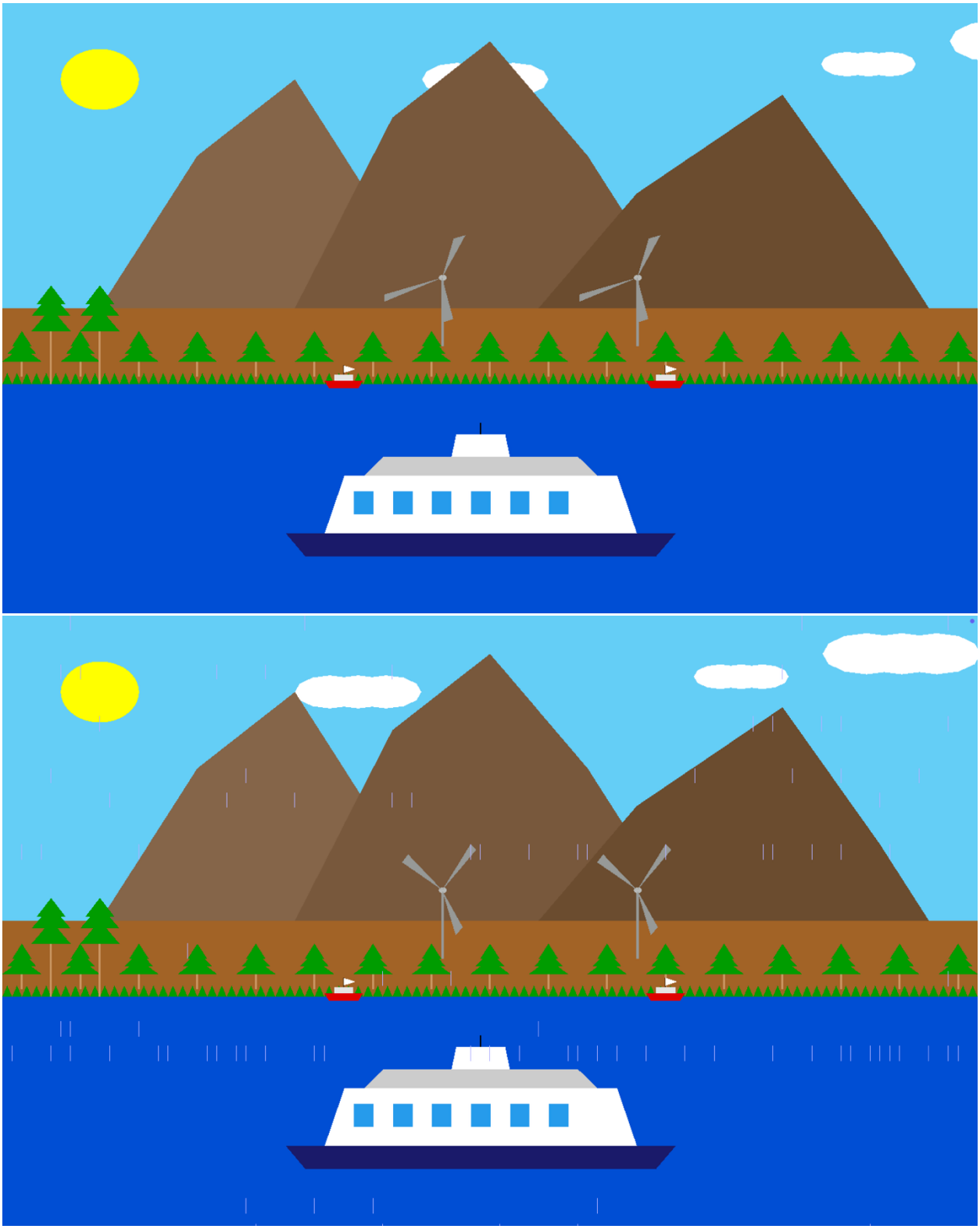
Screenshots For Rocket Station scene:

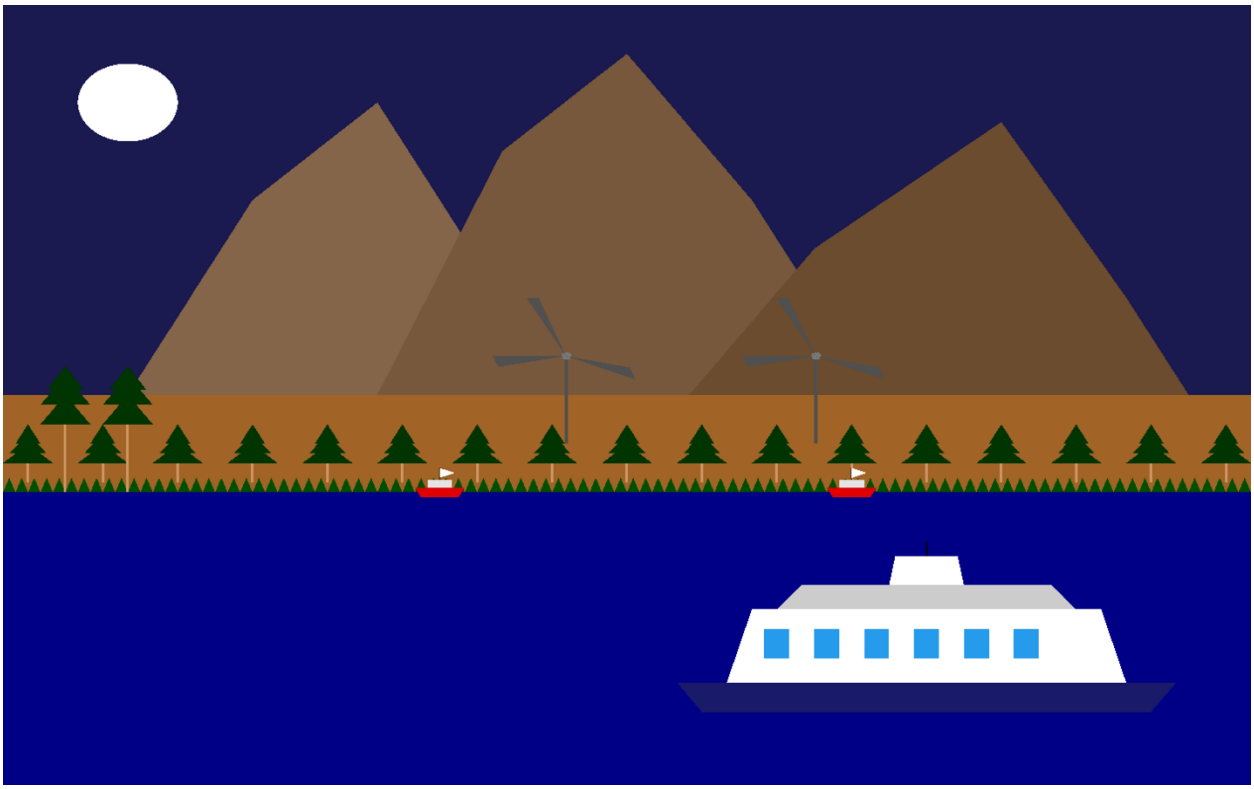
Screenshots For Futuristic City Scene:

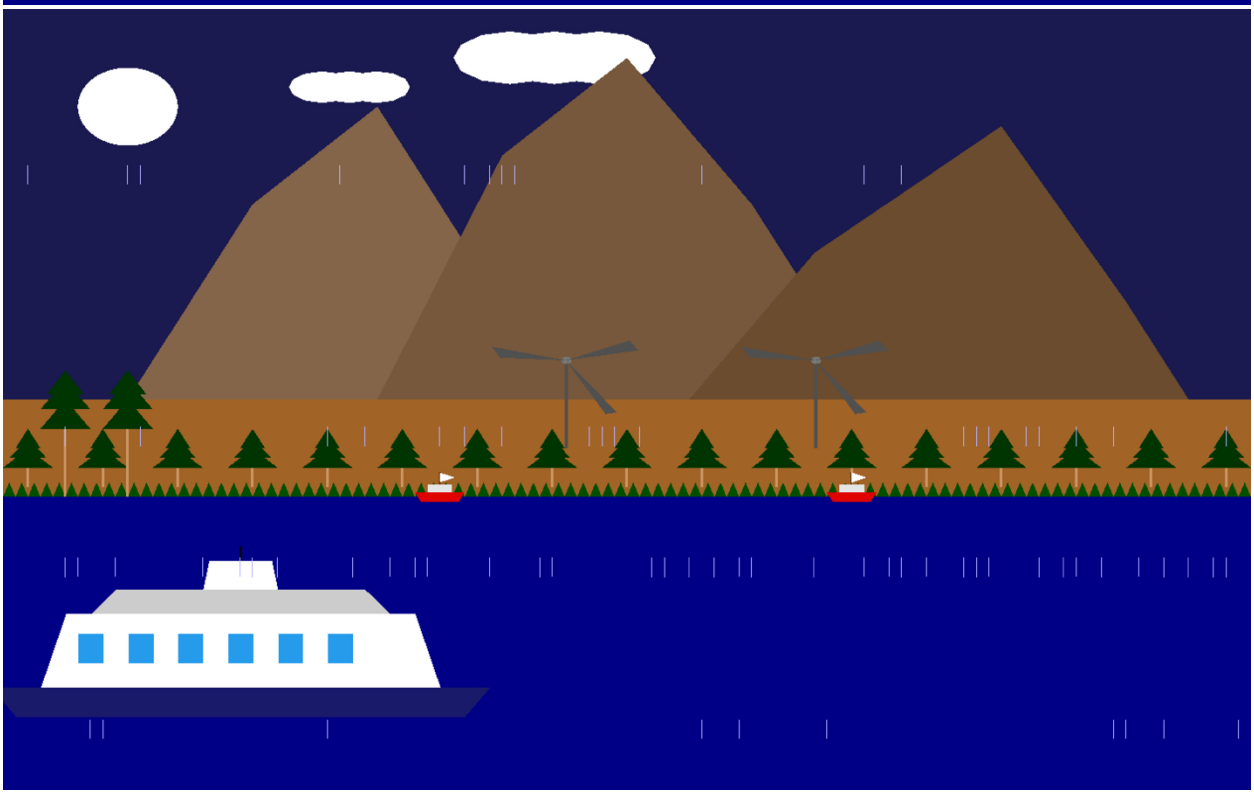




Screenshots For Sea Journey:







Conclusion

This project, "Changing Seasons: Urban Scenery Transformation," successfully demonstrates a comprehensive understanding and application of 2D computer graphics principles. The primary achievement is the creation of a multi-scene, interactive application using C++ and OpenGL. The project showcases procedural asset generation, complex state management for handling different scenes and seasonal themes, and the implementation of a smooth, delta-time-based animation system. The user is given significant control over the experience through an intuitive keyboard and mouse interface, fulfilling all initial project goals.

Despite its successes, the project has several limitations. The entire codebase is contained within a single file, which hinders scalability and maintainability. A more robust, object-oriented approach with classes for scene elements (e.g., Car, Building) would be a significant improvement. Furthermore, the project relies on the deprecated fixed-function pipeline of OpenGL. While suitable for this project's scope, modern graphics applications utilize programmable shaders for greater flexibility and performance. The weather effects, while visually effective, are simple particle systems and could be made more physically realistic.

Future work could focus on addressing these limitations. The first step would be to refactor the code into a class-based, object-oriented structure. Migrating the rendering from the fixed-function pipeline to modern OpenGL with shaders would be a major enhancement, allowing for more advanced visual effects like dynamic lighting and shadows. Other potential extensions include adding collision detection between animated objects, integrating sound effects for a more immersive experience, and allowing for the loading of external assets like textures to create more detailed and realistic scenery.

References

A. Angel, E., & Shreiner, D. (2014). Interactive computer graphics: A top-down approach with WebGL (7th ed.). Pearson.

B. de Vries, J. (n.d.). LearnOpenGL. LearnOpenGL.com. https://learnopengl.com

C. Foley, J. D., van Dam, A., Feiner, S. K., & Hughes, J. F. (1996). Computer graphics: Principles and practice (2nd ed. in C). Addison-Wesley.

D. Khronos OpenGL ARB Working Group. (2017). OpenGL 4.6 API specification. The Khronos Group Inc. https://www.khronos.org/registry/OpenGL/index_gl.php

E. Kilgard, M. J. (n.d.). The OpenGL Utility Toolkit (GLUT) programming interface API version 3. OpenGL.org. https://www.opengl.org/resources/libraries/glut/spec3/spec3.html

F. Reeves, W. T. (1983). Particle systems a technique for modeling a class of fuzzy objects. ACM Transactions on Graphics, 2(2), 91–108. https://doi.org/10.1145/357318.357320

G. Sellers, G., Wright, R. S., Jr., & Haemel, N. (2013). OpenGL superbible: Comprehensive tutorial and reference (7th ed.). Addison-Wesley Professional.

H. Stroustrup, B. (2013). The C++ programming language (4th ed.). Addison-Wesley Professional.