Towards a futuristic emerald isle

DAVID ESTESO CALATRAVA

This report documents the implementation of a computer graphics application that visualizes a futuristic "Emerald Isle," inspired by a futuristic landscape of Dublin. It was to be done using OpenGL with C++, using techniques such as chunk generation, lighting, and texturing, to create an interactive experience of an infinitely large city. It also included the implementation of object animations, camera interaction, and the use of 3D models of various iconic structures. As illustrated within this report, the results ensure a smooth simulation; the robustness tests revealed well-managed resources, with stable performance. This project represents knowledge gained along the entire CSU4405 Computer Graphics course.

1 INTRODUCTION

1.1 Lore

In a distant future, Earth was abandoned after centuries of change and migration to other planets. But the planet was not forgotten. An advanced civilization arrived in Ireland, landing in the ruins of Dublin. There, they found constant mentions of the "Emerald Isle." Although they didn't fully understand its meaning, they took it as a myth of hope and rebuilt Dublin in its honor.

In this new Dublin, emeralds became the heart of their society, used alongside solar energy to power their cities and technology. The "Emerald Isle" was no longer just a legend—it became a shining reality.

You are a time traveler from the 21st century. You brought three most iconic structures to the future: the Dublin Spire, the Wellington Monument obelisk, and the Utah teapot. With your spaceship, you now explore this futuristic Dublin, where everyone welcomes you with excitement and wonder.

1.2 Application

The simulation of the futuristic city is based on a chunk system, where sections of the environment are generated dynamically as the camera moves forward. This creates the illusion of an endless cityscape. Initially, the camera movement is restricted to the X and Y axes, allowing lateral and forward/backward movement using the **W, A, S, D** keys. The camera is controlled with the **cursor**.

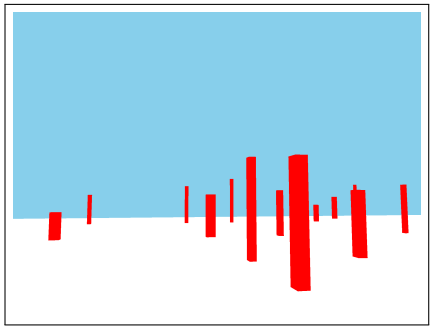
As you explore, you can approach the nearest aircraft. When you get close enough, the aircraft stops, signaling that you can board it by pressing **Enter**. Once onboard, the controls change: you can now use **Ctrl + Up** or **Ctrl + Down** to move vertically, giving you a faster and more comprehensive view of the city.

The city itself is designed to reflect the essence of the Emerald Society. Large blocks of emeralds form the foundational structures, symbolizing the society's energy source. The streets are inspired by the layout of 21st-century Dublin, providing a blend of past and future. To illuminate the city, floating streetlights cast a soft glow over the streets, while the emerald structures remain self-sustaining and do not require additional lighting.

The infinite city generation system is based on a procedural structure that divides space into square "chunks" of the same size, which are generated dynamically around the player depending on their position. Each chunk includes streets, buildings, and some decoration. Principal streets go in a grid pattern with internal variations for each block to create subdivisions in a lot of different sizes such as 1x1, 1x2, 2x1, and 2x2 to give it more of an organic feel to the layout. This generation uses a Cartesian coordinate system to index chunks, loading and unloading depending on the distance of the player from them in order to optimize real-time performance. In non-road areas, crystal clusters materialize into groups of crystalline structures, each one unique in its design. The structure varies in height and orientation, making an urban landscape so diverse. Dynamic elements of the city include aircraft, bots, and trees. Main central landmarks take their inspiration from various elements of Dublin's past-the Spire, Obelisk, and Teapot. Each chunk of road features a futuristic, streetlight-like sphere that always illuminates the streets.

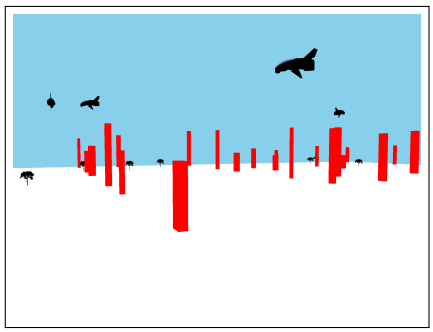
Currently, as the player moves through the city, it may feel like the dynamic generation of chunks is too noticeable up close. Addressing this is as simple as modifying the RENDER_DISTANCE variable of the infiniteCity Struct, along with adjustments like increasing the zFar and resizing the skybox. The current setup prioritizes FPS performance, given the hardware limitations of the machine being used (see Section 3).

2 DEVELOPMENT STEPS

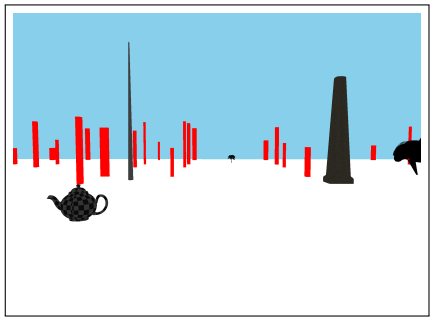


Step 1

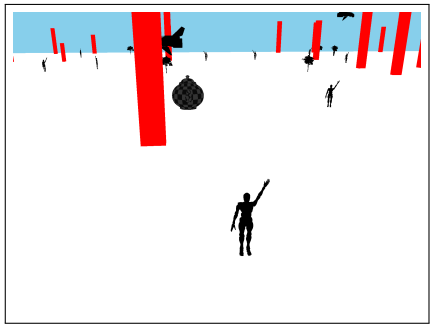
- The first step was to model camera movement and generate the chunk system to simulate the infinity of the environment.
- Then, the process involved learning how to load .obj models and include them in the project.
- Later on, the .obj models were loaded, and elements were contextualized with objects such as the obelisk, teapot, and spire, alongside modeling and applying their respective textures.
- Work in Blender continued with the aim of creating a keyframe animation of a waving bot. In addition, the aircraft model was animated to provide it with a floating effect.
- Spheres were implemented to create light points, simulating streetlights. Different lighting models were also introduced:



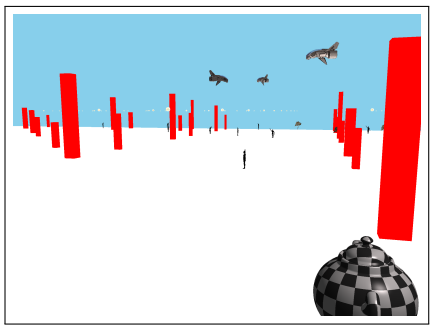
Step 2



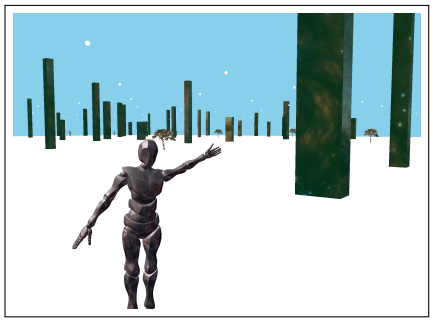
Step 3



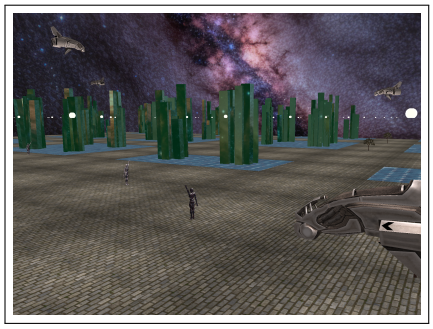
Step 4



Step 5



Step 6



Step 7

Blinn-Phong for efficient calculations on models loading throughout the scene, Phong for reflective models like the spire and teapot, and Lambertian for opaque surfaces like the obelisk.

- Environment mapping was applied to create a crystalline effect on the animated model and emeralds.
- Finally, textures were applied to the ground and the skybox. Mesh simplification techniques were later applied to the aircraft model.

Of course, this process section does not take into account all the iterations, debugging that have been needed, and other actions such as implementing interaction with the aircraft.

3 PERFORMANCE AND ROBUSTNESS

The performance of the application is robust and runs smoothly on my machine, equipped with an AMD Ryzen 7 7840U processor and integrated Radeon 780M graphics, 16GB RAM and 512 SSD. The OS is Windows 11, and the application was compiled using CLion 2024.2.2 with GCC 13.2.0. The application achieves a frame rate of approximately 30-40 FPS.

The FPS remains stable in most cases, with very slight drops when the camera moves far enough to load new chunks. However, the use of pre-initialization mitigates performance issues effectively. Performance tracking through Windows Task Manager shows that the OpenGL application results in serious resource utilization patterns: it goes up in RAM usage from about 9GB to 11GB, an increment of

2GB upon opening; also, the GPU usage leaps tremendously from about 20% to 90%, using about 0.5GB out of the available 2GB of GPU memory. Most interestingly, there is no change in CPU usage before and during the running of this application. These metrics, as visibly captured within the video footage, remained straight throughout the execution period with no noticeable fluctuation in RAM, GPU, or CPU usage, ensuring constant and well-managed resource utilization.

3.1 Tests for Robustness

To ensure robustness, these tests were conducted:

- **Stress Test:** The application was run for several minutes with continuous camera movement to verify memory management. This test was successful, thanks to the implementation of a cleanup mechanism that removes distant terrain chunks outside the rendering distance.
- **Object Scaling Test:** The number of objects rendered in the scene was incrementally increased to evaluate scalability. As expected, FPS decreased with higher polygon counts, but the pre-initialization step ensured acceptable performance. Although advanced optimization techniques like instancing were not implemented, the application satisfies the requirements.

3.2 Code Quality and Design

The application follows good coding practices, focusing on readability, modularity, and maintainability:

- **Inheritance:** The code enhances generalization and scalability by having all objects in the scene inherit from a base class called Entity. This class centralizes key functionalities such as rendering, transformations, and lighting management, making these processes more manageable.
- **Consistent Naming:** Variable names are consistent, improving clarity.
- **Modular Structure:** Each structure has its own header file, making the codebase easier to navigate and maintain.

While there is some code duplication, I believe that the codebase's structure and inheritance patterns have significant potential for reducing redundancy through proper generalization of shared functionality.

4 STRENGTHS, WEAKNESSES, AND POTENTIAL WORK

What I believe is unique in my project is, first of all, the original interpretation of the title and how the whole structure of the project is built according to this interpretation. Then again, there is the randomness part: bots are oriented randomly; the height of the aircraft and trees in dependence of their position and orientation, the emerald blocks structure and the ground simulating the road... almost everything is random, giving it a sense of infinity. Also, small details such as the light rays on the solar panels, the light that is on the spheres to resemble street lamps or the aircraft movement are nice touches in terms of showing the attention of details while adding some sort of motion and dynamism. Another strong point is the interactivity with the central aircraft.

On the other hand, some drawbacks still exist. In the light of what was said above, there is room for improvement in rendering the scene, such as with instancing. Regarding lights, each object is influenced by only one, not all of them. For example, when a bot is standing just at the border of a chunk, it should be influenced by two street lamps, yet it's lit by the lamp in its chunk. And it does not even cast shadows. In my opinion, it's probably the greatest weakness of this project and takes some points from the impression of realism and complexity.

Meaning, thinking of the work to be done for this project in the future, starting from the technical point of view: first, I would start with implementing full shadowing and to handle multiple lightsource, and after that, some advanced features to optimize FPS, such as instancing or levels of detail.

On a more creative side, I think the bots that great you when approaching could be better with a dialogue that triggers related to the lore discussed in Section 1, which I feel could be delved into deeper. It would also be interesting to see the city structure expanded and improved with more defined areas. For example, given that Dublin is a coastal city with a river, there could be more focus on these characteristics to make the environment feel more immersive, and would also give me the opportunity to explore how to work with complex physics in OpenGL, such as water physics

5 ACKNOWLEDGEMENTS

The following 3D models were used in this research and were obtained from online sources, in accordance with the terms of academic use and attribution. I do not claim to have created the models, and with this, I am giving proper credit to the authors. The use of these models is not for commercial purposes. All models were loaded using tinyObjLoader.

- **Obelisk**: This model was obtained from Free3D.
- **Spire**: This model was obtained from Free3D.
- **Teapot**: This model was obtained from Free3D.
- **Aircraft**: This model was obtained from Free3D.
- **Tree**: This model was obtained from Free3D.
- **Solar Panel**: This model was obtained from AmbientCG.
- **Ground**: This model was obtained from Polyhaven.
- **Sphere (Lamps)**: This model was obtained from Free3D.

ChatGPT-4 was helpful during the development of many aspects of this project. It helped create textures and visual effects like the sunbeam effect on solar panels, the texture on the obelisk or the glow effect of the lamps. Also helped resolving dependencies of files, or guiding on different aspects of C++ programming, such as generating random numbers, how header (.h) files work, and with different types of data structures. I also helped to understand .obj and .mtl files.