Visualizing Cities

COSC3000 Computer Graphics Project

Introduction

The aim of this project is to render a procedurally-generated cityscape. Cities are an important part of our modern world, and visualizing cities can provide valuable insights into fields such as transporation modelling, and urban planning. By creating cities and rendering cities digitally, insights can be gained without the production costs of building a city by hand.

Robert A Fraser
S45294051

Technology

This project was developed in OpenGL. I chose to develop the project in OpenGL over a raytracing approach mainly for performance – a cityscape would be significantly more useful if it can be rendered and explored in real-time.

The programming language of choice for this project was Python, using PyOpenGL and glfw for bindings to OpenGL. I chose to use Python due to my past experiences with this, which would (hopefully) allow for easier development. While Python is not quite as efficient as C or C++, most of the heavy-lifting in this project is done in the shaders. If there was additional aspects, such as simulating traffic, Python would not work as well – but this is outside the scope for the project.

For some 2D rendering, I chose to use the fantastic PIL/Pillow library. This library is used to load textures into the shaders, and also for some basic image pre-processing. For some UI controls, the imgui library proved to be invaluable in the early stages of development.

For debugging, I used RenderDoc when times got tough. RenderDoc was extremely useful for stepping through each frame and figuring out what was going wrong – this was invaluable for the beginning of the project, when nothing was displaying at all.

Finally, for modelling and texturing I used Blender (2.83) and Paint.NET. I had plenty of past experience with these programs, which was very useful for quickly getting some basic models working.

Development

Initial Setup

Before jumping into constructing a city, a lot of work has to be undertaken to setup a basic OpenGL environment. To help further my understanding of the topics covered in the course, I chose to re-implement a lot of the code instead of simply copying code used from the practicals. Unfortunately, this took significantly longer than I anticipated.

Debugging was especially a challenge – it's very difficult to figure out what's going on when nothing is showing up at all. At this point, I used RenderDoc to investigate what was going on, and it helped me identify several issues. RenderDoc proved invaluable to getting the basic code working.

For loading and rendering objects, I wrote a ObjLoader class based off the ObjModel code in the labs. This helped me gain familiarity with the obj file format. My approach to this helped separate loading the model and rendering it – this will prove useful when reusing the same building model many times in a large city.

Lighting & Basic Shaders

The next step once basic rendering was done was to setup a (relatively) simple lighting system for the city. My goal with this was to setup a basic daylight cycle, allowing for the sun position and colour to change over time.

Lighting shading is implemented with a simple phong model – ambient colour, diffuse colour, and specular reflections.

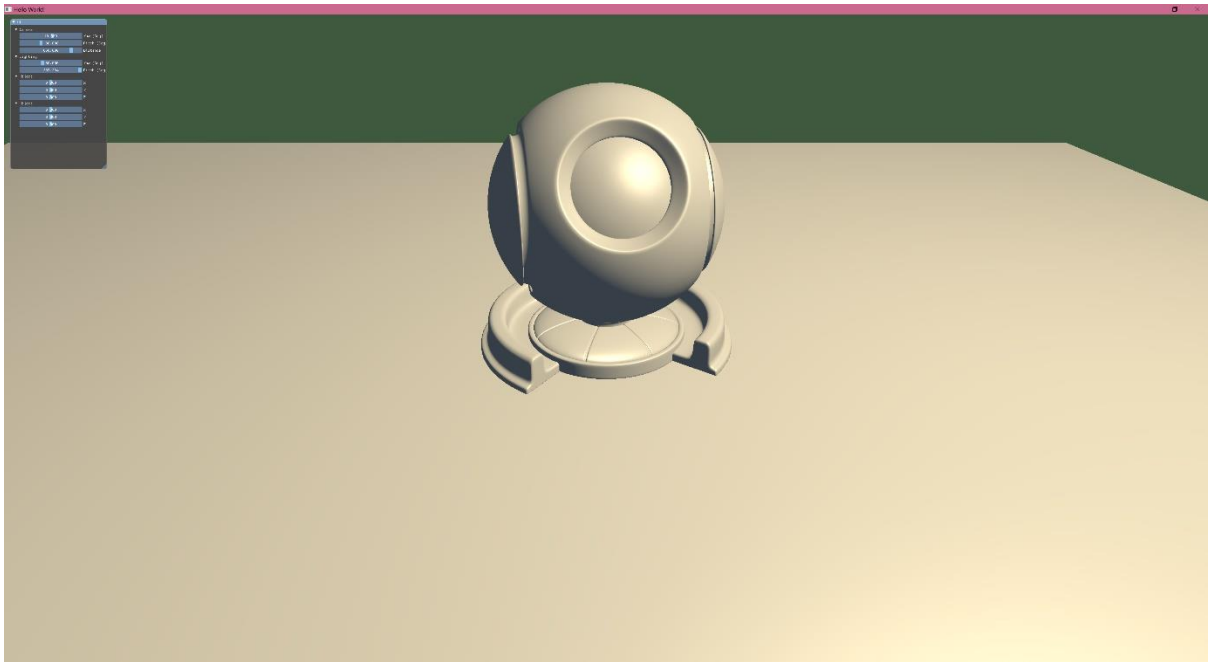


Figure 1: Example implementation of phong lighting model

For a simple daylight cycle, the pitch of the sun has to change over time, along with some different colouring based on the pitch.

I made the following simple gradient to approximate a daylight cycle:

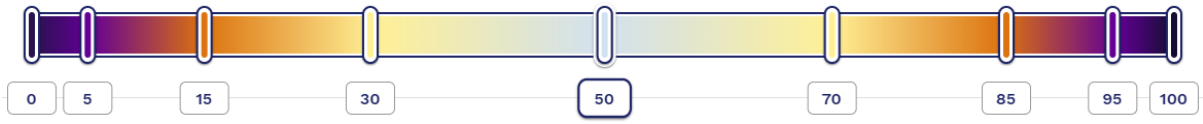
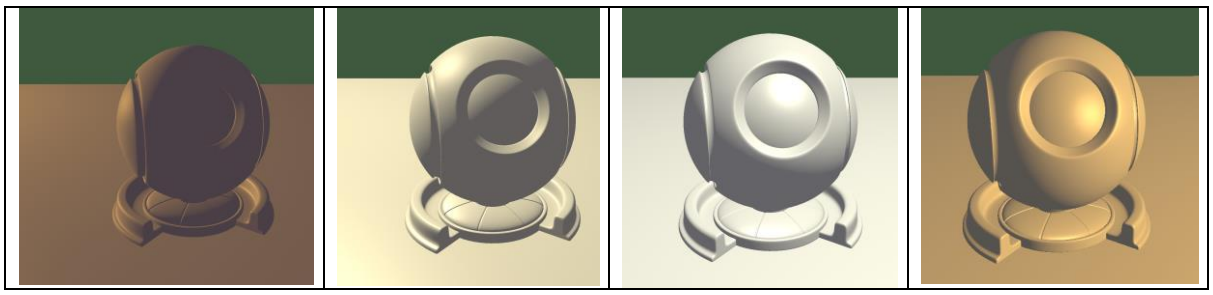
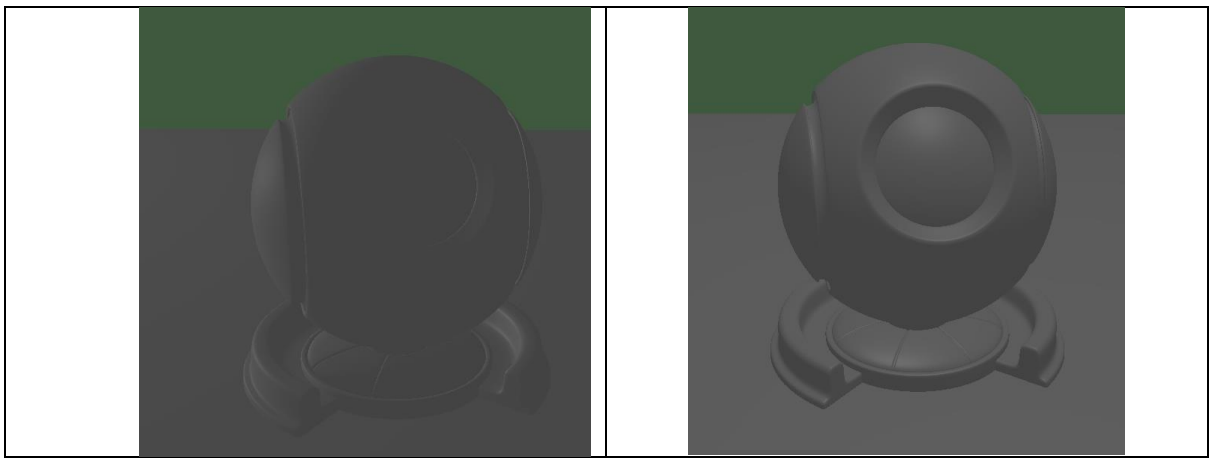


Figure 2: Example of the daylight cycle gradient

Similar gradients were implemented for ambient colour and strength to produce a decent looking daylight cycle:



Relying only on ambient light at night doesn't work too well, so a similar lighting system was introduced with significantly less strength to simulate a moon. This isn't a realistic effect, but it's still interesting enough to look at while providing adequate lighting to the night environment.



Construction Begins

The original plan for this project was for building meshes to be procedurally generated. After some investigation, I eventually decided to create the buildings by hand instead. The main reason for this was difficulty – handling everything such as face normals and UVs was too time-consuming to implement procedurally. I decided that having a small handful of individual building meshes would help populate a basic city and demonstrate the graphical capabilities of the program.

All meshes were created using Blender 2.8. I had past experience with Blender, so it was relatively easy to create the simple buildings. The main area I had trouble with was UV mapping the objects – since I'm using a single texture on all faces, I ran into some minor troubles here.

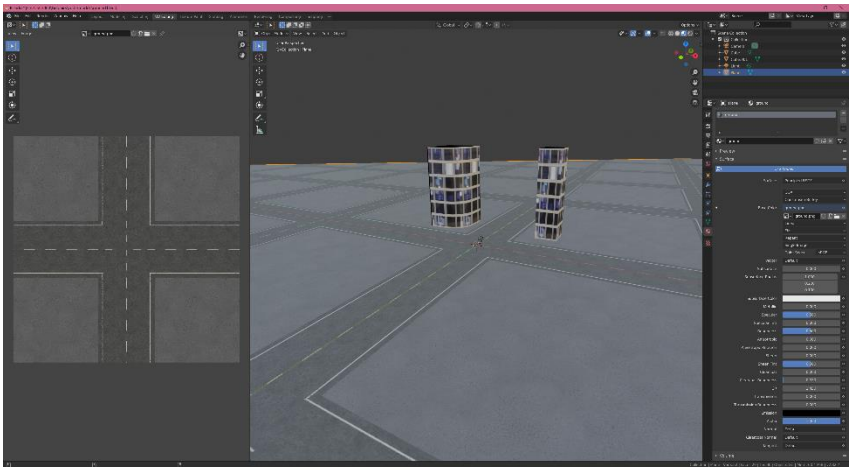


Figure 3: Development of the ground mesh in Blender. UV scales were carefully picked to work well with the building meshes (included for reference)

At this stage, some basic cube mapped reflections were also implemented. While the texture quality is not fantastic, the reflections add a subtle extra layer to the shading model. These cubemaps have been tied into the lighting system, and the texture changes depending on whether it is day or night. This could be expanded with further textures for more times of day.

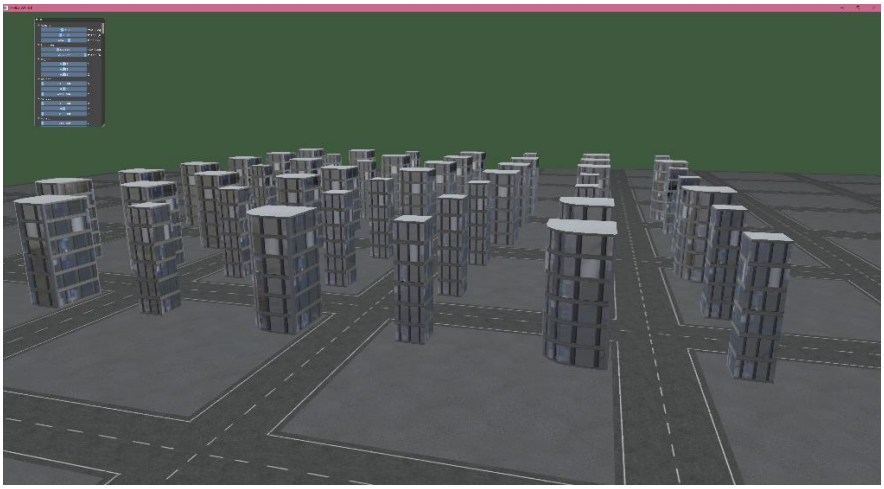


Figure 4: First view of a city with multiple buildings

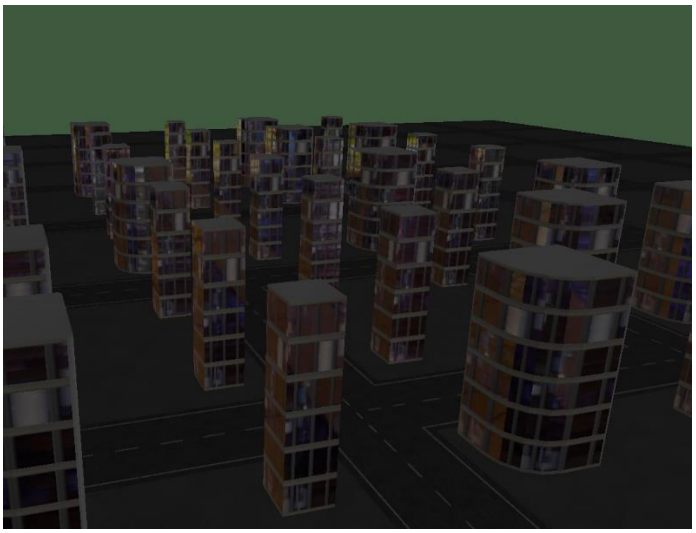


Figure 5: City at night, demonstrating a different cubemap texture

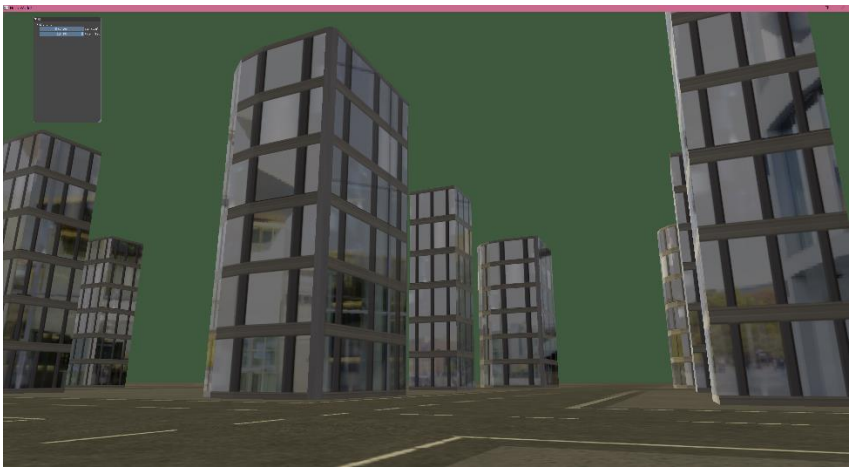


Figure 6: City viewed from street level

The city so far is very rudimentary, but this is a good showcase of the city so far. The buildings use two different shader models: a simple diffuse shader for the rooftop and other edges, and the phong shader model for the windows. Having two separate shaders for the mesh will allow for some more complex effects later as well.

At this stage I also added some terrible streetlights to the ground, to help the city feel more alive. I am unfortunately not an artist, but the basic effect is there. For the streetlights to pop I used a basic unlit emission shader – no actual light is emitted, but the effect is still nice.

Figure 7: City at night with streetlights added.

Performance Improvements

At this point, the city was starting to slow down when a lot of buildings were added – especially when drawn at 4K resolution. I believe that the main cause for this was excessive material switching – each model was being drawn individually, and each model had two different materials. This had a lot of redundant switching. For efficiency, it is better to group all components with a single material and draw them all at once.

This was relatively difficult to implement, but it was also nice to do some well-needed code clean-ups. After this step, cities could have roughly 5x as many buildings before experiencing the same slowdowns.

Interior Mapping

Interior Mapping is a relatively modern shader technique that is used to approximate the interiors of buildings. Originally described in a 2008 paper by Joost van Dongen, the technique can be seen in some modern games such as Simcity 2013 and Marvel's Spider-Man.

By implementing this shader technique into the city, it will allow for the buildings to have approximated interiors without requiring complicated modelling techniques. For implementing this, I did my best to follow the methodology outlined in the original 2008 paper.

Unfortunately, there are a few minor issues with the direction of the mapping – but most of the shader code seems to work fine, and the effect is functional from certain angles.

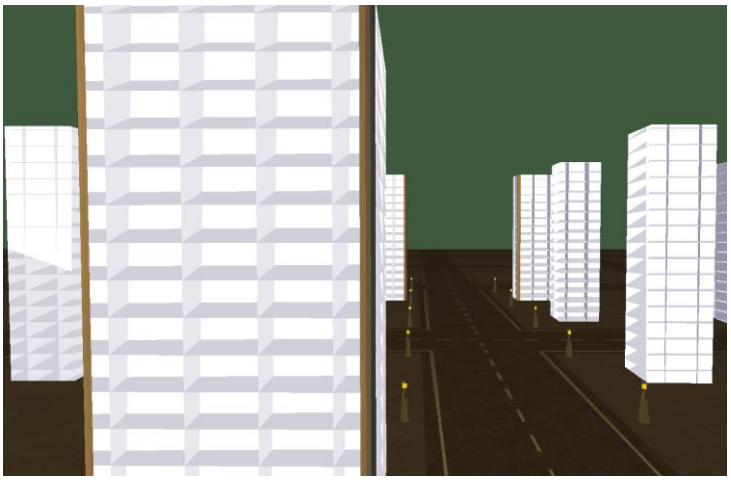


Figure 8: Example of the interior mapping implementation. Notice that the direction is broken in my implementation - this is most evident with the back wall breaking in some of the background buildings. The foreground building is the closest to a 'desired' implementation.

This interior mapping can then be mixed with the previous reflective shader to produce a realistic looking building:

Figure 9: Combined shader. Notice the reflections on the glass while still having the broken interior mapping. Also notice that the interior mapping works well on curved buildings (apart from the aforementioned bugs)

To further improve this effect, textures and varied lighting can be assigned to individual rooms to improve the interior realism. This is trivial to implement, however is not worth the time with the current bugs.

Reflection

Overall the implementation of the project worked decently. There was a significant shift in my project plans – I spent more time working on creating visually interesting environments and buildings instead of investigating procedural generation techniques. While procedural generation would be very useful for this project, I think that focusing on the graphics first was a good decision – these techniques can easily be applied to other, larger projects.

The biggest success of the project was the daylight cycle. Despite using a simplistic phong shading model, the smooth gradients for adjusting the lighting colours have a strong impact on the environment, and the end result feels dynamic and interesting. The system is very easy to customize to tweak the lighting. I'm also happy with how the cubemapped reflections change with the time of day. There's still more that can easily be built into this system – having the cubemaps be built based on the geometry in the program would help to make the environment looking significantly nicer.

The next step for the lighting system would be to add shadows. Adding shadows would provide a much better representation of the current time. This would be relatively easy to implement using a framebuffer. Another improvement to this system would be using a generated skybox – having stars in a night sky would make the cityscape feel much more magical. I briefly experimented with a gradient skybox, however did not get an aesthetically pleasing result so I have not covered it in this report.

I did not spend much time on the actual building meshes, which I think is one of the letdowns of the project. Only three unique buildings were created, so the cities look very repetitive. The main step from here would be to procedurally generate the building meshes – however this is a very complicated process, and a whole project on this ordeal itself could be done. The main problems I ran into when trying to implement procedural meshes was computing face normals and UVs – these are crucial for the shading techniques later, especially interior mapping. With more time and a better understanding of how mesh structures work, I would like to undertake this task in the future.

The interior mapping shader is something I would love to research more. It is unfortunate that I could not get it working properly, however, the key observations are still there and even when broken it adds a lot of realism to the cityscape. The next steps for this project would be fixing this shader and improving it (adding randomness, texture mapping etc.)

Further milestones would be improving the layout of the city. Currently, the city is generated in a grid-like structure. This is good for representing central city areas, but smaller cities have more interesting road networks. Additional work here could also include generating randomised terrains, and fitting a road network and buildings to the heightmap.

I think the main barrier for this project was the choice of programming language. I think that a project of this sort would benefit from stronger type-handling, and Python doesn't provide a good way to manage that. If I was to restart this project, I would reimplement it in C++. I believe that I'd have spent less time building the strong fundamental layer, and more time implementing interesting features in the resulting project.

However, I'm very happy with the resulting base code that I wrote. By doing everything from the ground-up, I learned a lot about graphics programming. There was a lot of difficulty learning everything from compiling shaders to parsing and rendering obj models, but I'm glad I put that work in as I now have a strong understanding of the fundamentals.

RenderDoc was an extremely useful program when setting up the groundwork for the project. Without some of the useful debugging insights, it would have taken significantly longer to fix bugs that occurred, especially when writing the code to render meshes.

Other tools that I used were very helpful in the development of this project. Blender was a fantastic program to use for creating the models – especially for adjusting the UV maps. Paint.NET was also a useful program to create the textures. The imgui library was also very useful for experimenting and debugging in the earliest stages of the project.

Overall, I'm happy with how the fundamental code turned out and I've learned a lot from this project. While I didn't get too deep into the procedural generation part of the project, I built some solid groundwork and learned a lot of graphics programming techniques. While the finished project is quite different from the original plan of a large sprawling city, I believe that the individual buildings are graphically interesting and this project is a better representation of my graphics programming abilities.

References

Resulting code for this project can be found at
https://github.com/rafraser/COSC3000/tree/master/Graphics/code

Resources

Cubemap textures were obtained from HDRI Haven (https://hdrihaven.com/)

Road textures were obtained from CC0 textures (https://cc0textures.com/)

Building and lantern textures were self-created.

Papers

Kelly, G., & McCabe, H. (2006). *A survey of procedural techniques for city generation.* Retrieved from http://www.citygen.net/files/Procedural_City_Generation_Survey.pdf

van Dongen, J. (2008). *Interior Mapping.* Retrieved from http://www.proun-game.com/Oogst3D/CODING/InteriorMapping/InteriorMapping.pdf