A5 Project Proposal
Surveying Neo Tokyo
Zhi Kai Lu
20666916
zklu

Final Project:

Purpose :

To create an interactive game where the user controls a helicopter surveying the chaotic dystopian city.

Statement :

This project aims to tell the story of a Neo-Tokyo from sci-fi film "Blade Runner", where the player explores the city from a newscopter equipped with searchlights and audio enhancing technology. Audio and interesting buildings are key components of this project, so a large chunk of time will be dedicated to sampling and/or creating these assets. Further, this game serves to demonstrate core computer graphics concepts covered in the course, alongside some more complex algorithms from academic papers.

The course notes briefly covered 'lighter' lighting techniques and possible optimizations; I'm interested in further exploring lighting algorithms to implement a robust technique that's suitable for quickly rendering with multiple lights for a moving camera, as is often the case in videogames. I also want to test the limits of the graphics lab's computers: at what point are there too many meshes to render on the fly? What optimizations can I do then? How much do these optimizations improve the speed of rendering? Similarly, I love RPGs with story-telling through game mechanics and details, and creating a complex environment with audio would be my homage to such games. To me, the most challenging part of the project will be juggling the big list of objectives, prioritizing progress on certain features so that other features can be sensibly built on top of other features.

Technical Outline :

Buildings will be created using the L-system concept from the course notes. Neo-Tokyo has a lot of skyscrapers so adequate width, height, and various roofs will be defined for the grammar. Texture map will be added to them, without bump mapping for difficulty and efficiency reasons - and also because the scene will take place in night with many buildings obscuring other ones.

Besides buildings, the models (people, cars, traffic lights, billboards, streetlights, etc.) will be hand created and defined as lua objects with a hierarchy for efficient transformation, varied characteristics and multiple instances. Some models (cars, lights) will change state (move, broadcast different sounds, show different colours) over time this AI will be handled with polymorphism in openGL's appLogic. For example, people will enter buildings and have different audio when they are in the building, cars will honk or move to an alley when the player shines light on them for too long.

Beyond simply meshes, each object in my scene goes through its own set of logic per render, producing some animation. Hopefully this doesn't result in any latency as the player navigates the scene. While I'm currently unsure of how many models the graphics computers can handle, I am prepared for reducing lag with a few strategies: clipping and hidden surface removal, idle state for when object is sufficiently far away, and a simplified representation of an object when it is somewhat far away. The specific distances will have to be tested with trial and error to avoid objects jumping out.

There will be many billboards, storefront signs, and car headlights as sources of light in the scene. I'd need a robust algorithm to compute lighting in this scenario. I looked at deferred shading based on the professor's recommendations, but there were many papers that simply namedrop deferred shading. I plan on following Policarpo's paper on the subject matter, and expanding out based on the difficulty and time constraints.

Infrared mode will be achieved by disabling hidden surface removal, rendering non-person objects by setting alpha values according to distances and the amount of objects the eye has already seen through. Logically, this sounds like a form of non-recursive ray tracing, which would be too computationally demanding alongside the proposed shading method and the sheer number of polygons. As such, we can approximate by using the information from the first pass of deferred shading, treating the camera as a light source that contributes alpha value.

The helicopter will be operated by the standard WASD input for movement, while the mouse will control the searchlights and audio enhancer from the helicopter. The helicopter will be responsive while reasonably simulating the swaying and inertia that a helicopter should have. I plan to do this through generating small amounts of random noise for its velocity, and tracking how long a WASD key was pressed before being released.

Because of the popularity of Blade Runner, there exists lots of fanart and audio samples that I can legally use. I also plan to record some of my own samples with a friend's USB mic, and edit the audio through Ableton. Adding these audio files to the program requires an audio library like FMOD or DirectSound. I'll play around with both to see which one uses less memory.

From the sound rendering paper, I will use the basic sound attenuation presented in the preface of chapter 5: 'Effects of distance and direction'. It assumes no object is between the transmitter and receiver. If time permits, I will try to implement a more complex algorithm is calculated during the steps of tracing rays (I'm not using ray tracer to shade but perhaps there's some workarounds to get the information needed for calculations). I will also implement doppler's shifts by keeping track of the player's velocity and altering the frequency of each sound file after attenuation, hopefully there's a library to do that easily.

Organization :

All files are located in the Project directory after unzipping the .zip file. It also contains all the source code for my project (barring the shaders).

- **Assets** folder contains resources and lua file for defining the base of my scene.
  - **sounds** folder contains all the audio used for the project. I sampled two sound files from Blade Runner, and the rest was voice acted my friends, Max and Helena, and me.
  - **images** folder contains the images for texture mapping. My Project.cpp file has a more detailed breakdown comment of how each texture is used. Most of these textures were from textures.com or construction blogs.
  - **Shaders** files are also present in this folder. I used the A3 shaders as a starting point, mainly modified fragmentShader.fs
- **irrKlang** folder is the audio library imported. I did modify one of its header files to pass default argument TRUE for enabled sound effects.

Compilation :

1. run "premake4 gmake" from current folder (where this README is)
2. run "make"
3. run "./Project Assets/scene.lua"

Disclaimer: on most of lab computers Audio does not work because audio drivers not installed.

Manual :

The Project always takes in a lua file "base" scene that the buildings are built on top of. My base scene contains a very largely scaled cube representing the ground (xz-plane at y=0). It also contains texture mapped "roads" that are built from a for loop. Any file following the lua format defined in A3 will technically work.

When the program begins running, the user controls the helicopter with the following:

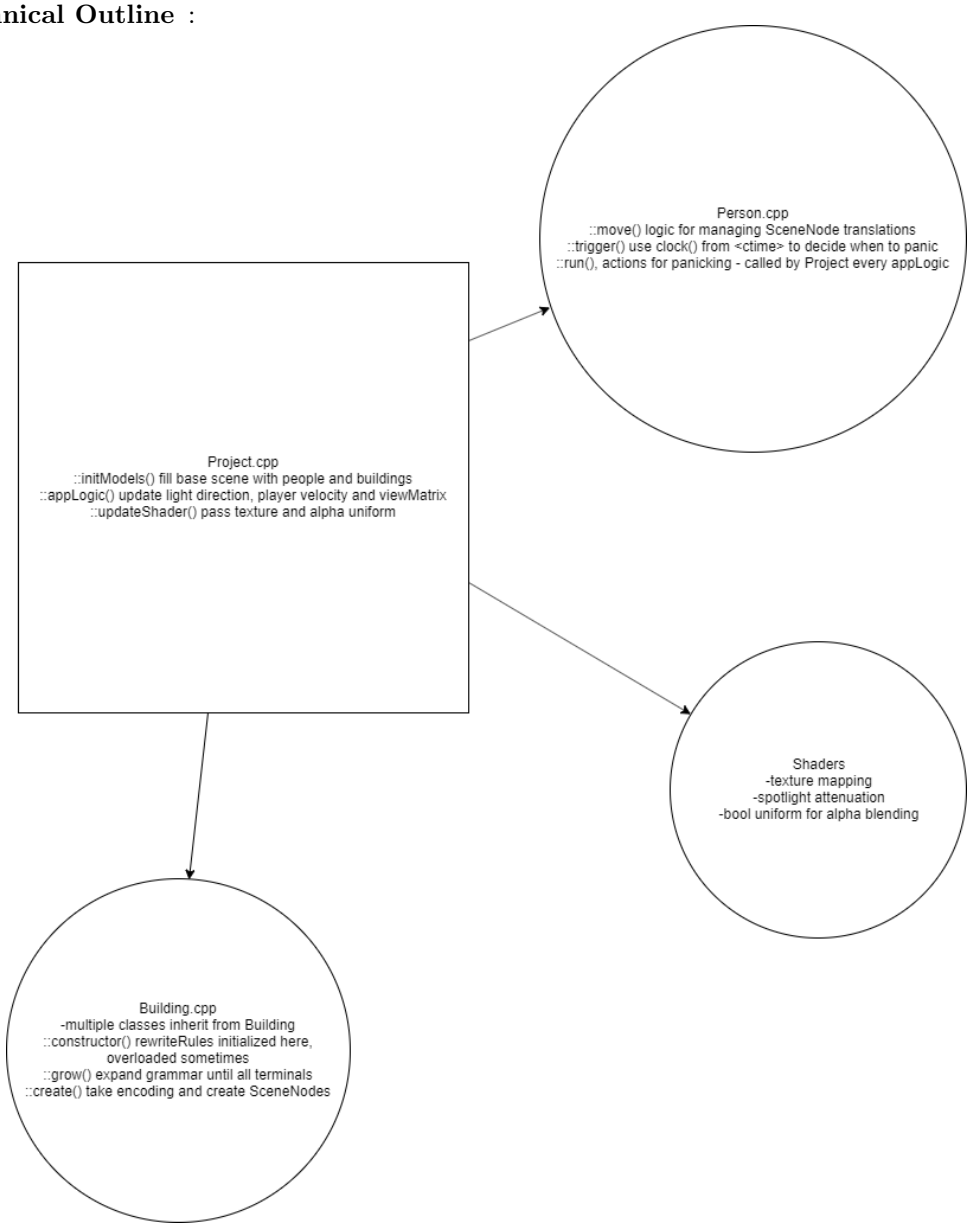
- Mouse movement within the window of the program controls the spotlight.
- WASD allows you to forward, backward, left, and right with respect to your current orientation.
- E and Q allows you to descend and ascend with respect to your current orientation.
- If you hold the shift key, WASD allows you to look up, down, left, and right.

- CTRL key being held down triggers infrared mode.

The program will only output to standard output if there was something wrong with loading the texture or audio files. Check your system's audio drives if an error comes up.

Side note: this project is best experienced with earbuds in. The irrKlang library supports LR audio-split and makes exploring the city more interesting.

Technical Outline :



Implementation :

Building object follows a factory OOP. Buildings all need to have their initial grammar grown with building:grow() before being converted to a tree of SceneNodes. If there are K rewriteRules, and N levels of growing (levels is passed in as a parameter), then the amoritized complexity is $O(N\hat{2})$. In each iteration, we step through each char in our current encoding (with some separators that we filter out with string manipulation), and see if it matches a rewriteRule, and then appends the resulting

symbol to a new encoding String we're constructing. Luckily, our rewriteRule is stored in a hashmap so it doesn't contribute to big O. We then proceed to generate the building in $O(N\hat{3})$, iterating through the height, the width, and each side, as one side of the building has doors attached to it.

Person object has a pairing logic that is similar to a doubly linked list. Normally, if you hover over a Person for too long, they scatter, repeating their own audio, disregarding anyone else's movement, they just want to go the opposite direction of the center of the light. For a Person that has a non-NULL Pair pointer, they know whether they are the leader (by checking the Pair's leader pointer value with itself), or the follower. The follower doesn't worry about anything and the leader will call all the actions the follower should do when the leader is doing them. When I first implemented this, I pushed the two Persons created into my Person vector (which will be used to call every Person's action in appLogic), before setting the Pair pointers and assigning it to both Persons, because I know vector push_back uses copy constructor to create new instances of Person. However, I forgot about vector constantly moving itself in memory location space, so I had to refactor my code to use Person vector to get around this issue.

To pick up on audio from where your spotlight hits, I keep track of the spotlight's direction (dependent on your mouse), and the position (which is always 0, -1, 0) since we do our shader lighting calculations in world space. When the player moves, it's really the models around the player that is moving - so to intersect with the ground plane, which I know to be 0, 0, 0 in modelSpace, I applied the inverse of the viewMatrix to the light's position (A) and vector direction (B). Then we can do line plane intersection to solve for A+tB = p like in Assignment 4.

I preprocess the textures needed by passing the images into openGL's texture2D array and binding it to a specific location. My 'textures' vector is really just storing these locations to be passed into the shader alongside the Mesh I'm rendering. If my Project involved spawning more People in/out as time went by, I could initialize a huge batch of ISound audio files and simply hand them out as needed, like Shader texturing.

I put lots of consideration into modulating my code, making sure proper logic is abstracted from my functions. My project's code is scalable and reusable with two caveats: the programmer knows what the Building's grammar represents, and the programmer is aware of the places where roads are (off limits for building construction). If the base scene was also defined in C++ instead of lua, the roads could be defined first and made off limits because the building constructor functions would 'see' them.

Besides the two hard objectives I did not get to implement, there are a few improvements I wish I could've addressed. I traverse from rootNode passed in from the scene.lua file to render my whole scene. Ideally, preprocessing would be done to flatten this tree, or removed entirely in favour of drawing buildings and people separately, and keeping track of them in their respective vectors. That being said, suppose we have P geometryNodes for all our Person objects, and B geometryNodes for our Building objects, we'd still need to process each geometryNode (or Mesh), so theoretically this change wouldn't reduce time complexity. It would, however, save the space of storing a modelView matrix that gets pushed as we traverse the tree. It would also make it easier to distinguish between Building and Person for when I implemented infrared vision, passing certain temporary colour values to the shader, alongside potentially adding array instancing - rendering multiples of the same set of meshes in various locations to generate the same amount of People and Buildings for less.

The initialization of the project involves a lot of procedural generation based on rand() and tweaking the chances something may take place (ex: a building along the street that's an apartment instead of a store, a person being assigned an audio file). Initially, I manually created a lot more buildings, but had to scale this down a lot in order to get above 10 frames per second. A better alternative to fine tuning the parameters I did to reduce processing strain would be to find the maximum meshes the computers can take, and disperse models throughout the scene based remaining meshes.

I've extended scene_lua.cpp to allow a lua file to define texture index NOT the texture path itself(since images are loaded to become openGL textures within Project.cpp). I feel that the extension,

alongside lua could've been discarded, and instead have the whole base scene be defined in C++ initialization, but it would've resulted in even more one-off functions being added into my Project file.

Spotlight is an extra feature implemented. Contrary to my belief before starting this project, it is not something that is supported in openGL simply by some function call before the shader is invoked. You can see the logic for spotlight lighting in VertexShader.vs. By passing in the light's direction, we can compare it with the dot product (cos angle) of the direction of light source to point on mesh to determine if light contributes to the mesh's colour.

Acknowledgements :
Special thanks to Max Saar and Helena Ip for being the many voices for this project.

References :

Agrawal, Kushal. "Plant-Drawing" (2017), Github repository, https://github.com/legobridge/plant-drawing.

de Vries, Joey. "Textures." LearnOpenGL, https://learnopengl.com/Getting-started/Textures.

Rost, Randi. "Spotlight Shaders." Khronos Forums, 10 Dec. 2004, https://community.khronos.org/t/spotlight-shaders/53154.

Müller, Pascal, et al. "Procedural modeling of buildings." Acm Transactions On Graphics (Tog). Vol. 25. No. 3. ACM, 2006.

Takala, Tapio, and James Hahn. "Sound rendering." ACM SIGGRAPH Computer Graphics 26.2 (1992): 211-220.

Objectives:

Full UserID: zklu **Student ID:20666916**

1: Buildings generated from L-Systems look cohesive and different from each other.

2: Scene layout is defined with cars, people, billboards, traffic lights, and parameteres to allow L-building generation.

3: Textures are appropriately mapped onto buildings and billboards.

4: Objects' AI interact with other objects and react to searchlights shining on them.

5: Use deferred shading to illuminate objects' surfaces with numerous (10-20) light sources, and a moving light source (helicopter's searchlight).

6: User can enter an Infrared mode to see people through structures depending on how many structures between camera and person.

7: Reflection implemented in appropriate materials.

8: User interface for controlling Helicopter: steering with keyboard (includes rotation around arbitrary axis), searchlight with mouse.

9: Appropriate sound file is triggered when sufficiently close to respective object.

10: Sound volume calculations: doppler shift and sound attenuation.