A5 Project Proposal
Title: Forest Scavenger Hunt
Name: Orson Baines
Student ID: 20652798
User ID: obaines

Final Project:

Purpose :

To present some advanced graphics effects in a fun, interactive scavenger hunt game.

Statement :

My project will consist of a scavenger hunt game where the user will be given a list of clues and need to locate the required items. The user interface will be modelled after open-world adventure games like Elden Ring and Skyrim, with movements and actions controlled by the keyboard and the view controlled by the mouse. There will be a clock to show in-game time, which will control the direction of the sun and the intensity of ambient and directional light. There will be a slider or key-bindings to control the ratio of real-world to in-game time, which by default will be about 1 min : 1 hour (like Terraria). All objects will cast shadows.

The terrain will consist of a heightfield for the ground along with trees, rocks and puddles as static terrain. There will also be at least one campfire or chimney producing smoke.

As additional controls the user will be able to enter binocular mode where the field of view is reduced to a very tiny angle allowing the user to zoom in on part of the screen, and a flashlight mode that adds a spotlight to the scene pointing in front of them.

Holding down the left mouse button will add crosshairs and then when the user releases the mouse button is will pick the item that they are focusing on. If the item is on the scavenger hunt list then they will be notified and the item will be crossed off.

This format of game will be both challenging and interesting, because it will give me lots of room for creativity in modelling the scene, and designing the list of clue and objects to spot. It gives me the chance to use lots of graphics techniques including shadow mapping, texture mapping, bump mapping, transparency, Perlin noise, and others.

For libraries I will use SDL2 for window and OpenGL context creation, as will as for sound. I will also use GLM for matrix math, and GLEW for loading the OpenGL functions and extensions.

Technical Outline :

Core Features (Objectives)

- **Terrain Heightfield:** I will use Perlin noise to generate a greyscale bitmap. I will refer to Perlin's original paper [1] and his improved algorithm [2] for the theory of Perlin Noise. I will then refer to [3] for an example of how to use the noise to generate a terrain. This article is platform generic, so it doesn't actually go into the details of rendering the terrain, but I will plan to store the generated terrain in a greyscale texture and read from the texture to calculate the height at each vertex in the mesh. I plan to render the terrain in two passes, one to draw all the ground, and a second pass to draw water on top of ground that is deeper than a certain threshold.
- **Bump Mapping:** I will use bump mapping to add a realistic surface to the tree trunks. I plan to create in software (GIMP, etc.) the heightfield for the surface of the trunk (with black lines in the grooves of the bark) and then calculate the texture derivatives to modify the normals. This will also rely on a cylindrical projection to map the vertices into UV coordinates. I'm referring to a tutorial on normal mapping [4], which is slightly different since it uses a normal map instead of a heightfield, but lots of the same ideas are still applicable.
- **Shadow Map:** I will need a shadow map for each light source, one for the sun and one for the flashlight. Since these lights are constantly changing, they will need to be updated each frame. This will involve creating a grey scale texture for each shadow map, and assigning the textures to a framebuffer, then binding the framebuffer an rendering the scene from each light's point of

view with that light's framebuffer bound. To prevent aliasing effects I will need to use a bias to make the shadows slightly deeper than how their recorded value, and also use front face culling. Shadow maps are explained in the learnopengl tutorials [5].

- **Transparency:** The smoke particles and water will be partially transparent. I will achieve that with alpha blending, which I will need to enable in OpenGL.
- **Particle System:** I will use a particle system to simulate smoke. [6] gives some examples on how to do a particle system in OpenGL, and [7] shows demo code (in processing, not OpenGL) to use a particle system to simulate smoke.
- **UI:** For the user interface I will need to draw 2D geometry on top of the 3d scene to show a clock indicator, progress indicator, and FPS counter. [8] shows how to render 2D text in OpenGL using a texture atlas.
- **Binocular Mode:** After looking at some examples of scoping in first person shooters, and looking at sample images on the web, it looks like a good strategy to achieve a reasonable looking binocular view is to decrease the field of view and render within two overlapping circles, and have the rest of the screen black. I will use alpha blending to make the edges between the binoculars and the occluded potion of the screen soft, so I will render the scene first and render the binocular overlay on top.

  I will add Depth of Field effects to the binoculars so that objects that are far away from the focus point are blurred. I can accomplish this by using a post-processing filter to achieve this effect. For example, the Godot game engine has a post-processing filter to achieve this effect [9], so I can refer to their code [10]. I also found a step-by-step guide to implementing the depth-of-field effect in DirectX 9 [11] that will help me understand how this filter works.
- **Stencil Buffer Reflections:** [12] gives a succinct step-by-step outline and how to do stencil buffer reflections, although it is written using the old OpenGL immediate mode rendering. It also has the reflected scene drawn first, and then the normal scene, but I think it would more efficient to do the passes in the reverse order. When drawing the reflected scene I will need to write to `gl_ClipDistance` in the vertex shader to clip the vertices by the mirror plane [13].
- **Skybox:** The Cubemaps OpenGL tutorial [14] explains how to use cubemaps to render a skybox. I will use the optimization technique described in the tutorial to render the cube at depth one after rendering the objects in the scene so that I don't waste resources writing pixels that will be overwritten later.

Bonus Features (To be implemented if I have time)

- **Texture mapping:** For the objects I am modeling I would like to use at least one of each of the 3 generic projector functions we used in class (cylindrical, spherical, planar). Depending on the model, I will probably do some of the mappings in blender and some of the mappings in code. I will compress my RGB/RGBA textures using the DXT formats (DXT1, DXT3, and/or DXT5) which are supported by an ubiquitous OpenGL extension `GL_EXT_texture_compression_s3tc` and write code to load them from the .dds file into OpenGL.
- **Keyframe Animation** I would like to extend my puppet from A3 with keyframe animations to animate the hips, knees, elbows and shoulders of the character as the move around the scene. If I do this, I will add a 3rd person viewpoint options to be able clearly to see the animation.
- **Static Collision Detection** I would like to add collision detection, where each object in the scene has a bounding volume and each time the avatar moves it ensures that the avatar's bounding volume does not overlap with any of the objects bounding volume.

Bibliography :

References

[1] K. Perlin, "An image synthesizer," *SIGGRAPH Comput. Graph.*, vol. 19, no. 3, pp. 287–296, Jul. 1985, ISSN: 0097-8930. DOI: 10.1145/325165.325247. [Online]. Available: https://doi.org/10.1145/325165.325247.

[2] ——, "Improving noise," in *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, ser. SIGGRAPH '02, San Antonio, Texas: Association for Computing Machinery, 2002, pp. 681–682, ISBN: 1581135211. DOI: 10.1145/566570.566636. [Online]. Available: https://doi.org/10.1145/566570.566636.

[3] A. Patel. "Making maps with noise functions." (2015), [Online]. Available: https://www.redblobgames.com/maps/terrain-from-noise/. [Accessed: Jun. 24, 2022].

[4] J. D. Vries. "Normal mapping." (), [Online]. Available: https://learnopengl.com/Advanced-Lighting/Normal-Mapping. [Accessed: Jun. 23, 2022].

[5] ——, "Shadow mapping." (), [Online]. Available: https://learnopengl.com/Advanced-Lighting/Shadows/Shadow-Mapping. [Accessed: Jun. 23, 2022].

[6] "Particles / instancing." (), [Online]. Available: http://www.opengl-tutorial.org/intermediate-tutorials/billboards-particles/particles-instancing/. [Accessed: Jun. 23, 2022].

[7] D. Shiffman. "Smoke particle system." (), [Online]. Available: https://processing.org/examples/smokeparticlesystem.html. [Accessed: Jun. 23, 2022].

[8] "2d text." (), [Online]. Available: http://www.opengl-tutorial.org/intermediate-tutorials/tutorial-11-2d-text/. [Accessed: Jun. 24, 2022].

[9] "Environment and post-processing." (), [Online]. Available: https://docs.godotengine.org/en/stable/tutorials/3d/environment_and_post_processing.html. [Accessed: Jun. 26, 2022].

[10] "Bokeh_dof.cpp." (), [Online]. Available: https://github.com/godotengine/godot/blob/master/servers/rendering/renderer_rd/effects/bokeh_dof.cpp. [Accessed: Jun. 26, 2022].

[11] G. Riguer, N. Tatarchuk, and J. Isidoro, "Real-time depth of field simulation," *ShaderX2: Shader Programming Tips and Tricks with DirectX*, vol. 9, pp. 529–556, 2004.

[12] "Planar reflections using the stencil buffer." (), [Online]. Available: http://www.bluevoid.com/opengl/sig00/advanced00/notes/node165.html. [Accessed: Jun. 26, 2022].

[13] "Vertex post-processing." (), [Online]. Available: https://www.khronos.org/opengl/wiki/Vertex_Post-Processing#User-defined_clipping. [Accessed: Jun. 26, 2022].

[14] J. D. Vries. "Cubemaps." (), [Online]. Available: https://learnopengl.com/Advanced-OpenGL/Cubemaps. [Accessed: Jun. 26, 2022].

Objectives:

Full UserID:_____ **Student ID:**_____

___ 1: Add a UI with an analogue clock to display and keyboard shortcuts to control in game time.

___ 2: Use Perlin noise to generate terrain heightfield.

___ 3: Apply Bump Mapping using heightfield.

___ 4: Apply Shadow mapping with 2 light sources.

___ 5: Do reflection using stencil buffer.

___ 6: Add transparency with alpha blending.

___ 7: Simulate smoke with a particle system.

___ 8: Use skyboxes (one for day and one for night).

___ 9: Use depth of field in a binocular view.

___ 10: Synchronize sound with game events (collisions, footsteps, picking)