COMP3011 Computer Graphics

Spring 2025

Assessment 2

Report Sheet

This report will help you prepare for your demo. **Submit this report to Moodle**.

**Student Name: Harry Bridgen**

**Student ID: 20421316**

**Username: psyhb7**

**I agree for my code & report to be published, with my name, to future students as an example (yes/no): No**

|  |  |  |
| --- | --- | --- |
| **Introduction** | | |
| *Please explain why you implemented this scene* | *Describe your inspirations* | *Provide a general description of the scene.* |
| I wanted to make an orbiting camera like Oldschool Runescape and terrain style like Valheim to see if they would pair well. | Oldschool Runescape (Camera)  Valheim (Terrain) | Procedurally generated hills, valleys and water. Contains trees that sway in the wind, a day night cycle, lighting and shadows. |
| **TR 2 – 3D Modelling** | | |
| Object 1 - procedurally generated | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of object* |
|  | **WATER**  water.cpp  line 12, init funct  line 36, render funct  **TERRAIN**  terrain.cpp  line 12, init funct  line 52, render funct  line 118, GetTileHeight funct  line 141, build mesh funct  noise.h  Noise for terrain generation | **WATER**  Procedurally generated water plane on y = 0 with vertex displacement for wave motion and a fragment shader that adds animated noise based shimmer.  **TERRAIN**  Procedurally generated terrain using a single quad base. Terrain is formed making many tiles to form a mesh then offsetting their y positions using fractal noise to create rolling hills and valleys.  Additionally, terrain under and around the water level is textured sand, while hard slopes are textured stone. |
| Object 2 - OBJ parser | | |
| *Please give a screenshot* | *Provide the URL for the OBJ files you submitted* | *Description of object.* |
|  | **HUMAN**  <https://free3d.com/3d-model/male-base-mesh-6682.html>  player.cpp  line14, LoadModel  **TREE**  <https://free3d.com/3d-model/tree-67970.html>  coursework2.cpp  line 273, loads mesh | The player object. The camera is centred around the player object and the player object can be moved by right clicking.  The tree is split up into two separate meshes; the trunk and the leaves. Then I apply the bark texture to the trunk and leaf texture to the leaves. I made a custom object loader to split the entire mesh into two materials, making texturing a bit easier. |
| **TR 3 – 3D Transformations** | | |
| Object 1 - procedurally generated | | |
| *Please give a screenshot of transformed object* | *reference specific code (filename and line)* | *Description of transformations* |
|  | **WATER**  Line 41  **TREE SHADOWS**  Line 116-118 | The water is moved down by three units, to better align with the generated terrain.  The tree shadows are translated, rotated and scaled to fit to the tree it represents. |
| Object 2 - OBJ parser | | |
| *Please give a screenshot of transformed object* | *reference specific code (filename and line)* | *Description of transformations* |
|  | **TREES**  tree.cpp  Line 36-38  Scale, rotation, position for each tree randomised.  Line 155-157  Usage of randomised variables for transforming trees. | All trees are scaled and rotated about their y axis randomly when they are initialized. This gives variety to the trees. |
| **TR 3 – Animation** | | |
| *Please give a screenshot of animated object* | *reference specific code (filename and line)* | *Description of animation* |
|  | **TREE SWAY**  tree.vert  Lines 18-25  **WATER MOVEMENT**  water.vert  Line 12-13 | This vertex shader “sways” the vertexes of both the trunk and leaf vertices of the tree. To make it look realistic, this effect is applied more the higher up the tree the vertex is.  This vertex shader moves the y coordinate of the vertices of the water plane by sin and cosine. This gives it a somewhat realistic waving motion. |
| **TR 4 – Camera** | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of camera* |
|  | **ORIBITAL CAMERA**  Camera.cpp  Line 13, UpdateVectors  Line 34, ProcessMouseMovement  Line 49, ProcessMouseScroll  Coursework2.cpp  Line 36, mouse\_callback  Line 102, scroll\_callback | The camera follows the player object and can be moved about the player by holding right click and panning around. You can also zoom in and out by using the scroll wheel. |
| **TR 5 – Texture** | | |
| Object 1 - procedurally generated | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of texture* |
|  | **TEXTURED TERRAIN**  Terrain.cpp  Line 16-18, loads textures.  Tile.frag  Line 59-64 blends textures together.  Texture.h  Loads textures and generates mipmaps.  [*https://polyhaven.com/a/cliff\_side*](https://polyhaven.com/a/cliff_side)  [*https://polyhaven.com/a/rocky\_terrain*](https://polyhaven.com/a/rocky_terrain)  [*https://polyhaven.com/a/sandy\_gravel\_02*](https://polyhaven.com/a/sandy_gravel_02) | The terrain has three textures associated to it: grass, stone and sand. Grass is the default texture, stone is mixed in depending on the slope of the tile, and sand is mixed in just above the water level and below.  This gives a very satisfying and smooth transition between textures depending on where they are placed. |
| **TR 6 – Lighting** | | |
| Object 1 - procedurally generated | | |
| *Please give a screenshot* | *reference specific code (filename and line)*  tile.frag  Line 67-72  Lighting | Handles shading based on the normals of the tile. If a tile is facing away from the sun, it is shaded darker (as seen in the first picture) and vice versa. Gives a sub-conscious awareness of the direction of light and depth in the terrain. The lighting system also contains specular highlights for the terrain. |
| Object 2 - OBJ parser | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of lights on object* |
|  | tree.frag  Line 50-54 | Similarly to the tile fragment shader, the trees also use normals to shade based on the direction of the sun. |
| **TR 7 - Shadow** | | |
| Object 1 - procedurally generated | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of shadow on object* |
|  | tile.frag Line 42-70 (ShadowCalc)  Line 101-102 (Usage)  terrain.cpp  Line 52-88 (Render) | Terrain shadows are calculated using a depth map, generated from the sun’s perspective. A light space matrix projects terrain geometry into light space, and the shadow map is sampled in the shader using PFC (Percent Closer Filtering) to soften edges.  Shadows become softer and less intense at lower sun elevations. |
| Object 2 - OBJ parser | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of shadow on object* |
|  | tree.frag  Line 19-39 (ShadowCalc)  player.frag  Line 15-32 (ShadowCalc) | Both the tree and player use the same shadow mapping system. Each object renders a depth pass from the sun’s view, and samples that in their fragment shader. The shadow is blended on the sun’s elevation, with softer shadows during dawn/dusk. |
| **TR 8 - Interactive object** | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of interactive object* |
|  | Coursework2.cpp  Line 40-76 (Mouse callback)  Player.cpp  Line 67-91 (Update) | The player character is an interactable object. Left clicking anywhere on the terrain performs a ray cast to find the clicked location, accounting for the variable in y values. The player rotates about the y axis while moving in a straight line towards the clicked tile. |
| **TR 9 – Curves** | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of curves* |
|  |  |  |
| **TR 10 – Transparency** | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of transparency* |
|  | Coursework2.cpp:  Line 191-192 (glEnable)  Water.frag  Line 40 (0.35 alpha) | The water surface uses alpha blending to appear semi-transparent, allowing terrain and objects below it to remain visible. Blending is enabled globally and controlled in the fragment shader. |
| **R&D**  Please provide details of any research and development you conducted, as additional techniques not in the lecture notes. | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of Research including websites, articles, references, etc.* |
|  | Fog\_post.frag | **Post Processing Fog**  I implemented a post-processing fog effect in a full-screen pass using the scene’s depth buffer and inverse view/projection matrices. This lets me compute the world position of each fragment in screen space and apply exponential fog based on the camera-to-fragment distance. The result is a smooth, volumetric fog effect that blends the scene colour with a dynamic fog colour depending on the time of day.  <https://cs.gmu.edu/~jchen/cs662/fog.pdf> |
|  | Sun.h  Line 23-24  Sun angle and elevation variables  Sun.cpp  Line 8-30  Update sun function  Tile.frag  Line 35-42  Example of PCF shadow | **Dynamic Sunlight and Shadow Mapping**  My lighting system extends the standard Phong lighting model taught in the lectures by implementing a dynamic directional light (the sun). The position and elevation of the sun changes over the course of the day. This impacts both the lighting direction and intensity, simulating a real day/night cycle.  I use a light space matrix to generate a shadow map using shadow mapping, then apply a Percent Close Filter (PCF) in the fragment shader to smooth the edges of the shadow. I believe PCF was not covered in the lectures.  Additionally, shadows are less intense depending on the sun’s elevation. This enhances realism, as we don’t get hard and tall shadows at dusk or dawn.  <https://www.ogldev.org/www/tutorial42/tutorial42.html> |
|  | Tile.frag  Line 58-64  Mix function used to mix textures depending on params. | **MixMap Texture Blending**  I implemented procedural terrain texture blending in the tile.frag shader. It blends multiple textures, like sand, grass and rocks. This goes beyond the standard UV mapping taught in the lectures, which use a single texture per mesh.  My shader for terrain tiles samples from 3 diffuse maps per fragment and blends them with smoothstep based on their slope and y height. In the example picture, we can see a sharp slope protruding from the grass, which is in turn textured with much more stone than grass. Another example from my program is that it is highly textured with sand under y = -1 because that is where the water layer is.  <https://stackoverflow.com/questions/1110844/multiple-texture-images-blended-together-onto-3d-ground> |
|  | Water.frag  Line 8-21  Noise function  Line 23-28  Shimmer function  Line 37 Usage of shimmer  Water.vert  Line 12-13  Sin and cos functs | **Water Noise Animation**  The water surface is procedurally animated using time based sine and cosine waves in the vertex shader to simulate motion. Additionally, the fragment shader uses layered procedural noise to create shimmering highlights that simulate the interaction between sunlight and small ripples.  [**https://www.jayconrod.com/posts/34/water-simulation-in-glsl#:~:text=The%20mesh%20gets%20drawn%20with,each%20wave%20are%20configurable%20parameters**](https://www.jayconrod.com/posts/34/water-simulation-in-glsl#:~:text=The%20mesh%20gets%20drawn%20with,each%20wave%20are%20configurable%20parameters)  [**https://arxiv.org/pdf/1109.6494#:~:text=compute%20a%20displacement%20map%20using,see**](https://arxiv.org/pdf/1109.6494#:~:text=compute%20a%20displacement%20map%20using,see) |
| N/A | Object\_loader.cpp  Line 93 LoadMeshByMaterial  obj\_loader.h  line 6 MeshSegment struct  Line 44 LoadMyObjWithNormals | Custom object loaders that allow for storing verts, materials and texturefile names in a struct and loading with normals. |
|  |  | **Other background watching/reading:**  <https://youtube.com/playlist?list=PLRIWtICgwaX0u7Rf9zkZhLoLuZVfUksDP&si=wt0xSlCy39WRCoAS>  <https://youtube.com/playlist?list=PLRIWtICgwaX23jiqVByUs0bqhnalNTNZh&si=MjuQzjViVnKwoh7Y> |