COMP3011 Computer Graphics

Assessment 3

Report Sheet (v6)

Use this table to help you prepare for your demo. **Submit this report to Moodle**.

**Student Name: Callum Berger**

**Student ID: 20306440**

**Username: psxcb5**

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

|  |  |  |
| --- | --- | --- |
| **Introduction** | | |
| *Please explain why you implemented this scene* | *Describe your inspirations* | *Provide a general description of the scene.* |
| This scene was implemented to capture the different elements learned within this module into a compact scene. | *I was inspired by my own memories of campfires during camping trips. I tried to depict how I would imagine a campfire deep within a forest would look like based on my memories of similar events.* | *The scene shows a campfire surrounded by trees. There are benches that surround the campfire that the camera to jump between and view from different angles.* |
| **TR 2 – 3D Modelling** | | |
| Object 1 - modelled by hand | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of object* |
|  | *File: Log.cpp*  *Line: 8* | The ‘benches’ and the ‘campfire logs’ are modelled by hand. These are hexagonal prisms transformed with rotation, scale and translate to reuse the model for different purposes. |
| Software was not used to model this shape | | |
|  | | |
| Object 2 - procedurally generated | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of object* |
|  | *File: River.cpp*  *Line: 10* | A procedurally generated river, created from an initial polygon. The next polygons are generated as an offset of the first, with a randomised y value; |
| Object 3 - OBJ parser | | |
| *Describe memory allocation and freeing, data structures, how you read the file and parsed the vertex attributes.* | *Provide the URL for the OBJ file you submitted* | *Describe how you transferred the vertex attributes to OpenGL* |
| Std::vector were used to store vertex, normal and colour data. The file is read by iterating through the OBJ file, checking the beginning of each file line for ‘vn’, ‘v’ or ‘f’. ‘vn’ refers to the normal, ‘v’ the vertices and ‘f’ the faces. For each of these categories found, the x, y and z data is extracted from the line and stored in the corresponding vector for that category of data. For the faces, the face vertex and normal data is then pushed into updated vertex and normal vectors producing the final vectors for the object. Colour data is retrieved from MTL files and applied to the vertices when requested by the OBJ file.  For textured objects, MTL is not included and UV cords are read through the ‘vt’ indicator in the OBJ file. | *URL:* [*https://www.turbosquid.com/3d-models/3d-assets-tree-grass-rocks-1498368*](https://www.turbosquid.com/3d-models/3d-assets-tree-grass-rocks-1498368)  *The tree, rock, grass, and mushroom object were obtained from this URL*  *URL:*  [*https://www.turbosquid.com/3d-models/blender-obj-free/1027135*](https://www.turbosquid.com/3d-models/blender-obj-free/1027135)  *URL for the moon object* | The final vectors created within the loading process are then used to pass the data to OpenGL. A vertex array object (VAO) is created for the objected and bound. For each vertex attribute (vertex, normal, colour) a vertex buffer object (VBO) is created and allocated memory in accordance with the size of the relevant vector of data it will be storing. The VBO is then bound. The attribute pointer then defines the format of the data and which index the attribute is for (in accordance with the vertex shader) before being enabled. |
| *Describe which elements are parsed from your OBJ file, e.g. vertex pos, normal, UV, number of textures or sub-objects etc.* | | |
|  | | |
| **TR 3 – 3D Transformations** | | |
| Object 1 - modelled by hand | | |
| *Please give a screenshot of transformed object* | *reference specific code (filename and line)* | *Description of transformations* |
|  | *File: Campfire.cpp*  *Line: 13* | *The campfire itself contains 6 transformed models by hand.*  *They are each rotated on the y axis by 60 degrees more than the last.*  *They are then translated on the z axis to move them outwards from the centre point.*  *They are then rotated by 45 degrees on the x axis to produce a leaning effect.* |
| Object 2 - procedurally generated | | |
| *Please give a screenshot of transformed object* | *reference specific code (filename and line)* | *Description of transformations* |
|  | *File: Render.cpp*  *Line: 197* | *The river object is translated and scaled in order to better fit the gap that it has been placed within.* |
| Object 3 - OBJ parser | | |
| *Please give a screenshot of transformed object* | *reference specific code (filename and line)* | *Description of transformations* |
|  | *File: Render.cpp*  *Line: 107* | All tree, grass, mushroom and particle objects are transformed. All provide a generated random rotation whilst trees also include a generated random scale. |
| **TR 3 – Animation** | | |
| *Please give a screenshot of animated object* | *reference specific code (filename and line)* | *Description of animation* |
|  | *File: FireParticle.cpp*  *Line: 256* | *Particles will gradually rise and move in directions as defined by a curve. The colour of these fragments will change from yellow to read as they move and rotate. They will eventually despawn when their lifetime has reached an end and will respawn at a new start position. This loops continuously.* |
| **TR 4 – Camera** | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of camera* |
|  | *File: camera.h*  *Lines:*  *45*  *77*  *106* | Camera has a start position looking at the campfire.  Pressing space moves the campfire to a new bench and different viewing angle.  The view can be moved through the mouse.  The camera can be moved using the keys (W, A, S, D). |
| **TR 5 – Texture** | | |
| Object 1 - modelled by hand | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of texture* |
|  | *ile: River.cpp*  *Line: 78* | *Water texture applied to a polygon. The use of a seamless texture allows it to be applied without alteration to a series of linear polygons* |
| Object 2 - procedurally generated | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of texture* |
|  | *File: River.cpp*  *Line: 78*  *Texture URL:* [*https://www.deviantart.com/berserkitty/art/Seamless-Cartoon-styled-Water-Texture-743787929*](https://www.deviantart.com/berserkitty/art/Seamless-Cartoon-styled-Water-Texture-743787929) | *Texture of water applied to the procedurally generated river object. UVs generated based on the vertices of the shape.* |
| Object 3 - OBJ parser | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of texture* |
|  | *File: texture.frag*  *File: texture.vert*  *File: Moon.cpp*  *Line: 102* | *Textured moon object using a texture2D wrapped onto a sphere using UV coordinates in a texture shader.* |
| **TR 6 – Lighting** | | |
| Light 1 | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of light* |
|  | File: Render.cpp  Lines: 241, 189 | Point Light: light is used to emulate the light from a fire. The light can change colour in accordance to the colour of the fire particles.  The light ambience is also altered within a range of 0.1 to 0.3 to give a flickering effect to the light. |
| Light 2 | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of light* |
|  | File: Render.cpp  Lines: 234, 189 | Directional Light: Used for a daytime effect on the scene. This light also provides a more consistent shadow effect with no changes on ambience or colour. |
| **TR 7 - Shadow** | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of shadow* |
|  | File: phong.frag  Line: 20 | Shadow for directional light. Uses shadow FBO and shadow texture to create a depth map. This is then used to calculate the shadows in the fragment shader. |
| **TR 8 - Interactive object** | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of interactive object* |
|  | *File: FireParticle.cpp*  *Line: 236* | *Using the keys (R, G, B) the colour of the fire can be changed as well as the spotlight colour. Colour change corresponds to the key pressed.* |
| **TR 9 – Curves** | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of curve(s)* |
|  | *File: FireParticle.cpp*  *Line: 205* | *Curves are used to generate the path for the fire particles to follow. The curve is generated using the Bezier curve function. Control points are generated randomly, with a slow increase in the y axis to make the particles rise.* |
| **R&D**  Please provide details of any research and development you conducted, as additional technique not in the lecture notes. | | |
| *Please give a screenshot* | *reference specific code (filename and line)* | *Description of Research including websites, articles, references, etc.* |
|  |  | *No code was used from an outside source, however the following URL:* [*https://github.com/VictorGordan/opengl-tutorials*](https://github.com/VictorGordan/opengl-tutorials)  *Provided very useful reference points into understanding how implementation works for certain aspects of OpenGL. This came in very useful into understanding why certain implementations weren’t working or how to approach issues that I had come across.* |
| **Conclusion** | | |
| *Please describe what you perceive to be the strengths and weaknesses of your project* | *Describe what aspect of it you are particularly proud of, and what you think would need to be improved.* | *Reflect on what you have learned during this project that you can apply in future projects to improve your performance.* |
| Strengths:  The main strengths of this are the effect of the centre fire. Simulating colour changing and the motion of an actual fire. This coupled with the environment objects helps create a nice looking low poly scene.  Weaknesses:  The core weakness is the issue with shadow. I attempted making shadow for the point light before referring back to the directional light to focus time on other components in the scene. Additionally the use of textures is minimal but I felt these weren’t as required in the scene. | I’m proud of the particle effect created and the definition that this has.  Shadows could use improvement and would help define a better scene. Point light shadows seem best implemented using a shadow cube map however I did not get round to implementing this.  With more time I would have improved on the procedurally generated mesh to include animation for the effect of moving water | What I have learnt through shaders, although I am still learning, will help understand how shaders work within other software, such as Unity, which I can use to create more interesting scenes. |