CG Assignment 3-2:

Rafiqi Rosli - a1682431 UG
Jamie Rogers - a1687523 UG
Katon Zaky - a1680219 UG

Our Game:

Target Finder - Beach Edition!

Controls:

-Use W-A-S-D to move the first person camera (FPC activated by ‘1’ but also default camera).
-Mouse movement affects the direction the camera looks.
-Pressing Left-Mouse-Button will spawn a sphere at the centre of the platform which is environment mapped.
-Pressing F will enable the flashlight effect displaying showing the use of a spot light
-Pressing 2 will switch to the world view camera which is centered on the middle of the platform, and can be zoomed in and out by holding the right mouse button and moving the mouse up/down.

Graphical Component Implementation Explanation:

Directional Light (1): Rafiqi
Static bright orange light casted over the map. Application of shadow mapping to simulate a sunlight effect.

Point Light (1): Rafiqi
Light hovers around the map with point shadow effect (Omnidirectional shadow mapping)..

Spot Light (1): Rafiqi
Press ‘F’ for flashlight effect. It does not have any shadow as it does not look as good as it should. The biggest challenge was getting the flashlight curve effect. This has to be done in the fragment shader and we have to calculate the attenuation and the amplifying effect of the light when it casts upon another light.

Loading Objects - (3): Katon
A multitude of objects were loaded onto the beach, such as the hammock, trees, rocks etc. This task took the longest effort for me because of the trial and error results that had to go through as the given objects were at different sizes, scaling and rotations also affects the placements of the object when translating.

Multiple Cameras - ‘1’ and ‘2’ (2): Katon
The game initially starts on the First-person camera (selected by pressing ‘1’).
You can also switch to a world-centred camera by pressing the ‘2’ key. The radius from the centre can be adjusted by holding the right mouse button and moving the mouse forwards and backwards.

Texture Mapping (1): Katon
All of the loaded objects utilize texture mapping to create a textured surface. Some of the textures that were provided with the objects did not work as intended because our object loaders could not utilise with the texture coordination placement on the objects. This was fixed by replacing the given texture with a patternable texture.

Bump Mapping - Brick Wall, Sand Floor (1): Rafiqi
Some models will have 2 textures. Diffuse texture and Normal texture.

Bump mapping uses the normal map texture as the normal attributes for the lighting. Applying the texture directly in the fragment shader will produce the 'bump' effect but the lighting position would be incorrect. To fix this problem, tangents and bitangents for each triangles have to be calculated and multiply the light position values with the calculated values.

Parallax Mapping - Brick Wall, Sand Floor (2): Rafiqi
This uses the tangents and bitangents from above to get the correct lighting position. It enhances the 3d effect from the 2d textures by applying a displacement texture. There are 2 additional techniques to give a more realistic feel to the effect and prevent distortion from closer angles. Steep Parallax mapping add layers effect by taking multiple samples instead of 1. Parallax Occlusion mapping linearly interpolate between the depth layer after and before the collision from the eye position.

Skybox (1): Rafiqi
The whole area is surrounded within a cubemap that has a skybox texture. The skybox texture consists of 6 images of different angles so that it would wrap around the cubemap nicely. The cubemap is done inside a separate shader file called cubemap.vert/frag. There was a problem with the view matrix and the skybox at first as the skybox texture would follow the viewer direction rather than staying still. To fix this, in our render() function the skybox would be drawn last and the view matrix will remove of its translation component.

Depth Cue - Fog (1): Jamie
Implemented in (debug.frag), uses an exponential function (fogFactor = exp2(-density * density * z * z * LOG2), works by calculating the distance between the camera and each fragment, and then adjusts the blending amount between the fog colour and the fragments actual colour depending on the distance. The further away, the closer it gets to the fog colour.

Alpha Blending - Ferns on ground, Smoke and fire particles (1): Katon
Allow the alpha properties in images to be displayed properly by taking alpha values to the fragment shader and enabling GL_ENABLE(GL_BLEND) inside OpenGL. One problem is that due to the sky box being implemented in another shader, it will create a grey background of the object when the camera is looking at the object whilst having the skybox in the background.

Multiple Vertex/Fragment shaders - Skybox, shadows, objects, depth. (2): Rafiqi
Our program has separate shaders for the shadows (depth.frag/vert & depthCube.frag/vert), objects and fog(debug.frag/vert) and more

Environment Mapping - Sphere projectile, Left-mouse button shoots projectile which has environment mapping (2): Rafiqi(env map)/Jamie (projectiles and key binding)
Pressing the 'LMB' Key will launch a environment mapped sphere from the centre of the platform towards the camera position (player).
The reflection is produced by introducing another fragment/vertex shader called water.vert/frag. It is supposed to be used to emulate water effect but since we proved that it is quite hard and time consuming we decided a simple reflection and refraction effect on an object will suffice.
The reflection is mostly done by glm/glsl? reflect() function.
Same thing goes for refract().

Multi-pass method: Shadow mapping (3): Rafiqi
The shadow is being casted from a light space matrix that is calculated by multiplying the light position and the projection of the light (orthogonal or perspective). We use the light space matrix to get the depth of the shadow as the light passes through the world objects. Then, use the result of the depth map as the shadow coordinates and generate a 2D shadow texture within the frame buffer. Some issues are Peter-panning effect (Shadows are detached/floating from the actual object) and over-sampling so it requires a bit of trial-and-error for correct far and near plane values.

Procedural generation of particles(?): Fire and smoke particles(3-5): Katon
I did not really intend on creating this at the start, but an idea came up after Jamie and Rafiqi successfully implemented the shooting sphere projectile. I realised I could use the same method in order to create the particles. Each particles are a 2D object with an assigned random starting coordination in a range of and 'timer'- an incrementer in the while loop which controls the movements of the particles. The particles does not die unless it has reached a certain 'timer' threshold. The problem that I

couldn't pass through is to get the rotation of the 2D object to face the camera. Getting xyz of the camera is possible, but to determine the angle it should rotate was quite complicated to calculate.

Collision Detection - player movement restrictions (1): Jamie
If the player tries to walk through the fence, a detection system will prevent the movement exceeding the boundary limit, additionally it will only stop the component of the direction which would take the camera outside the boundary, i.e. you can slide along the side of the fence.

Simple Sounds - Footsteps/Bullet hitting target (1): Jamie
Implemented by using the fork() command and a timer to create a child process which makes a call to system to play the sound. Using this method prevents stalling of the current process while the system call takes place. The sound implemented was footsteps which are activated by walking in the first-person camera view.

Total: 27

Extra Features:

Anti-Aliasing - Smoothing edges of 3d Models: Jamie
Remove the jagged lines from 3d models. We used glfw to do this for us by calling glfwWindowHint(GLFW_SAMPLES, 4) which gave us MSAA x4.

Gamma Correction - Adjust brightness: Katon
Gamma correction is used to change the intensity from the lights to the environment and how the environment reflects to it.
We made sure that the scene is a little bit dark so that the shadows and lights are more obvious.
Gamma is done by the exponentiation of the 1.0/gamma value and the output of the fragment.
Fragcolour = pow(result, (1.0/gamma));

Point Shadow (Omnidirectional shadow mapping) Rafiqi

Point shadow uses almost the same technique as the standard shadow mapping but it takes a cubemap instead of a depth map to do the shadow positioning. The cubemap will be used to calculate the shadow coordinates from the center of the light position to the surrounding objects (the light is transformed 6 times for 6 different directions). This is done inside the geometry shader. This type of shadow feels more dynamic but it is mostly used for point lights as the shadow is projected to all directions. PCF had also been applied here to create a more realistic shadow effect.