Graphics Document

Player Controls:

Keyboard Controls:

W: Move camera forward

A: Move camera left

S: Move camera backward

D: Move camera right

Q: Move camera downward

E: Move camera upward

Space: Alternate between arrow keys and mouse controls

(When in keyboard mode)

Up Arrow: Rotate camera upward

Left Arrow: Rotate camera left

Down Arrow: Rotate camera downward

Right Arrow: Rotate camera right

(When in mouse mode)

Mouse: Rotate camera

GUI Controls:

- Wireframe mode
- Depth of field blur

Shaders:

TessellationQuadShader:

This shader is for rendering the heightmap. The heightmap is sampled and the position of every vertex on the plane is offset by the colour value of the heightmap multiplied by a modifier. The four edges and the centre of every quad making up the plane are tessellated more when the player is close to them and less when the player is farther away. The heightmap is textured and lit using a normal map before being rendered.

Vertex shader:

- Alters nothing but is a necessary part of the shader pipeline.

Hull shader:

Passed in:

Camera position

- Takes the position of the camera through a buffer.
- Finds the position of the edges by averaging the positon of the two adjacent vertices and the position of the centre by averaging the position of two opposite corners.
- Samples the heightmap and offsets the edges and centre by its colour. This is to gain an accurate position for where they will be when the heightmap is tessellated.
- Checks the distance from each edge and the centre to the camera and if it is less than ten, increases the tessellation factor

Domain shader:

Passed in:

World, view and projection matrices

- Interpolates between the four vertices and finding the correct vertex, normal and texture given the correct coordinate.
- Samples the heightmap and offsets the position of the plane by the colour value of the heightmap multiplied by a modifier.
- Sets the position of the vertex (for light direction calculation) to the vertex position found earlier (stored as Position3D).

Geometry shader:

Passed in:

World, view and projection matrices

- Finds the vector representing the line between the first and second vertex of the triangle, and the second and third vertex of the triangle.
- Finds the normal of the triangle by finding the cross product of these two vectors.
- Finds the binormal by taking the cross product of the normal and of the up vector (0,1,0).
- Finds the tangent by taking the cross product of the normal and of the binormal.
- Streams the output position, texture, colour and position3D of the three vertices. These are the same as the input.

Pixel shader:

Passed in:

Light ambient colour

Light diffuse colour

Light position

Light direction

- Takes in the height map, normal map, texture and sampler.
- Takes in light information through a buffer.
- Samples the texture to find the colour of this pixel.
- Samples the normal map to find the value the correct normal at that pixel.
- Multiplies the normal by two and minuses one to find the normal value between -1 and 1 instead of 0 and 1.
- Finds the correct normal by multiplying the normal map's x value by the tangent, the y value by the normal and the z value by the binormal.
- Sets the colour to the ambient light value
- Checks if the light direction is set to (0,0,0) (meaning that it is a point light).
- If it is, sets the light direction as the negative of the pixel position minus the light position.
- If it is not, takes the negative of the light's direction.
- Normalises the light direction vector.
- Finds the intensity of the light by:
    - Finding the dot product of the normal and the light direction.
    - Clamping it between 0 and 1.
    - Selecting either that value or zero, whichever is greater.
- If light intensity is greater than zero, the diffuse colour multiplied by the light intensity and adds it to the colour of the pixel, then clamps this between 0 and 1.

- Multiplies to colour by the texture colour.

DepthShader:

This shader is for doing a depth pass on the height mapped plane. Using just a simple depth shader only found the depth values at the vertices and so a new shader was required which would go through a similar process to create a height mapped plane to find the correct depth values. The depth values are displayed as a greyscale colour value and then can be passed to other shaders.

Vertex shader:

- Alters nothing but is a necessary part of the shader pipeline.

Hull shader:

Passed in:

Camera position

- Same as TessellationQuadShader.

Domain shader:

Passed in:

World, view and projection matrices

- Same as TessellationQuadShader.

Pixel Shader:

- Finds the depth value by dividing the z component of the pixel's position by the w coordinate of the pixel's position.
- Set's the colour of the pixel to the depth position in the x, y and z components and one in the w component

DepthBlurShader:

This shader is a simpler version of the depth shader that does not need to offset vertices by a height map or tessellate as it is not used for finding the depth of the height map, only the other objects in the scene.

Vertex shader:

Passed in:

World, view and projection matrices

- Sets the output position to the input position multiplied by the world, view and projection matrices.
- Sets the depth position to the output position.

Pixel Shader:

- Finds the depth value as the depth position's z component divided by the w component.
- Returns the depth value as a colour

GeometryShader:

This shader is used to render the snow particle effect in the scene. It renders the point mesh containing ten thousand points, each rendering a billboarded quad with a texture of a snowflake.

Vertex shader:

- Alters nothing but is a necessary part of the shader pipeline.

Geometry shader:

Passed in:

World, view and projection matrices

Camera position

- Receives the camera's position through a vector.
- Finds the forward vector relative to where the quad should point by taking the camera's position away from the position of the point and normalizing the result.
- Finds the right vector relative to where the quad should point by taking the cross product of the forward vector and the global up vector (0,1,0) and normalizing the result.
- Divides the right vector by ten to make a small quad.
- Finds the up vector relative to where the quad should point by taking the cross product of the forward vector and the right vector and normalizing the result.
- Divides the up vector by ten to make a small quad.
- Defines the four corners of the quad by taking the point position and either adding or subtracting the up and right vector to find the points up, down, left or right of the point position.
- Multiplies these position values by the world, view and projection matrices and streams the output.

Pixel Shader:

- Samples the texture and sets the output colour to the colour from the texture.
- Clips the texture colour if the alpha value is below 0.5 to make the background transparent.

TextureShader:

The texture shader is a simple shader which is used to apply a render texture to an orthomesh when post processing.

Vertex shader:

Passed in:

World, view and projection matrices

- Sets the w component of the input position to make it four units for the matrix calculations.
- Sets the output position to the input position multiplied by the world, view and projection matrices.
- Sets the output texture to the input texture.
- Sets the output normal to the input normal multiplied by the world matrix.
- Normalises the output normal.

Pixel shader:

- Receives a texture and a sampler.
- Samples the colour of the texture at this pixel.
- Returns the texture colour

HorizontalBlurShader:

This shader does the horizontal component of the Gaussian blur that is used in the depth blur. It takes the texel colour for the four texels to the left and the right of the current texel and blends their colours together.

Vertex shader:

Passed in:

World, view and projection matrices

Screen width

- Sets the output position to the input position multiplied by the world, view and projection matrices.
- Finds the texel size
- Finds the texture coordinates of the nine pixels used in the blur by finding the point 4 x texelsize to the left to 4 x texelsize to the right.

Pixel shader:

- Finds the depth position by sampling the depth texture.
- If the depth is less than 0.95 the weighting is set so it oly takes colour from the current texel (no blur)

- If it is greater it sets the weights to take the other texels colour also. (weighting must add to 1 when done)
- Sets alpha to one.

VerticalBlurShader:

This shader is almost identical to the horizontal blur shader, however it takes the colour values from the four texels above and below the current texel not left and right.

LightShader:

The light shader is used to apply lighting calculations on objects. This is also done within other shaders however this shader takes in more than one light for the calculation.

Vertex shader:

Passed in:

World, view and projection matrices

Camera position

- Sets the output position to the input position multiplied by the world, view and projection matrices.
- Multiplies the normal by the world matrix
- Sets the position of the vertex in world space to the input position multiplied by the world matrix.
- Find the view direction as the camera position minus the world space position.

Pixel shader:

Passed in:

2 ambient colours

2 diffuse colours

2 light positions

2 light directions

- Samples the texture to find the texture colour.
- Initialises to the ambient colour.
- Loops for two lights.
- Finds if the light is point or directional and finds the direction as either the pixel position in world space minus the light's position or the light's direction if it has one.
- Takes the distance of the light direction.

- Takes that direction and checks in against the range the light will cover.
- Sets the colour to the light colour multiplied by the light intensity.
- Finds the attenuation value and multiplies the light value by it.
- Adds the colour values from both lights to the output value.
- Multiplies the output colour by the texture colour.

Shadow shader:

The shadow shader creates a shadow map and then checks what can and cannot be seen from the light that will be generating the shadows. Anything that is not seen but is still inside the shadow volume will be rendered in darkness.

Vertex shader:

Passed in:

World, view and projection matrices

Light view and projection matrices

Light position

- Sets the output position to the input position multiplied by the world, view and projection matrices.
- Sets the light view position to the input position multiplied by the world, view and projection matrices.
- Multiplies the normal by the world matrix
- Sets the position of the vertex in world space to the input position multiplied by the world matrix.
- Sets the light's position to the passed in light position minus the vertex position in world space.

Pixel shader:

Passed in:

Light ambient colour

Light diffuse colour

- Sets up the bias of the shadow map.
- Sets the colour to the ambient colour of the light.
- Find the projected coordinates of the shadow map.
- Samples the depth value from the depth map.
- Finds the depth of the light.
- Take the bias away from the depth of the light.
- If the depth value of the light is within the depth value sampled from the texture (and so the geometry is in light).

- Find the light intensity.
- Check if it is greater than zero.
- Add the light's colour multiplied by the light intensity to the output colour.
- Saturates the output colour.
- If the geometry is outside the shadow's range, it assumes it is in light and does the light calculations.
- Samples the texture colour from the texture and multiplies it by the output colour.

Critical analysis:

In the geometry shader rendering the tessellated height map, I am able to find the correct normals however the normal map is not implemented absolutely correctly. The lighting is also noticeably wrong when lit with a point light.I was not able to find a solution to this however when debugging it seemed there was a problem finding the pixel position in world space before the pixel shader.

The shadows are not rendered. This appears to be because the depth test from the light's perspective is not rendering any geometry however after much testing and referral to the lab work I was not able to find out why.

I originally planned to include specular lighting however when implemented it lit the entire sphere red, despite being set to white. I was not able to find the difference between this faulty code and the working code from the labs.

Sources:

To find what functions in hlsl did:

https://msdn.microsoft.com/en-us/library/windows/desktop/bb509561(v=vs.85).aspx

To find out how to use the normal map:

http://www.rastertek.com/dx11tut20.html

To find out how to find the up and right vector relative to a forward vector:

https://www.gamedev.net/forums/topic/388559-getting-a-up-vector-from-only-having-a-forward-vector/