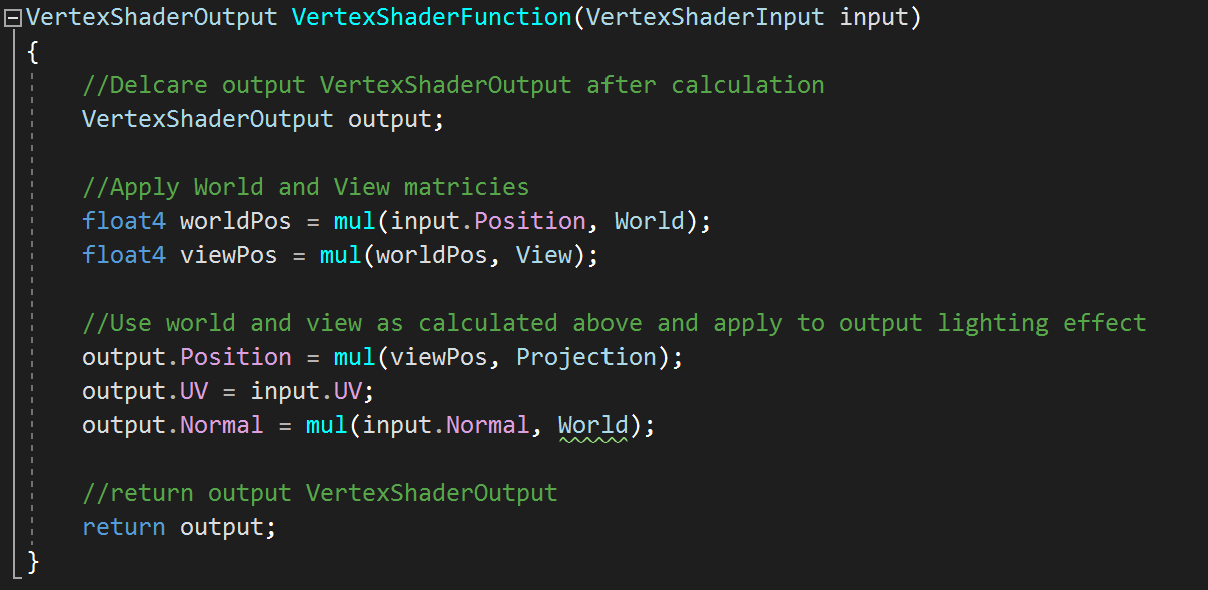
**Documentation for 3D graphics project**

**DirectionalLight.fx**

**Variables Used**

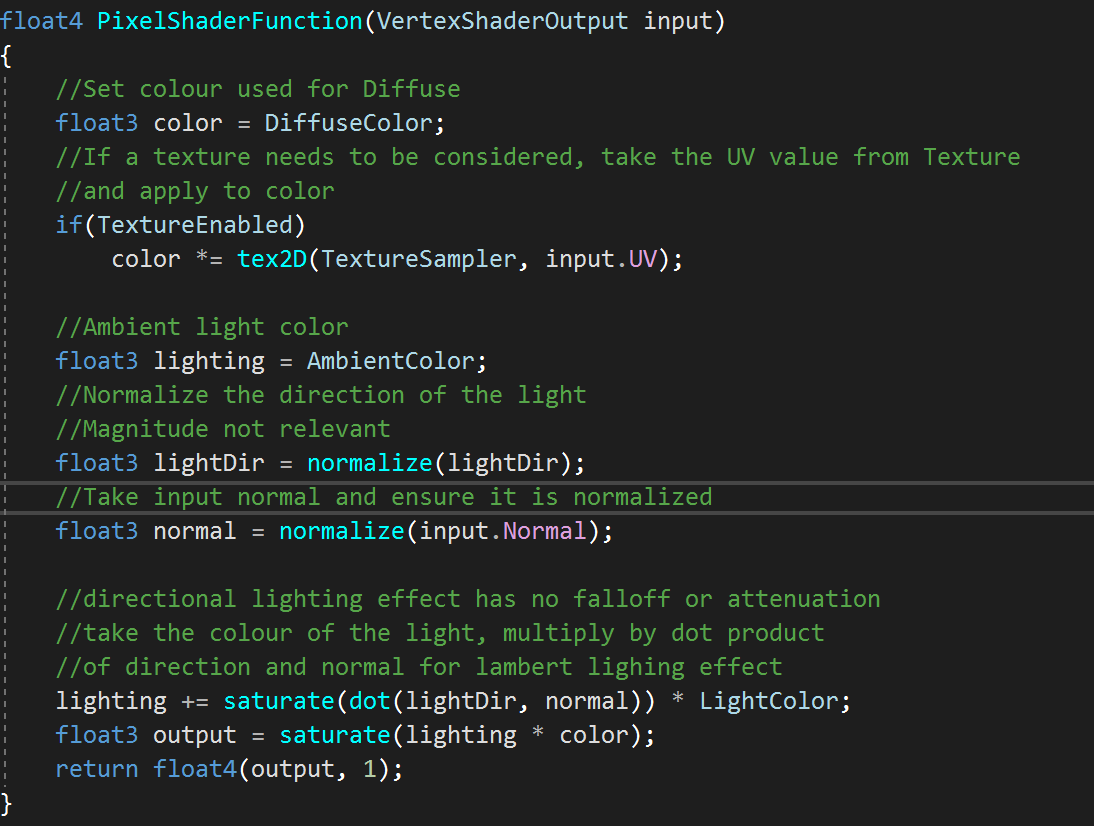
* float4x4 World;
  + Represents the world position matrix to be applied to the effect
* float4x4 View;
  + Represents the View matrix to be applied to the effect
* float4x4 Projection;
  + Represents the Projection matrix to be applied to the effect
* bool TextureEnabled;
  + Boolean to determine if a texture is enabled and hence if the effect needs to take this into account when in PixelShader
* texture Texture;
  + Texture is held here for use in PixelShader if a texture is enabled when effect is applied
* float3 DiffuseColor;
  + RGB representation of the colour used as the diffuse colour of lighting effect.
* float3 AmbientColor;
  + RGB representation of the colour used as the ambient lighting colour.
* float3 LightDir;
  + x, y, z representation of the direction the light will point.
* float3 LightColor;
  + RGB representation of the colour for this lighting effect.

**VertexShaderFunction**



The above function creates a VertexShaderOutput value that has the game’s world, view and projection matrices applied, with the same to the output’s normal and where on the texture map the effect is currently concerned with (UV value). This is used below when calculating the final pixel colour.

**PixelShaderFunction**



Lambert lighting was used for this directional light where

I : Intensity

L : Direction vector

N : Normal Vector

The intensity is clamped between 0 and 1 as it is applied to an RGB colour, the values of which will be diminished by the lambert lighting effect. That is, 1 keeps the intensity of the light and 0 will cancel any lighting effect.

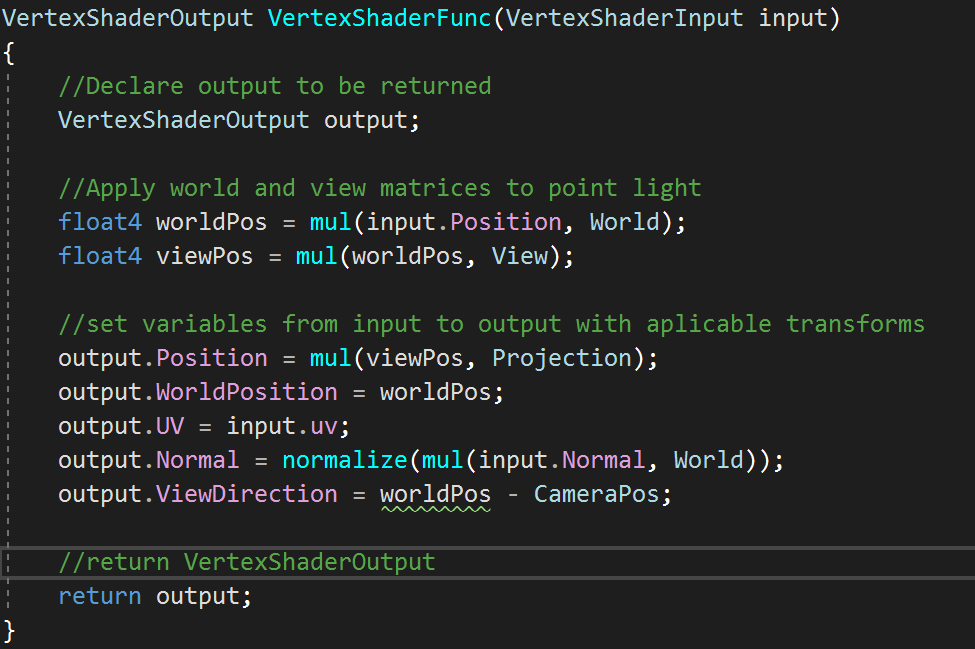
The values LightDir and normal and normalised because their value (magnitude) is not required and only the direction is relevant to the lighting calculations.

**PointLight.fx**

**Variables Used**

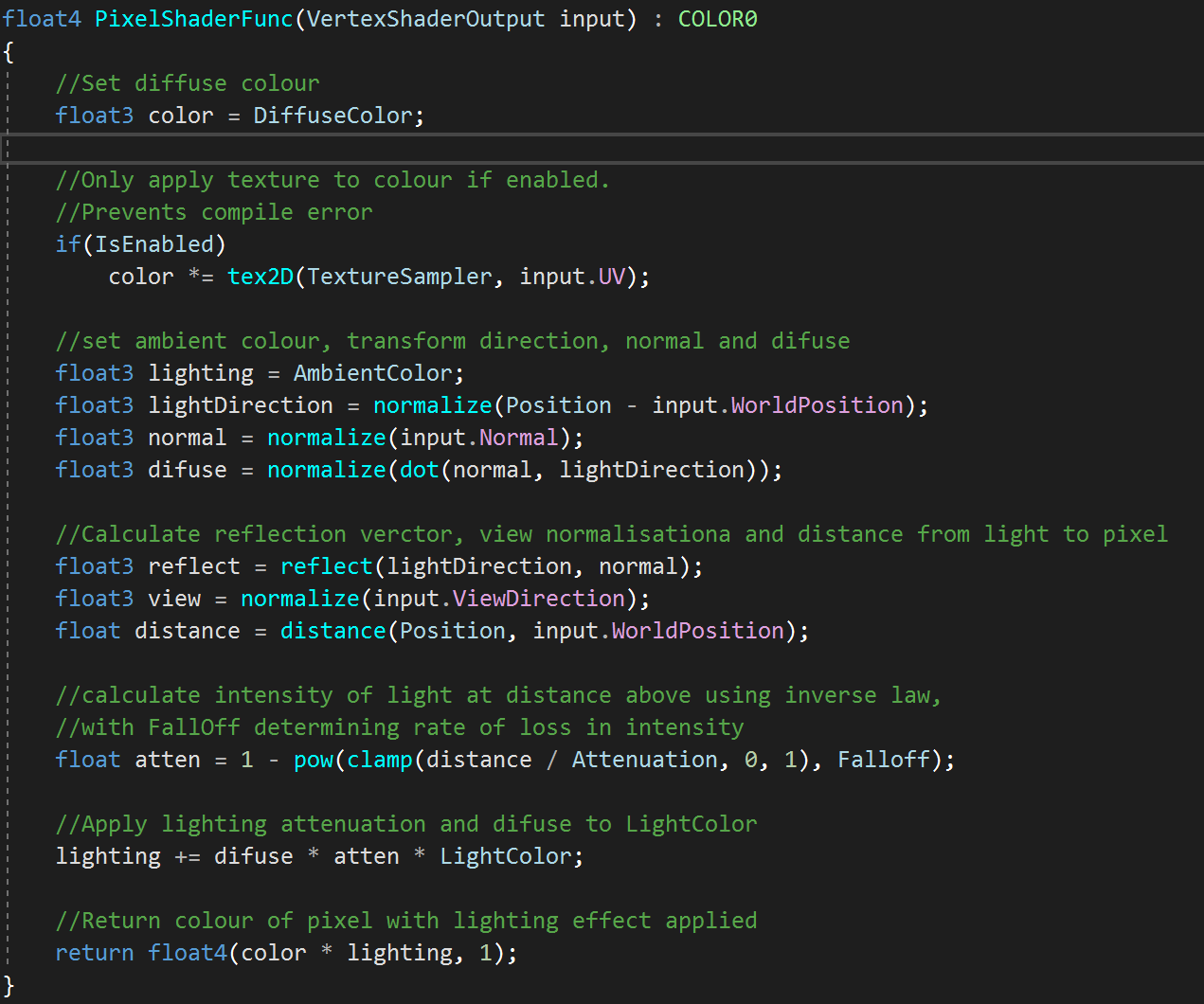
* float4x4 World;
  + World translation matrix.
* float4x4 View;
  + View translation matrix.
* float4x4 Projection;
  + Projection translation matrix.
* texture2D Texture;
  + Texture held here if texture is part of applied effect.
* bool TextureEnabled = false;
  + Boolean to determine if textureSampler is required in PixelShader, preventing exception thrown.
* float3 AmbientColor = float3(.15, .15, .15);
  + RGB representation of the ambient colour.
* float3 DiffuseColor = float3(1, 1, 1);
  + RGB representation of the diffuse colour.
* float3 LightColor = float3(1, 0, 0);
  + RGB representation of the colour of the lighting effect
* float3 Position = float3(0, 0, 10);
  + Position vector of point light
* float Attenuation = 20;
  + Attenuating power of light (how quickly the intensity falls
* float Falloff = 2;
  + Rate at which the lighting effect will diminish
* float3 CameraPos = float3(1, 1, 1);
  + Position of the camera relative to the world position

**VertexShaderFunction**



This Function works in much the same way as the directional light above, however since position of the light must be taken into consideration as well as directionality of the light, this must be calculated too.

**PixelShaderFunction**



Point lights will apply lighting effect that is dependant on the direction of the light and the attenuation of that light. That is, the further the pixel is from the light, the less intense that lighting effect will be.

This uses the inverse square law:

I : Intensity of lighting effect

Is : Intensity of light at source

D : Distance from source to surface

In addition, fall off (rate at which the light loses intensity) is taken into account, where some lighting effect will be more powerful, and thus have a lower fall off rate. It replaces the square in the denominator of the above equation.