# Midpoint Review Design Document

## Basic Rendering

### Buffer Classes

The three buffer classes (Vertex Buffer, Index Buffer and Constant Buffer) are all used to parse data to the GPU. The constant buffer type automatically tells DirectX the size in bytes the buffer will take up, automatically rounding up to the nearest 16 bytes because the packing of constant buffer data is important. Whereas the vertex and index buffers just give the exact size of the data in bytes. This byte width data amongst other information, is described using a D3D11\_BUFFER\_DESC object which is then passed into the respective buffer creation function.

### Shader Classes

The PixelShader and VertexShader classes are used to load in shader data from files and parse it to DirectX in the form of D3D11 shader objects. These can then be bound to the device context when they need to be used.

## User Input

### Windows Call-back

When a key is pressed, it is set up so that a function called WindowProc is called back to. This sorts the windows event data (wParam and lParam) into event data my classes understand using a switch statement.

### Keyboard and Mouse Classes

The keyboard and mouse classes store a keyboard/mouse event stack which is populated by the previously discussed call-back function. These events are then handled sequentially until there are no more to process every frame.

## Camera

### Camera Movement

In the main engine class, where the respective ReadEvent functions for the mouse and keyboard are called, several checks for certain events are made. These include keypresses for W, A, S, D, Q and E. When these events happen, the camera’s position is updated accordingly.

### Rendering the Scene

Before any objects are rendered, the view and projection matrices from the camera are loaded into a constant buffer and passed to the GPU. Both matrices are calculated using the inbuilt DirectX functions for doing so (XMMatrixLookToLH and XMMatrixPerspectiveFovLH).

## Basic Lighting

### Phong Model

The Phong shading model is used to calculate per-pixel colour values in the shaders.

### Directional Lights

Directional lights are modelled to be infinitely far away and thus, only a light direction and colour is needed to create one. The dot product of negative light direction and the surface normal clamped to the range 0 -> 1 can be used as a modifier for the diffuse component.

### Point Lights

Point lights can be placed into the scene which share a lot of the concepts of directional lights but originate from a singular point and have attenuation.

### Spot Lights

Spot lights are almost identical to point lights but only emit in a certain direction. The falloff inner and outer angles can be tweaked to make this cone of light larger or smaller.

## Model Loading

### Model Data

Model data (Vertices, Normals, Tangents, Bitangents, Tex-coords and Indices) is loaded in from model files using Assimp. Once loaded, it is processed and stored in two vectors before being converted into a Vertex and Index Buffer.

### Material / Texture Handling

Assimp also loads material files (.mtl) which store info about the ambient, diffuse, specular properties of the object. They also may contain the relative file path to any textures that that material may use. These texture file paths are passed to the respective DirectX texture load functions so that the texture data can be loaded.

## Advanced Lighting

### Normal Mapping

Normal maps store normal information where each component (x, y and z) are in the range 0 -> 1. These maps are passed to the shader and converted to the conventional range -1 -> 1 by multiplying by 2 and subtracting 1. Then the original surface normal can be adjusted using the sampled normal at that Texel. (Create transposed TBN matrix and multiply the sampled normal by it to get world space normal)

### Parallax Occlusion Mapping

Parallax occlusion mapping uses the TBN or tangent space matrix as well. It will offset the sampled texture cords based on several values in the given depth map.

### PCF Shadow Mapping

The depth information of the scene is rendered from the light’s perspective into a texture. Then in the main render pass, the 3d world position is transformed into the light space and the depth value is calculated. If it is greater than the value sampled in the depth texture, then the pixel is in shadow. This is done several times and averaged to create smoother shadows.

## Object Interaction

### Mouse/3D Picking

To find a Ray into the scene from the mouse, the mouse’s NDC coordinates are calculated and multiplied by the inverse viewProjection matrix. This will get the position in world space. Then you simply create a normalized Ray from the camera to this position. This Ray can then be used to check collisions with objects in the scene. The DirectX collision library is used to make this stage easier.

### Object Translation

When an object is being translated, the mouse to world ray is checked to see where it intersects a vertical plane perpendicular to each axis. Then the relative component of the intersection point is used to move the object. For example: when translating in the x axis the x component of the intersection is used.

### Object Rotation

For object rotation, a similar concept is used. A plane perpendicular to axis of rotation and an intersection point with the mouse Ray is computed. The difference in angle to the axis found with atan2(y/x) is used to work out how much the axis should rotate.

## Object Collision

### Physical Materials

Each collider object will store a physical material object that dictates how the object will behave when simulated in the engine. Properties include: Friction, Bounciness, Mass and Buoyancy.

### Collider Class

The collider class will store a position and the previously described physical material. Then several child classes (PlaneCollider, CuboidCollider, SphereCollider and MeshCollider) add more information and functionality too perform collisions. For Example: The PlaneCollider class will add a plane normal and the Sphere collider will add a sphere radius.

## Miscellaneous

### Error Logging

The ErrorLogger class allows for a windows error dialogue window to be created from anywhere within the program. It can also take an HRESULT type and retrieve a string message from it to ease debugging.