Contents

Overall

Member variables start with m_.

Global variables start with g_

COM pointers are used as much as possible.

There are 3 possible resolutions. And going full screen should be possible. Unless perhaps it tries to go borderless.

Scenes

Particle

This scene basically just displays different particle systems and a dynamic skysphere.

LOD

The scene is divided into oc trees, quad trees and instances.

Heat

Same scene as nr2 but displaying a color based on how many lights affect the area.

Animation

A character with 3 animation tracks, they can be combined with a selected joint as a breaking point, or blended together with an adjustable t-value.

Transform->Shape->DrawCall

Transform is a base class that can input and output transformations, be parented to other transforms etc. Instances of stuff works too. Basically, the stuff that moves inherits from this.

One of these things is the shape class, which basically handles models, this class contains a smart pointer to a draw call which is a struct with the minimal information necessary to be put in the rendering pipeline. Shapes can be culled against a camera since they contain bounds.

The draw call struct is used last.

Bigger Classes

AnimationHandler

Handles, Skeletons and Morph Targets.

Vertex Animation is not supported, Morph Targets are not supported either but might be in the future.

Each draw call object that supports skeleton animation has a map of skeleton tracks.

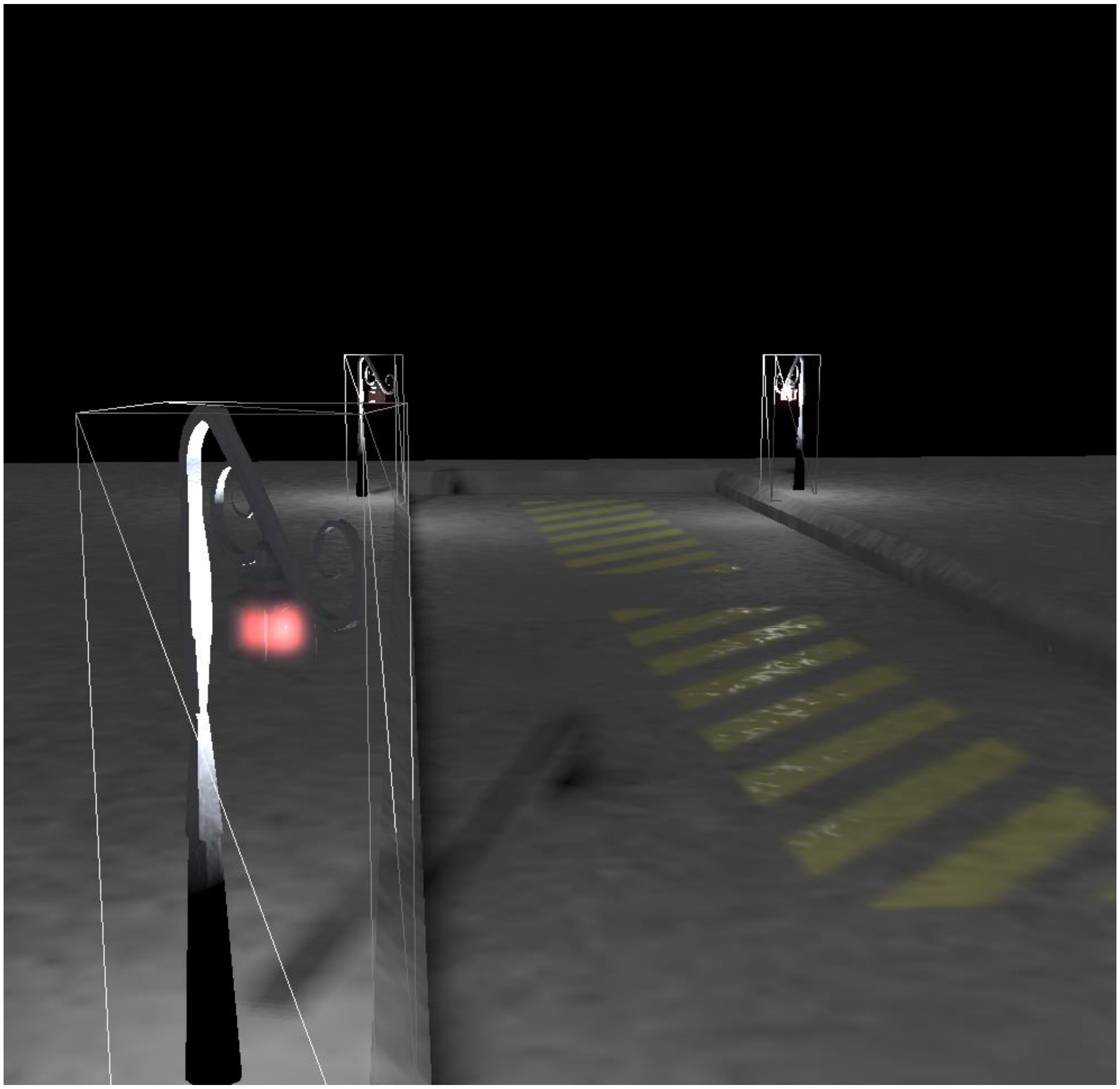
The skeleton tracks are found with global strings referring to what type of track it is, “walk”, “run”, “jump” etc.

The tracks are pointers, any update loop can access them and change the “current track” variable to matching string and the rendering pipeline finds what skeleton from the draw call to use when rendering.

The actual joint/bone matrices are stored inside a structured buffer instead of a constant one, since they are of a huge amount. The skeleton joint matrices are pointers.

BoundingBoxHandler

There are 3 types of bounds built, OBB, AABB, and a spehere. The OBB, Sphere and dynamic AABB culling functions don't work, likely because the math is wrong. It is based on the book - "christer ericson real-time collision". But the implementation is not correct in this engine.



Bounds can be imported from a modeling application, or generated by calculations inside the engine. Each "Shape" node, contains a boundingBox Handler so it can generate bounds, the handler can also generate a bigger set of bounds from a set of bounds.

BufferHandler

This class is used to create either constant buffers or structured buffers. Every object contains a pointer to the buffer, including srv and uav for the structured type.

Not every rendered object has it’s own buffer, but instead a struct that contains the information for a buffer to update with. Update sub resource is used when it’s not updated in real time. Otherwise, the mapping method is used, the actual information is exchanged with memcpy.

GUIHandler

DXTK is used to generate fonts and to display text on screen. DEAR IMGUI is also used to help change variables in real time and display information.

InputHandler

DXTK is used for input information.

LightHandler

Lighthandler is one of 2 classes that the object “Light” inherits from.

It is used to set the light information, creating it and defining what type it is. It can either be directional, point or a spotlight.

Only one directional light is currently supported but more could be added.

The one directional light if used is in a separate buffer than the other lights, it is always used in the scene if it exists for it.

Forward plus is used to cull the other lights when needed.

Because the lights are sent down the GPU inside a structured buffer, they all correspond to a simplified light struct each object has. The struct is updated when the object is.

LODHandler

Quad Trees are used for terrain, either calculated on the GPU when the terrain is generated from a height map, from the book “Introduction to 3D Game Programming with DirectX 11” by Frank D Luna.

Otherwise the mesh is split into nodes on the CPU with the Quad Tree Class.

The way it works is it uses a base AABB that covers the terrain, then it checks the number of triangles inside that box, if it excceeds a set threshold it splits the box down into 4 smaller boxes and repeats this process recursively.

When it's done splitting boxes it checks the amount of triangles inside each box the terrain has and splits it into new meshes dividing the piceses into each separate box.

The same event goes for the OcTree, however the OcTree has a list of all sorts of meshes in the scene and each parent node has 8 children instead of 4.

RenderPassHandler

The handler has a map of renderpasses, and a list of what passes are to be used for the current scene.

A render pass contains things that might need to be changed for a certain pass, such as render target views, srvs, uav, dsv, viewport etc.

They are updated whenever the resolution or the scene changes.

When a render pass is run it calls this object that already has the information needed for that specific pass.

Binding certain elements to null is done in the rendering loop, not by the passes.

ResourceHandler

Contains a scene handler, which does most of the work. It would be possible to split up the scenehandler into multiple objects each inside the resource handler but a the moment the scene handler is deleted and re-created whenever a scene changes.

SceneHandler

The Scene handler creates and imports different meshes/skeletons etc that are unique for each scene. The type of scene also dictates what rendering loop to use and what render passes to activate.

ShaderHandler

Has a pool of shaders, creates them at the start of the application, even if not all will be used immediately, this is because it was an easier implementation.

Also has input layouts. And a function to bind general shaders to null when needed.

SoundHandler

Sounds are also used with DXTK, it’s a static class so the sound effects are called wherever needed. Each sound and shader is loaded once when starting the application, a possible optimization would be to only load the shaders and sounds that are needed for the current scene, once. It would move some loading time from the start to each individual scene instead.

TextureHandler

The texturehandler is a singleton so it can be easily accessed, and only one object of this needs to exist.

It has a pool of texture resources accessed through strings which can be the the pathway or just a given name.

Samplers are here the same way as well.

To avoid having Booleans inside shader code, every mesh rendered has 5 textures minimum, Albedo, Normal, Rough, Metal, AO and Emissive.

When it lacks one of these a pre created texture with the size 1x1 is sent instead.

The color for this texture is either, blue, white, black or grey so it wont have a significant affect on the model when the slot is replaced.

Notes on some 3D Techniques

Third person

To make the character turn around his own axis, sin and cos was used. The camera is parented to the character in third person.

Selective Emissive

The emissive textures are rendered onto a screen quad, along with the rest of the scene, the color is on another target, the srv with the emissive texture is blurred in 2 passes then upsampled to the original size and added with the color texture.

Forward Plus

A combination of 2 apprpoaches was used. The frustums are generated in real time.

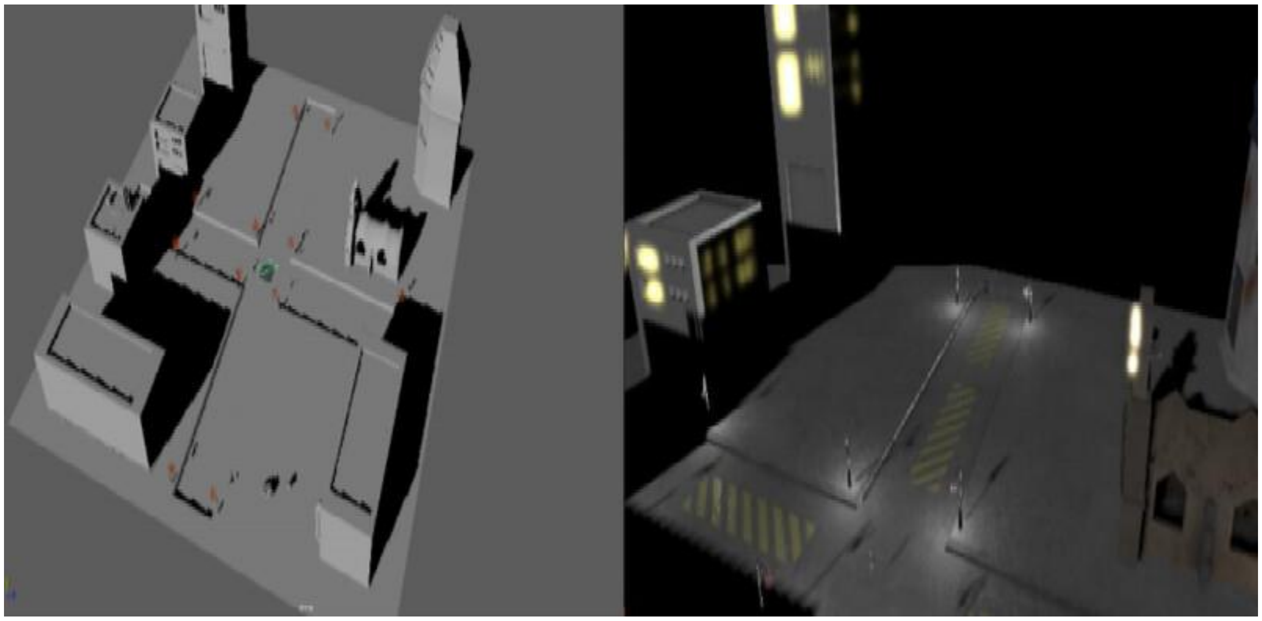
The screen is divided into groups of pixels called tiles.

Lights are counted per tile, not per pixel. Each tile puts the count of lights in there and where in the global index list they lie.

Shadows

Point shadows are done with a texturecube in a geometry shader, like the dynamic cube map.

Directional shadows are also supported.



Particles

The text bubble is a particle expanded into a billboarded quad.

The fire and smoke uses the same system but different textures.

The rain is just white lines.

The leaves is like the rain but at a custom location, not just following the camera. And it’s based on quads not lines.

Cel

The models with edges are rendered completely black in a pre pass, a little bit bigger along their normals, the winding order is swapped, when added together with the color that is clamped between certain values they look toon like.

Parser

Textures can be imported with a text file, telling what name of the model and what textures it has.

Similarly, it would be possible to implement data for instances or bounds.

FBX exporter

Using the FBX SDK and a GUI library the exporter was built in cpp.

It takes in an fbx file and transfers the relevant information over into a binary custom format just named .x.

The options are radiobuttons that are either yes or no.

So it collects what options are set through enum flags.

When it looks for lights it checks the axis system and converts if needed.

To get the direction it takes a forward vector(0,0,1 in DirectX and 0,0,-1 in OpenGl), and transforms it with a rotation matrix created from the rotation.

Translation vectors are converted to a different axis system as well if needed.

The exporter also looks through the file for meshes and materials.

If a mesh is found it also looks for morph targets.

There is a function that computes average normal for a smoothing effect, but normal, tangents and binormals can be found without it.

This function is also not compatible with morph targets.

The mesh loops through parents to get the global data if needed, local export is also supported.

To find indices it finds the vertices first then compares them, using a struct with an overloaded comparative function.

If vertices are similar the vertex data is not updated but an index list is updated with an indice to the vertex that was similar, otherwise the vertex is added as well as an updated indice.

Skeleton data is also found and mapped to the vertices if needed. Skeleton data really means transformation matrices mapped to an index and a weight.

There are offset matrices, one per bone/joint of the mesh.

Then there are keyframe matrices, one per bone/joint in the mesh times the amount of keyframes.

Custom binary file format Importer

To read the custom data, a static library is built, it takes the information and transfers it into structs that can be read by the engine.

Things To try and Implement in the Future

- Volumetric lights and clouds.
- Portal Culling and making binary tree's working properly.
- Distance fog would be fun.
- Culling OBB bounds would be interesting as well as culling dynamic bounds. Or having multiple bounds on the same model.
- More complex cel shading would also be interesting.
- Picking
- Motion Blur
- It's possible to link the engine to a modeling application like Maya through plugins, this can be used to for example play around with a camera inside the modeling application but view a scene in the engine. This has been done before in a previous engine.
- Water

Things To try and change

Animation time values should be in doubles and not floats.

Different instances should have different transform options as in some can be static while others dynamic.