700106 and 700120 Final Lab Book

Scott White – 202214904

700120 C++ Final Lab Book

Class Description

Cactus

Holds the values needed to create instances of the cactus. Provide update for burning and growing.

Camel

Holds the values needed to create instances of camel. This was created so I don't have to recreate a mesh using the camel obj every time and can use this instead.

Camera

This allows a camera to be created by holding the values needed to create a view and projection matrix and means multiple cameras can be created in the scene.

CameraManager

This holds all the cameras in the scene. It allows you to switch to different active cameras and update those active cameras.

Candle

Holds the values needed to create instances of candle. This was created so I don't have to recreate a mesh using the candle obj every time and can use this instead.

ConfigLoader

This is used to load in the position values from a file, so that they can be used within the scene. Meaning you can update object position without having to change the code itself.

Cube.

This allows for the procedural creation of a cube. It inherits from shape, meaning that it can then be rendered to the scene using things which it inherits from shape. This means you can create a cube whenever it is needed.

Cubemap

This is used to create a cubemap image, and creates all the vulkan resources needed for that so that it can be used for a skybox.

Cylinder

This allows for the procedural creation of a cylinder. It inherits from shape, meaning that it can then be rendered to the scene using things which it inherits from shape. This means you can create a cylinder when needed.

GlobeScene

This manages the objects which are created in the globe screen, by storing them, as well as containing the logic for things like rain, and day night cycles, this means there is a central location for the scene.

GraphicsObject

This is used to store positions for anything that renders on the screen, and can be inherited from to do that.

GraphicsPipelineBuilder

This allows for basic pipelines to be created such as the pipeline for gouraud and phong, by using the builder patter to create instead of writing out all the code needed everytime.

Image

This provides utilities for creating Vkimages, this is used for things like depth.

InputManager

This is static so that you don’t create an instance of it and allows you to check and handle input from the user using glfw.

IWorldObject

This is used to store the drawing, and updates for the prefab classes.

Light

This is used to store the variables relating to the different types of light like, directional and point.

LightingSystem

This is used to store the vkbuffers and memory that is needed for the light

Material

This is used to store the texture that will be used for a shape so that they can have unique textures.

Mesh

This is used to create a mesh using the model loader, so that you can render obj files.

ObjLoader

This is used to load in files in the obj format.

Particle

This is used to define information about a particle such as the velocoty and position.

ParticleSystem

This is used to create the different particle system so that you can spawn them in bursts.

RenderContext

This contains the things needed for rendering like the device and physical device.

Rock

This is used for creating different instances of the rock model.

Shape

This is what all the renderable shapes inherit from, and it contains the methods needed to render the shape, and contains the variables needed for that.

Sphere

This allows for a procedural sphere to be created.

Texture

This allows for a texture image to be created, by providing a file type, and then can be used to pass the sampler and image needed for unique textures.

TextureManager

This allows for the adding of textures, and then selecting textures using a name, so that they don’t have to be redefined.

Vertex

This contains the structure for vertex so that it has position

Class Diagrams

Interaction Diagrams

Design Critique

For the design I would have liked to change the way that I create Vulkan pipelines, this is because the builder pipeline was useful for more basic pipelines, but for more complicated pipelines like the skybox, I still had to create pipelines in the old way, meaning there was a lot of duplicated code.

Another issue with my design, was that I had a graphics object which was inherited from by light, and shape, but for light I didn't find this to be necessary, as all it was using was the position variable, so it would've been better if only shape inherits from, it and light declares position in itself. Another issue with this inheritance was that for shape I had all the functionality for the draw and rendering within that class itself, instead of making it virtual, I also had things like vertices stored within the shape section of the class, but it would be better for that to be defined within the child class instead.

Another issue with my design, was that I created classes for image, and then I also created a class for texture, but a texture is a type of image, so it would have been better to make image have functionality for dealing with different types of images.

700106 Real Time Graphics Final Lab Book

Algorithms

For geometry representation, I used local vertex and index buffers for the shape classes and then created them using that instead of a global one with different offsets. This was good for keeping shapes separated however with large numbers of objects this could lead to performance issues as it would lead to more vertex and index buffers being created based on the number of shapes.

For lighting I used Phong and gouraud, phong was used for the smaller objects in the scene. This is because they don't have enough vertices to accurate calculate the lighting. Gouraud was used for the larger objects, however it wasn't very successful as everything using this appeared very flat shaded. For the lighting I had a CPU side struct and a GPU side struct within the shaders, this wasn't good as every time I changed the layout within the CPU side it would cause the GPU side to break as well.

Application and graphic connection

For application object there was a class for shape which held the vertices, indices, uniform buffers and descriptor sets which were needed to create the shape. These were then passed to the GPU side by creating the different buffers and then binding that to the GPU side using that.

Application and graphic updates

To update the graphic side of things I used a unifomr buffer which is defined within the descriptor set this then means it can be passed to the shader, this was used for things like model, view and projection of the camera, and time within the particles. These can then be updates on the CPU side, and then you update the buffer which is holding the uniform buffer to be aware of that update to the value.

Potential Extensions

I would like to add a way of adding pipelines easier. As well as making a way of allocating descriptor pool size easier, so it updated dynamically based on the number of objects.

Additional Features