Information for the examiners

For us both it is our first time using the Vulkan API so we had to learn it from scratch.

For the base of the renderer we followed the resources provided on the site vulkan-tutorial and Vulkan youtube series by Brendan Galea. Our base renderer is heavily inspired by the Vulkan youtube series from which we learnt a lot and as a result we now have a basic grasp of the vulkan API hierarchy and how the structures and classes fit together. This series was also included in the TUWEL page of the course. We felt like the tutorer created a really nice class hierarchy upon which with our little experience in Vulkan we could not improve.

Controls

WASD - Camera movement

Arrow Keys - Camera rotation

Development Status

We are using Vulkan without any framework, as we described in the Submission 1: Task Description. We also needed to include an additional library: the stb library to actually load in the textures from the hard drive. This library was not initially included in the Submission 1.

- We tested the code on NVIDIA GTX 1660 and NVIDIA Quadro T2000 GPUs.
- Right now most of the core Vulkan part is implemented.
- We implemented a `render system` that users can edit the rendering pipeline configurations as they want. Right now we are using a default pipeline configured for solid, opaque, double-sided triangles with depth test enabled, and viewport size is adjustable.
- We implemented a `camera` class which adjusts the projection matrices. And it controls the camera movement along with a input management class.
- We implemented a `game object` class which has a transform component inside, and calculates the transformation calculations.
- We are using a left handed and -y up coordinate system.
- We implemented a `mesh` class that holds the vertex, index buffers also an array of Textures. A mesh object can hold multiple texture objects to support multiple textures.
- We implemented a `model` class that holds the model. The model class consists of an array of `mesh` objects. The models are dynamically passed onto the shader, only the path to the file is needed.

- We are using `assimp` as object loader.
- We can assing textures to objects, and it can be rendered.
- We implemented a `descriptor` class that generates and holds the descriptor pool and also the descriptor set layout and the descriptor set. Only support for a single descriptor set is impleneted currently.
- We implemented a `lightmanager` class that handles the position and instensity values of a number of `point` lights and one `directional` light.
- We implemented `diffuse shading` for now.
- No complext effect is implemented yet.