August van Casteren - 5905818
Shreyes Jishnu Suchindran - 7835711

Additional information for late submission:

We added a very simple top-level BVH builder, which builds a BVH over the scene's objects. This allows us to render extreme scenes at 30+ FPS on my machine, without using RTX.

Additional controls:
1, 2, 3, 4, 5, 6 keys to load respective scenes.
Scenes 1, 2 and 5 show off performance with insane amounts of dragons.
Scenes 3 and 4 show off specular objects.
Scene 6 shows single-object performance with 250k triangles.
Scene 3 loads by default.
Scenes 3 is located in GPURayTracer.cs at line 223

Further information

For the 3rd practical, we implemented a Whitted-style ray tracer on the GPU, using glsl compute shaders with OpenTK (a C# library with bindings for OpenGL).

Our renderer uses the wavefront approach as described in Aila, Laine, and Karras' paper. The first hit shader uses the persistent while-while algorithm with speculative traversal, as described in Aila and Laine's "Understanding the Efficiency of Ray Traversal on GPUs" paper.

The shader also has an optional parameter which makes it do an early exit upon finding any intersection at all, making it usable as any hit shader.

As both wavefront and while-while with speculative traversal are methods created for performance, we felt it was fitting to do some extra work to achieve realtime raytracing. For this purpose, we:

- Changed our BVH construction to optimize our first hit shader even further.
- Researched GPU architecture, and how to play into it.
- Did low level optimizations where possible. (Guided by the earlier mentioned papers, and other NVIDIA resources)

Controls:
WASD, shift and spacebar to move.
Drag the screen using the mouse to rotate the camera.
Hold J to increase movement speed exponentially, hold K to decrease it.
Press Y to swap to the outdated CPU ray tracer
Press + and - to add or remove samples for multi-sample anti-aliasing (1, 4, 9 samples).
Runs out of memory (suspectedly) on full screen with many samples though.

Functionality:

- The program prints some performance information every single frame (measured with the C# Stopwatch class). This is the average of all the frames since starting the program.
- Mesh loading using a library
- Instancing, using the same BVH for multiple instances of the same object, by transforming the ray
- Diffuse and reflective materials
- Normal interpolation for smoother looking surfaces

Most of the work was done together with discussion and screen sharing, and August did a lot of optimization alone (mostly for fun)