Personal Report of Max Oesterle
Project: Doomed

Proposal

Team members are Björn Ehrlinspiel and Max Oesterle. The game idea is to develop a 3D-Maze-Escape game: You spawn in the maze and must find an exit. There might be multiple stages which you have to reach in a certain time on the way. How complex the story of the game will be will depend on the time left at the end of the fourth milestone and on if we focus more on the rendering or on the game mechanics. The setting will most likely be space themed.

The game uses portals. Portals break the Euclidean space and can bring you to a different part of the maze. The world (the maze) will consist of many cells which are connected only through portals. Therefore, we will be able to arrange and connect them at will. Portals shall support light transport and shadows; they are supposed to look like normal windows (sometimes maybe with some effect on it). We try to create a tense, realistic atmosphere of a maze through ambient occlusion approximation and relief mapping.

M = Max Oesterle B = Björn Ehrlinspiel

Milestone 1: Basics

- setup and open window (M)
- phong shading (M)
- load objects and textures (M)
- wireframe rendering (B)
- imgui menu (B)
- generate/create simple world (later a node in graph) (B)

Milestone 2: World

- camera movement using WASD and Mouse (B)
- generate/create simple graph-based worlds, hierarchical data structures for nodes, e.g. BVH (B and M for designing boundaries between nodes which are similar to portals)
- multithreading, game loop (M)
- deferred shading approach (M)
- envmap (M)

Milestone 3: Game

- Implement portals extending the rendering approach of Milestone 2. Portals shall behave like windows (regarding shadows). (M)
- normal, relief mapping (respectively steep parallax occlusion mapping?) (M)
- game state (timer, visible map), entities (B)
- simple game logic (deadline to exit maze) and story (B)

Milestone 4: Fancyness

- AO (optional many lights, see below) (M)
- Add different lights in maze and implement a dynamic behaviour of the lights (e.g. turn on when player passes the first time, adjust intensity according game mechanics like time left) (B)
- Add multiple stages until exit is reached to game logic (B)

Optional:

- many lights (M)
- day/night cycle (B)
- Additional game logic, story (M, B)
- Addiitonal objects to interact with (M, B)
- HDR, Bloom (M)
- New puzzle-like scenes (B)

Libraries / APIs:

- IMGui
- OpenGL
- glad
- glm
- Assimp

Tutorials

- https://www.glfw.org/docs/3.0/quick.html
- http://www.opengl-tutorial.org/beginners-tutorials/tutorial-1-opening-a-window/ and following
- https://learnopengl.com/ -https://developer-blog.net/professionelles-loggen-unter-c/

Papers

- http://www.cemyuksel.com/research/lgh/real-time_rendering_with_lgh_i3d2019.pdf

Resources

- open source object models and textures

Personal Responsibilities for Max Oesterle

- Setup (OpenGL, glfw…)
- Rendering
- Model and Texture Loading
- Normal and Bump Mapping or Sophisticated Relief Mapping
- Environment Map
- Portal Implementation
- Indirect Light Approximation

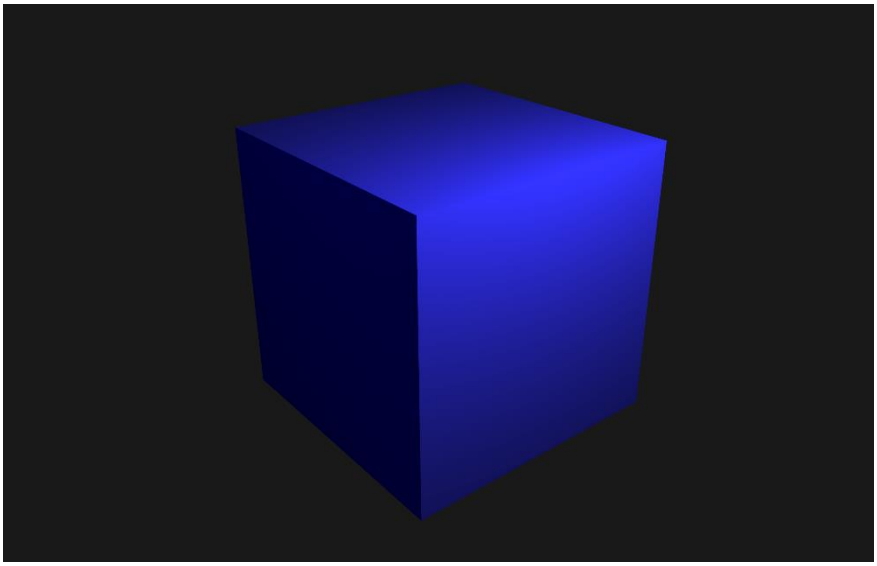
Milestone 1

Task 1: Setup

We decided to use cmake for compatibility but will mainly develop on Windows. Unfortunately, the Visual Studio support for cmake-generated projects seems to be rather poor but it works sufficiently. Although I never have setup an OpenGL project before, many tutorials and extensive documentation made it easy (but time consuming). I had to learn all the basics for glad and glfw and made some mistakes which have cost me a lot of time since debugging with glfw and glad seems to be mostly trial and error. It is easy to get the order of function calls wrong or break something which is not so well documented, e.g. by passing an object with the wrong C++ modifiers leading to generic, little helpful error messages.

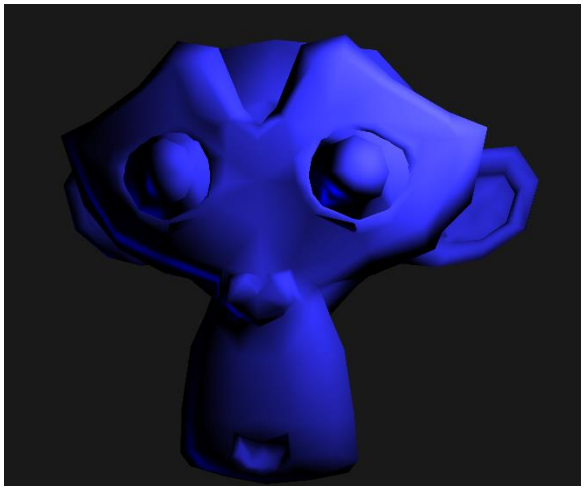
Task 2: Phong Shading

Straightforward, I remembered most of this from the computer graphics lecture. During this stage, we defined the objects which we used for debugging and the light conditions in code. The main difficulty of this task was to get the coordinate transformations right so that the camera is pointed at the object and the light is nearby. We noticed that a camera controller is more important right from the beginning on than we originally have thought. I used RenderDoc for the debugging of transformations because I could not pan around.

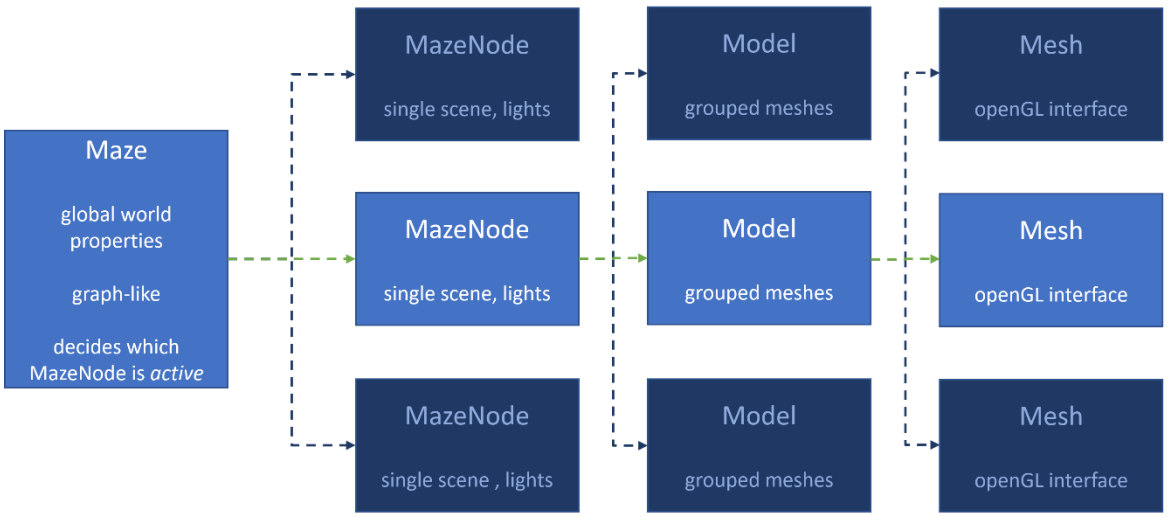


Task 3: Object and Texture Loading

By far the most complicated part of the milestone for me. First, I implemented a loader for very simple models without hierarchies and multiple meshes. Once it worked, I added texture loading. I also added some logical world layers as C++ classes so that the world and its parts are more organized.

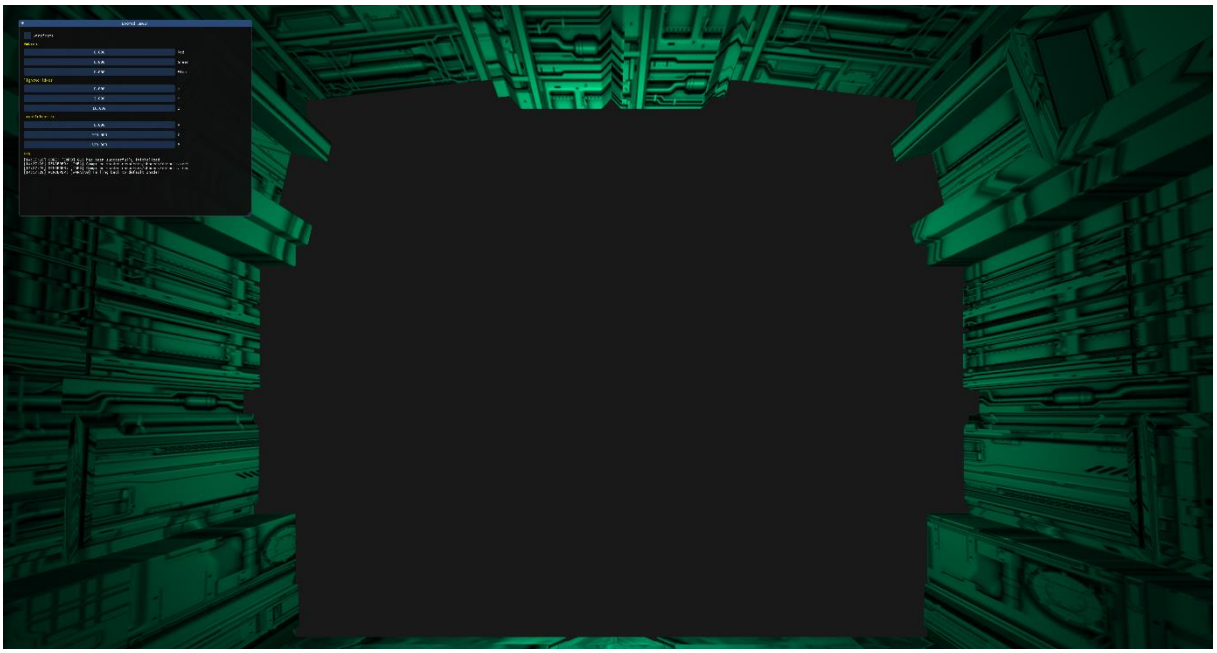


Then I noticed that the loader was not able to load at least somehow complicated models, so I extended it with the implementation of hierarchies and transformations between them. I also noticed that I mistakenly have put all the geometry into one mesh. In the process of fixing the loader I realized that I should rework the complete layers of the world structure. It now implements a strict layer design where each layer only has dependencies to the layer beneath:



The maze represents a collection of MazeNodes. MazeNodes are scenes which are totally independent of each other and are only connected by portals. This will allow us to create arbitrary maze layouts which do not follow the rules of a Euclidean space. The graph-like representation will also allow an easy calculation of distance in the maze which can be used in many creative ways, e.g. for adaptive lighting dependent on the distance to the exit. The MazeNode object is a classical scene with a set of models and lights. A model is a set of meshes with same properties, e.g. the same model matrix. The Mesh object is our interface to OpenGL.

Our result after Björn's work on the scene is shown in the following picture. We plan to assemble scenes from a set of objects like the one displayed in the scene.



Comments and Notes

- The camera implementation is part of our milestone two, but it would have been better to do it at the beginning. It is really annoying to try to figure out correct coordinates to put camera, light and objects into the correct position. In general, being able to set basic properties of the world like light position and intensity in a GUI at runtime helps.
- Proper logging is worth a lot since many errors don't lead to crashes but break something else.
- The optional CI will be postponed until we have nothing else to do, because vcpkg does not work well in the runner environment for some reason and there are many other annoying problems. We also do not have access to runners on a server and estimate the importance of CI for our project as low. We do code reviews in gitlab pull requests instead.