Personal Report of Max Oesterle
Project: Doomed

Proposal

The game idea is a 3D-Maze-Escape game. The maze has portals which break the Euclidean space and can bring you to a different part of the maze or create impossible rooms.
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 ideally shall support light transport and shadows; they are supposed to look like normal windows (sometimes maybe with some effect on it, e.g. lens or flickering).

M = Max Oesterle, B = Björn Ehrlinspiel (left the project after milestone 1)

Milestone 1: Setup

- setup and open window (M)
- phong shading (M)
- load objects and textures (M)
- wireframe rendering (B)
- imgui menu (B)
- create debug scene (B)

Milestone 2: Basics and Rendering

- camera movement using WASD and Mouse
- game loop
- deferred shading approach
- shadows

Milestone 3: Portals 1

- extend world to include multiple maze nodes
- portals world mechanics and teleportation
- portals rendering 1: render portals as solid objects correctly
- portals rendering 2: render scene behind portals

Milestone 4: Portals 2

- portals rendering 3: render portals which are visible in portals (up to a certain extend)
- create interesting world and extend the builder functionality
- add gameplay, e.g. interact with portals (toggle on/off, change destination)

Optional:

- shadows and light through portals
- portal effects (e.g. lens or flickering)
- texture normal/bump/relief mapping
- HDR, bloom
- dynamic light behavior (e.g. turn on when player passes the first time or adjust intensity according to distance to exit)

Libraries / APIs:

- IMGui
- OpenGL
- glad
- glfw3
- glm
- assimp
- stb
- spdlog

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/
- http://ogldev.atspace.co.uk/

Resources

- open source object models and textures

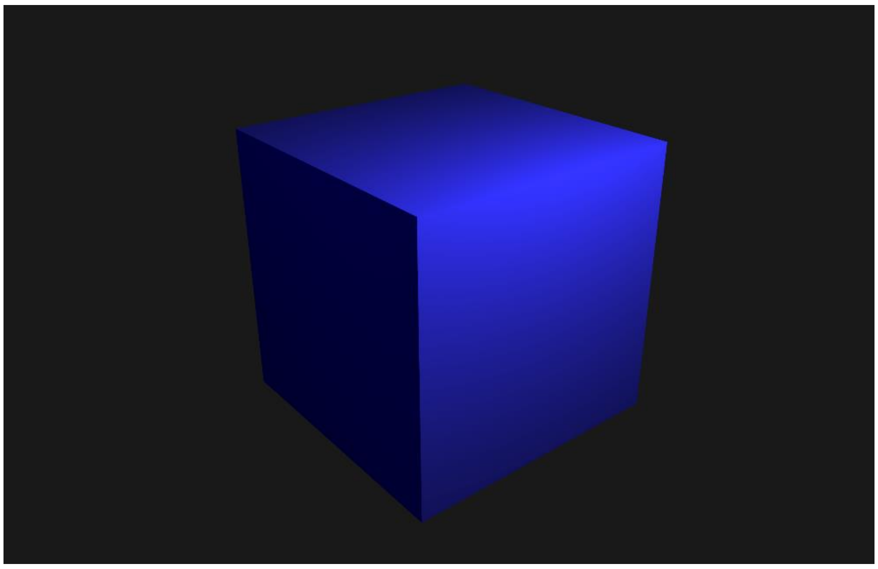
Milestone 1

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.

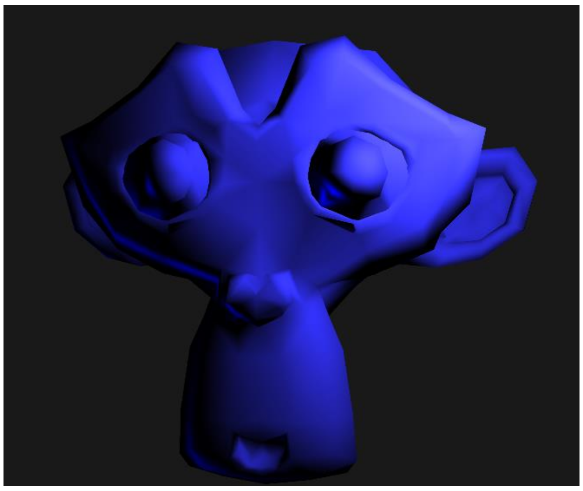
Task 2: Phong Shading

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.

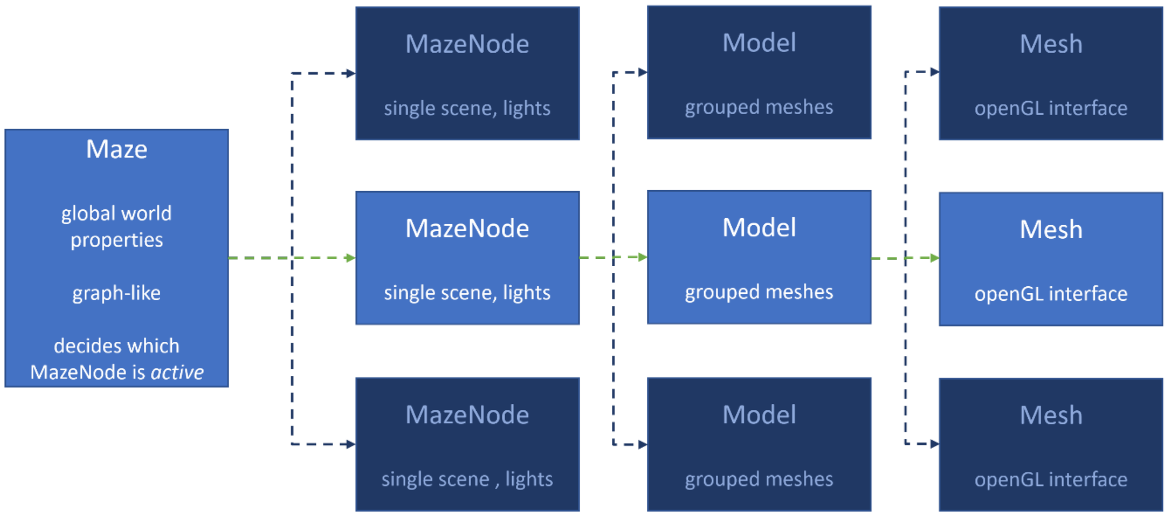


Task 3: Object and Texture Loading

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.

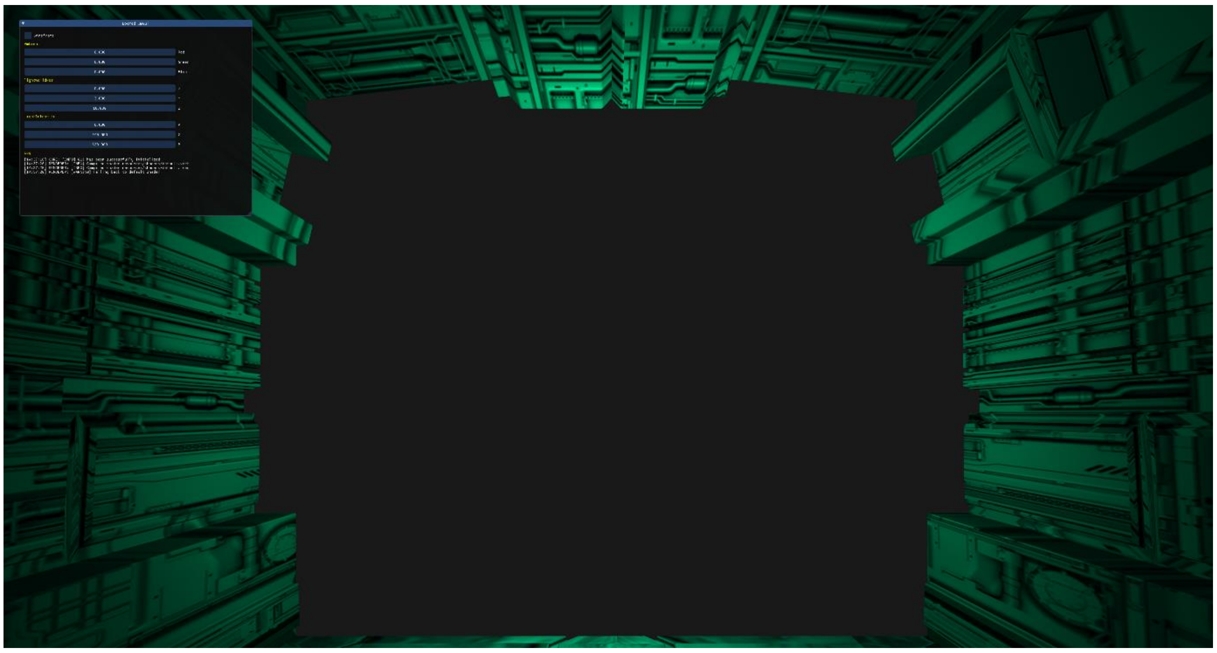


Then I noticed that the loader was not able to load at least somehow complicated models, so I extended it with the implementation of hierarchies and transformations between them. I also noticed that I mistakenly have put all the geometry into one mesh. In the process of fixing the loader I realized that 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.

Milestone 2

Milestone 2 was all about improving the rendering to be able to focus on the portals and the game itself in the second half of the project.

Task 1: Camera Movement (dropped the Game Loop)

The moving camera would make the debugging much easier and uncovered some rendering bugs once it was implemented. I went for an FPS-style camera, that means catching and disabling the cursor to be able to control the view with the mouse and the position with WASD. One effect is that you are no longer able to click on ImGui buttons. If there will be a need for this kind of interaction it will be implemented using keys like games usually do.

Until now, the camera was controlled by setting the view and projection matrix. For the moving camera it is much more feasible to set yaw and pitch angles, so I changed the interface to accept those and position changes. The actual deltas are calculated in an input dispatcher class where it is also adjusted for time since the last calculation and desired movement speed. Because this worked well, I decided to not implement an own game loop which would then be executed with a controlled frequency (in contrast to the render loop which is executed as frequently as possible). Finally, I clamped the pitch angle to +/- 89° to avoid the 90°point where Euler angles lead to confusing behavior.

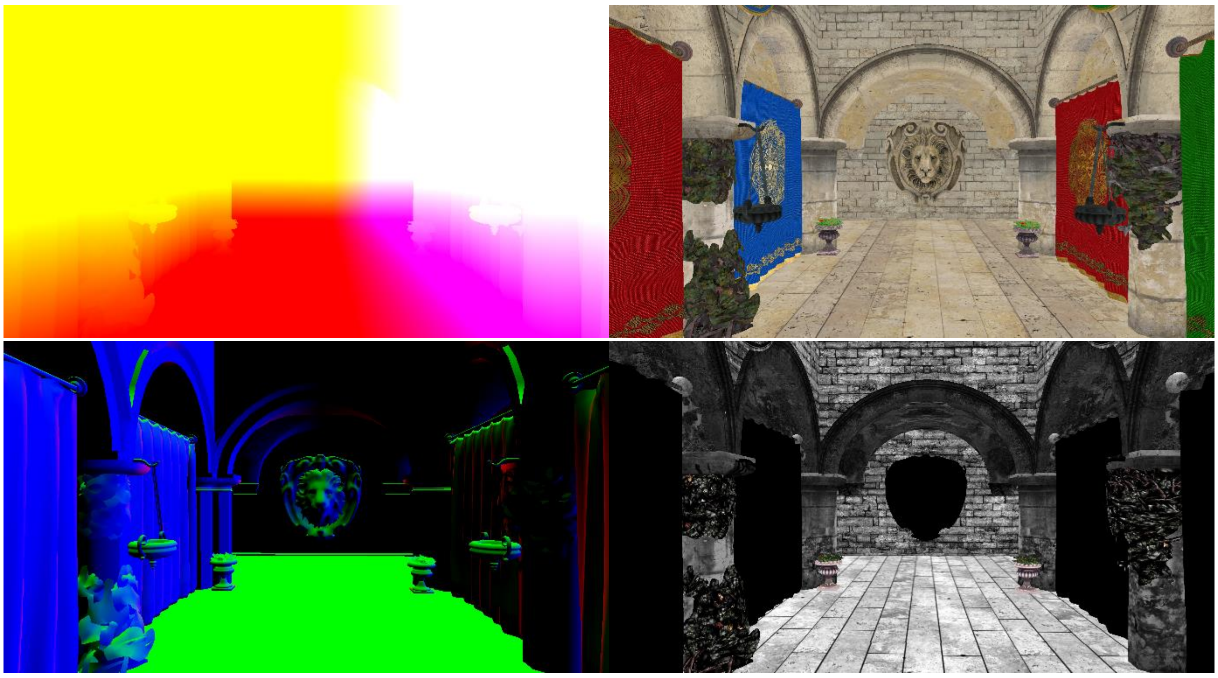
When I launched the game on another setup to test the camera speed adjustment on higher FPS, I encountered a glitch where the camera was constantly rotating counterclockwise. After a small investigation I found out that this was not a bug in my game but more a common glitch in games which occurs when a controller is connected to the PC.

Task 2: Deferred Shading

Deferred shading was the largest task of the milestone. I decided to implement it because the portals can lead to a larger number of lights and geometry in the view frustum. Also, a rough idea I had for the portal rendering originates from the light pass of deferred shading.

Geometry Pass

My gbuffer consist of four components: position, normals, diffuse color, and specular color. First, I wanted to render to the gbuffer and dump it on the screen by just copying it into the default framebuffer. I faced a problem where everything seemed to be implemented correctly but I ended up with a black screen. After extensive debugging I found out that my method just did not work with multisampling, so I set the GLFW_SAMPES window hint from 8 to 0 and it worked. This was one of the examples where a seemingly small and unimportant decision in the past lead to long and frustrating debugging in the future. For the next step I changed the oversampling back to 8. An example of a gbuffer is shown below. Please note, that the aliasing only occurs because of the downscaling of the screenshots, it is fine in the game.



Light Pass

For the light pass I used two additional models: A sphere and a quad. The quad is a 2-dimensional plane with uv-coordinates spanning the entire plane. It is used for directional light shading; the sphere is used as the bounding sphere of the lights for point light shading.

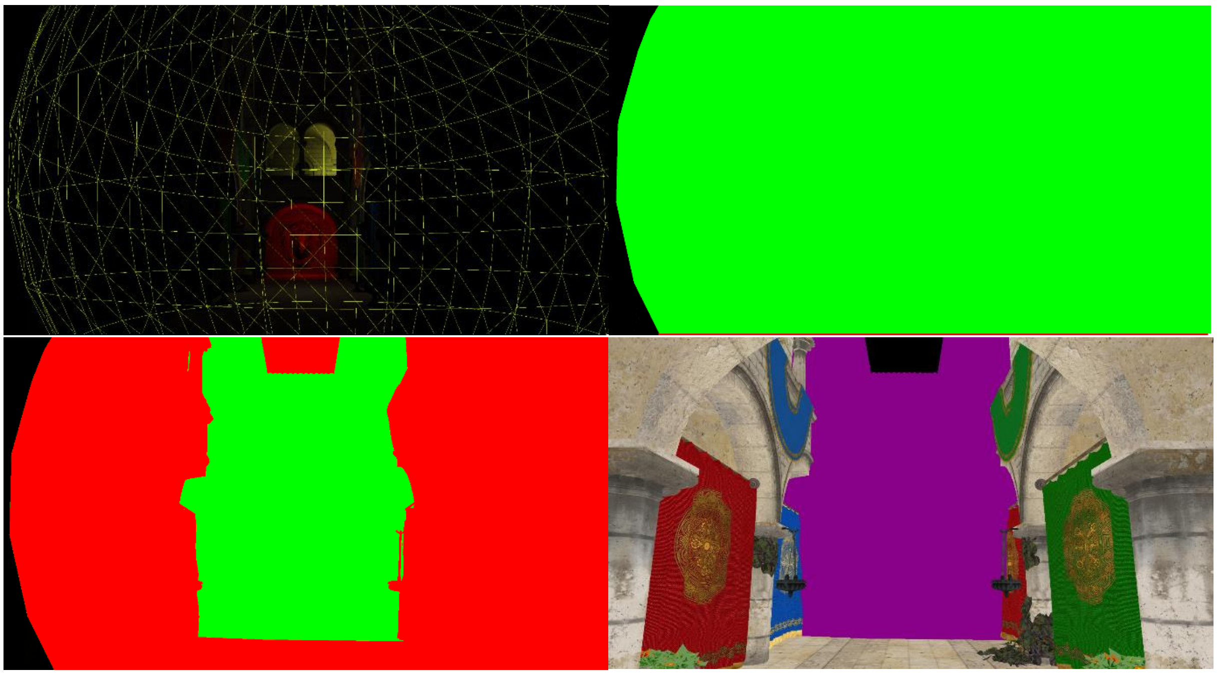
Point light shading works with two small passes: stencil and light. For the light pass, the sphere model is transformed in a way that it now acts as a bounding sphere of the light: No point outside of the projection of the sphere will get a contribution from that light. But there are two problems: Fragments, which are behind the sphere are still considered because they are in the projection of the sphere. Also, with back face culling enabled the light disappears when entering the bounding sphere with the camera. With back face culling disabled fragments get double contribution.

To solve those problems, I add a stencil pass before the light pass which sets the stencil buffer to a positive value if the fragment is within the sphere. The stencil pass increases (decreases/does not change) the stencil value if the fragment is a back face (fragment is a front face/depth test succeeds).

```
glStencilOpSeparate(GL_BACK, GL_KEEP, GL_INCR_WRAP, GL_KEEP);
glStencilOpSeparate(GL_FRONT, GL_KEEP, GL_DECR_WRAP, GL_KEEP);
```

Another option would be to discard the fragments which are behind the sphere manually in the fragment shader.

The following shows the bounding sphere of the yellow light in the background. Because the mesh is drawn over everything else it seems to be close, but it is centered around the correct point light. Top right shows the projection of the sphere onto the screen which is not affected by the depth test at all, bottom left shows the result of the stencil test. Bottom right gives an overview which parts of the scene get a contribution from the yellow light.

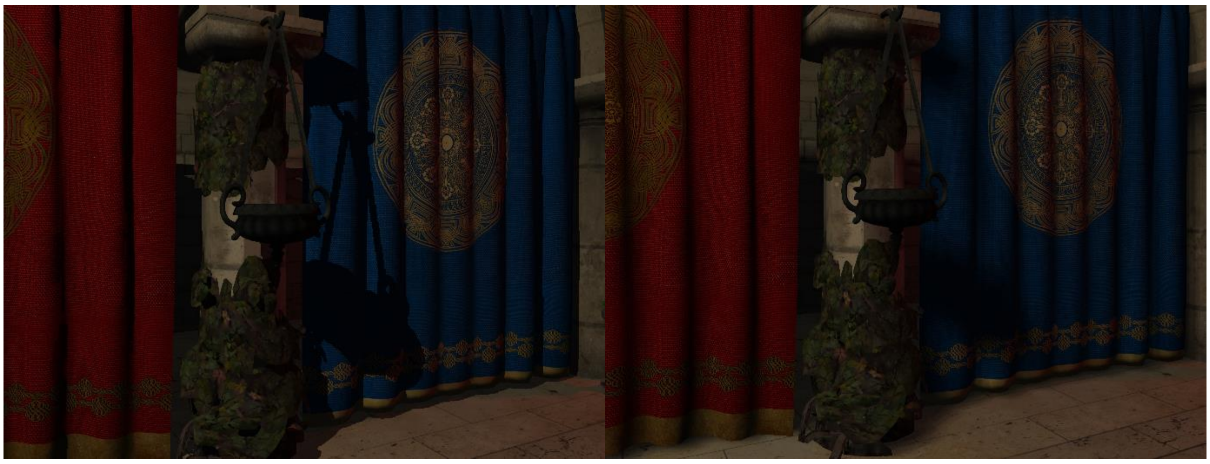


The directional light pass works by supplying the quad as a screen filling geometry, the fragment shader will be called for all pixels.

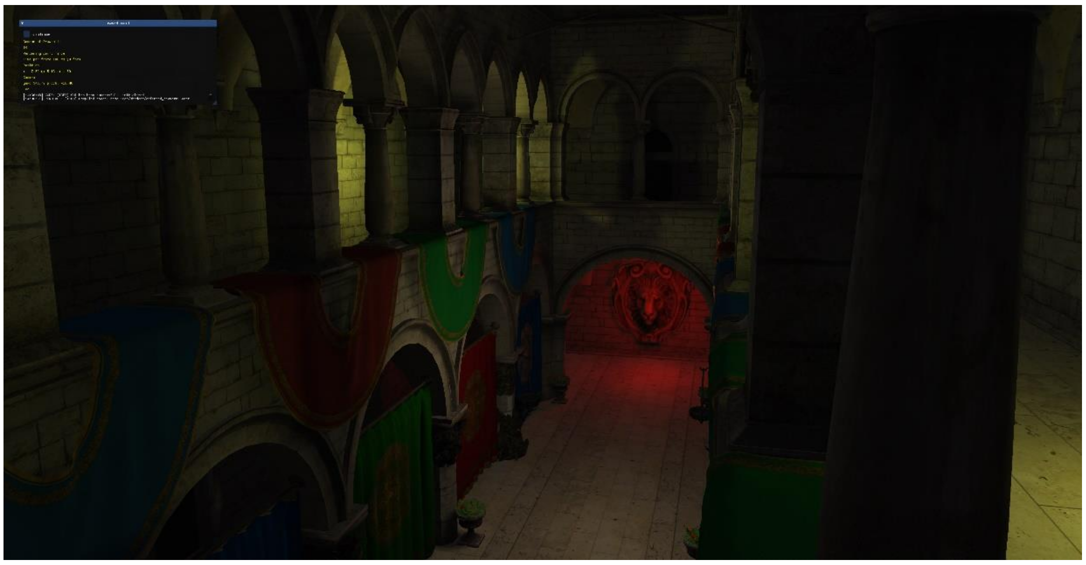
Task 3: Shadows

I implemented the point light shadows using cube maps with 1024x1024 faces. Depending on if the light is flagged as dynamic or not, the map is drawn once or per frame. Of course, you can also disable shadows for a light.

The cube map is drawn in one pass per light using the geometry shader and the `gl_Layer` variable. It is later supplied as an additional texture to the fragment shader of the light pass. The shadows which you get as a result of plain shadow mapping look not very good (left). To improve the shadows, I use percentage closer filtering which works sufficiently well (right). The aliasing on the curtains is not present in the scene, it occurs when scaling down the screenshots for this report.



To make use of the dynamic point light shadows and to test their influence on the performance I made it possible to add keyframes to the point lights.
The final result after this milestone (4 point lights, the two yellow lights are moving):



Optional Task: Normal Mapping

I decided to implement normal mapping just because it is relatively easy to implement and my sponza model included normal maps. It generally works, but the texture coordinates seem to be broken in the model. Because this seems to be time consuming to fix, I decided not to add it to the master branch for now. If I have normal maps available later with the real models of the game, I can still add it. I also think that plain normal mapping looks a bit unrealistic, especially with specular reflections.