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 world for the demonstration and extend the builder functionality

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

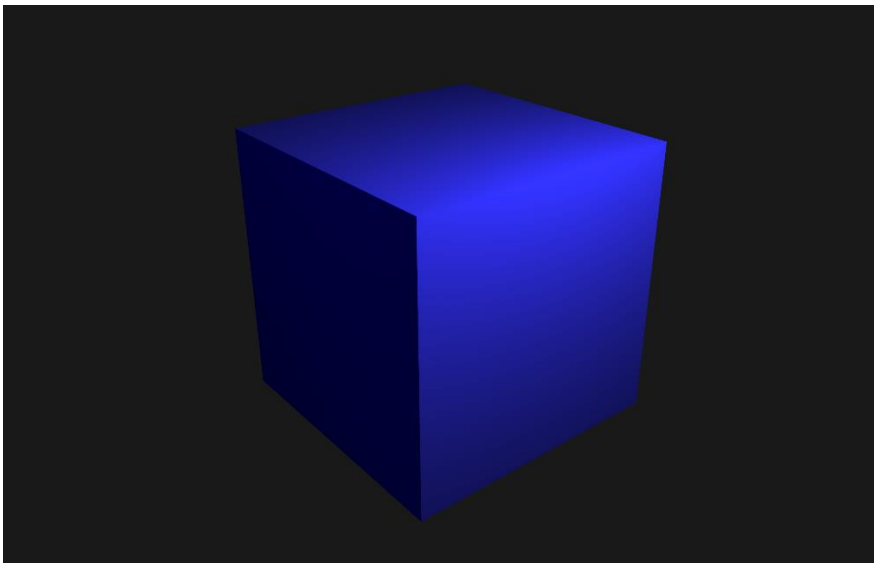
Milestone 1

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.

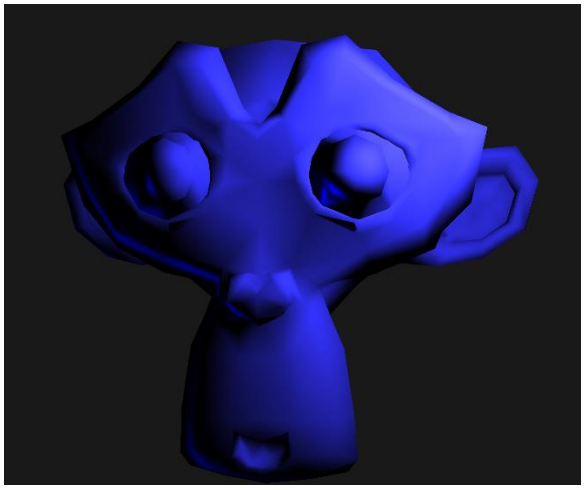
Task 2: Phong Shading

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.

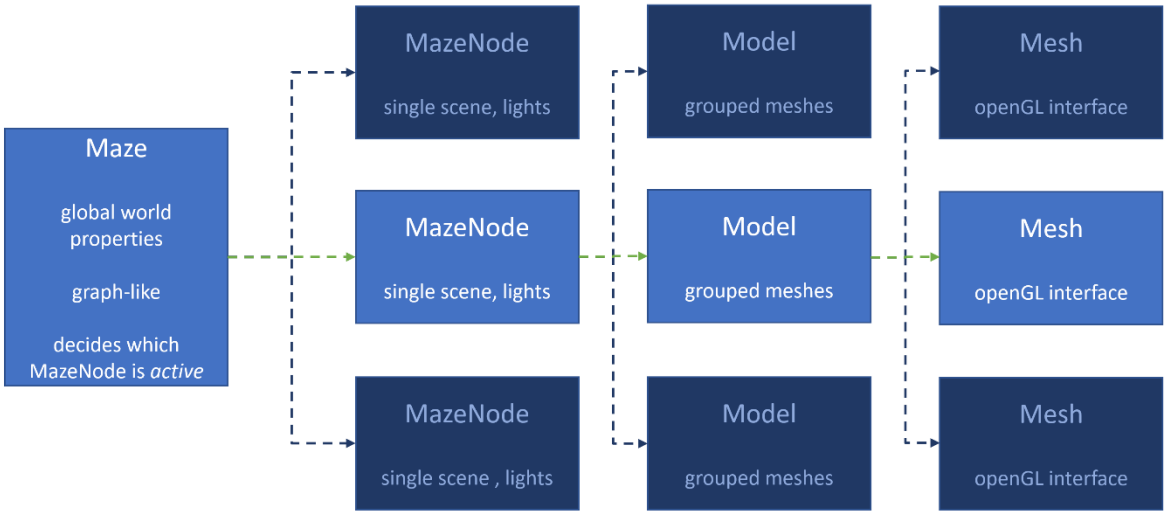


Task 3: Object and Texture Loading

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.

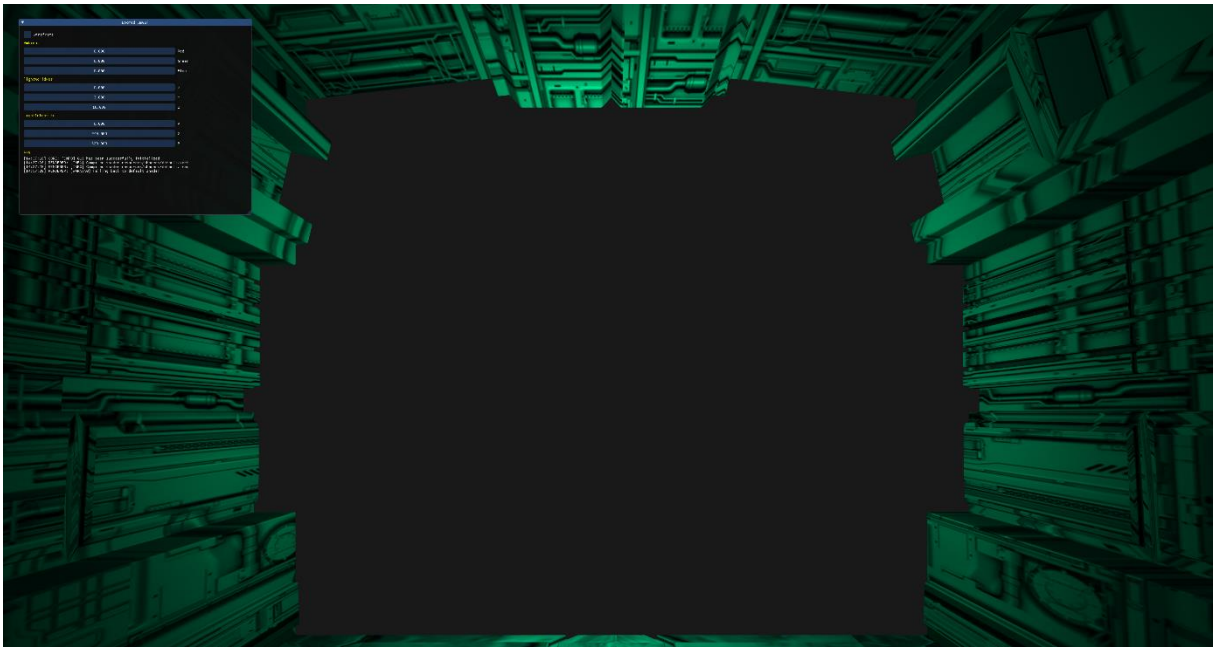


Then I noticed that the loader was not able to load at least somehow complicated models, so I extended it with the implementation of hierarchies and transformations between them. I also noticed that I mistakenly have put all the geometry into one mesh. In the process of fixing the loader I realized that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

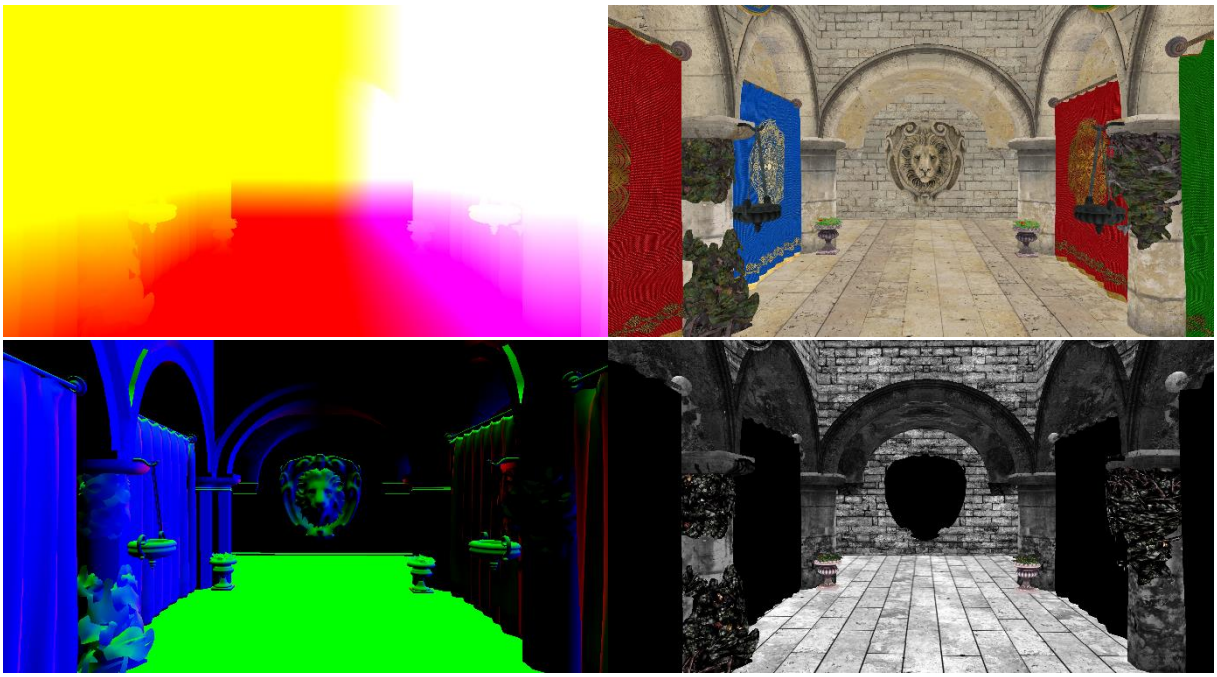
When I launched the game on another setup to test the camera speed adjustment on higher FPS, I encountered a glitch where the camera was constantly rotating counterclockwise. After a small investigation I found out that this was not a bug in my game but more a common glitch in games which occurs when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

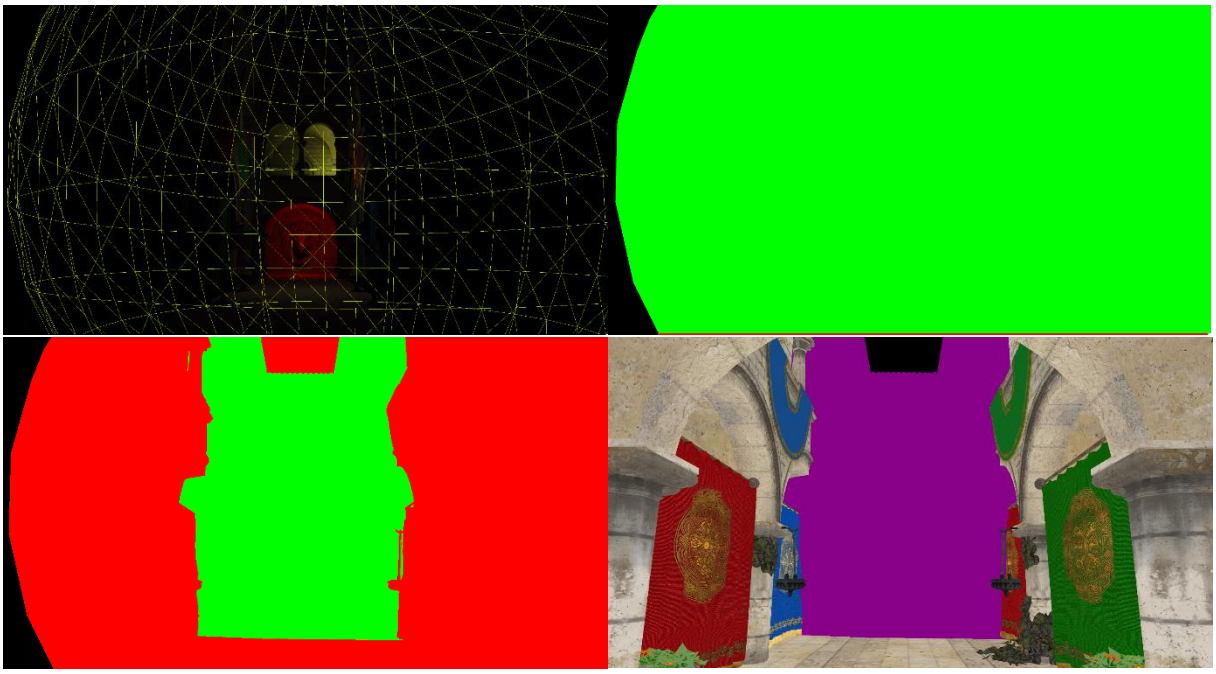
Point light shading works with two small passes: stencil and light. For the light pass, the sphere model is transformed in a way that it now acts as a bounding sphere of the light: No point outside of the projection of the sphere will get a contribution from that light. But there are two problems: Fragments, 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

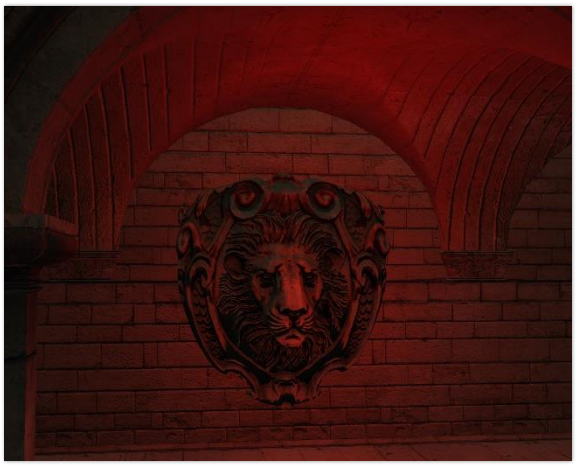
I implemented the point light shadows using cube maps with 1024x1024 faces. Depending on if the light is flagged as dynamic or not, the map is drawn once or per frame. Of course, you can also disable shadows for a light.

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.



Milestone 3

Milestone 3 started with setting things up for the portals. I rendered them as black surfaces at the beginning and concentrated on the teleportation which was more difficult than I had expected. The most difficult thing though turned out to be the view transformation for the virtual camera of the portals. I also started with the support of multiple portals (MS4) because I thought that it was not ideal to simplify it for now and then reimplement it completely in milestone 4.

Task 1: Extend World, Multiple Maze Nodes

This is a prerequisite for the teleportation. It was not difficult to support multiple maze nodes, but it needed a lot of attention because I ran into a lot of bugs. Most of them where due to wrong or missing updates of the player's current maze node. I also extended the renderers functionality to be able to render objects from multiple maze nodes. At the end, I added the framework and stubs for the portals.

Task 2: Teleportation

To enable teleportation, I needed to implement some collision detection for the portals. I decided that they should only work in wrong direction because this allows to use them as building blocks for more complex arrangement of portals or to just pair them up to get a two-sided portal.

For the collision detection I introduced a new coordinate system, centered at the portal's center point. Its base vectors are:

- The normal vector of the portal which is pointing into the direction of travel through the portal.
- The up vector (which is the same in the entire game).
- The vector, which is perpendicular to the first two vectors, following the right-hand-rule (width vector).

Using a transformation to the portal's coordinate system's base vectors, I express the player's position in relation to the portal. To trigger a teleportation, multiple conditions must be fulfilled:

- The width vector component must have a length smaller than half of the portal's width.
- The up vector component must have a length smaller than half of the portal's height.
- The normal vector component's signedness must change from negative to positive.

In practice, I changed the last point because a problem with the near clip plane occurred: The portal's surface was being clipped away before the teleportation was triggered when travelling through the portal. Therefore, the scene behind the portal was visible for a short amount of time and then the teleportation occurred. I added a margin to the normal vector component of the player's position, so that the teleportation happens earlier now: It is triggered when the player is moving towards the portal and is within the margin from the portal's surface.

A problem which I have not thought about beforehand was the change of the camera's direction after the teleportation. In general, the portal's target will not face into the same direction as the

portal's surface (the exit portal will not be parallel to the portal). If I would ignore this, the player's view direction would suddenly jump when moving through the portal.

In the simplest case, the player is traveling through the portal while looking straight forward into the portal's normal direction. Here, the teleportation must change the camera's direction to the direction of the exit portal's normal. If the camera is facing into a different direction, the change must be done in a similar way using the portal's base vectors:

1. Decompose the camera direction into the components of the portal's base vectors.
2. Exchange the vector base to the exit portal's base.
3. Calculate the linear combination of the exit portal's base vectors with the factors computed in 1.

Additional work was necessary to update the state of the world and the renderer correctly. Similar two the first task of this milestone, this needed quit a lot of bug fixing.

Task 3+4: Simple Portals (including significant milestone 4 work)

The portals were working at this point but there was no rendering of the scene behind the portal. I tried two approaches: The first one uses stencil and depth operation to mark the regions of the portal, the second one draws the portal's perspective from a virtual camera into a texture which is then drawn onto the portal. Both approaches are more difficult than expected. I spent more work with the first approach but especially for multiple portals, it might be better to use the second approach because the first one becomes immensely complicated.

Basic Stencil Approach

The current stencil approach, which is supposed to work for multiple portals too, consists of the following steps:

1. Draw scene normally to fill geometry and depth buffer with the geometry information of the maze node in which the player is.
2. Calculate a rendering order of the portals, so that portals which occlude other portals, are drawn earlier. I implemented a breadth-first search for this.
3. For each portal from front to back (the calculated rendering order), draw the portal in a stencil-only pass, setting the stencil value at the regions where the portal is to the portals maze node id. The stencil test is configured in a way that earlier drawings are overwritten. Therefore, portals which are drawn later will overwrite the stencil value of earlier draw calls. After this step, the stencil buffer will be filled with stencil values which determine the visible maze node at place on the screen.

Solving Problems

Now, one could just draw the multiple mazes with the according stencil test to get a result. But two problems have not been considered yet.

First Major Problem: The maze nodes different than the one the player is currently in must be drawn from a virtual camera positioned somewhere else. This causes other problems.

Getting the view transformation for the virtual camera right is very difficult but I had this problem in mind. The camera must be positioned in a way that the part of its view which is supposed to be visible through the portal must end up exactly where the portal is. First, I assumed that the virtual camera must be positioned at a fixed position behind the exit of a portal to get the desired perspective which shall be shown in the portal. But this is not the case. If the player sees the portal in the middle of the screen and turns right, the portal will move to the left side of the screen. The virtual camera must follow this movement. Otherwise, the visible part of the scene behind the portal would change. This can be done using the vector base transformations of Task 2, but it complicates the projection enormously. The texture approach for the portal rendering avoids this issue.

Another problem is that there might be geometry between the portal's exit and the virtual camera. This geometry must be hidden. I do this using an additional clip plane which OpenGL allows to configure using

```
glEnable(GL_CLIP_DISTANCE0);
```

The clip distance must then be calculated and exported in the vertex shader:

```
gl_ClipDistance[0] = dot(vec4(worldPosition, 1), nearClipPortalPlane);
```

Second Major Problem: If the portal is occluded by geometry which is not a portal itself, this will be ignored.

The occlusion of portals was something which I had not expected to be problematic, but it almost made this approach impossible. The problem is that the depth buffer at the regions where the portals are still holds the depth of the original maze node the player is in. When drawing the scene behind the portal, the depth test cannot be used. Geometry would be occluded by geometry of another maze node. Because I need the depth test, the depth buffer must be cleared at regions where portals are:

4. Reset the depth buffer by doing a depth only pass for all the portals.

Now the geometry pass for the portals can follow:

5. Geometry pass for each portal. Configure virtual camera transformations, stencil test, and additional clip plane for each portal and draw the geometry of the portal's maze node (depth and stencil testing enabled).

At this point, the geometry buffer is complete. It holds the correct information of the correct maze nodes at the correct place and has the matching stencil value (the node id) attached. Now the light pass can follow for each maze node using the node's light sources and the stencil test.

Note, that the stencil buffer has two uses: Portals and during the deferred light pass. Therefore, I only use the higher bits of the stencil buffer for the portals, the ones which are used during the light pass are not touched. This prohibits the use of increase and decrease stencil operations for the portals.

Outlook

Due to the mentioned problems, I consider using the texture approach in the future. I hoped to get a performance advantage by using the stencil approach, but I had to add a lot more operations to solve the above-mentioned problems. Therefore, this advantage seems to be gone and overshadowed by the immense complexity of the approach. The texture approach seems to be more feasible due to much easier transformations and no problems at all with chaotic depth buffers. A disadvantage is, that the light support of the portals seems to be difficult with the texture approach. It would need virtual lights and matching transformations in the same way, the virtual camera works. But since this is an optional task, I am willing to accept this.

Milestone 4

As mentioned in Milestone 3, I started experimenting with the texture approach. Reasons, why I finally decided to switch:

- Many problems appeared which I had not foreseen. Some problems seemed to be trivial, but they turned out to be complicated to solve. An example for this is the occlusion of portals behind normal geometry which lead to additional rendering passes. Other problems, which seemed more difficult to save, turned out to have interesting and elegant solutions. An example for this is the problem of clipping away geometry between portal and its virtual camera. I solved it using additional clip planes, an OpenGL feature I would have not learned about otherwise (see MS3 for details).
- The stencil portal approach was meant to save performance. The problems, which appeared, lead to solutions where I had to sacrifice performance. So, the performance advantage I hoped for was gone.
- I thought that the stencil approach was very close to be completed. We even discussed this matter in the meeting of milestone three. But at this point, it was “very close” for a long time and I was afraid of wasting even more time.
- I found out, that there are some working portal implementations using the texture approach and I found claims that this technique was used in the game “Portal” too.

At the end, I have almost lost the time and work of an entire milestone through following the stencil approach for too long. Looking back, this is a classic example of situations where you are not willing to throw away work which is already completed which leads to even more time being lost.

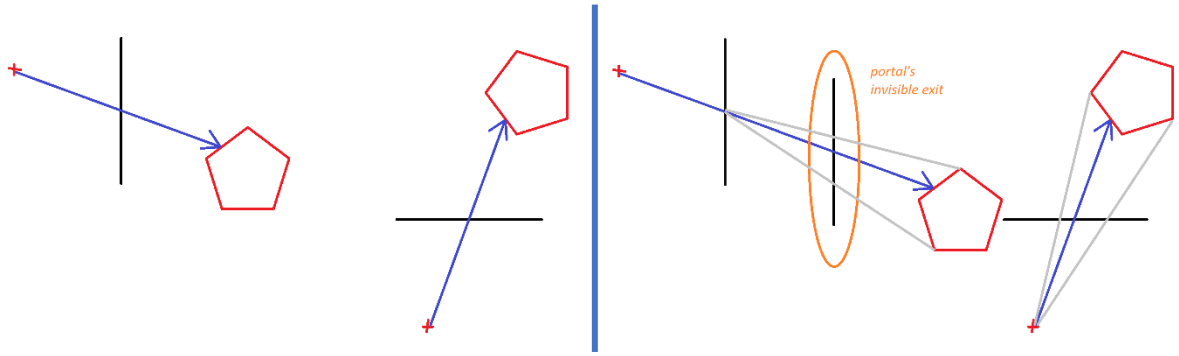
Nevertheless, I learned a lot about what can be done with OpenGL and most importantly, what cannot be done. Also, I learned a lot about hidden problems in rendering projects and debugging of shaders and renderers. I did not have any significant experience with rendering before (only the computer graphics lecture) and would now be able to plan such a project way better using this experience.

Task 1: Portal-to-Texture Rendering

I started the milestone by implementing the render-to-texture. First, I used the existing geometry buffer textures of the deferred rendering for this as a target. Later (during the next task), it turned out to be necessary that each portal has its own texture. I then applied the texture to the portal using a very similar shader to the directional light shader of the deferred rendering. The portals at this point looked very much like a wallpaper. To make it look like a window, I had to get the virtual camera transformation right. This included several steps:

- The distance between portal and the virtual camera which renders the portal's surface must respect the distance between the original camera (the player) and the portal's surface. It is not possible to change the field of view to get the correct projection, the camera must move backwards.
- The previous point introduces the already mentioned issue with geometry between the portal's exit surface and the virtual camera which occludes the view. This was solved in milestone 3.
- The virtual camera must pan and tilt like the original camera does. This has to happen in relation to the portal's exit direction.
- The projection was still incorrect which was surprising to me. The projection on the surface of the portal seemed to be to far away. It turned out, that the geometry which is shown in the portal is 50% more far away than it should.

The reason for the last point is that the virtual camera already is as far away from the geometry as the original camera should be away from it to render a realistic image. But the distance between original camera and the portal's surface adds 50% distance on top of this. The following illustrations show how I thought the situation would look like (left) and how it behaves in reality (right):



In black are the portals, on the left side of each image is the normal portal and on the right half of each image is the portals' exit surface. Therefore, on the right side, the red cross is always the virtual camera, on the left side the player's camera. On the ride side, the geometry is the actual geometry, on the left side it is virtual geometry, which is how it appears in the portal.

It is clearly visible that the projection is only correct when the player's camera moves very closely to the portal's surface. The right image shows that the distance between camera and the virtual geometry otherwise is greater than it should be.

The problem can be solved by scaling the texture up, so that the distance at which the geometry in the portal appears is smaller.

Task 2: Recursive Rendering of Portals

This appeared to seem the most difficult task of the entire project. Thanks to the new approach to textures, it is still easier than before.

To render portals which are visible within portals, a portal A, which is visible from a portal B, must be rendered first. Then it can be rendered like a normal object when rendering the scene for portal B.

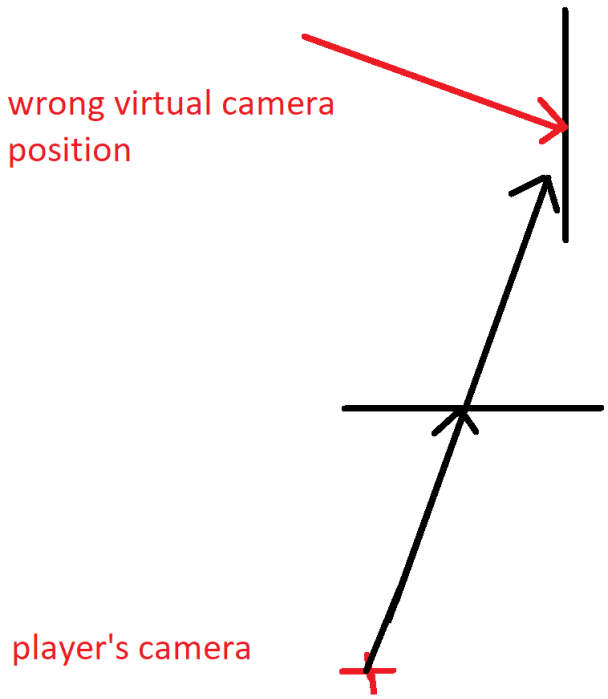
To achieve this, I implemented a modified breadth-first search. Then, I use the resulting breadth-first graph to order the portals for the rendering. The graph must not be a tree, nor a directed acyclic graph. The graph may contain cycles. In this case, a portal is visible from itself. The breadth-first search terminates only when the maximum depth is reached and ignores already scanned nodes.

Rendering portals which are visible in themselves is easily possible because I use deferred shading, which buffers all the shading information in a dedicated buffer and then writes to the portal's texture. Otherwise, I would have had the problem of reading and writing the same texture in one rendering pass.

Finally, I must chain the camera transformations for the portals. You cannot use the original virtual camera transformation for when a portal is not directly visible, but only visible within another portal.

The drawing on the right illustrates the problem. The upper portal is visible within the portal at the bottom. If I would apply the normal camera transformation, then the portal at the top would show the view from a camera positioned at the red arrow's base. This is clearly wrong; the correct viewpoint is from the player's camera.

The chained transformations are not completely correct at the time I am writing this. I might be able to find the mistake until the time of the meeting.



Task 3: Small World for the Demonstration

I wanted to create an own small world to demonstrate the intentions behind the portals: simulating a non-Euclidean geometry. I want to show the use of the portals to create impossible rooms in contrast to the use e.g., in the game “Portal”. I decided to build a simple world, because I did not have the time and resources to create the models and textures for a complicated world. The result is sufficient for the demonstrations and would also be sufficient for puzzle-style games. For this reason, I also implemented more builder functionalities to be able to create puzzle-like worlds more quickly.

Finally, I create a small portal demonstration using the sponza scene to be able to show that they also work well in a more complex world with many lights and dynamic shadows (see). The portal in this scene can be seen right in the middle, floating up in the air. The player’s camera is positioned at the upper gallery. The light next to the lion is red in the original scene, and green in the scene of the portal’s destination.