Cubemap Mirrors

Anton Nilsson antni004@student.liu.se
Morgan Nordberg morno368@student.liu.se

Introduction

The project goal was to create a 3D scene, where the "player" can walk around as a 3D model in a first-person-perspective; as well as to model mirrors that reflect the models in the world. Additional goals were to gradually improve the mirror implementation with various, gradually more advanced, techniques. Yet another additional goal was to make it possible to have multiple mirrors that can all see each other.

Background Information

Some basic datastructures needed for the overall discussion are defined here

Cubemap Texture

A cubemap consists of 6 textures that make up the 6 sides of the cube. There is built-in support for working with cubemap textures in various contexts directly in OpenGL.

FrameBufferObject

A FrameBufferObject (FBO) is an object in OpenGL which allows us to render the scene to a texture, instead of rendering to it the screen. It has its own texture and various buffers, such as a depth buffer. In our case it's useful for capturing the scene from various persepcives and positions before the final render.

Implementation Details

Here we will go into more detail on how various techniques were implemented as well as overall project structure.

Project Structure

The project structure contains all source code in src/main.cpp, all 3D models in models/, textures in textures/ and shader code in shader files in the shaders/ directory. The various dependencies are contained within the common/ directory.

Dependencies We used no additional dependencies besides those provided for

the lab assignments. We did however use OpenGL 4.6 in our shaders, because we could

Pre-rendered Cubemap

In OpenGL there is a built-in type for dealing with cubemap textures in a shader called "samplerCube". So in the shader you pass a uniform samplerCube, and a directional vector from the observers position to the mirrors surface, and then reflect the vector w.r.t the normal vector of the surface. The resulting vector is used for sampling color from the cubemap texture.

The problem with approach is that the cubemap can't account for changes in the enviorment and thus models in the scene wont be reflected. There is also issues with scale/perspective since conceptially one can think of a cubemap as an approximation of an infinitley large cube. In a small scene this will cause very apparent distortions in perspective in the mirror. The first problem is solved by using dynamic cubemaps, and the latter problem is solved by parallax correction. Both of this techniques are covered in later sections of the report.

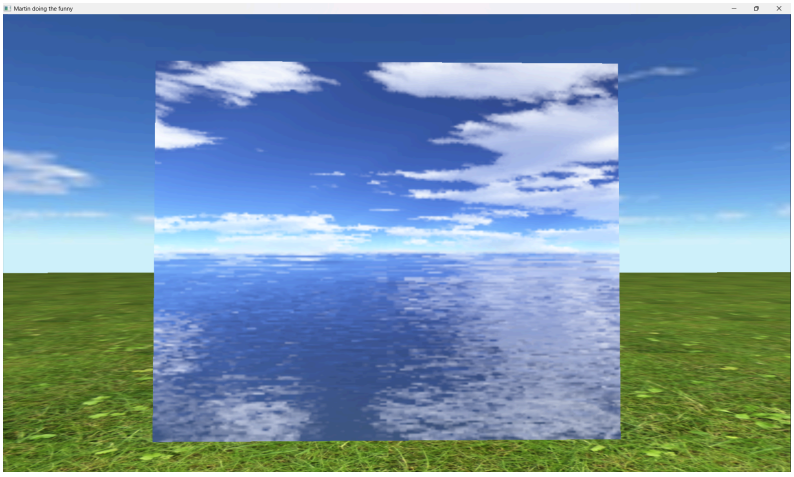

Figure 1: Pre-rendered cubemap.

Dynamic Cubemap

Dynamic cubemaps work by giving each mirror it's own FBO, that contains a cubemap texture. The scene can then be rendered from the mirrors position, in the positive and negative x, y and z directions. The result of these renders is saved in the cubemap texture of the mirror's associated FBO. This texture is then used for sampling when rays are reflected on the mirrors surface as depicted in the basic cubemap section. This does not solve the issue of distored perspective, but it makes the various models and changes in the scene visible in the mirrors reflections.

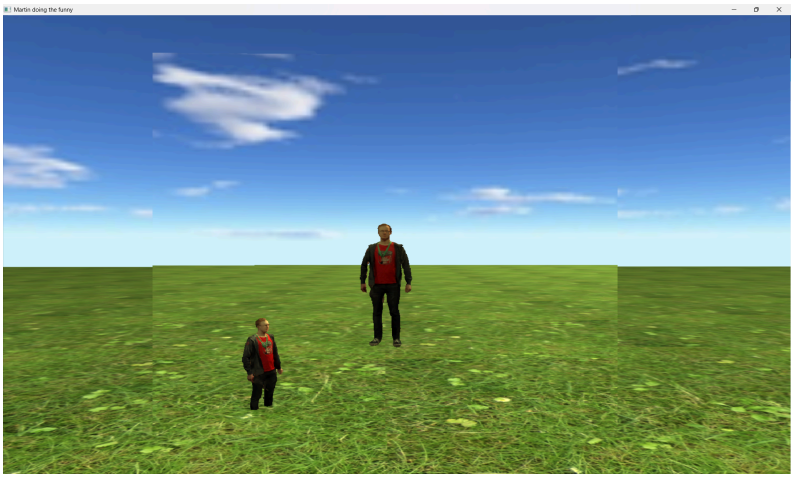

Figure 2: Dynamic cubemap.

Basic Bumpmap

By using simple sinus functions with surface position data as input we can distort the normal vectors on the surface of the mirror to create fun affects like those found in "fun houses" at amusement parks. A problem with this technique is that bumps in a real mirror could cause reflections of the mirror itself. Because of limitations with how the dynamic cubemaps are created, this effect can't be replicated. The mirror will also never be able to reflect it self since the actual mirror model is flat.

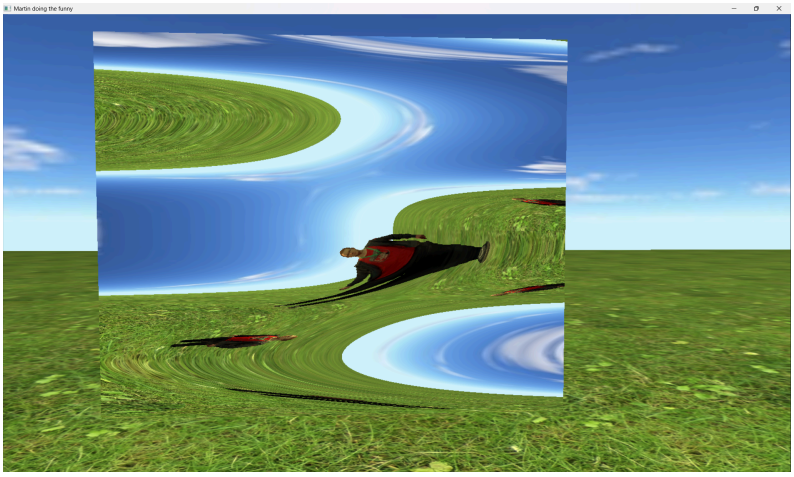

Figure 3: Normal vectors distorted.

Parallax Correction

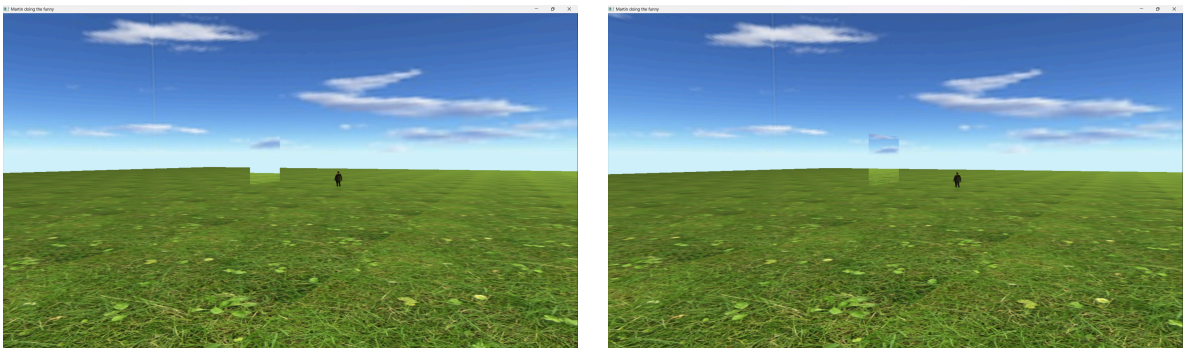

Figure 4: Parallax corrected cubemap on the left, normal cubemap on the right.

Recursive Mirrors

If there is multiple mirrors in a scene they won't be able to reflect eachother correctly when using only one FBO for each mirror. Since one mirror will render its cubemap texture before the other there will be visible artifacts. One idea to solve this is to use two FBO:s per mirror, which are switch between each frame. This will solve the problem by using the cubemap texture from the previous frame when rendering the cubemap texture for the current frame. This will cause each recursive reflection step to lag one frame behind, but after a while the changes will propagate through the reflections.

To solve the issue of multiple mirrors, using dynamic cubemaps, being able to see each other we used a recursive method. The idea is that each mirror has 2 FBO:s. And on alternating frames they will switch between which they read from, and write to. So one mirror will see the last frames version of the other mirror. This does cause a slight lag in the mirror reflection when reaction to movement since there will take several frames for the changes to propagate through the mirrors seeing each other.

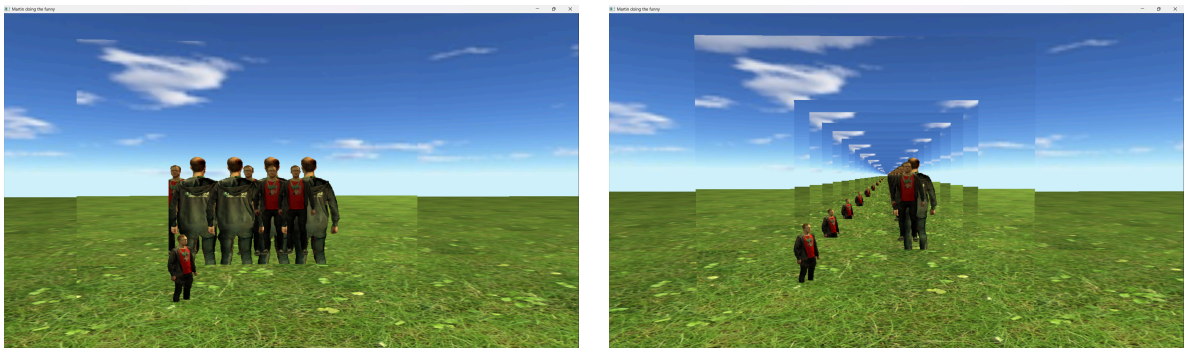

Figure 5: Comparison of recursive mirror with and whitout parallax correction

Interesting Problems

Conclusions

Overall, pretty much all of the techniques had various drawbacks, and were quite tricky to implement correctly. So it's possible that overall, for reflective mirrors/objects raytracing is probably a more approriate technique, since it has few or none of said drawbacks. Unsure what the performence cost comparison would be, but the best version of the scene we managed to achieve ran quite slowly on a decent-ish lapptop since it featured to mirrors, and that meant we had to render the scene 6 times per mirror, and then a final render from the "players" perspective.

There might be cases where these techniques are cheap and work well for flat reflective surfaces like still water, but for accurate mirrors it seems raytracing would be more worthwile.

It also generally doesn't work too well for multiple reflective surfaces reflecting each other recursivley, atleast not in our implementation. So for scenes with multiple reflective surfaces raytracing is probably prefreable.

Source Code

The code for the project is available in an open github repo: https://github.com/the-JS-hater/MartinFunnyHouse