Rendering “Portal” - Milestone 1

Torin Rudeen (torinmr)

Original Milestone Description:

“Able to render and move about an interior scene with statically-placed portals. Papers/blog posts on mirror/portal rendering understood.”

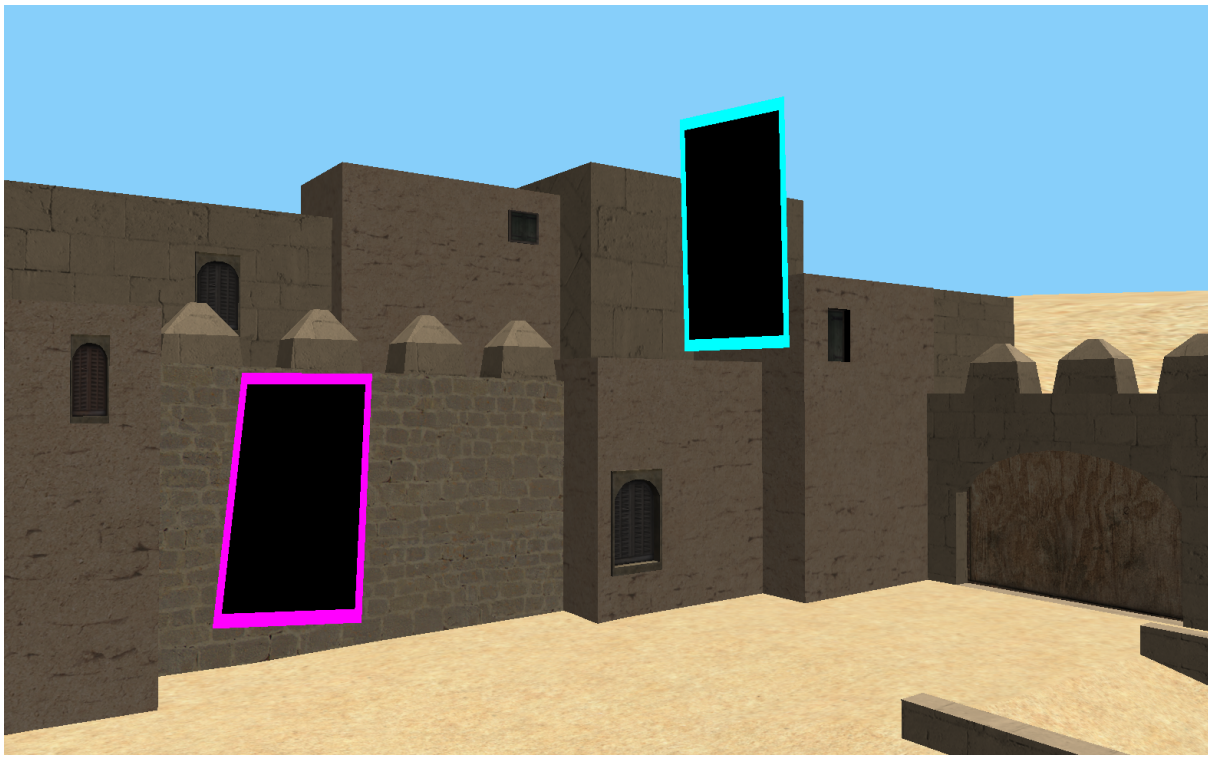
Current Progress

Overall, I think I’ve met the milestone goals, though perhaps not in the exact order I envisioned.

First of all, I created the basic infrastructure code for my assignment, including code for camera movement, loading and rendering a complex scene from a Wavefront (.obj) file, and basic lighting. This code was based on the excellent tutorials at learnopengl.com, including the use of the Open Asset Import Library (Assimp). The greatest challenge was in finding appropriate assets - the vast majority of freely available online models couldn’t be loaded by Assimp, didn’t include textures, required a proprietary program to load, and so forth. I finally found a suitable model, a fan recreation of a popular FPS level.

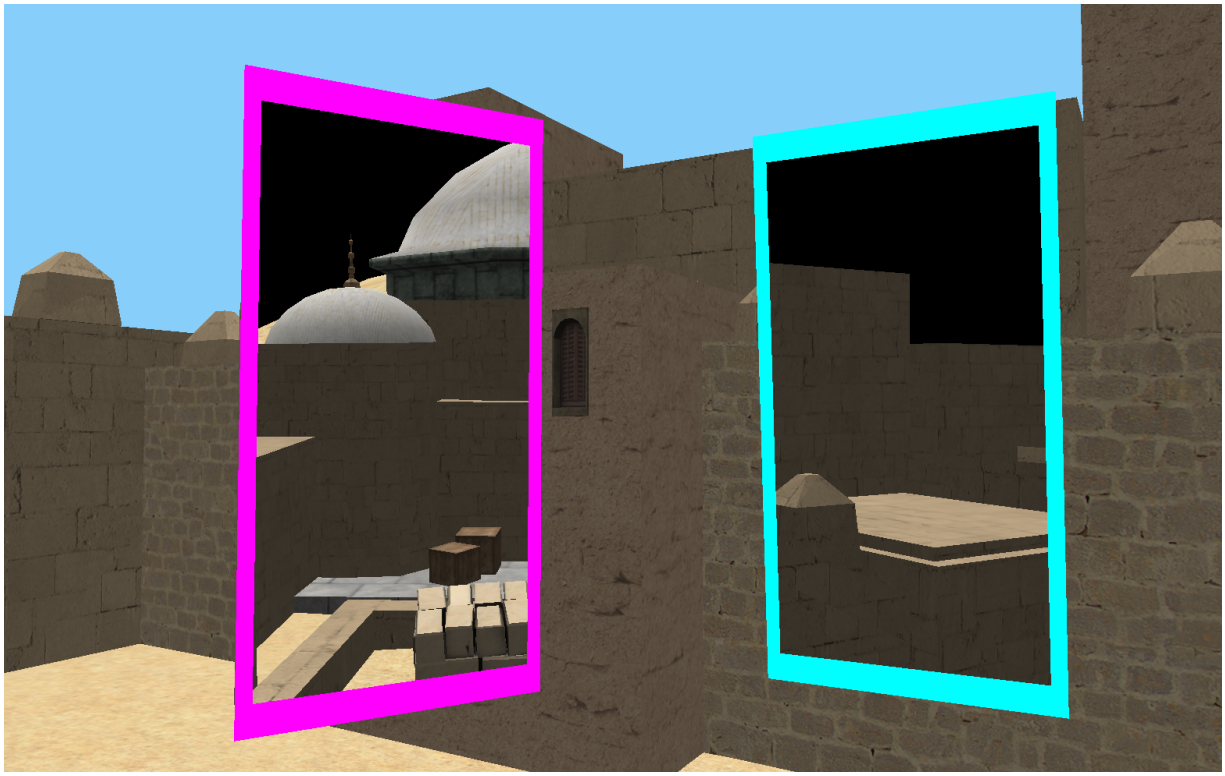
It’s beautiful! Well, functional.

Next, I created a C++ class to represent portals. Each portal stores its position, orientation, geometry for rendering its border, and a pointer to its linked portal. I made it so that portals could be created at will, but they do not automatically find the nearest flat surface to attach to as in the original game - rather, they simply appear a few meters away from the camera, oriented towards the player. This should have been straightforward, but I ran into two nasty bugs which took many hours to straighten out - the first was because when I introduced a second shader, I only called glUseProgram before rendering, not before calling glUniform, causing uniforms to be bound to the wrong shaders, with mystifying results. The second was because I passed the wrong stride value to glVertexAttribPointer, causing the normals in my portal vertex array to be interpreted as coordinates, which also had mystifying and frustrating results.



Non-functional portals placed at arbitrary locations

Finally, I got some rudimentary rendering *through* the portals working. I actually never got around to reading any papers, but I was able to figure out a method based on the basic idea of using a stencil buffer. Basically, when rendering the black interior of the portal, I also wrote to the stencil buffer, writing a different value for each of the two portals. Then, at the end of my rendering pass, I rendered the scene once more for each portal, but with that portal's stencil mask, and with the view matrix based on a view vector coming out of the other portal, rather than out of the camera.



Now you're rendering with portals!

This actually produces a pretty good effect, but it has some simplifications I'll need to improve on in the rest of the project. For example:

- Portals aren't yet rendered recursively. I think this will actually be relatively straightforward to implement - I just need to make the rendering call recursive (with a depth limit), and AND the stencil buffer each time for succesive portals.
- The view vector always comes straight out of the other portal, ignoring the angle you're looking into the original portal. This is a simple matter of algebra that I didn't get around to implementing.
- The view vector originates directly at the other portal, meaning objects seen through the portal don't appear smaller when you're further away from the original portal. Moving the view vector backwards won't work, because then a wall *behind* the output portal could be seen through the portal! I think it may be possible to fix this by tweaking the projection matrix in some way.