CS 194-26 Fall 2018 VR Panorama

Michael Gibbes

1 Overview

VR Panorama!

My Primary Goal:

The user should be surrounded by a 3D panorama of an image in realtime. The 3D panorama should shade certain interactable objects in the world. For instance, if you are standing in an open field and holding a round mirror ball, the ball should show the field in its reflection. My goal here is to create a visual experience such that you think maybe the scene could be taken from a real camera.

My Tools:

1. C# and HLSL languages
2. The Unity Engine
3. HTC Vive

2 VR Scene Setup with Unity

Though "VR" is in the title, for the most part this component is the "frame"[1] around which the rest of the project is built. Therefore, most of the VR-related discussion will be in this section.

Unity recently updated its input system to an interface called the **InteractionSystem**[2]. This system attempts to abstract away the act of pressing a button or thumbing the trackpad from the act of... well, performing a virtual interaction. I was able to successfully rig the scene using this new interface. Here are the most important components of my scene:

1. **CameraRig**: an object that holds the camera corresponding in virtual space to the VR headset's hardware location in real space. I kept the camera invisible in virtual space.
2. **Hand**: a standard Unity script just like it sounds. Unity released a polished "glove" model for the controller that I used.
3. **Throwable**: a standard Unity script that interacts with the Hand script. For an object that has realtime physics enabled and a rigidbody mesh, attaching this script makes for a satisfying experience

It is necessary to understand the Throwable script to understand how my panorama spheres work. When you as the user pick up an object such as a ball, the object replaces your hand model. Then when you throw it, your hand reappears, and the object takes on your throwing velocity and rotation. I demonstrate throwable objects through animation on my website (gibby.me).

3 Equirectangular Images

Equirectangular images enable me to do most of what I did in the latter part of the project. The classic example of an equirectangular image is a flat map of the earth. Towards the top and bottom of the picture, the view becomes stretched horizontally. And as we know, Antarctica doesn't really stretch out over the bottom of the Earth (unless you're a flat-earther). Among other properties, these images are 2:1 in size. Exploring how to create the properties of such an image would in fact be a computational photography project all to itself. Having the right hardware like a spherical camera makes taking an equirectangular photo trivial. I, on the other hand, do not have that hardware, so I pulled from a few online sources.

In Unity, the Cubemap object is a special data structure that makes it easy to place textures onto 3D meshes. It can be loaded in face-by-face, or (conveniently enough) you can use an equirectangular image.

Figure 1: 2D input cubemap layout[3]

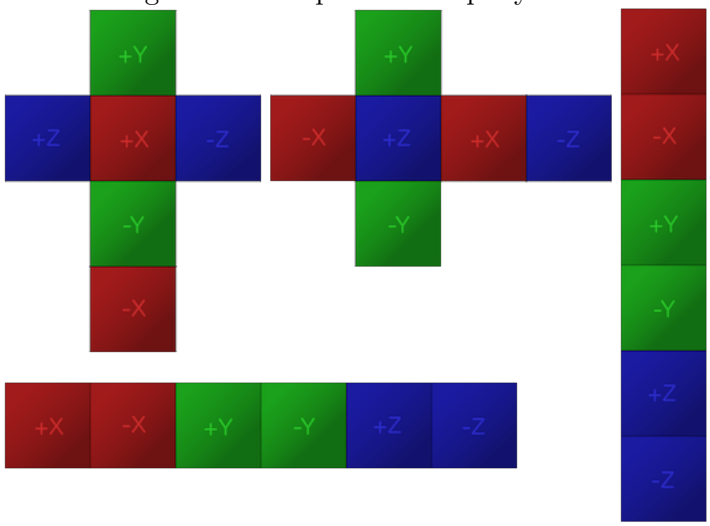
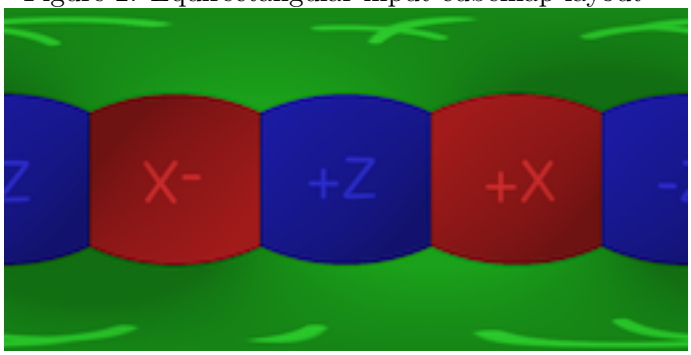


Figure 2: Equirectangular input cubemap layout[3]



Looking at the next page, you can see how I transferred a sample equirectangular image to a Unity cubemap, which in Unity is always represented as a sphere.

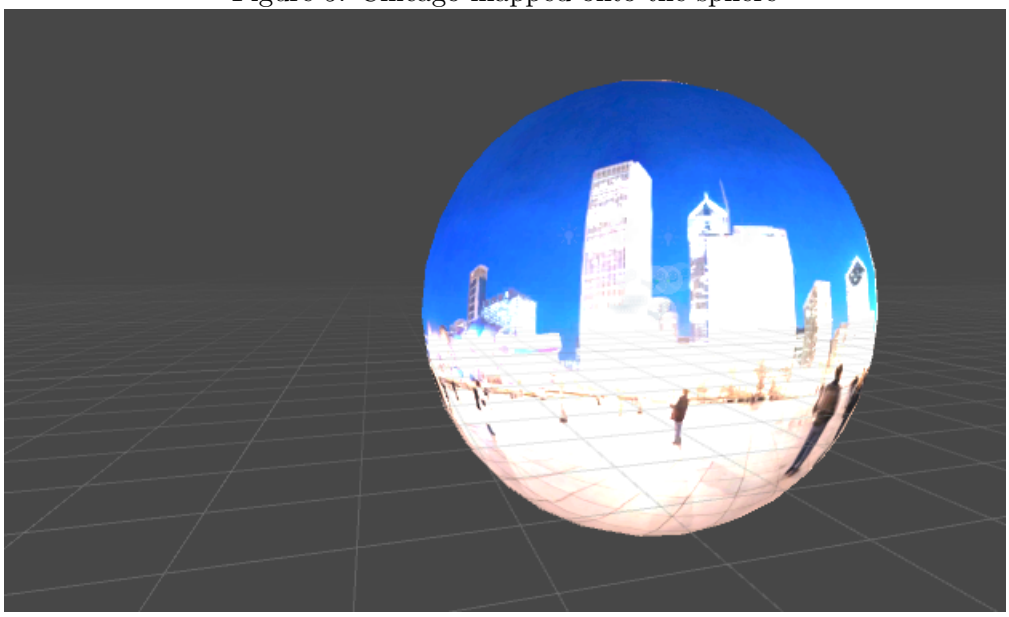
Figure 3: Chicago equirectangular image[4]

Figure 4: Cubemap of Chicago

4 Panorama Mapping

Once I had a functioning cubemap, I had to apply it to a sphere. To make my panorama idea work, I inserted a giant sphere into the Unity scene, big enough that the user should not be able to walk outside of it in virtual space (I conveniently did not implement teleportation to trap the user inside the sphere). To map this texture I needed a custom shader – I modified a script[4] so that it would work with my implementation (more about shaders in the next section). Finally, I reversed the normals[5] of the standard sphere mesh so that the surface would render from the inside, which involved iterating over some triangle soup, reversing the direction of the normal vectors, and also reversing the ordering of every triangle.

Figure 5: Chicago mapped onto the sphere

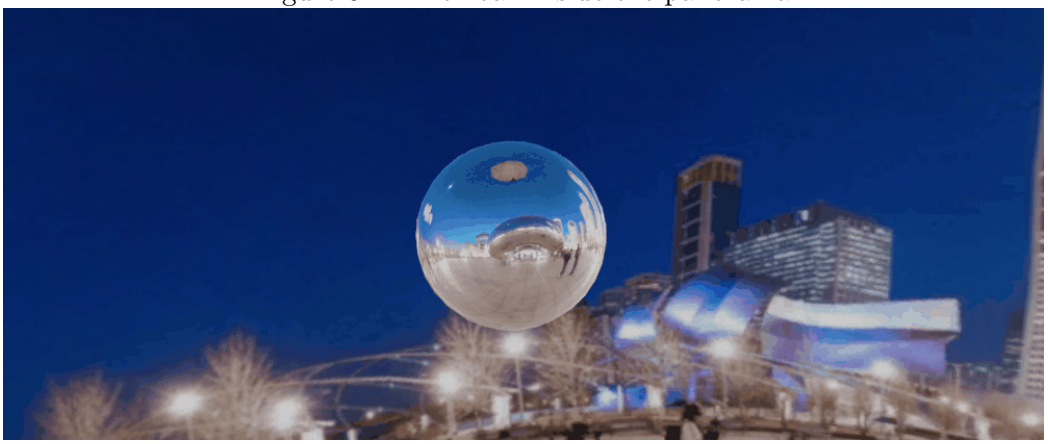


And with that, the panorama environment itself was ready! Again, you can view some of the custom animation GIFs on my website `gibby.me`.

5 Environmental Mapping with Spheres (Mirror BRDF)

Once inside the panorama, objects should reflect their environment, the panorama itself, as well as potentially any other 3D objects that might be present. Unity "reflection probes" trivialize this process. You can capture realtime data every frame by sticking one of these probes inside of the object and allowing them to capture spherical data in the form of cubemaps. However, those operations are really costly, and I wanted to implement my own probe. Observe, for one, that the user is trapped inside the sphere and also that the scene doesn't change. That means we only need to probe exactly once. (Note - a Unity reflection probe can do this by "baking" data, but I still wanted to do this with my own, single, custom probe.) After the initialization of the scene, I stuck a temporary camera at the origin (0, 0, 0) and used a Unity built-in function to render a spherical snapshot of the scene as a cubemap, which I then transferred to another script that I called MirrorMap. MirrorMap communicates the the probe data to the shader.

Figure 6: Mirror ball inside the panorama



The shading language HLSL was a bit hard to understand and heavily depends on Unity API, even though there exist some pretty nice walkthroughs[6] in the official documentation as well as a whole book[7] about it. However, I was able to find a public domain script[8] that does what I want, which is to compute the mirror's surface intensity using (1) the cubemap data, (2) the normal vector, and (3) the incoming view vector, much like what I did in my graphics projects.

Figure 7: Another sample from my program

6 Conclusion

My experimentation into producing this VR experience proved successful, largely because of the flexibility and power of the Unity engine and the improved interface with VR hardware. There are two main future considerations:

1. **Equirectangular Images**: These nifty data structures allow for so many applications. At its core, my project breaks down to producing and manipulating equirectangular images. While it was sufficient to acquire free resources from the Internet, it would be nice to be able to create custom ones. If somebody were to reproduce my project or if I were to extend it, I would have an in-depth exploration of how to capture photos and project them into well-formed equirectangular panoramas using clever transformations.

2. **GUI Considerations**: While not necessary, it would be nice to give the user some customizability. Some possible parameters might be the image itself (choose from many!), the reflectance properties of the balls, or the brightness/warpedness of the surrounding panorama. You might even be able to tie this app in with Google Maps to have maximum maneuverability.

7 References

1. Eric Van de Kerckhove, "HTC Vive Tutorial for Unity," https://www.raywenderlich.com/792-htc-vive-tutorial-for-unity, 2016

2. Lawrence Yang, "Steam VR Unity Plugin 2.0", https://steamcommunity.com/games/250820/announcements/detail/1696059027982397407, 2018

3. Unity, "Cubemap", https://docs.unity3d.com/Manual/class-Cubemap.html, 2018

4. Veronica Valls, "How to make a 360 image viewer with Unity3D," https://medium.com/game-development-stuff/how-to-make-a-360%C2%BA-image-viewer-with-unity3d-b1aa9f99cabb, 2016

5. Joachim Ante, "ReverseNormals," http://wiki.unity3d.com/index.php/ReverseNormals, 2012

6. Alan Zucconi, "A Gentle Introduction to Shaders," https://unity3d.com/learn/tutorials/topics/graphics/gentle-introduction-shaders, 2018

7. John P. Doran, Alan Zucconi, "Unity 2018 Shaders and Effects Cookbook," https://books.google.com/books?id=7cRiDwAAQBAJ&source=bl&ots=MgRsTglakf&sig=dUyhg4T-6tZBmudvHOtFYvPW
hl=en&sa=X&ved=2ahUKEwi38ubAlJffAhUzPnOKHVxmCWIQ6AEwBXoECAkQAQ#v=onepage&q=lightingsim
f=false, 2018

8. Wikibooks Public Domain, "Reflecting Surfaces," https://en.wikibooks.org/wiki/Cg_Programming/Unity/Reflecting_Surfaces, 2018