Labeling Polyhedral Scenes

Final Report and Contributions

Drew Mainprize

For: CIS*4900 – Dr. Neil Bruce

April 18, 2023

*** Disclaimer – all the assets from this project are available on my GitHub at https://github.com/DrewMainprize/CIS4900-PolyhedralLabeling. I will be referencing code and shaders that I used, and it will all be accessible from this repository.*

Introduction

In the growing world of Artificial Intelligence, we are constantly bombarded with new developments in the form of self-driving cars, or face and object recognition, or a magnitude of other areas. It seems that the field is growing faster than we can even keep up with. However, what if we took a step back and looked at how these advancements are made possible. How do self-driving cars see the road and pedestrians, how do neural networks know that they're looking at a face, and not an object with a similar structure. The labeling polyhedral scenes problem that I looked at over the course of the semester takes a deeper dive into how neural networks view scenes.

The polyhedral labeling problem aims to investigate the effectiveness of neural networks at labeling various edges in polyhedral scenes. By using scenes that are already labeled by their convex, concave and outer edges as input, we hope to train neural networks on these scenes to the point where we can test how well the neural network can come up with labels for new polyhedral scenes. The idea of the study is that if the neural network has a hard time labeling polyhedral scenes properly by their inside and outside edges, how well does an artificial neural network (ANN) really understand the images that it is interpreting. If an ANN cannot strip down the components of scenes to classify its basic edges, then how well can it really “see” the images that it is being shown.

Polyhedral Scene Generator

Before we could even begin looking at training the neural networks, we need a way to automate the process of generating polyhedral scenes with labeled edges. I modified code that was provided that could generate 3D polyhedral scenes based on the number of planes and probability that each plane is visible (CIS4900-PolyhedralLabeling/poly-scene-generator). The modifications I made were small and included adding a Python script that would run the scene generator a set number of times and store the files in a folder without triggering the GUI (CIS4900-PolyhedralLabeling/createPoly.py). The script can be extended to bake the convex and concave labels onto the images before storing them. It’s important to be able to generate a large number of images in order to test the methods we will be using to label them for correctness, and eventually, to train the AI model.

Unity Double Faces

Any image processing software, Unity included, was a totally new concept to me at the start of this project. One of my biggest personal achievements this semester is that I am now fully comfortable with Unity and Blender. I also feel that I have a strong grasp on image processing in computation in general, with concepts like shaders, baking, geometry nodes and scripts that handle image processing.

To tackle the problem of convex and concave edges, I started by looking at shaders in Unity. One issue with Unity that I discovered early on was that, when importing object files that were generated with the polyhedral scene generator code, Unity was unable to resolve the non-visible faces of the object. Half of the faces from every file were missing. To tackle this, I wrote a C# script that could be run on every polyhedral object that corrected this (CIS4900-PolyhedralLabeling/DoubleFaces.cs). The script takes the original object file and pulls all the triangles into an array. Then, each triangle is duplicated, with the duplicated triangle being inverted. This same process is done with the vertices and the normals. The end result is the same object file but it is now compatible with Unity as the back faces of the mesh are now visible. Without the back faces visible it would be impossible to properly label the convex or concave edges as we would not know the intersecting angle at the edge.

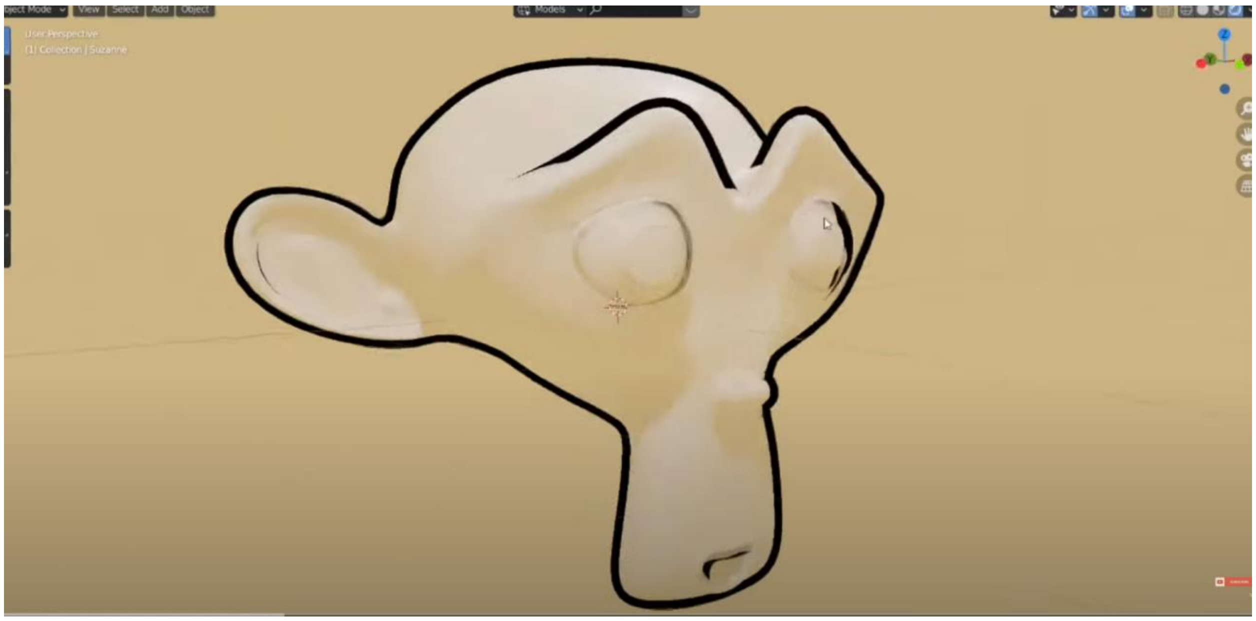
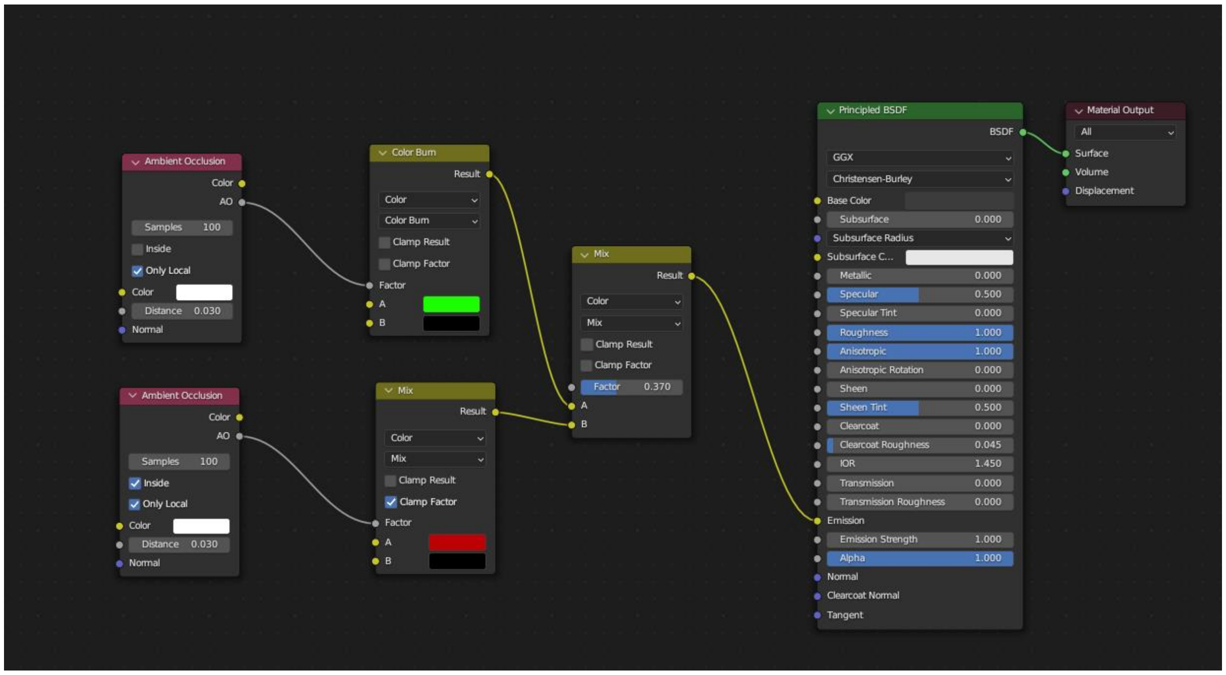


Figure 1. Example silhouette style outer edge labeling.

Overall, I ended up running into many issues working with shaders in Unity. As I expected after dealing with double faces, Unity did not handle the object files very well and I had a hard time grasping the concepts necessary to write my own shader to label the convex and concave edges. I shifted my focus to labeling the outer edges and attempted to replicate something like the outline shown in Figure 1. After more research, I found that this silhouette style outline only works on certain geometry. Namely, with more curves and less sharp corners. It was clear that this idea was not going to work for any arbitrary polyhedral scene as most of the scenes contain sharp corners with acute angles. I finally decided to shift gears from Unity and try working with Blender instead.

Blender Shaders

I had the most success using Blender to create shaders through individual nodes that could be fine-tuned and were easy to navigate. I found this much more manageable than creating shaders from scratch in Unity with no guidance. There were helpful tutorials and forums that focused on similar issues of labeling edges inside of objects and I was able to fully solve the concave and convex edges. The image in Figure 2 shows the shader that I created to label these edges.



Ambient Occlusion: Samples 100; Inside ☐; Only Local ✓; Color; Distance 0.030

Ambient Occlusion: Samples 100; Inside ✓; Only Local ✓; Color; Distance 0.030

Color Burn: Color; Color Burn; Clamp Result ☐; Clamp Factor ☐; A 🟩; B ■

Mix: Color; Mix; Clamp Result ☐; Clamp Factor ✓; A 🟥; B ■

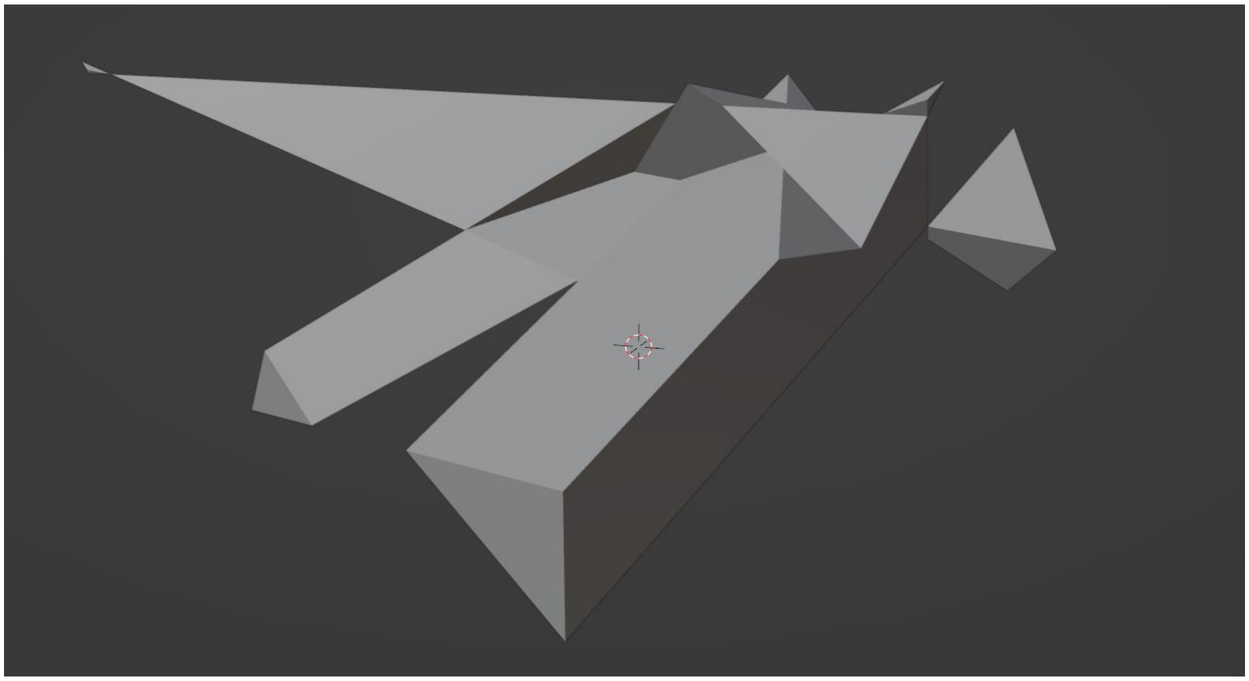
Mix: Color; Mix; Clamp Result ☐; Clamp Factor ☐; Factor 0.370

Principled BSDF: GGX; Christensen-Burley; Base Color; Subsurface 0.000; Subsurface Radius; Subsurface C...; Metallic 0.000; Specular 0.500; Specular Tint 0.000; Roughness 1.000; Anisotropic 1.000; Anisotropic Rotation 0.000; Sheen 0.000; Sheen Tint 0.500; Clearcoat 0.000; Clearcoat Roughness 0.045; IOR 1.450; Transmission 0.000; Transmission Roughness 0.000; Emission; Emission Strength 1.000; Alpha 1.000; Normal; Clearcoat Normal; Tangent

Material Output: All

Figure 2. Shader created in Blender for convex and concave edges.

The shader takes advantage of ambient occlusion in Blender to catch the various edges in objects based on their angles. By default, the ambient occlusion node catches concave edges of objects, however by setting the “inside” flag we can catch the convex edges instead. I used the “distance” scale to modify the intensity of each shader so that it looked like a proper label and combined the two ambient occlusion nodes into a single material. I ran into an issue where the node that was set to catch only the convex edges (the red node), was catching concave edges as well. This gave the impression that the concave edges were a mixture of green and red which would surely cause problems down the road. I discovered colour burning to combat this. Colour burn can be used for many reasons but what I was interested in was how colour burn both increases saturation and reduces highlights. Using this, I could make the green labels brighter without increasing their shadow and obscuring the object underneath it. This worked very well and after adjusting the intensities and values of both nodes I was left with a shader that gave very distinct labels for convex and concave edges. An example is



shown in Figure 4.

Figure 3. Example object before adding shaders.

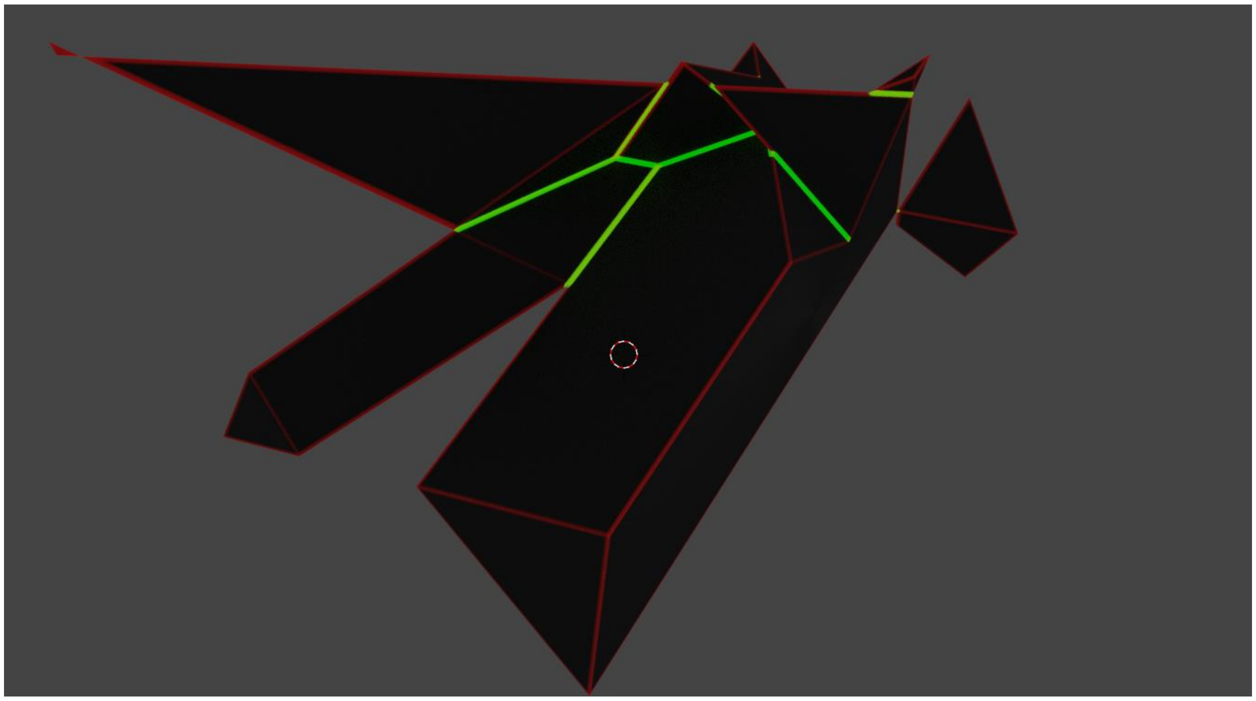


Figure 4. Example object file labeled with convex and concave edges.

As shown in Figure 4, the shader responds to concave and convex edges quite well, the red labels catch all edges on the object, while the green catches only the concave edges. Using the colour burning method mentioned earlier, the green edges overpower the red by being both brighter and thicker to present a clean object with no ambiguity between labels. I tested this shader on plenty of other objects, using the script to generate a large number of images and baking the textures to the object to speed up testing and avoid opening a huge number of Blender tabs to see the objects. The shader was highly effective when it came to convex and concave edges, but I still had no solution to the outside edges problem.

The outer edges problem proved to be my biggest difficulty over the semester. I tried working with convex hull to create a highlight behind the image based on the camera angle. This did not work and distorted the image with the shader that was already attached to it. I tried reverting to later builds of Blender to use old features that seemed like they could fix my problem. I was unable to find anything that could output a clean object with the outside edges labeled.

Conclusion

After weeks of trying to solve this problem, another student who had been working on the same problem had the idea of using Blender geometry nodes to create all the labels. They had progressed quite far in this problem, so I spent the last few weeks of the project helping them with testing, finding edge cases and refining their solution. Overall, there were many roadblocks I encountered over the semester. Between being completely new to Blender and Unity and the process of learning these tools with enough proficiency to solve problems that do not have a direct solution. I learned a lot not only conceptually but about the research process itself and the effort and strategies that are necessary to solve problems.