**Module 8 Option 2**

James Strauser

Colorado State University Global

CSC 405

Sep 10, 2023

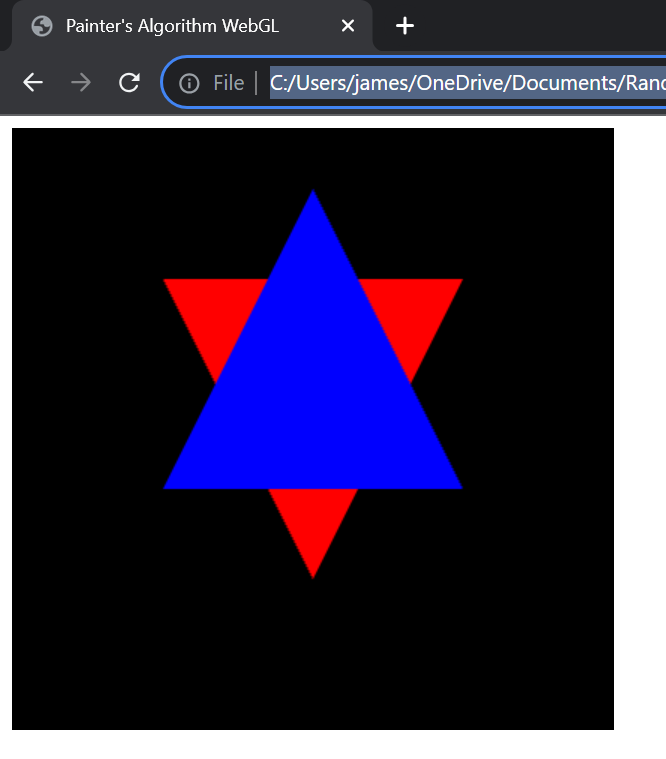
Lessons Learned

The course has the general topic of being an introduction to computer graphics, more specifically WebGL. Module 1 reflects this best as it serves as introduction to the topic of computer graphics and WebGL. This is always a good place to start a topic as learning skills without knowing their purpose or uses can be challenging but the module serves to remind students exactly what computer graphics is capable of. Module 2 is where we start learning how to actually code by doing a very interesting example in the form of the Sierpinski Gasket, a fractal which is something that is much harder or even impossible without a computer. The main take away I got from Module 2 was the idea of black box programming which has it’s own concept and definition but I’ve chosen to think of it as “If it ain’t broke, don’t fix it” which can speed up programming in WebGL a lot. For instance in the Sierpinski Gasket assignment I did not understand what many of the lines of code did, however that did not matter, I knew they worked and I know what bits to change to make the program I want so I could adapt the given example programs into something that worked as a finished project. I used this throughout the course as once I found something that worked I could just reuse it time and time again saving me a lot of time and headaches in later assignments where there were more important problems to learn. Module 3 took a step back from hands-on programming and focused on the topics of animation and input. Truthfully, I already knew this stuff, the animation discussed was just geometry topics and input devices I knew from my previous experiences in programming. It did teach me however what the right terms to use were, though I do not know how valuable that knowledge is.

Module 4 returned to programming with Geometric Objects and Transformations, or more precisely how to draw shapes in WebGL, which is a complex but also simple process. First you need all the set up for shapes, canvas, shaders, and such but you need that to draw anything. For shapes the normal approach is to define vertices and then draw the connecting lines and intermediary faces, the idea of a convex hull is one such way we can accomplish that. Module 5 was a doozy as it discussed perspective, luckily I had previously worked with Unity and CAD software so the terms were not unfamiliar, however implementing a viewer myself was much more challenging. It showed me the depths WebGL can have and how unsuited I am to programming with it, at least I can understand JavaScript well. I feel bad for those who struggle with both. Either way this was an important module as perspective is integral to good graphics. Equally integral is module 6 that covered lighting, and I had even more trouble with the assignment of this module. Although I fail to see the connection between understanding lighting, shading, and spheres the assignment was to approximate a sphere interactively. I got the program most of the way done but the bugs I encountered left me with little to go off of so eventually I learned an important lesson that is highly applicable in all real life scenarios, someone has probably done this challenge before I should go see what they did. In this instance I went with using a library to do the subdivision of spheres which is good practice for professional settings, a library you trust will be much safer and quicker to use than anything you can put together on a deadline. Module 7 was about other important ideas in computer graphics, textures and buffers . Not a lot to say since I’m avoiding regurgitating information and there was no critical thinking for this module. So I wouldn’t say a lot was learned since just reading is not the way I learn skills. Luckily buffers are a simple concept and I have experience with textures so there’s no gap in my knowledge. Module 8 (this one) is about changing the mindset from working with geometric primitives to working with the pixels that represent them. This opens countless doors for more complex graphics, and the hidden-surface removal problem is one of them. But more on that in the program analysis.

Overall I learned a lot in the class, and not just things related to computer graphics, which is helpful as I hope to avoid in my professional career. I know that as much as I want to understand every part of the code, if I can trust that it works as I expect it to then that is good enough. I also know that if I’m struggling with something, I can probably find a much better implementation of it if I look in the right places. Those two are the biggest lessons but there are a few others aside from feeling like I learned a new programming language in the form of WebGL. This class is definitely helpful for preparing myself for the future.

Program Analysis



Behold a star, and yet I did not define the vertices for a star. Well technically I did, but I didn’t draw them all at once as evidenced by the two colors, this is the power of overlapping shapes. Underwhelming I know but sort of interesting. As we are using the painter’s algorithm here the process is ultimately simple compared to normally drawing shapes. First when creating the triangle’s data include a z position to define it’s distance from the camera, as well as a color for pazazz. Next you sort the triangles by their z values so that the highest z and therefore farthest from the camera is drawn first. This way when closer objects are drawn they will naturally cover up the previous objects, just like how a painter can literally paint over sections they wish to cover up. All that’s left is to draw them in order, remember to not clear the canvas after every draw and to include sending color data to your shaders which you likely had to modify. Honestly a very simple solution however the flaws are apparent, there is no room for things to be transparent, opaque all the way here. As well as other things but honestly the opaqueness may as well cover up the other issues since it is a big hurdle. If you want things more transparent the Z-Buffer algorithm is more accommodating. Last is the discussion of the primitive I worked with, it’s triangles because they had the least number of vertices and I dislike hard coding vertices. Theoretically lines are the minimal vertices in which I can demonstrate the painter’s algorithm, but I was not sure how well it would work and I did not want to waste time on things that were impossible.