COMP 371 Project Report

OpenGL Anti-aliasing

Hy Khang Tran ID: 40181444

I) Recap

- This project was aimed to implement and introduce a way to inspect relevant anti-aliasing techniques used in rendering, using OpenGL.
- There were 6 main objectives + 1 optional objective in total:
    - **SSAA**, **MSAA**, NFAA (**FXAA**) and **SMAA** for anti-aliasing techniques.
    - **Zoom** + **grid**, **Performance** gatherer, (Optional) **Side by side** AA rendered image for showcasing those techniques.

II) Progress

a. Anti-Aliasing Techniques

- **SSAA:** I tried to implement this last but end up doing it implicitly when trying to render to texture with different resolution. What this means is that I render my 'No AA' triangle shape in different resolution onto a small fixed size display plane in order to compare the rendered shape. This unexpectedly makes it that if the triangle I render is of higher resolution than the quad, then it's a SSAA when I try to texture it onto the quad.
- **MSAA:** This technique was easy implementing directly on the scene, but because I want to render it in a different resolution and put it onto a display plane that I ended up

complicating the problem a whole lot more. From having to modify all the framebuffer configuration to using `GL_MULTISAMPLE`, to translating/copying it onto an intermediate buffer that stretch the multi-sampled triangle (using `glBlitFramebuffer`) onto the quad made this technique way too complicated for demonstration. Looking at the result, I'm quite sure I have partially failed on copying it onto the higher-res quad which explains all the artifacts on the multi-sample triangle. The result turns out to look better when we render the triangle in higher resolution. Nevertheless, there were no jagged edges, which I'm very happy to see.

- **FXAA:** I ended up choosing FXAA over NFAA since there are more readily available resources for it (thanks to NVIDIA). This one was difficult, really difficult because it all happened on the fragment shader. That means I really had to up my level in GLSL, while also jamming my brain with the math needed to calculate the approximation. While there are resources online, it's still very hard to adopt them into your own code. And even then, I can only get FXAA to work on the texture that I apply to my triangle (the triangle is of course render to the texture of the display quad). As for smoothing the edge, FXAA provides minimal 'smoothness' since I render objects separately, which means the algorithm can't really sample the edge pixels' neighbors that well.
- **SMAA:** This one is a complete bust. I can only blame my incompetency. While I know in theory how it supposed to work, and some of the calculation needed, I couldn't come up with a realistic way to implement it in OpenGL with the given time frame.
- **Overall,** I felt like I did fall short on trying to implement all 4 of my proposed AA techniques. But I did learn *a lot* on how, where and when to use them.

b. Showcasing

- Overall, I feel like I had more success in trying to showcase the techniques:
- **Side by side image:** I would like to start with this since this was the bane of this project. I spent so many hours trying to understand framebuffer, getting them to work in order to realize this. I even have to ditch my entire OpenGL abstraction! Nonetheless, I made it work, I can produce 3 side by side quads 'display plane' that I could render my 3 triangles with different AA techs onto them. It's not perfect but it's so satisfying to see the triangles line up in different appearance and fragments.
- **Performance gatherer:** This was a bit of a hack. I used Imgui to display the render time of the three triangles. The numbers check out and explain the reason why we have these different techniques but I still wish I had more time to really profile my GPU usage during these renders.
- **Zoom + gird:** I didn't do the grid since I feel like it's a bit unnecessary especially when I'll be looking at lower res triangles, still would have been nice to have. I was able to implement camera trackball almost no problems, as well as free movement and zooming functionality with some clever view matrix and vertex shader manipulation.

III) Reflection

- This was a hard project. The OpenGL multiple objects render really caught me off guard and at times I felt like I just couldn't make it. The render-to-texture with framebuffer duo also really hammered me, and some of the AA techniques even turned out to be unrealistic with my current skill level.

- However, I felt like I have given my best. It was not perfect nor outstanding in anyway but given what I had and what I had lost, I'll still be proud of myself, and willing to improve even more.

IV) Acknowledgement

I would like to express my thanks to the professor, the TAs and all of my friends in COMP 371 for helping me complete this project. This was a challenging journey!