**Anti-aliasing algorithm Midterm Report**

for CS-534 Computational Photography

By Xiaochao Yan

University of Wisconsin Madison

As mentioned, the result of the semester long project will be testing the final performance of different mode of Anti-aliasing.

For final testing, two types of Anti-aliasing mode will be tested.

1. Scene-Space based Anti-aliasing: Without using information from 3D-models used to create the image, only information about the 2-D image to smooth out the jagged lines.
2. Object-Space based Anti-aliasing: using information in 3D model space to smooth out the jagged lines.

**Prediction:** Scene-Space based Anti-aliasing should have a performance very similar to that of without Anti-aliasing at all, because calculate what color should be mapped to each pixel is what takes most of the time of render pipeline and this process is the same for rendering using Scene-Space based Anti-aliasing and for rendering without using Anti-aliasing.

On the other hand, Object-Space based Anti-aliasing will trying to calculate serval colors instead of one for each pixel. This will result (n-1) more times of work compared to without anti-aliasing. (n is the number of samples used).

**Test1:** Screen Spaced Anti-Aliasing vs. No Anti-Aliasing

This test is conducted on a program Xiaochao Yan wrote, rendering a knot, texture is of one color and there is no environmental reflection. Object’s texture only varies depending on the angle of light.

If the prediction is correct, the frame rate of screen space based anti-aliased and original program should not differ by a great amount.

**Tesing Environment:**

System: Win10 64 bit system

GPU : Nvidia 1080

Framerate recording Software: Fraps

Time : 60 seconds.

Monitor : Samsung C24FG70 (1920x1080 @ 144hz)

Result of No anti-aliasing of Running 60 Seconds

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Frames | Times | Min FPS | Max FPS | Avg FPS |
| 4513 | 60000 | 35 | 120 | 75.217 |

Result of Cel-Shading of Running 60 Seconds

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Frames | Times | Min FPS | Max FPS | Avg FPS |
| 4486 | 60000 | 36 | 120 | 74.767 |

The result does show that the overall performance of screen space based anti-aliased and original program are very similar.

But there some question unanswered, why there such great difference in Min FPS and Max FPS. While doing the test, I noticed there is time when graphics glitched for a slight moment (1 Second less) after running for 20+ Seconds. Possible causes are elements in testing environment or the way OpenGl handle certain issues.

Dec 1st Update:

Cause of the frame glitch found on OpenGl website below.

(<https://www.khronos.org/opengl/wiki/Performance>)

Citation:

**Performance is not Linear**

The Frame Time is not proportional to the amount of things that you render. For example, there can be a constant-time job that the driver has to do for every frame. Measuring the performance when rendering very little (or perhaps nothing at all) is meaningless. In fact, a faster GPU on a faster computer might actually be SLOWER when doing very small jobs. However, when rendering more things (as in most applications and games), the high-end computer would most likely be faster.

The frame glitch could also be the reason that I did not set Vertical -Sync setting in OpenGL. Since my monitor is 144Hz instead of normal 60Hz, the OpenGl might be forced at a different Max-Fram-Rate.

**Improvement Can be Done:**

1. Use test environment easies to monitor, such as Linux. For windows there is many background system services. Even for modern computer, those services take a very small fraction of system resources, inaccuracy can be caused by those services.
2. Use Direct-X instead of OpenGL for implementing different anti-aliasing techniques, because later version of Direct-X provides better framerate monitoring mechanism. Visual Studio of MircoSoft provide built-in frame rate control.
3. I choose OpenGL, because it was what I learned in Graphics Course in University of Wisconsin-Madison. If I had done a thorough search prior to writing codes, I should be able to have the experiment done in a more accurate way.

**Test Using a commercial software – 3DMarks**

System: Win10 64 bit system

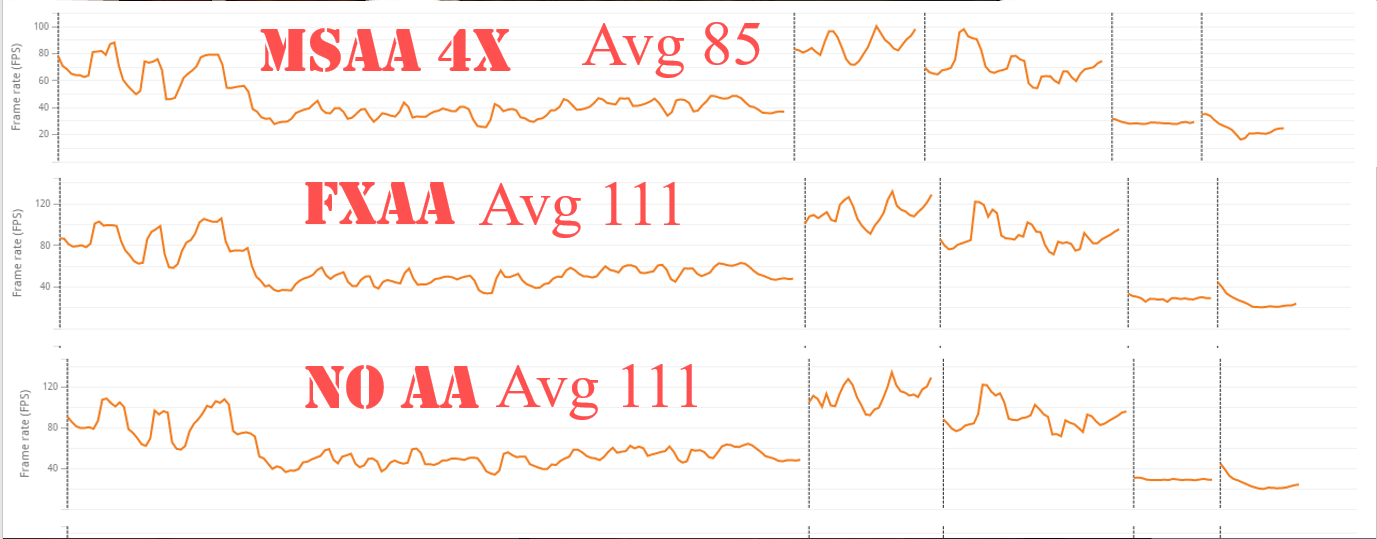
GPU : Nvidia 1080

Framerate recording Software: 3DMarks

Time : 60 seconds.

Monitor : Samsung C24FG70 (1920x1080 @ 144hz)

V-Sync Setting : Off



\* Avg – the average frame rate during testing

3DMarks is a software designed for testing Graphic Card Performance. It enables user to run through a pre-defined scene with custom setting and software itself record system information such as Framerate, GPU Temperature and so on. Detailed version of report for running 3DMarks test called *Fire* could be found in yxch1996 ‘s CS354 research GitHub page.

**Analyze:**

The result of 3DMarks did confirm the conclusion on the performance of the 2 different types. AA based on screen space (FXAA in this case) has nearly no effect on performance. AA based on object space (MSAA in this case) has a significantly lower average framerate.

Since the software used exactly the same scene, including same character movement, same lighting, same textures and so on. The workload for GPU at the same time frame should be the same. What interesting is that MSAA and FXAA nearly the same pattern in framerate. This similarity could suggest that the computational cost relation between FXAA and MSAA is linear instead of exponential.

This could be explained. Since the computational cost of MSAA 4x is known to be 4 \* (computational cost of no AA) and (computational cost of FXAA) ≈ (computational cost of no AA), FXAA and MSAA should have similar framerate graph.

What is not fully explained is that the average frame of MSAA (85) is only 75% of average frame of FXAA (111). Since the computational cost of other part of rendering pipeline is related to number of poly-surfaces in the scene to render, it is impossible to calculate with software like 3Dmarks.

Reference List

Philip Rideout, (2010). Antialiased Cel Shading

<https://prideout.net/blog/old/blog/index.html@p=22.html>

Timothy Lottes, (2009). FXAA

Bart Wronski, (2014). Temporal supersampling and antialiasing