

High Performance Computing

M6X01099

by Andrew Humphreys

FILE	2653_ANDREW_HUMPHREYS_HIGH_PERFORMANCE_COMPUTING_M6X01099_107698_1441664094.DOCX (2.58M)		
TIME SUBMITTED	14-APR-2019 07:52PM (UTC+0100)	WORD COUNT	2861
SUBMISSION ID	104753940	CHARACTER COUNT	17114

An Investigation on Performance Issues When Applying Anti-Aliasing Techniques on a CPU Compared to a GPU using CUDA.

Andrew Humphreys - P148043

MSc Computer Games Development

CLASS: High Performance Computing Games & Visualisation (M6X01099)

Lecture's Names: Andrew Baker & Gordon Dickers

November 2018 – April 2019

Contents

Contents.....	2
Table of Figures	3
Introduction	4
Aliasing	4
Supersampling	6
Grid.....	6
Quincunx	7
CUDA application.....	8
Tests.....	9
CPU Tests	10
GPU tests	11
Improvements	12
Conclusion	14
Appendix	16
Images of Blurring Done by Running the Application	16
Excel test data	17
References.....	18
Further Reading	20

Table of Figures

Figure 1: Mapping of image to screen	4
Figure 2: Popping in action	5
Figure 3: A pixel sub divided in the X and Y co-ordinates	6
Figure 4: The Quincunx process.	7
Figure 5: The result of applying Quincunx compared to 4x Supersampling	8
Figure 6: An example filter	9
Figure 7: Average time taken to complete number of passes on the CPU.....	10
Figure 8: Individual attempts for each pass test	11
Figure 9: CPU vs GPU timings.....	12
Figure 10: Graphs showing high occupancy	13
Figure 11: Graphs showing high occupancy	14
Figure 12: Two kernels with different grid sizes	14
Figure 13: Original image	16
Figure 14: Multi pass 10x	16
Figure 15: Multi pass 50x	17

Introduction

In this investigation we will be getting an understanding what aliasing is and the visual problems it causes in computer games, we will also be looking at the attempts to battle the problems that aliasing cause and then looking at the new problems that are caused by the remedies applied. Also, we will be creating a program using CUDA and analysing the speed differences on both CPU and GPU.

Aliasing

Earlier references to aliasing in computer graphics refer to a problem generated by the inability of a display to sample enough data for the creation of line structures, with line structures being derived from tracings or from images. Freeman explains that the initial forms of line structures are likely to contain noise, either from the original path or image, or it may have been introduced during the extraction process [1]. Today the problem is still here, however, the attention is now more towards how effectively we can resolve the issue in real-time [2].

To get a good understanding on aliasing it's probably best to explain how images are rendered to a computer screen. A screen is made up from pixels that are of a square shape, when an image is passed in to be rendered it's then mapped onto the screen and then decisions are made as to which pixel is to be illuminated. Figure 1 shows the process.

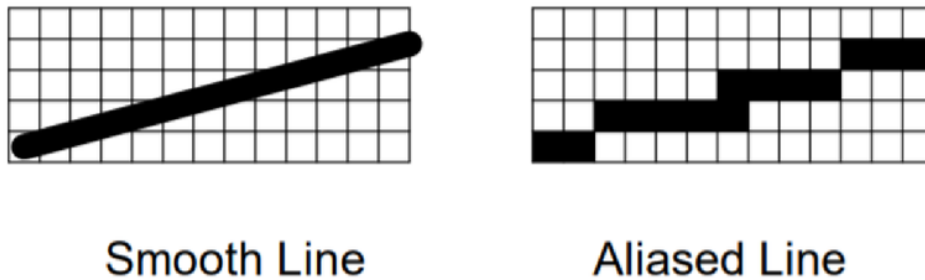


Figure 1: Mapping of image to screen [3]

In the first image it shows the line that is trying to be represented on a screen, most of the image does not 100% fully fit into a square and crosses two squares, so a decision needs to be made as to which squares will represent the current section. Unless the line is perfectly straight then these stair stepping patterns will appear called “jaggies” [1] [4] [5] [6] [7] [3] and turn into “crawlies” [2] [7] when animated. These become an issue as your eyes are more drawn to the boundaries of objects. Other issues that you can get, which is shown in figure 2, is the “popping” [4] [5] of pixels from the image where pixels may be drawn in one frame, however, calculated not to be a pixel in the next frame, thus objects could seem invisible in one frame and not the next giving you flickering [8]. It’s impossible to eliminate jaggies completely, however, there are three approaches you can do to lessen the effect, one attempt would be to colour in just the bottom square, you could also just colour in the next line above or colour both the squares. [1] [9] [10] [11] [12] [13]. One historical approach that Nvidia talk about is the modification of the screen resolution and if we were to reduce the size of the pixel then this will also decrease the size of the artefact. They also mention that this may not always be feasible as the user may already be using the maximum resolution that their monitor supports [14] [15]. As there seems to be limitations to the screen resolution we will investigate other options available to us that attempt to make the lines appear visually smoother by modifying the pixel and surrounding pixel colour values giving a more blurred picture, however visually more appealing. A drawback to these is that they are computationally expensive as you need to calculate the lighting values before you then consider anti-aliasing.

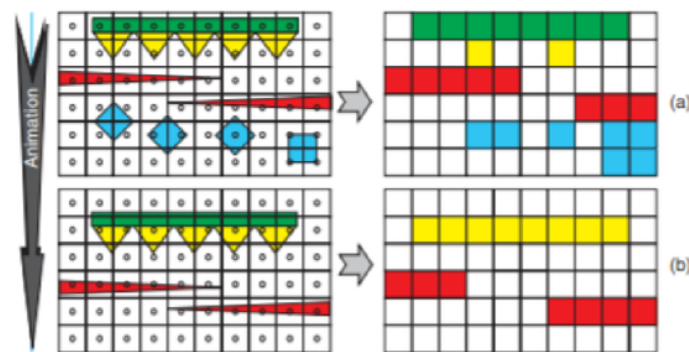


Figure 2: Popping in action, (a) - frame one, (b) - frame two where pixels are not represented correctly and pop off the screen. [5]

Supersampling

Supersampling is a technique used to smooth out the jagged edges, it works by taking an image at much higher resolution than your screen and then downsizes the image to fit to your screen by taking different weighting values from multiple sample points and condensing them into a single pixel [10] [11]. Sampling must be more than twice the maximum frequency of the signal to be sampled for it to work effectively. This is called the Nyquist rate or Nyquist limit after Harry Nyquist [2] [16]. Supersampling has been used in games like Assassin's Creed, AC2 and Brotherhood for the PS4 gaming Console [17]. Supersampling uses kernels to achieve this, kernels define the number and location of samples taken. There are several kernels that can be applied to calculate the values of the pixel that you want with varying levels of results from each of them.

Grid

A grid setup takes a pixel and then splits it up uniformly into sub sections along the x and along the y axis, it then takes sample points from the centre of each sub section to create the final value for that pixel as shown in figure 3. As the image does not pass through the centre point of the pixel then normally that would not be rendered, however, as it is now split into sub divisions we can now use a weighting from each segment and colour the pixel according to the weighting [2], this would give us a light grey pixel as there is more white than black.

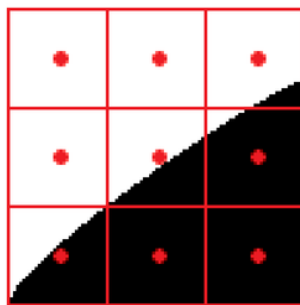


Figure 3: A pixel sub divided in the X and Y co-ordinates

There are many other variants that have been developed, some examples of these will be added to the [Further Reading section](#).

Another one that we will look over however, will be the Quincunx method, which was famously applied by Nvidia to its GeForce3 GPU and claimed to be the first product that could deliver HRAA at a consumer price point [14]

Quincunx

Quincunx works slightly differently where it shifts every other row across half a pixel and then takes weighting from different pixels [18] [19], an adaption to this to remove the need to shift pixels was to take the corner values of the pixel instead, the weighting normally used are 50% for the initial pixel and then $1/8^{\text{th}}$ of each corresponding corner pixels [2] [19]. The process is explained in figure 3 and the outcome is explained in figure 4, where the result is comparable to doing 4x Supersampling. The application that we will be developing uses a method like Quincunx and we will go over that next.

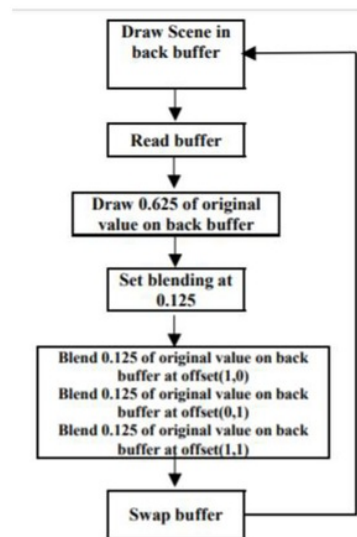


Figure 4: The Quincunx process.

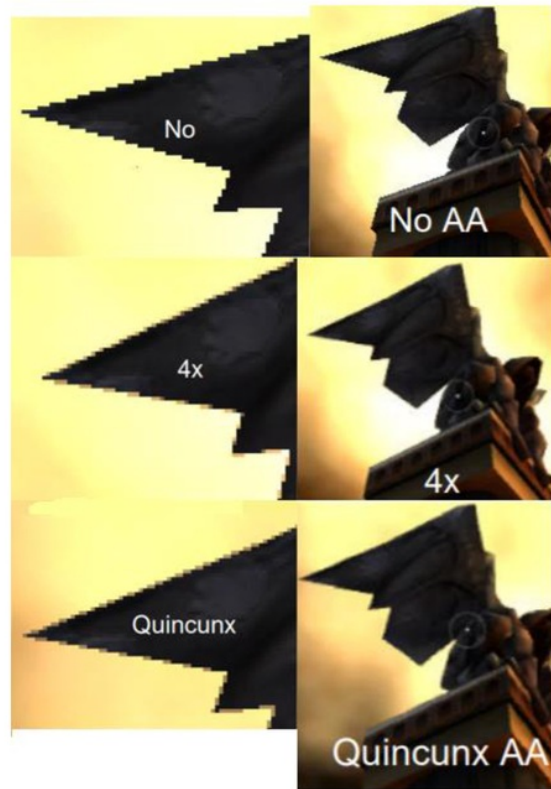


Figure 5: The result of applying Quincunx compared to 4x Supersampling [14]

CUDA application

An application will be created that blurs an image and uses the method like Quincunx developed by Richard Cant [19], this takes 5 sample points with a different weighting. The pixel we are calculating gets a weighting of $\frac{1}{2}$ whilst the four corner pixels associated with the pixel, we calculated gets a weighting of $\frac{1}{8}$. The sum of these values is then applied to the pixel of interest. An example of this will be shown below.

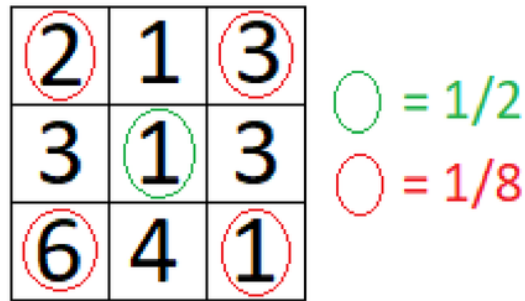


Figure 6: An example filter

If our pixel of interest is the centre pixel and we were to calculate the new pixel value then this would be $(2+3+6+1)/8 + 0.5$ giving a new value of the centre pixel of 2, this would proceed through all the pixels in this format until all the pixels have been covered.

The application will be first analysed on the CPU and then a CUDA kernel will be created to run threads in parallel and then the results will be compared. A multi-pass system was created so we could blur an image further and further until we got a desired effect. We will discuss a little about the test that will be carried out next.

Tests

The tests that will be carried out will be constant on both the CPU and GPU to make things fair, the device that we will be using for testing is an Intel(R) Core(TM) i5-4210M CPU with a clock speed of 2.60GHz and a NVIDIA GeForce 940M graphics card with 2GB of DDR3 memory. 5 tests of different passes will be used, with each of these tests being run 10 times to hopefully eliminate any anomalies. The number of passes for each of the 5 tests will be 1 pass, 10 passes, 20 passes, 30 passes 40 passes and 50 passes. An image of $1024 * 1024$ was used for the blurring process. The GPU will be using a block dimension of 1024,1,1 creating 1024 threads per block and a grid dimension of 1024,3,1 for each RGB value.

CPU Tests

Multiple tests on the CPU were completed that analysed the speed of completion and different blurring levels using multi-pass, the use of multi-pass was also required to highlight any issues with the way the CPU works and how it time slices between applications it needs to process. In figure 7 you can clearly see that this is happening, to give the CPU 10x as much work we would expect the CPU to take approximately twice as long to complete and have a near linear growth, however, in this case it was slightly faster to complete the 10 passes with the remaining passes getting close to a linear growth.

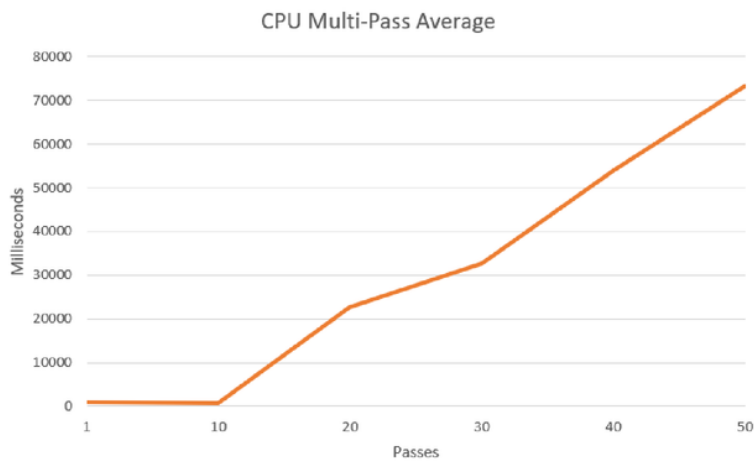


Figure 7: Average time taken to complete number of passes on the CPU

The reason for this is there are a few results where the CPU may complete things far quicker or slower than the previous pass as the CPU is either dealing with other applications that are taking longer or perhaps prioritising other applications over others. A clearer example is shown when we split up each test into their individual tests rather than looking at an average. Looking at the highest and lowest alone there is a gap of 2383.9ms making it over 6x slower to complete that pass. This does give us issues with consistency, if we wanted to know within a rough time window when something will be completed then that window will be much larger due to the spreading of the results.

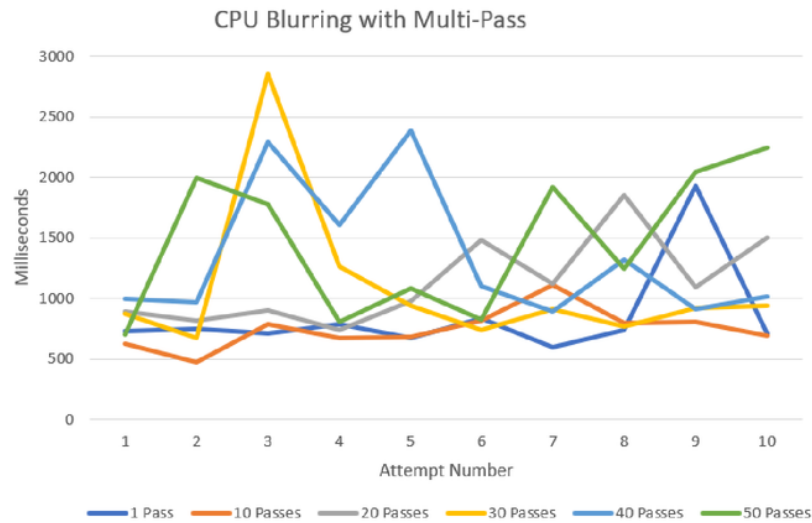


Figure 8: Individual attempts for each pass test

GPU tests

To test on the GPU data needs to be passed from the CPU to the GPU, code is written for the GPU called a kernel which does all the processing via using a setup of grids, blocks and threads. A grid can have many blocks, which can run many threads, each of these threads whilst not able to complete complex tasks can perform simple tasks fast. The key thing with this is that instead of running one loop on the CPU it can run the same loop on multiple threads simultaneously giving a massive speedup in computing data. One thread's responsibility will be to find out it's position within an image, find its corner pixels and sum the values of these in accordance to the weighting we discussed earlier and store that value into place for the final image. Once all the threads have completed the output is then passed back to the CPU to then work on. Whilst incredibly fast inside the kernel there is quite a considerable amount of time spent setting up and transferring data from the CPU to the GPU. The figure below shows the comparison between the CPU and GPU showing both the time it takes working in the kernel and the complete time taken after setting up the GPU and passing data back. An analysis from the graph is that unlike the CPU the GPU is reasonably constant with an almost linear output in both the setup and kernel. This would allow you to be more accurate in calculating how long the application will take to complete. The kernel alone is operating much faster

Page | 11

than CPU however is held back by the setup times, on further inspection the CPU appears faster on a low amount of passes due to the setup time but then becomes much slower the longer the application runs due to the CPU allocating resources elsewhere. Next, we will look at ways that this could possibly be improved.

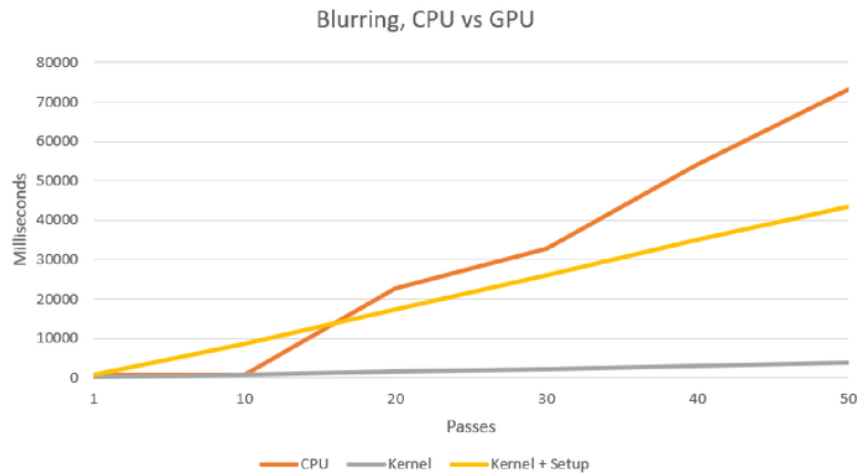


Figure 9: CPU vs GPU timings

Improvements

There are a few ways that you could go about trying to improve this, one way would be to try and get the CPU to prioritise the application, however time slicing will still exist, another possible improvement is to modify the grid, block and thread dimensions, on Nvidia's website [20] it does suggest modifying the block size to try and achieve a high occupancy and after entering the details in their spreadsheet also available on their website it shows that we are already at a high occupancy as seen in figures 10 and 11. To prove this further a test was done that modified the arrangement of the blocks to 64,16,1. This caused a major performance decrease as seen in figure 12.

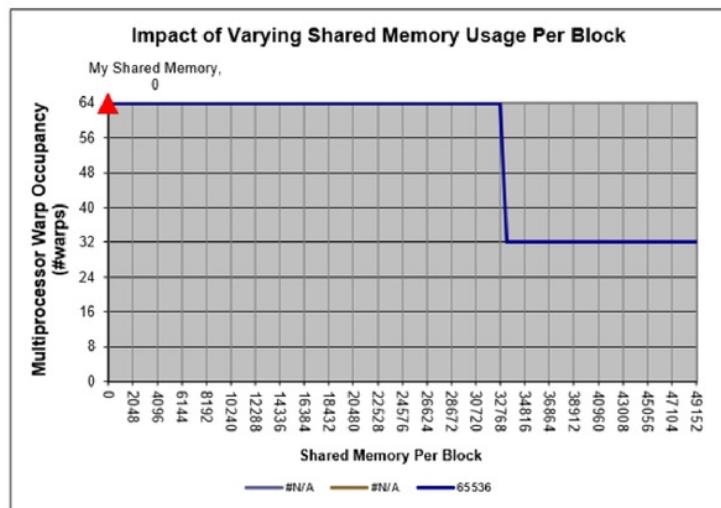
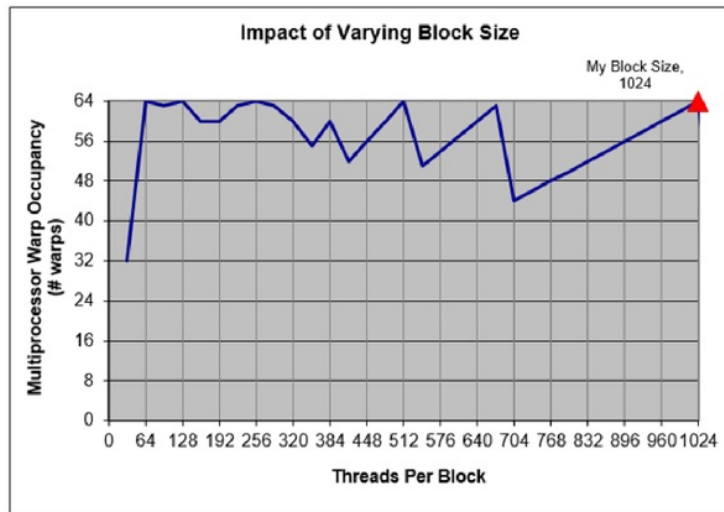


Figure 10: Graphs showing high occupancy

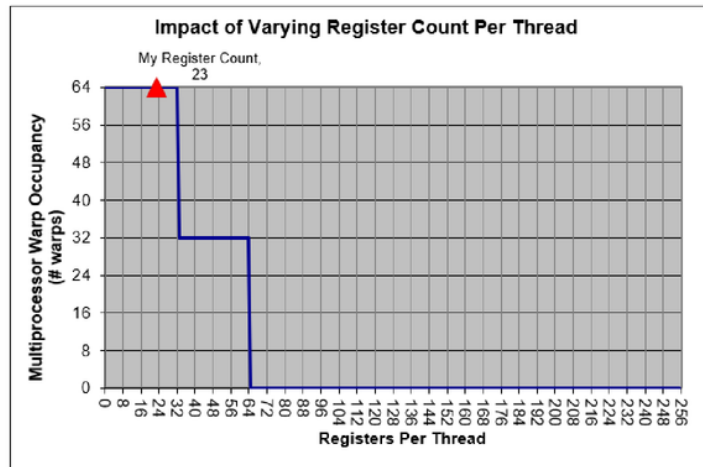


Figure 11: Graphs showing high occupancy

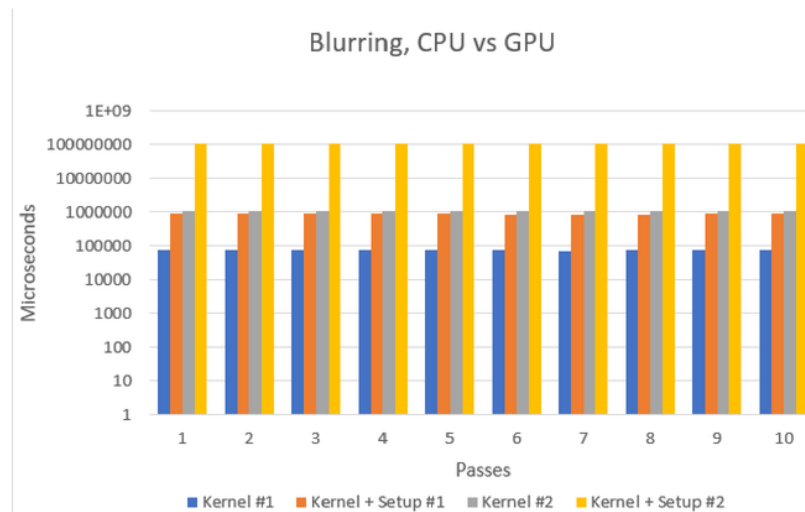


Figure 12: Two kernels with different grid sizes - Kernel 1 (1024,1,1) Kernel 2 (64,16,1)

Conclusion

Aliasing is an issue caused when rendering an image to screen, this can be improved with many techniques like super sampling and Quincunx, however these take time to compute on the CPU, by using CUDA and the graphics card we have a way to pass the information to be calculated by many threads at the same time increasing performance. Considerations need to be made regarding

the block, grid and thread size as incorrectly calculating these can cause the application to run much slower than intended. Also, as there are setup costs when passing data to CUDA then you need to be mindful that if you don't have enough data then it may be more efficient to run on the CPU.

Appendix

Images of Blurring Done by Running the Application



Figure 13: Original image



Figure 14: Multi pass 10x



Figure 15: Multi pass 50x

Excel test data



Cuda.xlsx

References

- [1] H. Freeman, "Computer Processing of Line-Drawing Images," pp. 57-97, 1974.
- [2] T. Akenine-Moller, E. Haines, N. Hoffman, A. Pesce, M. Iwanicki and S. Hillaire, Real-Time Rendering.
- [3] T. Akenine-Moller, "FLIPQUAD: Low-Cost Multisampling," 2002.
- [4] E. Catmull, "A Hidden-Surface Algorithm With Anti-Aliasing," in *Annual Conference on Computer Graphics*, 1978.
- [5] K. Beets and D. Barron, "Super-sampling Anti-aliasing Analyzed," 2000.
- [6] Ventuz, "Anti-Aliasing / Multisampling," [Online]. Available: https://www.ventuz.com/support/help/V5_04/AntiAliasing.html?itm=12. [Accessed 27 11 2018].
- [7] L. e. al, "METHOD AND APPARATUS FOR ANTI-ALIASING IN A GRAPHICS SYSTEM". United States Patent US 6,999,100 B1, 14 Feb 2006.
- [8] J. Jimenez, J. Echevarria. I, T. Sousa and D. Gutierrez, "SMAA: Enhanced Subpixel Morphological Antialiasing," 2012.
- [9] NVidia, "High-Resolution Antialiasing," NVidia, [Online]. Available: <https://www.nvidia.com/attach/151>. [Accessed 21 11 2018].
- [10] T. McREYNOLDS and D. BLYTHE, Advanced Graphics Programming Using OpenGL, 2005.
- [11] S. Buss R, "3d Computer Graphics A Mathematical Introduction with OpenGL," Cambridge university press, 2003.
- [12] J. Foley.D, A. Dam van, S. Feier. K, J. Hughes. F and R. Philips. L, Introduction to Computer Graphics, 1994, pp. 119-120.
- [13] S. Guha, Computer Graphics Through Opengl, 2nd, Ed., CRC Press, 2014, p. 524.
- [14] NVidia, "High-Resolution Antialiasing," Nvidia, [Online]. Available: https://www.nvidia.com/object/feature_hraa.html. [Accessed 21 11 2018].
- [15] D. Hearn and P. Baker. M, Computer Graphics, 1994, p. 171.

- [16] J. Proakis. G and D. Manolakis. G, "Digital Signal Processing," p. 26.
- [17] T. Morgan, "PS4 Pro's new super-sampling tested: big boosts for 1080p users," Eurogamer, 27 2018. [Online]. Available: <https://www.eurogamer.net/articles/digitalfoundry-2018-ps4-pro-super-sampling-tested-big-boosts-for-1080p-users>. [Accessed 22 11 2018].
- [18] P. e. al, "Quincunx Sampling Grid for Starting Array". United States Patent 5,602,391, 11 February 1997.
- [19] R. Cant, N. Chia and D. Al-Dabass, "New Anti-Aliasing and Depth of Field Technique for Games".
- [20] Nvidia, "CUDA Pro Tip: Occupancy API Simplifies Launch Configuration," Nvidia, 2014. [Online]. Available: <https://devblogs.nvidia.com/cuda-pro-tip-occupancy-api-simplifies-launch-configuration/>.
- [21] A. Reshetov, "Morphological Antialiasing".
- [22] Nvidia, "Adaptive Temporal Antialiasing," Nvidia, 2018.

Further Reading

Some other grid-based sampling methods

Real Time Rendering p.140, reference #2

Jittered Grid Super Sampling

Ordered Grid Super Sampling

Rotated Grid Super Sampling

<http://www.x86-secret.com/articles/divers/v5-6000/datasheets/FSAA.pdf>

Morphological Antialiasing which works out where the silhouettes or outlines of an object are and performs antialiasing on them alone rather than the whole scene.

<http://www.iryoku.com/aacourse/downloads/Filtering-Approaches-for-Real-Time-Anti-Aliasing.pdf>

http://www.cs.cmu.edu/afs/cs/academic/class/15869-f11/www/readings/reshetov09_mlaa.pdf

Adaptive Temporal Antialiasing using Ray Tracing

https://research.nvidia.com/sites/default/files/pubs/2018-08_Adaptive-Temporal-Antialiasing/adaptive-temporal-antialiasing-preprint.pdf

High Performance Computing M6X01099

GRADEMARK REPORT

FINAL GRADE

GENERAL COMMENTS

Instructor

85/100

PAGE 1

PAGE 2

PAGE 3

PAGE 4

PAGE 5

PAGE 6

PAGE 7

PAGE 8

PAGE 9

PAGE 10

PAGE 11

PAGE 12

PAGE 13

PAGE 14

PAGE 15

PAGE 16

PAGE 17

PAGE 18

GRADING FORM: SMU GENERIC_GJ

ANDREW HUMPHREYS**85****RESEARCH-INFORMED LITERATURE**

References are in general of a good quality and well chosen. Careful with formatting your references. One or two are incomplete.

6.5**KNOWLEDGE AND UNDERSTANDING**

A good review of relevant literature backed up with appropriate references. Relevant algorithms covered - diagrams supportive.

12.5**ANALYSIS**

The graphs are very helpful and demonstrate speed-ups, etc well. Occupancy and its effect on GPU execution speed is included but the definitions/concepts here could be more clearly discussed.

7.5**PRACTICAL APPLICATION AND DEP.**



Some excellent work here. Evaluation provided for code on both CPU and GPU. Good work Andrew.

55

SKILLS FOR PROFESSIONAL PRAC.



An abstract would be helpful here, though I appreciate the intro acts as a surrogate here. Report well structured. Diagrams and graphs helpful and support arguments well. Good.

3.5