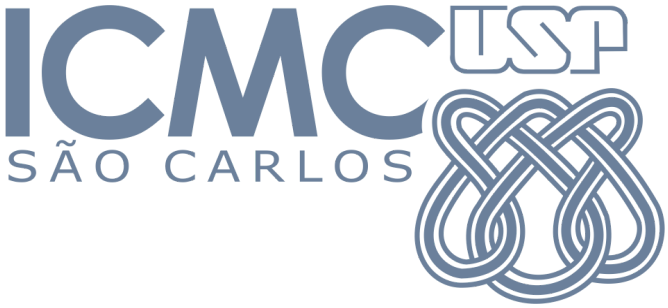

UNIVERSITY OF SÃO PAULO

INSTITUTE OF MATHEMATICAL AND COMPUTER SCIENCES

DEPARTMENT OF COMPUTER SCIENCE

SCC0251 – IMAGE PROCESSING

Alex Sander Ribeiro da Silva	9779350
Danilo Marques Araujo dos Santos	8598670

Implementing and Comparing Anti-aliasing Algorithms – Final Report

Prof. Dr. Moacir Antonelli Ponti

São Carlos

2019

Index:

1. Introduction

Anti-aliasing is a technique designed to add greater realism to a digital image by smoothing jagged edges on curved lines and diagonals. It is largely used in games. Aliasing manifests itself as jagged or stair-stepped lines (otherwise known as jaggies) on edges and objects that should otherwise be smooth.[1]

The objective of this project is to implement and compare different anti-aliasing algorithms regarding their effectiveness. One of the algorithms that we have decided to implement and analyze is the supersampling.

2. Input Images

Any image in the RGB space can be used by these algorithms, but we are mainly aiming at games. Therefore, most of the images are screen captures of the game itself. To provide better comparisons, multiple images of the same game are taken while looking at the exact same spot, avoiding any moving particles and disabling animations if possible. All the game settings are also maintained, except resolution and built-in AA.

Usually, three to five images are used per game:

- One at the target resolution and without any AA, either as a base for comparison or input for some of our algorithms.
- One at the target resolution and with built-in FXAA, if available, for comparison.
- One at the target resolution and built-in 4x MSAA, if available, for comparison.
- One at the target resolution and both built-in FXAA and MSAA, if available (which is rare), for comparison.
- One at a higher resolution, to be used as input for some of our algorithms.

[1] "What is Antialiasing? - Definition from Techopedia." https://www.techopedia.com/definition/1950/antialiasing. Accessed 27 May. 2019.

3. Algorithms Studied

3.1. Supersampling

Supersampling (SS) or postfiltering is the process by which aliasing effects in graphics are reduced by increasing the frequency of the sampling grid and then averaging the results down. This process means calculating a virtual image at a higher spatial resolution than the frame store resolution and then averaging down to the final resolution. It is called post-filtering as the filtering is carried out after sampling.[2]

The two steps in the post-filtering process are[3]:

1. Sample the scene at n times the display resolution. For example, suppose the display resolution is 512x512. Sampling at three times the width and three times the height of the display resolution would yield 1536x1536 samples.

2. The color of each pixel in the rendered image will be an average of several samples. For example, if sampling were performed at three times the width and three times the height of the display resolution, then a pixel's color would be an average of nine samples. A filter provides the weights used to compute the average.

We are implementing a simulated version of the Supersampling algorithm. It's "simulated" because we are only implementing the second step of the process, which is the image processing part, and using images already sampled at higher resolutions as the input. This is done mainly because we aim to achieve anti-aliasing on already existent images, which could then be applied to, for example, games that do not have that feature available.

[2] "Antialiasing methods." https://web.cs.wpi.edu/~matt/courses/cs563/talks/antialiasing/methods.html. Accessed 28 May. 2019.

[3] "Overview of Aliasing in Computer Graphics: Part 2." 4 Oct. 1999, https://www.siggraph.org/education/materials/HyperGraph/aliasing/alias2b.htm. Accessed 28 May. 2019.

3.2. Multisample anti-aliasing

Multisample anti-aliasing (MSAA) is a type of spatial anti-aliasing, a technique used in computer graphics to improve image quality. "multisampling" refers to a specific optimization of supersampling.

In multisample anti-aliasing, if any of the multi sample locations in a pixel is covered by the triangle being rendered, a shading computation must be performed for that triangle. However this calculation only needs to be performed once for the whole pixel regardless of how many sample positions are covered; the result of the shading calculation is simply applied to all of the relevant multi sample locations.[4]

With MSAA, instead of sampling one pixel on a much larger scale, two or more adjacent pixels are sampled together while rendering an image at its intended size. Because multiple pixels are being sample together, coverage points can be shared between them.[5]

In this project, since we are trying to implement MSAA using image processing, we are instead using our supersampling algorithm as a base line, but only applying it over a small area (usually 3x3) around the edges. The rest of the image remains at the original resolution.

Unfortunately, that means two things: this method requires two images, one at the target resolution, and another at a higher resolution; and it also requires an edge detection algorithm to decide where the supersampling should be applied or not.

The easiest option to solve these problems would be rendering the screen, at the same time, in two different resolutions. This would have a huge cost in performance and require some changes in drivers.

A more efficient approach would be to render pixels that are part of one edge at higher resolution, and others at lower resolution. Theoretically, this could result in good performance, but it could only be done if we had knowledge of where edges are located *before* the image has been rendered, which is against the goal of this project.

---

[4] "Multisample anti-aliasing - Wikipedia." https://en.wikipedia.org/wiki/Multisample_anti-aliasing. Accessed 24 Jun. 2019.

[5] "How To Choose the Right Anti-Aliasing Mode for Your GPU - Tested." 22 Oct. 2010, https://www.tested.com/tech/pcs/1194-how-to-choose-the-right-anti-aliasing-mode-for-your-gpu/. Accessed 24 Jun. 2019.

3.3. Fast approximate anti-aliasing

Fast approximate anti-aliasing (FXAA) is the least demanding type of anti-aliasing. Rather than running complex calculation depending on the geometry and colors displayed, FXAA simply applies extensive blurring to obscure the jagged edges. The end result is unnoticeable performance impact but a generally blurrier image.[6] It is ideal for low-end computers.

The downsides are that textures may not appear as sharp if they are included in the edge detection, and it must be applied before rendering the HUD elements of a game, lest it affect them too.[7]

Since we have no information from the renderer, only from the final image, we decided to implement the FXAA as follows:

1. Find all edges contained in the image. This is done using a simple edge detection convolution matrix and threshold filter, but expanded to include a small area outside surround each pixel that is considered to be an edge.

2. Smooth the edges: smoothing is applied as a per-pixel effect. That is, there is no explicit representation of the edges. A simple convolution filter with a blurring effect is applied to each pixel identified as an edge or surrounding one.

.

[6] "What is Anti-Aliasing? [Simple Explanation] - GamingScan." 9 Apr. 2019, https://www.gamingscan.com/what-is-anti-aliasing/. Accessed 24 Jun. 2019.

[7] "Fast approximate anti-aliasing - Wikipedia." https://en.wikipedia.org/wiki/Fast_approximate_anti-aliasing. Accessed 24 Jun. 2019.

4. Results obtained

4.1. Supersampling

We've coded our own simulation of a supersampling algorithm and have applied it to a 1440p image of a game, generating a 720p supersampled image (CSGO) in 1 minute and 37 seconds.

The figures 4.1.1., 4.1.2. and 4.1.3. show the original image (without anti-aliasing), the image generated by our algorithm and the image generated with the Fast Approximate Anti-Aliasing algorithm (FXAA) provided by the game:

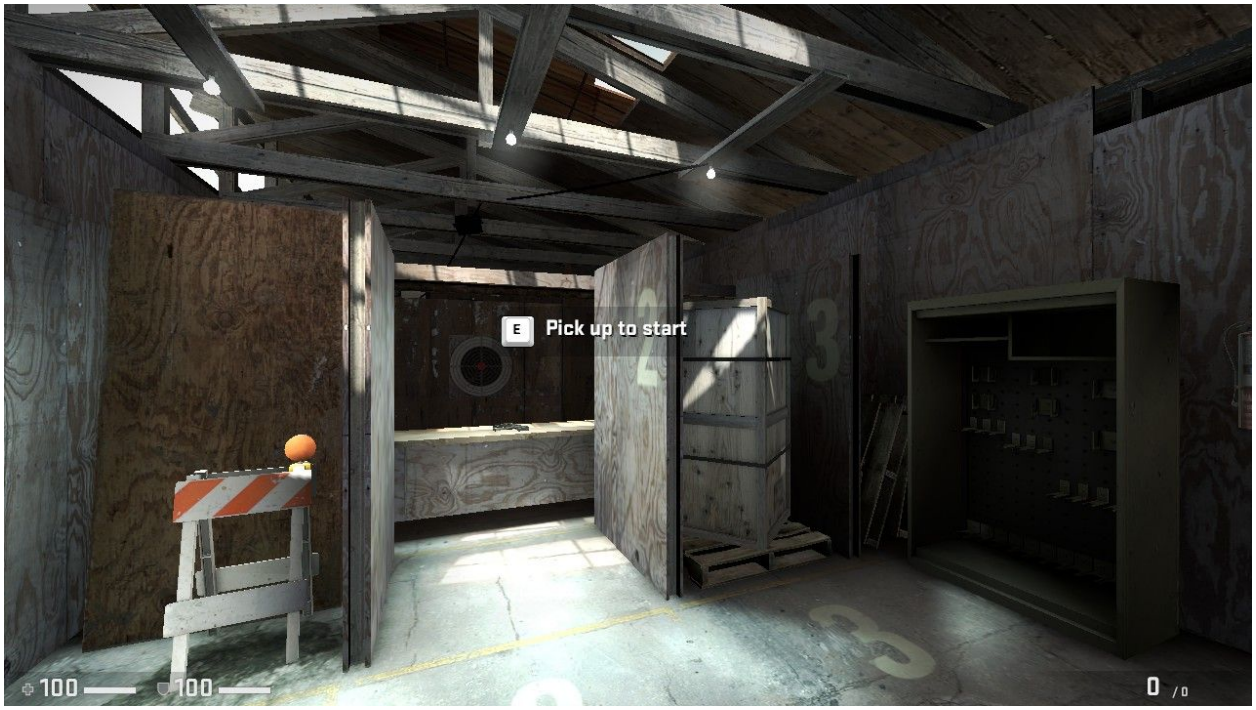

Figure 4.1.1. – original image. CSGO at 720p, no AA, captured from the game.

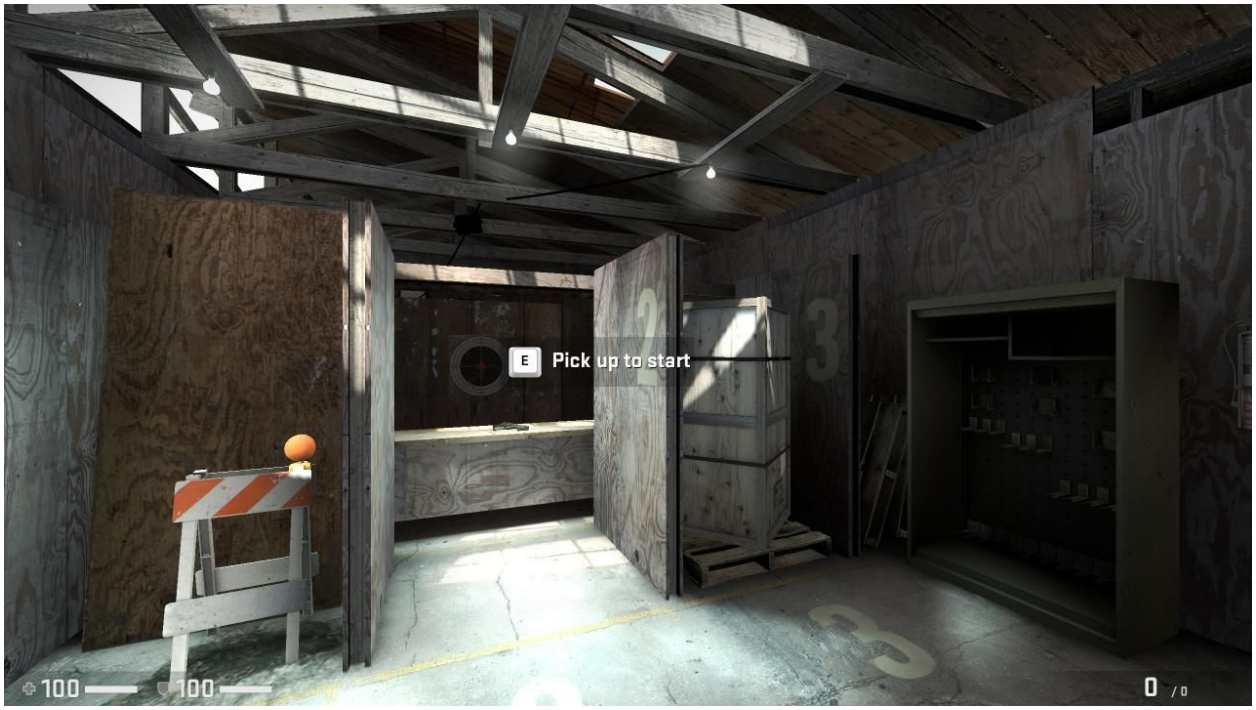

Figure 4.1.2. – image generated by using figure 4.1.1 as input for our SS algorithm (RMSE = 8.45).

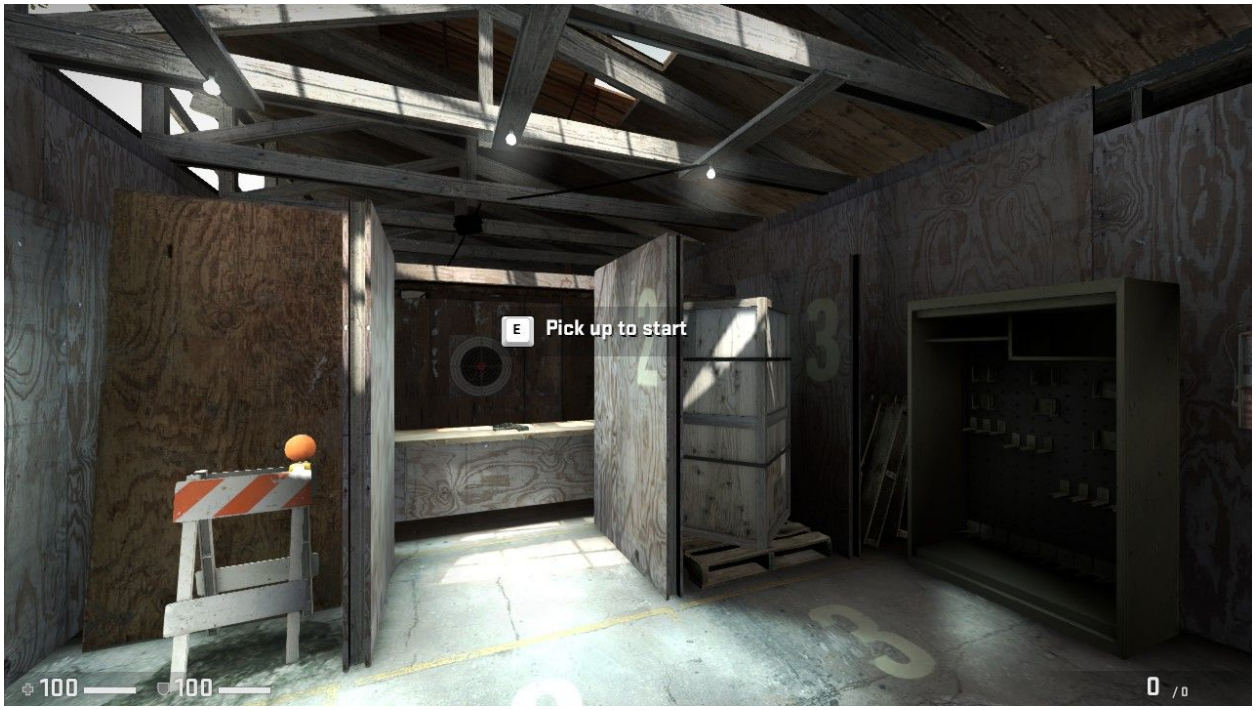

Figure 4.1.3. – comparison image. CSGO at 720p, built-in FXAA, captured from the game (RMSE = 5.012).

Of course, these images have a limited resolution on this page, but some differences can already be seen. The full sized images, as well as the results of two other games, can be found at https://github.com/dsantos-1/DIP_FinalProject/tree/master/Images.

4.2. Multisample anti-aliasing

We've coded our own simulation of a MSAA algorithm and have applied it to a 1440p image of a game, generating a 720p image (Paladins) in 46.8 seconds.

Unfortunately, we couldn't meet the requirement of producing two near identical images with two different resolutions. As a result, artifacts can be observed in the final image where there are moving objects or HUD.

The figures 4.2.1. and 4.2.2. show the original image (without anti-aliasing), and the image generated with our implementation of the MSAA algorithm:

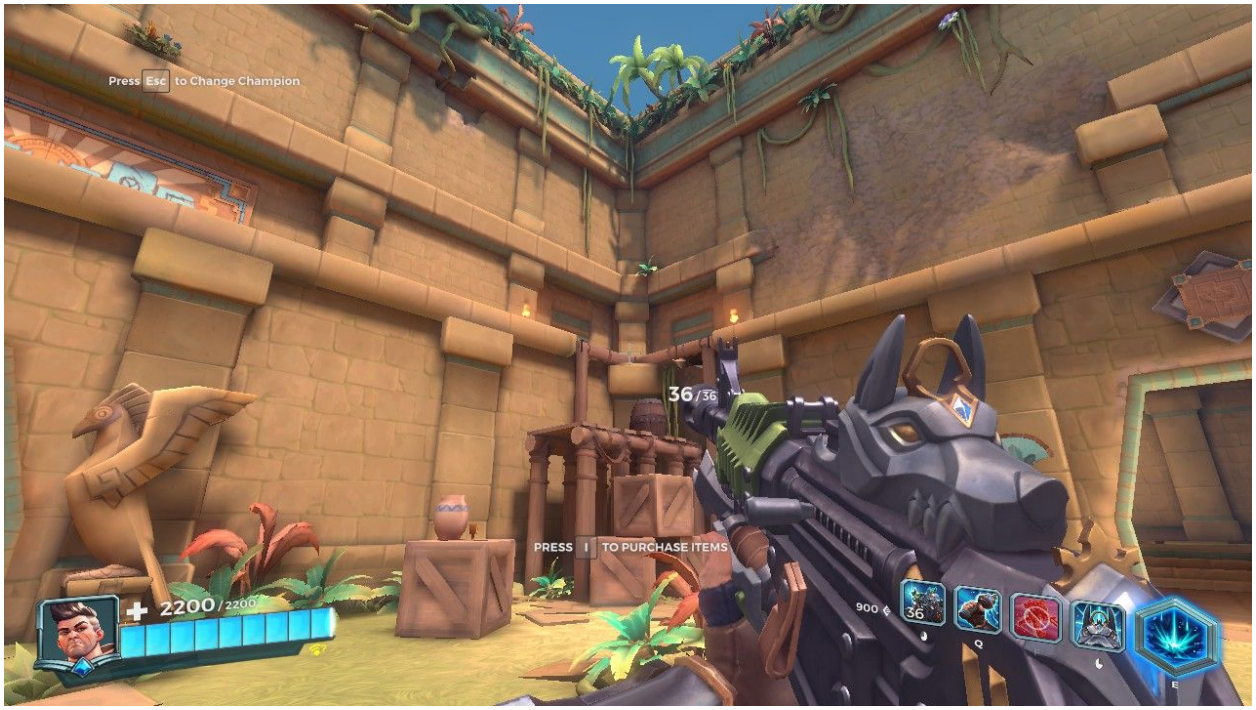

Figure 4.2.1. – original image. Paladins at 720p, no AA, captured from the game.

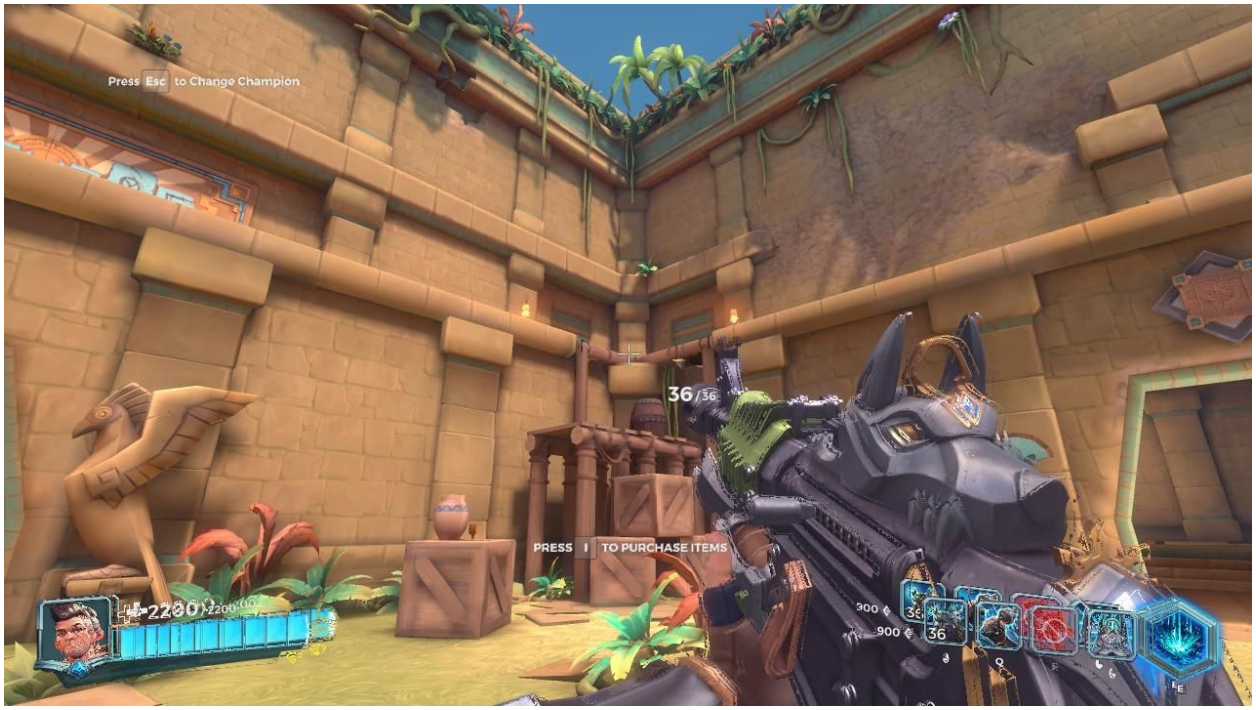

Figure 4.2.2. – image generated by using figure 4.2.1 as input for our MSAA algorithm (RMSE = 7.957).

The full sized images, as well as the results of two other games, can be found at https://github.com/dsantos-1/DIP_FinalProject/tree/master/Images.

4.3. Fast approximate anti-aliasing

We've coded our own simulation of a FXAA algorithm and have applied it to a 720p image of a game, generating a 720p image (Minecraft) in 44.8 seconds.

The figures 4.3.1. and 4.3.2. show the original image (without anti-aliasing), and the image generated with our implementation of the FXAA algorithm:

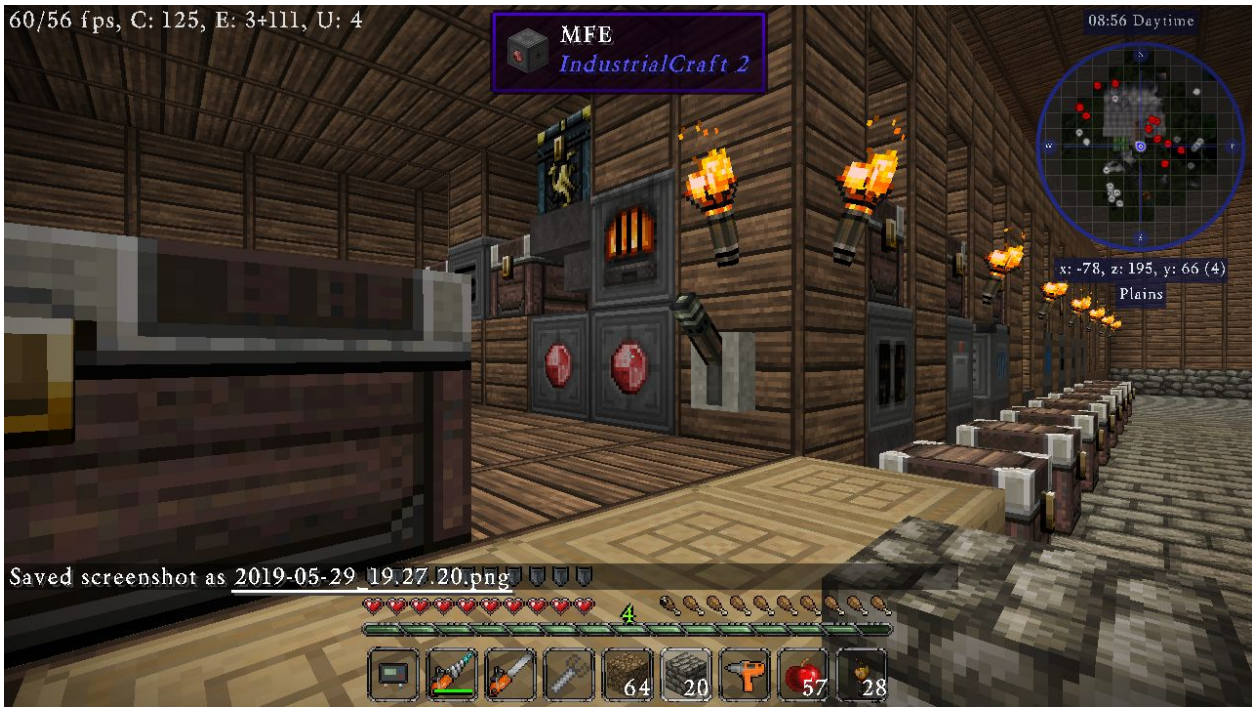

Figure 4.3.1. – original image. Minecraft at 720p, no AA, captured from the game.

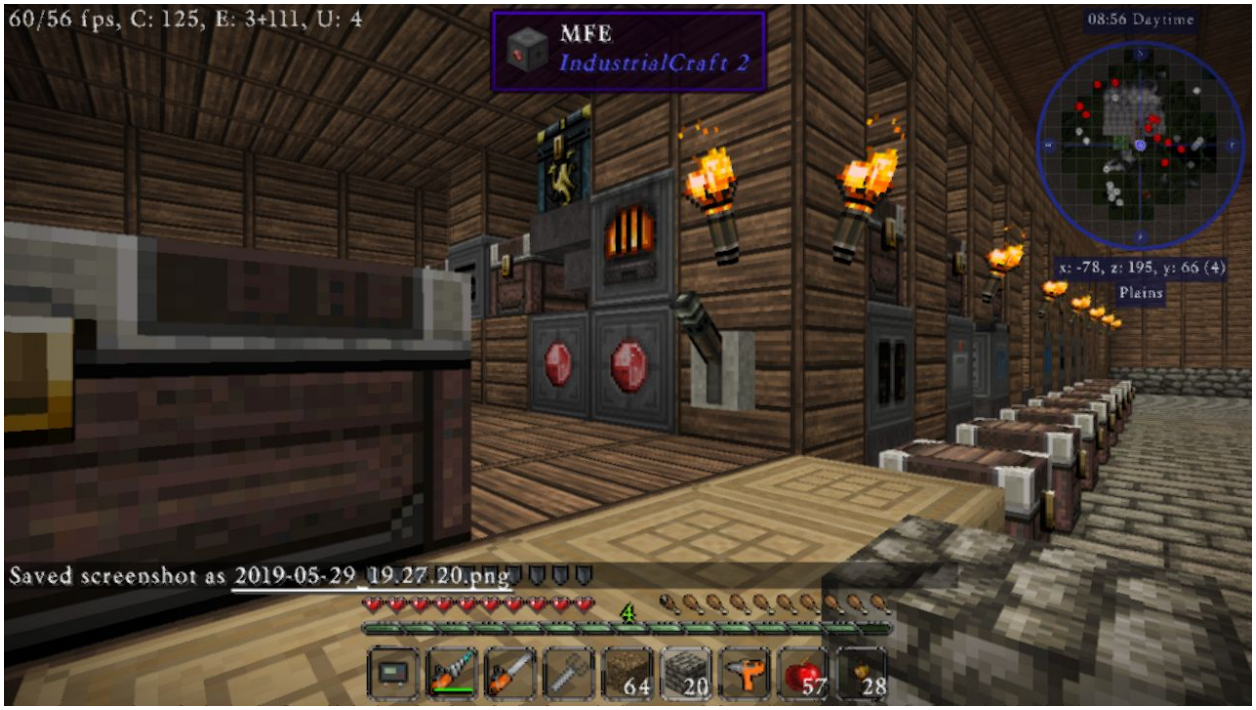

Figure 4.3.2. – image generated by using figure 4.3.1 as input for our FXAA algorithm (RMSE = 8.939)

The full sized images, as well as the results of two other games, can be found at https://github.com/dsantos-1/DIP_FinalProject/tree/master/Images.

5. Conclusions

The supersampling algorithm has achieved the best results in terms of image quality, only arguably losing to the MSAA in specific scenarios where preserving sharpness of non-edges is a priority. However, it's also the most expensive option, except when considering a flaw in the MSAA, as described below.

Despite achieving promising results in some regions of the image, using the MSAA through image processing is impractical in terms of performance and compatibility. While it appears to have done well, finishing in less than half the time the supersampling did, when you take the extra time required to render *TWO* images at once, one for the target resolution and another for the supersampled resolution, this algorithm becomes unappealing. When taking compatibility into consideration, which translates to how hard it is to produce two identical frames with moving objects and camera around, it becomes clear that MSAA is not a good option for image processing based anti-aliasing.

The FXAA algorithm is fast too, albeit a bit slower compared to the MSAA. It still managed to finish in about half the time it took for the supersampling to complete. Also, it does not require a higher resolution image at all, just one sample at the target resolution, which definitely makes it the fastest algorithm here, and also the most accessible. In terms of image quality, however, it's the worst. If we used a more complex version of this same algorithm, it wouldn't be so far behind, but some of the performance would be lost too.

In the end, it comes down to whether the supersampling is fast enough for the purposes it will be used for, and if you have access to a higher resolution image than the target one. A more optimized FXAA algorithm would be the next option either requirement is not met, and the MSAA does not seem to be viable at all without extra resources from the renderer.