Evaluation of Antialiasing Techniques on Mobile Devices

Tiago Manuel Videira Almeida

***Abstract*— The adoption of mobile devices such as smartphones has been massive, people use their mobile devices to play video-games, view virtual worlds in VR and experience augmented reality. In the field of real-time rendering, antialiasing is still an open problem, there have been a lot of recent advances, especially in post-processing and temporal antialiasing. We found that mobile antialiasing research is very scarce, MSAA is the most used solution in the last decade. Our work brings the state of the art of antialiasing techniques from desktop to mobile. We have created an Android application to test four antialiasing techniques on two scenarios and analyzed their performance and image quality. The results show that MSAA and FXAA provide the fastest execution and SMAA obtains the best image quality, overall the most balanced technique is MSAA. The results on mobile are similar to desktop, except MSAA which obtained better performance due to this technique benefiting from tiled-based rendering architecture present on mobile GPUs.**

I. INTRODUCTION

The adoption of mobile devices in the last years has been massive. In 2016 the mobile web usage was superior than desktop, also last year mobile game revenue has passed desktop and consoles.

The capabilities of mobile devices have improved over the past few years. The hardware is better, the displays have high pixel density allowing the visualization of high-quality images. The software has also been upgraded for better performance, better designed and secure. New technologies like virtual reality and augmented reality which are very hardware intensive are being run on mobile devices.

Aliasing is one of the major problems in graphics rendering. Pixel quality is important for real-time applications, noticeable artifacts can break immersion, especially in virtual reality. Pixel quality can be a quality differentiator that is noticed by the users.

In the last 7 years, there have been great advancements in graphics rendering for desktop, especially in the field of antialiasing. Unfortunately, some antialiasing solutions are not efficient on mobile devices, due to their architecture and constraints. Research on antialiasing for mobile is very scarce, the most used antialiasing technique on mobile is multisampling antialiasing (MSAA). On desktop and consoles, a variety of antialiasing techniques are being used in graphics applications such as games, these techniques so far have not been tested on mobile devices. There is no research of these new antialiasing algorithms on mobile devices.

The main objective of this work is testing and comparing state of the art antialiasing solutions on mobile. This work consisted of creating a 3D Android application to analyze how each antialiasing technique behave. The techniques that we considered relevant to study were:

- Multisample Antialiasing (MSAA)
- Fast approximate Antialiasing (FXAA)
- Subpixel Morphological Antialiasing (SMAA)
- Temporal Antialiasing (TAA)

This study purpose is to answer the following questions:

- Which antialiasing techniques are faster?
- Which one obtains the best image quality?
- What else should be explored on mobile antialiasing?

Our tests covered multiple configurations such as 3D scenes and resolutions and collected behavior data for analysis. We had a partnership with Samsung R&D UK, who provided us support from their research team and a Samsung Galaxy S8 device to run the tests.

II. RELATED WORK

Supersampling antialiasing (SSAA) and Multisample antialiasing (MSAA) are techniques that mitigate aliasing in the graphics pipeline. Supersampling is simple to implement in a rendering engine, it works by rendering to a higher resolution than the screen, normally a power of two (2^x) larger, then that image is downsampled into the intended output resolution[1]. Supersampling obtains high-quality images, the disadvantage of this technique is its performance. Because the rendering resolution increases, the fragment shader sampling rate is also increased. The fragment shader normally does a lot of calculations such as lightning and texture accesses, with supersampling these operations are executed at a higher rate, consuming more resources (see figure 1). The memory consumption also increases, because the size of the color buffer and z-buffer are increased[1].

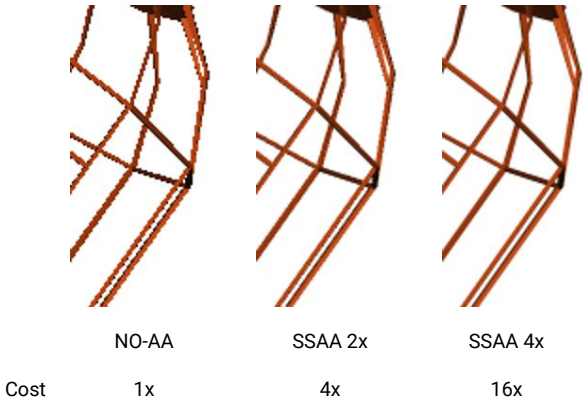

Fig. 1. Zoomed image results of a 3D scene rendered with supersampling using different number of samples. While the image quality increases, the rendering cost also increase.

> "A multisampling algorithm takes more than one sample per pixel in a single pass, and shares some computations among the samples." [2]

Aliasing is most visible on the edges of triangles, this is known as spatial or geometric aliasing, multisampling mitigates this form of aliasing. Multisampling is performed on the GPU, it is supported by most of the GPU manufacturers.

The main difference between multisampling and supersampling is that the fragment shader is executed only once per pixel. Coverage and occlusion tests are performed at a higher rate. The coverage test is executed with N sampling points inside the pixel, these are called coverage samples or subsamples, N is the multisampling factor. Figure 2 shows an example of the samples in 4x MSAA. The fragment shader

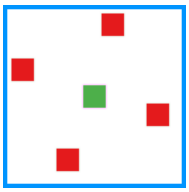

Fig. 2. Example MSAA sampling pattern of a pixel, the red squares indicate the position of coverage and occlusion samples (also called subsamples), the green square is the fragment shader sample position.

result is stored in each of the covered samples. The color buffer size must be larger to hold the shader result in each of the covered subsamples, similar to what happens with the Z-Buffer.

The color buffer has to be downsampled similarly to what is done in supersampling. In MSAA this process is called *resolve*, the most common method is to average the subsample colors to obtain the antialiased pixel color[1].

Post-processing is the process of applying effects to an image. Examples of such effects are ambient occlusion, depth of field and bloom. Antialiasing can also be performed as a post-processing step. This type of antialiasing has gained popularity since the appearance of Morphological Antialiasing (MLAA) developed by Intel[3]. Two post-processing antialiasing techniques are explained in this chapter: fast approximate antialiasing (FXAA) and sub-pixel morphological antialiasing (SMAA). One of the advantages of post-processing antialiasing algorithms is that the memory and processing cost on desktops is normally lower than the techniques described in the previous chapter. This is because post-processing techniques only use the information of the frame, unlike MSAA which increases the number of samples. The same cannot be said for mobile devices, due to their architecture MSAA 4x can be executed with minimal performance drop.

Fast approximate antialiasing (FXAA) is an open-source technique developed by Nvidia[4], which consists on finding the edges in an image and applying a filter to smooth them. FXAA takes as input an image and returns an antialiased version of the image. It is designed to run on a single pass on the fragment shader.

Enhanced Subpixel Morphological Antialiasing (SMAA) is a shader based antialiasing technique, it is a work made upon other technique called Morphological Antialiasing (MLAA) developed by Intel[5]. SMAA improves some areas of MLAA and goes even further with new features like improved edge detection, sharper geometric features and diagonal edges handling [3].

Temporal antialiasing (TAA) is the type of antialiasing techniques, it improves both geometric and temporal aliasing artifacts. TAA displays a motion blur effect and prevents the artifact of flickering pixels[6]. Lately, this technique has gained a lot of attention for its film-like features, it has been implemented in Unreal Engine and other game engines[7]. Unlike geometry antialiasing which increasing samples spatially, temporal AA spreads samples over time.

The goal of this thesis is to evaluate the antialiasing techniques on the mobile environment, mobile architecture differs from the desktop as there are some constraints that desktops do not have. Modern tiled architectures are divided into two steps: geometry processing and fragment processing[8]. First, all the geometry processing is done for the scene, producing the information for the pixel shading phase. Also in this step, the screen is divided into small tiles. Tiles can have different sizes, the Mali GPU uses 16x16 pixel tiles. In Qualcomm mobile GPUs the number of tiles is calculated by dividing the final framebuffer size by GMEM (specialized graphics memory). PowerVR GPUs have a more efficient tiling algorithm, they generate their tile list only where the geometry cover area of the tile, see figure 3. In the fragment

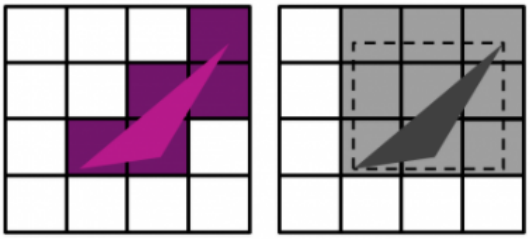

Fig. 3. PowerVR perfect tiling algorithm on the left and bounding box tiling on the right.

processing phase, each GPU core processes one tile at a time, until all pixel colors are calculated, then the tile is saved in RAM, this is done until all tiles are complete.

The tiled rendering architecture is transparent to software developers, as it is done on the GPU, there are many advantages. Because tiles are small, the working set memory of the fragment shader fits on local GPU memory, without needing to access RAM until the tile is completed. A good improvement over immediate rendering because accessing RAM is slow and uses a lot of battery on mobile devices. Another advantage is that since tiles are small, it allows storing extra samples for multisample antialiasing up to 16x, gaining image quality at a very low cost, the ARM Mali GPU contains this technology.

III. METHODOLOGY

To test the four antialiasing techniques we created an Android application to run the test cases and collected performance and image quality metrics. The architecture diagram of the application is shown in figure 4. The technologies that were part of the 3D test application were Android Java API, Java Native Interface (JNI), OpenGL ES version 3.2, OpenGL Mathematics (GLM) and the smartphone that ran our tests was a Samsung Galaxy S8. The implementation of the antialiasing techniques was built with C++ OpenGL ES APIs and open source code from the algorithm authors. We used two rendering passes, first pass for rendering the scene and a second pass that executes different code depending on the antialiasing mode. Each pass has a vertex shader and fragment shader, on the first pass the vertex shader transforms the geometry with the model-view-perspective matrix, the geometry on model space is transformed to clip space. The fragment shader performs Phong lightning with one light source and textures the materials of the scene. The second pass the geometry is a full-size quad that fills the window and the fragment shader code changes accordingly to the antialiasing technique selected.

The 3D scenes that we use for our tests are important to find the strengths and weaknesses of the antialiasing techniques. The scenes should have a lot of thin geometry and diagonals, high vertex count and textures, these features are prone to aliasing. Figure 5 shows the two scenes that were used in the application. Crytek Sponza is a 3D scene with very detailed textures, high triangle count and curved geometry. Post-processing antialiasing is known to use blurring as a way to mitigate aliasing with the downside of also filtering texture details. Sponza model is a good scene to analyze how the techniques perform on textured models. Hairball mesh has a huge number of polygons with very fine geometry, perfect to analyze spatial aliasing. Both scenes are really demanding on mobile hardware, the number of vertices is 153635 for Crytek Sponza and 1032654 on Hairball. Crytek Sponza average FPS is 60, that is the limit FPS value on the Android platform to minimize battery usage, the Hairball model runs at average 35 FPS.

To compare the antialiasing techniques we used various metrics extracted from the application or from the smartphone. The test cases vary on three aspects: 3D scene used, resolution and antialiasing technique used. We used different resolutions: Native (2220x1080) and 720P (1280x720). In each test case, we collected performance and image quality metrics. The performance metrics were collected directly on the application and by using the Snapdragon Profiler. Frames per second (FPS) metric was calculated in the application and sent to output by logs.

The GPU shader processing metrics considered are:

- % Shaders busy
- % Time ALU working
- % Time EFU working
- % Time shading fragments
- Number of fragment instructions

The % Shaders busy indicate the percentage of time that all the shader cores are busy. A technique that uses all the shaders might be an indicator that it is performing well. The other metrics are important to measure ALU operations, EFU which are operations like square root, exponentiation, sine, cosine, among others. The percentage of time shading fragments is a good way to know if the technique is spending more time in the fragment shader or in the vertex shader.

The GPU memory stats metrics are also important to take note because in tiled architectures memory can be a bottleneck that can affect performance. The GPU memory stats that we collected are:

- Average Bytes / Fragment
- Read Total (Bytes/Second)
- Texture Memory Read BW (Bytes/Second)

These metrics provide a good overview of the memory that is transferred from external memory and textures. GPU temperature was recorded during the tests to know if there are relevant temperature differences between the techniques. To measure image quality we captured images of the application running on the various antialiasing modes. We used the Structural Similarity Index (SSIM) to measure the quality of the images produced. We calculated the SSIM between a reference image, supersampled 4x, and each of the images produced by the antialiasing techniques.

IV. RESULTS

In this section, we present and discuss the results obtained during the execution of the various tests with the objective of studying each antialiasing technique's performance and image quality.

A. PERFORMANCE

The mean FPS of each technique was collected from the application in both scenes, the Crytek Sponza FPS results are displayed in figure 6. On this scene the mean FPS value without antialiasing was 60, MSAA 4x and FXAA obtained the best FPS results with the mean value of 59. SMAA variants obtained lower values, 43 FPS for 1x and 31 FPS on the T2x. MSAA performed slightly better than FXAA with the minimum FPS of 51 while FXAA minimum was 47 FPS. Temporal AA is marked in red because the implementation is not ghosting-free, some changes were needed to improve its quality. These changes on TAA would affect its performance and decrease FPS. Our TAA version obtained an average of 55 FPS, even if the implementation is incomplete, we included the results as it can be meaningful for comparison.

The mean FPS results on the Hairball scene can be seen in figure 7. On this test case, FXAA performed better than MSAA 4x, 33 FPS while MSAA 4x got 19 FPS, a huge difference considering that without applying AA the frame rate was 35. SMAA 1x and T2x executed at 27 and 20 FPS respectively, finally TAA achieved an average of 25 FPS. FXAA overhead was the lowest in both scenes, with the cost of 1 FPS on Crytek Sponza and 2 FPS on Hairball. MSAA performed well on Sponza but on Hairball the FPS was much lower, only 19 FPS while FXAA got 33 FPS.

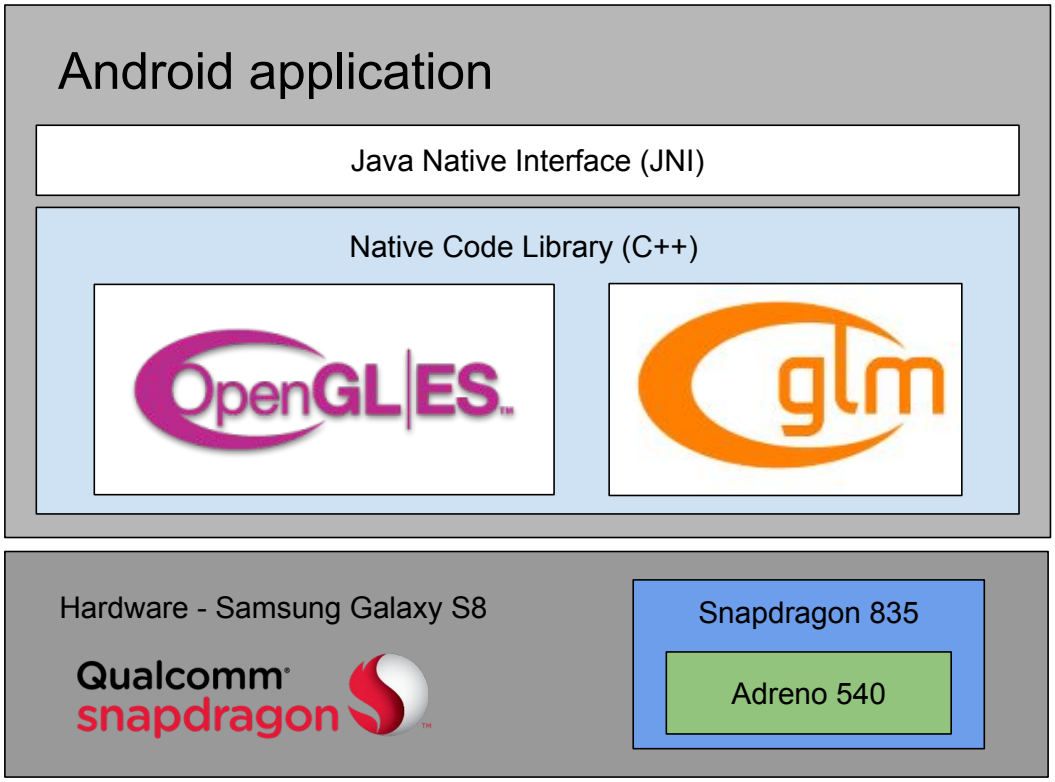

Fig. 4. Test application architecture diagram.

Fig. 5. Example render of the 3D scenes used, on the left Crytek Sponza and Hairball on the right.

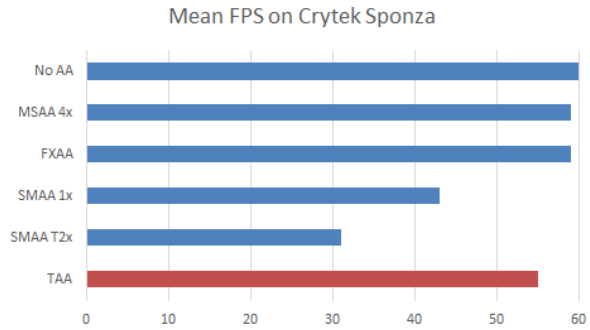

Fig. 6. Chart comparing the mean FPS of the antialiasing techniques on the Crytek Sponza scene.

The low FPS result on Hairball can be explained by the way MSAA works, the coverage and depth tests are executed at a rate of four times higher, in addition to this, hairball has a much higher number of vertices and more complex geometry making the GPU spend a huge portion of its work time in the rasterizer and depth test stages of the OpenGL pipeline. We can also observe that in the hairball scene the post-processing AA techniques performed better than MSAA, this happens because post-processing is not affected as much by scene complexity, unlike oversampling techniques like MSAA and SSAA.

We collected GPU shader processing metrics using the Snapdragon profiler for both scenes. Table I contains the results of the both scenes. The percentage of shaders busy metric indicates the percentage of time in which all the shader cores were active during the execution of the tests. A lower percentage indicates a lower usage of GPU processing resources, for example, if we get a value of 75%, it means that 25% of the time the shader cores were idle waiting for more work.

MSAA 4x performed better in this metric on Sponza, with an average of 54.9%, this happens because MSAA does not have a fragment shader pass thus less work is given to the shaders cores in this technique, this is also confirmed by the metric % Time Shading Fragments. The Shaders Busy results of FXAA, SMAA and TAA appear to be dependent on the fragment shader complexity of each technique, however, between FXAA and SMAA it was expected that FXAA

TABLE I

MEAN VALUES OF GPU SHADER PROCESSING METRICS COLLECTED.

	No AA	MSAA 4x	FXAA	SMAA 1x	SMAA T2x	TAA
			Crytek Sponza			
% Shaders Busy	76.1	54.9	83.2	81.7	82.0	73.8
% Time ALUs Working	37.7	37.6	37.2	32.3	29.3	22.8
% Time EFUs Working	5.0	5.1	3.5	2.3	3.2	5.1
% Time Shading Fragments	71.7	70.4	80.1	81.7	88.5	81.5
Fragment Instructions ($x10^9$) / Second	21.0	20.3	33.6	33.3	30.0	23.8
			Hairball			
% Shaders Busy	91.4	79.3	92.1	92.3	87.1	85.6
% Time ALUs Working	53.6	43.6	53.6	50.7	49.5	52.3
% Time EFUs Working	3.0	3.3	2.9	2.6	3.8	4.4
% Time Shading Fragments	37.4	38.7	41.0	48.8	53.2	44.8
Fragment Instructions ($x10^9$) / Second	20.7	21.0	23.6	27.1	23.9	18.9

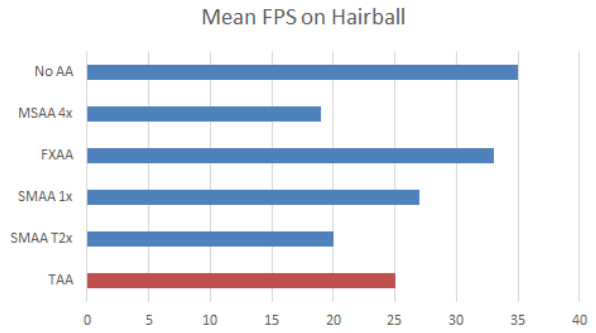

Fig. 7. Chart comparing the mean FPS of the antialiasing techniques on the Hairball scene.

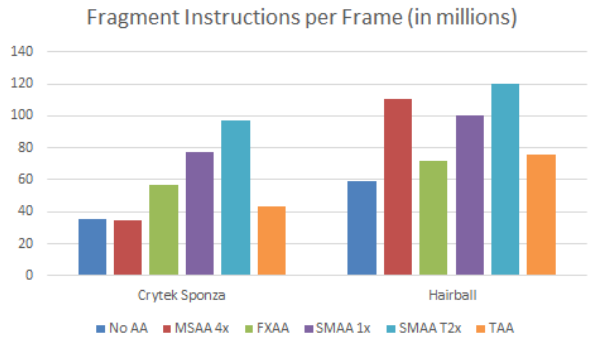

Fig. 8. Chart comparing the fragment instructions per frame.

would perform better in this metric because its shader has fewer operations and is more optimized than SMAA.

In the metric % Time ALUs Working, MSAA and FXAA obtained higher values, similar to the previous metric we believe that there are other factors influencing the results, time-based metrics are difficult to compare when the number of FPS between techniques has a huge disparity.

When comparing the results from both scenes, we can notice higher on the % of Shaders Busy metric and % Time ALU working in all techniques, this happens because of the geometry complexity of the Hairball scene. The % Time Shading Fragments is much lower because most of the time the GPU is shading vertices instead of fragments.

The Fragment Instructions is not easily comparable since the techniques have different frame rates, for this reason, we calculated the number of instructions per frame (see figure 8).

During the execution of the tests, we collected data related to the GPU memory, we considered three metrics: average bytes per fragment, total bytes read by the GPU from memory per second and texture memory read in bytes. Tiled architecture aims to minimize the memory transfers between GPU and RAM, the memory stats collected are important to analyze how each antialiasing technique performs.

The results for both scenes are present on table II.

From the memory stats we can see that both SMAA variants have less memory transferred, at first we could see this information as good behavior from these techniques, however, we think that this is not the case. Tiled architectures are known for batching the work on the GPU, for example, when executing two draw calls and using the result from the first draw, the GPU tries to execute everything together to gain performance and avoid memory transfers. While MSAA, FXAA, TAA have a higher value of memory being used, this might indicate that the GPU is batching all the work, on the other hand, SMAA looks like cannot execute all at once, and needs to separate the work in more GPU passes. We believe this is the case because the implementation of FXAA uses a lot less texture memory than SMAA T2x.

The results obtained on the Hairball scene differ a lot from the previous scene because that this 3D model does not have textures. In this case, we can verify that MSAA performed best in all the metrics, the difference on the read total memory among the techniques is not so significant however as we discussed before these metrics are difficult to analyze due to the techniques having different frame rates. The Read Total per Frame is a more reliable metric because it takes into account the FPS obtained, SMAA T2x and TAA use a higher amount of memory because the previous frame is saved in memory. SMAA and TAA obtain much higher values on the Average Bytes / Fragment and on Texture Memory Read BW because these techniques implementations need lookup

TABLE II
MEAN VALUES OF GPU MEMORY STATS COLLECTED.

	No AA	MSAA 4x	FXAA	SMAA 1x	SMAA T2x	TAA
			Crytek Sponza			
Average Bytes / Fragment	14.4	9.4	14.1	7.6	7.2	11.4
Read Total (GB/s)	8.2	9.4	7.8	6.0	7.8	11.6
Read Total per Frame (MB)	137.8	159.7	131.8	138.4	251.4	210.2
Texture Memory Read BW (GB/s)	5.5	4.7	5.0	3.2	3.8	6.4
			Hairball			
Average Bytes / Fragment	0.0	0.0	0.0	0.6	2.1	3.5
Read Total (GB/s)	13.4	8.9	12.8	11.1	10.3	12.0
Read Total per Frame (MB)	383.7	466.8	386.9	412.5	516.9	481.8
Texture Memory Read BW (MB/s)	0.2	0.1	0.1	23.1	88.7	121.1

textures to perform antialiasing. The results obtained of fragment instructions per frame make more sense, the more intensive techniques have a higher number of instructions.

In terms of GPU temperature, the results are similar to previous results among the AA techniques, MSAA obtains lower temperatures followed by FXAA, TAA and SMAA. The results are displayed in figure 9. Comparing the temperature on both scenes, Hairball obtains higher temperatures, 4 to 8 degrees higher. In terms of temperature within the same scene, MSAA performed better with 47.3 ° C on Crytek Sponza and 53.2 ° C on Hairball. SMAA 1x got higher temperatures, 53.3 ° C on Crytek Sponza and 58.8 ° C on Hairball. In the tests, we noticed that when the camera rotated to more difficult areas of the scene, the frame-rate lowered and the temperature rose.

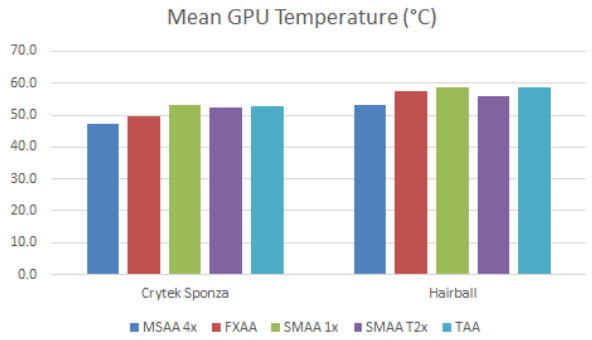

Fig. 9. Chart comparing the mean GPU temperature obtained during the execution of antialiasing techniques on both scenes.

B. Image quality

To study the quality of the image produced by the antialiasing techniques we used two methods: a quantitative method using SSIM and a more subjective method by visually analyzing how each technique performed in certain cases.

We applied the SSIM method between a supersampled 4x image (16 samples per pixel) and the resulting image from each technique. The techniques that obtained higher values on the structural similarity index have visual results that are approximated to the supersampled version, thus have better image quality.

Figure 10 contains the SSIM results on both scenes. We can observe that on Crytek Sponza, SMAA T2x obtained the best result with 0.94, followed by TAA, SMAA, MSAA and FXAA.

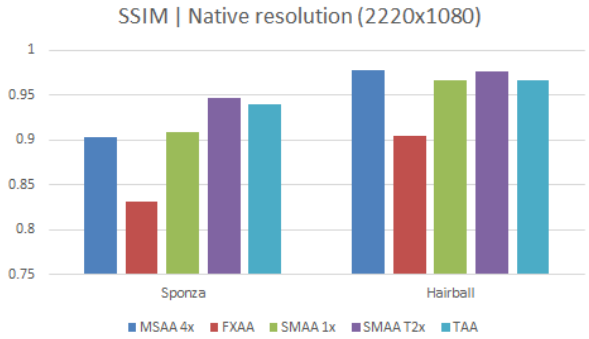

Fig. 10. Chart comparing the SSIM results on native resolution (2220x1080) on both scenes.

On Hairball, MSAA 4x obtains the best result with 0.977 of similarity to the supersampled reference. It performs better than on Sponza because MSAA cannot improve texture details, unlike temporal SMAA. SMAA T2x again performs well with the SSIM result of 0.976, followed by TAA, and SMAA 1x both with 0.966 and FXAA with 0.9.

Considering all test cases, we can say that SMAA T2x obtains the best SSIM results, MSAA comes in second place, FXAA performs worst in this metric.

Besides SSIM comparison, we extracted a zoomed region on the images produced by each technique, to compare how they performed in different cases, see figure 11. The following aspects were observed in the results of the antialiasing techniques: Shape preservation, Diagonals, Textures, Multiple edges and Fine geometry.

The shape preservation case is difficult to be handled by antialiasing techniques because of the round geometry and texture on the center. Between SSAA 4x and No AA, we can observe that SSAA has more intermediate detail on the edges, less abrupt changes and the texture is more detailed. Comparing each AA technique, SMAA T2x has the most

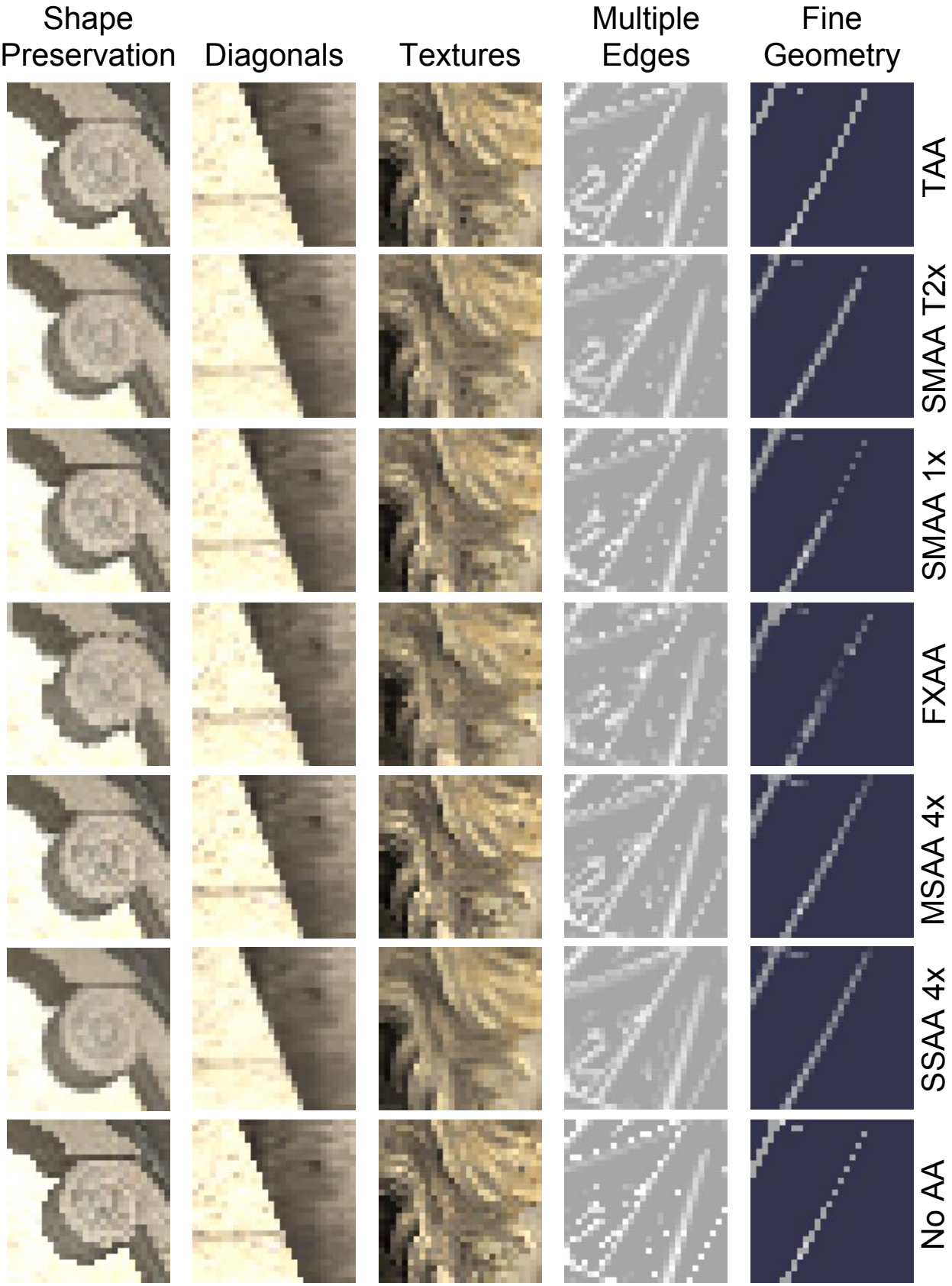

Fig. 11. Zoomed details of the antialiasing techniques.

similar result to supersampling. The other techniques also improve the result, FXAA on this case has the least appealing result, especially on the horizontal edge above the texture which loses its original shape.

Diagonals are prone to aliasing in 3D applications, when the results show abrupt changes between pixels they are known as *jaggies*. SMAA diagonal handling really shines in this particular case, obtaining edge gradients similar to the supersampled version. MSAA result is better than FXAA. Our TAA version does not improve the diagonal significantly, one thing that would help improve this case is a more distributed sampling sequence, we only use two different sample locations. Unreal Engine TAA version uses the Halton Sequence with 8 samples for jittering which is very well distributed.

We can observe that MSAA does not improve the texture quality at all because this technique works at the geometry level. FXAA reduced the overall contrast from the original image, making it look less pixelated. When comparing FXAA with SMAA and TAA it can be said that FXAA results are more blurred, some texture details are lost. The temporal techniques improve the texture slightly without much blur, again in this case a better sampling distribution spread across frames could improve the quality.

In multiple edges category, we can verify that there are high frequencies on the no AA version, looking more pixelated than usual. All the AA techniques smooth the highlights and improve the image in general. SMAA result is the most similar to the supersampled version.

In this last comparison, its a problem of geometry aliasing, the geometry is so thin that rasterizer did not cover some pixels. MSAA obtained similar results to the supersampled reference, oversampling works well in this case. FXAA reconstruction of the geometry gets poor quality results. The temporal techniques manage to reconstruct the shape.

C. Overall analysis

In terms of performance, the techniques that obtained the best results were MSAA 4x and FXAA, they performed well in every metric. MSAA cost increases linearly when the scene complexity increases, which was proved in the Hairball scene.

Regarding the image quality, the temporal variant of SMAA obtained the best results. It can be said that temporal techniques work very well to combat aliasing, especially using jittering improves the texture quality. MSAA also performed well, in the Hairball scene the SSIM results between MSAA and SMAA were very close.

We believe that the most balanced option between performance and image quality for antialiasing is MSAA.

V. CONCLUSION

The objectives of this work are testing and comparing antialiasing techniques in a mobile environment. We chose four state-of-the-art antialiasing techniques to be compared in terms of performance and image quality. The techniques subject to tests were MSAA 4x, FXAA, SMAA and TAA.

The test results show that MSAA and FXAA have the best performance in the evaluated metrics, with a higher number of FPS, use less GPU processing power and memory. In terms of image quality, SMAA obtains the best results in SSIM, meaning that it produces results very similar to the supersampled reference. In the visual analysis, we found that SMAA results are better than the other techniques. MSAA gets the second best results in terms of image quality. Considering both aspects performance and quality, MSAA is the most balanced technique.

The post-processing antialiasing solutions, FXAA and SMAA, have good results, SMAA is more recent and obtains results very similar to supersampling.

In our work, the temporal antialiasing implementation was not perfect, it is a complicated technique that needed some improvements, especially to remove ghosting. Even so, it obtained good visual results and performance was decent.

We believe that there are some optimizations that could have been done in the test application. When using Framebuffer objects they should be invalidated in case they are not needed anymore in that render cycle. Compression of textures could also have been an optimization, using compression improves memory bandwidth which improves performance.

In the techniques FXAA and SMAA, the default presets were used, tweaking the parameters of these techniques might have improved their performance. One of the parameters is edge search distances which iterate a sample point along a texture, in the tiled architecture this type of texture access might occur outside of the current tile, causing a halt until the neighbor tile is rendered, which is a huge performance drop.

SMAA had the best image quality, it would be interesting improving the performance of this technique in tiled rendering.

REFERENCES

[1] Peter Young. Coverage sampled antialiasing. Technical report, NVIDIA, 2006.

[2] Tomas Akenine-Möller, Eric Haines, and Naty Hoffman. *Real-Time Rendering 3rd Edition*. A. K. Peters, Ltd., Natick, MA, USA, 2008.

[3] Jorge Jimenez, Jose I. Echevarria, Tiago Sousa, and Diego Gutierrez. SMAA: Enhanced subpixel morphological antialiasing. *Computer Graphics Forum*, 2012.

[4] Timothy Lottes. FXAA. Technical report, NVIDIA. 2011.

[5] Alexander Reshetov. Morphological antialiasing. *Proceedings of the 1st ACM conference on High Performance Graphics*, 2009.

[6] Jorge Jimenez, Diego Gutierrez, Jason Yang, Alexander Reshetov, Pete Demoreuille, Tobias Berghoff, Cedric Perthuis, Henry Yu, Morgan McGuire, Timothy Lottes, et al. Filtering approaches for real-time antialiasing. *ACM SIGGRAPH Courses*, 2011.

[7] Brian Karis. High-quality temporal supersampling. *Advances in Real-Time Rendering in Games, SIGGRAPH Courses*, 2014.

[8] Steven Molnar, Michael Cox, David Ellsworth, and Henry Fuchs. A sorting classification of parallel rendering. *IEEE computer graphics and applications*, 1994.