Real-Time Rendering Techniques



Full Screen Anti-Aliasing

[30077548@students.southwales.ac.uk](mailto:30077548@students.southwales.ac.uk?subject=Rule%20Based%20Systems%20Report)

Sebastian Ha

30077548

Table of Contents

[Table of Acronyms 4](#_Toc220087851)

[Introduction 5](#_Toc220087852)

[Scene Setup and Rendering Pipeline 6](#_Toc220087853)

[Scene Description 6](#_Toc220087854)

[Rendering pipeline 6](#_Toc220087855)

[Shaders 6](#_Toc220087856)

[Texture usage 6](#_Toc220087857)

[Performance Tracking 6](#_Toc220087858)

[Base scene without Anti-Aliasing 7](#_Toc220087859)

[Visual notes 7](#_Toc220087860)

[Screenshots 7](#_Toc220087861)

[Performance 8](#_Toc220087862)

[Multi-Sample Anti-Aliasing (MSAA) 9](#_Toc220087863)

[How does MSAA work? 9](#_Toc220087864)

[How MSAA was enabled in the scene 9](#_Toc220087865)

[Sample counts 9](#_Toc220087866)

[Screenshots 10](#_Toc220087867)

[Performance and visual notes 11](#_Toc220087868)

[Super-Sample Anti-Aliasing (SSAA) 12](#_Toc220087869)

[Implementation 12](#_Toc220087870)

[Sampling resolutions 12](#_Toc220087871)

[Screenshots 13](#_Toc220087872)

[Performance cost 14](#_Toc220087873)

[Comparison and evaluation 15](#_Toc220087874)

[Conclusion 15](#_Toc220087875)

[Appendix 16](#_Toc220087876)

[Asset licenses 16](#_Toc220087877)

[Rubik’s Cube 16](#_Toc220087878)

[Moon 16](#_Toc220087879)

[Grass 16](#_Toc220087880)

[Code tutorial Sources 17](#_Toc220087881)

[Code 17](#_Toc220087882)

[Scene Renderer Class 17](#_Toc220087883)

[Scene FBO Class 17](#_Toc220087884)

[Basic Shader 17](#_Toc220087885)

[Source File 17](#_Toc220087886)

# Table of Acronyms

|  |  |
| --- | --- |
| FSAA | Full-Screen Anti-Aliasing |
| MSAA | Multi-Sample Anti-Aliasing |
| SSAA | Super-Sample Anti-Aliasing |
| AA | Anti-Aliasing |
| VBOs | Vertex Buffer Objects |
| VAOs | Vertex Array Objects |
| FPS | Frames Per Second |
| FBO | Frame Buffer Object |

# Introduction

Anti-Aliasing is a technique used in computer graphics that reduces aliasing effects by smoothing jagged edges or stair-like steps on curved lines and diagonals on computer screens. This matters in real-time rendering as these effects become more noticeable especially during movement within the rendered scene, which can cause the visual quality of the scene to drop. Anti-aliasing helps rendered objects appear more realistic due to smoother edges. Two full-screen anti-aliasing techniques (FSAA) will be compared against a base 3D real-time rendered scene. The techniques compared are Multi-Sample Anti-Aliasing (MSAA) and Super-Sample Anti-Aliasing (SSAA).

MSAA is a type of FSAA that takes multiple depth samples within a single pixel without needing to recalculate lighting or textures. On the other hand, SSAA renders the whole scene at a higher resolution and combines multiple samples per pixel. This increases the performance cost and the visual quality. The goal of this report is to compare the quality and the performance between both FSAA techniques against a base 3D scene that has no anti-aliasing (AA) techniques applied. The comparison will determine which AA technique is better suited for modern real-time rendering applications.

# Scene Setup and Rendering Pipeline

## Scene Description

The scene is constructed of a simple 3D setup consisting of a textured ground grass plane, a sphere representing a moon and a Rubik’s cube. The background of the scene is a flat blue colour instead of a skylight or background, making sure nothing affects the AA technique comparisons. The scene was kept simple intentionally so that comparisons could be shown clearly using fewer contrasting models. Overcomplication could reduce how clearly the results are shown from the different aliasing effects.

## Rendering pipeline

The rendering pipeline consists of the scene implementing OpenGL version 3.3 Core. GLFW is used for configuration, input callback functions and window creation while GLAD is used to load all OpenGL function pointers. Geometry is stored within Vertex Buffer Objects (VBOs), objects are then configured using Vertex Array Objects (VAOs). Depth testing is enabled ensuring visibility and occlusion are correct within the scene, otherwise the perception of depth would not be portrayed correctly if parts of an object were not rendered in order. GLFW disables vertical synchronisation using swap intervals to show the frame rate of the scene. The camera class provides an object that displays the view and projection matrices, allowing the user to see the scene from a viewport that can also be controlled.

## Shaders

The shaders used within this scene were a basic shader and a Phong lighting shader, each implemented as vertex and fragment shaders. The basic shader is used on the Rubik’s cube and grass plane models to render the textured objects where detailed surface lighting is not necessary. In contrast, the moon model is rendered using a Phong shader for per-fragment lighting. This allows smooth shading across the surface of the sphere which produces more realistic results. These shaders were chosen to show a clear contrast between simple and complex shading methods while ensuring the scene remained appropriate for comparing AA techniques.

## Texture usage

The models within the scene were texture mapped using PNG image files loaded at run time and sampled in the fragment shader. Material files were not used, as implementing image textures allowed the scene setup to be more controlled and simpler so focus could be on the comparison of different anti-aliasing techniques.

## Performance Tracking

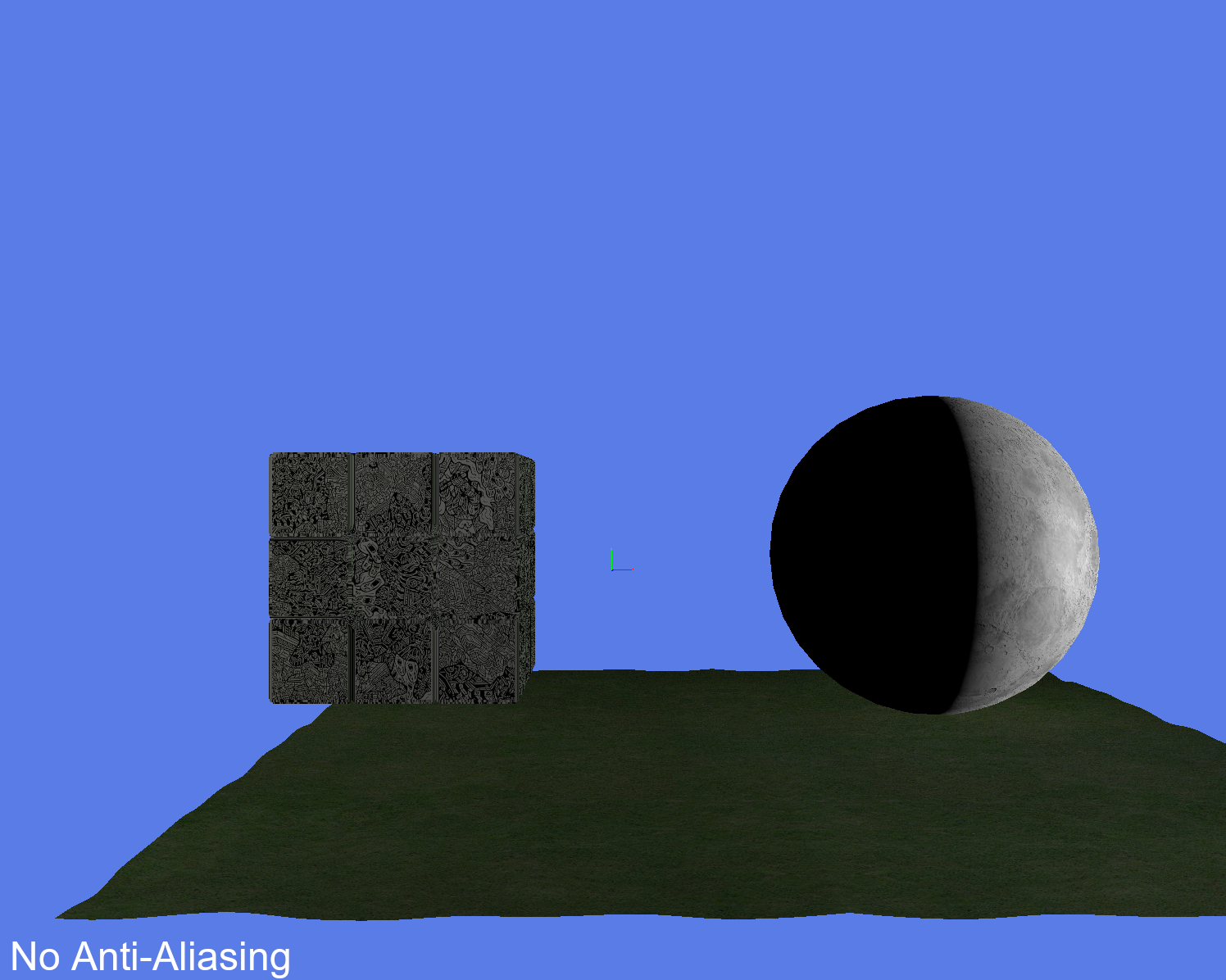
To track performance within the scene when different AA effects are applied to the scene, a tracker was implemented that tracked the frames per second (FPS) and how many milliseconds each frame takes. These values are updated once a second and displayed on the scene’s window title, this allows performance comparisons in real time.

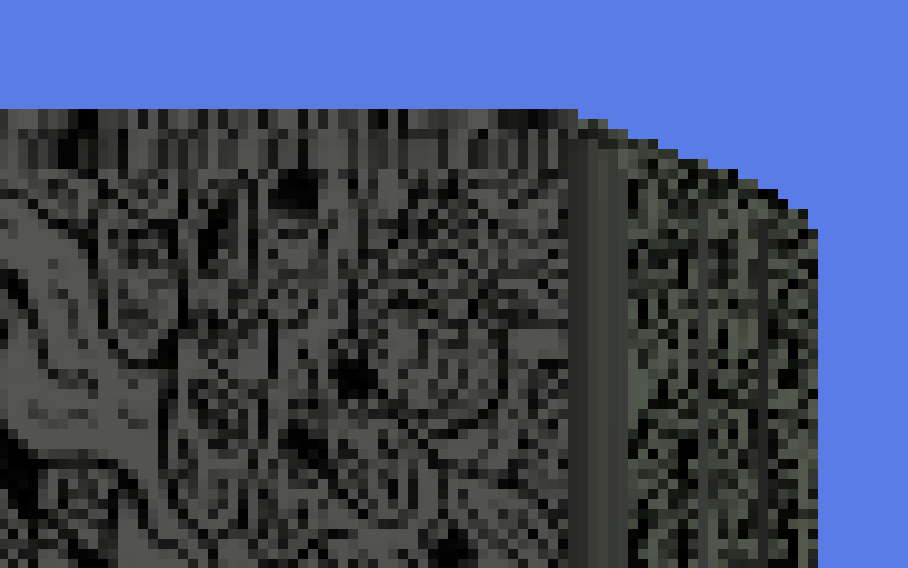
# Base scene without Anti-Aliasing

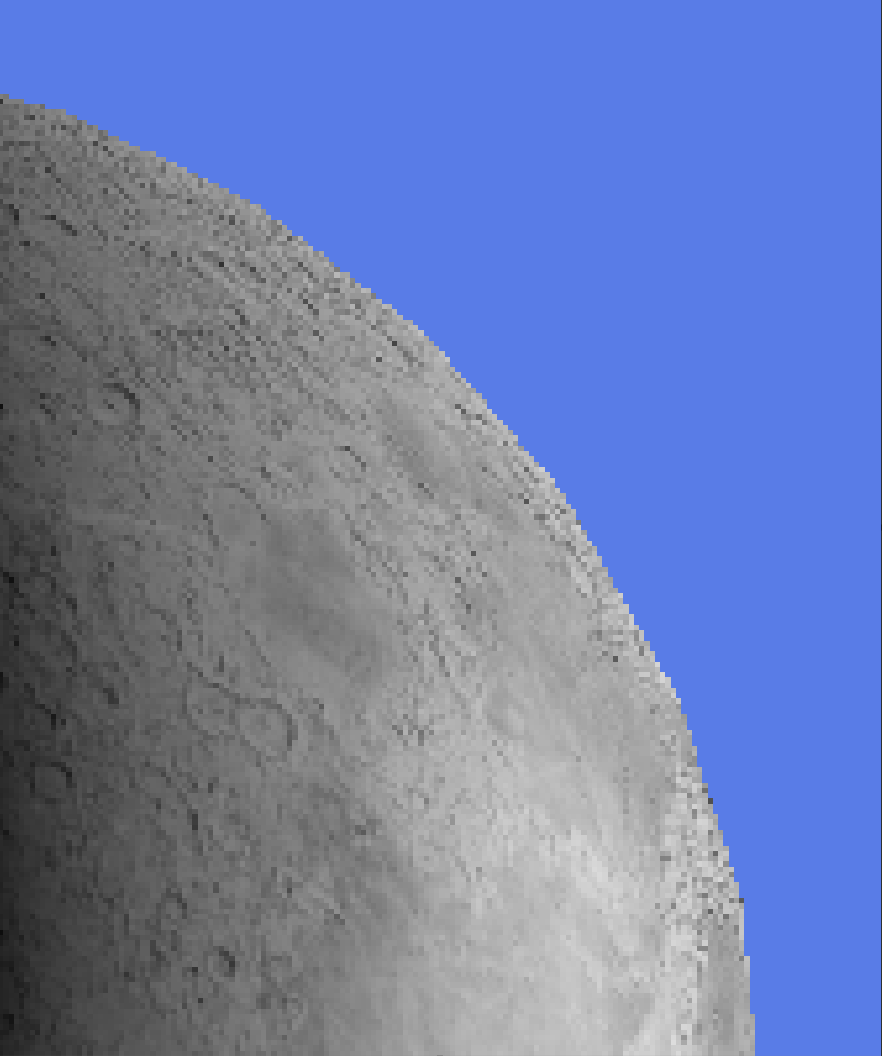
## Visual notes

Upon rendering the scene, jagged edges along the models are prominent, with the grass model being the most affected due to the uneven surface. Models that are curved, such as the moon show noticeable stair-stepping effects along the outlines. When the moon model is rotating around its own axis, shimmering effects are observed as the jagged edges change relative to the camera view making the edge appear unstable.

## Screenshots









## Performance

The scene’s FPS performed within a range of five thousand to seven thousand five hundred frames. This shows the scene sustains a high-performance level even with the three models rendered. When moving around the scene, the frame rate drops to around five thousand FPS, but when the camera is still the frames can range between six thousand upwards to seven and a half thousand.

# Multi-Sample Anti-Aliasing (MSAA)

## How does MSAA work?

MSAA works by having multiple depth samples taken at different sub-pixel locations within a pixel. These samples share the same texture and lighting calculations while allowing different depth values per sample. This reduces the computational overhead significantly compared to SSAA. MSAA is widely used in modern game engines due to its ability to balance between quality and performance.

## How MSAA was enabled in the scene

MSAA was enabled during the initialisation and configuration of GLFW. The number of samples was specified before the window was created.



Once the window was created, multi-sampling was handled using OpenGL. This allowed the scene to enable and disable multi-sampling when needed. No changes were required to the shaders used within the scene as MSAA is handled internally by the hardware.

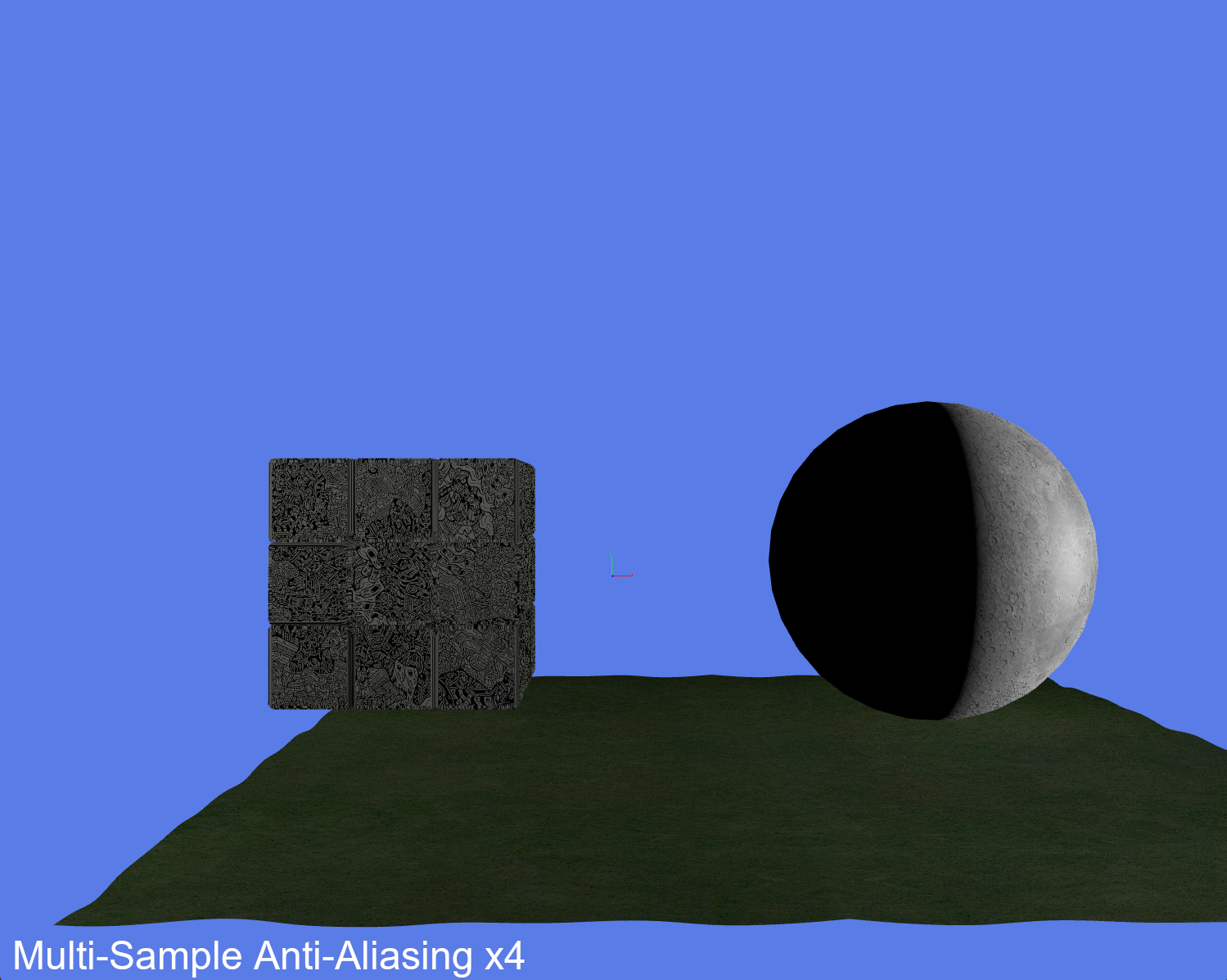


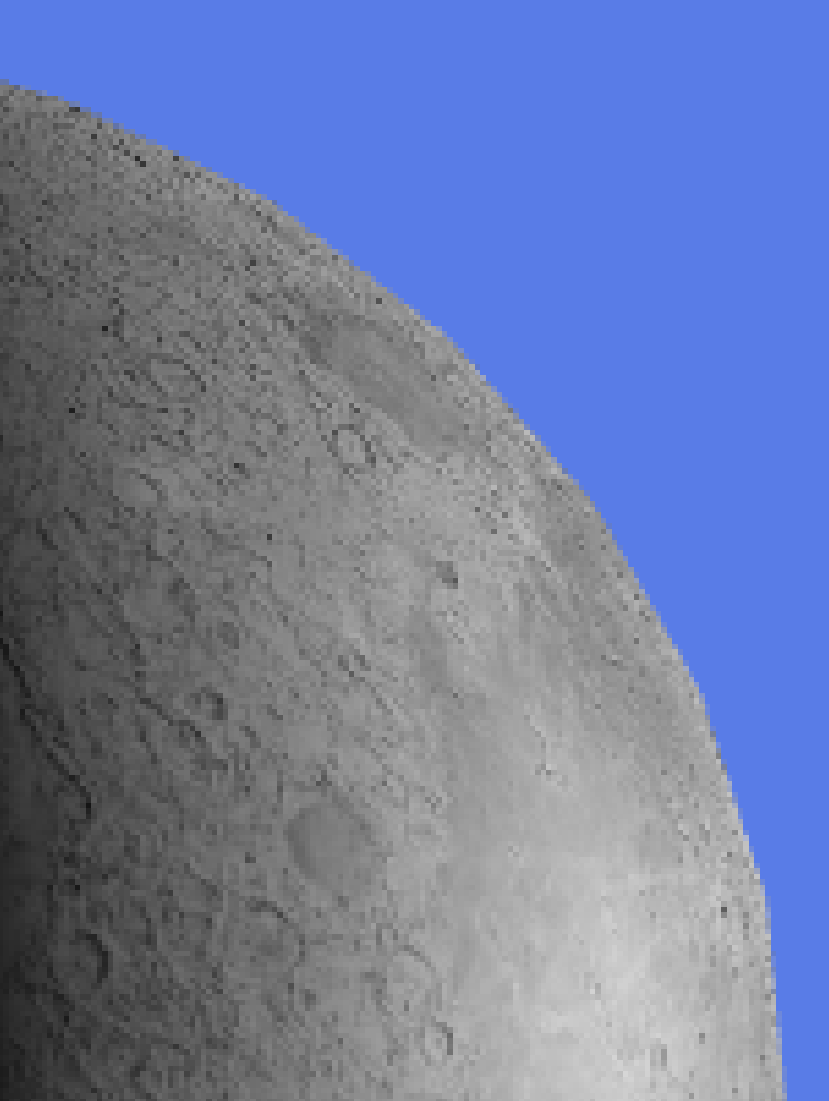
## Sample counts

During development, multiple sample counts were tested, including 2x, 4x, and 8x sampling. As the sample count increased, edge smoothing became more prominent, especially on curved model edges. However, higher sample counts increased the performance cost, with minimal differences visually after 4x multi-sampling.

From the observations made, 4x MSAA was chosen for the final screenshots. This was so performance remained high while seeing a clear improvement in visual quality compared to the base scene that had no AA.

## Screenshots







## Performance and visual notes

Enabling MSAA resulted in an average of four thousand one hundred to four thousand three hundred FPS. Although this shows a reduction of around one thousand FPS, there was still a considerably high-performance rate within the scene while displaying noticeably smoother object edges. MSAA helped to reduce aliasing artefacts along non-straight edges and improving visual stability during camera movement within the scene.

# Super-Sample Anti-Aliasing (SSAA)

## Implementation

SSAA requires off-screen rendering where a framebuffer object (FBO) is used to render the scene before it is displayed to the viewport. Rendering to the FBO allows the scene to be displayed at a higher resolution. The render target resolution attached to the FBO is higher than the window resolution. The higher resolution produces multiple samples per output pixel, resulting in improvements to the visual quality of the scene. However, this increase in quality comes at a significant performance cost, as SSAA has a substantial amount of computational overhead.

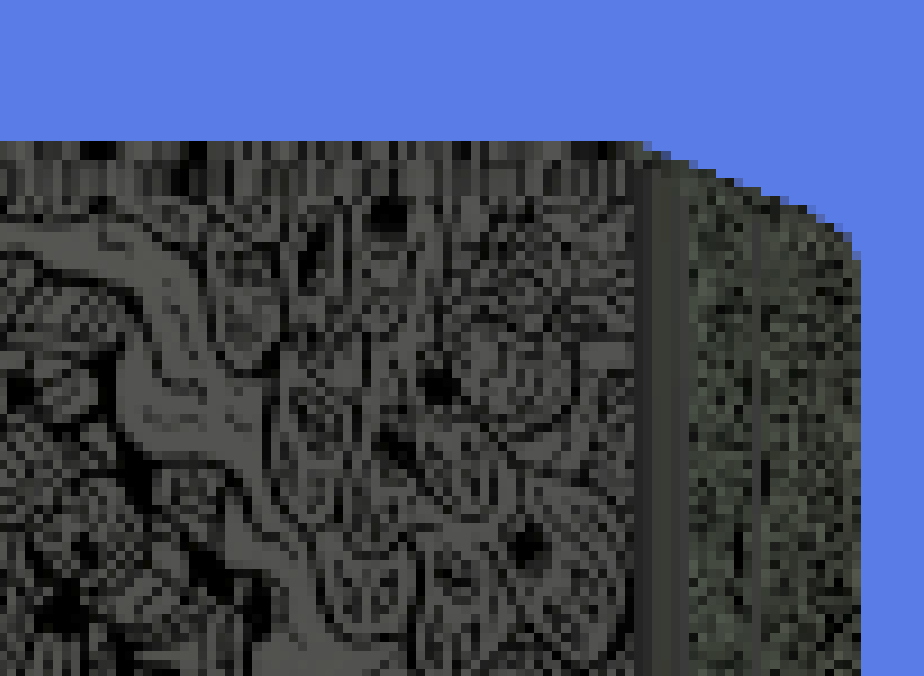
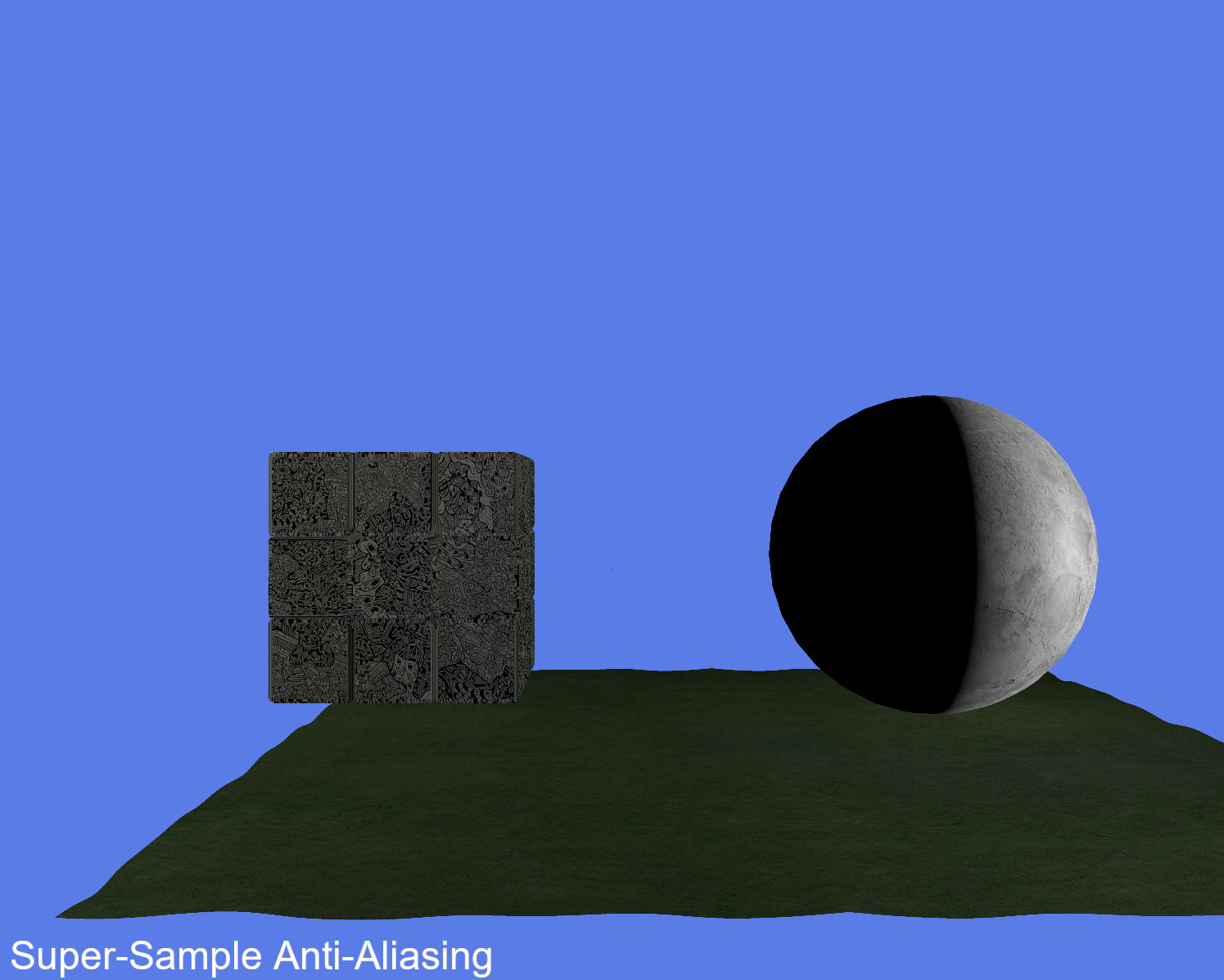
Rendering the scene with SSAA consists of two passes. The scene is rendered to the off-screen FBO at a higher resolution in the first pass, which is then used by the second to render the scene to the viewport. Without the initial pass to the FBO, the scene is unable to be rendered using SSAA properly.

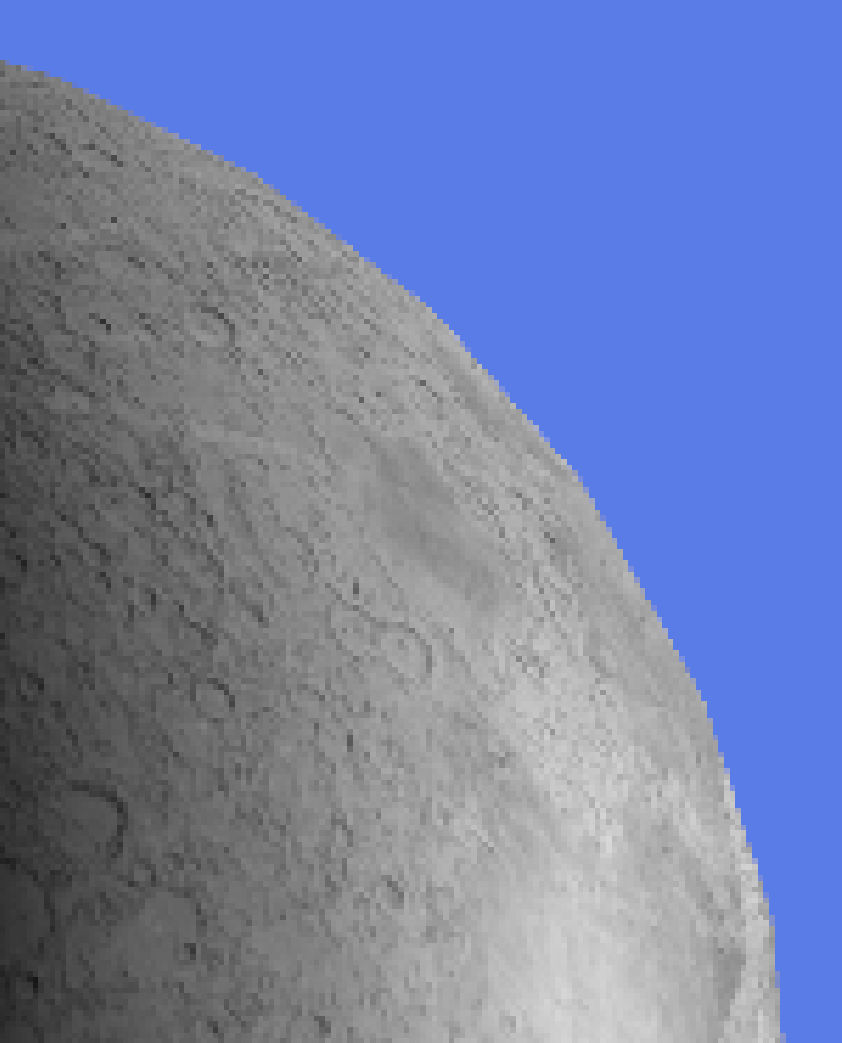
The high-resolution texture attached to the FBO was rendered using a textured quad that was aligned to the screen and a basic texture shader. No modifications were made to the shader, which down sampled the higher resolution image during the second pass to the resolution of the viewport window. By ensuring the quad aligned to the screen, the scene rendered correctly and camera movement was still possible as if looking at the scene rendered without AA or MSAA.

## Sampling resolutions

When testing, 2x, 4x and 8x resolutions were explored during development. Although, 8x resolution was only briefly evaluated as the performance cost was significantly higher with no additional visual improvements at all. Due to this, SSAA 4x was chosen for comparisons to other AA techniques.

## Screenshots





## Performance cost

When enabled, the scene performs within a range of five hundred to one thousand two hundred FPS. This significant reduction in performance was due to the scene being rendered at a higher resolution, which increased the number of fragments processed each frame. A correlation was observed where the higher the sample resolution became, the lower the performance of the scene was. With the scene being simple on purpose, this highlights how SSAA would be even more expensive computationally when applied to more complex and larger scenes regularly seen within video games.

# Comparison and evaluation

* Direct comparison table or discussion:
* No AA vs MSAA vs SSAA
* Visual quality vs performance trade-offs
* Why modern engines prefer MSAA (or TAA, FXAA — optional mention)

# Conclusion

* Summary of findings

MSAA more efficient than SSAA

* What technique would be used in a real time engine

Multi Sample Anti-Aliasing is popular with modern game engines and both OpenGL and DirectX provide built in support for this technique.

* Brief reflection of performance vs quality

# Appendix

## Asset licenses

### Rubik’s Cube



<https://www.turbosquid.com/3d-models/rubik-cube-with-dbrand-robot-skin-1587120>

License: <https://blog.turbosquid.com/turbosquid-3d-model-license/>

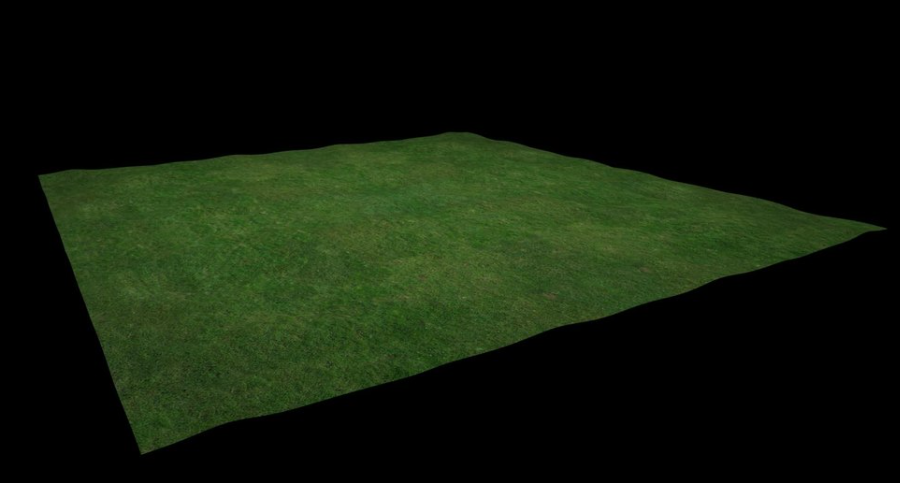
### Moon



<https://www.turbosquid.com/3d-models/moon-3d-model/535681>

License: <https://blog.turbosquid.com/turbosquid-3d-model-license/>

### Grass



<https://www.turbosquid.com/3d-models/3d-grassland-duco-3d-1822648>

License: <https://blog.turbosquid.com/turbosquid-3d-model-license/>

## Code tutorial Sources

FPS counter video: <https://www.youtube.com/watch?v=BA6aR_5C_BM>

Enabling MSAA tutorial video: <https://www.youtube.com/watch?v=oHVh8htoGKw>

## Code

### Scene Renderer Class

### Scene FBO Class

### Basic Shader

### Source File