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](#_Toc220068149)

[Introduction 5](#_Toc220068150)

[Scene Setup and Rendering Pipeline 6](#_Toc220068151)

[Scene Description 6](#_Toc220068152)

[Rendering pipeline 6](#_Toc220068153)

[Shaders 6](#_Toc220068154)

[Texture usage 6](#_Toc220068155)

[Performance Tracking 6](#_Toc220068156)

[Base scene without Anti-Aliasing 7](#_Toc220068157)

[Visual notes 7](#_Toc220068158)

[Screenshots 7](#_Toc220068159)

[Performance 8](#_Toc220068160)

[Multi-Sample Anti-Aliasing (MSAA) 9](#_Toc220068161)

[How does MSAA work? 9](#_Toc220068162)

[How MSAA was enabled in the scene 9](#_Toc220068163)

[Scene sample counts 9](#_Toc220068164)

[Screenshots 10](#_Toc220068165)

[Performance 11](#_Toc220068166)

[Super-Sample Anti-Aliasing (SSAA) 12](#_Toc220068167)

[Comparison and evaluation 14](#_Toc220068168)

[Conclusion 14](#_Toc220068169)

[Appendix 15](#_Toc220068170)

[Asset licenses 15](#_Toc220068171)

[Rubik’s Cube 15](#_Toc220068172)

[Moon 15](#_Toc220068173)

[Grass 15](#_Toc220068174)

[Code tutorial Sources 16](#_Toc220068175)

[Code 16](#_Toc220068176)

# 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 |

# 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

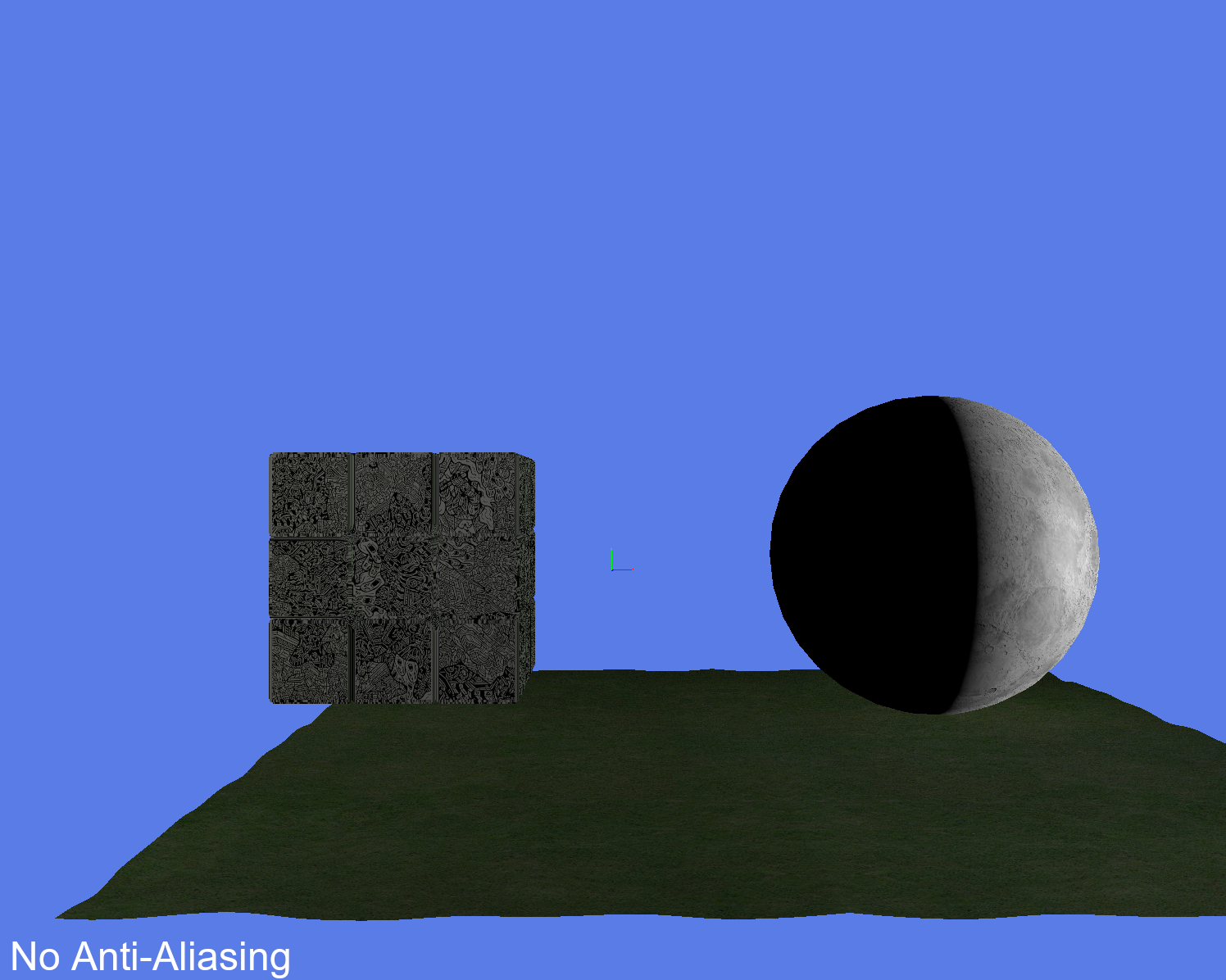
To track performance within the scene when different AA effects are applied to the scene, a tracker was implemented that tracked the frames per second (FPS) and how many milliseconds each frame takes. These values are updated once a second and displayed on the scene’s window title, this allows performance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

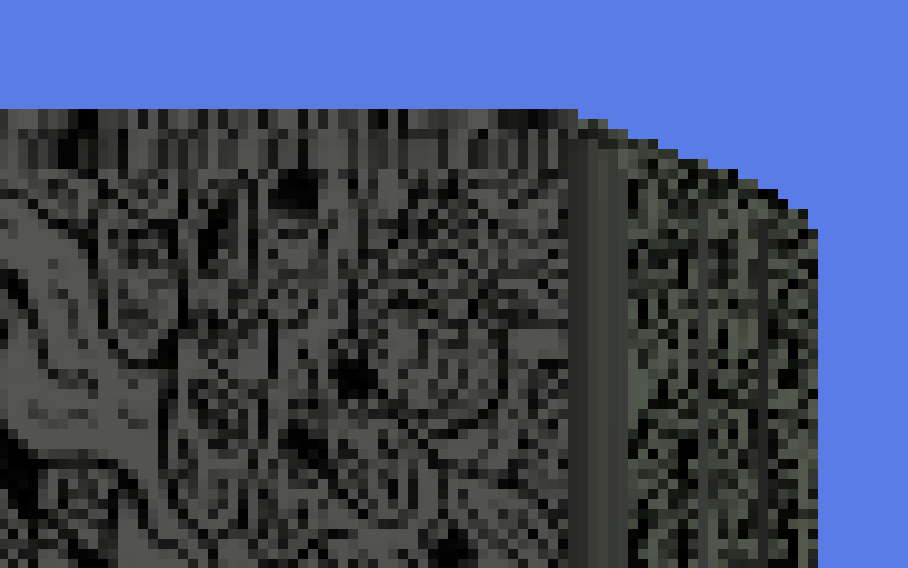
# Base scene without Anti-Aliasing

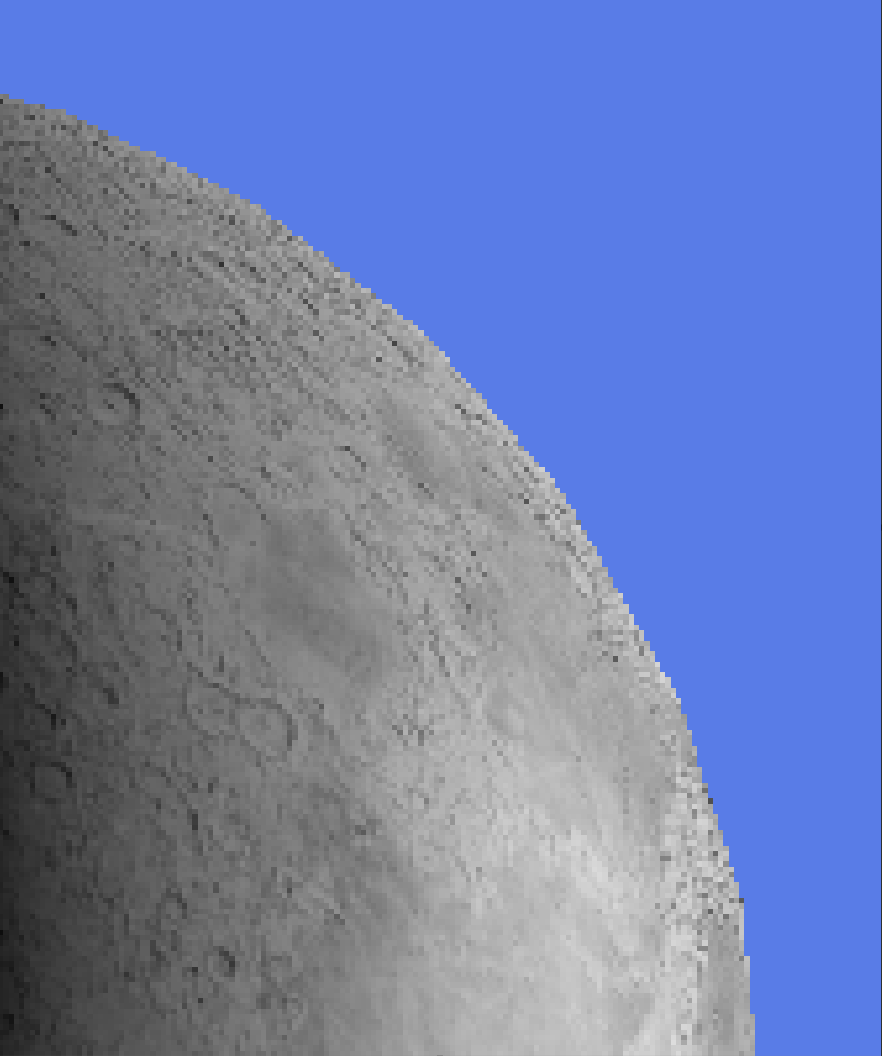
## Visual notes

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.

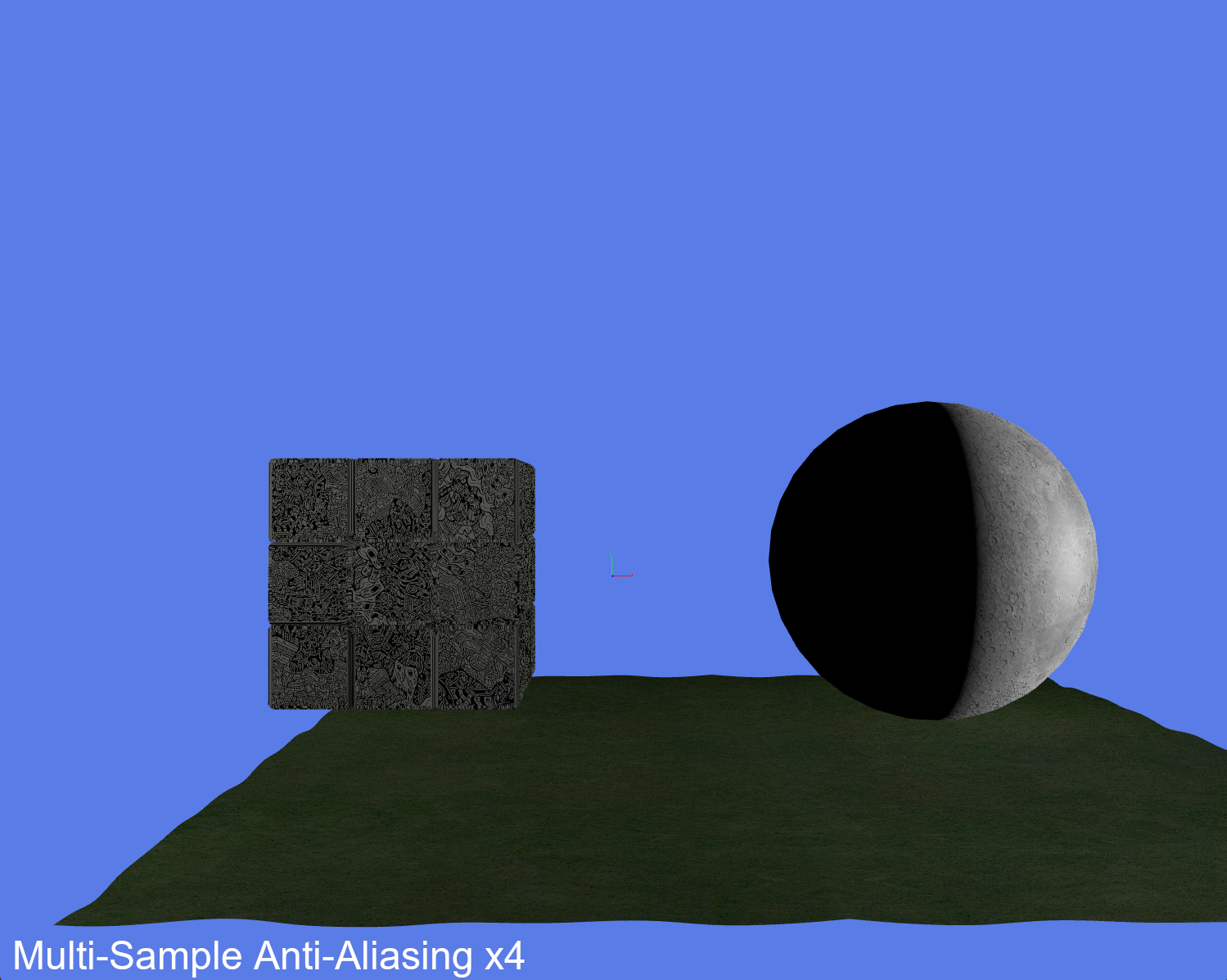


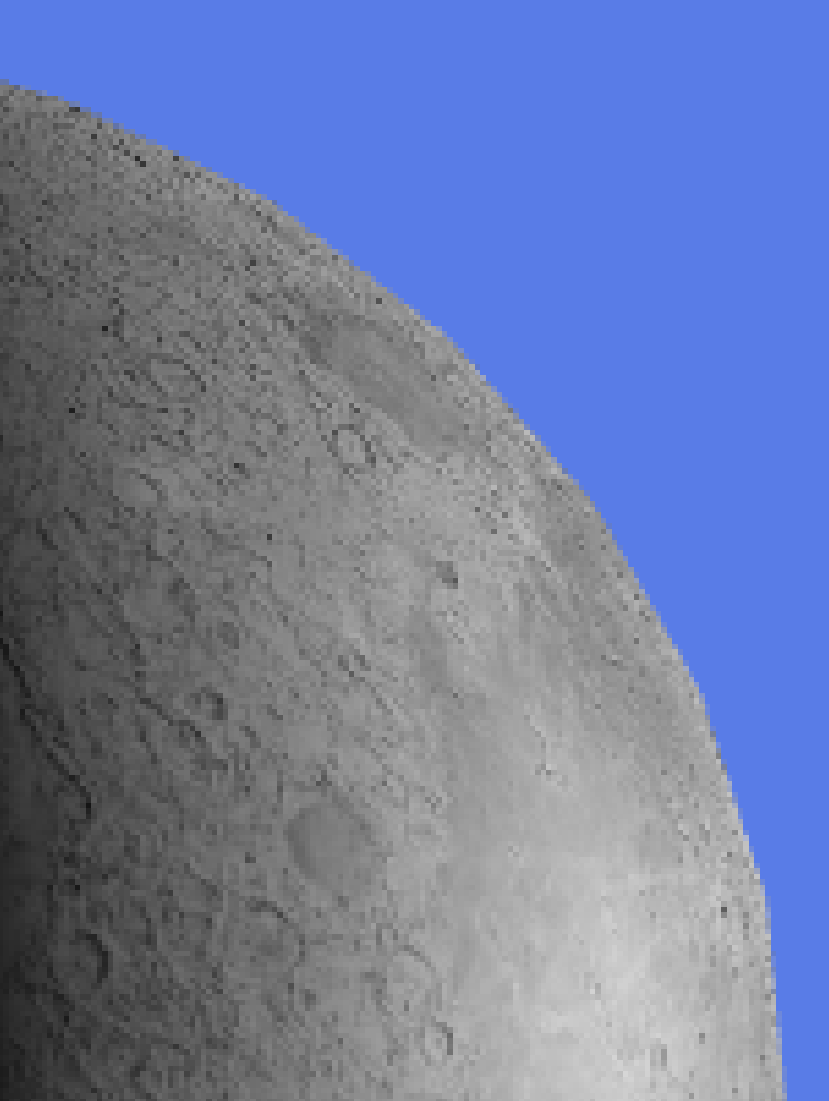
## Scene sample counts

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

* Edge smoothing
* Where MSAA helps and where it doesn’t
* Cost compared to base scene

# Super-Sample Anti-Aliasing (SSAA)

* Off-screen framebuffer (FBO)
* High resolution render target
* Two pass rendering approach
* Screen-aligned quad
* Resolve shader and averaging

**5.2 Sampling Patterns and Resolutions**

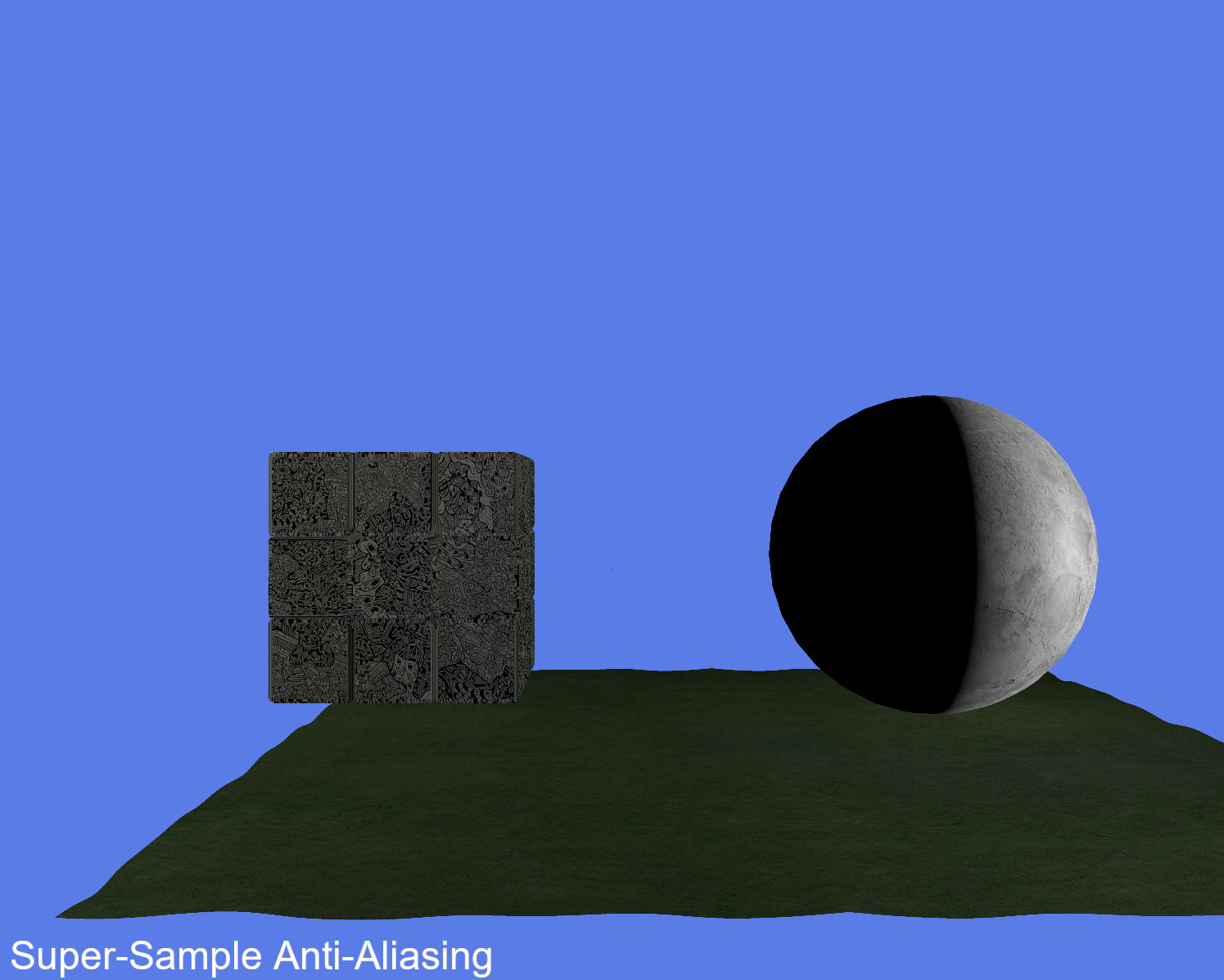
Include:

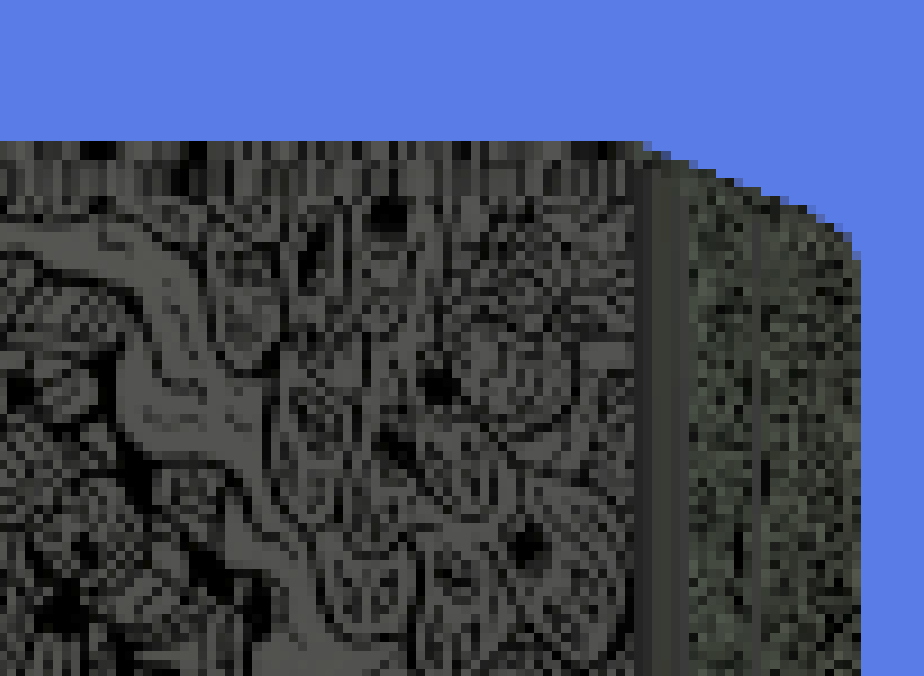
* 2× vs 4× resolution
* Sampling patterns (grid, box, etc.)
* Number of samples per pixel

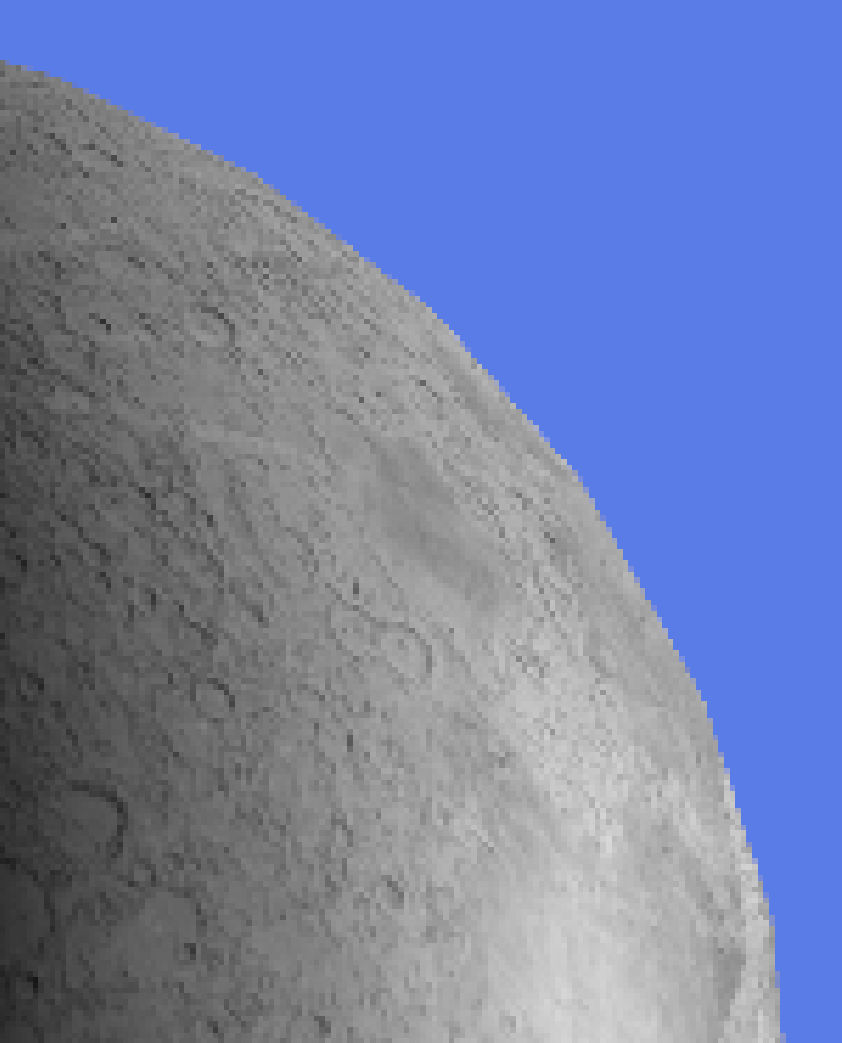
**5.3 Results**

Include:

* Visual improvements
* Screenshots
* Performance cost
* Comparison to MSAA









# 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/>

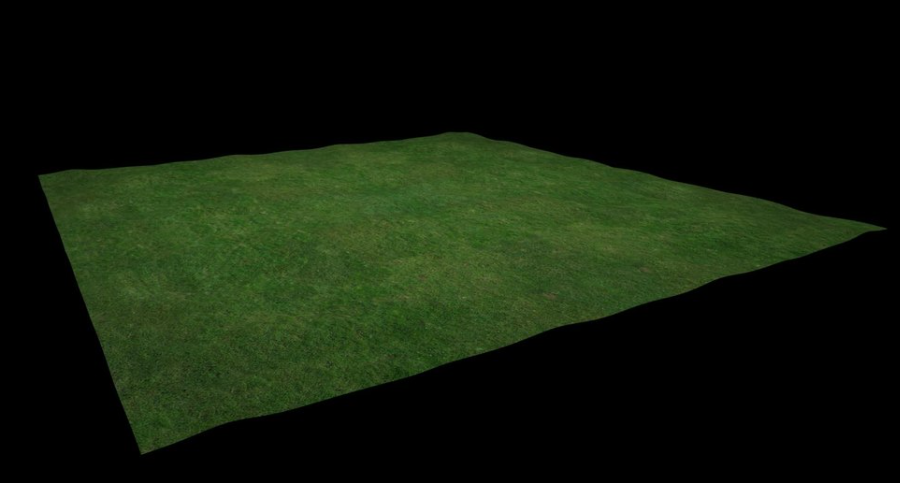
### Moon



<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/>

* Edited shader code
* FBO setup code
* SSAA resolve shader

## 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