School of Computing
FACULTY OF ENGINEERING

UNIVERSITY OF LEEDS

Signed Distance Fields as a Means of Improving the Speed of Rendering Three-Dimensional Fractals

Eleanor Joan Mills

Submitted in accordance with the requirements for the degree of MSc High Performance Graphics and Games Engineering

2021-2022

The candidate confirms that the following have been submitted.

Items	Format	Recipient(s) and Date
Project Report	Report	SSO (19/08/22)
Project Code	Zip File	SSO (19/08/22)

Type of project: Exploratory Software

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of Student) Nell Mills

Summary

<Concise statement of the problem you intended to solve and main achievements (no more than one A4 page)>

Acknowledgements

$<$The page should contain any acknowledgements to those who have assisted with your work. Where you have worked as part of a team, you should, where appropriate, reference to any contribution made by other to the project.$>$
Note that it is not acceptable to solicit assistance on 'proof reading' which is defined as the "the systematic checking and identification of errors in spelling, punctuation, grammar and sentence construction, formatting and layout in the test";
see http://www.leeds.ac.uk/gat/documents/policy/Proof-reading-policy.pdf.

Contents

Chapter 1

Introduction

<Brief introduction to project>
<Brief introduction to report and report structure>

1.1 Project Aim

1.2 Deliverables

1.3 Risks and Mitigation

1.4 Methodology and Version Control

Maybe put this in introduction instead of implementation.
Version control - GitHub
Methodology - Waterfall, since all Vulkan setup needs doing first
Show Gantt chart maybe... probably not.

Chapter 2

Background

This project aims to improve the efficiency of real-time rendering of 3D fractals. This chapter will focus on background theory for fractals, signed distance functions and sphere tracing. Afterwards, two possible methods of improving efficiency will be discussed, namely signed distance fields and temporal caching.

2.1 2D Fractals - The Mandelbrot Set

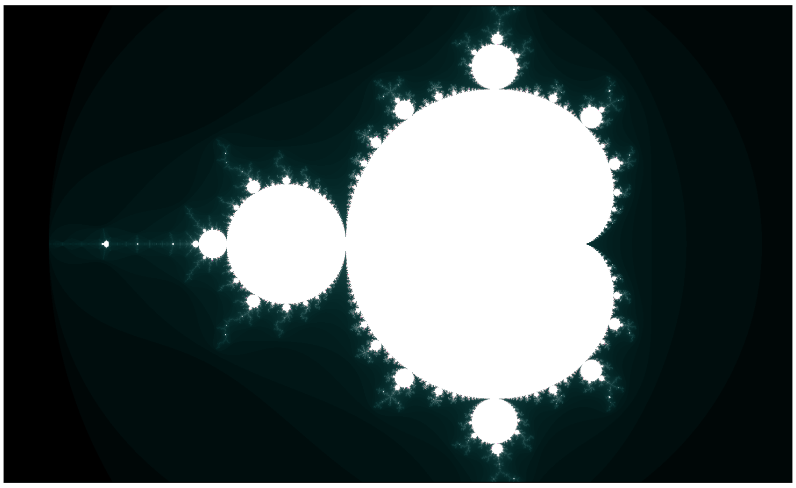


Figure 2.1: The Mandelbrot set. The white points in the centre are inside the set.

The Mandelbrot set is the set of two-dimensional points that satisfy a certain constraint on the following complex quadratic equation:

$$Z = Z^2 + C \tag{2.1}$$

where Z and C are complex numbers. The constraint on the points is that their orbit must be bounded. The value of Z is initialized to 0 and equation 2.1 is iterated over, each new value of Z being placed back in to the equation in the next iteration. If the length of the point Z does not exceed a threshold, then the point (represented by C) is in the Mandelbrot set [1].

Figure 2.1 shows a generated Mandelbrot set. The real part of the point C is represented by the x-axis, and the imaginary part by the y-axis. Equation 2.1 is iterated over a maximum of five hundred times, and the threshold value is two. The pixels are coloured according to how many iterations are achieved before the length of Z exceeds the threshold.

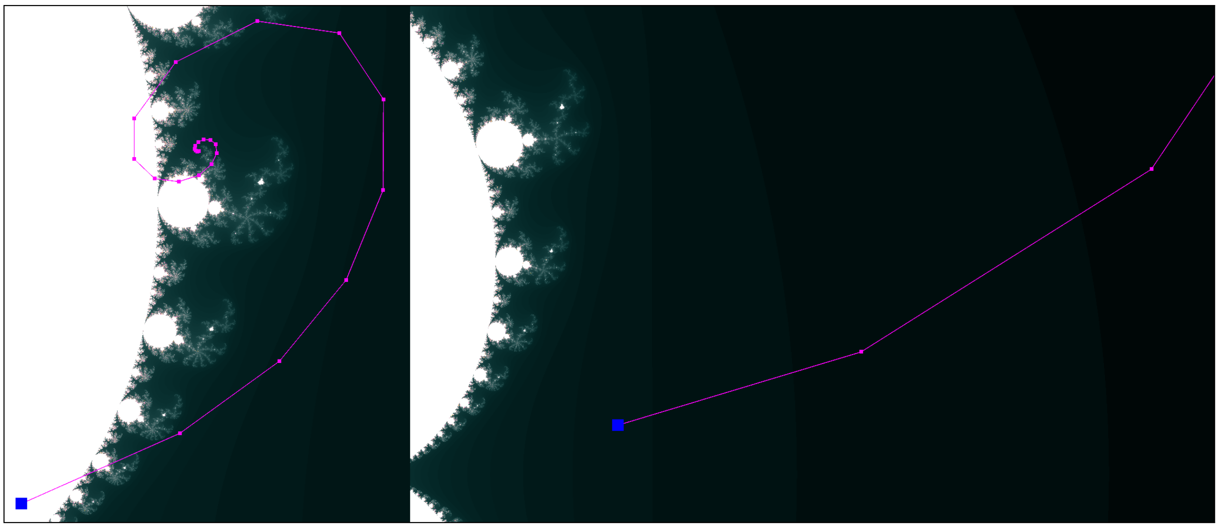


Figure 2.2: Visualization of the first twenty five iterations of equation 2.1 on the initial points [0.3, 0.05] (left) and [0.5, 0.04] (right). The initial points are shown in blue.

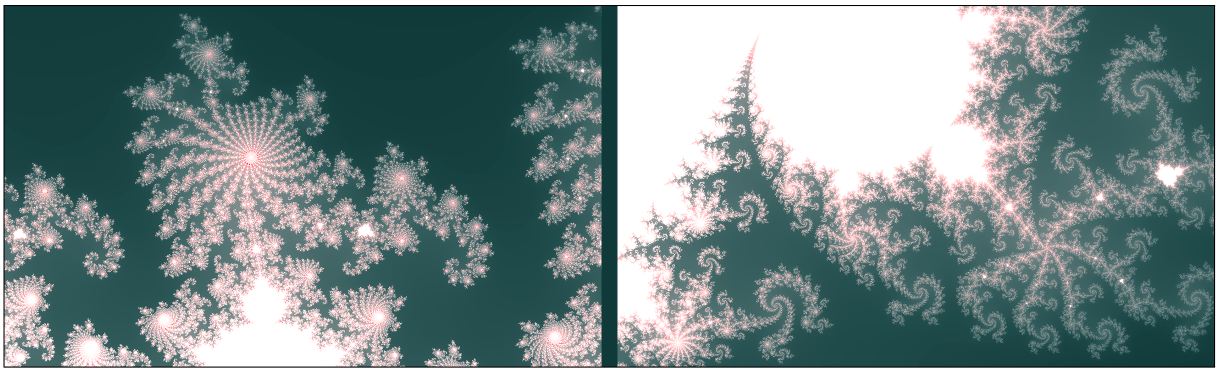


Figure 2.3: Two different views of the Mandelbrot set, zoomed in.

Figure 2.2 illustrates the first twenty five iterations on two different points. For the first point, the iterations converge in a spiral shape and the length of Z never exceeds the threshold of two, therefore the point is in the Mandelbrot set and is coloured white. For the second point, the iterations diverge and exceed the threshold of two within a few iterations, so this point is not in the Mandelbrot set and is coloured dark.

Figure 2.3 shows two zoomed-in views at the edge of the original shape. New patterns can be seen, as well as repeated ones, and even new instances of the original shape. This is because the Mandelbrot set has infinite detail, so if one decreases the range of the axes, new patterns will emerge [2].

This project makes use of three-dimensional fractal rendering, so the next section will look at the challenge of bringing this infinite level of detail to three dimensions.

2.2 3D Fractals - The Mandelbulb

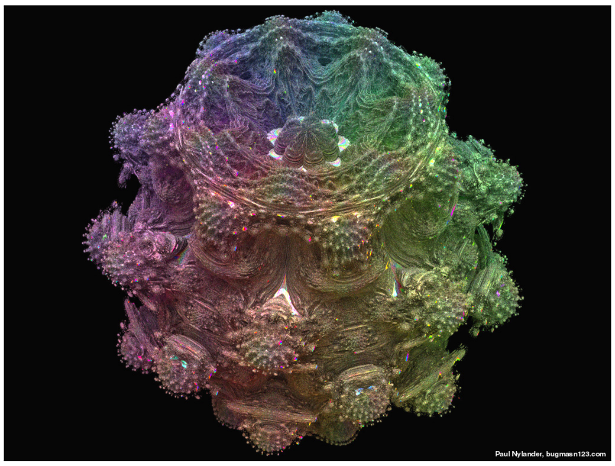


Figure 2.4: First look at the Mandelbulb, from Paul Nylander's website [3].

Since the Mandelbrot set is in two dimensions, and since complex numbers have only two components (real and imaginary), obtaining a true three-dimensional Mandelbrot set is a challenging task, and a mathematically rigorous three-dimensional Mandelbrot has not yet been found [4].

One attempt that seems to have come close was originated by Rudy Rucker in 1987. Rucker thought that expressing the three-dimensional points in spherical coordinates would allow the manipulation of the points in a similar way to the complex-number operations performed on the points in the two-dimensional Mandelbrot set [5].

Rucker did not have the computational power to accomplish a rendering of this idea, so it was put aside for twenty years, until Daniel White independently published a formula in 2007, which took the approach proposed by Rucker. White decided to approach the problem by considering the geometrical consequences of multiplying numbers in the complex plane, which amounts to rotating them [4].

White's formula produced images that looked promising, but they didn't have the level of detail that was expected from a true three-dimensional equivalent of the Mandelbrot set. A mathematician, Paul Nylander, raised White's formula to a higher power (eight), which would be equivalent to increasing the number of rotations of the point. The resulting image is shown in figure 2.4. The shape maintains excellent detail, even at high levels of magnification [4].

The new shape, known as the Mandelbulb, has roughly the same formula as the Mandelbrot set (equation 2.1):

$$Z = Z^k + C \tag{2.2}$$

where Z is raised to an arbitrary power like so:

$$Z^k = r^k(sin[k\theta]cos[k\phi], sin[k\theta]sin[k\phi], cos[k\theta]). \tag{2.3}$$

The variable r is the norm of Z (|Z|), θ is equal to arctan(Z_y/Z_x) and ϕ is equal to $|(Z_x, Z_y)|/Z_z$. The spherical coordinates of the point Z/|Z| are represented by θ and ϕ. Equations 2.2 and 2.3 are sourced from Chapter 33 of the book Ray Tracing Gems II [6].

2.3 Signed Distance Functions

Signed distance functions provide an estimate of how close a point is to the surface of a shape. If the result of the function is positive, then the point is outside the surface. If the result is negative, then the point is inside the surface. If the result is zero, then of course the point is exactly on the surface of the shape described by the function [7].

A signed distance function can be derived using the Böttcher map for the fractal formula, which is a deformation of the space. Closer to the surface of the fractal, the space is deformed to a greater degree than parts further away from the surface, as shown in figure 2.5. The deformations of the space occur in such a way as to map the exterior of the fractal to the exterior of a unit disk [8].

A rigorous mathematical explanation of the Böttcher map will not be given in this paper, but a brief introduction is helpful to understand the derivation of the distance function for the Mandelbulb. The map can be calculated as follows:

$$\phi_C[Z] = lim_{n\to\infty}[f^n[Z]]^{k^{-n}} \tag{2.4}$$

where the value of f[Z] is the same as in equation 2.2 and n refers to the current iteration of the equation [6].

Looking at equation 2.4, take a point Z that is not in the fractal, so that the function f^n[Z] grows as the number of iterations increases, tending towards infinity. For a large enough value of n, the term f$_n$[Z] will become vastly larger than the value of C (as the point is far from the fractal), meaning that C can reasonably be discarded [6].

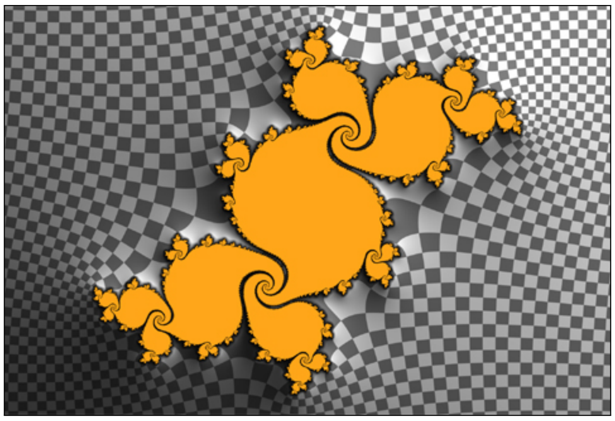


Figure 2.5: The Böttcher map generated for the Julia set (another two-dimensional complex fractal related to the Mandelbrot set), from the website of Inigo Quilez [8].

From this casting away of C we obtain:

$$f^{n+1}[Z] \approx (f^n[Z])^k \tag{2.5}$$

for points that are sufficiently far away from the surface of the fractal. Since the point at iteration n is far from the fractal, if one were to undo all of the iterations to get back to the original point at $f_0[Z]$, we would obtain this expression:

$$f_0^n[Z]^{k^{-n}} = Z \tag{2.6}$$

and by extension, returning to the Böttcher map:

$$\phi_0[Z] = Z \tag{2.7}$$

which is the approximate result that equation 2.4 gives when the function $f^n[Z]$ ultimately diverges [6].

Next, we will use something called the Hubbard-Douady potential, equal to the logarithm of the modulo of the Böttcher map. This is a map of points onto a unit disk. Recall that the Böttcher map maps the exterior of the fractal to the exterior of a unit disk. This now becomes important, as it enables us to use the Hubbard-Douady potential for all of our complex fractals. We now have a function that tends towards zero as the points approach the boundary of the fractal [8]:

$$G[Z] = lim_{n\to\infty} \frac{log|f^n[Z]|}{k^n}. \tag{2.8}$$

Equation 2.8 is not ready to be used as a distance measurement yet. However, it's possible to make it so. More detail is given in Ray Tracing Gems II, but a brief explanation will be given here. If we divide G[Z] by its gradient (obtaining this from the first-order Taylor expansion of the function), then we obtain an upper bound on the distance to the surface, which is desirable. The final equation for distance estimation therefore is [6]:

$$d(Z) = lim_{n \to \infty} \frac{|f^n(Z)| log |f^n(Z)|}{|(f^n)'(Z)|}. \tag{2.9}$$

2.4 Ray and Sphere Tracing

Ray tracing is a technique used for rendering scenes. Generally, a ray is a line that is cast from the camera or eye through each pixel of an image. Tests are performed to see which object (if any) is encountered by the ray first. By bouncing the ray from object to object, the program can gather the various light contributions from objects that reflect, emit or refract light [9].

Distance estimators can be used for ray tracing. If the approximate distance to the nearest surface can be calculated then the current point can be moved along the ray safely, until it is close enough to the surface to stop, according to this formula:

$$P_{n+1} = P_n + \mathbf{v} d(P_n) \tag{2.10}$$

where P$_{n+1}$ is the point at the next step, P$_n$ is the point at the current step, **v** is the unit vector representing the ray direction and d(P$_n$) is the distance function acting on the current point [6].

Using a distance function in this way is known as sphere tracing. The magnitude of the result of the distance function can be considered as the radius of a sphere. This sphere is guaranteed not to go through any part of the surface, making it safe to step according to the distance function result in any direction. For this guarantee to hold, the distance function must either be an exact calculation of the distance, or an underestimate [10].

See figure 2.6. This shows three scenarios for tracing a ray with the sphere tracing technique. At the bottom, the distance function is an exact measure of the distance to the nearest point on the surface. This is the ideal scenario, for correctness and performance. The top left image shows the result of a distance function which underestimates the distance each time. As a result, significantly more steps are taken along the ray, which will result in decreased performance. Lastly, the top right image shows the result of overestimating the distance. The ray ends up stepping past the boundary of the shape.

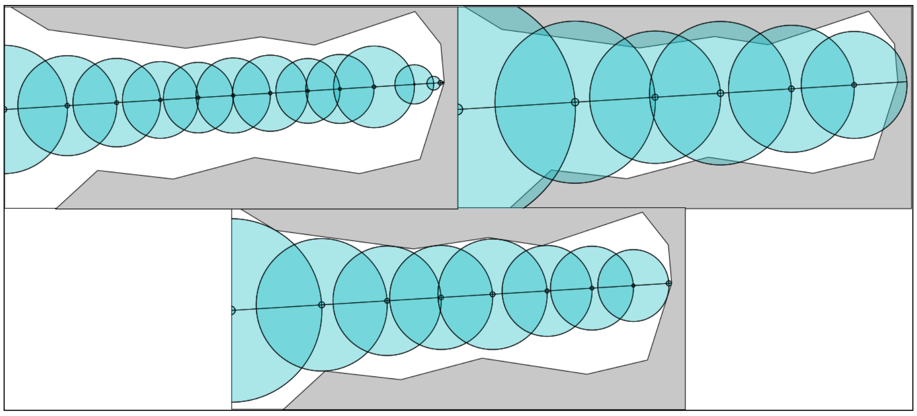


Figure 2.6: Three scenarios in sphere tracing. Top left: Distance function underestimates distance, resulting in a loss of performance. Top right: Distance function overestimates distance, resulting in a loss of accuracy. Bottom: Distance function is exact.

2.5 Signed Distance Fields

Papers:

- Signed distance fields: A natural representation for both mapping and planning [11].
- Hierarchical hp-adaptive signed distance fields [12].
- Interactive ray tracing of distance fields (page 91) [13].

- Stored values from signed distance function, stored as 2D texture or 3D grid.
- Saves having to recompute SDF all the time.
- Fractal scenes with high view distance could benefit from some pre-calculated values to give them a head start - show room of pillars and remark on frame rate (no SDF).
- Remark on memory costs.
- Briefly talk about storage methods - octrees.
- Talk about 2D methods - particularly used for fonts.

2.6 Temporal Caching

Chapter 3

Implementation

This chapter will go through the implementation of the project. First, the system information and general program structure will be looked at. Then, the rendering of fractals will be covered, followed by the attempts at improving performance, and the method of measuring performance. Finally, there will be a section on debugging.

In terms of general optimizations, there were no specific efforts to optimize the code or the basic algorithms, since the most important thing was to remain consistent across the different scenarios, and the relevant measurements were the performance differences (if any), not the raw performance.

3.1 Operating System and Hardware

The operating system used was Linux Mint 20.1. The project compiles on Linux using Make. Considering that the project was written in C, it is likely very portable (as long as the system can use Vulkan as well), but compilation on other operating systems has not been tested, and would require changes to the method of compilation.

The CPU used was an Intel©Core^{TM} i7-9750H with 6 cores.
The GPU used was an NVIDIA TU106M (GeForce RTX 2060 Mobile).
The Vulkan version being used was 1.3.205.

3.2 Program Structure and Libraries

3.2.1 Libraries Used

Vulkan

Vulkan was chosen over OpenGL because it is performant and modern, and because there are some changes on the way that will, I think, help with the efficiency of rendering using one of the optimization methods chosen (though it is not supported currently, unfortunately). Additionally, it seemed like the logical choice, as I am at the moment more familiar with Vulkan than with OpenGL and have, as of writing this, worked with it more recently. It is licensed under the Apache License 2.0 [14].

Volk

The third-party library, Volk, is a meta-loader for Vulkan. It allows one to load the Vulkan API without needing to link to Vulkan. This makes setup much easier to manage. It is licensed under the MIT license [15].

GLFW

GLFW is a library for creating windows and surfaces, which can be used for Vulkan development. It is easy to use and multi-platform, and written in C, so is the logical choice for this project. It is licensed under the zlib/libpng license [16].

3.2.2 Programming Language

The programming language of choice for this project was C. C was chosen because it is the language that the libraries used (Vulkan and GLFW) were written in. It was chosen over C++, because the usual desirable additional features, such as vectors, were not needed, so memory management was easy.

Using C over C++, some additional complexity was added due to Vulkan objects needing explicit destruction. In C++, it's possible to use the destructors of classes to handle this automatically. However, a pattern for destroying Vulkan objects was established, and did not add much extra work.

GLM is a library in C++, often used for handling matrix and vector operations. For this project, there was comparatively little need for such operations, when compared to a program that renders pre-calculated or stored geometry such as meshes, so a very lightweight set of functions was written for the few vector operations that were needed (and no matrix operations at all).

Overall, the project was intended to be lightweight and simple, and it was decided that using C was the best choice, and that using C++ would not significantly reduce complexity. C++ is certainly worth using if dynamic data structures like vectors are needed, or explicit memory management would add a lot of complexity to the code. In this project, that was not the case.

3.2.3 Vulkan Setup

The project used Vulkan for rendering. The rendering process was split into two passes, one for geometry and one for colour. This was done because the speed at which the geometry of the scene was obtained was of interest, and the speed of colouring the fractals was not, so splitting them enabled measurement of the geometry render pass on its own.

No vertices are passed in to the shaders; the vertex shaders conjure up fullscreen triangles using the vertex indices, which are then worked on by the fragment shaders, where all the calculations and colouring happen.

Specific setups for different optimization methods will be discussed in the relevant sections.

3.2.4 Program Layout

The program takes arguments which specify the setup when the program is loaded. The user can change which fractal to display (including the 2D Mandelbrot set, the Mandelbulb and the Hall of Pillars), which optimization method to use, whether to vary any fractal parameters (which can be used to animate the fractal) and whether to take any performance measurements during runtime. Some of the arguments affect which shaders are loaded, and the Vulkan setup.

The user is able to control the camera with the mouse and keyboard, speed up and slow down, and print the current position and camera front vector. Input is handled by GLFW.

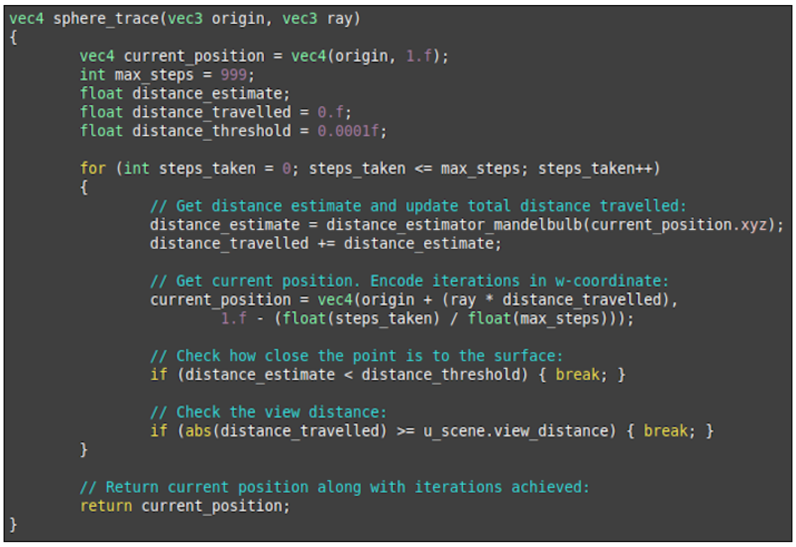
3.3 Rendering 3D Fractals

The project made use of two fractal formulae. One was for the Mandelbulb, as described in the background section. Another was chosen to provide variety, specifically with regards to the depth of the image. The Mandelbulb is neatly contained within a box, and has no "interior", but the alternative fractal (named "Hall of Pillars" by me given its lack of another name) takes up the entire screen, and is reminiscent of a room with many archways, which I thought might give different results in terms of the performance measurements when using the optimizations developed in the project.

The calculations for the fractals were done entirely within shaders, except for the generation of the 3D signed distance field, which was calculated on the CPU upon loading the program, before any rendering occurred. Single-precision floating point numbers were used for all calculations within shaders. These were chosen over double-precision numbers, as the scale of the fractals was reasonable, and high-detail zooms into the fractal were not necessary for the project.

3.3.1 Basic Sphere Tracing Implementation

The basic sphere tracing algorithm was implemented in GLSL and is largely the same across the fractal types and optimization methods. Figure 3.1 below shows the implementation.



```
vec4 sphere_trace(vec3 origin, vec3 ray)
{
        vec4 current_position = vec4(origin, 1.f);
        int max_steps = 999;
        float distance_estimate;
        float distance_travelled = 0.f;
        float distance_threshold = 0.0001f;

        for (int steps_taken = 0; steps_taken <= max_steps; steps_taken++)
        {
                // Get distance estimate and update total distance travelled:
                distance_estimate = distance_estimator_mandelbulb(current_position.xyz);
                distance_travelled += distance_estimate;

                // Get current position. Encode iterations in w-coordinate:
                current_position = vec4(origin + (ray * distance_travelled),
                        1.f - (float(steps_taken) / float(max_steps)));

                // Check how close the point is to the surface:
                if (distance_estimate < distance_threshold) { break; }

                // Check the view distance:
                if (abs(distance_travelled) >= u_scene.view_distance) { break; }
        }

        // Return current position along with iterations achieved:
        return current_position;
}
```

Figure 3.1: GLSL code snippet of the sphere tracing algorithm.

The most important things that must remain consistent between different optimizations on the same fractal are the termination conditions, which are as follows:

- The maximum number of iterations allowed.
- The distance threshold.
- The view distance, contained in the scene uniform.

3.3.2 Mandelbulb Fractal

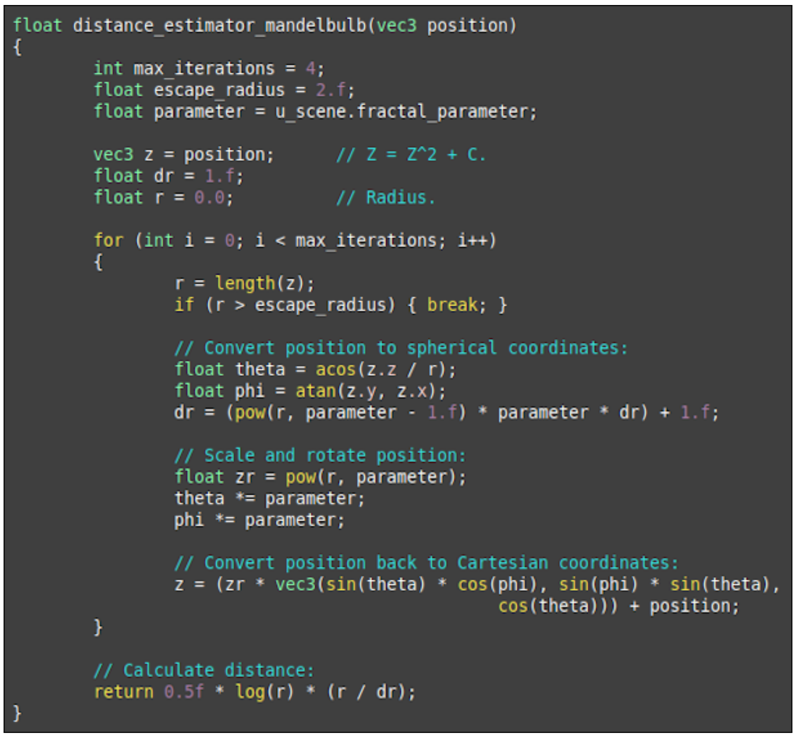
Shader implementation of signed distance function for Mandelbulb
View distance limit

3.3.3 Alternative Fractal

Got fractal shader code from Shadertoy (link, mention license), have named it "Hall of Pillars". Show signed distance function shader.
View distance limit is higher because fractal is bigger.

3.3.4 Colour, Ambient Occlusion and Fog

Be very brief here, colour isn't important. Fog is for graceful cutoff of view instead of sharp drop.



```
float distance_estimator_mandelbulb(vec3 position)
{
        int max_iterations = 4;
        float escape_radius = 2.f;
        float parameter = u_scene.fractal_parameter;

        vec3 z = position;      // Z = Z^2 + C.
        float dr = 1.f;
        float r = 0.0;          // Radius.

        for (int i = 0; i < max_iterations; i++)
        {
                r = length(z);
                if (r > escape_radius) { break; }

                // Convert position to spherical coordinates:
                float theta = acos(z.z / r);
                float phi = atan(z.y, z.x);
                dr = (pow(r, parameter - 1.f) * parameter * dr) + 1.f;

                // Scale and rotate position:
                float zr = pow(r, parameter);
                theta *= parameter;
                phi *= parameter;

                // Convert position back to Cartesian coordinates:
                z = (zr * vec3(sin(theta) * cos(phi), sin(phi) * sin(theta),
                                cos(theta))) + position;
        }

        // Calculate distance:
        return 0.5f * log(r) * (r / dr);
}
```

Figure 3.2: GLSL code snippet of the distance estimator function for the Mandelbulb fractal.

Number of iterations is good in this case for viewing performance differences, but also for ambient occlusion. So it is combined with the colour.

Number of iterations is raised to a power, since the number of iterations allowed is very high (999). Raising it to a power amplifies the differences. This is done consistently so won't affect comparison of images, but will make it easier and more visible.

3.4 3D Signed Distance Field

Centered around fractal - Mandelbulb

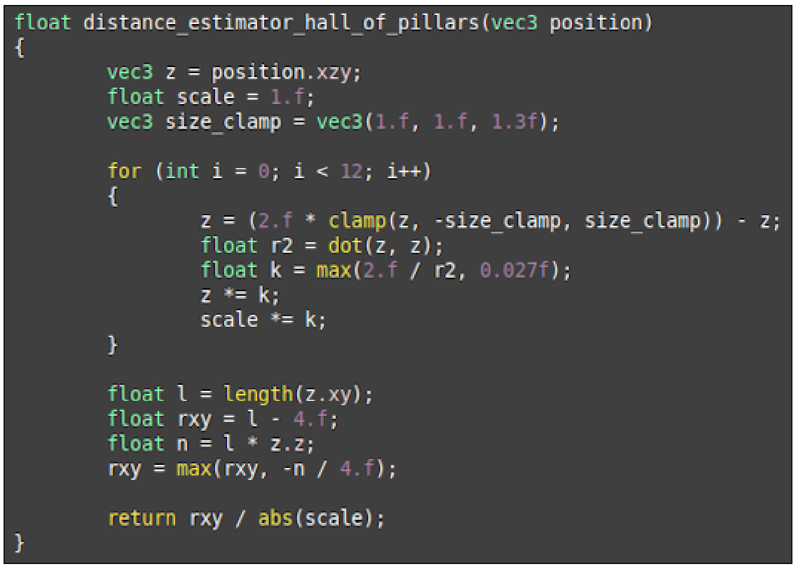
Centered around camera - Hall of Pillars?

3.5 Temporal Caching

Single image - memory of 4 previous frames.

Why did I choose length / 4? Be more scientific about it.

Talk about how you dealt with movement - temporal reprojection? Wanted to deal with the specific scenario where the distance recorded is greater than the true distance. My method relies on the landscape not being smooth, so these changes happen often. Maybe find another method.



```
float distance_estimator_hall_of_pillars(vec3 position)
{
        vec3 z = position.xzy;
        float scale = 1.f;
        vec3 size_clamp = vec3(1.f, 1.f, 1.3f);

        for (int i = 0; i < 12; i++)
        {
                z = (2.f * clamp(z, -size_clamp, size_clamp)) - z;
                float r2 = dot(z, z);
                float k = max(2.f / r2, 0.027f);
                z *= k;
                scale *= k;
        }

        float l = length(z.xy);
        float rxy = l - 4.f;
        float n = l * z.z;
        rxy = max(rxy, -n / 4.f);

        return rxy / abs(scale);
}
```

Figure 3.3: GLSL code snippet of the distance estimator function for the alternative "Hall of Pillars" fractal. Credit goes to Dave Hoskins for the formula [17].

3.6 Performance Measurement

3.6.1 Data Types Collected

Mention image copying time and how it could be avoided altogether, so didn't measure.

Measured geometry pass time.

Median, min, max, frame time.

3.6.2 Representative Views

Hall of Pillars - Room with archway

Hall of Pillars - Flat ground

Hall of Pillars - Room with huge distance differences and bottlenecks

Hall of Pillars - Room with very short distance to back, maybe bottlenecks

Mandelbulb - Full view, lots of white space

Mandelbulb - Zoomed in so it covers the screen

Mandelbulb - Try to find a view with bottleneck

Mandelbulb - Zoom in to get detail

3.6.3 Animation

Hall of Pillars - Flythrough

Hall of Pillars - Varying parameter?

Mandelbulb - Varying parameter

Mandelbulb - Fly across? Probably not.

3.7 Debugging and Optimization

Manual debugging - printing out Vulkan handles, error handling with messages, printing current position/rotation, using Renderdoc to view different pipeline stages to make sure they are coming out correctly. Adding special return values in shaders to output black if something has not gone right.
No extra optimization has been done, since the project is focused on performance difference between methods, not on raw performance numbers. Therefore, optimizations such as early ray culling (the Mandelbulb fractal, for example, is contained more or less within a cube with side lengths of 2.2) have not been implemented.

Chapter 4

Results

4.1 3D SDF

4.1.1 Mandelbulb

4.1.2 Hall of Pillars

4.2 2D SDF

4.2.1 Mandelbulb

4.2.2 Hall of Pillars

Show four images:

- Top-Left: No SDF, no movement.
- Bottom-Left: No SDF, movement.
- Top-Right: 2D SDF, no movement.
- Bottom-Right: 2D SDF, movement.

Use these to demonstrate that the performance benefits only exist for still images generally.
Also, have an image of the 2D SDF with movement and no checks for changes in distance travelled - show artefacts and performance. Explain that if you want a performance boost all the time, you need to put up with these artefacts.
In "further work", maybe could say that it would be great to find a different method of removing artefacts without removing performance benefits of SDF. Mention that you tried sampling from an area but it was wobbly, and mention that adding a small decay didn't work either because of the difference in depth between the closer and farther parts of the image, and the performance benefit comes from skipping the edges anyway. Would have to find a way of detecting these artefacts and removing them.
Include results separately for having the second timestamp before and after the image copying. Remark that the shader execution time is significantly smaller but that the copying offsets that. Also investigate why on earth the framerate is so much better, but the timestamps don't reflect this...

Chapter 5

Conclusion

Things to mention:

- Ambient occlusion and how reducing the number of iterations ruins it.

Further work:

- Figure out a way to remove artefacts from movement without resetting distance for everything. Artefact detection.
- Find a way to copy the 2D SDF image faster.
- Make the 3D SDF move with the camera so that it can handle movement through the scene.
- Move the generation of the 3D SDF to the shaders, maybe make it coarser so it can be regenerated every frame (so it can handle animation)?

References

[1] R. L. Devaney, "The mandelbrot set, the farey tree, and the fibonacci sequence," *The American Mathematical Monthly*, vol. 106, no. 4, pp. 289–302, 1999.

[2] D. Ashlock, "Evolutionary exploration of the mandelbrot set," in *2006 IEEE International Conference on Evolutionary Computation*, pp. 2079–2086, IEEE, 2006.

[3] P. Nylander, "Hypercomplex fractals." `http://www.bugman123.com/Hypercomplex`, 2009. accessed: 31/07/22.

[4] J. Aron, "The mandelbulb: first 'true'3d image of famous fractal," *New Scientist*, vol. 204, no. 3736, pp. 54–55, 2009.

[5] R. Rucker, "In search of a beautiful 3d mandelbrot set." `https://www.rudyrucker.com/blog/notebooks/rucker_mandelbulb_ver7_sept24_2009.pdf`, 2009. accessed: 31/07/22.

[6] A. Marrs, P. Shirley, and I. Wald, *Ray Tracing Gems II: Next Generation Real-Time Rendering with DXR, Vulkan, and OptiX.* Springer Nature, 2021.

[7] C. Robles, H. Lee, S. Yin, and V. Dhar, "Procedural rendering w/ray marching,"

[8] I. Quilez, "Distance to fractals." `https://iquilezles.org/articles/distancefractals`, 2004. accessed: 5/08/22.

[9] E. Haines and T. Akenine-Möller, *Ray Tracing Gems: High-Quality and Real-Time Rendering with DXR and Other APIs.* Springer, 2019.

[10] J. C. Hart, "Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces," *The Visual Computer*, vol. 12, no. 10, pp. 527–545, 1996.

[11] H. Oleynikova, A. Millane, Z. Taylor, E. Galceran, J. Nieto, and R. Siegwart, "Signed distance fields: A natural representation for both mapping and planning," in *RSS 2016 workshop: geometry and beyond-representations, physics, and scene understanding for robotics*, University of Michigan, 2016.

[12] D. Koschier, C. Deul, and J. Bender, "Hierarchical hp-adaptive signed distance fields.," in *Symposium on Computer Animation*, pp. 189–198, 2016.

[13] O. Jamriška and V. Havran, "Interactive ray tracing of distance fields," in *Central European Seminar on Computer Graphics*, vol. 2, pp. 1–7, Citeseer, 2010.

[14] K. Group, “Vulkan github repository license.” `https://github.com/KhronosGroup/Vulkan-Headers/blob/main/LICENSE.txt`, 2022. accessed: 7/08/22.

[15] Zeux, “Volk github repository license.” `https://github.com/zeux/volk/blob/master/LICENSE.md`, 2022. accessed: 7/08/22.

[16] GLFW, “Glfw license.” `https://www.glfw.org/license.html`, 2022. accessed: 7/08/22.

[17] D. Hoskins, “Fractal explorer.” `https://www.shadertoy.com/view/4s3GW2`, 2016. accessed: 9/08/22.

Appendices

Appendix A

External Material

Nulla malesuada porttitor diam. Donec felis erat, congue non, volutpat at, tincidunt tristique, libero. Vivamus viverra fermentum felis. Donec nonummy pellentesque ante. Phasellus adipiscing semper elit. Proin fermentum massa ac quam. Sed diam turpis, molestie vitae, placerat a, molestie nec, leo. Maecenas lacinia. Nam ipsum ligula, eleifend at, accumsan nec, suscipit a, ipsum. Morbi blandit ligula feugiat magna. Nunc eleifend consequat lorem. Sed lacinia nulla vitae enim. Pellentesque tincidunt purus vel magna. Integer non enim. Praesent euismod nunc eu purus. Donec bibendum quam in tellus. Nullam cursus pulvinar lectus. Donec et mi. Nam vulputate metus eu enim. Vestibulum pellentesque felis eu massa.

Appendix B

Ethical Issues Addressed