Real-time Rendering of 3D “Fractal-like” Geometry

Deliverable 1: Final Year Dissertation

Solomon Baarda
SOLOMON BAARDA [Company address]

Abstract

Ray tracing getting popular, ray marching

I, Solomon Baarda confirm that this work submitted for assessment is my own and is expressed in my own words. Any uses made within it of the works of other authors in any form (e.g., ideas, equations, figures, text, tables, programs) are properly acknowledged at any point of their use. A list of the references employed is included.

Signed:

Date:

Table of Contents

1 Introduction

https://github.com/SolomonBaarda/dissertation

1.1 Aims & Objectives

The aim of this project is to develop a prototype real-time rendering engine, capable of displaying complex 3D "fractal-like" geometry. The performance of the engine will be benchmarked across various systems to determine whether the "real-time" aspect of the project has been achieved.

Create a realtime 3d fractal rendering engine

Must have good performance to be realtime

Find a good balance between looking good and performance

Need approximations

1.2 Project Description

Is the topic meaningful, complex and challenging?

2 Literature Review

2.1 Realtime Rendering Methods

2.2 Ray Tracing

Ray tracing is a method of rendering 3D environments, often with photorealistic detail. In ray racing, a ray (simply a line in 3D space) is extended or traced forwards from the camera position until it collides with the surface of an object. From there, the ray can be absorbed or reflected by the surface, taking into consideration light absorption, reflection, refraction, and fluorescence.

Ray tracing is ideal for photorealistic graphics, as it takes into consideration many of the properties of light, but because of this it is computationally very expensive. Often, true ray tracers are not real-time, and they can take up to hours to render a single frame of video. To make a ray tracer capable of rendering in real-time, many approximations must be made when calculating the image. Hybrid approaches are often used.

One of the strengths (but also limitations) of ray tracing is that an accurate ray-surface intersection function must exist for every object in the scene. This is well suited for any Euclidian surfaces, such as primitives and meshes, which are made up of vertices, faces and edges, as points of intersection can be calculated relatively easily on these shapes.

This approach, however, is not suitable for any geometries for which a ray-surface intersection function does not exist [1], such as fractal geometries.

2.3 Ray Marching

Ray marching is a variation of ray tracing, which differs in the method of detecting collisions between the ray and objects. Ray marching uses an iterative approach, where the current point is moved/marched along the ray in small increments until it lands on the surface of an object.

This approach is more computationally complex in the worst case when compared to ray tracing, however, it does provide several benefits. Most notably, ray marching does not require a surface collision function like ray tracing does, so it can be used to render geometry for which these functions do not exist. This allows more complex shapes to be rendered.

Ray marching also has infinite precision when zooming in, as no polygons are used.

Use ray marching optimisation techniques

Ray marching – lots of effects (ambient occlusion, hard or soft shadows, glow, fog) are free

To determine how feasible a real-time rendering of 3D environments using ray marching.

2.3.1 Signed Distance Function

A distance function is a function which given any point in 3D space, will return the distance to the surface of the closest object. A signed distance function is simply a distance function which contains a positive sign if the point is outside of the object, and a negative sign if the point is inside of the object. If a distance function returns 0 for any point, then the point must be exactly on the surface of an object.

DIAGRAM TODO

The sign returned by the distance function is very useful as it allows the ray marcher to determine if a camera ray is inside of an object or not, and from there it can use that information to render the objects differently.

In the scene below

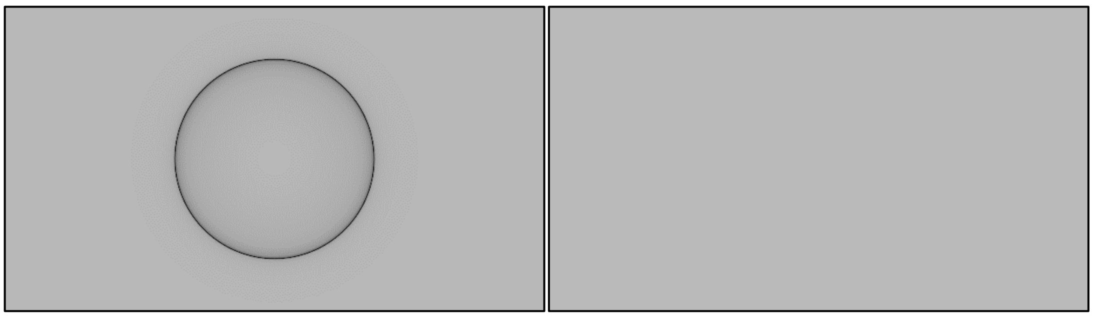

Figure 1 - DF and SDF

2.3.2 Primitives

Signed distance functions exist for most primitive 3D shapes, such as a sphere, box, plane etc.

For a sphere with radius R, positioned on the origin 0,0,0, the

$$sphereSDF(p) = |p| - R$$

$where\ p\ is\ a\ vector\ in\ the\ form\ \{x, y, z\}\ and\ R\ is\ the\ circle\ radius\ in\ world\ units$

Deriving an SDF

2.3.3 Alterations & Combinations

Signed distance functions can be translated, rotated, and scaled.

Signed distance functions can be combined using union, subtraction, and intersection operations.

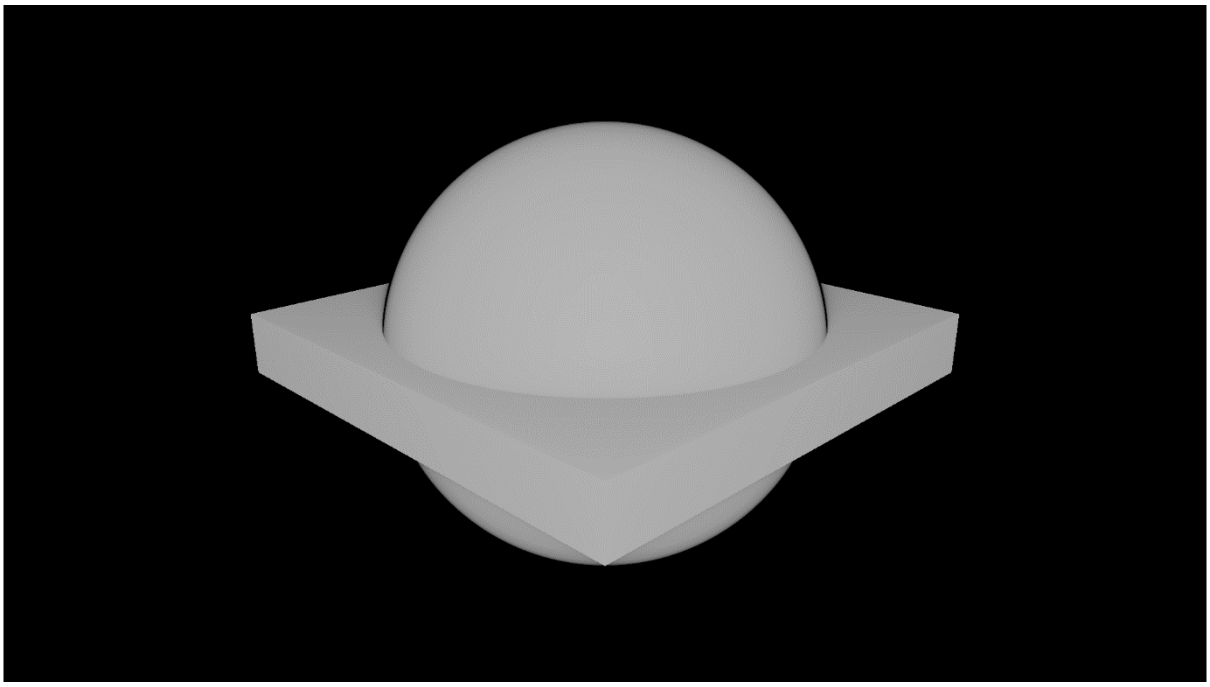

Figure 2 - Union of Sphere and Box

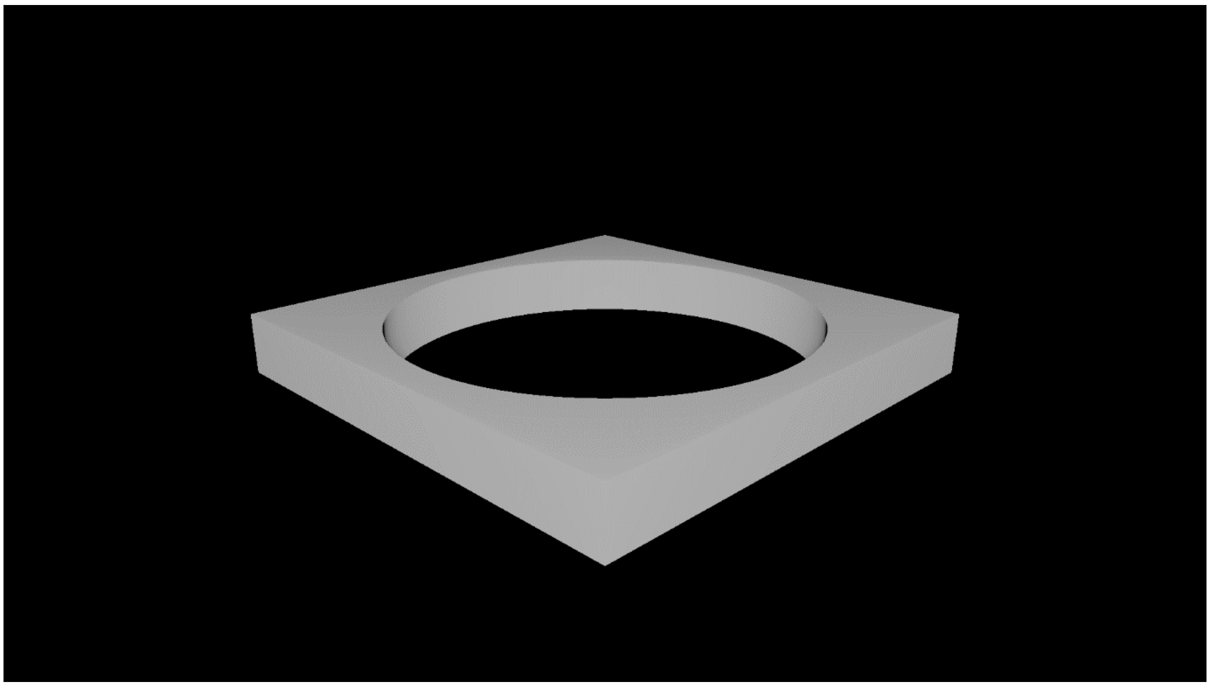

Figure 3 – Intersection of Sphere and Box

Signed distance functions can also be combined using a version that uses smoothing.

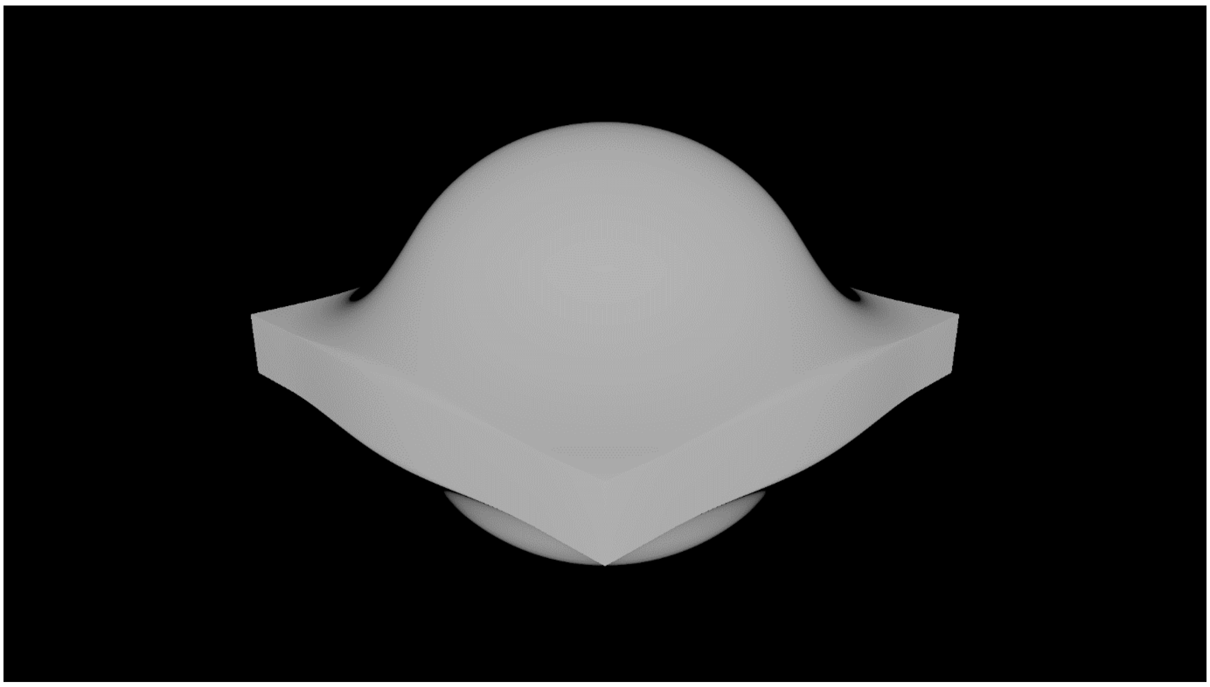

Figure 4 - Smooth Union of Sphere and Box

There are several alterations that can be applied to primitives once we have their signed distance function. A primitive can be elongated along any axis, its edges can be rounded, it can be extruded, and it can be "onioned" – a process of adding concentric layers to a shape. All these operations are very cheap.

Signed distance functions can also be repeated, twisted, bent, and surfaces displaced using an equation e.g., a noise function or sin wave.

2.3.4 Surface Normal

The surface normal of any point on the surface of an SDF can be determined by probing the SDF on each axis, using an arbitrary epsilon value.

$$\begin{aligned} normal = normalise(\{\, & SDF(p + \{e, 0,0\}) - SDF(p - \{e, 0,0\}), \\ & SDF(p + \{0, e, 0\}) - SDF(p - \{0, e, 0\}), \\ & SDF(p + \{0,0, e\}) - SDF(p - \{0,0, e\}) \,\}) \end{aligned}$$

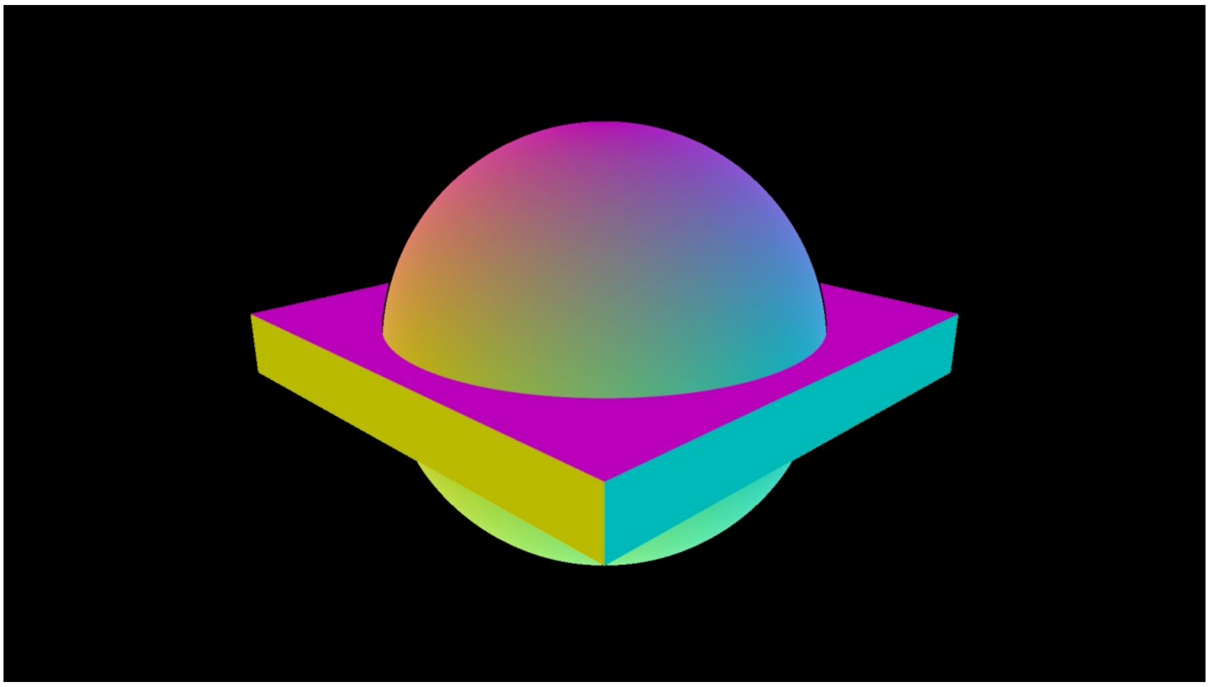

Figure 5 - Surface Normal of Sphere and Box Scene

2.3.5 Fractals

In mathematics, a fractal is a complicated pattern built from simple repeated shapes, which are reduced in size every time they are repeated. https://dictionary.cambridge.org/dictionary/english/fractal These shapes are self-similar, though not often symmetrical.

The idea of fractal geometry appeared in the late 1970s, inspired through the work of Benua Mandelbrot and his book "Fractals: form, chance and dimension", released 1977. This book introduced the concept of a "fractal dimension", a measure of the complexity of how the detail in a pattern will change in respect to the scale at which it is measured.

How they are calculated – running sum etc

3D fractals

2.3.6 Collision detection

Collision detection is possible

Marble marcher

2.4 Existing Projects

2.4.1 Fragmentarium

https://github.com/Syntopia/Fragmentarium

https://github.com/3Dickulus/FragM

2.4.2 Smallpt

https://www.kevinbeason.com/smallpt/

2.4.3 Ray Tracing in One Weekend

https://github.com/RayTracing/raytracing.github.io

3 Requirements Analysis

3.1 Use Cases

3.2 Requirements Specification

3.2.1 Functional Requirements

ID	Name	Description	Priority	Justification
		The application must be capable of rendering scenes in real time	MUST	
		The application must		

3.2.2 Non-functional Requirements

ID	Name	Description	Priority	Justification
	Executable	The application must run from a compiled executable	MUST	
	Display resolutions	The application must support the following common display resolutions: 1366x768, 1920x1080, 2560x1440 and 3840x2160		
	Maximum framerate	The application should limit the maximum framerate		

4 Software Design

4.1 Structure

4.2 Technologies

5 Evaluation Strategy

6 Project Plan

6.1 Design Methodology

6.2 Legal, Ethical & Social Issues

A well-researched consideration of any Professional, Legal, Ethical, and Social Issues pertinent to the project. (e.g. codes of conduct (BCS), codes of practice, standards, computer law, ethical decision making, intellectual property, social aspects, copyright, data protection, and so on)

6.3 Risk Analysis

6.4 Timetable

7 References

[1] J. C. Hart, D. J. Sandin and L. H. Kauffman, “Ray Tracing Deterministic 3-D Fractals,” 1989.

8 Appendices