

Marching Beyond

Julian Allchin - jallchin
Diego Bustamante - diegobus

Implementation Summary

For our final project, we implemented raymarching in Unity to render signed distance functions (SDFs). We first implemented simple SDFs to model primitives, including spheres, cubes, toruses, cylinders, and prisms. Then, we implemented modification functions to combine, blend, and cut these primitives (**Fig. 1** and **Fig. 2**) in order to create shapes that would be non-trivial to render using traditional triangular meshes (**Fig. 3**). After these basic ray-marching tasks, we extended our implementation to render trippy fractal SDFs such as the Mandelbulb and an environment-wide [folding-fractal](#) (**Fig. 4**). To make an immersive user demo of the power of raymarching, we then implemented a two-pass rendering scheme in Unity to allow a user to navigate a cute spacecraft through this alien-like fractal world (**Fig. 5**).

You can access the GitHub for this project here: [MarchingBeyond](#)

Figures

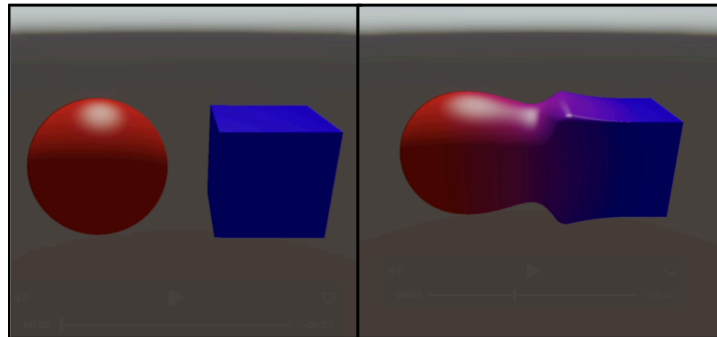


Figure 1. Combining Primitives with Smooth-Min Approach

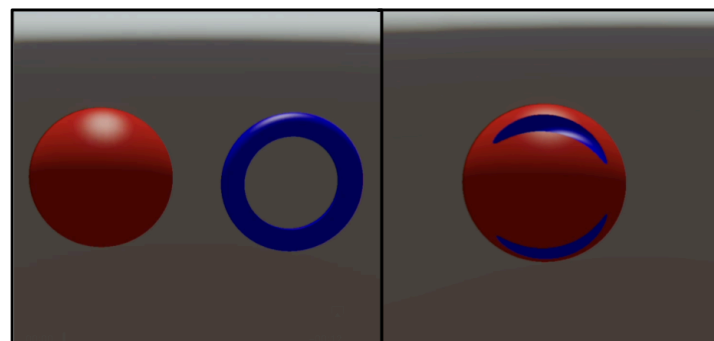


Figure 2. Cutting Primitives by Comparing Shape Distances

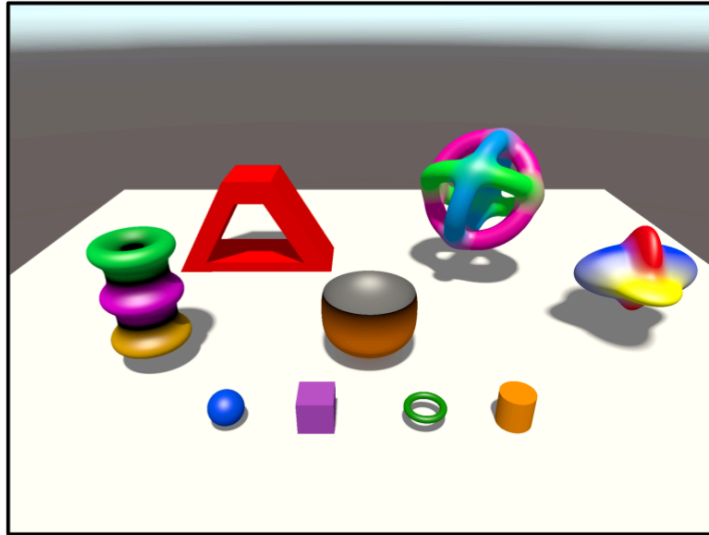


Figure 3. Collection of primitives and complex shapes made by blending and cutting

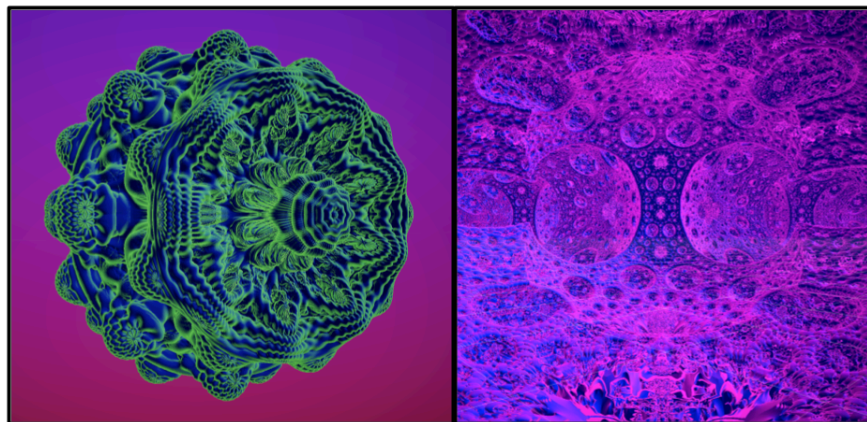


Figure 4. Mandelbulb and Folding Fractal SDFs




Figure 5. Frame of interactive spaceship in fractal demo rendered using SDF




Implementation Tasks

While we were successful in completing all of our “Core Implementation” features, we did not complete all of our “Nice to Have” elements. Unfortunately, VR equipment was not available from the sources we had intended to borrow from, and we decided to pivot to an intuitive third-person navigation demo instead. Here are the tasks from our proposal:

Core Implementation

- **Ray marching (in Unity)**  **Fully implemented**
 - Implement HLSL fragment shaders for efficient ray marching
 - Create a library of 3D signed distance functions for various shapes
 - Spheres, cubes, tetrahedrons
 - Develop methods for combining SDFs through operations like union, intersection, and subtraction
 - Be able to render other abstract SDFs such as [mandelbulb](#).

Nice to Have

- **Two-Pass Rendering Solution**  **Fully implemented**
 - First pass: Render spacecraft interior using traditional techniques
 - Second pass: Render windows using ray marching to show SDF worlds
 - Composite both passes for the final frame
- **VR Integration**  **Not Implemented**
 - Set up VR camera and controls compatible with our rendering approach
 - Implement intuitive navigation through SDF worlds
- **Spacecraft Design**  **Fully implemented by modifying online asset ([link](#))**
 - Develop a visually interesting spacecraft interior with large viewing windows
 - Create control interfaces that allow selecting different SDF environments
 - Implement visual feedback that helps players understand their movement through SDF space

Implementation Challenges

Both scenes

- Camera transformations and perspective issues
 - Minor differences in the FOV of the camera would misalign the ray marched shapes with the unity transform gizmo.

SDF

- Connecting our raymarching engine with Unity’s built-in lighting to add highlights and object-to-object shadows

- Ended up using a similar light transport simulation to path tracing, where for each hit, we cast another ray toward the light source to determine shadows

Fractal

- Collisions for the spaceship
 - For each frame, we checked the distance to the fractal on the CPU and pushed the ship away if it got too close
- Rendering with raymarching and rasterization
 - The ship is rendered with rasterization, and the fractal is rendered with raymarching. We implemented a two-pass rendering approach, essentially drawing the ship as an overlay on top of the fractal.
 - This still needs improvement, as the ship is ALWAYS drawn on top, leading to confusing scenarios where the ship is behind the fractal yet still drawn on top.