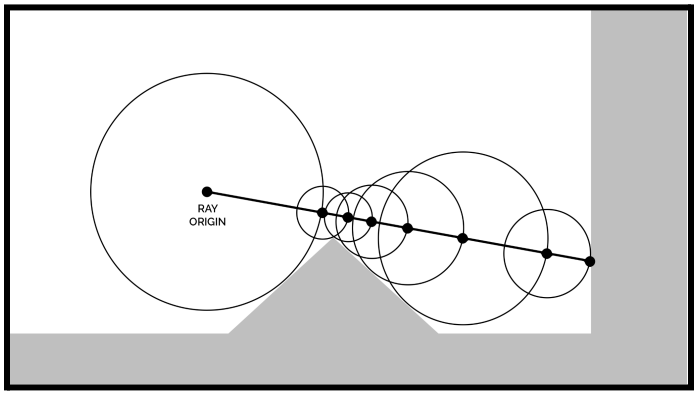
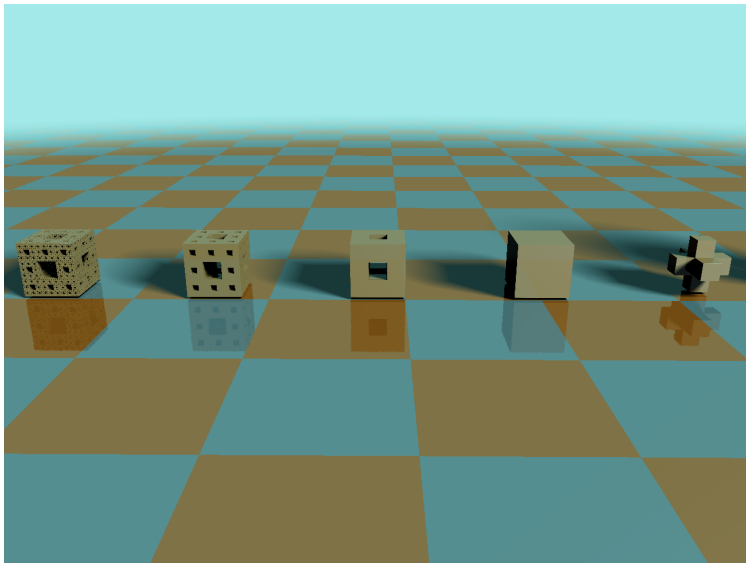
Ray Marching Fractals, Volumes, and Other Things
By: Alec Neiman

This project attempts to explore the computer graphics technique of raymarching. This technique involves casting rays from the camera into the scene and stepping them forward until they hit something. In order to speed up the process of ray marching, this project makes use of signed distance functions (SDFs) in order to march the rays as far forward as possible. These SDFs are also used to evaluate when a ray has actually hit a surface by checking if the distance is under a certain threshold.

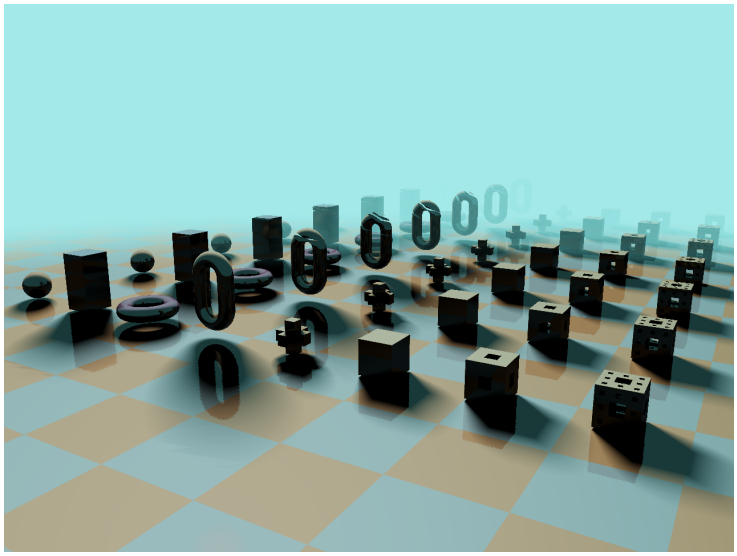


- *Visualization of ray marching using SDFs*

In addition to speeding up the ray marching process, these SDFs are able to be easily combined, negated, duplicated, intersected, etc. using very cheap and simple operations such as min, max, mod, and boolean checks. This allows for the generation of complex surfaces from simpler components.

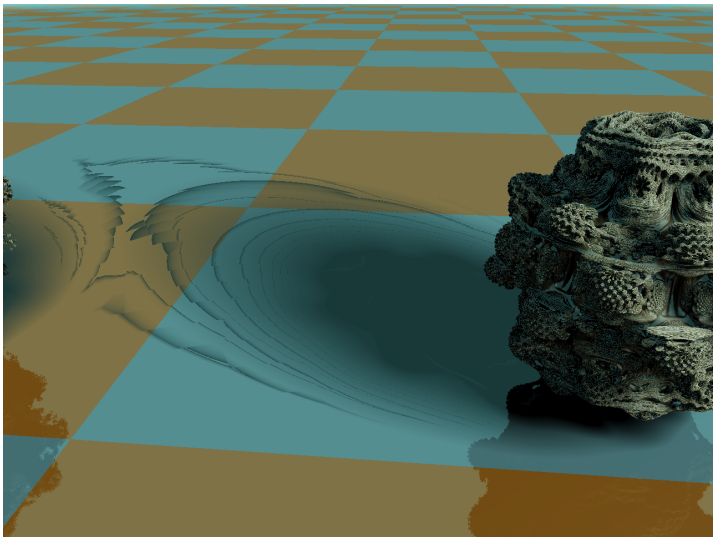


- *Generation of a menger sponge by iteratively negating a cross from a cube*



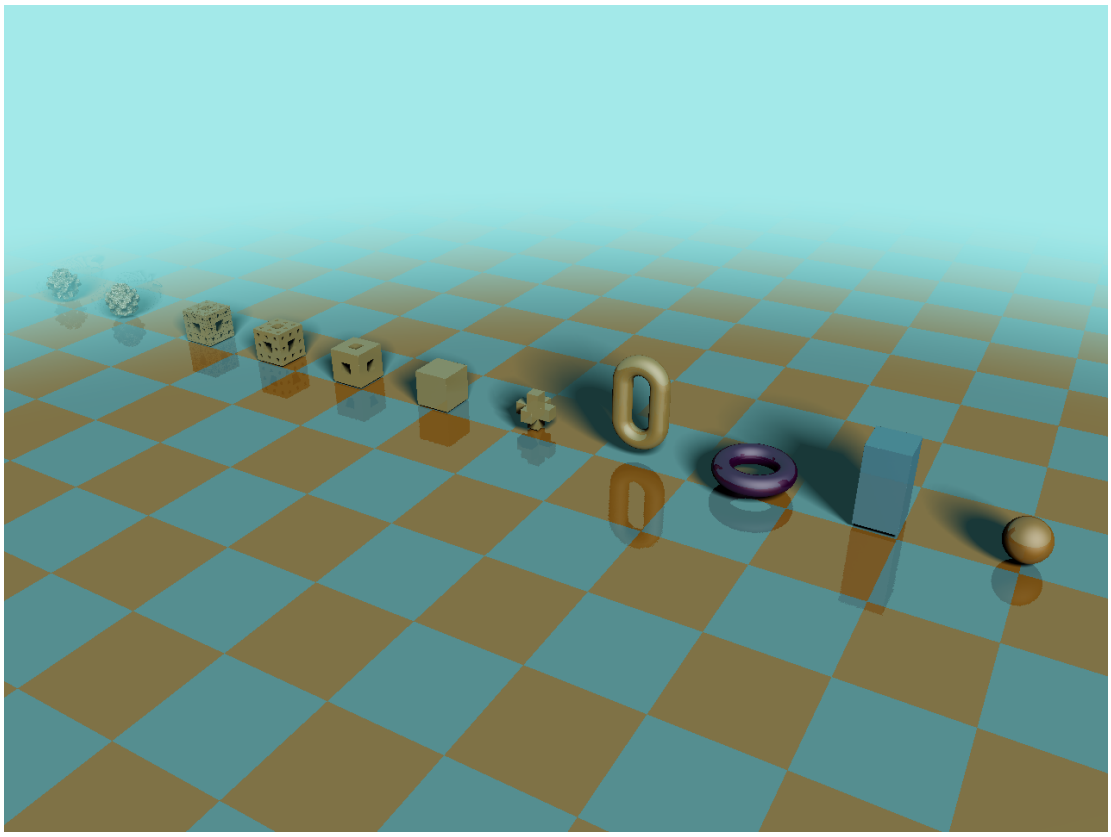
- *Duplicating SDFs by taking the modulus of the plane intersection z position*

Another advantage of using this technique over rasterizing triangles is that neighboring geometry is very easy to access. Reflections and shadows are able to be calculated by simply shooting out shadow and reflection rays from the intersection point, just like the project 1 raytracer. Soft shadows are also very cheap, and are generated by taking into account near misses by the shadow rays.

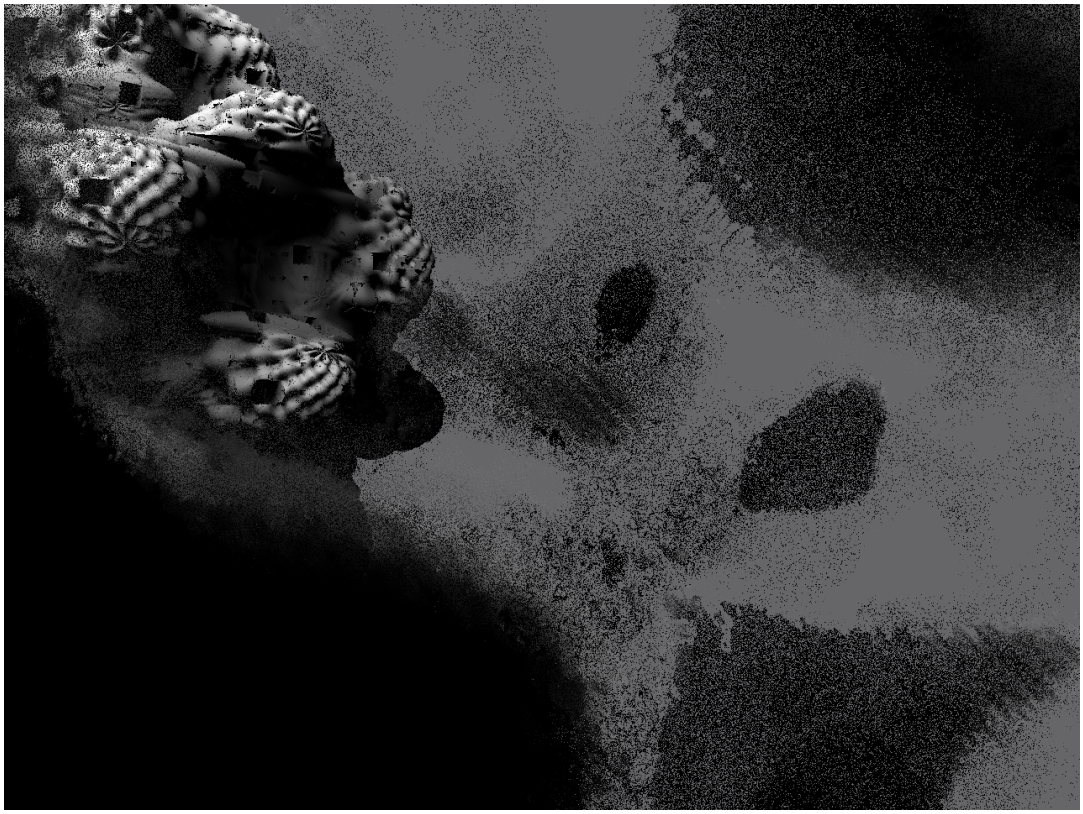


- *Although soft shadows are simple to calculate, tuning them to avoid banding artifacts can be fairly difficult with certain SDFs.*

This project consists of two scenes, one showcases a selection of SDFs rendered using ray marching on a flat tiled grid, and the other shows a large mandelbulb fractal with menger-like holes carved out of it, surrounded by dark clouds. Both scenes are able to be easily rendered in real time, although the 2nd scene can be a bit laggy on certain GPUs. The threshold for when a ray is considered to be "hitting" a surface is determined by the distance between the camera and the intersection point. This vastly speeds up rendering time by only rendering fine detail when it needs to be seen. Another significant optimization was the inclusion of bounding boxes around expensive scene areas, these boxes act as both acceleration structures for groups of objects, as well as cheap wrappers around certain expensive to calculate SDFs (mandelbulb, menger sponge).



- *Screenshot of the first scene.*



- *Screenshot of the second scene.*

Although the dark clouds in the second scene do not use SDFs to assist in rendering, they are still raymarched. The step size for the cloud is determined by the distance from the camera, as well as the current density surrounding the ray as it marches. The shape of the clouds are determined by a 3D perlin noise function with several frequency octaves. The color is determined by adding up the sample densities, multiplying them by their length, and applying Beer's law. You may also notice that the large amount of random noise in the scene, this is the result of randomly varying the step size of the marching rays. This serves two purposes, one is to avoid banding artifacts, and the other is to act as a cheap method to smoothly transition from the clouds to the gray background.