Raymarching Toolkit for Unity

A highly interactive Unity toolkit for constructing signed distance fields visually

Kevin Watters, Fernando Ramallo

Submitted 13 February 2018

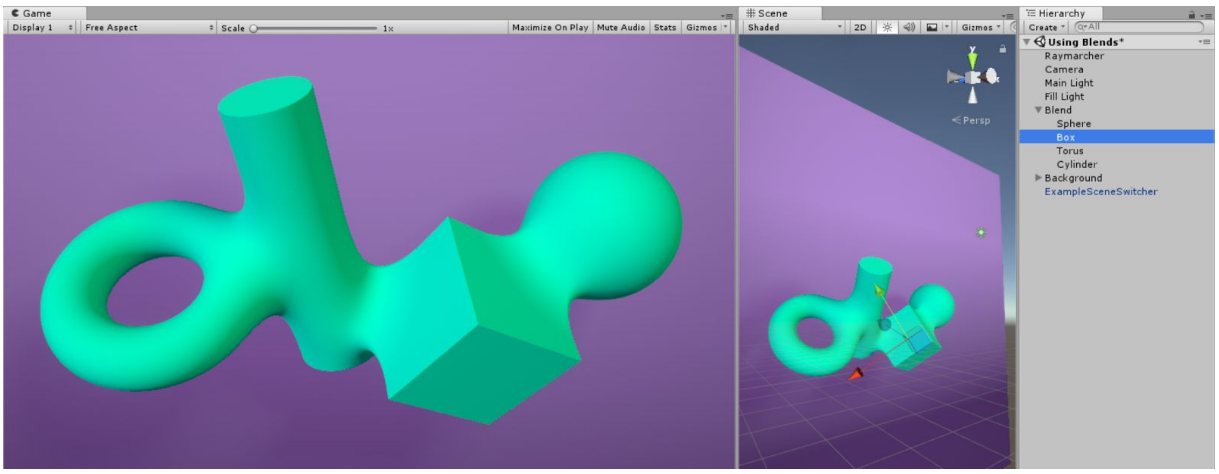


Figure 1: Shapes blended together in Unity

Abstract

Raymarching signed distance fields is a technique used by graphics experts and demoscene enthusiasts to construct scenes with features unusual in traditional polygonal workflows--blending shapes, kaleidoscopic patterns, reflections, and infinite fractal detail all become possible and are represented in compact representations that live mostly on the graphics card. Until now these scenes have had to be constructed in shaders by hand, but the Raymarching Toolkit for Unity is an extension that combines Unity's highly visual scene editor with the power of raymarched visuals by automatically generating the raymarching shader for the scene an artist is creating, live.

Keywords: Signed distance functions, raymarching, content-creation tools, rendering

Introduction

Graphics enthusiasts have for many years used signed distance fields and raymarching to construct scenes by casting rays at mathematical primitives instead of rasterizing polygons [Hart, 1996 https://graphics.cs.illinois.edu/sites/default/files/zeno.pdf]. The technique has been especially popular in the demoscene community, because of both the compact representation you can leverage to fit a complicated scene into a tiny mathematical description, and the unique rendering styles and possibilities it affords.

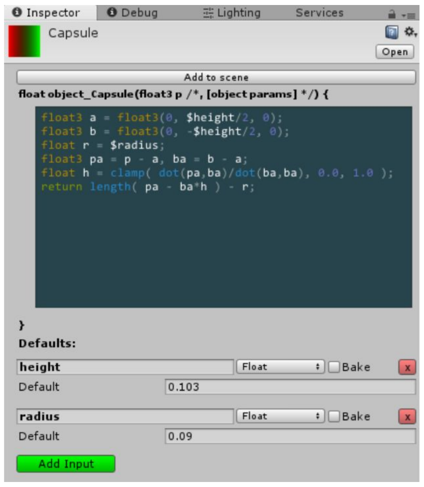
Traditionally SDF scenes of this kind are created by hand in shaders by blending together primitives with code. The authors, coming from a game development background, were more accustomed to the expressive and highly visual Unity Editor, where a "scene camera" shows you the scene you're constructing live, as you create it, and where a full-featured suite of gizmos, handles, and controls lets you tweak the position, rotation, and scale of objects in your scene quickly. They wanted to bring that expressive and more artistic flow state to the world of raymarching.

Thankfully, the Unity Editor is very extensible, and it's possible to write tools that add new functionality to the editor itself. They set about imagining a tool for Unity that removes the grudge-work from SDF scene creation, and the Raymarching Toolkit for Unity is the result.

Methods

To make a shader, we created a graph-rewriting algorithm to take Unity's hierarchy of scene objects, recognizing special Components that represent raymarched objects, blends, etc., and using that to create an intermediate representation (a binary tree) of "raymarched operations." From that representation it is trivial to generate shader code (in Unity's case, an HLSL variant) that renders the scene correctly.

To make the toolkit extensible, we introduce the idea of "snippets"--bits of shader code you can write to create building blocks from which the toolkit can create objects and blends.



Capsule

```
float object_Capsule(float3 p /*, [object params] */) {
    float3 a = float3(0, $height/2, 0);
    float3 b = float3(0, -$height/2, 0);
    float r = $radius;
    float3 pa = p - a, ba = b - a;
    float h = clamp( dot(pa,ba)/dot(ba,ba), 0.0, 1.0 );
    return length( pa - ba*h ) - r;
}
```

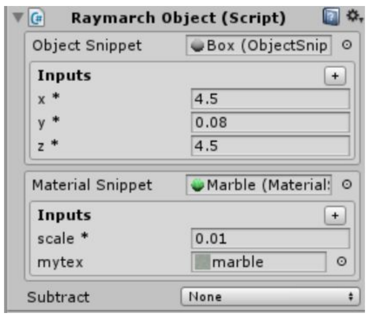
Defaults:

height Float Bake
Default 0.103

radius Float Bake
Default 0.09

Figure 3: Editing a code snippet that defines a capsule shape in the raymarcher.

Then, when it comes time to construct your scene, you use the values you defined as "inputs" to the snippet and tweak them live to create the scene you want.



Raymarch Object (Script)

Object Snippet: Box (ObjectSnip

Inputs
x * 4.5
y * 0.08
z * 4.5

Material Snippet: Marble (Material:

Inputs
scale * 0.01
mytex marble

Subtract: None

Figure 4: Editing the values of an object.

Results

We found the Raymarching Toolkit for Unity created a very unique way to create striking scenes that aren't possible in a traditional polygon-based game engine workflow. Since the toolkit keeps your content front and center visually, it's easy to tweak values and find interesting and beautiful combinations quickly.

The Raymarching Toolkit for Unity proved to be a fruitful experiment in tool design and in fleshing out a new direction for creating SDF content. Leveraging Unity's extensibility meant that we could create a useful and playful tool for non-programmers, giving creators a new and interesting medium to experiment with. The authors look forward to creating more with the toolkit in the future.

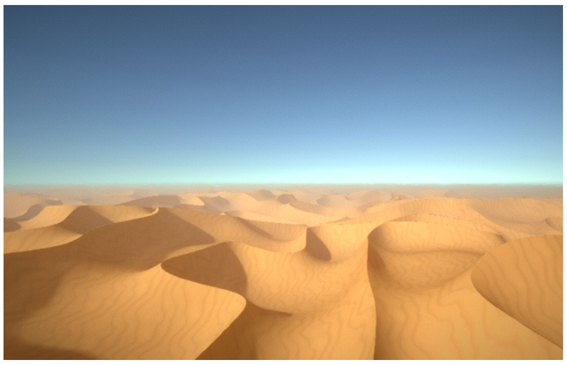


Figure 5: An infinite desert made by combining Perlin noise with an atan() function, and applying textures.

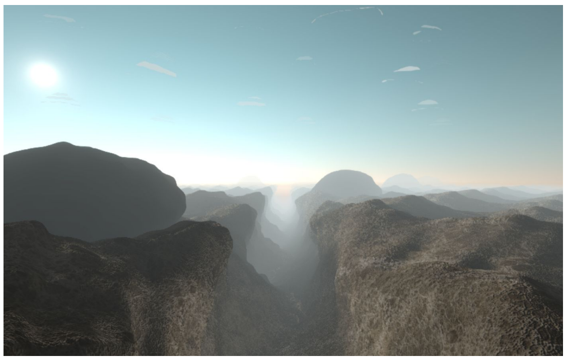


Figure 6: An infinite canyon constructed from layered noise, with a cylinder carving out the valley.

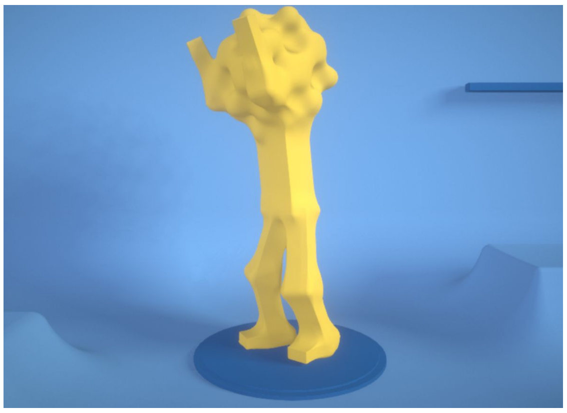


Figure 7: A character made from blending boxes stacked on an existing skeleton together.

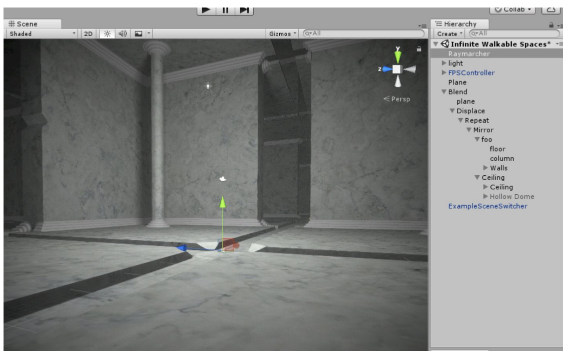


Figure 8: An infinite crypt you can walk around in, and the Unity hierarchy window showing the simple primitives it is constructed from.