Ruaumoko

Iso-surface creation suite for Unity
created by *Dr. Ben*

Contents

1. Introduction

This package provides the core functionality to generate amazing, professional iso-surface terrain. The results can be fully destructible and deformable on-the-fly, either generated in the editor or paged infinitely in your game!

The heart of this engine is a Marching Cubes implementation, which can generate polygonal meshes for an iso-surface from any 3D scalar field data (also called voxels). Details of the algorithm can be found at

http://en.wikipedia.org/wiki/Marching_cubes

Inspiration for the Ruaumoko implementation came from GPU Gems 3:

http://http.developer.nvidia.com/GPUGems3/gpugems3_ch01.html

2. FAQs

Q: Can I save/load the surfaces that I deform during run time?

A: Not yet, it's a feature that is in the works (or you can hack it yourself!)

Q: Why is the performance ___ (not absolutely wicked fast)?

A: The biggest performance hit is updating the MeshCollider, which is completely controlled by Unity and has nothing to do with this package. Updating MeshColliders requires Unity to stop the main thread, graphics, physics, etc., and reload the run-time pipeline with the new collider on board. As for the actual MarchingCubes algorithm if

you want to give the main thread every extra consideration use the Coroutine version of "March", but make sure it's done running before you call it again from another game object.

A secondary consideration is that C# is inherently slower at processing large arrays of data than complied c/c++ plugins or dlls will be. However, C# is highly portable (to any Unity platform) and editable, allowing you (and me!) to update things quickly.

Q: Why is MarchingCubes a singleton? Why can't I run *x* copies simultaneously?

A: in order to increase performance time MarchingCubes allocates all of its working array space in advance, allowing you to call MarchingCubes again and again without any instantiation hit. Running it in a multi-instance mode uses up a lot more memory, hits the garbage collector hard, and slows things down a lot. Because the primary performance hit is Unity's mesh collider updating a non-singleton implementation is a minor consideration.

Q: Is this multi-threaded?

A: No. A multi-threaded version was found to perform the same as the current release version. The primary performance hit is updating Unity's mesh colliders, which can only be called from the main thread. Running Marching Cubes in a second thread isn't difficult, but there is no measurable performance gain.

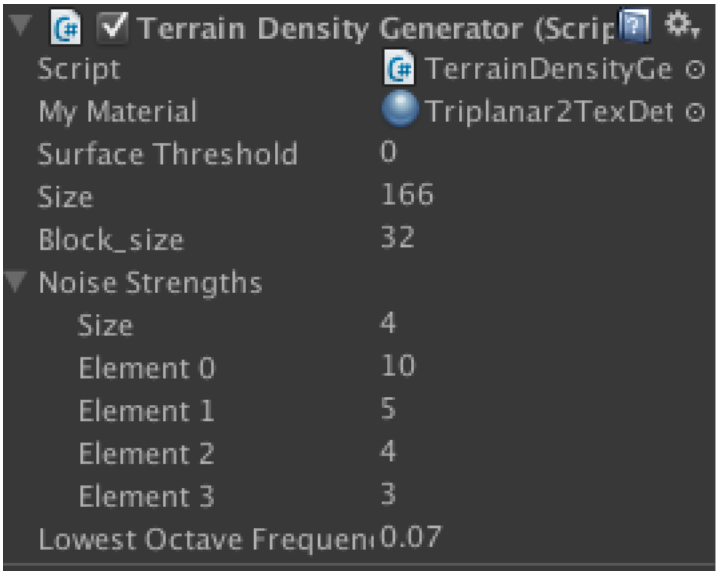
3. Demos

Included with the package are several demo scenes to display some of the capabilities of this engine. Some details to get you started are presented here:

1.Isosurface Demo

This demo generates a section of a world using octaves of classic noise, loads a 3D scalar field, and then "marches cubes" to extract the polygonal mesh for the player to walk on.

The Terrain Density Generator script contains all of this functionality. Typical settings are shown in the image below:



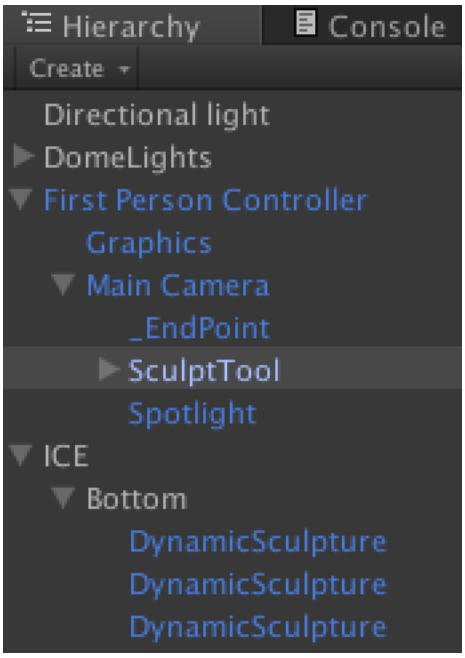
Terrain Density Generator Parameters:

- My Material: the material to apply to the terrain.
- Surface Threshold: where to extract the isosurface from the voxelized data
- Size: how many cells to generate (try larger–it takes longer to load)
- Block_size: how many voxels to lump into one mesh. Try other sizes!
- Noise Strengths: the octaves of noise for generating the terrain. Powers of 2 generate more conventional terrain (16, 8, 4, 2, etc) while other scales produce more twisted, unrealistic surfaces. Element 0 is the lowest octave.
- Lowest Octave Frequency: period of lowest noise octave. Smaller produces smoother surfaces, higher produces rougher surfaces.

2.IceSculpting

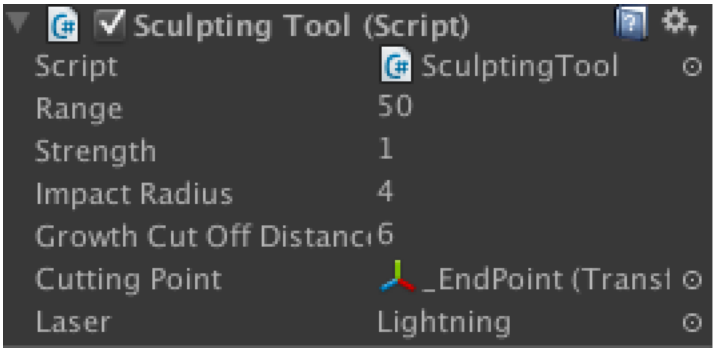
A destructible/constructible world demo. Blocks of voxel data are created and the player dropped inside. Using ray-casts against the blocks' mesh colliders the blocks are melted & regrown at the player's whim.

Primary functionality is on the SculptTool (attached to the First Person Controller) and the DynamicSculpture blocks.



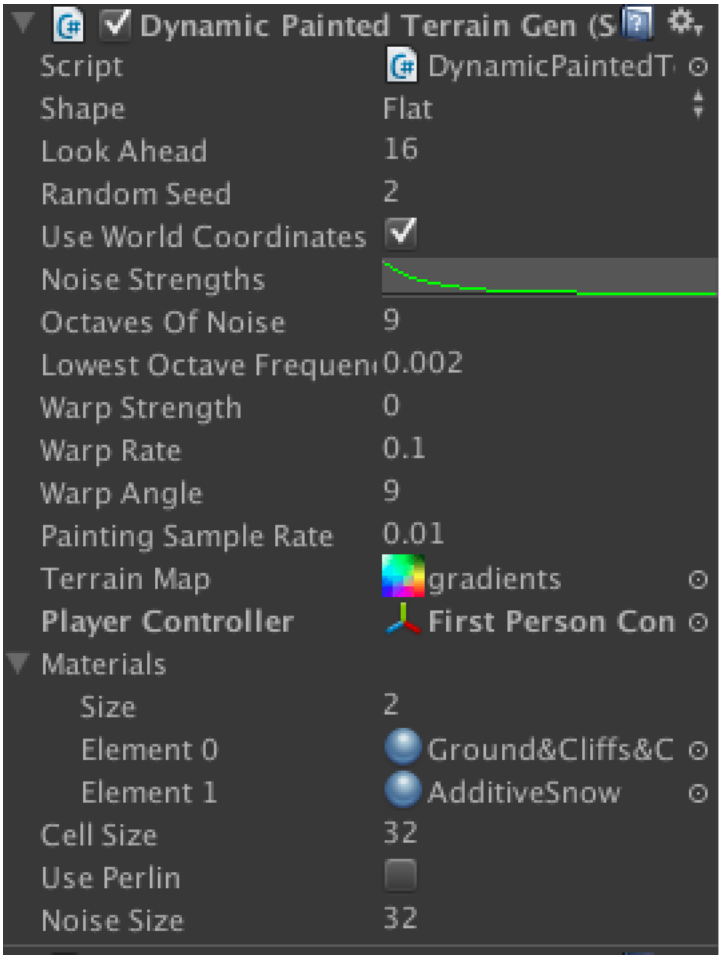
The Sculpting Tool exposes the following parameters:

- Range: how far the sculpt ray-casts reach
- Strength: how quickly the "dynamic sculptures" melt or grow
- Impact Radius: the "brush size" of the tool
- Growth Cut Off Distance: the near-distance at which point growth (freezing) will stop. This prevents the player from freezing the surface over themselves and falling through the world.



3.Paging Terrain

This demo uses a paging mechanism to dynamically generate blocks of terrain as the player explores the world. The "DynamicTerrain" object contains all the main functionality and the "Dynamic Painted Terrain Gen" script does all the real work. A screen shot of the typical settings are included below.



Parameters:

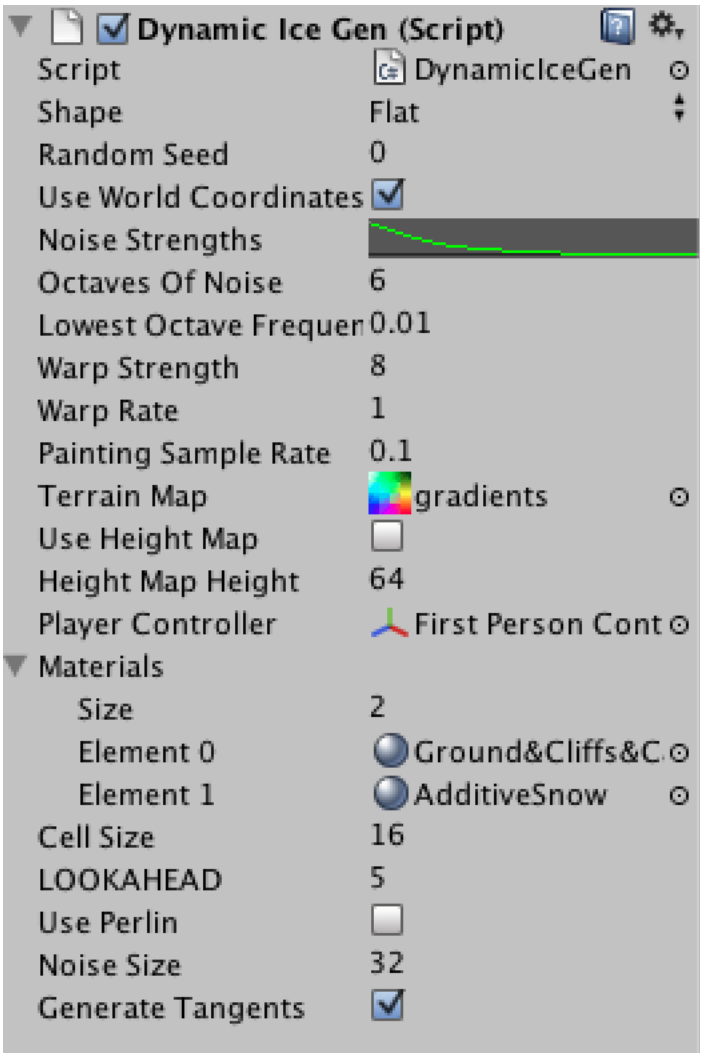
- Shape: selects the function that populates the density field for the surface generation. The names are chosen to roughly characterize the results.
    - Height Map: this mode uses the alpha channel of the Terrain Map as a height map, seeding the terrain just like a regular height-map terrain system. However, the noise (below) can be used to modulate this map, causing holes, cliffs, etc.
- Random Seem: set this to produce the same results every time (or set to 0 to get random results!)
- Use World Coordinates: uses the object's transform position so that the results are keyed to global position (you can move the object and the surface will generate the same way). If unchecked it will operate in local space.
- Noise Strengths: controls the random generation. A steeply declining curve produces more "realistic" surfaces, while different shapes will make more caves and floating islands.
- Octaves of Noise: fewer octaves produces simpler surfaces while more octaves produce more refined surfaces (and take more CPU time to calculate).
- Lowest Octave Frequency: how stretched vs compacted the variations in the surface are (hills & valleys). Setting this very low (0.01 and lower) produces

spread out hills and mountains.

- Warp Strength/Rate/Angle: setting these causes the octaves of noise to be twisted around in a spiral while being sampled. Increasing the strength and rate will cause weird, alien type surfaces.
- Painting Sample Rate: controls the frequency that the Terrain Map is sampled (see below).
- Terrain Map: this is an RGB Texture that controls different aspects of the noise over the generation of the surface. Think of this as a height map that controls three different aspects of the random generation across the land.
    - The Red channel causes the noise to be vertically squashed, producing pancake like plateaus in some places.
    - The Green channel affects the strengths of the highest 1/3 of the noise octaves. This can make some areas smooth and give others finer detail.
    - The Blue channel affects the warp strength, and thus only has impact if Warp Strength > 0.

4.World Sculpting

This demo uses elements from the previous two demos in combination: dynamic paging of new world cells and the real-time deformation. DynamicTerrain is the primary game object that drives this demo and the settings are very similar to the previous demo.



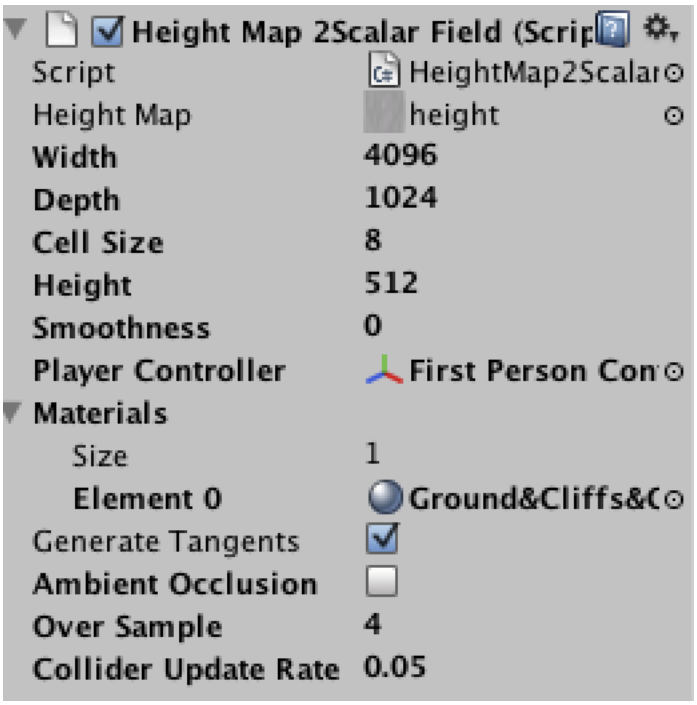
4b.World Sculpting-Height Map

This demo is nearly identical to Demo 4, but is now setup to use the height-map generation feature.

5.Height Map-Voxels

Another height map oriented demo. This one uses a feature of the marching cubes that allows the generation of more blocky surfaces (where the smoothness of the marching cubes is overridden to make a "mine-craft" like cube world).

HeightMapIsosurface is the main game object in this demo:



Try playing with the Height property (controls how high the mountains are) and the Width & Depth (how big the world is before it starts repeating). Smoothness controls the blockyness, with 0 being fully blocky and 1 being fully smooth.

Over Sample causes the height map sampling algorithm to look at neighboring pixels in the bitmap to smooth over the hillsides if the Width or Depth is larger than the bitmap. If it's set to 0 every bit change in the height map will produce blocky edges in the surface.

6.Planets Demo

A demo showing how to put spherical planets in your game that are still fully deformable.

6b.BigPlanet Demo

This demo creates one planet and allows you to change its size and recreate it, to test different planet sizes. Be careful, big planets (512x512x512 voxels and up) take a LONG time to create!

3. Package Contents

a. Core: Overview

The primary code entry point is through MarchingCubes.cs. This relies on other utility

scripts in Core/Plugins. Extracting the core functionality to another project requires copying everything in Core/Plugins.

MarchingCubes.cs

The marching cubes script runs on any scalar data (i.e. a 3D array of floats) that can be sampled for "density" values. Values below 0 are inside the mesh surface, values over 0 are outside, and the Marching Cubes finds all the 0 crossings to create a mesh iso-surface.
Preparing your data is matter of filling a jagged 3D array with numbers that can be used to describe your object.

How To Use MarchingCubes.cs in your code:

1) Access the singleton through MarchingCubes.Singleton

```
MarchingCubes myCubes = MarchingCubes.Singleton;
```

2) Pass in a 3D jagged array of density values (floats, use SetDensityMap()) representing the mesh you want to construct. Density values < 0 are considered outside the mesh, and density values > 0 are inside the mesh. (0 = surface threshold.)

The property "surfaceThreshold" determines the cross over point where the geometry will be constructed, and is typically left at 0.

```
myCubes.SetDensityMap(densityMap);
```

2b) (optionally) set the gutter size. This is how many extra values you have supplied around the edges of your array. This must be 2 or greater. If you intend to calculate Ambient Occlusion values (see below) then this is also the number of steps the algorithm will take and higher values will produce noticeably better results (gutter size 4 or greater is typically good).

```
myCubes.GutterSize = 2;
```

3) Call "March" (or MarchCoroutine for the asynchronous non-blocking version).

```
myCubes.March();
```

3b) (optionally) Call GenerateAmbientOcclusion if you will be using an ambient occlusion shader.

```
myCubes.GenerateAmbientOcculsion();
```

4) Call "UpdateMesh", passing in the mesh you want to hold the geometry, and

that's it! [Once March, etc. has been called you may call UpdateMesh() passing in many different meshes to create many copies of the same surface. However, because MarchingCubes is a singleton, setting the density map and marching again will clear out all previous mesh data.]

```
Mesh mesh = GetComponent<MeshFilter>().mesh;
myCubes.UpdateMesh(mesh);
```

Additional Utility Methods:

GenerateAmbientOcclusion - calculates ambient occlusion values based on the folding of the surface, storing the results in the alpha component of the vertex colors. This can then be used by Ambient Occlusion shaders. Typically this would be called immediately after "March()".

```
myCubes.GenerateAmbientOcculsion();
```

CenterAndScaleVerts - transforms the generated vertices to [-1,1] range, rather than 0-(density map length). This makes it easier to control the scale of the mesh using the transform, independent of the density map size. Note: call this before "UpdateMesh()" or it will not have any effect.

```
myCubes.CenterAndScaleVerts();
```

Other Plugin Scripts

MCTables.cs

This contains a large amount of lookup data that is necessary for the successful operation of the marching cubes algorithm. Do not change any of these numbers, unless you're absolutely certain you know what you are doing!

Noise3D.cs

A class that creates a static array of random samples, generated when first used. In combination with setting Random.seed it allows reproducible results of random fields. Noise3D is accessed through a singleton pattern, although many instances may be accessed through the index parameter of the Noise3D.singleton(x) function.

TangentSolver.cs

This is a utility script to efficiently calculate tangents for an existing mesh. This is necessary for real-time shadows, normal/bump maps, and parallax shaders. It is based on a script from the Unify wiki and can be used easily with any mesh, regardless of

origin.

TrilinearSample.cs

A utility script to interpolate samples within a jagged 3D array. Enables the smooth sampling of a arrays at any frequency.

4. Persistence: Saving/Loading

a. Overview

As of version 0.3b a fast serialization technique is employed to allow voxel data to be saved and loaded very efficiently during run-time. This is accomplished using .NET binary serialization and a byte packing scheme to greatly improve performance. To make a world savable:

The voxel cells will use VoxelCellData.cs to load and save data. Look at IceGen.cs for an example of how this file is used (in Go() and Save() functions).

Loading data is currently implemented as each cell is generated in the world. Thus each cell has to know what file-name to use, check to see if previous save data is available, and either load that data or generate a new cell if not.

Saveing data must be implemented in a "Save" function, using the VoxelCellData class to push data to the disc.

The simple GUI is in VoxelLoadSave.cs. It must be attached to the game object that has the primary world generation script on it. The world generation script needs to inherit from SavableTerrain and implement "Save()" (see DynamicIceGen.cs as an example). VoxelLoadSave then finds the world generation script and will call "SaveAllCells" when the user presses the Save button.

Notes:

Changing parameters that generate a world with save data in place will cause holes to open up in your world. If you change the parameters the best work around is to run the scene and hit the "Clear Files" button, then stop and run the scene again. Now all the cells will be generated afresh using your new parameters.

The demos use the scene name as the prefix for the saved data. If you want multiple saves, or the ability to pick which one you load, look in the cell files that specify the file

name (HeightMapCell.cs, IceGen.cs, PlanetCell.cs, etc.).

Binary serialization does not work on Web players. A workaround using Unity player prefs will be included soon.