VOXELFORM 2.0

Author: Mark Davis

Date: 12/19/2011

Thank you for purchasing Voxelform. I hope you have as much fun playing with this as I've had working on it. My contact info is available below, and I'd love to hear from you.

Please note that most of the documentation is in the form of XML code docs.

What's New in version 2.0:

This version is a complete overhaul of the Voxelform code base. The API is not backwards compatible with version 1.1.1, but should be similar enough to allow for fairly simple refactoring in existing projects. The code has been refactored so as to not conflict with the old API, but there should be no reason to continue using v1.1.1.

The biggest new feature is full multi-material support. Voxel terrain now has support for 64K unique materials… more than anyone would ever use, I hope!

The voxel data source has been abstracted and can be replaced with custom implementations.

Many new shaders have been added, including triplanar shaders with support for normal and specular mapping.

Additional voxel alteration methods have been added.

Menu commands for saving, loading, and creating terrain are available.

A beta version of a voxel terrain editor is included as well. It is the only component that requires Unity Pro, and it’s far from essential.

A new demo meant to double as a simple runtime editor.

Getting Started:

You can get started by loading up a demo scene, and running it, using its default settings. If you're a Unity Pro user, to get the visual quality seen in the screen shots or online demos, you may want to enable the Sun's directional light shadows and add other effects such as Bloom and Flares.

How to Play the Demos:

Free Editor Demo Controls:
Mouse – Look
WASD/QZ – Move
LMB – Dig
RMB – Build
Wheel – Change brush size
Shift – Slow down
E – Scroll material palette down
R – Scroll material palette up

Game Demo Controls:
Mouse – Look
WASD – Move
Space – Jump
LMB – Dig
RMB – Build
Wheel – Change brush size

In both demos, the Voxel Interaction script inspector properties can be manually altered during gameplay.

Mostly, it's just about flying around the world, sculpting terrain on the fly, and admiring your creations. Physics and lighting update as soon a change occurs.

When the demo starts, if single frame loading is disabled, you may notice the world rapidly loading in, away from your starting position. The world loads in while you play and should maintain a smooth and reasonable frame rate. When it finishes loading, you will likely see a huge increase in your frame rate. If single frame loading is enabled, then the world will load on the first frame.

Voxel Terrain's Inspector interface: (Very Important to read through this)

Quite a bit of tweaking can be done via the Voxel Terrain object's Inspector interface. More extensive documentation can be found in the code, but here's a quick breakdown of the various options:

Chunks:

The terrain is made up of voxel chunks. A chunk may contain anywhere from 1x1x1 voxels to 16x16x16 voxels. You can go higher, but performance drops are quite noticeable. Chunks do not need to be cubes. Any shape is acceptable. 8x8x8 seems optimal for general purpose use.

The smaller the chunk size, the more chunks Unity has to render. A larger chunk size will have a larger overhead per change.

Chunk counts determine the number of chunks in the terrain for a given axis.
Voxel counts determine the number of voxels per axis in a chunk.

Scale:
The scale uniformly scales the terrain and each chunk. Use this instead of the transform.

Isolevel:
This is the normalized minimum voxel volume required to create a surface. In other words, lowering this value will create denser terrain. Increasing it, will create more open spaces.

Observer:
This is usually the main camera. But you can set it to anything you'd like.

Observer Viewing Distance:
This is the minimum distance at which to disable chunks. This is here both for ease of use and in case the acceptable minimum viewing distance isn't the same as that of the camera.

Enable Queuing:
This can be left alone and ignored unless you have a good reason to change it. See the code for more details.

Enable Single Frame Loading:
This will cause the voxel terrain’s chunks to load in all at once instead of loading in slowly in the background. If enabled, a one-time delay may occur. If disabled, there should be no noticeable delay, but frame rate may suffer while the loading progresses. Additionally, physical objects could fall through if the chunk they reside on is not active. Custom solutions are in order here.

Material Palette:
The terrain must be hooked up to a valid material palette. This contains an indexed collection of the materials the terrain will use. Materials are standard Unity materials, and Voxelform ships with numerous prebuilt materials, and 15 shaders built with voxel terrain in mind.

Materials and Shaders:

Voxelform ships with many prebuilt materials, and 15 volumetric shaders. Different shaders exist to not just provide different looks, but also to provide performance options. Performance in this case means the number of textures used by the shader.

Shader Types:

Triplanar (1,2,3):
Triplanar shaders allow textures to be applied to a mesh based on x,y,z coordinates as opposed to the usual u,v texture coordinates. Which texture is applied to which side is determined by normals. If a shader uses multiple textures, then textures will smoothly blend from one to the other where possible.

Triplanar shaders and materials are labeled with 1, 2, or 3. This refers to the number of diffuse textures required. If normal and specular maps are supported, then the cost will double or triple respectively. To clarify, that’s the cost of the texture sampling, not the overall cost of the shader.

Solid Textured:
These shaders remix a 2D texture into 3D space in such a way that it takes on not just the look of natural stone, but also the volumetric properties. When stone is carved away, it’s continuous. The palette based noise shaders also provide this property.

Diffuse, Normal, Specular (DNS):
Various shaders and materials are labeled with D, DN, or DNS. This specifies whether the shader simply takes diffuse textures, or also takes normal and/or specular maps. Normal and specular maps do not need any extra processing by Unity. In fact, they shouldn’t be processed any differently than a regular texture. The Voxelform shaders handle them just fine as is, and things work a little differently for triplanar texturing anyway.

Pal:
Some shaders and materials are labeled with Pal. This means that the shaders use a palette texture. The palette texture can be anything, and is sampled with volumetric noise coordinates to produce color. Like the Solid Textured shaders, these shaders allow for a continuous look while carving stone away. They also blend in triplanar texturing for additional detail, and that is not continuous.

The Code:

The code is documented on two levels. “Pop-up” tips in MonoDevelop will provide basic information, but remarks contain extra information that is worth looking at as well, so it's a good idea to expand the docs and read through the code… a really good idea.

It’s not likely that this project will do everything you want, but the code was intended to be customized; so feel free to dig in and make it do whatever you'd like.

A Deeper Look:

This is a bit of a tutorial to get you started customizing Voxelform for your game. Some work is involved. If you’d prefer to just get something up and running, and have some fun tweaking things out a bit, then load up a demo scene, and have it. It’s a perfectly valid way to get started. Then later perhaps, take a look back here and have a go at things from scratch.

Perhaps the first bit of code to look at is: VFVoxelTerrainExampleGenerateVoxels.cs

Notice that we’re subclassing VFVoxelTerrainBaseExample and above that, VFVoxelTerrain. VFVoxelTerrain is now an abstract class and must be subclassed. This doesn’t just make it easier to reuse this code, but is also required for allowing custom voxel data sources (IVFVoxelDataSource) to be hooked into VFVoxelTerrain by overriding the VFVoxelTerrain.CreateDataSource method.

VFVoxelTerrainExampleGenerateVoxels or VFVoxelTerrainExampleLoadFromFile can be used as examples for creating your custom voxel terrain script. Read the comments in those classes for more information.

Once you have a voxel terrain script, you’ll need a voxel terrain to run it on. The easiest way to create voxel terrain is via the new “Create Voxel Terrain” wizard in the Voxelform menu.

The only required field in the wizard is the Material Palette. If you don’t have one, then it’s time to run the “Create Material Palette” wizard. You’ll need to add at least one material to the material palette as well. Choose any material from the Materials folder, and if you have trouble deciding, try this one: “Triplanar1 - DN - Cracked Stone 1”

Now from the “Create Voxel Terrain” wizard, select the material palette and create an otherwise default terrain. If you do decide to alter the defaults a bit, then please note that creating a terrain this way allows you to specify exactly how many voxels you want as opposed to how many voxels per chunk you want, as would be done via the VFVoxelTerrain object itself.

Also note the System Type Name property in the wizard. Just replace VFVoxelTerrainExampleGenerateVoxels with the name of your custom script if you're using one.

So now you should have a new voxel terrain object sitting in your scene. Supposing that your scene consists of a camera, light, material palette, and voxel terrain, you'll be able to see the terrain, drop objects or a character onto it if you add them, but not modify it yet at runtime.

To modify the voxel terrain at runtime, you'll need to write some code. Take a look at the AlterVoxel method in VFVoxelTerrainFabrication.cs. This will let you easily alter voxels at x,y,z in voxel space. You also have AlterVoxelBox and AlterVoxelSphere at your disposal as well.

Of course something's missing, and that's the conversion of world space to voxel space. You'll almost always want to correlate an alteration to a specific world space coordinate. Take a look at the VFVoxelInteraction.EditVoxels method for an example with more documentation on that.

You may notice that VFVoxelInteraction.EditVoxels also uses ray casting. If that's part of your plan, then it should be a decent example of how to get from casting a ray to altering a voxel.

Cyclops Framework:
This is a framework that I've included as a support library, but in source form. It would take some time to explain the full functionality of the framework, but in short, it's being used here as an alternative to co-routines while code is running in the editor. As of this writing, co-routines don't work in the editor, and that would have been a problem. The framework provides pseudo-threading concurrency (like co-routines), sequencing, tweening, control flow, messaging, cascading tags, and loads more. It's something that I've used across multiple languages, and numerous commercial games for over 4 years. At some point, I'll document the C# version, add some extra goodies, and perhaps open source it as I've done with the AS3 version. More info at: cyclopsframework.org

Menu Quirks:
Note that when a voxel terrain is created, it has the name: "Voxel Terrain". For now, the Voxelform menu items for saving and loading expect to find an object named "Voxel Terrain". I'm expecting the UI programming model to change in the near future, hopefully with Unity 3.5. When that happens, I'll fix this along with Pro Editor code which is very rough and could stand a rewrite.

The Marching Cubes Algorithm:

Marching cubes is an algorithm commonly used for MRI visualization, and also for metaballs rendering in popular 3D modeling packages. I highly recommend doing some additional research into how this algorithm works. It'll shed some light on that end of the code.

Good starting points include:

Paul Bourke's marching cubes example. It's geared for metaballs rendering and doesn't contain information on calculating normals, but some of the source is based on this. Most documentation and examples for marching cubes point back to this link:
http://local.wasp.uwa.edu.au/~pbourke/geometry/polygonise/

And of course, it probably goes without saying:
http://en.wikipedia.org/wiki/Marching_cubes

I also highly recommend reading the first chapter in GPU Gems 3. I based some of the new shader code on an example in that chapter.

Uses for Voxelform:
Mining games, cave exploration, and first person shooters with terrain tools and weapons immediately come to mind. Or perhaps the ultimate butter slicing sim! The sky's the limit, and I'm really looking forward to seeing where this goes.

Road Map:
I'll continue to update the v2.x branch for the foreseeable future, and this time in smaller increments. Some of the things slated for v2.0 didn't make it or were removed. I'd like to continue investigating larger terrains, mobile development, and modifications to the Marching Cubes algorithm for greater flexibility.

The Pro-Editor is currently in beta and may be completely rewritten.

Missing Documentation:
What's not in this document may be in the code docs, but if you find that something important has been overlooked, please feel free to give me a shout and I'll do my best to include it in either the code docs or this document.

I do plan to add more documentation, examples, and perhaps a video tutorial.

Website and Demos:
http://voxelform.com

News and Discussion via the Unity Forums:
http://forum.unity3d.com/threads/94224-Introducing-Voxelform-voxel-terrain-(Marching-Cubes)-withreal-time-deformation.?highlight=voxelform

Facebook:
https://www.facebook.com/pages/Voxelform/233379350024952

Contact Info:
Please feel free to drop me a line if you have any questions or concerns.
I can be reached at:

Mark Davis
voxelform@gmail.com

Again – Thank you so much for your support of this product.
If you have the time to rate it, I'd be eternally grateful.