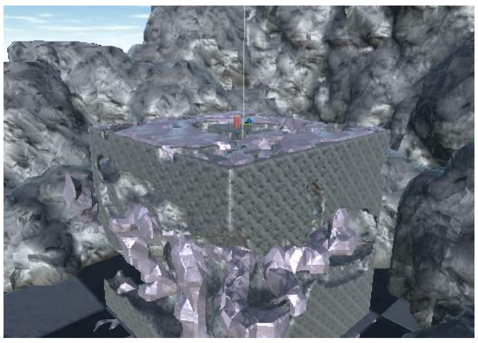
VOXEL GENERATOR

INTRODUCTION:



Voxels are volumetric structures which are used to simulate real world atoms. Each voxel simulates one atom which can be modified. Voxel engines are often used in computer tomography and games such as Minecraft and Space Engineers. Often these games completely optimized towards the voxel engine which limits the usability by an great amount,

The aim of this asset is to provide a way to use voxels as an additional feature in your game while maintaining a normal workflow. Additionally, this asset also offers full flexibility over the data structure itself as nothing is hidden behind DLL walls or C++ bridges as it is often found in other assets since burst compilation makes this obsolete.

COMPATIBILITY INFORMATION

The voxel generator is fully compatible on every build target as it works fully independent of any unity function with the exception of burst and job system. The burst and job system is not mandatory but increases performance by a huge amount. The generator works on build targets without burst support but without the performance boost. Here is a list of tested build platforms:

- **Windows/Mac/Linux:** Normal and supports burst.
- **Android/Ios:** Normal and supports burst. Beware the lower memory on mobile devices!
- **WebGL:** Works but no burst support (may change by Unity in the future)
- **Consoles:** I don't own dev kits so I cannot guarantee functionality. Expected to work but burst may not be supported.

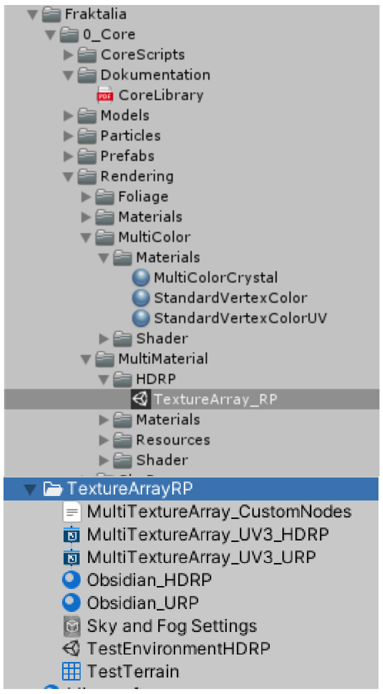
Technology Compatibility:

- **AR:** The voxel block can be placed as AR augmentation into a mobile app and can be modified after that. Keep in mind the reduced amount of memory on mobile devices.
- **VR:** Full VR support as the engine was implemented for a VR sculpting application.

Limitations:

- **32 Bit systems**: Memory limit to 4096mb RAM could be exceeded easily which results to app crash. Burst version 1.5.0 – 1.5.4 crashes the app on 32 bit builds for unknown reasons. ! (Engine warns you!)

RENDER PIPELINE INFORMATION



- Fraktalia
  - 0_Core
    - CoreScripts
    - Dokumentation
      - CoreLibrary
    - Models
    - Particles
    - Prefabs
    - Rendering
      - Foliage
      - Materials
      - MultiColor
        - Materials
          - MultiColorCrystal
          - StandardVertexColor
          - StandardVertexColorUV
        - Shader
      - MultiMaterial
        - HDRP
          - TextureArray_RP
        - Materials
        - Resources
        - Shader
- TextureArrayRP
  - MultiTextureArray_CustomNodes
  - MultiTextureArray_UV3_HDRP
  - MultiTextureArray_UV3_URP
  - Obsidian_HDRP
  - Obsidian_URP
  - Sky and Fog Settings
  - TestEnvironmentHDRP
  - TestTerrain

The Voxel Generator works independent of the render pipeline used. The important change is related to the material which has to be used when multi material setups should be used.

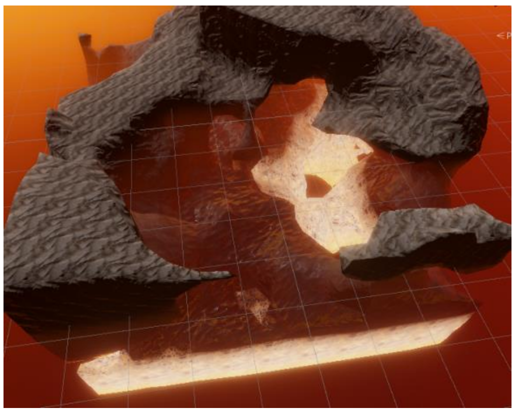
Almost all samples use the built in standard shader. The only exception are the materials which use texture arrays in order to smoothly blend between materials.

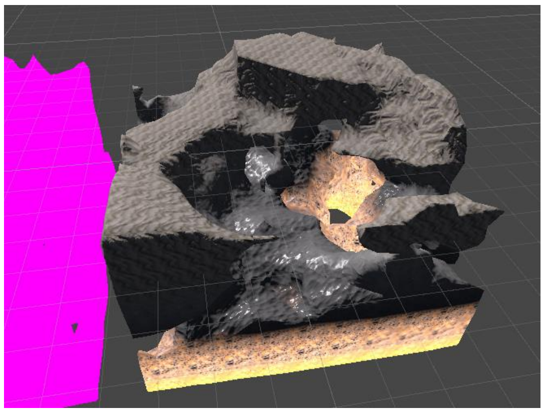
The texture array related shaders are used across multiple assets and are inside the 0_Core folder. This folder now has a HDRP section, containing a “TextureArray_RP.unitypackage” file which contains shader for the dedicated functionality.

You have to either have HDRP or URP applied correctly in order to use those shaders. Double click to the TextureArray_RP file. After unpacking, the new content contains the .shadergraph files for HDRP and URP. Additionally a test scene is also included

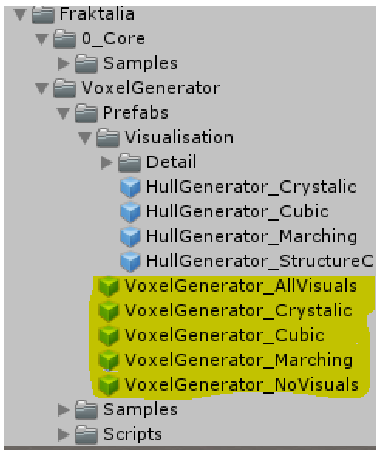
The sample scene contains one object representing the HDRP version and one showcasing the URP version. Since you cannot have both render pipelines active, the unused version shows the error like in the image below. The left one is the HDRP version and the right one shows the URP version.

The shader graphs itself are almost identical but minor things are still different which is the reason why separate versions are required.





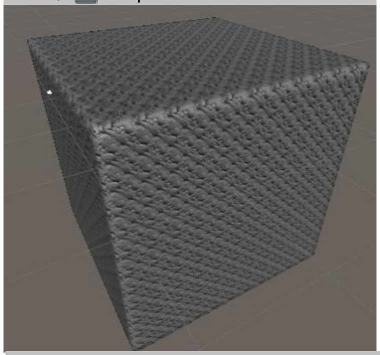
HOW TO USE:



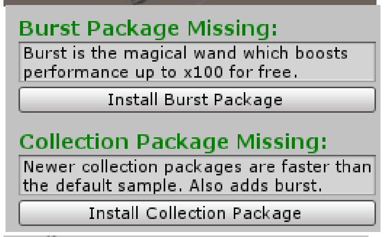
```
▼ Fraktalia
  ▼ 0_Core
    ▶ Samples
  ▼ VoxelGenerator
    ▼ Prefabs
      ▼ Visualisation
        ▶ Detail
          HullGenerator_Crystalic
          HullGenerator_Cubic
          HullGenerator_Marching
          HullGenerator_StructureC
        VoxelGenerator_AllVisuals
        VoxelGenerator_Crystalic
        VoxelGenerator_Cubic
        VoxelGenerator_Marching
        VoxelGenerator_NoVisuals
    ▶ Samples
    ▶ Scripts
```

For fast testing, just drag in one of the provided VoxelGenerator prefabs in your scene. All VoxelGenerator prefabs with the exception of the “NoVisuals” version have visualizations attached so the initial block becomes visible immediately.

The prefab contains the VoxelGenerator, VoxelSaveSystem and a VoxelPainter component which allows you to modify the block by holding CTRL + Left/Right click. The main settings for the VoxelGenerator is the Root Size which defines the volumetric size, the subdivision power which modifies the data structure and the initial value which is usually at 255 for completely solid blocks and 0 for empty blocks.



The VoxelSaveSystem provides options for saving the volumetric dataset and automatically saves the voxel data when saving the scene itself. The standard setting “Scene” is the most convenient method as it simply stores the data into the “.unity” scene file and is fine for voxel date with smaller resolution. However the Unity serialization slows down the Unity Inspector when the size of the voxel data becomes too large. For such cases, it is better to save it as ScriptableObject in order to keep the Unity scene file small.



Burst Package Missing:

Burst is the magical wand which boosts performance up to x100 for free.

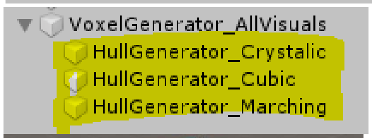
Install Burst Package

Collection Package Missing:

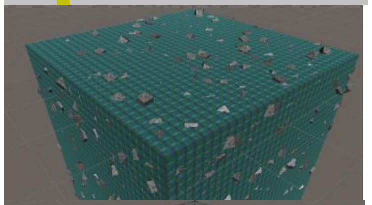
Newer collection packages are faster than the default sample. Also adds burst.

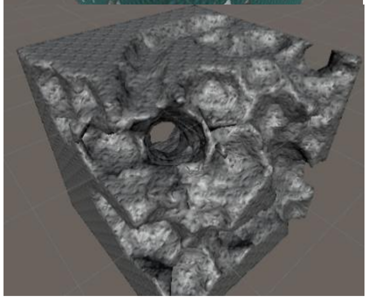
Install Collection Package

When importing this asset the first time in a fresh Unity project, the **Burst and Collection** package probably is not installed. Fortunately it is now easy to install the latest version by clicking on both buttons which are visible at the VoxelGenerator component. Burst will boost the performance and Collections updates the internal native data structure.



```
▼ VoxelGenerator_AllVisuals
    HullGenerator_Crystalic
    HullGenerator_Cubic
    HullGenerator_Marching
```





The visualization itself is separated from the main VoxelGenerator GameObject and is a separate hull generator attached as child to the VoxelGenerator. If you drag the VoxelGenerator “AllVisuals” into the scene, a mixure of blocks combined with an inconsistent pattern of randomly oriented cubes will appear. Hidden behind the surface is a smooth layer of marching cubes because this VoxelGenerator has 3 hull generators attached as children. The first hull generator Crystallic generates the randomly orented layer, Cubic the blocks and at last the Marching cubes layer. Visualisations can be mixed together in any combination as it is explained in the detail sample scene.

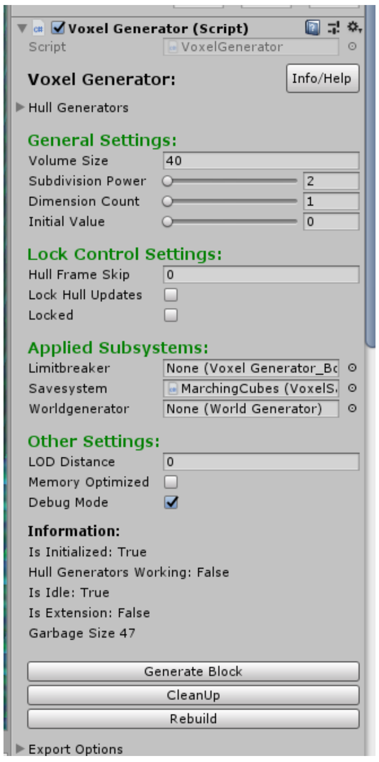
When the VoxelGenerator inside the editor is selected, you can modify the block by holding CTRL + Left/Right click. Saving the scene will automatically save modified VoxelGenerators. On scene initialization, the voxel data is regenerated by the VoxelSaveSystem or if disabled, can be loaded dynamically later. The initial mode is set to load and finish everything on scene initialization which is important if physical objects should collide with the hull else they may fall through the unfinished hull.

MAIN COMPONENTS:

The main components are the VoxelGenerator, VoxelSaveSystem and the Voxel Modifier which is used to modify the block during edit mode. Each VoxelGenerator could exist without the VoxelSaveSystem or VoxelModifier if saving and editing is handled by other scripts for example procedural systems similar to those seen in Minecraft.

VOXEL GENERATOR

The VoxelGenerator is the main component of the whole voxel engine and provides the hierarchical data structure for the voxel data.


Voxel Generator (Script)

Script	VoxelGenerator

Voxel Generator:

General Settings:

Volume Size	40
Subdivision Power	2
Dimension Count	1
Initial Value	0

Lock Control Settings:

Hull Frame Skip	0
Lock Hull Updates	☐
Locked	☐

Applied Subsystems:

Limitbreaker	None (Voxel Generator_Bc
Savesystem	MarchingCubes (VoxelS.
Worldgenerator	None (World Generator)

Other Settings:

LOD Distance	0
Memory Optimized	☐
Debug Mode	✓

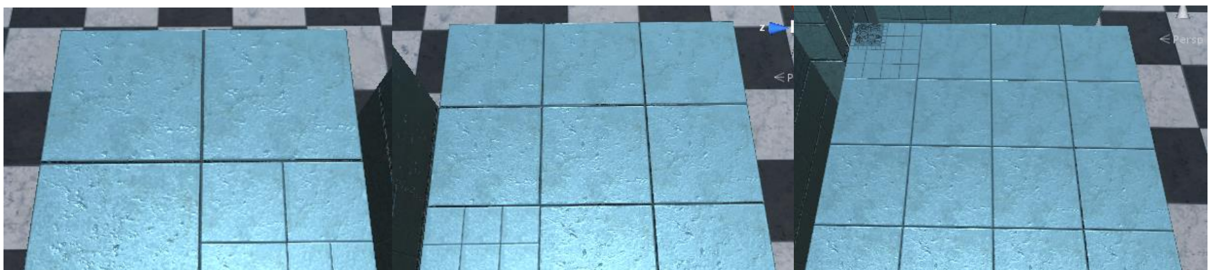
Information:
Is Initialized: True
Hull Generators Working: False
Is Idle: True
Is Extension: False
Garbage Size 47

Volume size: Defines the physical dimension of the voxel block, the amount of space it will occupy. The blue frame in editor view gives an indication about the size. Negative Values will not be accepted. Values between 1 and 100 are ideal performance wise. Values too low or too high could lead to floating point precision errors.

Initial Value: The initial value of a freshly generated voxel block. Value of 255 represents fully solid and 0 means usually empty. However the real behavior is dependent on the attached hull generators.

Subdivision Power: Defines the layout of the underlying data structure. The blocks in the images below use the DataVisualisation component to visualize the data structure. These samples can be observed in the VoxelTrees scene and show how the data structure is subdivided to generate the desired appearance.

The most basic subdivision power of 2 represents the traditional Octree which is used in many other voxel engines. It has the lowest memory footprint but is costly when traversing down into finer resolutions. Modifying tools can use different resolutions for coarse and fine works. Higher subdivision power increases the memory demand but reduces the traversal time down to target resolutions. Experiments have shown that Subdivision Power of 3 has the best Memory/Performance ratio.



Dimension Count: The Voxel Generator has a fourth dimension. If this value is greater than one, the Voxel Generator will manage more than one Voxel Data simultaneous. The first dimension is always used to describe which regions are solid. Additional dimensions are hidden unless the hull generator uses the additional dimension for some shenanigans such as Multi Texture support where the second dimension describes the material type.

LOCK CONTROL SETTINGS:

Hull Frameskip: Delays update of hull generators if greater than zero. Will decrement every frame and is set by third party scripts to control hull update behavior.

Lock Hull Updates: If true, changes to the dataset will not update hull generators. The save system influences this value when loading voxel data. Voxel modification is still handled. World generators also lock the generator during generation.

Locked: If true, voxels are not modified and hulls are not updated.

SUB SYSTEMS

Save System: Confirmation that a save system is connected to the Voxel Generator.

World Generator: Confirmation that a world generator is connected to the Voxel Generator.

LimitBreaker: Confirmation that a VoxelGenerator_BoundaryBreaker script exists as parent. When this is the case, the experimental infinite world system is activated (see extra dedicated chapter) and the Voxel Generator will act as template for the infinite world generation system. The image below shows how the result will look like when this feature is applied.

Will be replaced by a new system which will be distributed on discord.

OTHER SETTINGS

LOD Distance: Parameter to define LOD and can be used by Hull Generators to implement LOD.

ChunkHash: Chunk information which is important for World Generators.

Memory Optimized: Forces the generator to delete lists when not needed. Highly improves memory demand but slightly lowers performance as the allocation of memory is not free. The optimization is really huge so it is recommended to have this set.

Debug Mode: Used for development and visualizes information in order to verify correctness.

INFORMATION

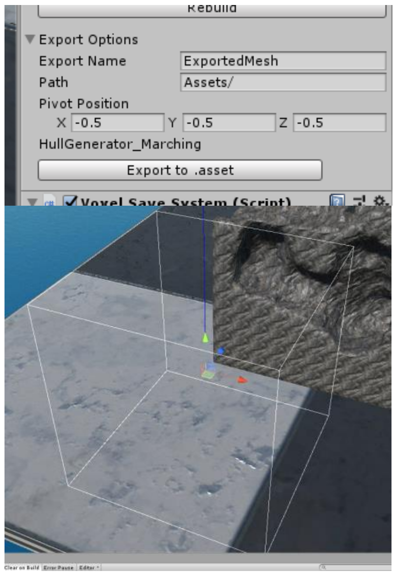
- **Is Initialized:** If true, generator is initialized.
- **Hull generators working:** Shows if hull generator are updating the visualisation.
- **Is Idle:** Generator is idle if no hull generator is working and also no voxel modifications have to be processed.
- **Is Extension:** True if it is an extension. Is used with infinity world systems or other third party scripts.

BUTTONS

Generate Block:	Generates a fresh voxel block using the initial value as starting parameter
CleanUp:	Destroys the existing voxel block.
Rebuild:	Regenerate the hull generators.

EXPORT OPTIONS

The Voxel Generator also provides an option to export mesh generated by hull generators. When opening the export options, a white box appears at the center of the local position. You can define the name, path and the pivot point. By default the pivot point is (-0.5,-0.5,-0.5) which is in the center of the full voxel block. In the right image, (0,0,0) is the bottom corner and (1,1,1) is the upper right corner.


Rebuild
Export Options
Export Name: ExportedMesh
Path: Assets/
Pivot Position: X -0.5 Y -0.5 Z -0.5
HullGenerator_Marching
Export to .asset
Voxel Save System (Script)

The white box is a helping indicator to see where the content will be relative to the pivot point of the game object. The right image shows that the pivot of the export product matches the center of the game object.

The exported file is a .asset file as exporting as .OBJ or .FBX is only possible with dedicated third party tools due to legal reasons.

TECHNICAL PART:

GenerateBlock(); Generates a fresh voxel block using the initialValue as starting parameter.

ResetData(): Resets all voxel maps on this generator. The initial id of the root will become 0.

SetVoxels()/SetvoxelsAdditive(): Provides direct functionality to modify voxels either directly or in an additive fashion. Basic parameter usually is the local position, target depth and the ID. These functions come in a variety which allows lists and NativeVoxelModificationData which usually is used by the save system or procedural block generators (future content).

Rebuild(): Regenerates the hull generators.

Cleanup(bool includeStatics): Destroys the voxel block. Also cleans static native containers if includeStatics is true and is only required inside the Unity editor to prevent memory leaks.

SetRegionsDirty: Marks region as dirty so the visuals have to be updated.

SetNeighbor(int ID, VoxelGenerator generator): Connect the voxel generator with another voxel generator. ID is the position where 0 is bottom left front and 26 is top right back.

RemoveNeighbor(int ID): Removes the connected voxel generator. Alternate RemoveAllNeighbor function removes all neighbors connected.

GetVoxelSize(int depth): Get the voxel size at target depth. Depth 0 is the volume size. The higher the depth, the smaller the voxels become.

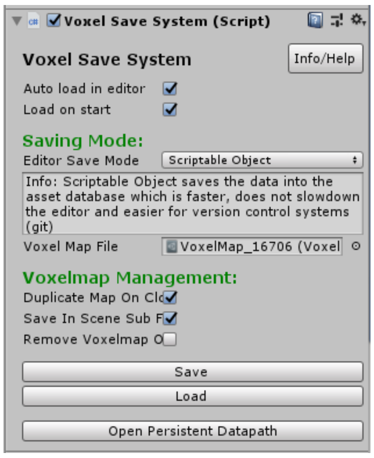
GetBlockCount(int depth): Calculates the amount of voxels the generator would have if every voxel would have the same depth.

GetBlockWidth(int depth): Amount of voxels the generator would have in one direction if every voxel has the given depth. Result * Result * Result would be the Block Count.

Show/Hide Visuals(): Shows or hide the visuals (meshes generated by attached hull generators)

ClearMeshes(): Clears the content generated by hull generators.

VOXEL SAVE SYSTEM:


Voxel Save System (Script)

Voxel Save System — Info/Help

Auto load in editor ✓
Load on start ✓

Saving Mode:
Editor Save Mode: Scriptable Object

Info: Scriptable Object saves the data into the asset database which is faster, does not slowdown the editor and easier for version control systems (git)

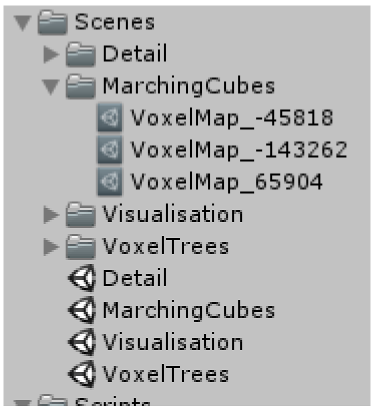
Voxel Map File: VoxelMap_16706 (Voxel

Voxelmap Management:
Duplicate Map On Clo ✓
Save In Scene Sub F ✓
Remove Voxelmap O ☐

Save
Load
Open Persistent Datapath

The VoxelSaveSystem is the main component for persistence and should always be together with the VoxelGenerator. The only exception occurs if the content is completely procedural and persistence is never intended. This component is also required since modifications inside the editor should be saved. If the VoxelGenerator was modified in during edit mode, it will be marked dirty and saves the voxel data automatically when saving the scene. (pressing ctrl+s)

There are 3 different saving options where the first one is “Scene” which saves the voxel data directly into the scene. This saving method is fast and convenient since there are no extra files to manage. However this mode will slow down the inspector when high resolution voxel data is used.

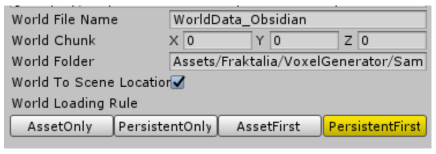


The second mode is “Scriptable Object” which stores the data as scriptable object in the asset database. This option causes less overhead for Unity and keeps the scene file small. Also causes no serialization overhead which is useful for high resolution voxel maps. Also this method is better for version control systems such as git since there are more small files than one big scene file. Also merge conflicts are easier to avoid since voxel maps can be merged while Unity scene files still have merge issues in git. Also data from the “Scene” option is removed when saving with this mode since having data in the scene file is obsolete when it already exists as scriptable object. Additionally, this saving option comes with a management system and has following settings:

- **Duplicate Map on Clone:** If true, duplicates the voxel map if you duplicate the VoxelGenerator inside the editor. It clones the original voxel map and automatically assigns it to the cloned object. The naming is “VoxelMap” + the instance id of the cloned GameObject. The location of the cloned map is in the same folder as the original map.
- **Save in scene sub folder:** If true, a new voxel map is generated and placed into a subfolder where the Unity scene is located during saving if no map is assigned. The subfolder is created automatically if it doesn’t exist yet. If false, newly generated voxel maps are placed into the assets folder.
- **Remove voxel map on delete:** If true, voxel map is automatically removed from the asset database. Note: Removing is not undoable and is permanent!

The third mode is “Persistence Datapath” and just requires a name as identifier (without “.VOXEL” ending). It directly stores the voxel map as .VOXEL file into the persistent data path of your target device. This mode is supposed for persistence during gameplay where the player can save voxel maps. Savings are not handled inside version control systems since these files are probably not inside the repository and will only exist locally unless shared by external services. This saving mode is the fastest for saving and loading and is optimized for real time saving/loading.

Saving as Persistent Datapath has 2 sub variations which is **"World" and "Region".** The difference is just the naming and management in order to be able to match it to chunk positions.


World File Name: WorldData_Obsidian
World Chunk: X 0 Y 0 Z 0
World Folder: Assets/Fraktalia/VoxelGenerator/Sam
World To Scene Locatior ✓
World Loading Rule: AssetOnly | PersistentOnly | AssetFirst | [✓ PersistentFirst]

"World" and is similar to "Persistence Datapath". However this saving method is supposed to be used with the infinite world system. The main parameter is the World File Name and describes the name of the infinite world. Also the voxel map file is stored as .voxelworld.

"Region" is similar to "World" and can only be used in combination with the infinite world system. The settings are identically to the "World" mode but with the addition of a World Region Size parameter. The file type of this mode is .voxelregion and each file contains the voxel data of World Region Size * World Region Size * World Region Size voxel blocks. If .voxelregion files exist at the target path, the World Region Size parameter cannot be modified.

For example if the World Region Size is 8, each file will cover 512 chunks. Therefore an infinite world has much less files which reduce hard drive read/write operations by a significant amount. This reduces performance impact at the cost of higher working memory demand since a large amount of voxel data will be stored inside the RAM.

The World Chunk is set automatically and describes which chunk is stored.

The World Folder is used when storing world information in the asset database and is assigned automatically to the scene asset location when "World To Scene Location" is flagged.

World Loading Rule describes which data storage has priority. If the infinite world is modified during edit mode, it is stored in the asset database since the persistent datapath only allows local data. When voxels are modified during gameplay, the modified data is in the persistent datapath since it was created during gameplay. The rule describes what should happen when a chunk has data in the Asset Database and in persistent datapath at the same time.

If **Auto Load In Editor** is true, voxel map will be loaded automatically when entering edit mode or loading the scene.

If **Load On Start** is true, the voxel map will be loaded on scene initialization during play mode. Also the hull generators are forced to finish their work immediately (else RigidBodies could fall through half-finished voxel blocks).

TECHNICAL PART:

Load(): Loads the voxel data and rebuilds the VoxelGenerator

Save(): Saves the voxel data instantly. (causes performance spike)

SaveBinary(): Saves the voxel map as binary file. VoxelName member variable is used as identifier for example "VoxelMap1" without ".VOXEL" as ending.

RemoveBinary(): Removes the binary file at persistent data path. VoxelName is filename.

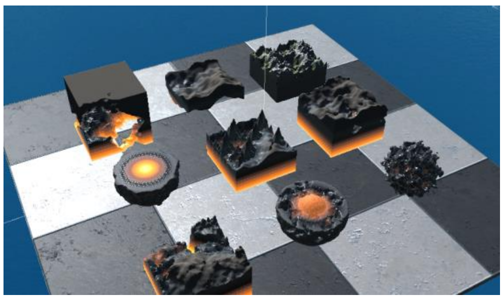
DynamicSave(): Real time saving option which does not cause performance spikes. Saves the voxel map as binary file into the persistent data path and should be called without interrupting gameplay. VoxelGenerators are Locked during saving process.

DynamicLoad(): Real time method to load any voxel map from any source without interrupting gameplay. VoxelGenerator can be modified during load but hull generators are suppressed until loading is finished.

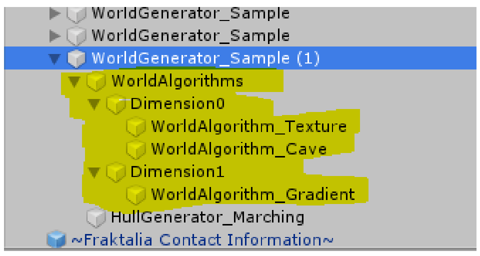
ConvertRaw(VoxelGenerator generator): Converts the voxel data of a initialized VoxelGenerator into a format which can be stored. Shrinks the data by gathering the leaves only. Returns RawVoxelData which can be saved or applied to an existing VoxelGenerator.

ApplyRawVoxelData: Applies RawVoxelData to a target VoxelGenerator

WORLD GENERATOR



The world generator was previously related to the deprecated infinite world system and became now the second sub system similar to the save system. The main difference between the World Generator and the Save System is that the Data from the World Generator is generated procedurally. The world generator will generate voxel data based on algorithms whenever the chunk which should be loaded has no saved data.



```
▶ WorldGenerator_Sample
▶ WorldGenerator_Sample
▼ WorldGenerator_Sample (1)
  ▼ WorldAlgorithms
    ▼ Dimension0
        WorldAlgorithm_Texture
        WorldAlgorithm_Cave
    ▼ Dimension1
        WorldAlgorithm_Gradient
    HullGenerator_Marching
~Fraktalia Contact Information~
```

How the world is generated is defined inside a World Algorithms hierarchy. Since it is possible to use multiple voxel dimensions, the generation is separated into **WorldAlgorithmCluster** sub objects (Dimension0, Dimension1)

The **WorldAlgorithmCluster** only defines the target depth and target dimension. **WorldAlgorithms** can be attached to the cluster in order to mix different algorithms together.

Biome support probably may be implemented as derivate of a WorldAlgorithmCluster as smooth transition between 2 biomes is also required.

The world generator is applied automatically to the Voxel Generator component attached on the game object. Clicking on **Generate World** will initialize the generator and apply the world data. It also uses the **ChunkHash** parameter provided by the Voxel Generator to define the position. This is important for multi-block systems.

It is also recommended to set scale invariant to true as it allows changing the volume size of the Voxel Generator without changing the result.

You can also apply the World Generator to any Voxel Generator using these functions:

Initialize (VoxelGenerator generator): Initializes the world generator and all algorithms attached to it. Should be called once or when algorithm parameters changed. The input generator is then used as reference.

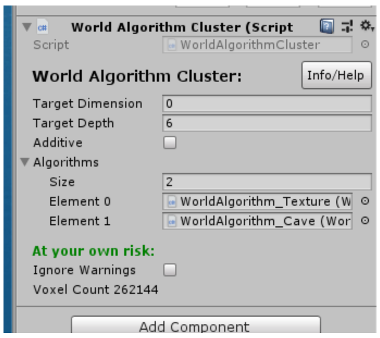
Generate (VoxelGenerator targetGenerator): Requests a world generation for the target generator when safety is valid. Requests are added to a queue and processed one by one.

CleanUp(): To clean everything up.

WORLD ALGORITHM CLUSTER & WORLD ALGORITHMS

The World Algorithm Cluster is the main container of World Algorithms and therefore can contain a mix of different World Algorithms. The Cluster itself only defines the Target Dimension and Depth. Modification parameters are then provided by the specific algorithms.

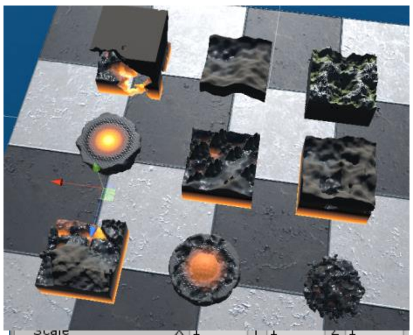
The sample in the right image has 2 algorithms attached as children. The first one creates a terrain based on a texture and the second one subtracts material to create caves.



```
World Algorithm Cluster (Script
Script              WorldAlgorithmCluster
World Algorithm Cluster:                Info/Help
Target Dimension    0
Target Depth        6
Additive            ☐
▼ Algorithms
    Size            2
    Element 0       WorldAlgorithm_Texture (W
    Element 1       WorldAlgorithm_Cave (Wor
At your own risk:
Ignore Warnings     ☐
Voxel Count 262144
              Add Component
```

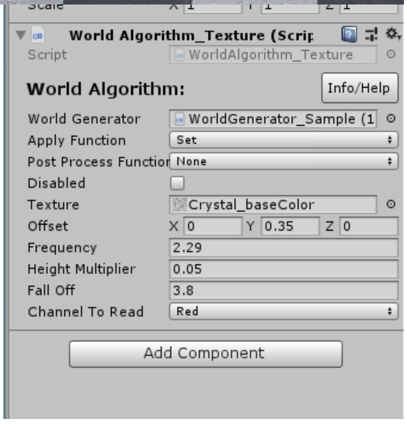
Another important feature is the auto updating system which helps designing the procedural. It has a very strict safety system as it also updates the result when changing the Dimension and Depth. Increasing the Depth otherwise could quickly freeze Unity3D.

The world algorithms are the main components for terrain generation. There is a variety to choose from and is regularly expanded.



Describing every algorithm implemented is pretty overwhelming so it is easier to check the sample scene and experiment with them as they are updating automatically.

Every algorithm has an Apply Function which is important for mixing.

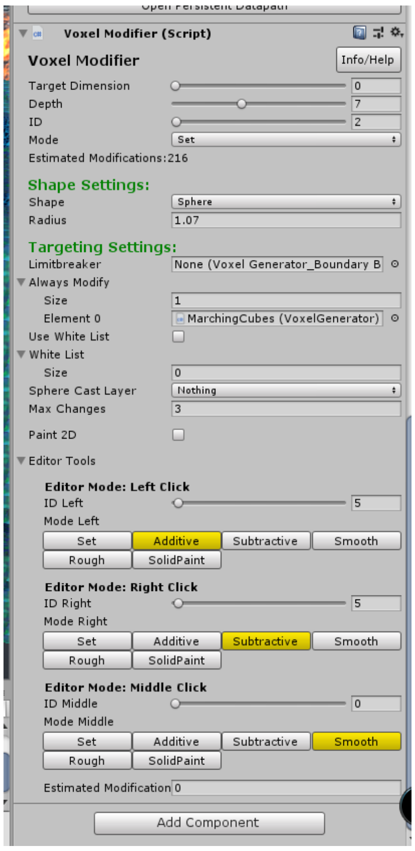


```
World Algorithm_Texture (Scrip
Script                   WorldAlgorithm_Texture
World Algorithm:                          Info/Help
World Generator          WorldGenerator_Sample (1
Apply Function           Set
Post Process Functior    None
Disabled                 ☐
Texture                  Crystal_baseColor
Offset                   X 0      Y 0.35    Z 0
Frequency                2.29
Height Multiplier        0.05
Fall Off                 3.8
Channel To Read          Red
              Add Component
```

The current options are Set, Add, Subtract, Min, Max and their inverted versions. The first algorithm usually has Set. The inverted versions apply 255 – value instead.

It is also important that some Apply functions are highly sensitive to negative values especially Add/Subtract. Therefore a post process function is also included.

VOXEL MODIFIER



Voxel Modifier (Script)

Voxel Modifier — Info/Help

Target Dimension	0
Depth	7
ID	2
Mode	Set
Estimated Modifications:	216
Shape Settings:	
Shape	Sphere
Radius	1.07
Targeting Settings:	
Limitbreaker	None (Voxel Generator_Boundary B
Always Modify	
Size	1
Element 0	MarchingCubes (VoxelGenerator)
Use White List	☐
White List	
Size	0
Sphere Cast Layer	Nothing
Max Changes	3
Paint 2D	☐
Editor Tools	
Editor Mode: Left Click	
ID Left	5
Mode Left	Set [✓ Additive] Subtractive Smooth Rough SolidPaint
Editor Mode: Right Click	
ID Right	5
Mode Right	Set Additive [✓ Subtractive] Smooth Rough SolidPaint
Editor Mode: Middle Click	
ID Middle	0
Mode Middle	Set Additive Subtractive [✓ Smooth] Rough SolidPaint
Estimated Modification	0

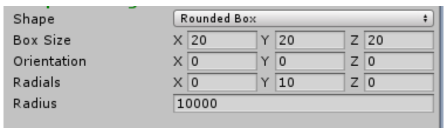
Add Component

The VoxelModifier is the third component and the main tool to modify voxels. This script allows painting during edit mode and provides other functionality to modify the dataset. It allows 3D painting on solid geometry but also supports 2D painting in cases where you want to use Voxel for 2D games. The most important parameters are the radius and depth. A safety system is implemented which prevents insane settings like radius 10000 and depth of 20.

Target Dimension: The dimension which should be modified. Standard value is dimension 0 which is the normal Solid-Nonsolid voxel map. What exactly is modified is highly dependent on the hull generator attached. Dimensions greater than 0 may be used for material, temperature, humidity, holy/unholy ground etc.

Depth: The target resolution.
ID: Independent ID used to modify the dataset when ModifyAtPos is executed.

Shape Settings:
The main basic shapes are Sphere, Box and Rounded Box and Single:



Shape	Rounded Box		
Box Size	X 20	Y 20	Z 20
Orientation	X 0	Y 0	Z 0
Radials	X 0	Y 10	Z 0
Radius	10000		

Sphere is the most basic form and is only defined by the radius parameter.
Box is defined by the Box Size which defines width, height and depth and a rotation parameter.
Rounded Box is similar to Box shape but has additional parameters for radials. This option provides the maximum amount of freedom without using more complex functionality. The sample image shows the settings for a cylinder with a radius of 10 in Y axis.

Single Modifies one single block (Minecraft Style). This mode has no parameter as it only modifies one single voxel according to the given world position.

The function called WorldPositionToVoxelPoint is used to evaluate the exact position of a voxel according to the target depth and returns the voxel position as world coordinate. Additionally a **normal direction** and **normal offset** can be used to apply an offset.

In order to simulate Minecraft like behavior (left click destroy, right click place), the **normal offset** should have a value of -0.5 when destroying a block and 0.5 when the user places a block with right click. The **normal direction** must be fetched from the raycast hit information.

Estimated Modification Count: Indicator of the safety system which shows how many voxels would be modified. Safety limit are 100000 voxels per modification. The referenced Voxel Generator is the one which is targeted first as multi block editing is possible.

Modes: There are separate modes for left and right click including a separate target ID applied.

- Set: Sets the voxel ID to the target ID.
- Additive: Adds the target ID to the voxel ID (target ID can be negative)
- Subtractive: Subtract the target ID to the voxel ID (target ID can be negative.
- Smooth: Applies mean filter to the target region. Target ID is used as strength.
- Rough: Unsmooth target region. If Voxel ID is greater than Target ID, it becomes 255 else it becomes 0.
- SolidPaint: Changes voxel ID to target ID if voxel ID is not zero (for AtlasCubes painting)

This component can be used to modify multiple VoxelGenerator at once. Therefore the VoxelModifier has a variety of options to determine the targeted generator. The simplest way to assign a generator is by simply adding one or more VoxelGenerator to the “Always Modify” list.

If **White List** is true, only VoxelGenerators assigned to the whitelist will be modified if they are fetched by the sphere cast.

The **Sphere Cast Layer** defines which layers can be hit by the sphere cast.

MaxChanges limits the amount of voxel generators which can be modified at the same time.

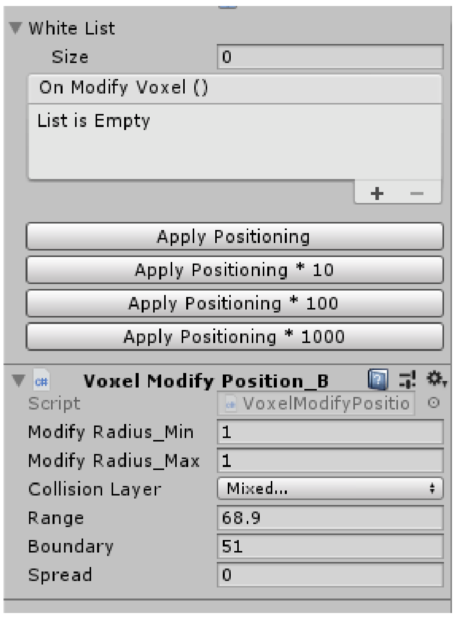
LimitBreaker: Is required when infinite world mode is active. When a VoxelGenerator_BoundaryBreaker script is assigned, the automatic fetching system is determined by the infinite world. This allows boundless sculpting.

Editor Tools: These are entries for editor sculpting and provide an option for left, right and middle mouse clicks. Each option has an ID and Mode parameter.

The main function **“ModifyAtPos”** will execute a modification command and requires the target world position as input parameter. This function is called by the sample controller whenever the player holds CTRL while clicking on a solid surface. This function can be called from anywhere such as a projectile calling this function on impact to destroy the environment. Usually you can also set ID and Radius to the desired values before calling it.

Additionally an “ApplyPositioning” function is included which requires a VoxelModifyPosition component and can be used for procedural voxel map generation.

POSITIONING ALGORITHMS



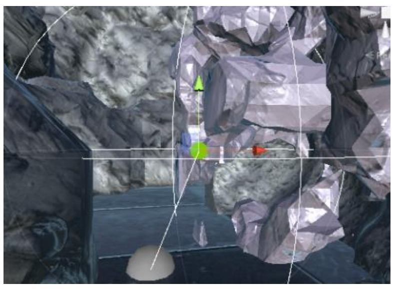
```
▼ White List
    Size                 0
  On Modify Voxel ()
  List is Empty
                                   +  −
           Apply Positioning
         Apply Positioning * 10
         Apply Positioning * 100
         Apply Positioning * 1000
▼ Voxel Modify Position_B
  Script                 VoxelModifyPositio
  Modify Radius_Min      1
  Modify Radius_Max      1
  Collision Layer        Mixed...
  Range                  68.9
  Boundary               51
  Spread                 0
```

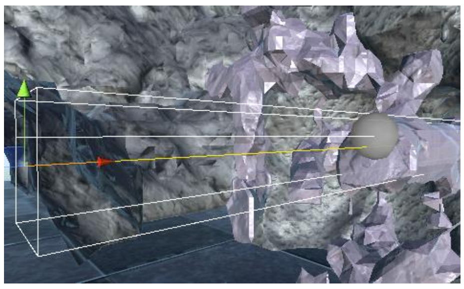
The **VoxelModifyPosition** are components used for procedural positioning for modifications. The sample scene Marching Cubes contains a snow producer GameObject which uses such component.

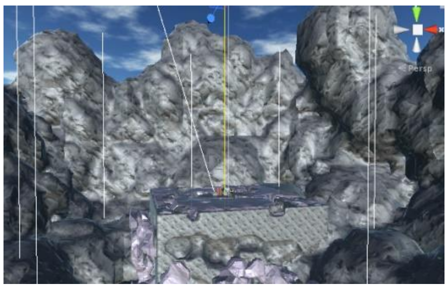
If the GameObject containing a VoxelModifier component also contains a VoxelModifyPosition, the VoxelModifier component will contain buttons to apply the positioning algorithm during edit mode.

The main parameter of a VoxelModifyPosition is the ModifyRadius Min/Max for randomness and the collision layer. When such GameObject is selected during edit mode, a preview will indicate where the positioning algorithm could be applied.

Currently included positioning algorithms are Sphere, Beam and Box. The spherical positioning can be used for explosion effects as It does ray casts from a spherical direction. Beam and Box are “lasers” where ray casts are shot in one direction where Beam is cylindrical and Box is rectangular.







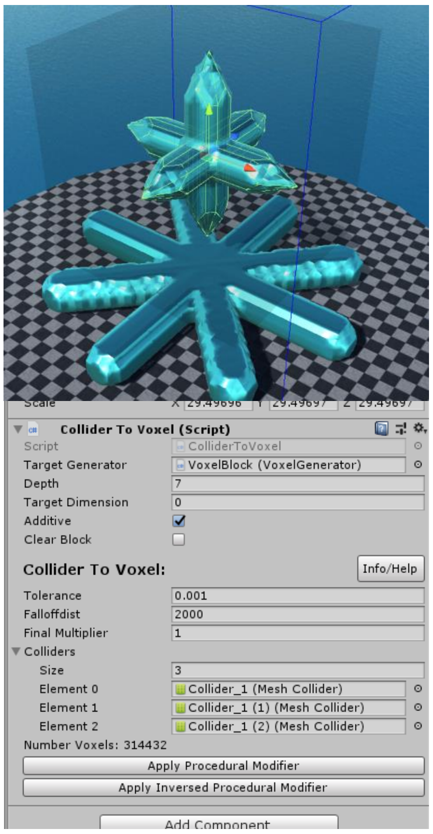
PROCEDURAL VOXEL MODIFIER

Procedural Voxel Modifier is a group of scripts supposed to modify the Voxel Data using algorithms and are completely independent.

Every Procedural Voxel Modifier has the **Target Generator, Depth and Target Dimension** as main parameters. Additionally you can decide if the result should be applied in an additive way or not and if the block should be cleared completely before applying the modifier (for biomes).

COLLIDER TO VOXEL CONVERTER

This procedural modification script converts any Unity3D collider into a voxel representation. Any collider attached as child to the GameObject will be used when the modification is applied.



Collider To Voxel (Script)

Script	ColliderToVoxel
Target Generator	VoxelBlock (VoxelGenerator)
Depth	7
Target Dimension	0
Additive	✓
Clear Block	☐

Collider To Voxel:

Tolerance	0.001
Falloffdist	2000
Final Multiplier	1
Colliders	
Size	3
Element 0	Collider_1 (Mesh Collider)
Element 1	Collider_1 (1) (Mesh Collider)
Element 2	Collider_1 (2) (Mesh Collider)

Number Voxels: 314432

Scale X 29.49696 Y 29.49697 Z 29.49697
Info/Help
Apply Procedural Modifier
Apply Inversed Procedural Modifier
Add Component

Mesh colliders must have “Convex” enabled. Therefore the best way to convert a concave model into a voxel representation is to create several convex mesh pieces and attach them to the converter.

The first parameter “**Tolerance**” defines how close a voxel must be to the collider that it becomes solid. This value usually is very small otherwise the result will become less accurate.

Falloffdist defines the solidity falloff when a voxel is not close enough to one or more collider. Large values cause a sharp falloff which makes the result blocky. Ideal value is between 1000 and 10000 which creates a smooth result.

Final Multiplier is a simple multiplier. When additive is set and the value is negative, material is removed instead.

In the editor window, the darkened array gives an indication about the voxel area which will be modified. The safety system limits the amount of voxels which can be modified to 5 million.

SUPER MESH TO VOXEL CONVERTER

This mesh to voxel converter was an attempt to improve the original mesh to voxel converter by implementing a custom evaluation method to determine if a voxel is inside or outside the mesh geometry. The functionality is identically to the original mesh to voxel converter but the parameters are slightly different:

Final Multiplier: Final multiplier of the values.

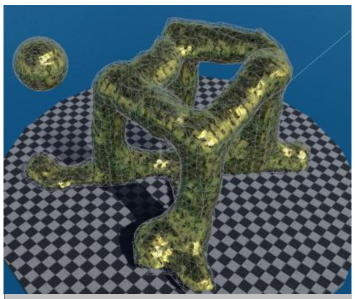
Smoothing; Value defines the smoothing filter applied to the result by applying an average filter over the result.

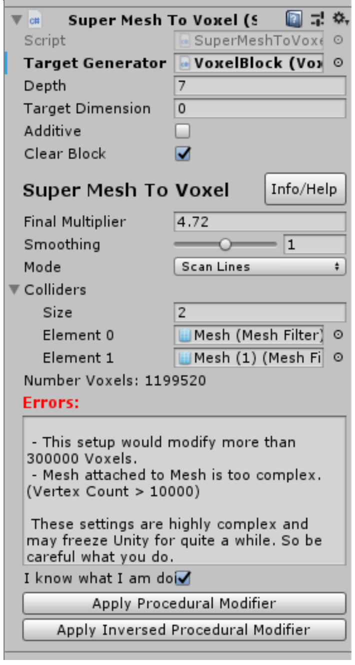
Mode: Evaluation algorithm used to define if inside or outside.

Brute force simply checks every voxel if it is inside or not. Brute force mostly is faster on simple meshes because it is very cache friendly especially when the evaluation is cheap.

Scan Lines uses the scanlines algorithm to convert a mesh to a voxel representation. It has to build the tree first which is costly but then the evaluation is much faster than brute force. Therefore it is faster when the mesh has a higher vertex count.

It is important to increase the depth only step wise as the number of voxels increases exponentially. Also the resolution of the hull generator should match the target depth in order to fully see the result.





Script	SuperMeshToVoxe
Target Generator	**VoxelBlock (Vox**
Depth	7
Target Dimension	0
Additive	☐
Clear Block	✓

Super Mesh To Voxel

Final Multiplier	4.72
Smoothing	1
Mode	Scan Lines
Colliders	
Size	2
Element 0	Mesh (Mesh Filter)
Element 1	Mesh (1) (Mesh Fi

Number Voxels: 1199520

Errors:

- This setup would modify more than 300000 Voxels.
- Mesh attached to Mesh is too complex. (Vertex Count > 10000)

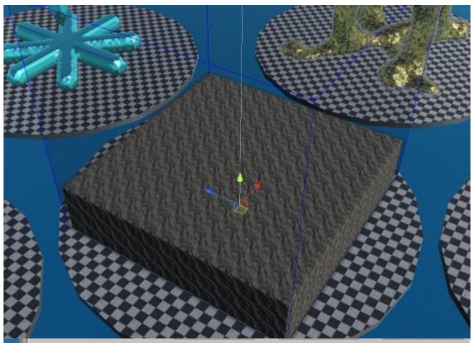
These settings are highly complex and may freeze Unity for quite a while. So be careful what you do.

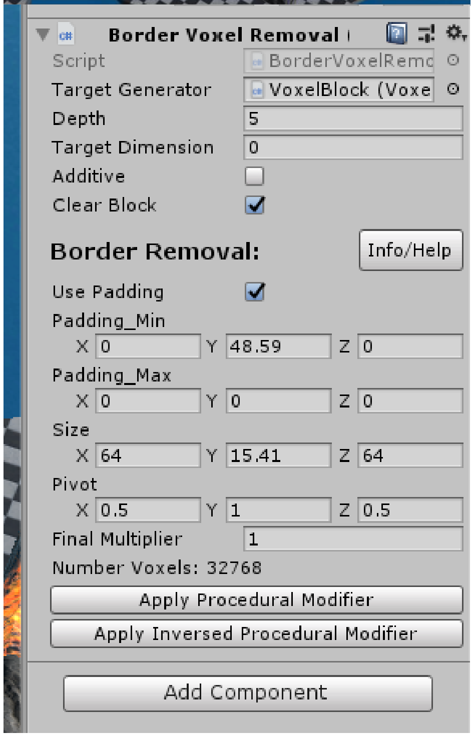
I know what I am doi ✓

BORDER REMOVER

This procedural modifier does nothing more than removing the border region of a block. And is the primary method to initialize a voxel block with a defined block shape.

It only has values to define the padding for each side but also has the option to use Size and Pivot Point. The affected volume always matches the volume size of the Target Generator.





Border Voxel Removal

Script	BorderVoxelRemc
Target Generator	VoxelBlock (Voxe
Depth	5
Target Dimension	0
Additive	☐
Clear Block	✓

Border Removal:

Use Padding	✓
Padding_Min	X 0 Y 48.59 Z 0
Padding_Max	X 0 Y 0 Z 0
Size	X 64 Y 15.41 Z 64
Pivot	X 0.5 Y 1 Z 0.5
Final Multiplier	1

Number Voxels: 32768

TEXTURE TO VOXEL

Texture to voxel converts a set of textures into a voxel representation. This modifier is very efficient and is able to generate highly accutate results. The result is generated layer by layer like a 3D printer because it uses the slices as input parameter. Every slice requires a texture assigned and defines one voxel layer.

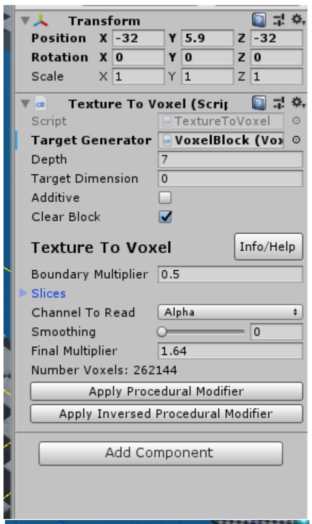
The right sample uses 32 slices in order to create the ornament like structure. In this case all 32 slices have the same texture(third image) assigned. Also it is possible to assign different textures for each slice so it is possible to visualize scan systems which create image slices.

The amount of voxel is automatically calculated using the depth and boundary multiplier so it fits into the voxel block perfectly.

The boundary multiplier is used to oversample or undersample the measurement as creating a 3D object using 2048*2048 may be too much.

You can also select which channel you want to read. Scans are usually in gray scale where it has no influence.

Also you can smooth the result afterwards which removes noise if your slices suffer from noises.



Transform

Position	X -32	Y 5.9	Z -32
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

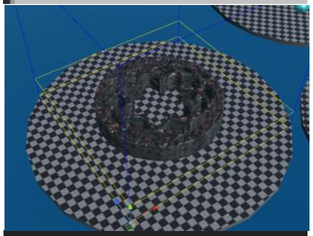
Texture To Voxel (Scrip

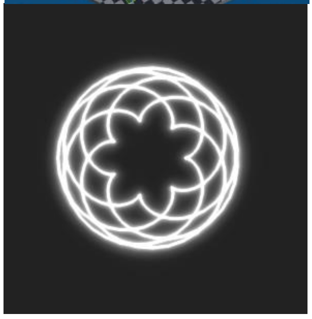
Script	TextureToVoxel
Target Generator	VoxelBlock (Vox
Depth	7
Target Dimension	0
Additive	☐
Clear Block	✓

Texture To Voxel

Boundary Multiplier	0.5
Slices	
Channel To Read	Alpha
Smoothing	0
Final Multiplier	1.64

Number Voxels: 262144

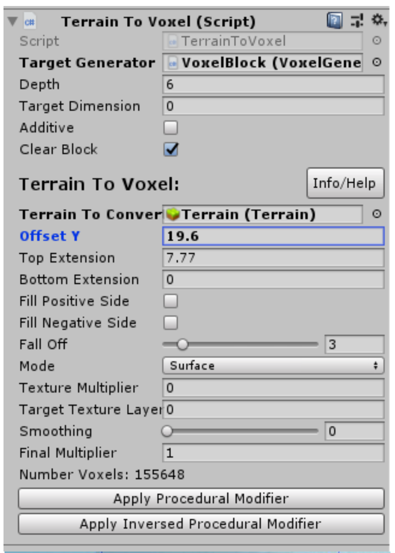




TERRAIN TO VOXEL

The terrain to voxel converter was a feature requested by customers and allows the conversion of Unity terrain into a voxel representation. One key aspect of terrains is that they have no 3D layout so it must be extrapolated with parameters. Also terrains have multi texture support which means that every texture added inside the terrain editor has a separate alpha map.

Therefore to fully convert a terrain into a voxel representation, you need 2 Terrain To Voxel modifiers. The first one has Surface mode selected and writes to Target Dimension 0 in order to create the surface geometry. The second modifier has to read the texture heightmaps of the terrain in order to write into the texture dimension (target dimension 1 or higher). The mode must also be set to "Dominant Texture" or "Individual Layer". Also the generator, modifier, and terrain ideally have the same world position like in the right images.

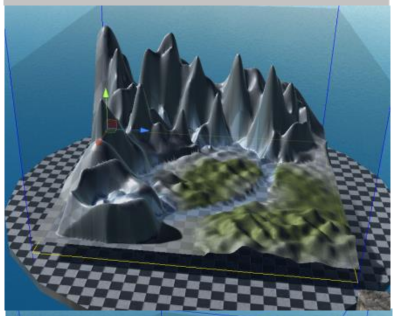


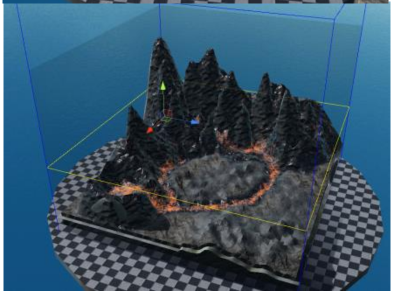
Script	TerrainToVoxel
Target Generator	VoxelBlock (VoxelGene
Depth	6
Target Dimension	0
Additive	☐
Clear Block	✓

Terrain To Voxel:

Terrain To Conver	Terrain (Terrain)
Offset Y	19.6
Top Extension	7.77
Bottom Extension	0
Fill Positive Side	☐
Fill Negative Side	☐
Fall Off	3
Mode	Surface
Texture Multiplier	0
Target Texture Layer	0
Smoothing	0
Final Multiplier	1

Number Voxels: 155648





This modifier has 3 modes. The first one is **"Surface".** Surface reads the heigth map in order to create a 3D gradient which can be moved with the **Offset Y** parameter. Also the modifier automatically calculates the boundary which can be extended using **Top/Bottom Extension**

Fill Positive/Negative sides would completely make voxels below or above 0 completely solid if set.

The **Fall Off** defines how fast the Density decreases the further away the voxel is from the surface.

Mode is the most important setting as it defines the evaluation method:

- **Surface:** Reads the height value of the terrain. Used to create the solid geometry.
- **Dominant Texture:** Reads all texture layers and uses the index of the texure layer with the highest value. Used for multi texture lookup and is multiplied by the Texture Multiplier parameter.
- **Individual Layer:** Reads the value of the layer defined by Target Texture Layer

As usual, the result can be smoothed by increasing the **smoothing** parameter.

TEXTURE TO VOXEL TERRAIN

This modifier is identically to the Terrain to Voxel modifier as it converts a 2D map into a voxel representation. The main difference is that it uses a texture as input parameter instead.

It has the same paramters as the Terrain to Voxel modifier.

First you have to select the channel you want to read. Options are Red, Green, Blue and Alpha.

Invert inverts the evaluation value. (0 becomes 1, 1 becomes 0).

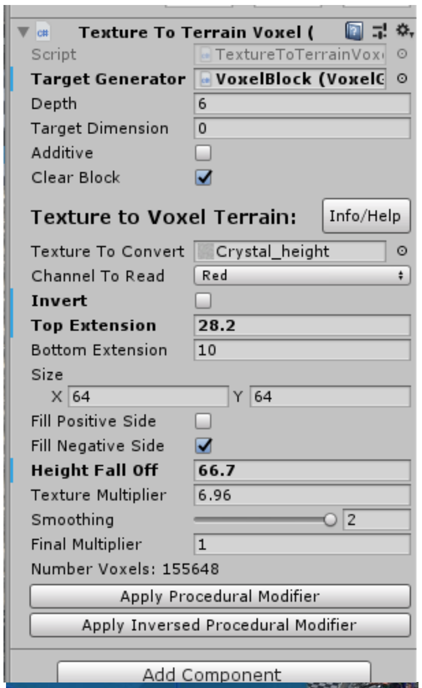
Size defines the X and Y size of the final result. The lookup is adapted automatically.

Height Fall Off defines how fast the voxel ID decreases the further it is away from the surface. If 0, the result has no fallof and behaves like **texture to voxel** modifier .

Texture Multiplier is multiplied into the texture value.

If the result should have solid ground, setting Fill Negative Side is recommended, else the result will be a pillar.

Unless Terrain To voxel, this modifier has no mode as there is only the assigned texture to read. The right image is created using 2 modifiers where one writes Target Dimension 0 and the other one Target Dimension 1.



Texture To Terrain Voxel (

Script	TextureToTerrainVox
Target Generator	**VoxelBlock (VoxelG**
Depth	6
Target Dimension	0
Additive	☐
Clear Block	✓

Texture to Voxel Terrain: Info/Help

Texture To Convert	Crystal_height
Channel To Read	Red
Invert	☐
Top Extension	**28.2**
Bottom Extension	10
Size	X 64 Y 64
Fill Positive Side	☐
Fill Negative Side	✓
Height Fall Off	**66.7**
Texture Multiplier	6.96
Smoothing	2
Final Multiplier	1

Number Voxels: 155648

Apply Procedural Modifier
Apply Inversed Procedural Modifier
Add Component

VOXEL COLLISION MODIFIER

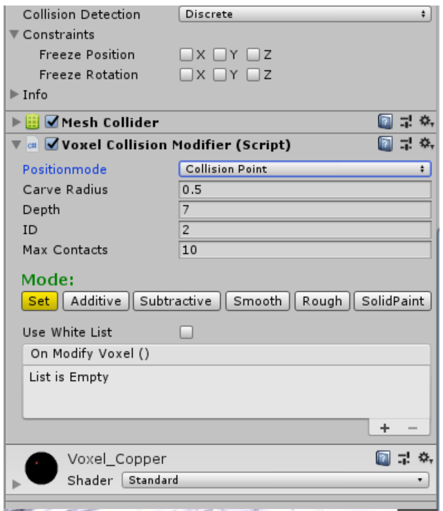
This component can be used on RigidBodies to modify a voxel map when the RigidBody collides with it. The main parameters also are **Depth, ID and Carve Radius**. The target VoxelGenerator is fetched automatically during collision.

Also the **positioning** can be set to center point or collision point. Center point will apply modification where the center of the RigidBody is used as center point. Collision Point will use the impact point calculated by the collision detection system as center point.

Max Contacts are used to determine how many contact points are used for modification.

It is also possible to limit the possible target generators by using whitelist When **Use White List** is true, a list appears where you can assign VoxelGenerators which should be modified by the RigidBody.

Mode can be used to define the modification type as it is seen in other modifiers.



Collision Detection	Discrete
Constraints	
Freeze Position	X Y Z
Freeze Rotation	X Y Z
Info	

Mesh Collider

Voxel Collision Modifier (Script)

Positionmode	Collision Point
Carve Radius	0.5
Depth	7
ID	2
Max Contacts	10

Mode:

[✓ Set] Additive Subtractive Smooth Rough SolidPaint

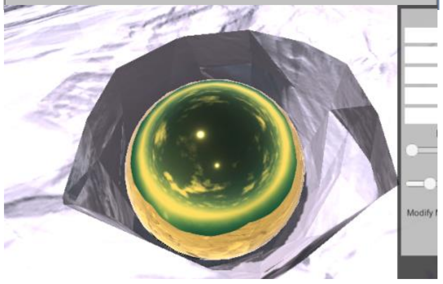
Use White List

On Modify Voxel ()

List is Empty

Voxel_Copper

Shader Standard



It is recommended to keep the amount of “Hot Objects” low since the whole process is very expensive. The sample scene MeltDown shows how a gold sphere is melting through the snow.

VOXEL UTILITY:

The main function provided by the Voxel Utility is the **ModifyVoxels** function and comes in 2 variations: One function for spherical and one for box shape modifications. The main input parameters for the standard **ModifyVoxels** function are:

- generator: the target VoxelGenerator:
- Center: center point of the modification in world coordinates
- Orientation: rotation of the modification
- Size: box size of the modification
- Radials: Vector4 parameter for radial limits: X, Y, Z is the radius for axis aligned circles and can be used to obtain cylindrical modification shapes. The last value W is used as spherical radius.
- Depth: target depth
- ID: ID of the modification
- Mode: Modification mode.
- Dimension: Target Dimension if voxel generator has more than one dimension.

Modes: There are separate modes for left and right click including a separate target ID applied.

1. Set: Sets the voxel ID to the target ID.
2. Additive: Adds the target ID to the voxel ID (target ID can be negative)
3. Subtractive: Subtract the target ID to the voxel ID (target ID can be negative.
4. Smooth: Applies mean filter to the target region (ignores target ID)
5. Rough: Unsmooth target region (128 becomes 255, 127 becomes 0)
6. SolidPaint: Changes voxel ID to target ID if voxel ID is not zero (for AtlasCubes painting)

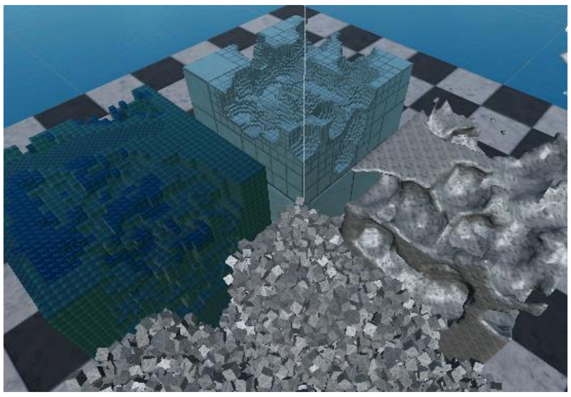
The spherical variant of the **ModifyVoxels** function has the same input parameter except size and radials as it is replaced by an radius parameter.

The box variant has the same input parameters except the radials as it is not needed.

Additionally, this helper class provides functions to obtain information about the modification before it starts. **EvaluateModificationCount** will calculate how many voxels will be modified. This function only needs the generator, radius or size and target depth as input parameter.

Before modifying anything, it is recommended to check, if the modification is save. Therefore **IsSave** function is provided which also requires generator, radius or size and target depth. This function will return true when the amount of voxels which will be modified is greater than the included safety limit of 100000 (you can change it on your own risk).

HULL GENERATORS



Hull generators are responsible for the visualization of the voxel map and must be attached as children to VoxelGenerator GameObjects. Each hull generator has its own settings such as resolution and material and any number of hull generators can be attached to the VoxelGenerator. This allows the generation voxel blocks with very unusual properties.However also keep in mind that more hull generators result in a slower update rate when the voxel map is modified as there is more work to do.

The basic parameters which can be found on every hull generator is the **Engine** itself which is fetched automatically when attached to the VoxelGenerator.

Applied Hideflags are set to “Hidden Don’t Save” by default which hides generated GameObjects and prevents them from being saved into the Unity scene file. Saving the hulls is not necessary because it can be reconstructed at any time by the VoxelGenerator which also applies inside edid mode. Therefore saving generated hulls is obsolete and only would bloat up the scene file. Other options are “Normal”, “Don’t Save” and Hidden which are all combinations of Visible and being saved.

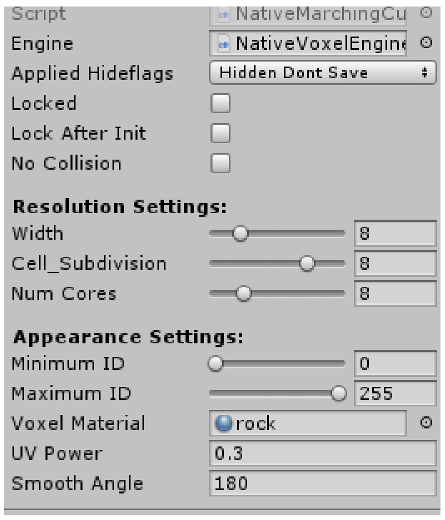
If **Locked** is set, the hull generator will not update his content.

If **Lock after init** is set, the hull generator will only update its content when the VoxelGenerator is initialized. Afterwards the hull generator will be locked until the VoxelGenerator is cleaned.

MARCHING CUBES

Marching cubes is the most common visualization method for voxels and is mostly used for terrain or other natural environments because the appearance is completely smooth. The smoothness is caused by the ID value which is between 0 and 255 so even voxel maps with low resolution have smooth surfaces.

The first settings are the resolution settings. These are common in other hull generators and define the resolution of the final geometry. Changing most settings while the VoxelGenerator is initialized will trigger rebuilding the hull generators.



Script	NativeMarchingCu
Engine	NativeVoxelEngin
Applied Hideflags	Hidden Dont Save
Locked	☐
Lock After Init	☐
No Collision	☐
Resolution Settings:	
Width	8
Cell_Subdivision	8
Num Cores	8
Appearance Settings:	
Minimum ID	0
Maximum ID	255
Voxel Material	rock
UV Power	0.3
Smooth Angle	180

Width: Defines the resolution of each mesh piece. For example a width of 8 means that one piece contains 8x8x8 or 512 voxels.

Cell_Subdivision: Defines how many mesh pieces will be generated. For example a value of 8 means that 512 mesh pieces will be generated where each mesh piece will cover 512 voxels if width is set to 8. The total voxel amount covered by this voxel block would then be 512x512 (262144) voxels.

NumCores: This value defines how many mesh pieces are processed each frame when the hull must be updated. The works of every mesh piece are evenly distributed along the CPU cores (is decided by the Unity Job System)

The appearance settings are specific for each type of hull generator and define the visual appearance. It usually contains a material, UV power and optional smoothing of vertex normals.

MinimumID/MaximumID : The ID range covered by marching cubes**.** Solid geometry starts to appear when the ID value of a voxel is in the upper half of the covered ID range. For example solid geometry begins at 128 when the default range of 0-255 is used.

VoxelMaterial: Material used for the mesh pieces.

UV Power: UV Multiplier for the mesh pieces

Smooth Angle: Smoothing Angle of the vertex normals.

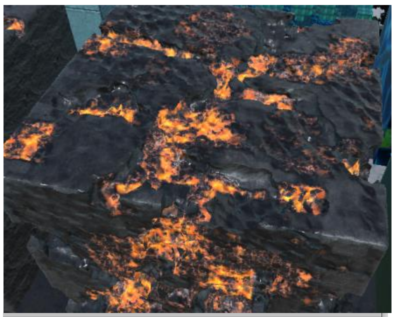
MULTI TEXTURE SUPPORT (ADVANCED)

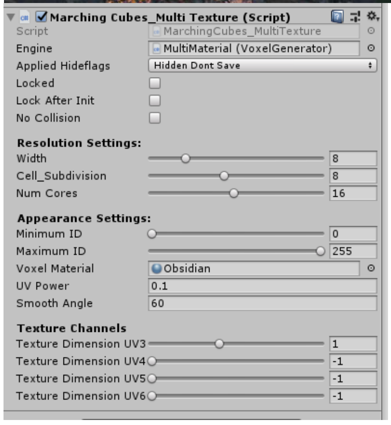
Since Update 1.5 Multi Texture is possible. In order to use this feature, use MarchingCubes_MultiTexture instead. Also the dimension count of the VoxelGenerator must be two or more. The first dimension is used to describe solid or non-solid and the additional dimensions are used to write into the respective UV coordinates. The texture channels define which UV set should be affected by which dimension.

In the example image, the second dimension writes to the UV3 coordinate of the procedurally generated mesh. The shader of the assigned material uses the UV3 value to define which texture should be applied. The material uses a specific shader which uses Texture2DArrays instead of normal textures.

The simplest shader included in the core library reads UV3 in order to determine the blending value.

Also this hull generator uses the vertex color array of the mesh as barycentric coordinate which is required to implement 100% seamless tessellation.





Marching Cubes_Multi Texture (Script)

Script	MarchingCubes_MultiTexture
Engine	MultiMaterial (VoxelGenerator)
Applied Hideflags	Hidden Dont Save
Locked	☐
Lock After Init	☐
No Collision	☐

Resolution Settings:

Width	8
Cell_Subdivision	8
Num Cores	16

Appearance Settings:

Minimum ID	0
Maximum ID	255
Voxel Material	Obsidian
UV Power	0.1
Smooth Angle	60

Texture Channels

Texture Dimension UV3	1
Texture Dimension UV4	-1
Texture Dimension UV5	-1
Texture Dimension UV6	-1

USING TEXTURE ARRAYS

The included core library contains such shader which replicates the Unity3D Standard shader but with tessellation and multi texture arrays. Most settings are almost identically to the standard shader.

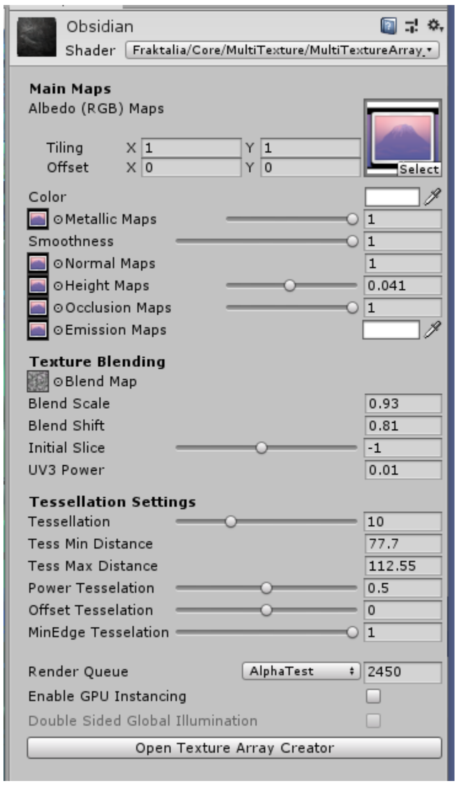
Texture Blending contains settings which define how to blend between texture slices inside the assigned texture arrays. A blend map can be assigned in order to add some variation to the blending. **Blend Scale and Blend Shift** describes the influence of the Blend Map.

Initial slice defines the initial blending state.

UV3 Power describes how the UV coordinate influences the blending state. Since possible values are between 0 and 255, this value is usually very low (1 / 256).

The Tessellation settings are used to further enhance the voxel representation even on low resolution voxel maps. The first parameter is the tessellation strength where higher values subdivide the mesh the most.

Min/Max Distance: Parts of the mesh which are further away from the camera are not subdivided in order to save GPU power. Parts below the minimum distance are subdivided the most.



Obsidian

Shader: Fraktalia/Core/MultiTexture/MultiTextureArray_

Main Maps

Albedo (RGB) Maps

	X	Y
Tiling	1	1
Offset	0	0

Color	
Metallic Maps	1
Smoothness	1
Normal Maps	1
Height Maps	0.041
Occlusion Maps	1
Emission Maps	

Texture Blending

Blend Map	
Blend Scale	0.93
Blend Shift	0.81
Initial Slice	-1
UV3 Power	0.01

Tessellation Settings

Tessellation	10
Tess Min Distance	77.7
Tess Max Distance	112.55
Power Tesselation	0.5
Offset Tesselation	0
MinEdge Tesselation	1

Render Queue	AlphaTest 2450
Enable GPU Instancing	☐
Double Sided Global Illumination	☐

CREATING TEXTURE ARRAYS:

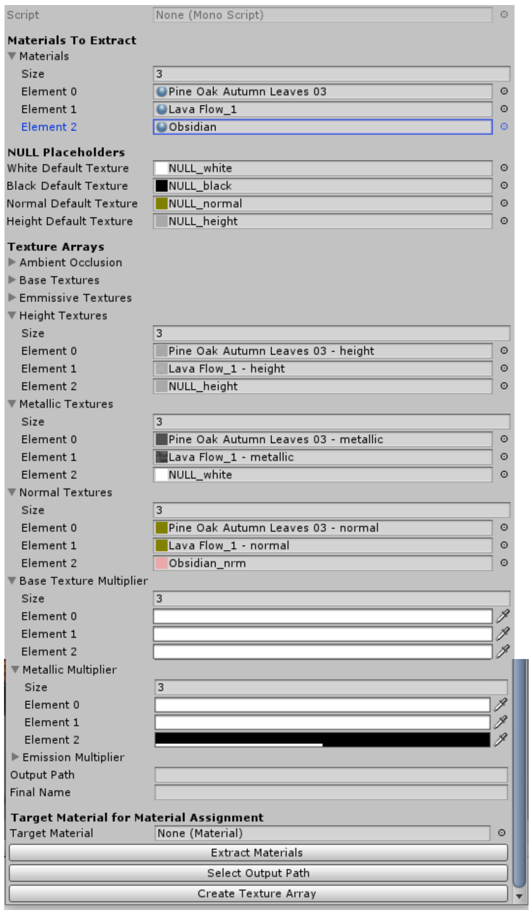
Unfortunately, Unity3D does not provide a user friendly way to create texture arrays. Therefore when clicking the “Open Texture Array Creator”, a custom editor window will popup which allows you to create Texture Arrays.

This tool can extract the textures used in selected materials which use the standard shader and bakes those textures into texture arrays. If the material does not use the standard shader, Null textures are used instead. The perfect materials are Substance materials.

Color arrays will also be multiplied into baked textures since every material could have different Metallic, Emission and Main Colors.

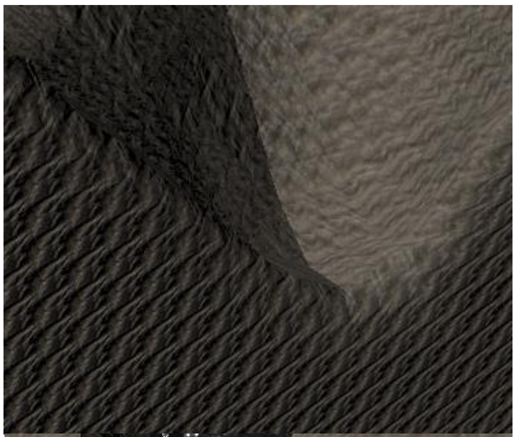
However it is also possible to assign materials manually by simply filling the arrays.

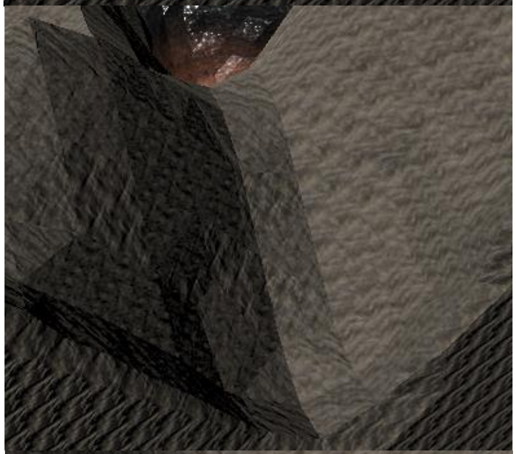
Then you can select the output path and name. If the output path is not valid, the root asset folder is used instead. You can assign a target material which supports texture arrays and the created texture arrays are assigned automatically for you.

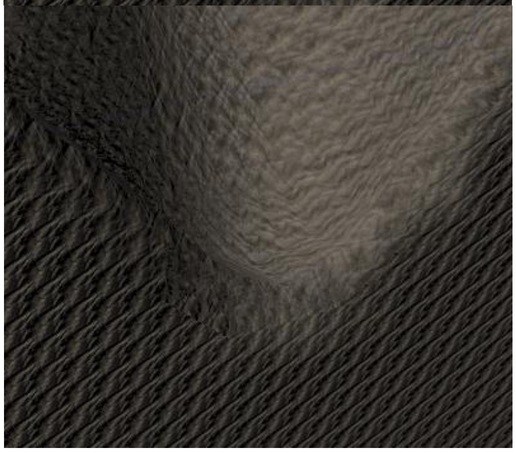


Script	None (Mono Script)
Materials To Extract	
▼ Materials	
Size	3
Element 0	Pine Oak Autumn Leaves 03
Element 1	Lava Flow_1
Element 2	Obsidian
NULL Placeholders	
White Default Texture	NULL_white
Black Default Texture	NULL_black
Normal Default Texture	NULL_normal
Height Default Texture	NULL_height
Texture Arrays	
▶ Ambient Occlusion	
▶ Base Textures	
▶ Emmissive Textures	
▼ Height Textures	
Size	3
Element 0	Pine Oak Autumn Leaves 03 - height
Element 1	Lava Flow_1 - height
Element 2	NULL_height
▼ Metallic Textures	
Size	3
Element 0	Pine Oak Autumn Leaves 03 - metallic
Element 1	Lava Flow_1 - metallic
Element 2	NULL_white
▼ Normal Textures	
Size	3
Element 0	Pine Oak Autumn Leaves 03 - normal
Element 1	Lava Flow_1 - normal
Element 2	Obsidian_nrm
▼ Base Texture Multiplier	
Size	3
Element 0	
Element 1	
Element 2	
▼ Metallic Multiplier	
Size	3
Element 0	
Element 1	
Element 2	
▶ Emission Multiplier	
Output Path	
Final Name	
Target Material for Material Assignment	
Target Material	None (Material)

NEW CALCULATOR







Marching Cubes has now a new option to use an improved calculator which derives the vertex normal from the voxel map.

The vertex normal from the standard normal calculation has artifacts between cells when the smooth angle is greater than 0 (Top Image). Also the calculation of smoothed angles is very expensive when the mesh itself is used to calculate those normal.

Flat shading (Middle Image) does not have these artifacts as they are calculated from each face and is cheap to compute. However it only makes sense in low poly environments.

The bottom image shows the result of the new normal calculator which is seamless and perfectly smooth. The only drawback is the imperfect Cube-Based UV coordinates. Therefore using a triplanar shader is highly recommended.

TEXTURE ATLAS SUPPORT

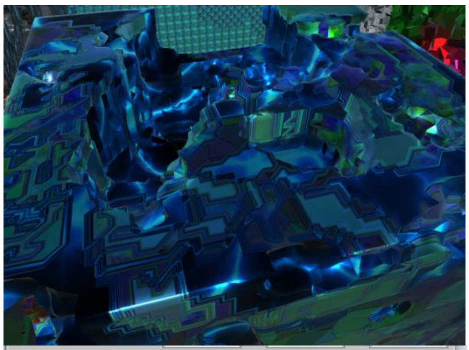
While the first multi texture support creates natural locking results since it allows seamless blending between multiple textures, this method uses a conventional texture atlas which cannot create seamless transitions. Since this method only modifies the UV1 coordinate for the texture atlas, this method does not require special texture array shader.

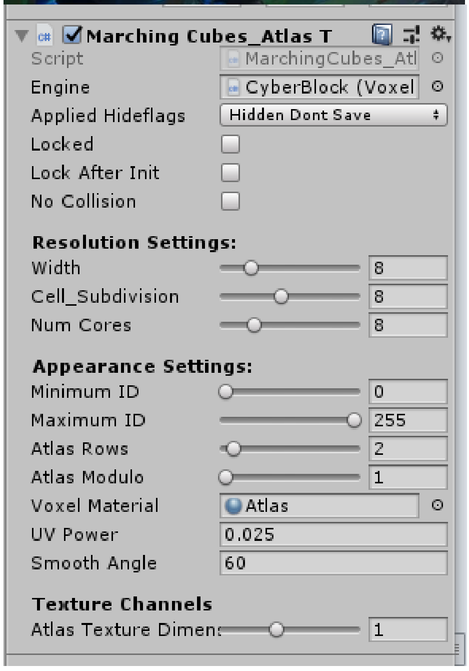
The disadvantage is that smooth blending between textures is impossible. This gives the result an artificial and tech-like result. The parameters are almost identically to the normal marching cubes hull generator.

Atlas Row defines the subdivision of the atlas texture. For example if the atlas contains 4 textures, the row count is 2.

Atlas Modulo is used to define vertex skips and alters the final appearance. Nice values are 0, 3 and 6.

The last parameter is the **texture dimension** which defines the voxel dimension for the texture evaluation.

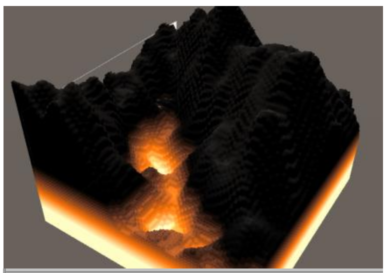


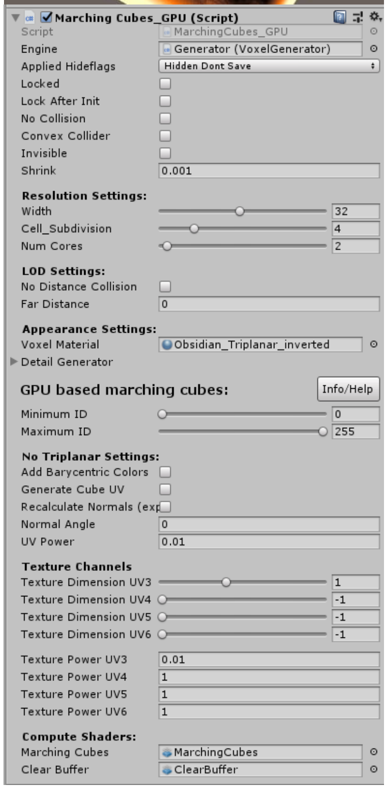


Marching Cubes_Atlas T

Script	MarchingCubes_Atl
Engine	CyberBlock (Voxel
Applied Hideflags	Hidden Dont Save
Locked	☐
Lock After Init	☐
No Collision	☐
Resolution Settings:	
Width	8
Cell_Subdivision	8
Num Cores	8
Appearance Settings:	
Minimum ID	0
Maximum ID	255
Atlas Rows	2
Atlas Modulo	1
Voxel Material	Atlas
UV Power	0.025
Smooth Angle	60
Texture Channels	
Atlas Texture Dimen:	1

GPU BASED MARCHING CUBES





```
Marching Cubes_GPU (Script)
Script                        MarchingCubes_GPU
Engine                        Generator (VoxelGenerator)
Applied Hideflags             Hidden Dont Save
Locked                        ☐
Lock After Init               ☐
No Collision                  ☐
Convex Collider               ☐
Invisible                     ☐
Shrink                        0.001

Resolution Settings:
Width                         32
Cell_Subdivision              4
Num Cores                     2

LOD Settings:
No Distance Collision         ☐
Far Distance                  0

Appearance Settings:
Voxel Material                Obsidian_Triplanar_inverted
▶ Detail Generator

GPU based marching cubes:                      Info/Help
Minimum ID                    0
Maximum ID                    255

No Triplanar Settings:
Add Barycentric Colors        ☐
Generate Cube UV              ☐
Recalculate Normals (exp      ☐
Normal Angle                  0
UV Power                      0.01

Texture Channels
Texture Dimension UV3         1
Texture Dimension UV4         -1
Texture Dimension UV5         -1
Texture Dimension UV6         -1

Texture Power UV3             0.01
Texture Power UV4             1
Texture Power UV5             1
Texture Power UV6             1

Compute Shaders:
Marching Cubes                MarchingCubes
Clear Buffer                  ClearBuffer
```

All components of the Voxel Generator are using the CPU for calculations while the GPU is only used for the traditional rendering. Modern GPUs are very powerful and their computation power can be used for hull generation. Therefore this hull generator now combines CPU and GPU in order to maximize computation speed.

The main disadvantage is that this hull generator only works on hardware which supports Compute Shaders. Check page provided by Unity3D if your hardware supports compute shaders:

https://docs.unity3d.com/Manual/class-ComputeShader.html

The settings are identically than the normal CPU based Marching Cubes but is Triplanar by default. The vertex normals are directly calculated using the Voxel Map and allows perfect Tessellation.

The No Triplanar Settings allow the extra calculations of desired properties at the cost of extra computation time. Especially the recalculation of normal based on the mesh (method used by all other hull generators) is very expensive.

Also by default, no UV1 coordinates are used as it is highly recommended to use the now included triplanar shader but it is possible to generate the Cube-Based UV coordinates like in CPU based Marching Cubes.

The last properties are the compute shaders which can be assigned but are loaded for you if they are not assigned.

This hull generator generates a crystalline structure similar to the structure found in the Crystal Generator asset. This generator also has the resolution settings which are also found in marching cubes hull generator. The only difference between crystal and marching cubes are the crystal shape settings:

Crystal Mesh: Defined crystal shape. Low poly shapes recommended.

Voxel Material: Material applied.

Offset Min/Max: Positional offset.

Scale Min/Max: Random scaling of each crystal

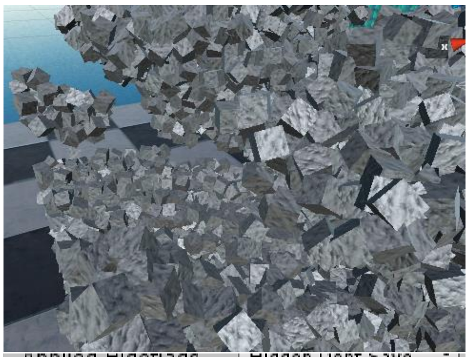
Scale Factor Min/Max: Multiplier.

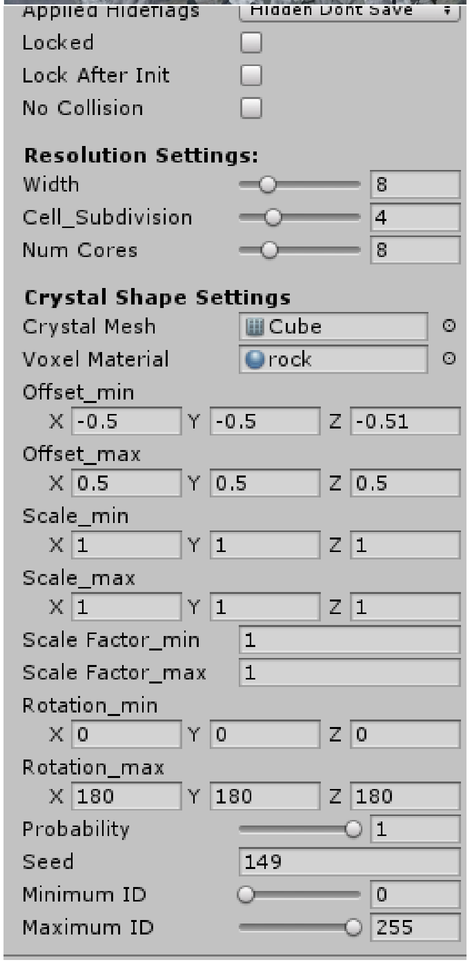
Rotation Min/Max: Random rotation of each crystal

Probability: Probability of crystals being generated for each voxel creating a surface.

Seed: Seed used for randomness.

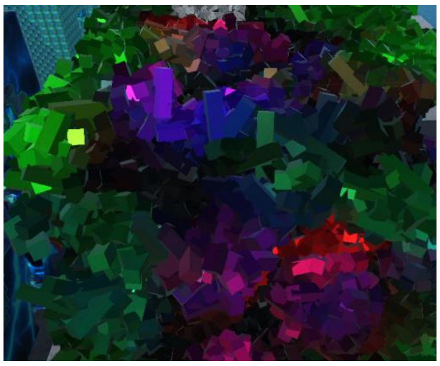
Min/Max ID: Similar to marching cubes and used to determine surface regions.





Applied Hideflags	Hidden Dont Save		
Locked	☐		
Lock After Init	☐		
No Collision	☐		
Resolution Settings:			
Width	8		
Cell_Subdivision	4		
Num Cores	8		
Crystal Shape Settings			
Crystal Mesh	Cube		
Voxel Material	rock		
Offset_min	X -0.5	Y -0.5	Z -0.51
Offset_max	X 0.5	Y 0.5	Z 0.5
Scale_min	X 1	Y 1	Z 1
Scale_max	X 1	Y 1	Z 1
Scale Factor_min	1		
Scale Factor_max	1		
Rotation_min	X 0	Y 0	Z 0
Rotation_max	X 180	Y 180	Z 180
Probability	1		
Seed	149		
Minimum ID	0		
Maximum ID	255		

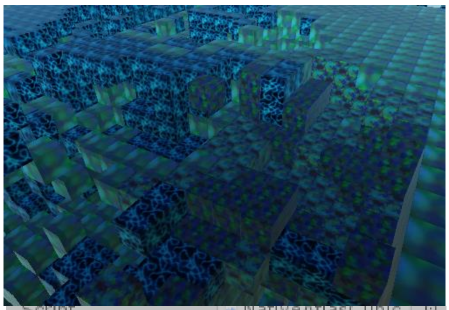
MULTI COLOR CRYSTAL (ADVANCED)

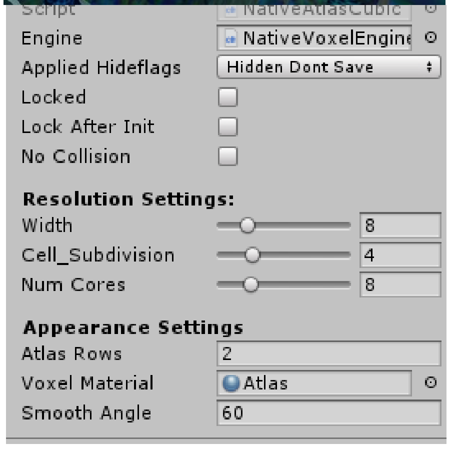


This variation implements multi colored crystals and has the same properties as the crystal hull generator. It has extra channels which modify UV3-UV6. This shader also requires a special shader which uses those extra UV coordinates. The core library includes a standard surface shader which uses UV3-UV6 as color values.

This hull generator can also be used to create crystals with different materials using texture arrays such as the marching cubes variations. In the end, the assigned material decides the purpose of the extra UV coordinates.

ATLAS CUBES





Script	NativeAtlasCubic
Engine	NativeVoxelEngine
Applied Hideflags	Hidden Dont Save
Locked	☐
Lock After Init	☐
No Collision	☐
Resolution Settings:	
Width	8
Cell_Subdivision	4
Num Cores	8
Appearance Settings	
Atlas Rows	2
Voxel Material	Atlas
Smooth Angle	60

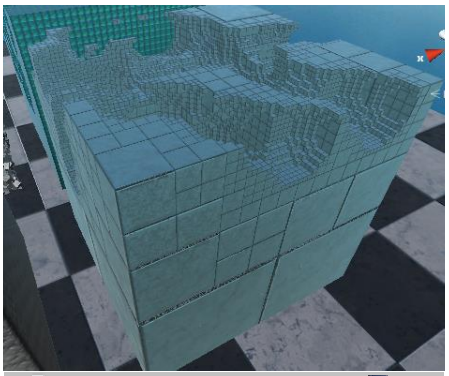
Atlas cubes is the typical hull found in Minecraft where ID has its own UV coordinate assigned. This allows the visualization of 255 different blocks. The original sprite sheet is a 2048x2048 texture which is subdivided into 16x16 cells. The current method in Minecraft uses strings which are not supported by the Unity Job system yet.

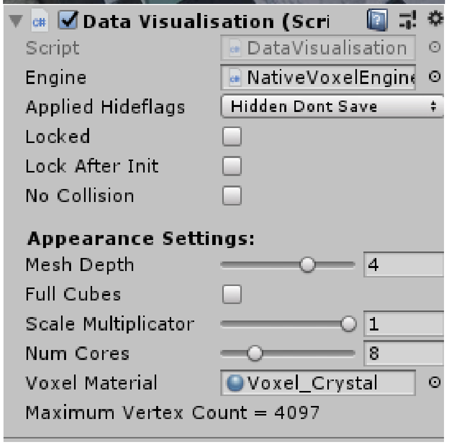
This hull generator also has the resolution settings found in the previous hull generators. The only new setting is the **Atlas Rows** which defines the cell subdivision of the sprite sheet.

The cheap sprite atlas provided in the sample has 2 rows and 2 columns. Therefore the Atlas Rows is set to 2.

Voxel Material and Smooth Angle are identical to the previous hull generators.

DATA VISUALIZER





Data Visualisation (Scri

Script	DataVisualisation
Engine	NativeVoxelEngine
Applied Hideflags	Hidden Dont Save
Locked	☐
Lock After Init	☐
No Collision	☐

Appearance Settings:

Mesh Depth	4
Full Cubes	☐
Scale Multiplicator	1
Num Cores	8
Voxel Material	Voxel_Crystal

Maximum Vertex Count = 4097

This hull generator works completely different than the previous hull generators. Instead of covering a fixed region of voxels, this hull generator directly converts the voxel map into a visual representation. Voxel ID greater than 0 is visualized as solid block. Also this generator has no resolution setting and the main settings related to the appearance only.

Mesh Depth: Describes the complexity of each mesh piece. Lower value means more GameObjects are required to represent the voxel map. Higher value mean that less GameObjects are required but the vertex count of each mesh piece is higher. This setting also includes a safety check especially when the subdivision power of the VoxelGenerator is greater than 2.

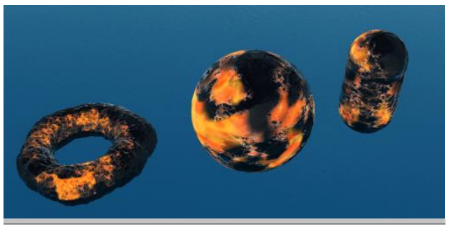
Full Cubes: Walls which are obstructed by solid voxels are usually not generated in order to reduce vertex count. If true, this is suppressed and therefore only useful when the scale multiplier is smaller than one.

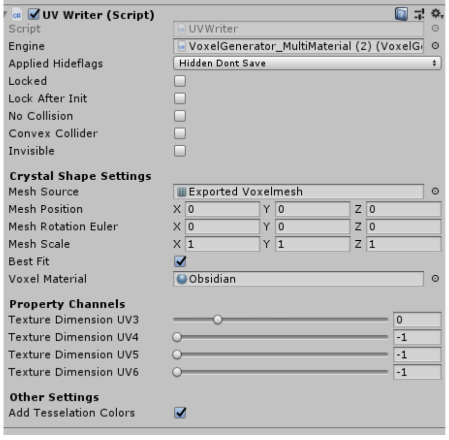
Scale Multiplier: Reduces the scale of individual voxel blocks. The result can be used to generate fancy effects.

NumCores: Update speed evened out on multi core CPU as seen in other hull generators.

VoxelMaterial: Applied Material

UV WRITER





✓ UV Writer (Script)

Script	UVWriter					
Engine	VoxelGenerator_MultiMaterial (2) (VoxelG					
Applied Hideflags	Hidden Dont Save					
Locked	☐					
Lock After Init	☐					
No Collision	☐					
Convex Collider	☐					
Invisible	☐					
Crystal Shape Settings						
Mesh Source	Exported Voxelmesh					
Mesh Position	X	0	Y	0	Z	0
Mesh Rotation Euler	X	0	Y	0	Z	0
Mesh Scale	X	1	Y	1	Z	1
Best Fit	✓					
Voxel Material	Obsidian					
Property Channels						
Texture Dimension UV3	0					
Texture Dimension UV4	-1					
Texture Dimension UV5	-1					
Texture Dimension UV6	-1					
Other Settings						
Add Tesselation Colors	✓					

The UV Writer is a hull generator which manipulates the additional UV coordinates of an assigned mesh shape. The sample shows the default sphere, capsule and a exported mesh which are modified using the UV Writer.

This hull generator makes only sense if the additional UV channels 3-6 are utilized by the material. Therefore custom shader who use those channels are mandatory (or you will see no effect).

Mesh Source: The reference mesh which should be modified. It is important to know that the result is a new mesh which is generated during initialization. Therefore the original mesh is never modified. Use the mesh exporter from the VoxelGenerator if the result should be saved.

Mesh Position/Rotation/Scale: Defines how the mesh should be positioned inside the volume boundary.

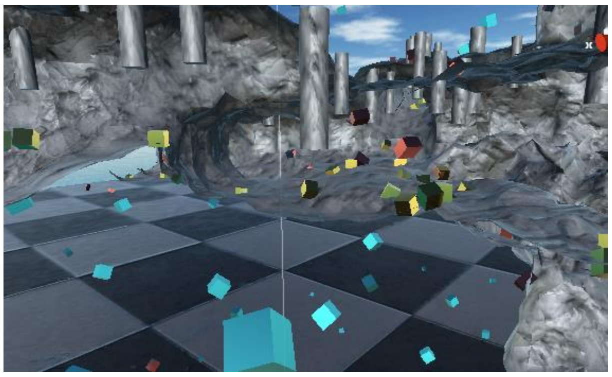
Best Fit: The mesh is positioned inside the volume boundary so it fits perfectly while keeping aspect ratio.

VoxelMaterial: Applied Material

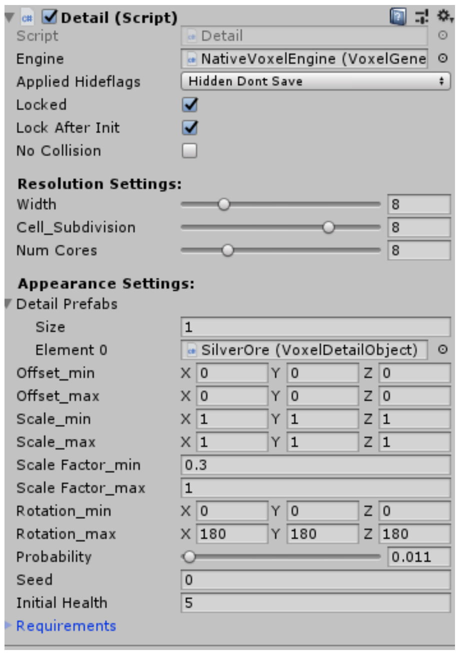
Property Channels: Define which voxel dimension should be written into which UV coordinate. A value of -1 will write nothing into the affected UV coordinate.

Add Tesselation Colors: When true, barycentric coordinates are written into the vertex color channels which are important when using wireframe materials or seamless tessellation features.

DETAIL



This hull generator is the most complex generator and allows the placement of detail objects into the interior of the voxel map. Any GameObject can be assigned as Detail object which will then be placed over as it is shown in the right image. Silver cubes are generated inside the solid volume. Gold Ore is only generated on the ground surface, Copper on any surface and Stalagmite Cylinders on the ceiling. The placement behavior is defined by a variable amount of requirements. Every detail at a voxel position can only be generated once. Also for ores which should be dig out, it is recommended to set LockAfterInit to true else ores may spawn while the player is digging where some voxels meet the requirements which were not met before and place details. Sometimes you actually want this feature for example the randomly placement of toxic gas on the surface when the player removes material.



Detail (Script)

Script	Detail					
Engine	NativeVoxelEngine (VoxelGene					
Applied Hideflags	Hidden Dont Save					
Locked	✓					
Lock After Init	✓					
No Collision	☐					

Resolution Settings:

Width	8
Cell_Subdivision	8
Num Cores	8

Appearance Settings:

Detail Prefabs

Size	1					
Element 0	SilverOre (VoxelDetailObject)					
Offset_min	X	0	Y	0	Z	0
Offset_max	X	0	Y	0	Z	0
Scale_min	X	1	Y	1	Z	1
Scale_max	X	1	Y	1	Z	1
Scale Factor_min	0.3					
Scale Factor_max	1					
Rotation_min	X	0	Y	0	Z	0
Rotation_max	X	180	Y	180	Z	180
Probability	0.011					
Seed	0					
Initial Health	5					

Requirements

The Detail hull generator also has the resolution settings as found in previous hull generators. The most important parameter of the appearance setting is the **Detail Prefabs** list where you assign any number of GameObjects which have a special VoxelDetailObject script as component.

When a voxel meets the requirements and the probability also matches, a random object from the Detail Prefabs list is chosen and a instance is placed at the voxel coordinates.

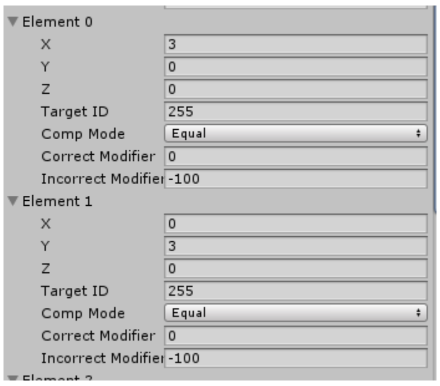
The position, rotation and scale of placed detail objects are further manipulated by:

- Offset_Min/Max
- Rotation_Min/Max
- Scale_min/max, Scalefactor

The **probability** parameter decides if a detail is placed even when requirements are met. It is generally recommended to keep the probability low as the amount of GameObjects could massively increase if the requirements are not strict enough. The randomness is completely deterministic and is based on the **seed** parameter.

NOTE: THIS HULL GENERATOR IS NOT SUITABLE FOR INFINITE WORLD SYSTEMS

REQUIREMENTS



Element 0	
X	3
Y	0
Z	0
Target ID	255
Comp Mode	Equal
Correct Modifier	0
Incorrect Modifier	-100
Element 1	
X	0
Y	3
Z	0
Target ID	255
Comp Mode	Equal
Correct Modifier	0
Incorrect Modifier	-100

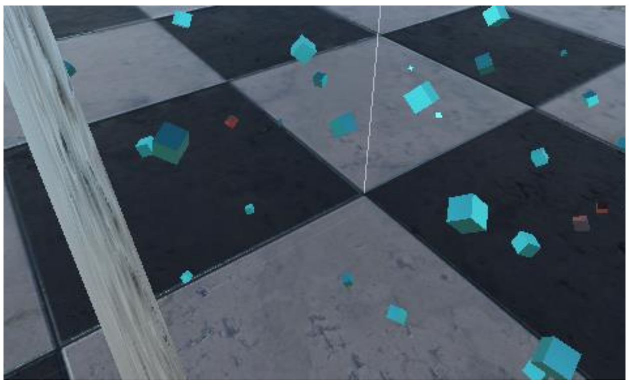
The requirements are used to evaluate if the environment of a voxel has specific values and fit given specifications. The Requirements list can contain any number of detail entries where the **InitialHealth** is the main starting parameter.

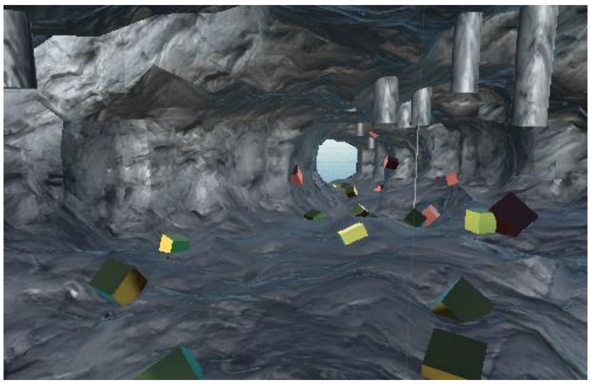
When a voxel is evaluated, the InitialHealth is the starting parameter. A voxel meets the requirements if the InitialHealth is greater than 0. The right sample entry checks the ID of a voxel which is 3 blocks away from the evaluating voxel. If the ID of the neighbor is not 255 or in other words completely solid, the health will drop to -100 invalidating the evaluating voxel immediately. If position X, Y, Z is 0, the ID of the evaluating voxel is checked. This sample has 4 more entries which also checks the neighbor for the 4 other directions and the resulting effect is that details can only be placed when surrounded by solid geometry.

Detailed Parameter Information:

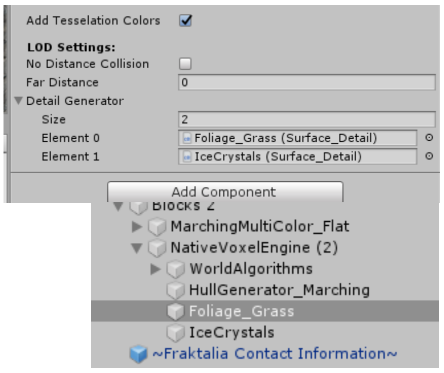
- **X, Y, Z:** Coordinate of neighbored voxel which should be checked. If zero, the voxel which should be evaluated is checked.
- **Target ID:** The target ID which a neighbored voxel should be compared to.
- **Comp Mode:** Comparison mode. Possible options are Equal, Not Equal, Greater and Smaller.
- **Correct Modifier:** Changes in health if this specific requirement is correct.
- **Incorrect Modifier:** Changes in health if incorrect.

The sample scene “Detail” shows what is possible. It is possible to place objects inside the solid volume like in the left image where silver cubes are placed but it is also possible to place objects on the surface only like in the right image. In the right image, gold ores will only spawn on ground while stalagmites can only appear on the ceiling.





SURFACE MODIFIER

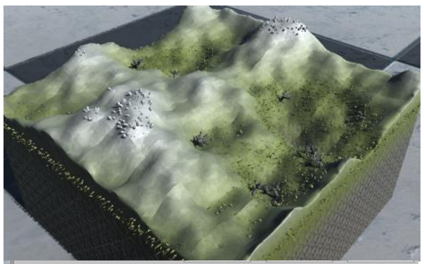


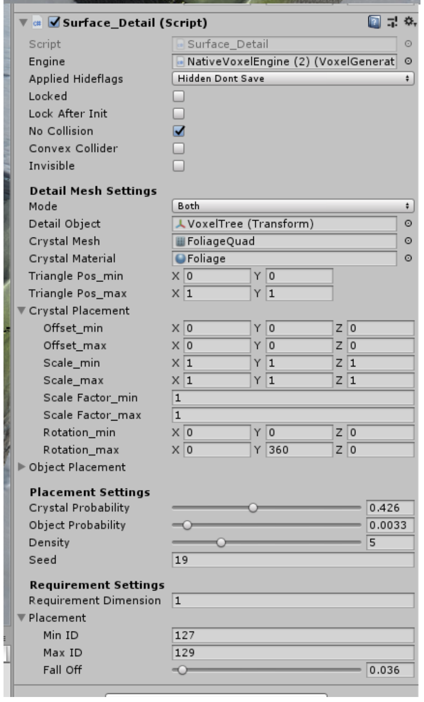
```
Add Tesselation Colors      ✓
LOD Settings:
No Distance Collision       ☐
Far Distance                0
▼ Detail Generator
    Size                    2
    Element 0               Foliage_Grass (Surface_Detail)
    Element 1               IceCrystals (Surface_Detail)
Add Component

▼ Blocks 2
  ▶ MarchingMultiColor_Flat
  ▼ NativeVoxelEngine (2)
    ▶ WorldAlgorithms
      HullGenerator_Marching
      Foliage_Grass
      IceCrystals
  ~Fraktalia Contact Information~
```

Surface modifiers alter the surface generated by a hull generator. Currently only Marching Cubes multi texture supports surface modifiers.

They inherit from Basic Surface Modifier and are applied after the hull generator finished hull generation. A hull generator supporting surface details has a Detail Generator list which can be filled with multiple detail generators.

SURFACE DETAIL





```
▼ ✓ Surface_Detail (Script)
Script                      Surface_Detail
Engine                      NativeVoxelEngine (2) (VoxelGenerat
Applied Hideflags           Hidden Dont Save
Locked                      ☐
Lock After Init             ☐
No Collision                ✓
Convex Collider             ☐
Invisible                   ☐

Detail Mesh Settings
Mode                        Both
Detail Object               VoxelTree (Transform)
Crystal Mesh                FoliageQuad
Crystal Material            Foliage
Triangle Pos_min            X 0      Y 0
Triangle Pos_max            X 1      Y 1
▼ Crystal Placement
    Offset_min              X 0      Y 0      Z 0
    Offset_max              X 0      Y 0      Z 0
    Scale_min               X 1      Y 1      Z 1
    Scale_max               X 1      Y 1      Z 1
    Scale Factor_min        1
    Scale Factor_max        1
    Rotation_min            X 0      Y 0      Z 0
    Rotation_max            X 0      Y 360    Z 0
▶ Object Placement

Placement Settings
Crystal Probability         0.426
Object Probability          0.0033
Density                     5
Seed                        19

Requirement Settings
Requirement Dimension       1
▼ Placement
    Min ID                  127
    Max ID                  129
    Fall Off                0.036
```

Surface Detail is a modifier which places detail objects to the surface. Mostly Details should have no collision especially if the detail is foliage like the sample grass. The surface detail generator has 2 modes which is Crystallic and object mode.

Crystallic is recommended for details with low vertex count such as grass sprites or small pebbles. It uses the Crystal Mesh parameter as shape and is a simple quad and a crystal material.

The crystal placement provides position, rotation and scale settings for variation.

Object placement is recommended for complex objects such as trees and props with more vertices. It places instances of the Detail Object and uses the Object placement for variation.

Crystal and Object have both a Probability. Crystals are also multiplied by the Density as more crystals are possible as the performance impact is lower than with highly detailed objects such as trees.

The Seed is used as initial parameter for randomness in order to keep the result deterministic (else the props would jump around rapidly)

The last settings are the requirements to define when a detail should be placed. You can define which dimension to read and the ID the voxel at the surface should have to be suitable for placement. The sample restricts the placement of grass to a region which contains grass. (127-129)

The falloff defines how sharp the probability is reduced when the surface is outside the desired ID range.

Also the right sample uses Both as mode which places crystals (grass) and objects (trees).

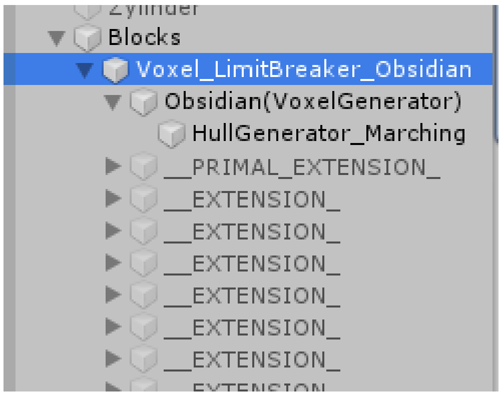
INFINITY EXPANSION (REMOVED IN 1.6.8!)

NOTE: IS DEPRECTATED AND WILL BE REWORKED FROM THE GROUND UP AS IT IS PRONE TO ERRORS!

NOTE: Deprecated version is removed with 1.6.8 as it causes conflicts with new systems.

A complete new version of this system is available as downloadable beta content on Discord.

The infinite expansion system is the biggest addition to the voxel generator as it allows the simulation of an infinite yet fully destructible world. The whole system is in still in the experimental stage and is highly influenced by Minecraft. All necessary implementations like chunk loading, saving and visualization are finished. The missing functions which will come in the future are related to procedural content and world generation including biomes. However before implementing biomes, I want to make sure that the Infinite World system is as perfect and optimized as possible.

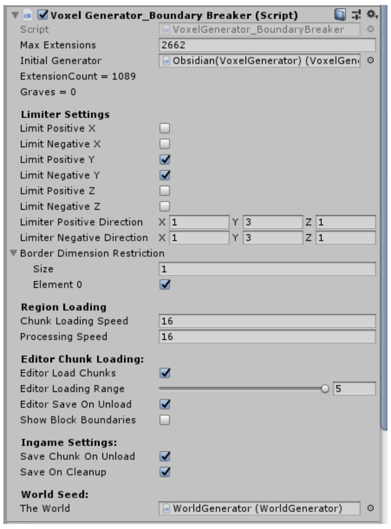


Zylinder
Blocks
Voxel_LimitBreaker_Obsidian
Obsidian(VoxelGenerator)
HullGenerator_Marching
__PRIMAL_EXTENSION_
__EXTENSION_
__EXTENSION_
__EXTENSION_
__EXTENSION_
__EXTENSION_
__EXTENSION_

This system is activated when the voxel generator is attached as child to a GameObject which contains the “VoxelGenerator_BoundaryBreaker” script. When this is the case, the infinity system is applied to the attached VoxelGenerator. When you hit the generate button from the attached VoxelGenerator, the infinity system is activated and a Primal Extension is generated. This extension is then used as template.

The infinite world system subdivides the world into chunks where each chunk has one extension generator assigned. Extension generators are instances from the template VoxelGenerator and cannot be modified in the inspector. The original VoxelGenerator will only act as control tool.

The template will contain attached hull generators and the save system which ideally is set to "**World**" in order to enable saving options.

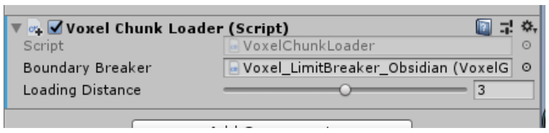


```
Voxel Generator_Boundary Breaker (Script)
Script                        VoxelGenerator_BoundaryBreaker
Max Extensions                2662
Initial Generator             Obsidian(VoxelGenerator) (VoxelGen
ExtensionCount = 1089
Graves = 0

Limiter Settings
Limit Positive X              ☐
Limit Negative X              ☐
Limit Positive Y              ☑
Limit Negative Y              ☑
Limit Positive Z              ☐
Limit Negative Z              ☐
Limiter Positive Direction    X 1    Y 3    Z 1
Limiter Negative Direction    X 1    Y 3    Z 1
Border Dimension Restriction
  Size                        1
  Element 0                   ☑

Region Loading
Chunk Loading Speed           16
Processing Speed              16

Editor Chunk Loading:
Editor Load Chunks            ☑
Editor Loading Range          5
Editor Save On Unload         ☑
Show Block Boundaries         ☐

Ingame Settings:
Save Chunk On Unload          ☑
Save On Cleanup               ☑

World Seed:
The World                     WorldGenerator (WorldGenerator)
```

Most parameters of the BoundaryBreaker are for verification purposes and are assigned automatically. Initial Generator is just a reference to the original VoxelGenerator. Extension Count shows how many extensions are used. Graves shows how many extensions are unloaded and "dead". Unloaded extensions are reused when new chunks are loaded. The MaxExtension value shows how many extensions are allowed and is calculated automatically.

The limiter settings limit the world generation and contain options for each axis. The limiter of positive Y axis is activated in the sample scene. When the player stands on the top of the uppermost voxel block, the visuals will show an endless plane.

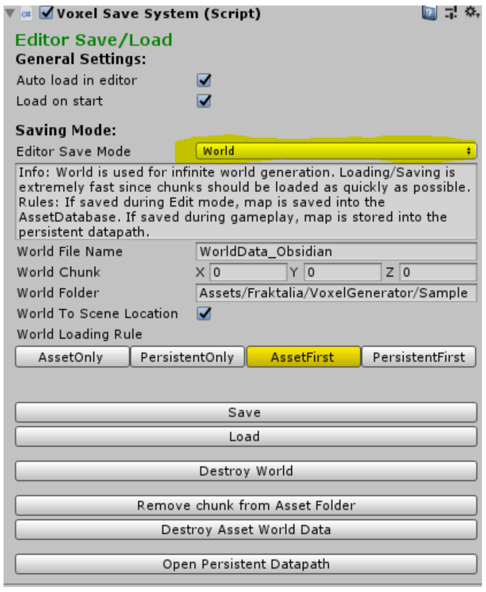
The border dimension restriction describes the behavior of border chunks. The standard setting is to just restrict the solid dimension 0 in order to prevent open hulls at the borders. (First element 0 is true)



```
Voxel Chunk Loader (Script)
Script                VoxelChunkLoader
Boundary Breaker      Voxel_LimitBreaker_Obsidian (VoxelG
Loading Distance      3
```

The next settings are basic editor settings. If Editor Load Chunks is activated, the chunks surrounding the Editor Camera will be loaded according to the loading range.

During gameplay, every object which should interact with the world must have a VoxelChunkLoader script attached. Otherwise the object or mostly the player will fall into the endless depths of Unity. The loading distance simply defines the radius of the chunk loading area. Overall this script has the same functionality used in Minecraft where chunks are only loaded if a chunk loader is nearby (player has one and blocks can have one in modded versions).



```
Voxel Save System (Script)
Editor Save/Load
General Settings:
Auto load in editor       ☑
Load on start             ☑

Saving Mode:
Editor Save Mode          World
Info: World is used for infinite world generation. Loading/Saving is
extremely fast since chunks should be loaded as quickly as possible.
Rules: If saved during Edit mode, map is saved into the
AssetDatabase. If saved during gameplay, map is stored into the
persistent datapath.
World File Name           WorldData_Obsidian
World Chunk               X 0    Y 0    Z 0
World Folder              Assets/Fraktalia/VoxelGenerator/Sample
World To Scene Location   ☑
World Loading Rule
[AssetOnly] [PersistentOnly] [✓ AssetFirst] [PersistentFirst]

[Save]
[Load]
[Destroy World]
[Remove chunk from Asset Folder]
[Destroy Asset World Data]
[Open Persistent Datapath]
```

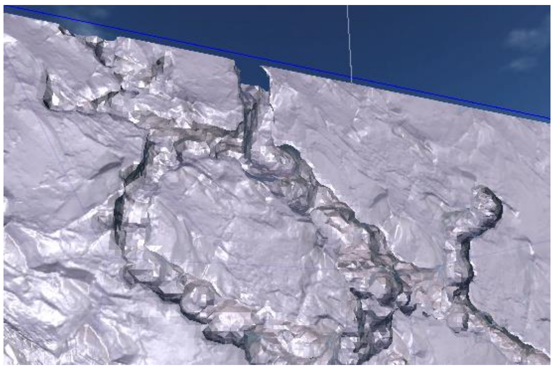
The original VoxelGenerator usually has a save system attached. The only saving method of the infinite world system is "World" and has rules for edit mode and play mode. Chunks modified during edit mode will always be saved into the asset database. The location is the world folder which can be set manually or automatically as scene subfolder (highly recommended). Be careful when you move folders around as nothing is loaded if the location is not correct.

When the infinite world is created, the volume size of the original VoxelGenerator is used as chunk size and is written into the save files. Therefore changing the volume size of an existing world is not possible as it is highly important to keep the size constant.

MOBILE SUPPORT:

VoxelGenerators can be used on any mobile device. However it is recommended to use lower resolution voxel maps and less detailed visual hull constructions since mobile devices are not as powerful as modern gaming computers. The recommended mesh resolution for marching cubes is a width of 4 and cell subdivision of 4. The main limitation is GPU based because the VoxelGenerator is optimized for multi core CPU usage since the GPU already is busy rendering the geometry.

VR/HIGH END SUPPORT:



VoxelGenerators can be used in VR applications. Actually the development of this asset was part of the application for my master thesis which was about sculpting in VR. There it was possible to sculpt the voxel block using nasty power tools like chainsaws and concrete cutters. Back then the voxel engine was much less optimized than it is now.

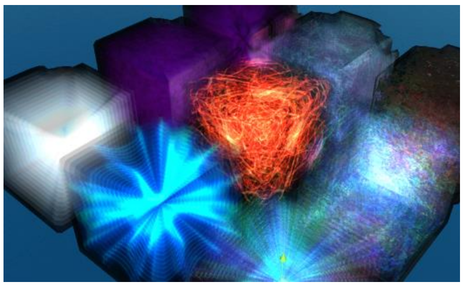
Creating a sculpting tool for VR is done by simply adding a collider to the handle with a voxel modifier and call ModifyAtPos using the collision point(world position) as input parameter.

For high end stuff, a resolution where the width is 16 and sell subdivision is 16 already is fine enough where individual voxels start to blur and become not visible anymore. Going smaller like width of 32 and cell subdivision of 32 is so fine that it enters the realm of scientific visualization where no one cares if it works in real time or not (and where money doesn't matter).

ASSET EXTENSIONS

This section contains information about extensions included in other asset packages. An asset extension combines the Voxel Generator with a different unity asset. The simplest example would be marching cubes using a gemstone material from the Advanced Gem Shader package (costs 15$ or two unhealthy McDonalds meal). Expansion assets are .UnityPackage files which are located in the extension folder of the other asset. The extension content will not work properly if the Voxel Generator is not included in the Project. For example if you import the extension from the Advanced Gem Shader asset, the sample prefabs simply have "Missing Scripts" as the VoxelGenerator doesn't exist.

ADVANCED GEM SHADER



The Advanced Gem Shader package contains a large library of shader to implement gemstones and especially gemstones with unusual properties like inclusions and volumetric interior effects. This allows the creation of sculpable blocks with volumetric interior effects. The extension contains a sample scene containg VoxelGenerator game objects using Gemstone material. Such object has a marching cubes to generate the surface and a UV Writer hull generator for the volumetric interior which is visualized using gemstone complex shader provided by the Advanced Gem Shader package.

FREQUENTLY ASKED QUESTIONS:

Q: Why does a VoxelGenerator has boundaries?

The underlying data structure occupies a defined volume (root size) which is fixed in order to maintain consistency. The LimitBreaker provides the option to make the VoxelGenerator limitless.

Q: Does this asset work on lower Unity versions?

The minimum required version to officially get this asset is 2019.2.11f1. It compatibility may work below 2019 until it hits the hard bottom of 2018.1. Below the asset will not work because the burst compilation does not exist below 2018.1

Q: What are the best settings?

The data structure itself does not care which resolution is used. But for mobile, use Width of 4 and Cell subdivision of 4 or lower. Target depth lower than 5.

Higher width means more vertices per GameObject which increases GPU efficiency but increases CPU load. Higher subdivision count but lower width means more GPU load but less CPU demand.

Increasing the Subdivision Power of the data structure reduce lookup time but at the cost of more memory.

Q: Is it possible to implement physical interaction such as floating voxels falling down?

Physical behavior in voxel engines is a very complex part and requires ton of work. Implementing this would actually double the price for this asset. Getting this implemented would require at least one more year.

Q: I need accurate voxel geometry for scientific visualization?

If you need the engine for medical or scientific stuff, it is always recommended to use Texture to voxel converter as it is the most accurate solution. Mesh to voxel converter is not accurate enough in my opinion.

Q: How do I increase the resolution?

The resolution is dynamically and the data itself can get as fine as desired by increasing the depth value of the modifier.

When increasing the depth, you also have to increase the precision of the hull generator by increasing the Width and Cell Subdivision. Finer details will be missed if the hull generator is too coarse.

For example Marching Cubes:

Generator: Subdivision Power = 2
Modifier: Depth = 7
Hull Generator: Width = 8, Cell Subdivision = 8

Generator: Subdivision Power = 2
Modifier: Depth = 8
Hull Generator: Width = 16, Cell Subdivision = 8

Generator: Subdivision Power = 2
Modifier: Depth = 9
Hull Generator: Width = 16, Cell Subdivision = 16

High Resolution:

Generator: Subdivision Power = 2
Modifier: Depth = 10
Hull Generator: Width = 32, Cell Subdivision = 16

Whenever the depth increases, you also have to double either width or cell subdivision!

If the hull generator is too fine, the result is a blocky appearance.

AUTHOR NOTES:

This voxel engine uses no external stuff which is the connected with Unity3D and therefore has no “Bridges” or other APIs as it is common in other voxel engines. Also this voxel engine was developed with full control over the engine itself in mind so every functionality can be modified. The performance is only possible through the inclusion of the Unity Job system and Burst compilation system. Therefore it is highly recommended to have those installed.

I am glad that I was allowed to work with voxels as part of my master thesis. For all those who have to write one, always choose a topic you are really interested in.

LAST NOTES:

If you have any questions, suggestions, bug reporting don’t hesitate to contact me. If you are going to sell a game which uses this asset, inform me because I may buy your game and play it ☺

Contact Information:

E-Mail: m.hartl@fraktalia.org

Homepage: http://fraktalia.org/