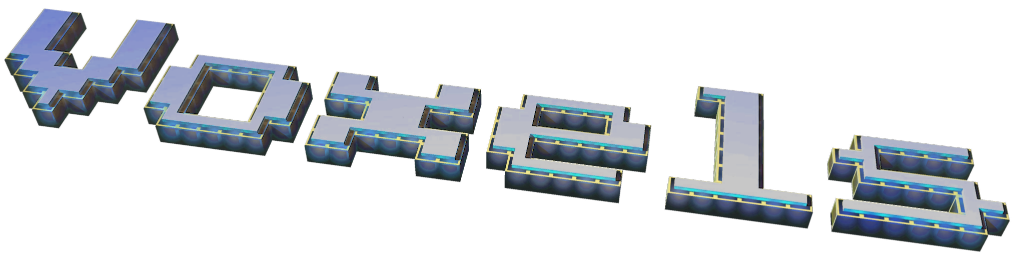


for

unity

Rasterizer

Version 1.1

November 11, 2017

Table of Contents

About

“Voxels for Unity” is a plug-in extension for the Unity engine and development environment (www.Unity3D.com). It utilizes the graphics hardware to rasterize renderer objects into a three-dimensional grid instead of a 2D planar surface, which is done to present them on a screen or to compute effects most of all. As opposed to other existing solutions it is able to bake a lot of shading and lighting conditions into the target color of every grid cell at the moment of the processing. Such cells are called voxels, which is a composition of **vo**lume and pi**xels**. You can use them to display grid aligned (e.g. blocky) models or particles, which are generated from existing assets. Those particles can be used to create visual effects in combination with the original objects or you transfer the data to an existing voxel engine respectively a rendering framework.

The extension is executable in the editor and at runtime and therefore provides an inspector and a programming interface, which can be accessed from other scripts. And the complete package includes two sample processing classes, which export the collect information to popular file formats, and three, which are converting the gathered data to visible game objects. One creates a hierarchy of meshes while another passes the grid elements to a particle system. The last stores color data to a 2D texture and can be used to highly optimize the meshes, which are created by the first one. You are able to adapt the existing examples or write your own processor scripts using the derivation from a base class. However the provided processors are already capable of creating assets that are applicable in the production.

Unfortunately not all rendering features can be baked into the final voxel colors because the culling system of Unity hides them in the scanning process. Known issues are dynamic shadow maps and particles. The first one can still be applied to converted voxel objects but cannot be included in the color of the object itself. We are still looking for workarounds or alternative rendering methods to be provide the support in future releases.

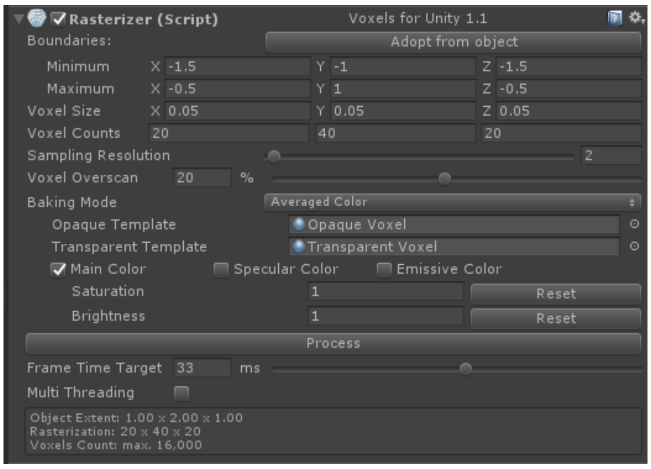
Using the editor

Voxel Rasterizer

After you have imported the package to your Unity project, you can attach the Rasterizer plug-in to any game object, even multiple times. In the hierarchy any object under the modified one or the modified one itself should contain a visual component like a mesh renderer or a terrain. Otherwise nothing could be sampled to the volume grid.

You get an inspector for every converter instance. If you turn the switch off, it will not automatically being executed at the start of the scene.

Boundaries: *Minimum* and *Maximum* are the limiting 3D vectors of the volume in the scene, which is being scanned for voxels. By pressing on the *Adopt from object* button both values are measured by combining the current bounds of all attached objects.



```
✓ Rasterizer (Script)              Voxels for Unity 1.1
Boundaries:                        [Adopt from object]
   Minimum        X -1.5       Y -1        Z -1.5
   Maximum        X -0.5       Y 1         Z -0.5
Voxel Size        X 0.05       Y 0.05      Z 0.05
Voxel Counts      20           40          20
Sampling Resolution                                2
Voxel Overscan    20    %
Baking Mode                        Averaged Color
   Opaque Template                 Opaque Voxel
   Transparent Template            Transparent Voxel
  ✓ Main Color    ☐ Specular Color    ☐ Emissive Color
     Saturation              1           [Reset]
     Brightness              1           [Reset]
[Process]
Frame Time Target 33    ms
Multi Threading   ☐
Object Extent: 1.00 x 2.00 x 1.00
Rasterization: 20 x 40 x 20
Voxels Count: max. 16,000
```

Rasterizer inspector with all properties

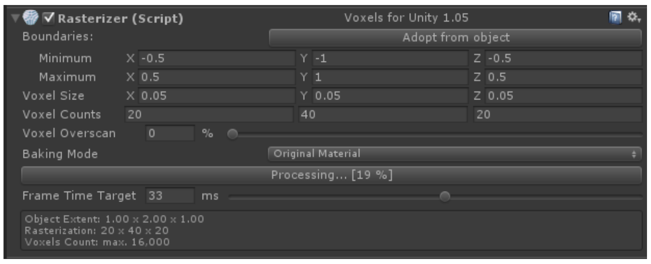
Voxel Size: This is the extent of one grid cell in the space of the scene.

Voxel Counts: It describes the number of columns, rows and slices the volume grid contains.

Voxel Size and *Voxel Counts* are affecting one another, where the counts are the outcome of the total volume and the single cell size.

Sampling Resolution: Super-sampling can be used to scan the source object(s) in higher resolution and generate finer color results by combining multiple neighbor samples into one target voxel. The value here is a factor for each dimension, so 2x needs to render eight times the amount of pixels compared to factor 1 for example.

Voxel Overscan: That percentage value increases the depth of currently scanned layer, which can be helpful if the graphics hardware produces errors in rendering so that some voxels will be missing. It happens for terrains if the details becomes to small because the resolution is high.



```
✓ Rasterizer (Script)              Voxels for Unity 1.05
Boundaries:                        [Adopt from object]
   Minimum        X -0.5       Y -1        Z -0.5
   Maximum        X 0.5        Y 1         Z 0.5
Voxel Size        X 0.05       Y 0.05      Z 0.05
Voxel Counts      20           40          20
Voxel Overscan    0     %
Baking Mode                        Original Material
[Processing... [19 %]]
Frame Time Target 33    ms
Object Extent: 1.00 x 2.00 x 1.00
Rasterization: 20 x 40 x 20
Voxels Count: max. 16,000
```

Voxel Rasterizer shows the current progress

Baking Mode: The shaded colors of a source object can be interpreted in various ways. By using *Original Material* the color is not baked at all. The source material, which is located at a cell, is transferred directly to its voxel. On the contrary *Averaged Color*, *Darkest Color* and *Brightest Color* burn the shaded result color to a material, which is used in association with renderer of the single voxel. *Most Frequent Color* works similar but does not simply averages all detected cell colors or builds the mini- or maximum of them but filters by the amount of their occurrences.

Opaque Template: This option only appears, if you have selected a color baking mode. You can specify a default material, which is used, if the alpha value of the grid cell is 1. This means that the voxel is originally not a transparent one. Without a selection a standard legacy material is created.

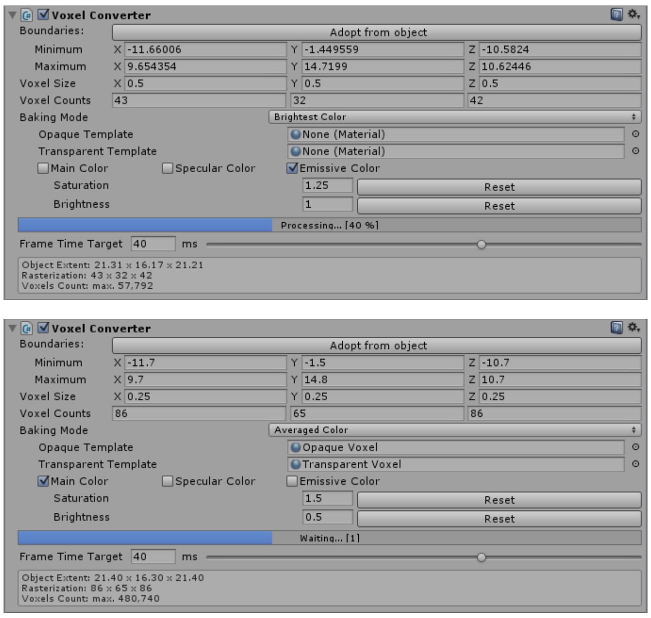
Transparent Template: Comparable to Opaque Template it allows the choice of a material, which is used for semi-transparent cells, which means that the alpha value is between 0 and 1.

Main Color / Specular Color / Emissive Color: Here you can select, which component of the applied material is modulate by the result color of the baking.

Saturation: This factor allows the manipulation of the result color by making it more or less colorful.

Brightness: A material can be made darker or lighter by modifying it.

Process / Processing… / Waiting…: That button starts or stops the conversion of the selected object from the editor. It also shows the current state: *Processing…* signals that the object is currently being processed and shows the progress, whereas *Waiting…* represents that another object is currently in conversion and indicates the position in the queue.



```
Voxel Converter
Boundaries:            Adopt from object
  Minimum      X -11.66006    Y -1.449559   Z -10.5824
  Maximum      X 9.654354     Y 14.7199     Z 10.62446
Voxel Size     X 0.5          Y 0.5         Z 0.5
Voxel Counts   43             32            42
Baking Mode                   Brightest Color
  Opaque Template             None (Material)
  Transparent Template        None (Material)
  ☐ Main Color   ☐ Specular Color   ☑ Emissive Color
    Saturation                1.25   Reset
    Brightness                1      Reset
Processing... [40 %]
Frame Time Target 40 ms
Object Extent: 21.31 x 16.17 x 21.21
Rasterization: 43 x 32 x 42
Voxels Count: max. 57,792
```

```
Voxel Converter
Boundaries:            Adopt from object
  Minimum      X -11.7        Y -1.5        Z -10.7
  Maximum      X 9.7          Y 14.8        Z 10.7
Voxel Size     X 0.25         Y 0.25        Z 0.25
Voxel Counts   86             65            86
Baking Mode                   Averaged Color
  Opaque Template             Opaque Voxel
  Transparent Template        Transparent Voxel
  ☑ Main Color   ☐ Specular Color   ☐ Emissive Color
    Saturation                1.5    Reset
    Brightness                0.5    Reset
Waiting... [1]
Frame Time Target 40 ms
Object Extent: 21.40 x 16.30 x 21.40
Rasterization: 86 x 65 x 86
Voxels Count: max. 480,740
```

Rasterizer inspector at runtime while processing

At runtime the button does not appear and will be replaced by a progress bar during the conversion.

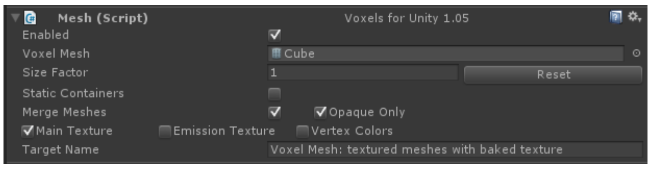
Frame Time Target: This is a global setting, which affects all converter instances and adjusts the aimed frame rate. Because the processing cannot be done in real-time it happens in cooperation with the other work load of Unity and your scene. The approximate value allows you to specify to amount of time, which should be used for total processing between two frames, but it can be more in complex scenarios.

At last there is a box with some information about the volume size, the grid resolution and maximum count of resulting voxels, if all grid cells would get filled. You can make use of the data to adapt setting for a target specification.

Multi Threading: Enabling that flag can shorten processing time by using more CPU cores. If you encounter issues, you may need to disable it.

Voxel Mesh

One of the three sample processor scripts constructs a hierarchy of meshes from the data, which has been collected by the converter instance. That instance transfers gathered voxels to active processor components, which are attached at the same game object like the converter.



Mesh (Script) Voxels for Unity 1.05

Enabled	✓
Voxel Mesh	Cube
Size Factor	1 Reset
Static Containers	☐
Merge Meshes	✓ ✓ Opaque Only
✓ Main Texture	☐ Emission Texture ☐ Vertex Colors
Target Name	Voxel Mesh: textured meshes with baked texture

Mesh inspector

Enabled: The flag activates the processing of incoming data.

Voxel Mesh: You have to select a template, which is used to represent a voxel. Most often a cube might be the general choice. Nevertheless you are able to choose any imported mesh asset like a sphere. But beware of the vertex / polygon count: Thousands of voxels multiplied by thousands of triangles turn into millions of polygons that need to be processed per frame.

Size Factor: That value increases or decreases the extent of one cell mesh, which is normalized to fit the voxel size declared in the converter.

Static Containers: It is a flag, which is copied to new game object that will be created during the processing. If you want to animated single voxels, you should turn it off.

Merge Meshes: Multiple voxels that share the same material will be merged into larger meshes. So single cells cannot be moved by individual game object transform components any longer but shaders can be used to do so.

Opaque Only: Whenever *Merge Meshes* is enabled that flag skips the combination of voxels, which have got semi-transparent alpha values. This is important if you want to retain the correct blending order from camera view with the standard Unity solution.

Main Texture / Emissive Texture: Either the script builds a Texture2D object or uses the attached component, if its *Voxel Mesh Usage* flag is active, and replaces the corresponding texture on the materials, which are being applied to the target objects. That way different colors can be used without the need for various material instances, which will result in much better performance for merged meshes.

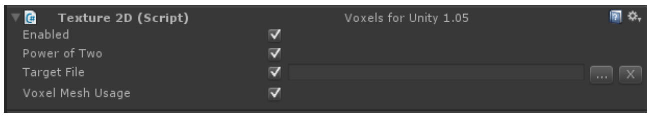
Vertex Colors: Comparable in the effect of *Main Texture / Emissive Texture* the collected voxel colors are stored to the vertices of the result mesh(es). But you need to set a template material with a shader, which can handle them.

Target Name: The main container of the target object is named after the string, which is given here.

Voxel Texture 2D

Mentioned above this class stores all unique colors, which had been sampled before, into a two-dimensional texture.

Enabled: Equally to Voxel Mesh it activates or deactivates the processing of the component.



Texture 2D (Script)	Voxels for Unity 1.05
Enabled	✓
Power of Two	✓
Target File	✓
Voxel Mesh Usage	✓

Texture 2D inspector

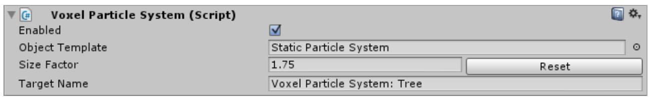
Power of Two: The resulting Unity texture will have a 2^n width and height.

Target File: Specify a path, if you want to store the content of the texture into an image file. That way you are to able to edit it afterwards.

Target Name: The recently instantiated template gets the stated name.

Voxel Particle System

The next sample class builds a system of particles using the incoming data. You are able to combine it with other processors, so that the same data is processed into different objects.



Voxel Particle System (Script)	
Enabled	✓
Object Template	Static Particle System
Size Factor	1.75 Reset
Target Name	Voxel Particle System: Tree

Voxel Particle System inspector

Enabled: Equally to Voxel Mesh it activates or deactivates the processing of the component.

Object Template: That needs to be game object instance with a predefined particle system, which is instantiated as a new one for the target.

Size Factor: It is the factor, which is multiplied with the standard particle size of a voxel cell.

Target Name: The recently instantiated template gets the stated name.

Magica Voxel Exporter

In contrast to the previous processors that one does not create data inside of Unity or the app. Instead a file in the format of MagicaVoxel, which is a popular voxel editor, with the specified path name is written. You can import the data, which had been generated by the Rasterizer, into the program afterwards.

Because of the restrictions of the format only 256x256x256 maximum voxels can be stored and the color amount is limited to 254. So the exporter has to reduce the collected materials before writing the results.

Point Cloud Exporter

The last sample class is comparable to the previous one but without the restrictions. All voxels are stored as a point cloud to a PLY file. Although it is called Polygon File Format no indices for connecting vertices to define polygons are stored but regardless a lot of point cloud viewers and processing applications support that file format.

Using the programming interface

`namespace Voxels`

All structures, enumerations and classes of the extension are collected in that main namespace.

`struct`	`Vector`
	`int x`
	`int y`
	`int z`
		A vector with a coordinate (column, line, slice) in voxel space

`enum`	`BakingOperation`
	`Undefined`
	`OriginalMaterial`
	`AveragedColor`
	`DarkestColor`
	`BrightestColor`
	`MostFrequentColor`
		Method to interpret sampled colors per volume cell
		• Use material of the source object(s)
		• Apply average, darkest, brightest or most frequent sampled color of a cell to a derived material

`class Voxels.Rasterizer.Settings`

This class is used to deliver scanning and properties to the engine.

`UnityEngine.Bounds`	`bounds`	Center and size of volume to scan
`UnityEngine.Vector3`	`voxelSize`	Size of a volume cell in scene space
`Voxels.Vector`	`volumeResolution`	Number of columns, rows and slices in the scan volume
`float`	`voxelOverscan`	Factor to increase depth of a scanning step
`int`	`samplingResolution`	[not supported yet]
`Voxels.BakingOperation`	`bakingOperationMode`	Type of method to use for casting scanned material or color
`UnityEngine.Material`	`opaqueTemplate`	A material, which is used to apply non-transparent cell colors to
`UnityEngine.Material`	`transparentTemplate`	A material, which is used to apply semi-transparent cell colors to
`bool`	`mainColorModulation`	Flag to modulate main color of the instantiated material by the cell content
`bool`	`specularColorModulation`	Flag to modulate specular color of the instantiated material by the cell content

`bool`	**`emissiveColorModulation`**	Flag to modulate emissive color of the instantiated material by the cell content
`float`	**`saturationFactor`**	Modifies chroma of the cell color
`float`	**`brightnessFactor`**	Modifies luminosity of the cell color
`float`	**`budgetTime`**	Frame time target for that specific conversion (<0 for standard value)
`bool`	**`multiThreading`**	Use multiple CPU threads for the processing of the scanned data

There are no other methods beside the two constructors, which are quite similar. One is for setting the size per voxel, the other to use a target resolution for the complete volume.

```
Settings(
      UnityEngine.Bounds bounds,
      UnityEngine.Vector3 voxelSize,
      float voxelOverscan,
      int samplingResolution,
      Voxels.BakingOperation bakingOperationMode,
      UnityEngine.Material opaqueTemplate,
      UnityEngine.Material tranparentTemplate,
      bool mainColorModulation,
      bool specularColorModulation,
      bool emissiveColorModulation,
      float saturationFactor,
      float brightnessFactor,
      float budgetTime = -1,
      bool multiThreading = true
      )
```

Initialize values including voxel size and set volume resolution to zero, which leads to an automatic calculation.

```
Settings(
      UnityEngine.Bounds bounds,
      Voxels.Vector volumeResolution,
      float voxelOverscan,
      int samplingResolution,
      Voxels.BakingOperation bakingOperationMode,
      UnityEngine.Material opaqueTemplate,
      UnityEngine.Material tranparentTemplate,
      bool mainColorModulation,
      bool specularColorModulation,
      bool emissiveColorModulation,
      float saturationFactor,
      float brightnessFactor,
      float budgetTime = -1,
      bool multiThreading = true
      )
```

Initialize values including volume resolution and set voxel size to zero, which leads to an automatic calculation.

Parameters:

See member variables

class Voxels.Rasterizer

That is the instance for a game object to equip it with serializable conversion settings.

`UnityEngine.Vector3`	**`minimumBound`**	Smaller edge of the volume to scan for voxels
`UnityEngine.Vector3`	**`maximumBound`**	Larger edge of the volume to scan for voxels

`UnityEngine.Vector3`	**`voxelSize`**	Extent of a voxel cell
`Float`	**`voxelOverscan`**	Increasing depth factor for a scan slice
`int`	**`resolutionWidth`**	Number of columns in the volume
`int`	**`resolutionHeight`**	Number of rows in the volume
`int`	**`resolutionDepth`**	Number of slices in the volume
`Voxels.BakingOperation`	**`bakingOperationMode`**	Method to process scanned materials / colors
`UnityEngine.Material`	**`opaqueTemplate`**	Material to use for non-transparent cells
`UnityEngine.Material`	**`transparentTemplate`**	Material to use for cells with alpha value smaller then one
`bool`	**`mainColorModulation`**	Flag to multiply main color of the template / default material with scanned cell one
`bool`	**`specularColorModulation`**	Flag to multiply specular color of the template / default material with scanned cell one
`bool`	**`emissiveColorModulation`**	Flag to multiply emissive color of the template / default material with scanned cell one
`float`	**`saturationFactor`**	Value makes cell colors more or less colorful
`float`	**`brightnessFactor`**	Value makes cell colors brighter or darker
`float`	**`budgetTime`**	Target time to limit total computation per frame to

```
bool Process(
    )
```

Transfer the game object for voxel conversion to crafting singleton using the applied settings and attached processors.

Return Value:

`true`, if processing initialization could be successfully completed.
`false`, if a failure occurred.

```
bool Stop(
      )
```

End the conversion of the game object.

Return Value:

`true`, if game object has been successfully removed from processing.
`false`, if object is not being processed.

```
float GetProgress(
      )
```

Read current conversion progress.

Return Value:

`> 1`, if object is waiting to be processed.
`0 … 1`, if object is currently being processed.
`-1`, if object is not included in the processing queue.

```
void RecomputeBounds(
      )
```

Adopt boundaries of the volume, which is scanned for voxels, from the renderers of the game object and its children.

```
bool SetVoxelSize(
      float width
      float height,
      float depth
      )
```

Change the size of a voxel cell in scene space. Calling this function invalids volume resolution values.

Return Value:

`true`, if arguments are valid.

Parameters:

width	Width of a voxel
height	Height of a voxel
depth	Depth of a voxel

```
bool SetVolumeResolution(
      int width,
      int height,
      int depth
      )
```

Change the numbers of cell columns, rows and slices of the whole volume. Calling this function invalids voxel size values.

Return Value:

`true`, if arguments are valid.

Parameters:

`width`	Cell columns
`height`	Cell rows
`depth`	Cell slices

```
bool IsVolumeResolutionDefined(
      )
```

Check if volume resolution is set. Otherwise the voxel size is used for generating cells.

Return Value:

`true`, if volume resolution is used to define number of scanning cells.
`false`, if voxel size is used to calculate number of scanning cells.

`abstract class Voxels.Processor`

This class is used as a base interface to implement functions to handle the voxel data, which had been collected by the engine right before. You have to derive your own classes from it like `Voxels.Mesh`, `Voxels.Texture2D` and `Voxels.ParticleSystem` are.

```
delegate void Informer(
      UnityEngine.Object[] objects,
      object parameter = null
      )
```

This is the definition of a callback function to inform the script, which has started the rasterizer, about the creation of Unity objects in a processor script.

Parameters:

`objects`	Array of object, which have been created by the processor
`parameter`	Optional parameter, which is passed through from the executing script to informer function

`bool`	**`process`**	This flag indicates, if the processor is being executed by the engine.

```
float Build(
      Voxels.Storage voxels,
      UnityEngine.Bounds bounds,
      Voxels.Processor.Informer informer = null,
      object parameter = null
      )
```

The method is repeatedly called to process incoming data until it signals the completion.

Return Value:

`0 … <1` is the current progress. Processing has not been finished.
`>=1` signals the end of processing.

Parameters:

voxels	Instance containing collected voxel data
bounds	Boundaries, which have been used to scan the source object(s) and therefore define the volume of the data

```
int GetPriority(
      )
```

Processors for a rasterizer with a lower value are being handled before those with a higher one.

Return Value:

The processor priority to determine the processing sequence

class Voxels.Processor.MaterialReducer

That inner processor class is a helper to decrease the amount of various materials. It combines materials with neighbor colors into single ones until the specified limit is reached.

```
MaterialReducer(
      UnityEngine.MonoBehaviour monoBehaviour,
      System.Collections.Generic.List<UnityEngine.Material> inputMaterials,
      int countLimit
      )
```

Constructor expecting a runtime instance for co-processing, a list of source materials and the maximum amount of materials to return.

Parameters:

monoBehaviour	Mono-behaviour of the calling instance
inputMaterials	List of source materials to reduce
countLimit	Maximum number of materials to generate

```
float GetProgress(
      )
```

Obtain current processing progress value

Return Value:

Interpolation value between 0 and 1

```
bool GetResult(
      out UnityEngine.Material[] outputMaterials,
      out System.Collections.Generic.Dictionary<int, int> assignments
      )
```

Receive array of output materials and an assignment table between input and output indices.

Return Value:

`true`, if processing has been finished and results are valid

Parameters:

`outputMaterials`	Array of merged materials
`assignments`	Hash table containing indices of the source materials assigned to target ones

`class Voxels.Rasterizer.Engine`

For scanning and transferring the result data to processor classes a singleton of this type is created. You cannot instantiate it from outside but that is not necessary because the public methods are static and therefore no object is required to call them.

`bool`	**`Processing`**	Read-only property returns if the engine is currently at work
`UnityEngine.Bounds`	**`ObjectBounds`**	Read-only property returns center and size of source object, which is currently being processed

```
bool Process(
      Voxels.Rasterizer converter,
      Voxels.Rasterizer.Settings settings,
      Voxels.Processor[] processors,
      Voxels.Processor.Informer informer = null,
      object parameter = null
      )
```

Append given voxel converter component altogether with settings and processor instance to waiting queue and start conversion, if previous entries have been completed or aborted.

Return Value:

`true`, if converter could be successfully added to the list.

Parameters:

`converter`	Component of the source object, which should be converted
`settings`	Conversion settings
`processors`	Array of processor class instances, which handle the result data
`informer`	Callback to be executed after a processor has finished generating a result
`parameter`	Free defined parameter for the callback function

```
bool Process(
      UnityEngine.GameObject source,
      Voxels.Rasterizer.Settings settings,
      Voxels.Processor[] processors,
      Voxels.Processor.Informer informer = null,
      object parameter = null
      )
```

Append given game object altogether with settings and processor instance to waiting queue and start conversion, if previous entries have been completed or aborted. There is no converter instance used for enhanced access.

Return Value:

`true`, if converter could be successfully added to the list.

Parameters:

`source`	Source object, which should be converted
`settings`	Conversion settings
`processors`	Array of processor class instances, which handle the result data
`informer`	Callback to be executed after a processor has finished generating a result
`parameter`	Free defined parameter for the callback function

```
bool Stop(
      Voxels.Rasterizer converter
      )
```

Remove given voxel converter from processing queue.

Return Value:

`true`, if converter could be found in the waiting list or processing has been stopped.

Parameters:

`converter`	Component of the source object, whose conversion has to be canceled

```
float GetProgress(
      Voxels.Rasterizer converter
      )
```

Retrieve current conversion progress for given converter instance.

Return Value:

`> 1`, if object is waiting to be processed.
`0 … 1`, if object is currently being processed.
`-1`, if object is not included in the processing queue.

Parameters:

`converter`	Component of the source object, whose state is requested

```
bool IsActive(
      )
```

Method answers if the singleton is currently alive.

Return Value:

`true`, if engine instance exists.

```
UnityEngine.Bounds ComputeBounds(
      UnityEngine.GameObject gameObject
      )
```

Compute the bounding box of the given object and all its children.

Return Value:

Center and half size (Unity standard)

Parameters:

`gameObject`	Top object to be measured

class Voxels.Storage

The result data is filed and can queried using this class.

`int`	**`Width`**	Read-only property returns number of cell columns
`int`	**`Height`**	Read-only property returns number of cell rows
`int`	**`Depth`**	Read-only property returns number of cell slices
`int`	**`FacesCount`**	Read-only property returns number of faces per cell (fixed to 6)
`int`	**`Count`**	Read-only property returns total number of stored cells

```
bool SetIndex(
      int x,
      int y,
      int z,
      UnityEngine.CubemapFace face,
      int index,
      int sx = 0,
      int sy = 0,
      int sz = 0
      )
```

Store the given (material) index for the given direction to the given coordinate.

Return Value:

`true`, if cell content could be written.

Parameters:

x, y, z	Coordinate of the cell in volume space
face	Direction of the cell cube plane to store index for
index	Value to write
sx, sy, sz	Sub-sampling coordinate in cell space

```
bool SetColor(
      int x,
      int y,
      int z,
      UnityEngine.CubemapFace face,
      UnityEngine.Color color,
      int sx = 0,
      int sy = 0,
      int sz = 0
      )
```

Store the given color value for the given direction to the given coordinate.

Return Value:

`true`, if cell content could be written.

Parameters:

x, y, z	Coordinate of the cell in volume space
face	Direction of the cell cube plane to store index for
color	Value to write
sx, sy, sz	Sub-sampling coordinate in cell space

```
bool AddColor(
      int x,
      int y,
      int z,
      UnityEngine.CubemapFace face,
      UnityEngine.Color color,
      int sx = 0,
      int sy = 0,
      int sz = 0,
      Voxels.BakingOperation operationMode = Voxels.BakingOperation.Undefined
      )
```

Include the given color value for the given direction to the given coordinate using the given or the internally stored baking mode.

Return Value:

`true`, if cell content could be written.

Parameters:

`x, y, z`	Coordinate of the cell in volume space
`face`	Direction of the cell cube plane to store index for
`color`	Value to account
`sx, sy, sz`	Sub-sampling coordinate in cell space
`operationMode`	Baking operation to use for applying input against existing color

```
UnityEngine.Material GetMaterial(
      out UnityEngine.Color color,
      int x,
      int y,
      int z,
      Voxels.BakingOperation operationMode = Voxels.BakingOperation.Undefined,
      int normalized = true
      )
```

Return the source material of the cell or a newly instantiated one with the optionally modulated color of the voxel.

Return Value:

An existing or a new material instance

Parameters:

`color`	Cell color, which is being applied to template material or retrieved from source one
`x, y, z`	Coordinate of the cell in volume space
`operationMode`	Baking operation to use for applying input against existing color
`normalized`	Flag to remove pre-multiplication of alpha from red, green and blue components

```
Voxels.Storage.Iterator GetIterator(
      )
```

Create and return a new iterator instance to access cells with content only.

Return Value:

New iterator instance

class Voxels.Storage.Iterator

To access collected voxels in a fast way the iterator class is perfect because it returns the material for filled cells only.

int	**Number**	Read-only number of the cell, which is being accessed next

```
UnityEngine.Material GetNextMaterial(
      [out UnityEngine.Color color,]
      out int x,
      out int y,
      out int z,
      Voxels.BakingOperation operationMode = Voxels.BakingOperation.Undefined,
      int normalized = true
      )
```

Return the source material of the next iterated cell or a newly instantiated one with the optionally modulated color of the voxel.

Return Value:

An existing or a new material instance

Parameters:

color	Cell color, which is being applied to template material or retrieved from source one
x, y, z	Coordinate of the cell in volume space
operationMode	Baking operation to use for applying input against existing color
normalized	Flag to remove pre-multiplication of alpha from red, green and blue components

```
UnityEngine.Color GetNextColor(
      out int x,
      out int y,
      out int z,
      Voxels.BakingOperation operationMode = Voxels.BakingOperation.Undefined,
      int normalized = true
      )
```

Return the color of the next iterated cell, which has been retrieved from scan results.

Return Value:

Sampled and processed color of the cell

Parameters:

x, y, z	Coordinate of the cell in volume space
operationMode	Baking operation to use for applying input against existing color
normalized	Flag to remove pre-multiplication of alpha from red, green and blue components

class Voxels.HSVColor

It is kind of a helper class to apply color modifications and stores color values as hue, saturation and value components.

float	**h**	Hue component
float	**s**	Saturation component
float	**v**	Value component
float	**a**	Alpha value (inverted transparency)

```
HSVColor(
      float h,
      float s,
      float v,
      [float a]
      )
```

Common constructor to initialize member variables

Parameters:

See member variables

```
HSVColor(
      UnityEngine.Color col
      )
```

Initialize member variables using a RGBA color value as input.

Parameters:

See member variables

```
UnityEngine.Color ToColor(
      )
```

Build a RGBA color value from instance.

Return Value:

Converted color result

```
Voxels.HSVColor Lerp(
      Voxels.HSVColor a,
      Voxels.HSVColor b,
      float t
      )
```

Do a linear interpolation between two HSV color values.

Return Value:

Blending result value

Parameters:

a, b	Source colors
t	Interpolation factor; should be between 0 and 1