

Home



Quick Start

[Basic Principles](#)
[Quick Start Guide](#)

Menus

[Brush](#)
Block Types:
 [Land Blocks](#)
 [Object Blocks](#)
 [Grass Blocks](#)
[Mode and Ranges](#)
[General Settings](#)
[Horizon Mesh](#)

Terrain Generate

[Generate Menu](#)
[Generators](#)

Scripting

[Serializing voxeland data](#)
[Editing terrain in playmode with scripts](#)

Frequently Asked Questions

[General FAQ](#)

Basic Principles

Despite of the non-cubical look, Voxeland terrain internally made of cubes. Imagine the Minecraft terrain where all of the surfaces are welded, smoothed and subdivided. Each of the cubes (named **blocks**) is one unit in size by the default but Voxeland could be scaled up or down to change the size of the cubes.

Chunks

All the terrain is split into smaller meshes called **chunks**. By the default the chunk size is 30*30 blocks in X and Z axis. The chunk size is not limited, chunks are stored planar: each chunk has only X and Z coordinate.

When the terrain is changed only those chunks are recalculated that were affected by change. When the terrain is [built](#) from scratch it is built chunk-by-chunk.

Areas

All of the "infinite" Voxeland terrain is split into **areas** - the squares of 512*512 size by the default. Each area is [generated](#) independently. Just like Voxeland is built chunk-by-chunk, it's generated area-by-area.

The Voxeland chunk is not wired to the area size in any way - generally chunk reads an information from 1 area, but it can be 2 if it is located on the seam between areas, or even 4 if it is on the corner.

Unlike chunk, it's not possible to "see" the area: it's just an internal entity, that determines the way data generated and organized.

Data Structure

Each area stores the following information:

- Land Blocks: Consider it as a losless-compressed 3-dimensional array of bytes (or a 3-dimensional image). Each byte (or pixel) represent a land type number. 255 (or the number specified in Data.emptyBlock) is for an empty block. Voxeland data contains no information about individual block texture, specular or other values. It just stores a single byte, that could be a reference to the land texture type (number in array) in Voxeland object.
- Grass Blocks: a separate data for the grass types. Similar to the land blocks data, except that it is a 2D "array" instead of 3D. It is spread in X and Z axis. Each byte determines the grass type at the current position.
- Object Blocks: a dictionary of the objects used in area. Despite it's not an array, it stores only the reference byte to the object type in Voxeland object itself, similarly to the land blocks and grass blocks.

By the default Voxeland data is saved with the scene, but you can save it to the separate .asset file in [Settings](#).

"Build" and "Generate" difference

When using infinite terrain each chunks is undergoing a number of calculations:

- In the beginning the chunk has nothing at all except coordinates and some rules on terrain generate defined by Voxeland or MapMagic generators.

First step is calculating it's voxel data. Consider voxel data a 3D array of bytes - each byte sets the number of block type used in a cell. This stage is called ***generate***.

- When you get closer to chunk it's mesh is calculated, smoothed, painted, etc using the block data. This stage is called ***build mesh***.

For the infinite terrain none of the chunks is "generated" or "built". They are generating and building only when camera approaches them.

If you change a chunk in an infinite terrain it will be ***pinned***: it's block data will be kept, and it will not be "re-generated" no matter how far you travel from it. But it does not mean that it's mesh will be persistent, it's mesh still could be removed and "build" from block data dynamically.

Texture Arrays

By the default Voxeland uses the texture arrays shader instead of the standard textures. This allows to blend 24 block types, and do it a bit faster then blending the standard textures.

However, this approach has some drawbacks: Texture2DArray is not just an array of separate textures, it's a single object that could not be edited in Photoshop or other graphics software. Texture arrays are generated in Voxeland using the standard source textures. So, after the editing the texture it should be reloaded to the texture array manually by pressing the **Load** button. It will not be updated automatically the way the standard textures do. Moreover, texture arrays are not supported in DX9.

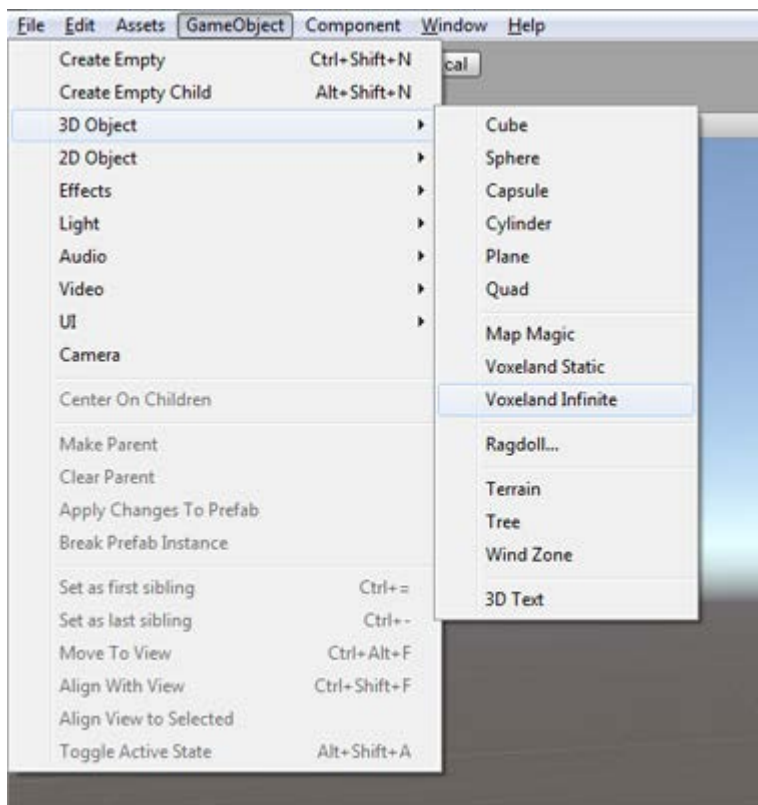
Voxeland is shipped with the two types of shaders: one for the texture arrays and one for the common textures, so you can turn texture arrays feature off and use common texture shader. It supports 4-9 different block types (depending on the platform) but works on all platforms.

Quick Start Guide

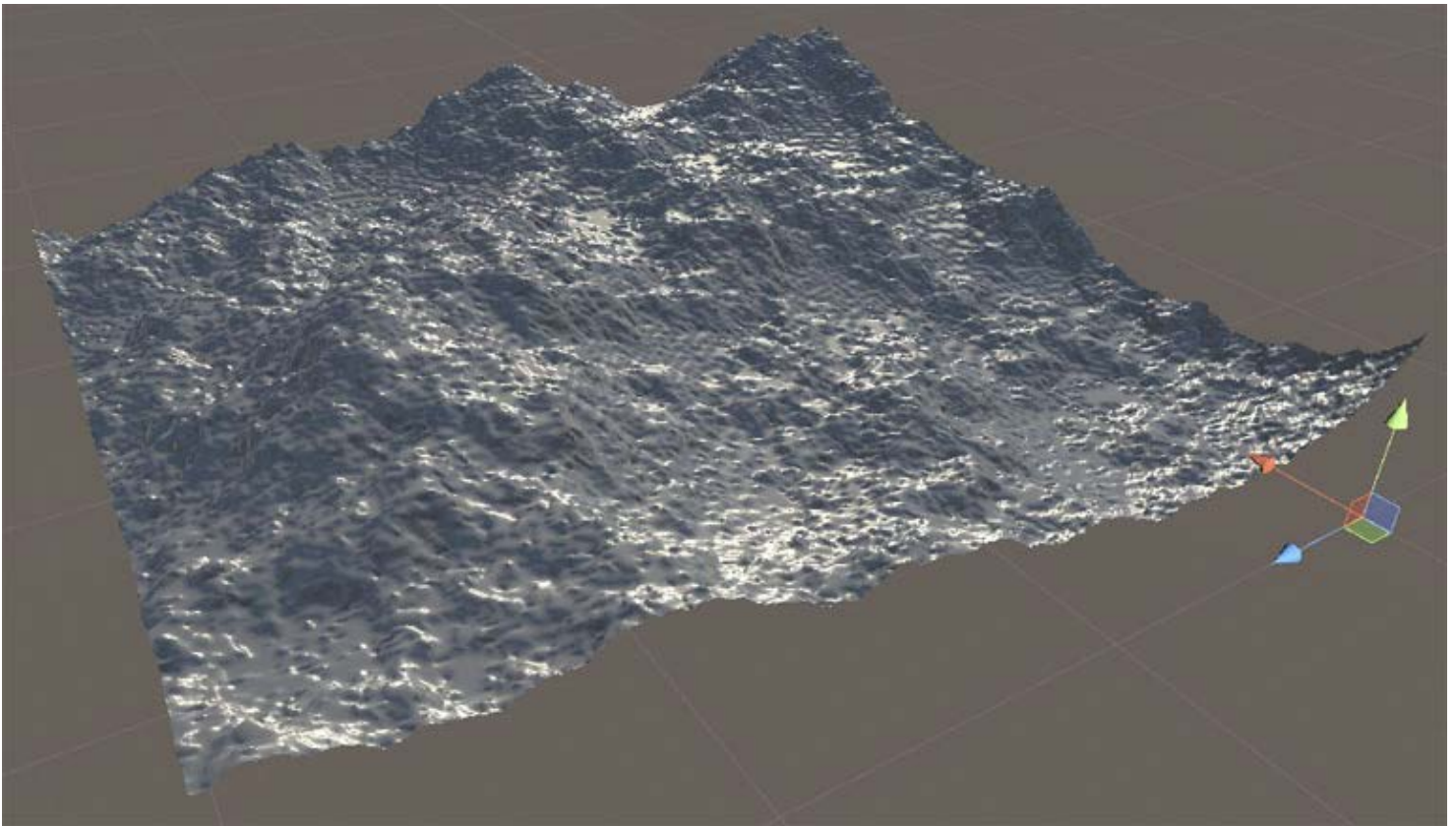
To create a new Voxeland object click Game Object -> 3D Object ->
Then, depending on the terrain type you are going to use, click:

- Voxeland Static - for small terrains about 300 units in size
- Voxeland Infinite - for infinite or just large terrains.

You can change the terrain type later in Settings.



You can see the new Voxeland object appeared in scene. If you use the infinite terrain it should build region just around main camera, otherwise it will create a terrain at 0,0 - 300,300 rect.



The initial terrain has the "Noise" generator enabled by the default. If you'd like to have a perfect flat terrain

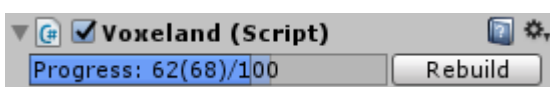
- go to Generate menu in inspector tab (Voxeland object should be selected);
- choose Generator Type "Planar";
- and press "Generate" button.

To add the first land texture

- go to Land Blocks menu in inspector tab (Voxeland object should be selected);
- here you can see the only land block type named "Block";
- click at the chevron icon ◀ right to the "Block" label to open it's properties;
- click "Load" button next to the Albedo/Height label to append the texture to the texture array. You can choose any texture from the project Assets folder, it ***does not have to be readable*** (read/write enabled toggle checked).
- click the similar button next to Normals label to add a normal map. Note that normal map texture should have "Normal map" type enabled in texture import settings.

Many settings changes require Rebuild operation to take effect. This could be done with the **Rebuild** button. Pressing it will remove all of the terrain chunks and create the new ones using the current voxel data. All of the land changes made with Voxeland will persist (since they are stored in voxel data).

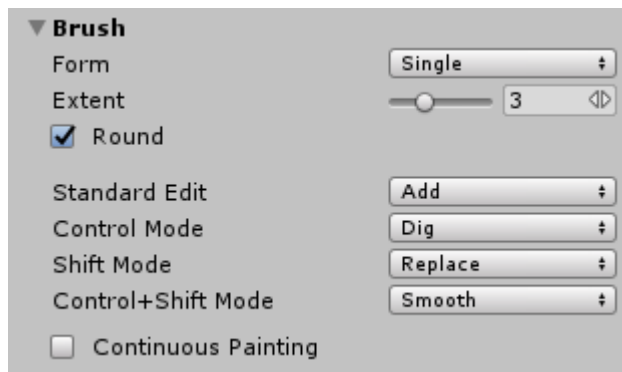
Next to the Rebuild button is a build process gauge. The chunks calculated in a separate thread and waiting for apply in main thread are displayed as pale blue. The ready applied chunks are displayed as opaque blue.



To edit the terrain click on the terrain:

- left click to dig the land
- ctrl-click to add the land

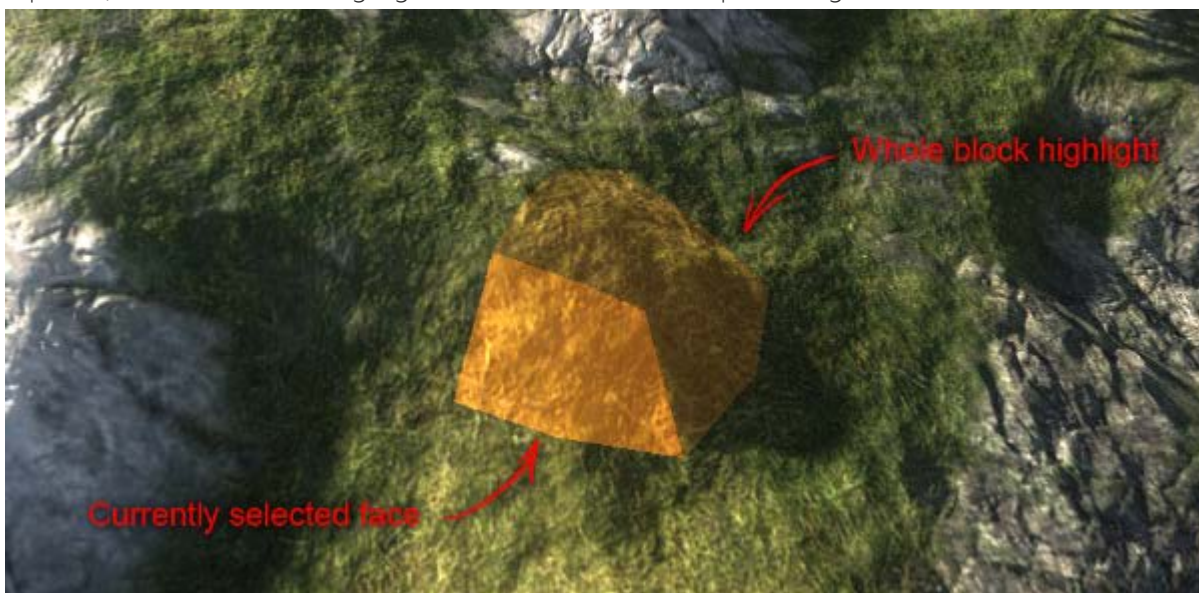
Brush



Brush Form:

Defines the basic brush behavior:

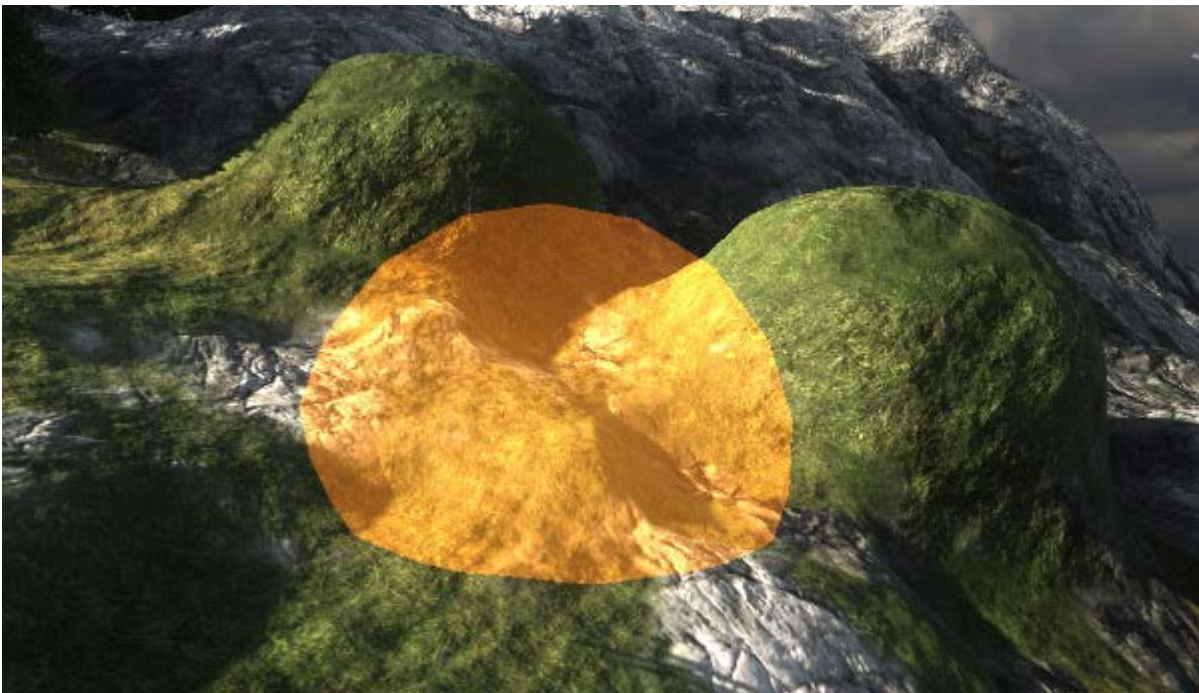
- Single: allows editing only one block at a time. When the single mode is selected the block highlight in scene view displays two zones: larger (the pale one) shows the full block (all of it's faces) that is going to be changed (digged or replaced), while the intense highlight shows the block face, representing an add block direction.



- Blob: highlights the blocks neighbor to the aimed one. The size of the brush is set with an Extent value. Editing with this brush acts like s Single mode, except it will change the neighbor blocks displayed. Note that all blocks changing are laying at the surface, that distinguish this brush from the Volume one.



- Volume: 3-dimensional brush in a form of cube (or sphere if Rounded mode is turned on). Editing with this brush will change all of the blocks within a cube/sphere, including the blocks that lay below the terrain surface.



- Stamp: 3-dimensional brush of any arbitrary form. When this form is enable an additional field to get stamp appears:

Get Stamp		<input checked="" type="checkbox"/>				
Min:	X	-2	Y	-1	Z	-2
Max:	X	2	Y	4	Z	2

Enabling "Get Stamp" will read the terrain data in a cube without changing it. The cube size could be set using Min and Max values (all the values are relative to the currently highlighted block).



Extent:

The size of the brush for the blob and volume brush forms. Consider it as a brush radius for the round brushes (or the half-size for the cube/square ones). Note that using a blob brush with a large extent can slow down display performance, so consider using Volume brush when working with the large extents.

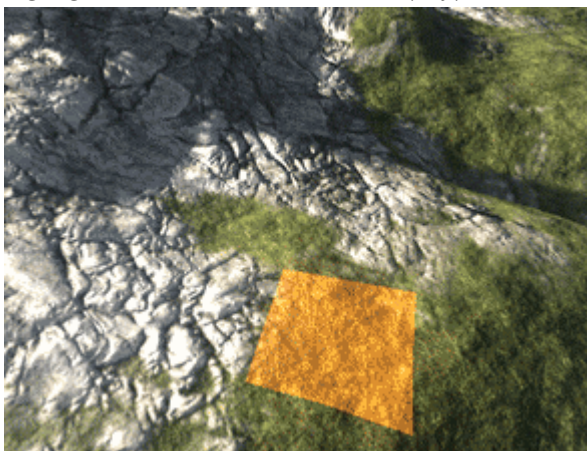
Round:

Cuts the brush square/cube edges so that it becomes round. Note that since Voxeland is a block terrain a true perfect roundness is not possible here, there will be still some block "stairs" effect.

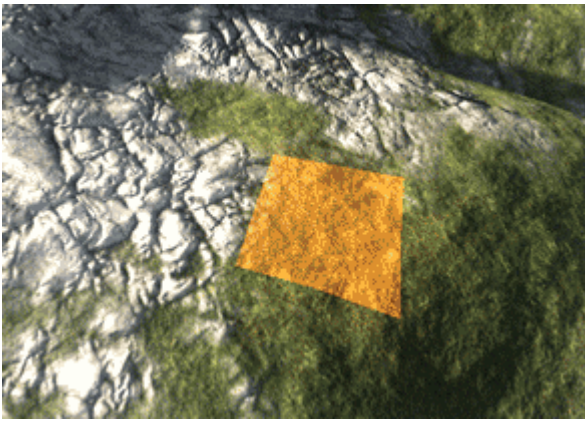
Edit Mode:

You can assign one of the following edit modes to edit the standard way (by left-clicking the mouse) or for one of the following alternative modes: shift-click, control -click or both shift+control mode:

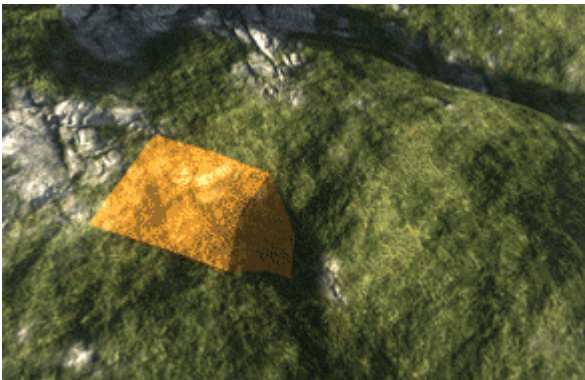
- Add: creates new block(s) of the selected Type. If single or Blob form is selected the block will be added next to currently highlighted face. If Volume or Stamp type is currently selected the blocks will fill the brush volume.



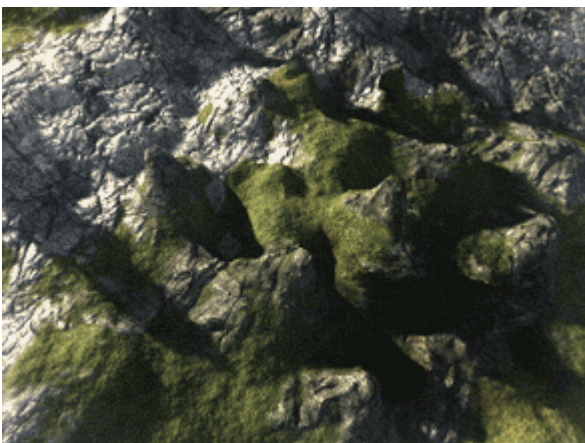
- Dig: will remove a block that is currently under the brush. If Volume or Stamp type is selected it will remove all of the blocks within the brush volume.



- Replace: will change all of the existing (filled with stratum) blocks with a selected type. All «air» blocks will not be changed, applying replace brush will not add or remove any blocks.



- Smooth (when using Volume or Stamp brush only) will add blocks in concave areas within the brush volume, and remove all protruding blocks. It will not add extra smoothness to the terrain surface, just will make internal cube blocks structure a bit more sleek.

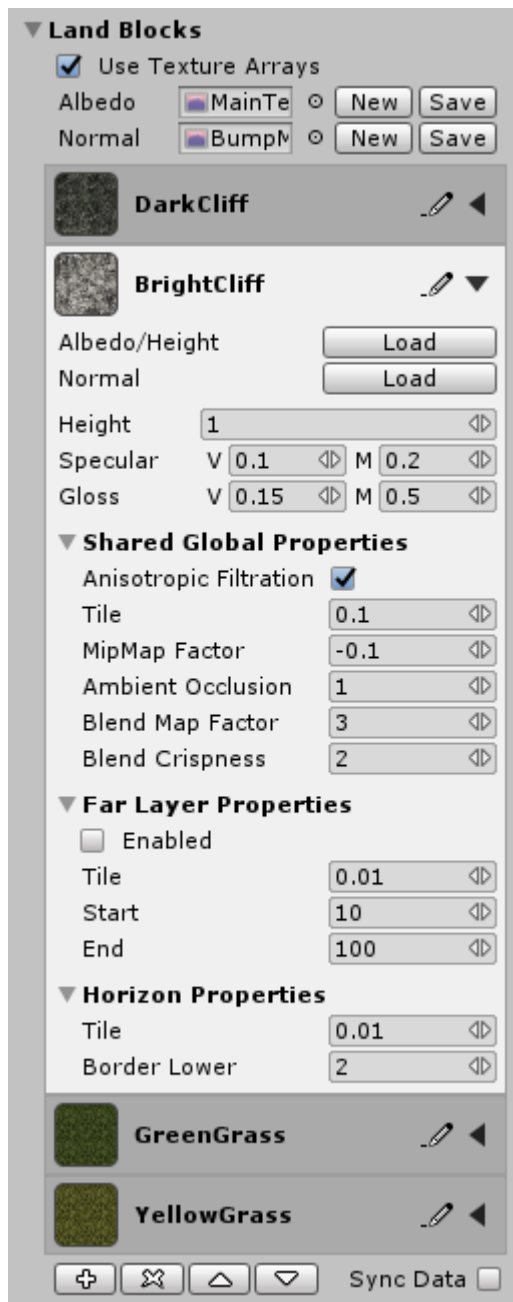


Note that when the Land blocks Type is selected all of the terrain changes are applied to the land only, not affecting objects or grass. When the Grass Type is selected only the grass is editable, the land and objects could not be changed. When the Objects Type is selected only objects could be edited.

Continuous Painting:

By the default (off) the terrain is edited only on click. Enabling Continuous Painting will change the terrain by dragging a mouse. Each edit step occurs when mouse cursor goes to the next block (i.e. dragging a mouse within a single face will not change a terrain).

Land Blocks



Voxeland uses the one material for all of the terrain chunks + one material for the horizon mesh.

Voxeland is shipped with two terrain shaders: the default one, called **Voxeland/Land TextureArray**, supports [texture arrays](#) and can use up to 24 different block types (diffuse+normal texture sets). The alternative, legacy **Voxeland/Land** shader supports 16 block types on DX11, 6 block types on Macs and 4 on other platforms. In addition, texture array shader is a bit faster. So, if your target platform [supports](#) texture arrays there is no reason to switch back to simple Land shader.

Texture Arrays

By the default Voxeland uses the texture arrays shader instead of the standard textures. This allows to blend 24 block types nearly on all the platforms, and do it a bit faster then blending the standard textures. However, this approach has some drawbacks: Texture2DArray is not just an array of separate textures, it's a single object that could not be edited in Photoshop or other graphics software. Texture arrays are generated in Voxeland using the standard source textures. So, after the editing

the texture it should be reloaded to the texture array manually.

Moreover, texture arrays are not supported in DX9.

If you do not wish to use the texture arrays uncheck the **Use Texture Arrays** toggle AND change the land material shader.

- **Albedo:** the albedo (RGB) and height texture array. By the default it is stored in scene - press "Save" to store the texture array as a separate asset file. Pressing "New" will clear the texture array.
- **Normal:** the normal map texture array. By the default it is stored in scene - press "Save" to store the texture array as a separate asset file. Pressing "New" will clear the texture array.

Using non-saved (non-stored to .asset file) texture arrays with saved material (stored to .mat file) will not serialize the assigned texture arrays. On any scene or game save/load the material textures will be displayed as blank. It's recommended to save both materials and texture arrays.

Types List

The block type could be selected by clicking on it's name or icon. All of the terrain edit is done with the currently selected type. Selecting the block type will deselect all of the others types (including grass, objects or other land types).

Type name could be changed with a pencil (✎) button.

Pressing the chevron (◀/▶) button at the right will roll out / roll up the currently selected type properties.

Type Properties:

Diff/Height texture: albedo map (RGB) and heightmap (A). The heightmap is used to blend terrain textures basing on their height values, this will give a pretty good result of rocks protruding from the grass/sand and such.



Not a big deal if alpha map will be absent, but it's highly recommended at least to try to restore heightmaps using normal maps. For instance, demo scene heightmaps were restored from normals using Knald.

Normal: a standard normal map used in any other normal mapped shader.

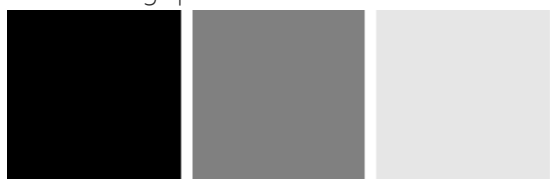
Height: heightmap (diffuse alpha) multiplier. Increase this value if you want to make the heightmap more crisp, and decrease it if you need more «flat» look.

Voxeland uses triplanar multichannel shader. The most performance-dependent part in such shaders is texture sampling. Voxeland got some shader optimizations like skipping sampling if the texture is not used according to vertex data, and using only two textures (diffuse/height and normal) in a texture set. All of the other textures like metallic and specular are generated from the diffuse data.

Specular block material specular values.

V (value) sets the constant specular value. It's uniform for all of the texture.

M (map) is the multiply factor for the grayscale diffuse texture. The more the value - the more specular the texture is. The resulting specular value is the sum of the V and M values.



V (value): 0, 0.5, 0.9



M (map): 0, 0.5, 1



V 0.5 + M 0.5

Changing these two values you can adjust the brightness and contrast of the grayscale diffuse texture to set the specular values.

Gloss: the glossiness values. Similar to the specular values,
V (value) sets the constant gloss value. It's uniform for all of the texture.
M (map) is the multiply factor for the grayscale diffuse texture. The more the value - the more gloss the texture is.

You can preview the resulting Metallic and Glossiness maps by switching Main Material "Preview" mode to Metallic/Smoothness respectfully.

Global Properties

These properties are shared between all of the types within a Voxeland object (but are not shared between different Voxeland objects or different scenes).

Anisotropic Filtering enables the [anisotropic filtering](#) for the texture arrays. Disabling it will **greatly** (up to 2-3 times) improve the shader performance with some noticeable visual quality loss.

Tile: number of times texture repeat itself in one world unit. Tile size of 0.1 means that the applied texture size is 10*10 units (meters).



Tile: 0.1, 0.2, 1

MipMap Bias: [mipmap](#) sampling factor. When the value is 0 the default mip is using. When value is above 0 more high-resolution mip is used, resulting in a more sharp, but noisy look. When the value is below zero low-resolution mip is used with a blurry look.



MipMap Bias (texture array shader): 1, 0, -1

Ambient Occlusion: the effect of the ambient occlusion effect in caves, holes or under overhangs. If you are going to turn ambient occlusion off (set to 0) don't forget to disable ambient occlusion generation in Settings to increase performance.



Ambient Occlusion: 0, 0.5, 1

Blend Crispness: How explicit the transitions between textures would be. Using low values makes the blending smooth, while high values make the blending sharp.

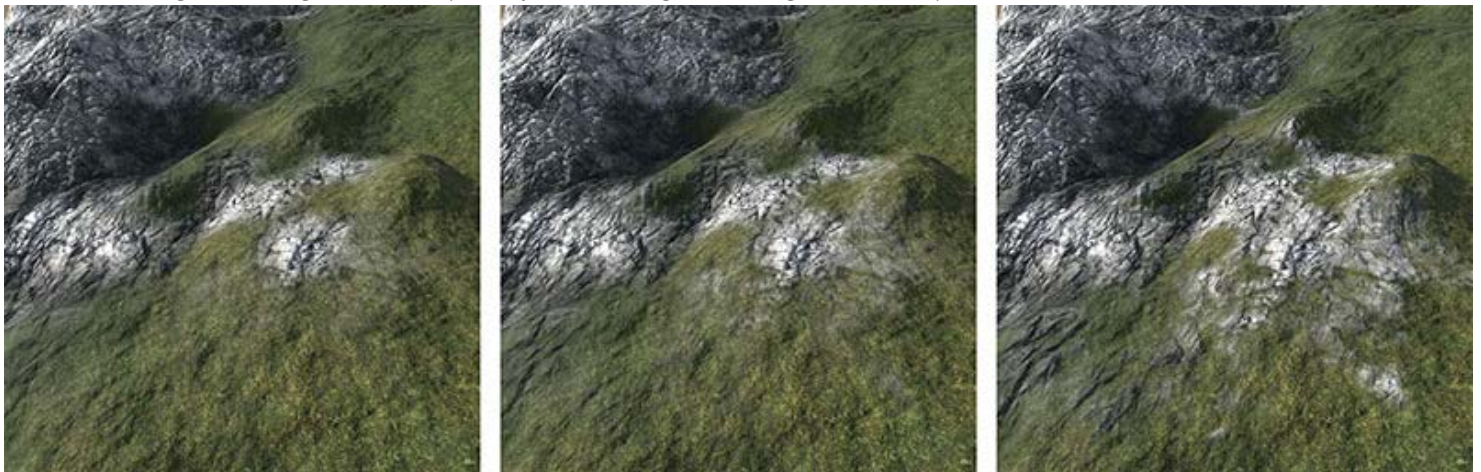
Be careful using low (below 0.5) or high (above 5) blend crispness values, especially when using low or high blend map factor values - it can cause artifacts.



Blend Crispness: 0.5, 2, 4

Blend Map Factor: The ratio between using the standard blend (like default terrain shader) and height-based blend. High values will give more pronounced heightmap effect, but can confuse the player about the real block types. Consider using low values if your game is related with digging or construction.

Be careful using low or high values, especially when using low or high blend crispness - it can cause artifacts.

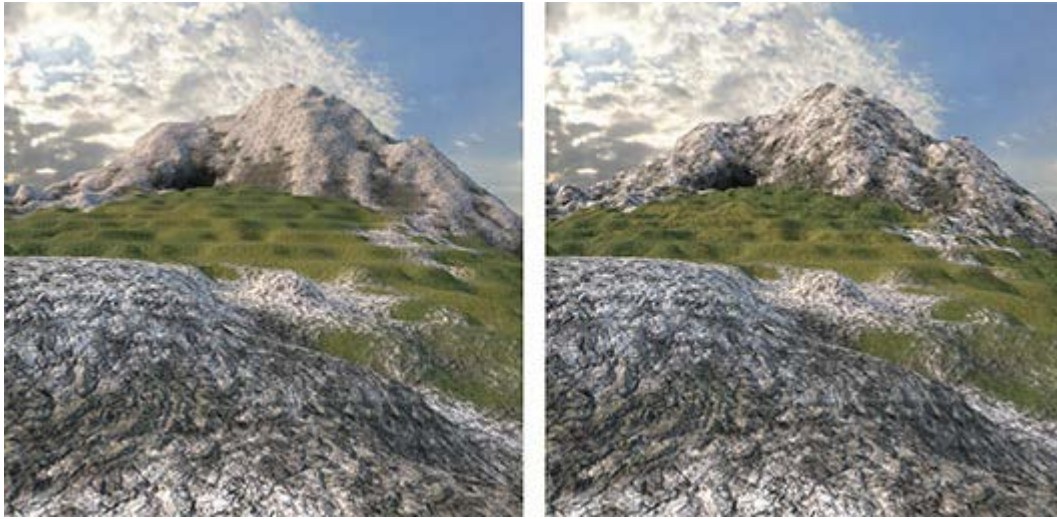


Blend Map Factor (Blend Crispness = 2): 0.1, 1, 3

Far Layer:

Enabling Far Layer feature reduces the shader performance twice. Avoid using it if your target platform has large display resolution.

Enables two-layer mode: one layer for close display, one for the distance. Using this feature can prevent tiling effect on the far scenery while keeping the resolution high in close range.



Far layer (tile = 1): off, on

Far Tile: Tile value for the far layer. Similarly to the Tile parameter, it is the number of times texture repeats itself in one world unit. This value should be much lower than the main Tile value, otherwise double layer feature will have no effect.



Far Tile (tile = 1): 1, 0.1, 0.01

Transition Start: the distance where the far layer starts to appear.

Transition End: the distance at which far layer is 100% visible.



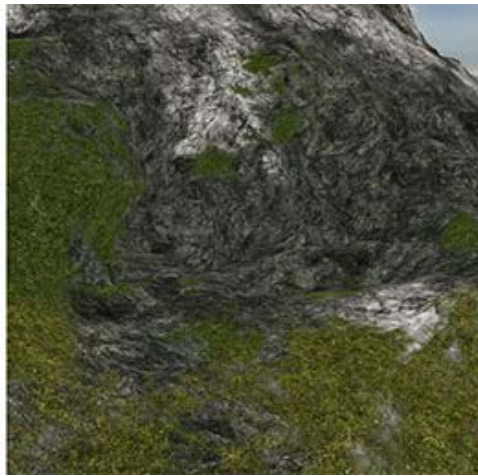
Transition Start: 0, 10, 40 Transition End: 5, 15, 45

Horizon Mesh:

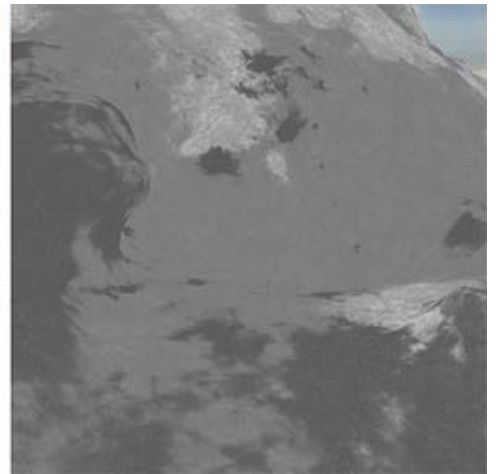
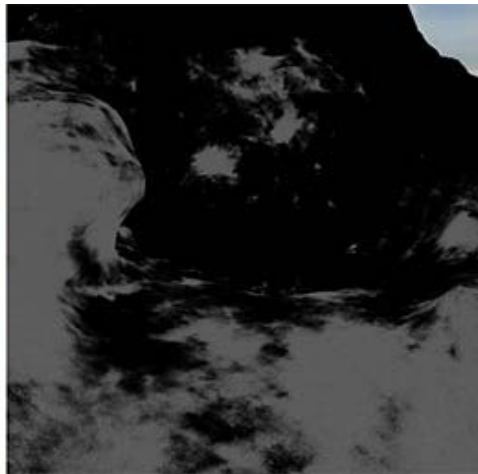
The special settings for the [Horizon Mesh](#). It uses the same material with the `_HORIZON` keyword enabled and slightly different settings. **Tile**: same as tile for the main material, it sets the number of times texture repeat itself in one world unit. It's recommended to set the horizon tile way lower than the main tile for better visual quality. **Border Lower**: lowers the horizon mesh borders on the seams between the voxel meshes and horizon mesh. Increase this value if you get seams on the horizon.

Preview Mode:

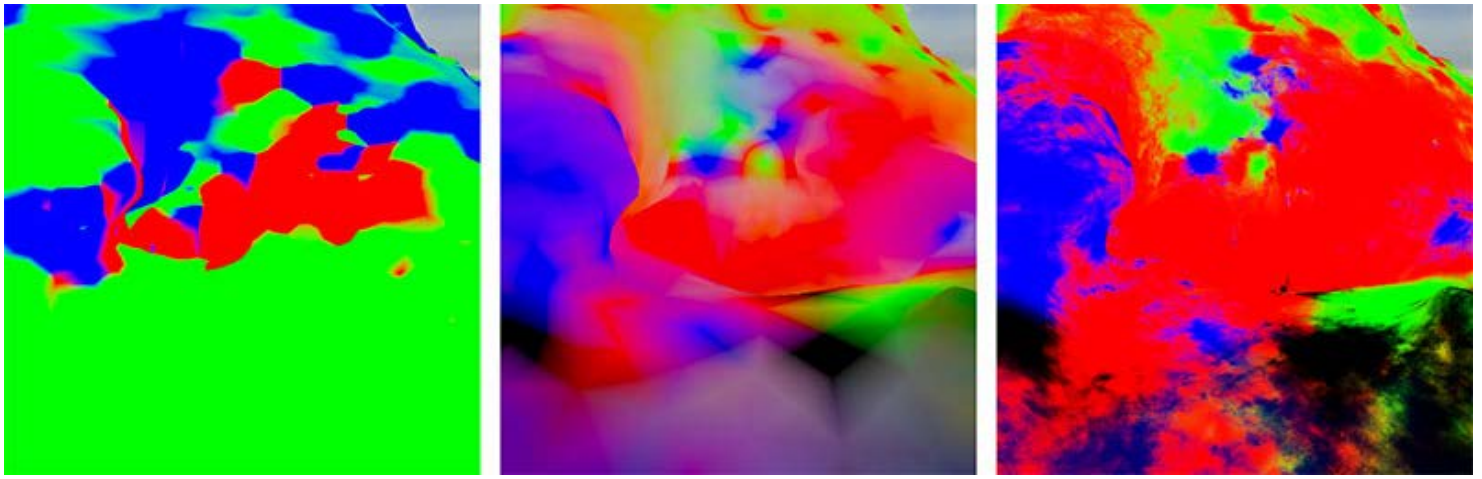
- Preview: various debugging display modes. Could be handy to set metallic/gloss values, monitor ambient level in caves, etc.



Preview: Off, Albedo, Ambient







Preview: Normals, Metallic, Smoothness(glossiness)



Preview: Directions, Vertex Blends, Final Blends

Organizing Types:

The types could be added, removed or organized using these buttons:

-  Add: will add new layer atop of the selected one.
-  Remove: removes currently selected layer.
-  Up: will change the layers order by moving the selected layer up.
-  Down: will change the layers order by moving the selected layer down.

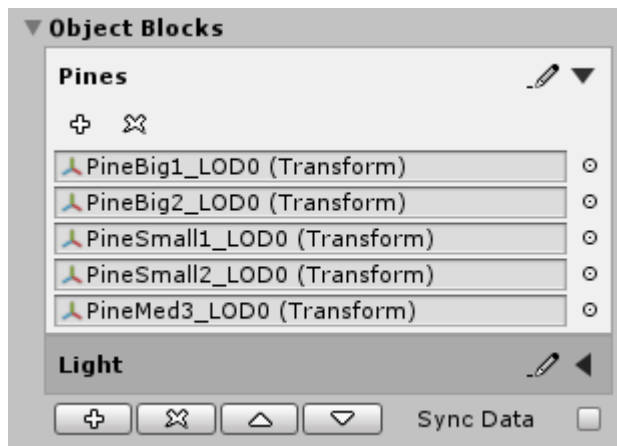
Sync Data: will swap all the internal data bytes when changing the layer order. When disabled, swapping layers will re-assign block textures and other data, when enabled the internal data will be changed along with the layer swap, making order change unnoticeable.

Optimizing Graphics Performance:

Since the Voxeland shader uses multi-blended triplanar mapping it uses really a lot of texture sampling instructions: 3 samples for each visible layers in each fragment (invisible layers are skipped). This way shader performance could be slow on high monitor resolutions. Consider these steps to improve it:

- Set MipMap Bias factor to 0;
- Do not use Far Layer mode;
- Turn off Anisotropic Filtering in Voxeland Settings.

Object Types



The objects type could be selected by clicking on it's name or icon. All of the terrain edit is done with the currently selected type. Selecting the block type will deselect all of the others types (including grass, land or other object types).

Type name could be changed with a pencil (✎) button.

Pressing the chevron (◀/▶) button at the right will roll out / roll up the currently selected type properties.

Type Properties:

Each objects block type contains an array of prefabs. Each prefab is spawned on the appropriate coordinates at random. Use plus button (+) to add a new line, or cross (✖) to remove the last line.

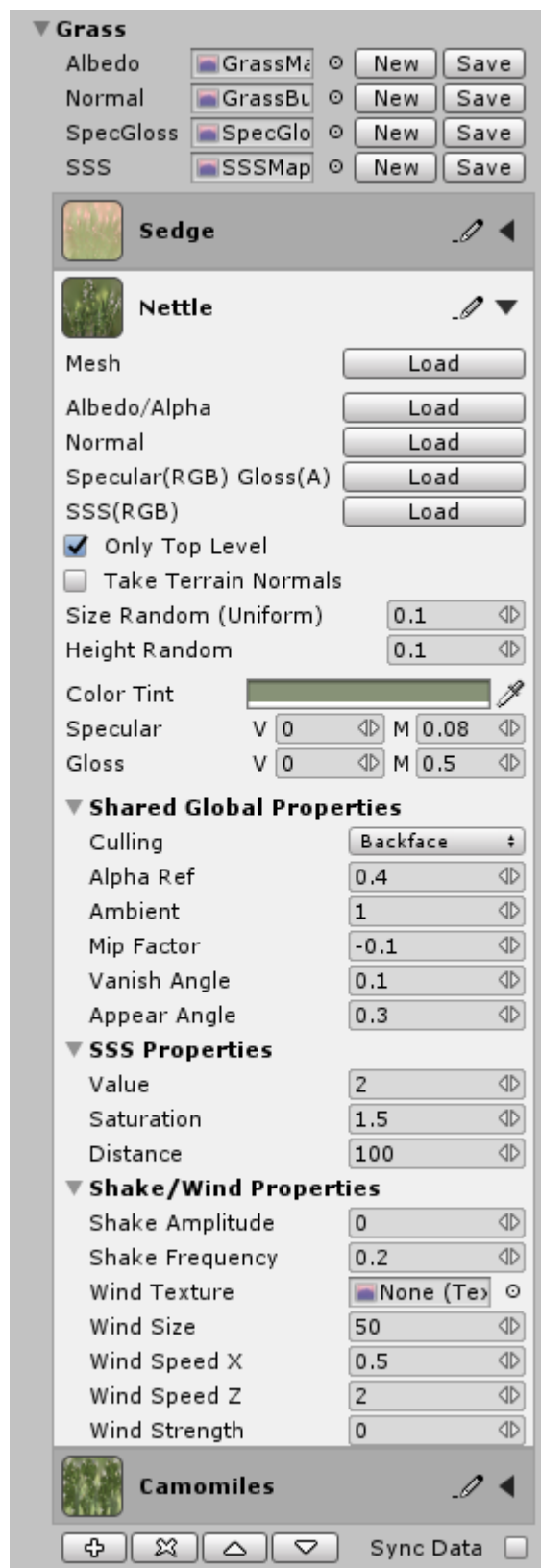
Organizing Types:

The types could be added, removed or organized using these buttons:

- + Add: will add new layer atop of the selected one.
- ✖ Remove: removes currently selected layer.
- ▲ Up: will change the layers order by moving the selected layer up.
- ▼ Down: will change the layers order by moving the selected layer down.

Sync Data: will swap all the internal data bytes when changing the layer order. When disabled, swapping layers will re-assign block textures and other data, when enabled the internal data will be changed along with the layer swap, making order change unnoticeable.

Grass Types



The grass type could be selected by clicking on it's name or icon. All of the terrain edit is done with the currently selected type. Selecting the block type will deselect all of the others types (including land, objects or other grass types).

Type name could be changed with a pencil (✎) button.

Pressing the chevron (◀/▶) button at the right will roll out / roll up the currently selected type properties.

Texture Arrays

By the default Voxeland uses the texture arrays shader instead of the standard textures. This allows to use 10 grass types nearly on all the platforms, and do it a bit faster then the standard textures. However, this approach has some drawbacks: Texture2DArray is not just an array of separate textures, it's a single object that could not be edited in Photoshop or other graphics software. Texture arrays are generated in Voxeland using the standard source textures. So, after the editing the texture it should be reloaded to the texture array manually.

Moreover, texture arrays are not supported in DX9.

If you do not wish to use the texture arrays uncheck the **Use Texture Arrays** toggle AND change the land material shader.

- **Albedo/Alpha:** the albedo (RGB) and alpha-test opacity (A) texture array. By the default it is stored in scene - press "Save" to store the texture array as a separate asset file. Pressing "New" will clear the texture array.
- **Normal:** grass bump texture. "New" and "Save" buttons are equivalent to the ones used in albedo.
- **Specular(RGB) Gloss(A):** specular power (color channels) and smoothness value (alpha) array.
- **SSS(RGB) Vanish(A):** grass subsurface scattering (translucency) value (red channel) and vanish map (alpha) array.

Using non-saved (non-stored to .asset file) texture arrays with saved material (stored to .mat file) will not serialize the assigned texture arrays. On any scene or game save/load the material textures will be displayed as blank. It's recommended to save both materials and texture arrays.

Type Properties:

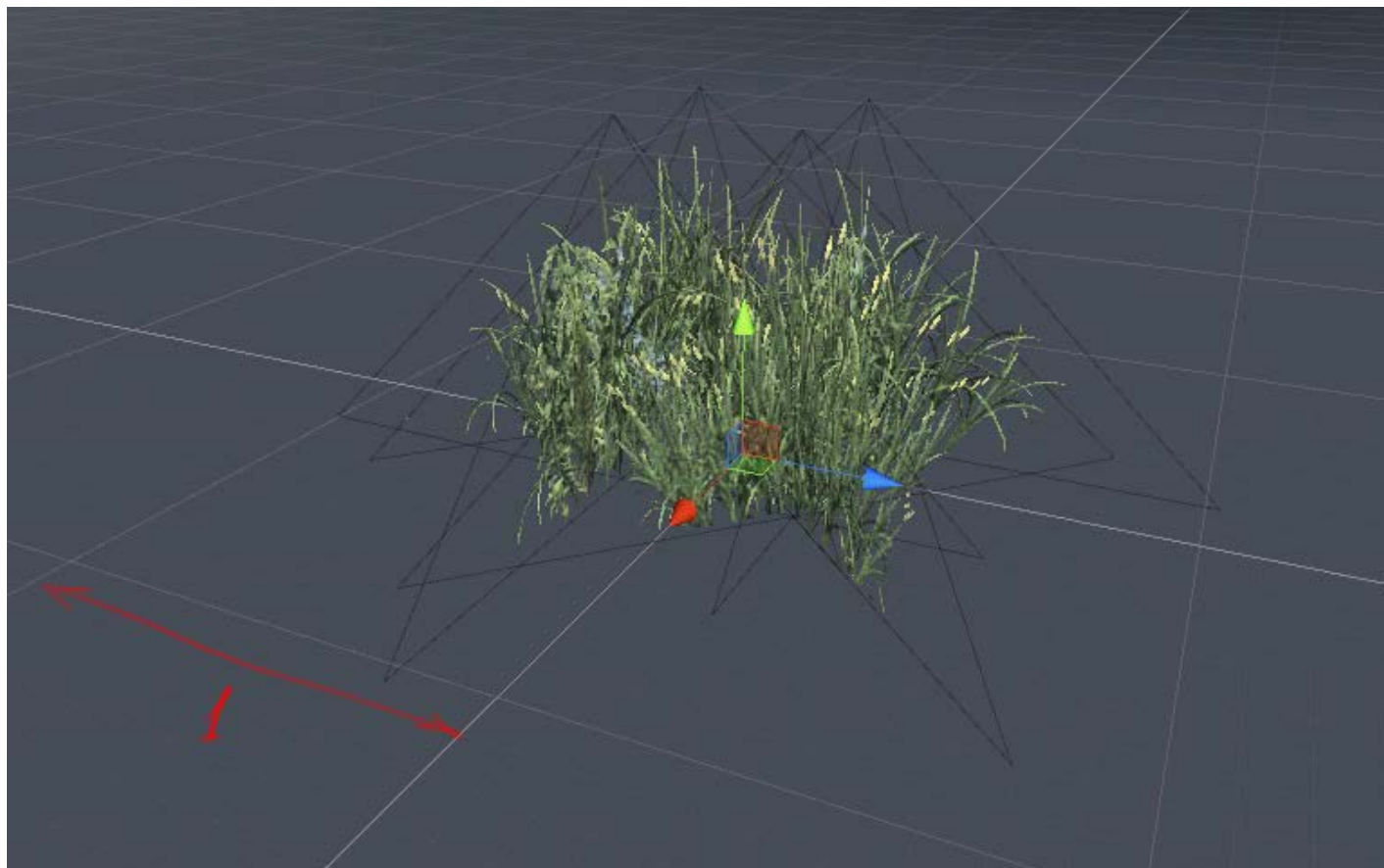
Mesh a grass mesh used to scatter grass. Consider it as a grass bush 1 meter * 1 meter in size. Each block face facing up will have a grass mesh applied on top. Note that the grass bushes are Skewed and bended to fit the face orientation.



Grass mesh requirements:

- 1*1 units in size. Not a big deal if the grass mesh will be bigger, but keep in mind that this way it can spread to nearby blocks.
- object pivot and scene center both located at the center of the object on X and Z axes.

- object pivot and scene center should be located at the ground level at Y axis. Object can be *вылезать* down a bit to avoid bushes floating in the air in curvy terrain.
- this way ideal grass mesh should fit in $(-0.5, 0, -0.5) - (0.5, 1, 0.5)$ bounds, but commonly it can exceed these bounds.
- object Y axis should face up.
- since the bushes of grass types are batched together **no** multi-material is allowed.
- to use the [wind animation](#) UV4 channel should be used.



Albedo/Alpha: the albedo (RGB) and alpha-test opacity (A) texture array. **Normal:** grass bump texture. Normal map texture type should be defined in a loaded texture import settings. **Specular(RGB) Gloss(A):** specular power (color channels) and smoothness value (alpha). **SSS(RGB) Vanish(A):** grass subsurface scattering (translucency) value (red channel) and vanish (alpha) map.

Alpha map is a Vanish channel used to mask grass planes depending on a view angle to prevent showing triangles nearly perpendicular to the camera. The less the alpha value - the faster the pixel will be hidden.

Only top level:

Grass blocks map is 2-dimensional. This means that if the grass grows at some X and Z coordinates at some elevation it will grow in caves with the same X and Z.

To prevent this use Only Top Level feature – this will make grass grow only one bush per coordinate, and only at the top surface level possible.

Take Terrain Normals: will apply terrain surface normals to the grass bush that grows upon. This will make the grass lighted the same way terrain does, making the grass fit in terrain better.

Size Random: the uniform size random factor, sets the amount of the random value. Random value is unique for each of the grass block. Every vertex position in each grass block is multiplied with this value.

Height Random: similar to the Size Random, but affects only height. The result is multiplicative with Size Random.

Color Tint: color that multiplies the albedo texture.

Specular block material specular values.

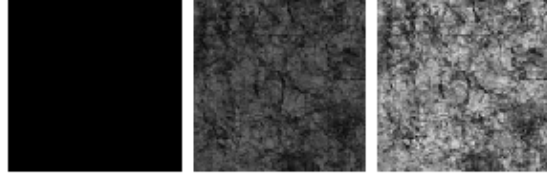
V (value) sets the constant specular value. It's uniform for all of the texture.

M (map) is the multiply factor for the grayscale diffuse texture. The more the value - the more specular the texture is.

The resulting specular value is the sum of the V and M values.



V (value): 0, 0.5, 0.9



M (map): 0, 0.5, 1



V 0.5 + M 0.5

Changing these two values you can adjust the brightness and contrast of the grayscale diffuse texture to set the specular values.

Gloss: the glossiness values. Similar to the specular values,

V (value) sets the constant gloss value. It's uniform for all of the texture.

M (map) is the multiply factor for the grayscale diffuse texture. The more the value - the more gloss the texture is.

You can preview the resulting Metallic and Glossiness maps by switching Main Material "Preview" mode to Metallic/Smoothness respectfully.

Global Properties

These properties are shared between all of the types within a Voxeland object (but are not shared between different Voxeland objects or different scenes).

Alpha Ref: alpha cutoff value to determine the cutoff point for the which areas will be shown.



AlphaRef: 0.1, 0.4, 0.8



Ambient: same as block type ambient: the effect of the ambient occlusion effect in caves, holes or under overhangs. If you are going to turn ambient occlusion off (set to 0).

MipMap Bias: [mipmap](#) sampling factor. When the value is 0 the default mip is using. When value is above 0 more high-resolution mip is used, resulting in a more sharp, but noisy look. When the value is below zero low-resolution mip is used with a blurry look.



MipMap Bias: -1, 0, 1

Vanish Angle: Voxeland masks the grass planes that are nearly perpendicular to the camera, leaving just planes that are more or less facing camera. This will remove noticeable triangles with a stretched texture making a grass look more natural. This value determines the view angle at which the plane is fully hidden. Measured in percent: 0 is fully perpendicular triangle, 1 is a triangle facing camera.

Appear Angle: Determines the view angle at which the face is fully displayed. All the values in-between appear vanish angle and appear angle are hidden or showed using the SSS/Vanish texture alpha channel.



Vanish = Appear: 0, 0.2, 0.5

SSS

Makes the back surfaces of the grass emit light, simulating the effect of the lights scattered within the grass leaf.

Value: translucency amount. At the value of 2 the amount of color emitted by the back side of the surface is equal to the amount of the light received by the front surface. The value of 1 and above is not physically correct.



SSS Value: 0, 2, 4

Saturation: color saturation factor for the light emitted by back surface. Increasing this value adds more color to emitted light. Setting the value to 0 will result in desaturated SSS color.



SSS Sturation: 0, 1, 4

Distance: the SSS effect range. For the faces located farer than this distance the SSS effect is turned off to prevent “glowing” objects at the ranges where shadows are not casted. Make sure that SSS Distance value is twice lower than the shadow distance to prevent visual artifacts.

Shake/Wind

The shake and wind values use the UV channel 4 of the grass map. The Y axis of the uv4 map determines the amount of wind effect: this value should be equal to 0 at the grass roots (to prevent grass “floating” at the land surface). The X axis of the uv4 map determines the frequency factor to make the shake non-uniformly. Generally all of the vertices should be set to 0.5-1.25 at the X axis,. The higher the grass - the lower X value it should have.

Shake amplitude and frequency define the constant shake parameters. Additionally, you can set the wind feature that will bend the grass using the special wind texture. Resulting wind value is multiplied with a uv4 X value too.

Shake Amplitude: the amount of the constant shake.

Shake Frequency: the speed of the constant shake.

Wind Texture: the wind normal map. Each pixel represents the bend direction at the current texture position.

Wind Size: wind texture size in world units.





Wind Speed X: the wind texture move speed in X direction

Wind Speed Z: the wind texture move speed in Z direction

Wind Strength: the factor the wind affects vertices.

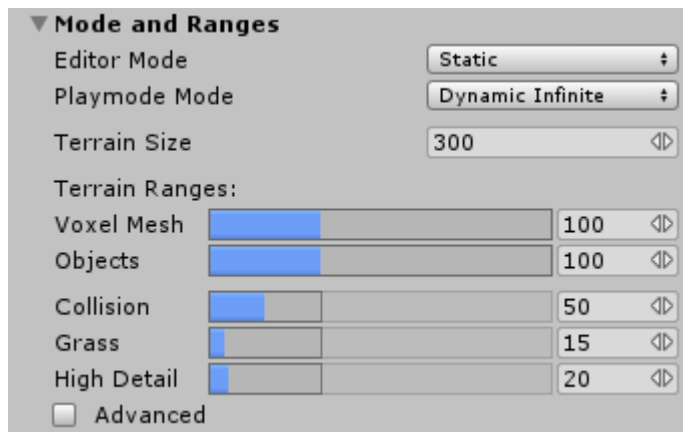
Organizing Types:

The types could be added, removed or organized using these buttons:

-  Add: will add new layer atop of the selected one.
-  Remove: removes currently selected layer.
-  Up: will change the layers order by moving the selected layer up.
-  Down: will change the layers order by moving the selected layer down.

Sync Data: will swap all the internal data bytes when changing the layer order. When disabled, swapping layers will re-assign block textures and other data, when enabled the internal data will be changed along with the layer swap, making order change unnoticeable.

Mode and Ranges



See [Build and Generate difference](#) for guidance.

Modes

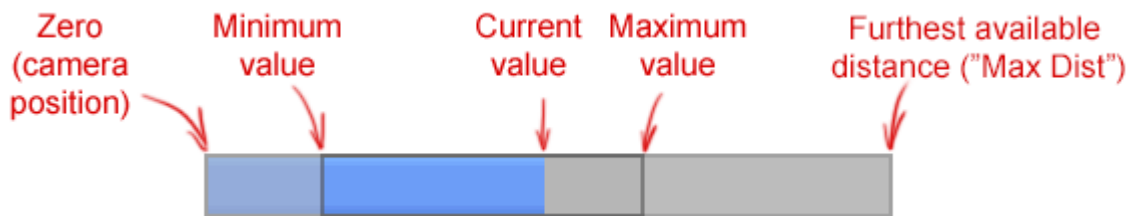
- **Dynamic Infinite:** Voxeland creates only several chunks within a given radius around camera, adding the new chunks as camera moves through the scene. The new land is created as camera moves through the scene, making a terrain virtually infinite. An object(s) with some specific tag like character could be used with or instead of the main camera. Voxeland has floating point origin solution implemented to make the infinite terrain independent from float values precision.
- **Dynamic Limited:** a compromise between a static and infinite terrain. It build all of the chunks at once (on Voxeland enable), but still switches LOD and enables/disables chunks to maintain visible chunks only within given range from camera (but does not build them from scratch). It's not recommended to use limited terrains more than 400-500 meters in size because of performance reasons (although the built chunks are not displayed, they still consume significant amount of memory).
- **Static:** builds all of the chunks at once, and does not change the terrain in any way depending on the camera range (no LOD switches, no chunk enabling/disabling). You get exactly the same terrain no matter of point of view or camera or other objects position. Use this mode for tiny terrains (100-200 units in size) because of performance reasons.
- **Baked:** same as the static mode, but built chunks will be stored in scene as meshes. Voxeland will nor re-build them on scene load. It's still possible to change the terrain in baked mode – Voxeland will automatically re-bake changed chunks. It's recommended to use baked mode with some third-party streaming solution.

You can assign different build modes to the playmode and the editor:

- **Editor Mode:** the build mode that's active when the Unity play mode is off. It uses the scene camera as the main camera to determine chunks within build ranges.
- **Playmode Mode:** the build mode that's active when "Play" button is pressed (and in the built project). Note that switching to the scene view while in playmode will not enable "Editor Mode". Editor mode is disabled while the playmode is on no matter of the current active view – this way you can debug Voxeland chunks built around camera/character in game.

Ranges:

Each of ranges is represented with a gauge made using a following scheme:



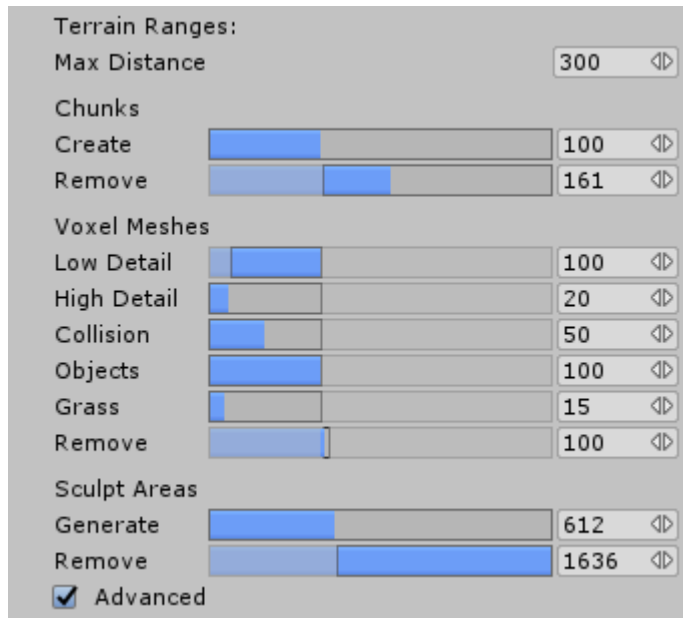
The left side of a gauge is a "zero", no offset from the camera position.

The "Minimum" and "Maximum" values usually depend on the other settings - for example, collider enabling distance could not be higher than mesh build distance, there is no collider while the mesh itself is not calculated.

The right side of the gauge - is the biggest distance among all of the sliders. You can set it manually in an Advanced Mode with a "Max Distance" value.

- **Voxel Mesh:** the mesh build distance. If chunk is within this range it should be building or already built.
- **Objects:** the objects placement distance. If chunk is within this range all of it's objects blocks should be placed. Setting this distance higher than build distance will show the objects before the mesh (on [horizon](#)), it could be handy to display far objects such as trees.
- **Collision:** collider enabling range. Since enabling a mesh collider is the operation that can cause the most noticeable lags, it's recommended to keep as less chunks as possible within Collision range. Typically it's the terrain aiming and NPC spawning distance.
- **Grass:** grass build distance.
- **High Detail:** LOD switch distance. Meshes out of this range have 4 times less triangles.

Advanced Mode



Could be enabled with **Advanced Mode** toggle. Gives more control over distances: all of the ranges are controlled manually.

Max Distance sets the right gauge value for all of the ranges except Area ranges (Generate and Remove). Only GUI setting, does not play any direct role.

- **Chunks Create:** mesh build distance.
- **Chunks Remove:** the created chunks are not removed right after player leaves them, there is some margin in case player would like to return. Remove distance sets the range of chunk removing.
- **Low Detail:** LOD meshes range (4 times less triangles)

- **High Detail:** main chunk mesh range
- **Collision:** collider enabling range
- **Objects:** the objects placement distance
- **Grass:** grass build distance.
- **Area Generate:** terrain generate distance (i.e. use MapMagic or internal generator to create voxel data)
- **Area Remove:** range of removing un-pinned (i.e. unchanged) generated areas. Similarly to Chunks Remove distance it should be higher than generate distance to leave some margin.

General Settings

▼ Settings

Data

DemoDataCopy (Data) ○

Reset

Release

Save as Copy

Mesh Margin

2

◀▶

Ambient Margin

7

◀▶

Ambient Fade

0.7

◀▶

Smooth Normals

0

◀▶

Relax Strength

1.95

◀▶

Relax Iterations

2

◀▶

Anisotropic Filtration

☒

Color Channels (RTP Mod)

☐

Edit in Playmode

☐

Multithreading

☒

Max Threads

3

◀▶

☒ Auto

Max Apply Time

15

◀▶

Lock Selection

☒

Hide Frame

☒

Hide Collider Wire

☒

Max Brush Size

10

◀▶

Large brush extent can slow down Blob Brush display

Save Meshes with Scene

☒

Save Non-pinned Areas

☐

Default Chunk Name

Chunk

Copy Layers and Tags to

☒

Copy Components to Chu

☐

Object Blocks

☐ Regard Prefab Rotation

☐ Regard Prefab Scale

☐ Instantiate Clones

☐ Debug (Requires re-compile)

Data:

The voxel block types data. Consider it as a losless-compressed 3-dimensional array of bytes (or a 3-dimensional image). Each byte (or pixel) represent a land type number. 255 (or the number specified in Data.emptyBlock) is for an empty block. Data stores grass and objects “arrays” too. It does not store any information about block type textures, material preproperties, object prefabs, etc. Just the pure numbers.

You can assign a saved data in **Data** slot - this will replace your current graph with an assigned one. Re-selecting a MapMagic object is required for changes to take effect.

- **Reset** removes all of the block information in data structure and replaces it an empty one. This operation is not undo-able;
- **Store to Assets** will export current data as an .asset file and assign the stored scene in a data slot. All the further blocks

changes will cause changing of the stored asset.

- **Save as Copy** will duplicate current data and save it as an .asset file. Changing the terrain will not affect the stored asset.

Margins:

- **Mesh Margin:** number of invisible blocks around each chunk Voxeland will calculate to remove the geometrical seams (holes) between chunks. Mesh Margin of 2 with a chunk size 30 means that actual chunk size is 34 ($2+30+2$), but only central 30 of them is visible. Decreasing this value will increase the chunk build performance, but may cause see-through holes between chunks. This value should be used together with the Relax Iterations value, if you haven't changed it then leave Mesh Margin default.
- **Ambient Margin:** number of invisible blocks around each chunk that will be used to calculate ambient occlusion. Increasing this value will help to lower or remove a brightness seams between chunks, especially in cave entrances, canyons, overhangs and other areas with an ambient gradients. Increase this value after increasing Ambient Fade to remove brightness seams.
- **Ambient Fade:** how far the ambient lights проникает in caves, holes and other dark areas. Ambient occlusion smoothness. Each next block ambient not exposed to "sun" (sky) will receive an ambient value of a previous block multiplied with Ambient Fade value. This way using the value of 1 will not fade ambient at all, while value of 0 will light only the blocks exposed to sun. Ambient Margin value should be increased after increasing this value.

Relax:

- **Smooth Normals:** sets additional visual terrain smoothness without actually changing the geometry by smoothing only the vertex normals. Increasing this value does not require changing Mesh Margins and does not affect the performance, but this smoothness is a bit "fake", it can cause artifacts when using crisp normal maps.
- **Relax Strength:** sets the terrain smoothness by moving chunk vertices. This is the factor that controls vertex relax per iteration. Using the value of 1 will return the maximum terrain smoothness with given Iterations count. Raising this value to more than 1 will move vertices farther than their maximum smoothed positions, this can produce visual artifacts. Changing Relax Strength value does not affect the mesh build performance.
- **Relax Iterations:** the number of iterations used to smooth the terrain. Image. Raising this value slightly lowers the mesh build performance, but requires raising Mesh Margins to weld the seams between chunks (while high margins can affect mesh build performance a lot).

Threads:

- **Multithreaded:** when enabled, Voxeland chunk build works in a several separate threads. Turning this parameter off forces Voxeland to work in the main thread only using co-routines only (this could be useful for debugging of custom generators or for compatibility reasons).
 - **Max Threads:** the number of threads Voxeland uses to calculate chunks. The recommended value is number of current processor threads minus one.
 - **Max Threads Auto:** sets the number of threads to processor thread count - 1. This way, on the different hardware the number of threads will be different. Auto mode works both in editor and in build.
 - **Max Apply Time:** Although calculation happens in other threads, applying calculated results to the terrain, the final stage, including the most resource-intensive part is applying a mesh collider, is done using the co-routines. This value sets the maximum time spent per frame by co-routine in milliseconds. Increasing this value will shorten the overall apply time, but can create a noticeable lag. Note that in some cases co-routine could not be split in smaller parts, so setting this value to 1 will not help to avoid some profiler spikes.

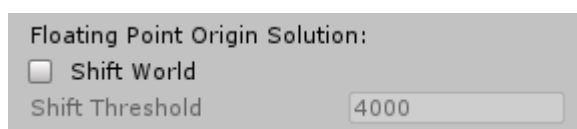
Other Settings:

- **Color Channels (RTP Mode):** by the default mesh texture blending information (i.e. what block texture should be used at current mesh surface) is stored in UV channels - since Voxeland uses triplanar shader it does not require mapping coordinates. But ReliefTerrain Pack reads blending information from Color channel. Turn this toggle on if you are going to use RTP's triplanar shader instead of a built-in one. Chunks will not be displayed properly when this mode is enabled without RTP shader.
- **Edit in Playmode:** makes the terrain editable in playmode. Adds a possibility to edit terrain in-game the way it is done in editor. It is recommended to use special Edit() controller instead (see Demo VoxelandController example script). Using a custom controller you can assign your own mouse or button controls or fire special events (like spawning a pick-able object) or conditions (like mouse hold timer before the actual edit, Minecraft style).
- **Lock Selection:** when enabled will not select any object when clicked in scene view, but will edit terrain instead. This is the default behavior. To select chunk use a hierarchy menu or select other object before clicking in scene.
- **Hide Frame:** hides selection frame around Voxeland object.
- **Hide Collider Frame:** hides collider wireframe when Voxeland is selected. This will disable all MeshCollider gizmos in scene, not only Voxeland ones.
- **Max Brush Size:** the right maximum value of a brush size slider in Brush foldout. Editing with a large brush can slow down mesh build performance, using the large brush extent with a Blob brush will slow down visual performance.
- **Save Meshes with Scene:** will save built meshes to scene. Consider it as baking meshes. Storing meshes will greatly increase the scene size, but Voxeland will not re-build them on scene load. It's still possible to change the terrain in baked mode - Voxeland will automatically re-bake changed chunks. It's recommended to use baked mode with some third-party streaming solution.
- **Save non-pinned Areas:** enabling will store all of the generated voxel data with the scene. When disabled only the changed chunks voxel data is stored.

Chunk Objects:

- **Default Chunk Name:** the default name for the Voxeland chunk game object. Note that the chunk coordinate postfix will be added to this name, for example ChunkName 30,-60.
- **Copy Layers and Tags to Terrains:** when turned on Voxeland will copy all of the tags and layers assigned to the Voxeland object to terrains on generate.
- **Copy Components to Terrains:** when turned on Voxeland will copy all the scripts assigned to the Voxeland object (except Voxeland itself) to terrains on their generate.

Floating Origin Solution



One of the problems related with the vast or infinite worlds lies in floating point precision limitation. When camera is placed far from the zero coordinate lighting, shadows and z-fighting problems could be experienced. There's a solution called [Floating Origin](#) that can help to solve that. It places the camera closer to origin when it exceeds some threshold from zero coordinate. All of the world objects are placed relatively along with the camera so that player does not notice that shift had happened.

Voxeland already has Floating Origin integrated, so it safely offsets camera and Voxeland object. Since MM works in multiple threads and co-routines this offset can happen during the terrain generate - in this case Voxeland ensures that the generated terrains appear at the right position, with the objects scattered properly.

To turn on Floating Origin feature enable the **Shift World** toggle. Set the **Shift threshold** to the offset step, so that the

camera will not exceed the double of this distance (for threshold of 4000 maximum camera position is 8000 on any axis).

Object Pool:

Voxeland object blocks creation properties (like trees from demo scene).

- **Regard Prefab Rotation:** when disable, will create a prefab with a zero rotation (no matter what the prefab rotation is), and then rotate it random degrees along Y axis. When enabled, will create a transform with a rotation given in a prefab, and then add random rotation around Y axis. This is useful for prefabs exported from 3DS Max that have the default rotation -90 degrees along X.
- **Regard Prefab Scale:** similarly to the Regard Prefab Rotation, if enabled, the final scale will be multiplied with the initial prefab scale. Useful for placing small objects that were upscaled in a prefab.
- **Instantiate Clones:** by default in editor mode new objects will be created maintaining the prefab connection, in playmode objects are instantiating using prefab clones. Enabling this will make MM instantiate objects using prefab clones both in editor and playmode. This will increase the apply speed in editor, but the objects instantiated will loose prefab connections.

Debug:

Enables #WDEBUG compile directive, forcing execution of all of the code within this directive. It could be drawing of debug gizmos, special modes, etc. It's mainly used for Voxeland creation and debugging. When a user will encounter some bugs the maker of Voxeland might ask him to take a screen with a debug mode. By the default it's better disable it, especially for when building a final product.

This setting is shared between Voxeland, MapMagic and Erosion Brush.

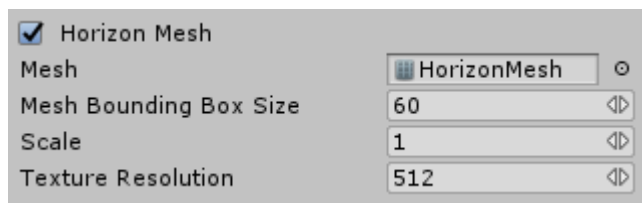
Switching it requires scripts re-compile.

Horizon Mesh

In most cases the pure voxel terrain requires too much overhead to be created and displayed. Generally it's not needed to display voxels in a distance. Along with a voxel engine, Voxeland has a built-in feature to display conventional heightmap terrain. It's called **Horizon Mesh**.

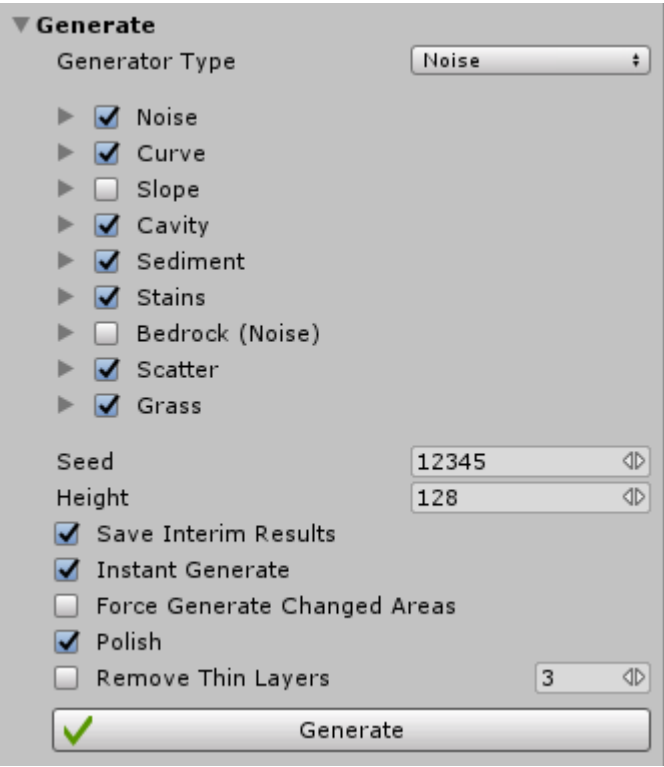
Note that it's still a mesh terrain, not the standard Unity terrain. It's build and displayed faster than the standard one, and way faster than voxel terrain.

Horizon Mesh works only with the Infinite and Limited Dynamic terrain modes.



- **Mesh:** a planar mesh used to determine the horizon topology. In most cases you will not need to change it, but in case you do here are mesh requirements: ** vertex and triangle count should be less than 64K; ** should be made in coordinate system where each chunk is 1 unit. So, if you need horizon with a size of 100 chunks, it should have 100 units. ** Y facing up ** planar mesh: all of the verts have zero height
- Mesh Bounding Box Size: the size of the horizon mesh, in chunks, to avoid re-calculating it. The size of the default mesh is 60 chunks (i.e. 60 units in mesh).
- Scale: horizon mesh scale factor. You can adjust the size/density of the default mesh using this value.
- Texture Resolution: horizon mesh heightmap and normal map sizes. The heightmap is used to displace mesh vertices, while the normal is used to light the mesh. It does not use vertex normals, using normal map instead, so increasing the resolution can gain relief detail.

Generate



See [Build and Generate difference](#) for guidance.

Special algorithms used to generate voxel data: i.e. determine the terrain form: mountains, plains, forests, etc. Terrain generate is essential for the infinite terrain (it would be silly if all of the infinite terrain will be planar), but also could be handy to create limited terrain blank or prototype.

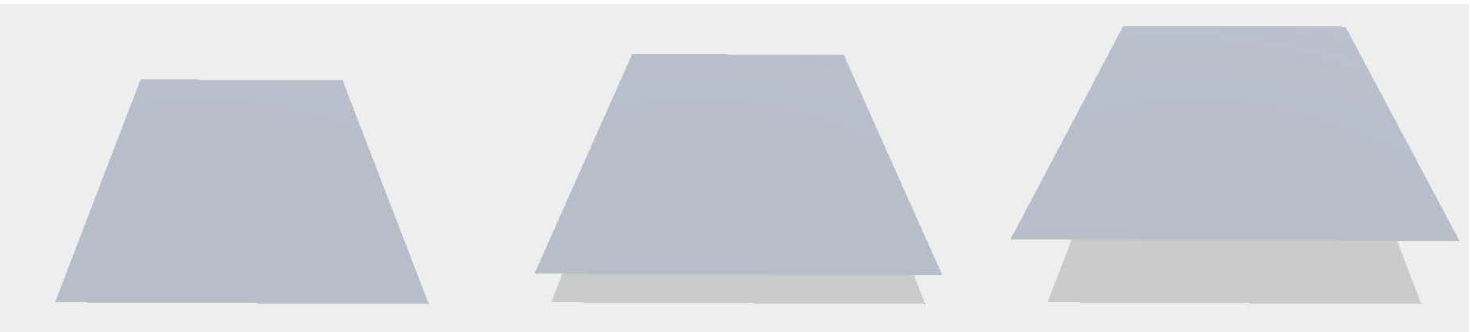
Generator Type

Algorithm that should be used for terrain generate:

Planar

Creates a map filled with a given value. This is the most simple generator, used to create flat terrain filled with blocks up to some specified level.

- Value: the map will be filled with this value on generate. Value ranges between 0 (no fill, transparent fill or zero ground level depending on the output where it will be used) and 1 (fully opaque fill or maximum terrain height).



Value: 0, 0.5, 1

Noise

Creates a complex terrain of different block types based on the initial noise algorithm. Some of the [MapMagic generators](#) are used to generate terrain, they are stacked together in a fixed order. You cannot create caves or other advanced stuff with it, but it could be set up to create simple, but decent terrain.

Use checkbox next to the generator name to enable/disable the generator.

Click the chevron (►) button to roll up the generator properties.

MapMagic

The [MapMagic](#) graph is used as a terrain generator. Here you can link any generators in any order, create caves, overhangs and other advanced stuff. MapMagic World Generator is required to use this mode.

Note that assigning a pure MapMagic graph in this slot will not make it work with Voxeland. You've got to use a special Voxeland Outputs for that. In some cases graph conversion is not possible at all: [FAQ: Is there a way to change graph to work with Voxeland](#).

- **Graph:** graph asset used to generate terrain
- **Create:** creates and assigns a new graph
- **Store:** saves the graph and assigns saved file in a Graph slot. All of the changes made to the graph will change the saved file.
- **Save as Copy:** saves current graph, but does not assign it. The saved file will not be changed when modifying currently assigned graph

A [video tutorial](#) that describes the principles of the MapMagic/Voxeland generator.

Heightmap:

Use it if you are going to create a heightmap-based terrain.

- **Texture:** a texture in any format Unity supports.

Note that the texture should have Read/Write enabled toggle checked in Advanced settings.

- **Reload:** by the default Voxeland reads the texture and stores it in the internal format to access it in multithreaded mode. So the texture changes do not take an effect unless Reload button is pressed (if Reload Each Generate is turned off).
- **Reload Each Generate:** reloads the texture before each generate. This will slow down generate performance a bit, but will take into account all of the texture changes.
- ****Auto Scale to terrain size:** will set the Size value to make the texture fit into a limited terrain size.
- **Scale:** when set to 1 each texture pixel corresponds to the voxel block (or, more precisely, a column of blocks). Increasing this value make blocks share the same pixels. In a size of 2 four blocks (2*2) are set by a single texture pixel.
- **Offset:** moves the heightmap apply position

- **Wrap Mode:** sets the texture tiling effect:
 - Once: will repeat texture once and set all of the out-of-texture pixels to zero.
 - Clamp: will repeat texture once and set all out-of-texture pixels to the closest texture values
 - Tile: will repeat texture endlessly (obviously, your texture should not have tiling seams to tile seamlessly)
 - Ping-pong: will mirror texture and repeat texture+mirrored texture endlessly.

Generate properties

- **Seed:** global seed value. This number is used to initialize pseudo-random generators. Note that pseudo-random generators use their own Seed parameter, so the final result depends both on the Global Seed and Generator Seed parameter. Change this value to achieve a different look for all the generators.
- **Height:** the height value of 1 in generators (of "white color" in heightmap). This isn't the maximum terrain height, it's just a range used to generate terrain. You can manually add blocks above that level.
- **Save Interim Results:** Voxeland/MapMagic keeps the output results for each chunk for each generator, so on generator change there is no need to re-generate everything: only the changes in the generator and its dependences are calculated. When the parameter is disabled Voxeland/MapMagic will clear all generator maps after generation has been completed. Disabling this option can reduce the memory footprint, but each time any of the generators change the entire graph will be re-generated from scratch, which can take some time depending on your graph's complexity.
- **Instant Generate:** when this parameter is enabled there is no need to press the "Generate" button after changing any of the generator parameters: Voxeland triggers the change and forces generation automatically as you make changes to each node in the graph.
- **Force Generate Changed Areas:** will overwrite all of the "pinned" areas on the next generate. All of the manual changes will vanish.
- **Polish:** will remove all of the small peaks and hollows after generate:



- **Remove Thin Layers:** the minimum allowed terrain thickness. Use this to remove blocks floating in air after cave generate with MapMagic.

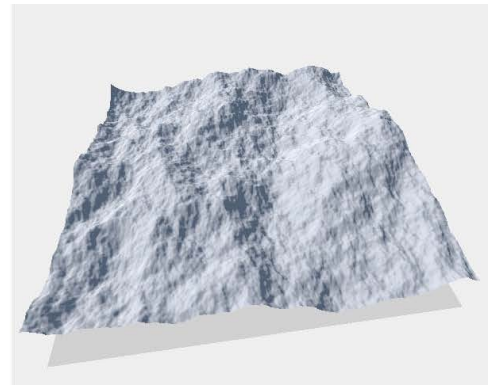
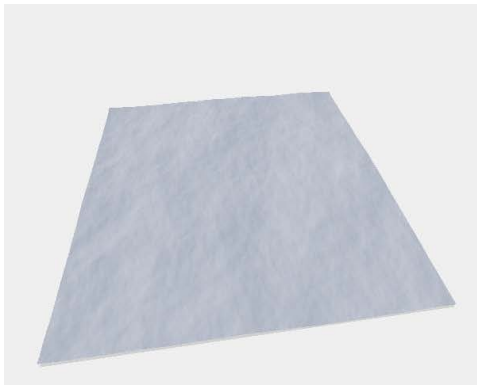
Generators

Note that most of the generators are shared with MapMagic, so MapMagic terrain is used at the images below. Voxeland generators produce exactly the same result, but note that Voxeland cannot display the smooth height gradients, it will have more "stair-like" effect. The original MapMagic generator names are given in parentheses.

Noise:

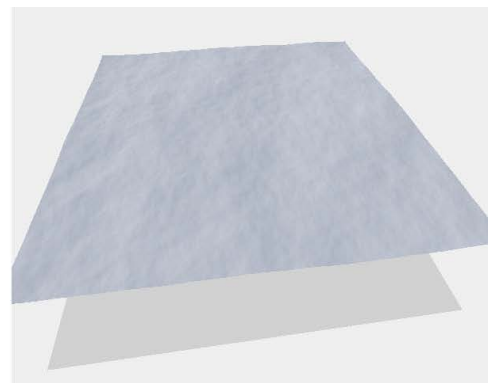
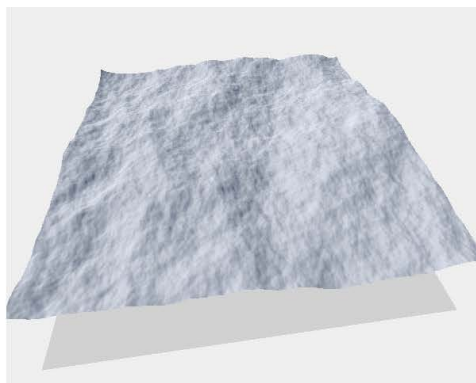
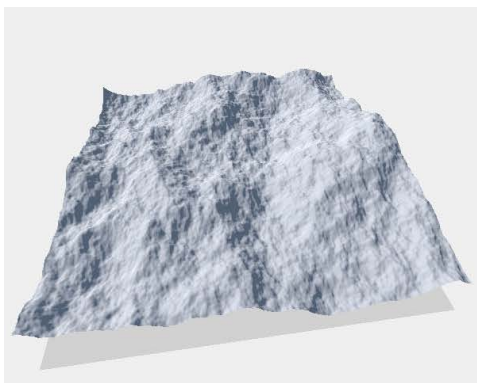
The basic noise generator used to create initial heightmap.

- Seed - a number used to initialize pseudo-random noise generator. If two generators use equal seed numbers the resulting pattern will be the same.
- High (intensity)- sets the highest noise value. As the name says, consider it as the noise intensity.



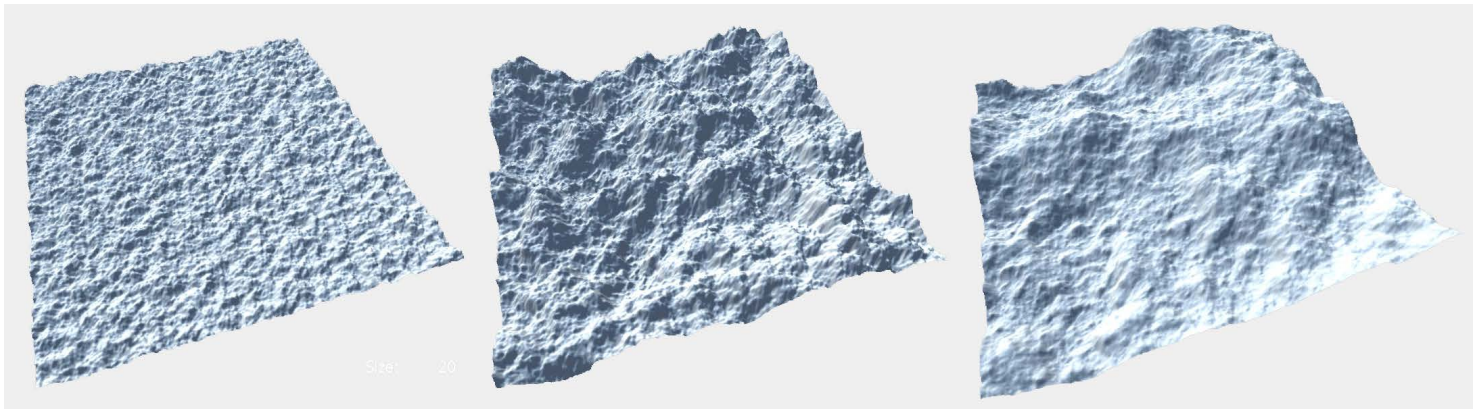
High (intensity): 0.1, 0.5, 1

- Low - sets the minimum noise value. All of the noise values generated lay within the Low-High range. You can increase this value to make the noise less contrast, or lower it below zero level to make the noise sharp.



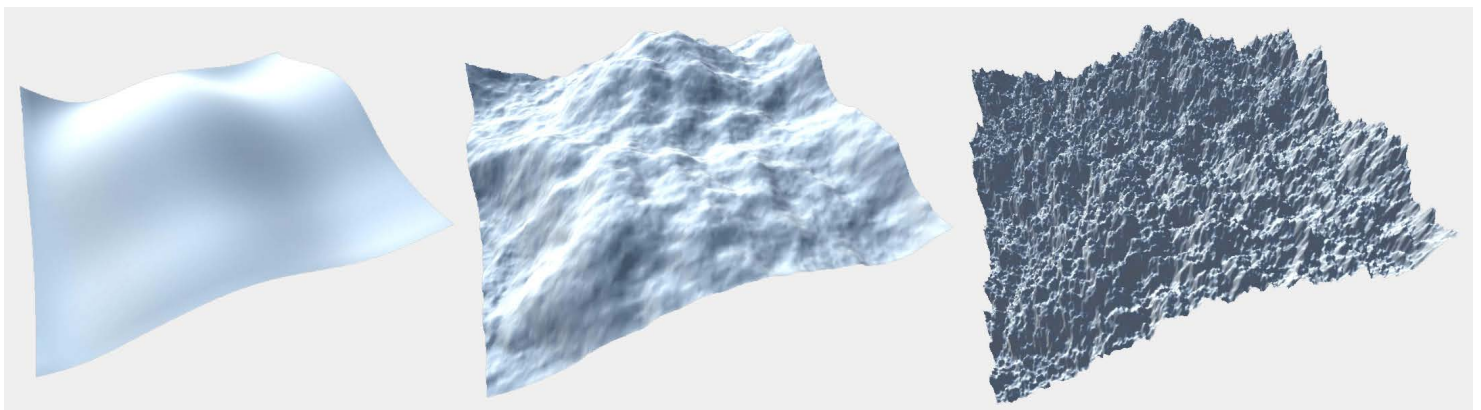
Low: 0, 0.5, 0.9

- Size - this parameter determines the size of the biggest fractal. Noise above this value ceases to be fractal being a standard perlin noise. Lower size values result in a very homogeneous and predictable noise, while high values can create diverse noise and scenic heightmaps.



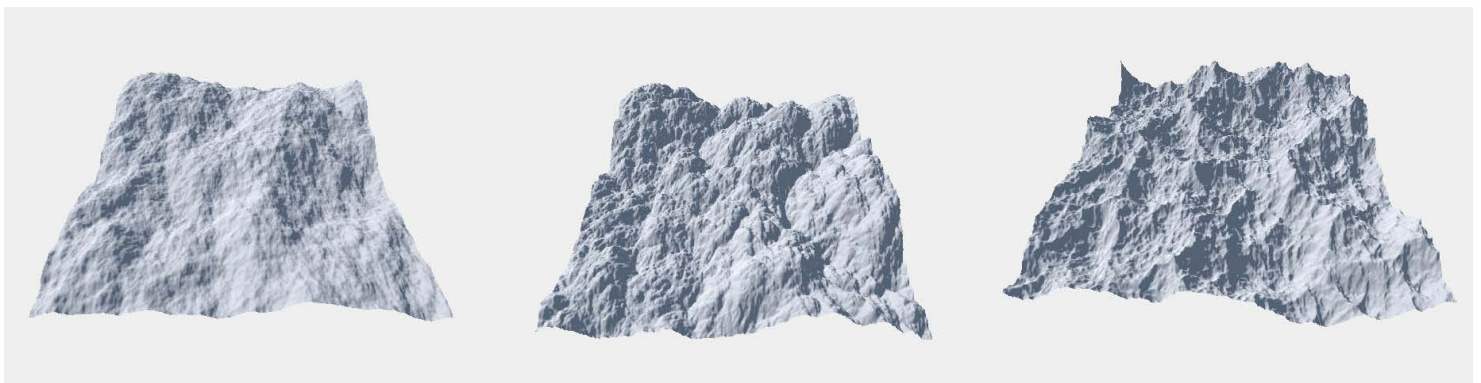
Size: 10, 100, 200

- Detail - this parameter determines the big and small fractals' bias. When the parameter is higher than 0.5 then small fractals have greater impact, which results in a more 'noisy' map. When parameter is below 0.5 small fractals are less significant than the big ones, which results in a smoother noise.



Detail: 0, 0.4, 0.6

- Turbulence - creates "bubbles" (positive) or "ridges" (negative).



Turbulence: 0, 1, -1

- Offset - this parameter defines the noise pattern position, shifting it along the x and z axes. These values can be positive or negative. Offset applies to all of the terrains and sums with the terrain position, so the noise maps continue seamlessly on all the terrains.

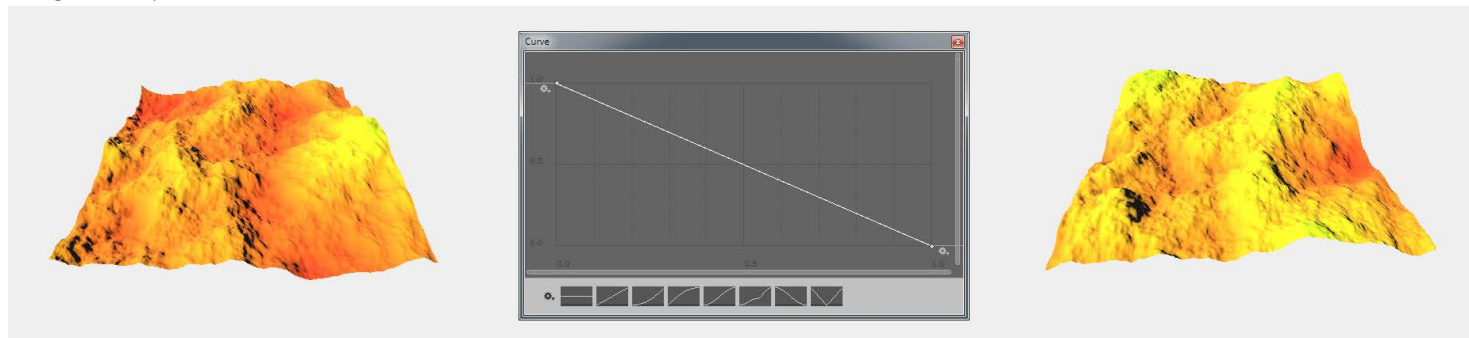
Curve:

Adjusts heightmap values using a user-defined curve. It works similar to the curves adjustment in graphics editors like Photoshop. The horizontal axis of the curve chart is the input value, vertical axis is the output value.

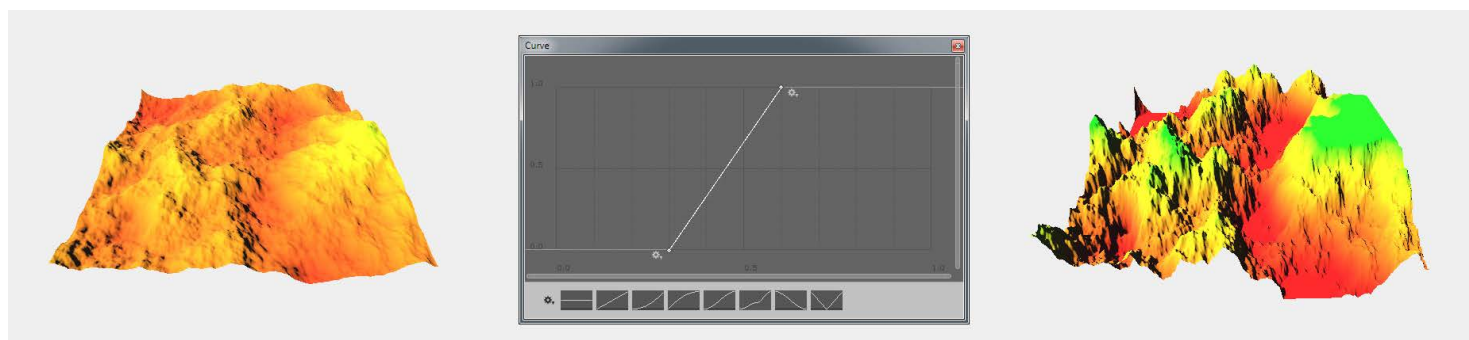
This generator uses Unity's animation curve interface for curve editing, so working with the Curve Generator is done the same way. Keys and tangents can be dragged with the left mouse button, the right mouse button is used to create new keys and access key properties.

- Curve field: displays a curve preview. Clicking on it will open up a Curve Editor window - the standard Unity curve editing tool (see Unity manual: <https://docs.unity3d.com/Manual/EditingCurves.html> for more information).
- Range: by default the whole value range from 0 to 1 is used for editing. For more precise editing, minimum and maximum values could be set for input and output. The left row sets min and max for the input values (curve horizontal), while the right row sets output min and max (vertical).

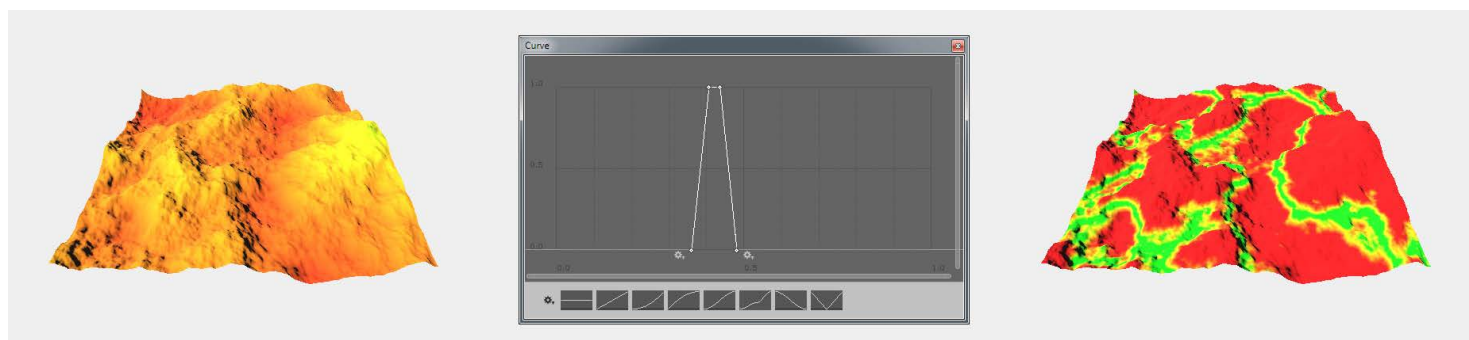
Usage examples:



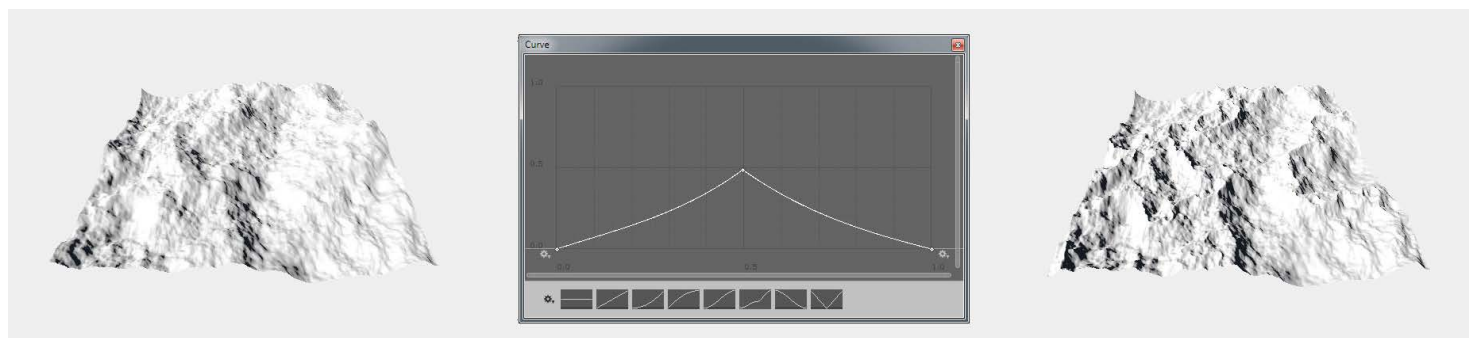
Invert



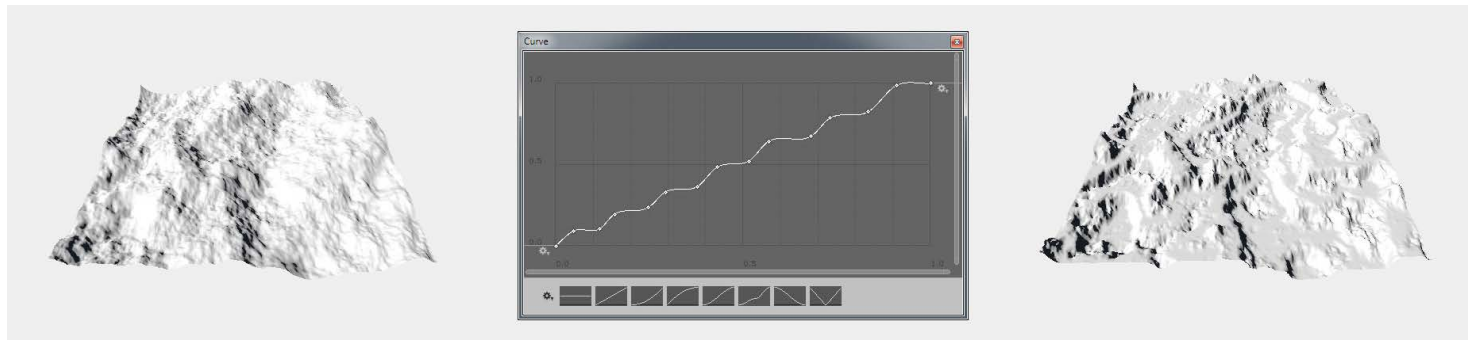
Contrast



Select Range



Turbulence Ridge

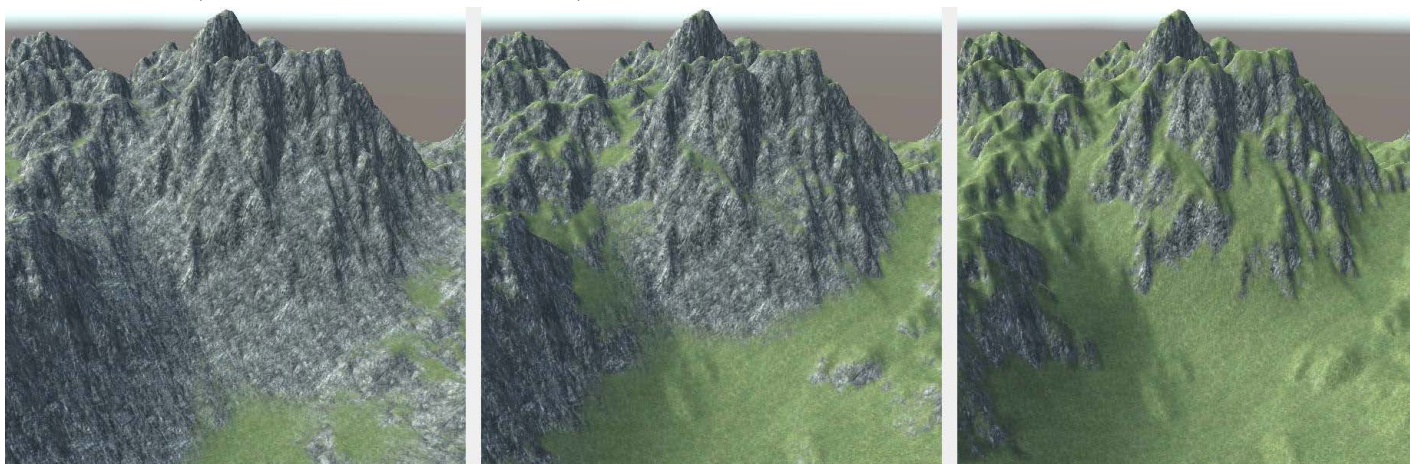


Terrace

Slope:

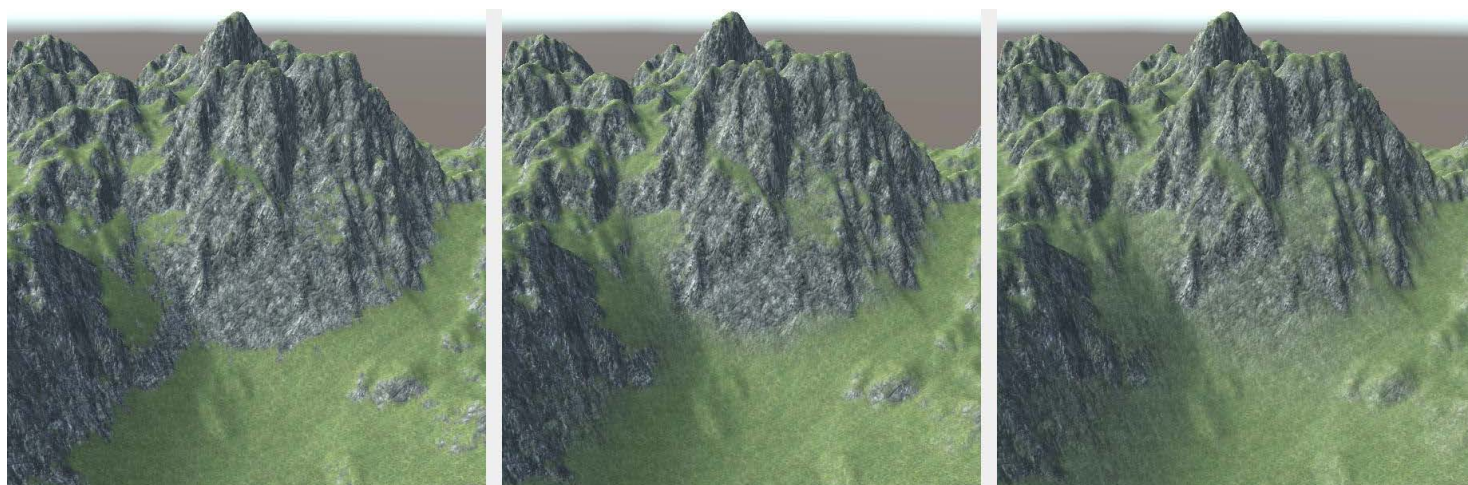
Generates the height difference map. For each map pixel it calculates the average height delta to four nearby pixels. Could be useful for separating horizontal and vertical surfaces (to fill them with grass and cliff respectively).

- Steepness: defines the incline range that will be filled with slope map (in degrees). Minimum parameter sets the start incline of the stop, maximum - the end of the slope.



Steepness: 5-90, 35-90, 80-90

- Range: how gradual will be a slope transition (in degrees)

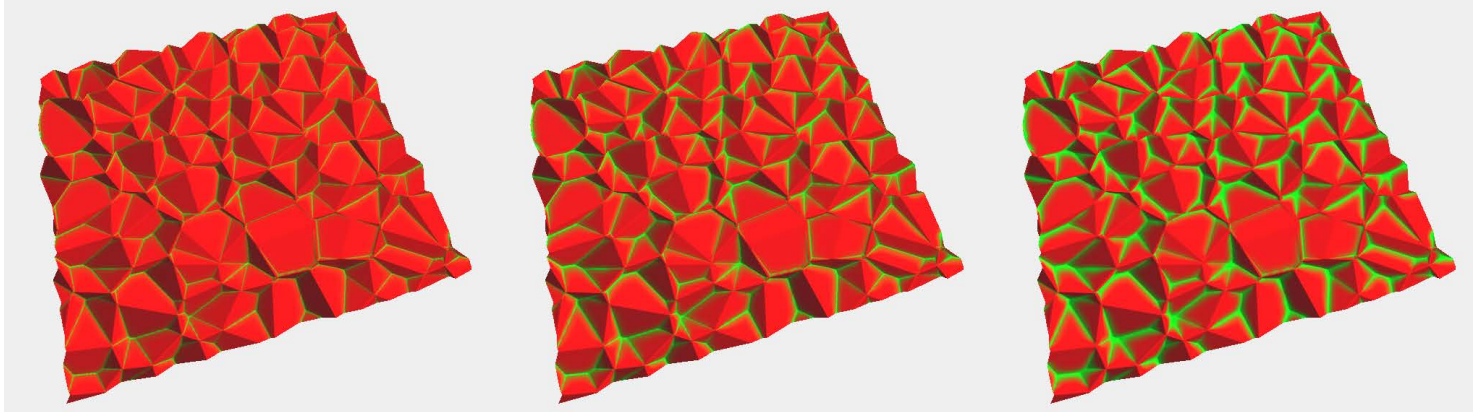


Range: 1, 5, 15

Cavity:

Generates the maps of concavity and convexity. All of the bulges and cambers are regarded as convex and the hollows are regarded as concave. Note that output maps do not intersect: one pixel can not be convex and concave at the same time.

- Intensity - the amount of influence of the generator on the input map. When the value is set to 0 the generator effect is not visible. In other words, it is the generator's effective opacity.
- Spread - since really curving terrain areas are rather rare, the Cavity Generator can expand resulting information on more level areas.



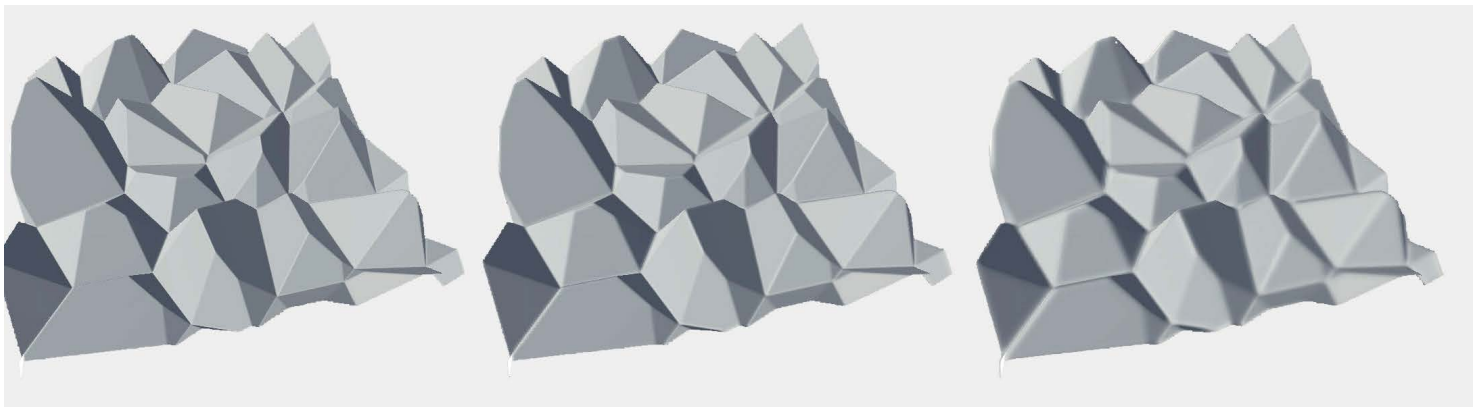
Spread: 1, 2.5, 5

- Normalize Convex+Concave: ensures that all of the convex values combined with concave ones are always equal to 1. If this parameter is off only convex or concave map is calculated, its antipode is not taken into account. Turn it off to perform rude but faster calculations.

Sediment (Blur)

Smooths the heightmap, then applies the smoothed map using the MAX algorithm: for pixels (block columns) where original height is bigger than the smoothed one - nothing changes. For pixels where smoothed height is bigger - it sets smoothed value, filling the gap with a block type selected.

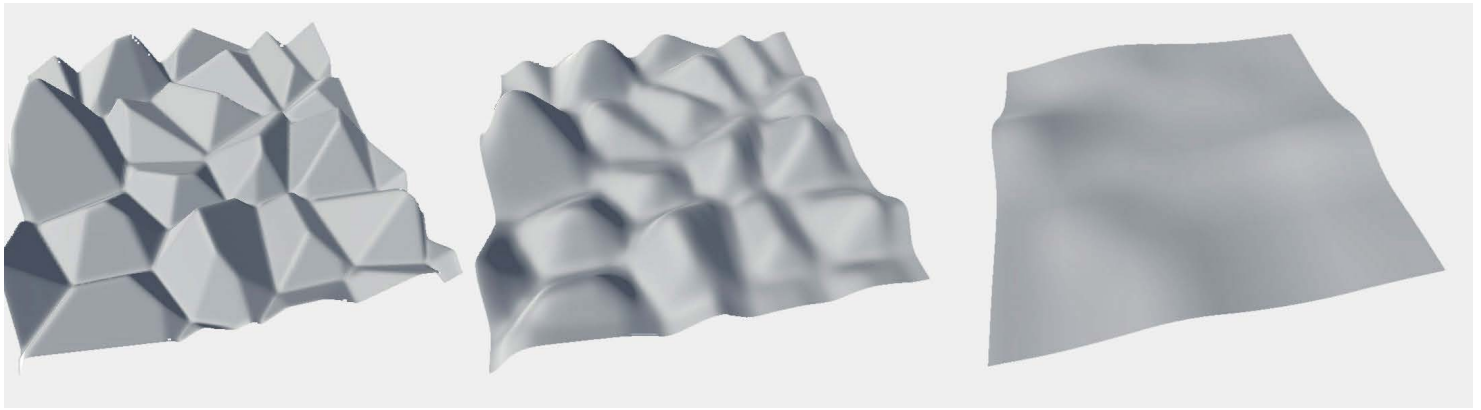
- Intensity: amount of blur applied per iteration.
- Iterations: number of blur iterations applied. Note that increasing the iterations count will reduce blur performance. The maximum reasonable amount of iterations is about 10, if you want more map smoothness use the Loss parameter.



Iterations: 0, 1, 10

- Loss: forces the generator to skip some of the input pixels. This will result in a bit more 'pixelated' look, but will give the map extra smoothness. A value of 1 means that no pixels will be skipped. The Loss parameter has virtually no effect on performance so it could be used to create extremely smooth maps. Use it together with high Intensity and Iterations

counts to avoid noticeable pixelization.



Loss (with Iterations = 10): 1, 20, 60

Do not try to make your map extra-blurry by increasing Iterations value only. This will result in poor Blur performance. Use Loss value instead.

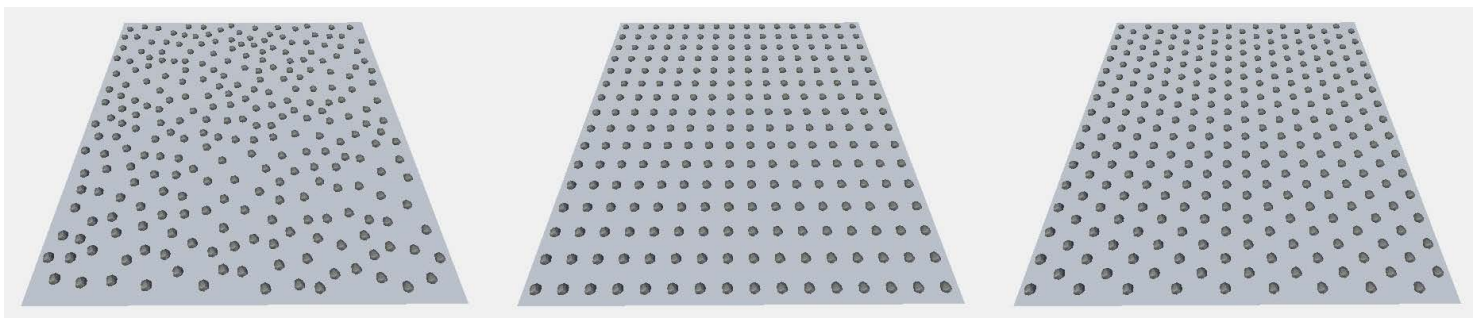
Bedrock (Noise)

One more noise generator to create an underground layer of the other (for example, understandable) stratum. All of the noise properties are used.

Objects (Scatter)

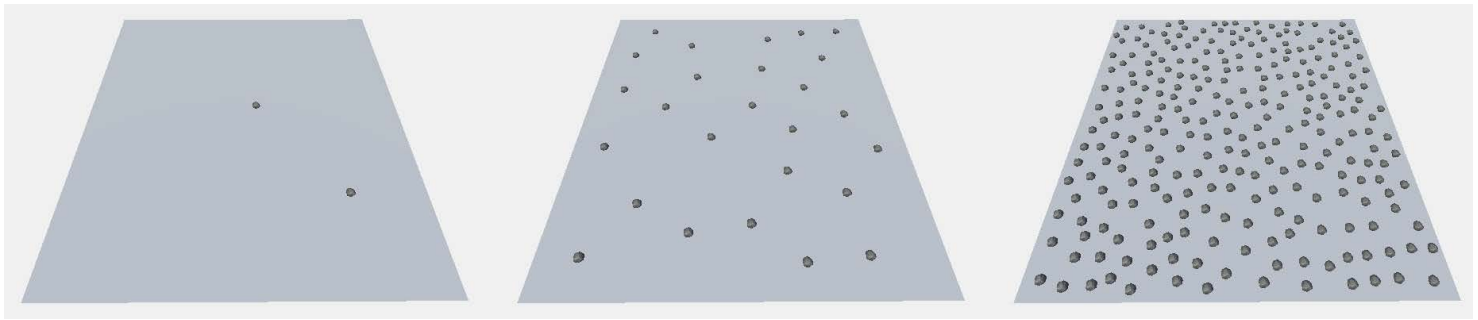
Scatters objects in the terrain area, randomly creates new object positions.

- Algorithm: determines the pattern objects are placed before randomizing:
 - Random: no pattern, pure random
 - Square Cells: objects are placed using square grid
 - Hex Cells: objects are placed using triangle (hex) grid



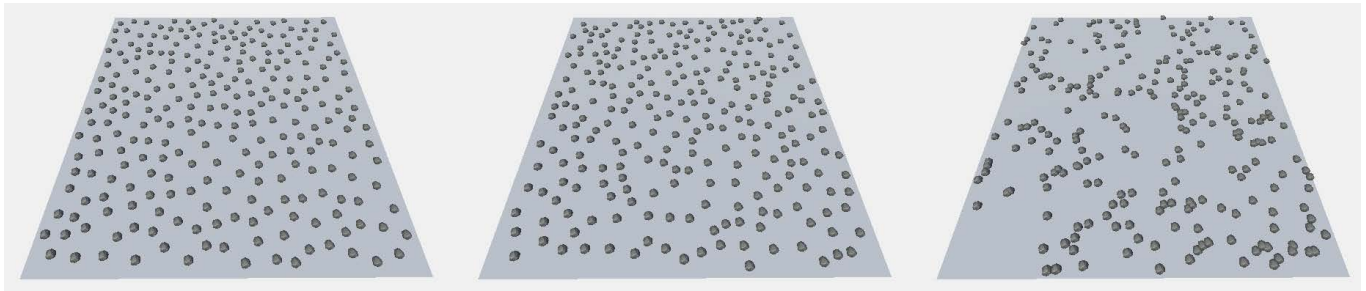
Algorithm: Random, Square Cells, Hex Cells

- Density: the quantity of scattered objects per a square kilometer (100*100 units). Note that when using a probability map the final count is less than the value because of generator's probability occlusion.



*Density (terrain size is 500*500 meters): 10, 100, 1000*

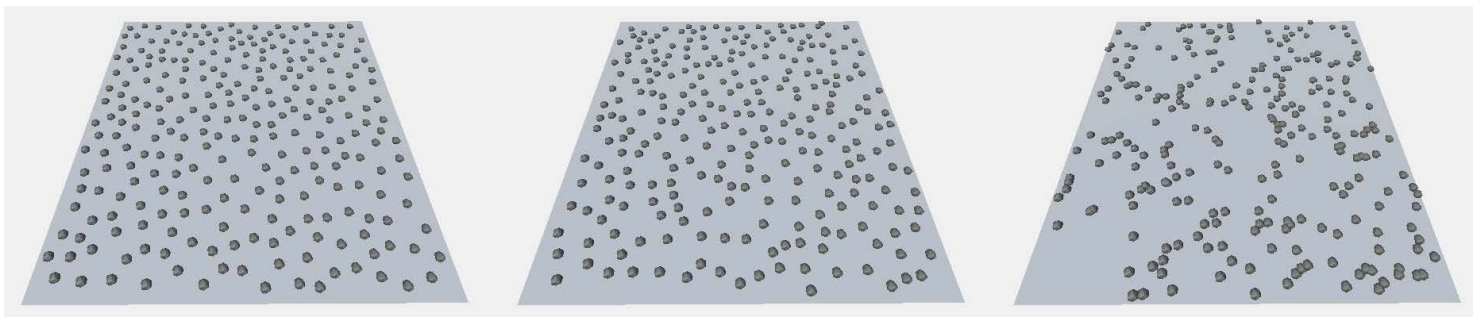
- Uniformity: determines how evenly objects are distributed along the terrain.
 - When using square or hex cells algorithm this parameter determines objects offset range after they have been placed on terrain



Uniformity (square cells): 1, 0.7, 0.3

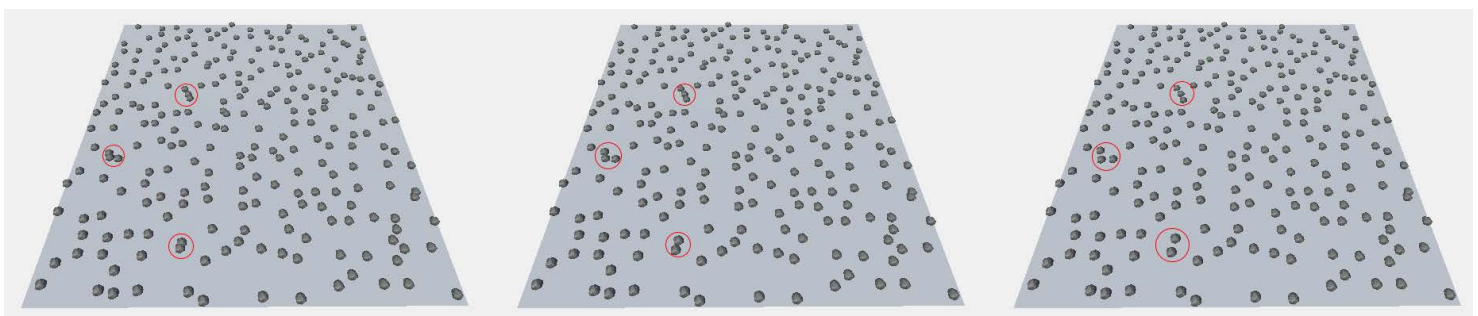
- When using Random algorithm this parameter determines the number of iterations to find equidistant object position.

Note that using high Uniformity value with high objects count in Random mode can result in greatly increased generate time. Usually there's no point in using Uniformity more that 0.1 in this case. Consider switching to Square or Hex Cells mode when the objects count is <1000.



Uniformity (random cells): 1, 0.1, 0.001

- Relax: pulling apart closely located objects after the random has been applied. Useful at low Uniformity values.



Relax: 0, 0.25, 0.5

Serializing voxeland data

In the vast majority of the projects you will need to have an ability to store all of the changes to send them via the net or to use in the other game session. That is possible to do with the `Data.ToArray` and `Data.FromArray` functions.

```
public byte[] ToArray (bool pinnedOnly=false)
```

Returns the terrain block data stored in a byte array. Note that it stores only block type information, and does not contain any data related to block properties and textures, it does not even knows the total blocks count in Voxeland object. Consider it as a compressed 3-dimensional array of **block numbers**.

Enabling `pinnedOnly` parameter will store only that areas that have any manual change, leaving the unchanged areas to regenerate on load. Disabling it will store all of the currently generated areas.

```
public void FromArray (byte[] bytes)
```

Will convert previously saved bytes array back to the Voxeland data. This will erase all of the currently generated areas, replacing them with the loaded ones.

Note that all of the loaded areas will be marked as "pinned" (i.e. changed), and when `ToArray` will be called again will be saved regardless of `pinnedOnly` value.

Serializing to file

Here is an example of saving the data to file and loading it:

```
public void Save (Voxeland5.Voxeland voxeland)
{
    byte[] byteArray = voxeland.data.ToArray();
    System.IO.File.WriteAllBytes("Assets\\SavedVoxelandData.txt", byteArray);
}

public void Load (Voxeland5.Voxeland voxeland)
{
    byte[] byteArray = System.IO.File.ReadAllBytes("Assets\\SavedVoxelandData.txt");
    voxeland.data.FromArray(byteArray);

    voxeland.Rebuild(); //optionally, to make the loaded data be displayed
}
```

Serializing via the net

You can send the byte array via the net too, for instance, to provide all the changes made in terrain to newly connected players (just an example code, it could be obsolete):

```
public void SendVoxelandData (Voxeland5.Voxeland voxeland, NetworkView networkView)
```



```
{
    byte[] byteArray = voxeland.data.ToByteArray();
    networkView.RPC("RecieveVoxelandData", RPCMode.Others, byteArray);
}

[RPC]
public void RecieveVoxelandData (byte[] byteArray)
{
    voxeland.data.FromByteArray(byteArray);

    voxeland.Rebuild(); //optionally, to make the loaded data be displayed
}
```

Editing terrain with scripts

struct CoordDir : coordinates of the block/face. Contains 3 ints (**x**, **y** and **z**) of the block coordinates and a **dir** byte that determines the face direction: front, back, up, down, left, right (in that order). The face direction of 7 means that the face does not exist.

If you have a single Voxeland object in scene then the easiest way to get is **Voxeland.instances.Any()**

Get coordinates by ray

CoordDir PointOut(Ray aimRay) : If ray intersects terrain will return intersection block coordinates. Will return CoordDir(0,0,0,7) if no intersection detected.

This code will return the block coordinate of the camera > mouse cursor ray:

```
Voxeland voxeland = Voxeland.instances.Any();
Ray aimRay = Camera.main.ScreenPointToRay(Input.mousePosition);

CoordDir aimCoord = voxeland.PointOut(aimRay);
```

Get block type at given coordinates

byte Data.GetBlock(int x, int y, int z) will return the block type number at x,y,z.

BlockType voxeland.landTypes.array is an array of the block types used by current Voxeland object

You can get currently aimed block type name (or specular value, for instance) with:

```
Voxeland voxeland = Voxeland.instances.Any();
Ray aimRay = Camera.main.ScreenPointToRay(Input.mousePosition);
CoordDir aimCoord = voxeland.PointOut(aimRay);

string blockTypeName = "None";
float blockTypeSpecular = 0;

if (aimCoord.exists) //i.e. id dir==7
{
    byte blockTypeNum = voxeland.data.GetBlock(aimCoord.x, aimCoord.y, aimCoord.z);
    BlockType blockType = voxeland.landTypes.array[(int)blockTypeNum];

    blockTypeName = blockType.name;
    blockTypeSpecular = blockType.specularM;
}
```

Highlighting selected block



`void Highlight (CoordDir center, Brush brush)` will draw a highlight of a given brush at the center coordinates.

This will highlight the selected block using currently selected brush type:

```
Voxeland voxeland = Voxeland.instances.Any();
Ray aimRay = Camera.main.ScreenPointToRay(Input.mousePosition);
CoordDir aimCoord = voxeland.PointOut(aimRay);

if (aimCoord.exists) voxeland.Highlight(aimCoord, voxeland.brush);
else voxeland.highlight.Clear();
```

Edit terrain

```
public void Alter (
    CoordDir center,
    Brush brush,
    EditMode mode,
    int landType=-1,
    int objectType=-1,
    int grassType=-1)
```

Will apply **brush** to the terrain using current **mode** at the given **center** coordinate.

One of the types: landType, objectType or grassType should be 0 or larger. This way way Voxeland determines what exactly should be changed. The value should be equal to the type number we are going to add (or perform other edit).

For edit modes reference see [Brush: Edit modes](#).

This will add the first (0) block type at the mouse cursor on mouse click:

```
Voxeland voxeland = Voxeland.instances.Any();
Ray aimRay = Camera.main.ScreenPointToRay(Input.mousePosition);
CoordDir aimCoord = voxeland.PointOut(aimRay);

if (Input.GetMouseButtonDown(0) && aimCoord.exists)
    voxeland.Alter(aimCoord, voxeland.brush, Voxeland.EditMode.add, 0, -1, -1);
```

Complex terrain aiming, highlight and edit

This is the code from VoxelandController.cs demo script. It is used to edit the terrain in the included scene - note that "Edit in playmode" is disabled here.

```
//reading controls
bool leftMouse = Input.GetMouseButtonDown(0);
bool shift = Input.GetKey(KeyCode.LeftShift) || Input.GetKey(KeyCode.RightShift);
bool alt = Input.GetKey(KeyCode.LeftAlt) || Input.GetKey(KeyCode.RightAlt);

//getting edit mode
Voxeland.EditMode editMode = Voxeland.EditMode.none;
if (leftMouse && !shift && !alt) editMode = Voxeland.EditMode.dig;
```

```
else if (leftMouse && alt) editMode = Voxeland.EditMode.add;
else if (leftMouse && shift) editMode = Voxeland.EditMode.replace;

//getting aiming ray
Ray aimRay;
if (cursorLocked) aimRay = Camera.main.ViewportPointToRay(new Vector3(0.5f,0.5f,0.5f)); //use your own bool variable to determine if we should use cursor or screen center when cursor is locked
else aimRay = Camera.main.ScreenPointToRay(Input.mousePosition);

//aiming terrain
CoordDir aimCoord = voxeland.PointOut(aimRay);

//highlight
if (voxeland.highlight!=null)
{
    if (aimCoord.exists) voxeland.Highlight(aimCoord, voxeland.brush, isEditing:editMode!=Voxeland.EditMode.none);
    else voxeland.highlight.Clear(); //clearing highlight if nothing aimed or voxeland not selected
}

//altering
if (editMode!=Voxeland.EditMode.none && aimCoord.exists)
{
    voxeland.Alter(aimCoord, voxeland.brush, editMode,
        landType:voxeland.landTypes.selected,
        objectType:voxeland.objectsTypes.selected,
        grassType:voxeland.grassTypes.selected);
}
```


General F.A.Q.

Is there any way to change standard MapMagic graphs to make them work with Voxeland?

Unfortunately, I don't believe it's possible. Textures Output cannot be just replaced with Voxeland Output node. Voxeland Output uses absolutely different (and more complex) algorithm, it does not just blends layers in a Photoshop manner, but it creates layers in 3D space. This requires not only the significant graph changes, but changing the whole approach to the graph creation. That's why I'm not converting Island or Demo scenes to be used with Voxeland - they will require the whole graph created from scratch, and it will not have much in common with the original one.

You can find MM+Voxeland tutorial here, JIC: <https://youtu.be/jlLALBsAUZY>

How can I know that Voxeland is currently generating?

You can use `ThreadWorker.IsWorking("Voxeland")` method to determine if Voxeland is still generating.

I want to add durability to blocks so that mining should not be instant

You've got to add some "life timer" that would start when the player press mouse aiming the block, and resets on changing the block or releasing mouse. Note that it should be not a "per block" timer, but a single timer for all of the game.

What is the area data structure exactly?

Area is not a pure 3D array, but rather complex structure. It is split in horizontal slices called Lines. I guess "Slice" would be a better name, but anyways. Each "Line" has only one block in width (along the Z-axis), has blocks in X-axis and almost infinite number of blocks up in height (Y-axis). Each line is divided in Columns - a vertical 1-dimensional sets of blocks.

In the early Voxeland versions each Column had two arrays (lists, in fact) - one for the block types, and one for the number of consequent blocks of the same type (i.e. layer depth). So, the column of the blocks 1,1,1,1, 2,2,2, 3,3,3,3,3 became 1,2,3 and 4,3,5. In current Voxeland version columns does not have individual lists, all of the lists are stored in a Line consequently. So, if area had 3 columns with blocks types 1,2,3,4 and 1,2,3 and 3,2,1 the final Line array will be 1,2,3,4, 1,2,3, 3,2,1 (and the same approach is used for layer depth). Each column has a "start" value to know where it should start in Line arrays, and "count" value to know when to stop reading array. So, for the column 2 in this example start will be 4, and count is 3.