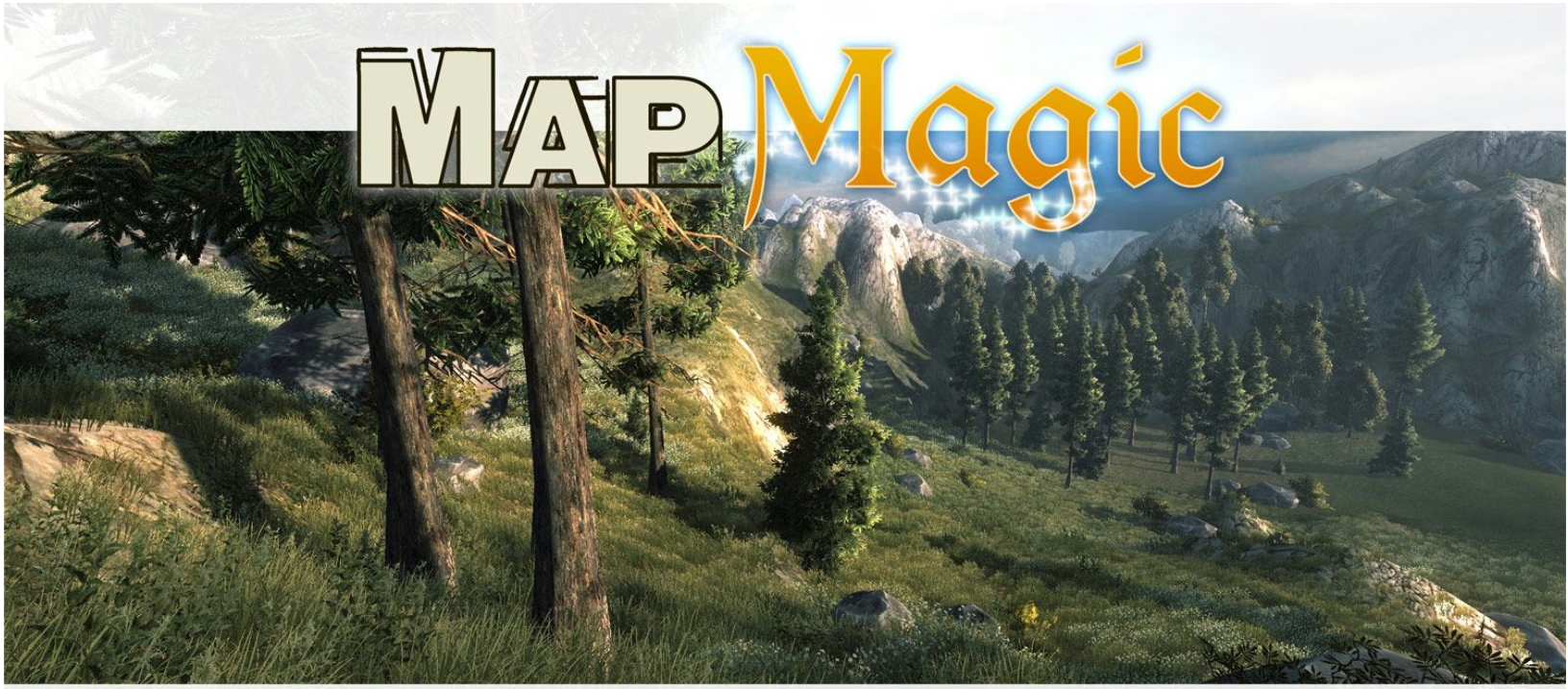


The node based automatic map creator for Unity3D

Table of contents:

Welcome to MapMagic - a platform for terrain creation and automatic game object placement.

This tool uses a node-based visual scripting interface to determine creation logic. Each node represents a separate algorithm called a “generator”. Examples of generators include: noise, voronoi, blend, curve, erosion, object scatter, forest, etc. All nodes are presented field called the “graph”.

Each generator node has input and/or output connectors. A generator takes some map or object information as inputs, processes it and returns the result as outputs. Normally all of the nodes in the graph are interconnected in a desired sequence. Currently there are two types of inputs/outputs:

-  Map inputs/outputs - stores information in the form of a square matrix. Think of it as a 16-bit, 1-channel map.
-  Object inputs/outputs - some kind of coordinate list. Each list member contains additional information like rotation along Y axis or uniform scale.

The chains of generators should end in one of the “Output Generators”. There can only be one of these generators per graph. There are 5 types of output generators:

- Height output - specifies terrain height and relief information.
- Textures output - colors the terrain with assigned textures.
- Objects output - instantiates the prefabs in the scene.
- Trees output - instantiates the prefabs as tree instances.
- Grass output - draws grass on terrain.

MapMagic can work with multiple terrains (that’s what gives MapMagic the ability to create infinite worlds). Terrains can be divided into two types:

- Pinned terrains - These are manually selected in editor. They are always visible in Playmode, no matter how far away the main camera is. They are displayed in Editor mode as well.
- Procedural terrains - These are automatically created in Playmode when the camera gets close to them and are destroyed when camera is far enough away. Procedural terrains appear only when the “Infinite land” feature is turned on. They are not visible outside of Playmode.

Quick start guide

To create a MapMagic object click the GameObject menu -> 3D Object -> MapMagic. A new game object named “MapMagic” will appear in the hierarchy menu. It already has the initial pinned terrain and three example generators, so the terrain is not flat. Clicking this object will show basic settings in the Inspector.

The “Pin Terrain” button allows pinned terrains to be selected in the Scene view. Left-clicking on an empty area will pin a terrain, and it will appear in a moment, after it’s been generated. Left-clicking on an already pinned terrain will unpin it. To exit pinning mode click on the “Pin Terrain” button once more.

Any change made to pinned terrains will be re-written on the next generate. To prevent this use the terrain lock by pressing the “Lock Terrain” button and selecting a pinned terrain that should be locked. When using a locked terrain try not to modify a heigh greatly, otherwise you can get a noticeable borders between unchanged locked terrains and their neighbours with the new heightmap.

The “Show Editor” button will open up the editor window. Here you can see three initial generators: Noise, Curve and Height (output generator) connected together. The Editor window can be scrolled using the middle mouse button. To zoom in or out use the scroll wheel. To drag a generator left-click anywhere on a generator window.

Changing any of the generator parameters will force the generator and it’s dependent generators to re-calculate. Once the generators’ results are ready the terrain will be changed.

The Generate button in the MapMagic window will clear all the generators, forcing them to be re-calculated from scratch.

Note that the Generate button icon changes its appearance depending on the current generation state:

- ✔ Green “Ok” mark is displayed when all terrains have been generated successfully and all of them are up-to-date.

-  Grey “Generating” mark(animated) is displayed when at least one terrain is currently in the progress of being generated.
-  Red “Error” mark means that the generator graph has connection error(s): either mandatory connections are empty, or connections have the wrong type, or the graph has an infinite loop. Error connections will be displayed in red on graph.
- When nothing is displayed beside the “Generate” button then terrains have not yet been generated. This is common when the “Instant Generate” feature is turned off in the settings.

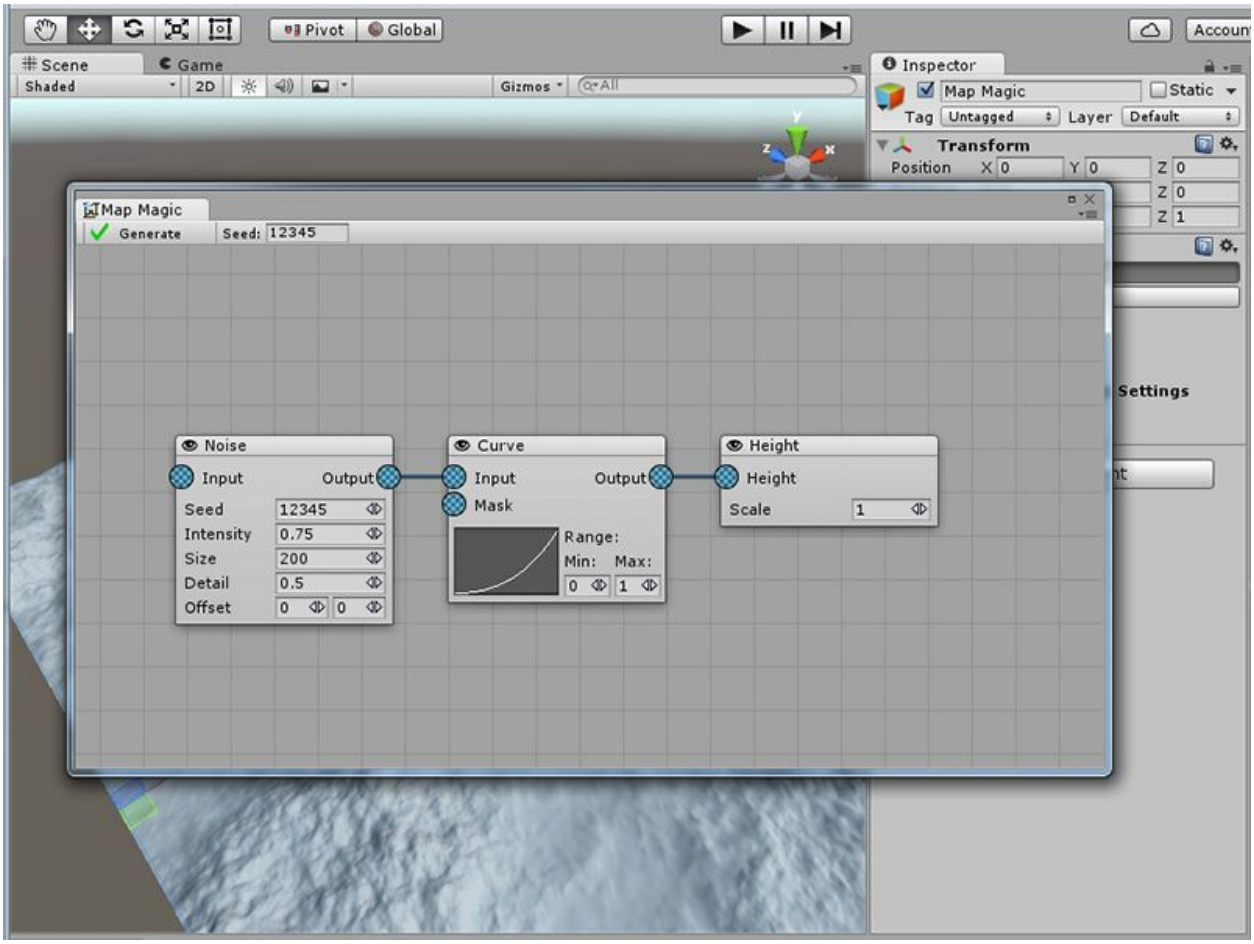
Next to the generate button is the Global Seed field. This number is used to initialize pseudo-random generators. Note that pseudo-random generators use their own Seed parameter, so the final result depends both on the Global Seed and Generator Seed parameter.

A new generator can be created by right-clicking on an empty field and then selecting Create -> Generator Type -> Generator from the popup menu. Note that all of the generators are grouped by type:

- Map Generators only work with maps. Their inputs and outputs are always Maps.
- Object Generator will always have an Object input or output. They may or may not have map inputs/outputs, but the Object is always present.
- Output Generators are the ones that directly set terrains and place objects. They cannot have any output connections.
- Portal is not actually a generator but a connection technique. Don’t let the name fool you - it is not about spatial holes to other terrains or dimensions, it’s just a way to organize the graph.

A generator can be removed by right-clicking on the generator and selecting Remove from popup menu.

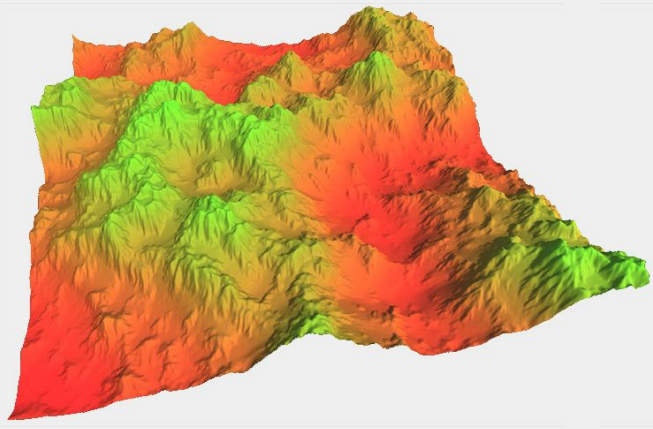
A newly created generator can be connected to a currently existing generator system. Just drag and drop its input or output icons to the other generator’s input or output. Note that the generator’s output can have several connections, while the generator’s input connections are solitary. Some generator inputs are mandatory, that means that they always require some input and generating will fail if they are not connected. Empty mandatory inputs are highlighted with a red mark.



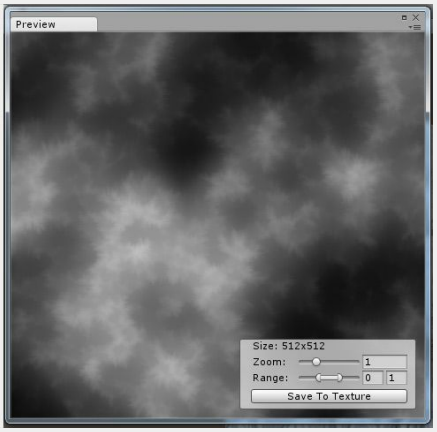
A resulting output map can be viewed using the preview feature. Right-click on the map output and select Preview -> On Terrain. This will color the terrain in a red to green range of colors according to the preview map values. High map values are coloured green, while low values are red.

To exit preview mode right-click anywhere and select Preview -> Clear.

Moreover, a map can be previewed in a separate window. Clicking Preview -> In Window will open up a window with a grayscale map displayed. This map can be scrolled and zoomed the way the main graph can.



Preview on terrain



Preview Window

Settings

General and Terrain settings can be found in the Inspector tab when a MapMagic object is selected.

General settings set up MapMagic's parameters:

- Resolution sets the inputs and outputs map size. Increasing resolution will create a more detailed terrain, but note that generation time will increase quadratically. For example, calculating Erosion Generator - one of the slowest generators - with 5 iterations for the map size of 512 takes about 2 seconds, while the same action for the map size of 1024 takes more than 7 seconds.
- Terrain size sets the length of terrain side in units. This parameter just changes the scale of the terrain object and does not affect performance. Note that changing the size parameter requires terrain adjustments to make the terrain look proportional.
- Terrain height sets the maximum terrain elevation, i.e. map pixels with value 1 will set terrain at this level.
- Generate Infinite Terrain - when enabled this generates new terrains in Playmode when the main camera comes close to edge, creating a virtually infinite world.
    - Generate Range - the minimum distance to the terrain border that triggers terrain generation. Note that the terrain will not appear immediately when the Generate Range is reached, as it will take some time to generate, depending on the graph generator's complexity. This parameter ensures that all of the terrains within range are generated or started generating.
    - Trail Count. MapMagic removes terrains when they are no longer within Generate Range but in case the player would like to return to the terrain just after he left it, there is a pool of terrains that are kept undestroyed named "Trail": when moving along the line these terrains form a trail behind the camera. This parameter sets the undestroyed terrains pool size.

  Note that **pinned terrains are never destroyed in a playmode**. This makes it possible to modify them manually, for example, to include handmade villages among automatic wilderness terrain.
- Multithreaded: when enabled MapMagic works in a several separate threads. Turning this parameter off forces MapMagic to work in the main thread only (this could be useful for debugging of custom generators or for compatibility reasons).

- Instant Generate: when this parameter is enabled there is no need to press the “Generate” button after changing any of the generator parameters: MapMagic triggers the change and forces generation automatically.
- Save Intermediate Results: MapMagic keeps the output results for each terrain 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 MapMagic will clear all generator maps after generation has been completed. This can reduce the memory footprint but each time any of the generators change the entire graph will be re-generated from scratch.
- Weld Margins: Since all of the terrains are generated independently some algorithms like blur or erosion could give different results on a terrain’s borders. To get rid of this difference MapMagic creates a smooth transition from one terrain to another. The Margins parameter is the size of the transition in pixels. When the parameter is 0 then terrain welding is off.
    - Height Weld Margins: margins applied for heightmap welding
    - Splat Weld Margins: margins that are applied for terrain texturing welding. Since texturing differences are less noticeable than height differences, it is recommended to set this parameter lower than Height Weld Margins.

  Keep in mind that using high margins values results in longer apply time (i.e. the time when Unity freezes after generating) and can cause floating objects bugs (when terrain is lowered under the objects to perform the weld). Keep these values as low as possible and use the Safe Borders parameter to negate the terrain difference.
- Hide Far Pinned Terrains: hides pinned terrains if they are out of the generate range. Otherwise they are always displayed - this can give the effect of terrains floating in the air and reduces performance.
- Copy Layers and Tags to Terrains: when turned on MM will copy all of the tags and layers assigned to MM object to terrains on their generate.
- Copy Components to Terrains: when turned on MM will copy all the scripts assigned to MM object (except MapMagic itself) to terrains on their generate.

Terrain Settings and Trees, Details and Grass Settings are equivalent to the settings used for each terrain in the standard terrain’s Settings tab. The only difference is that they are applied to all of the MapMagic terrains. See the Unity docs for more information.

Map Generators

Figuratively speaking, map is a one-channel, square raster image (or 2-dimensional array of floats if you are more of a programmer than an artist). All of the Map Generators are united by one principle: their input and output connections type is Map.

Most generators use a 0 - 1 value interval. In Height Output, for instance, 1 is the maximum height defined in settings; for masks 0 means full transparency, 1 - fully opaque.

Many of the Map generators have equivalent parameters. For convenience they are listed here and will not be repeated for each generator. Some generators can have all of these parameters, others can have only some of them and the rest have none at all.

Inputs:

- Input - the default map to be processed by the generator.
- Mask - multiplies the generator intensity by the mask map's value. For the mask pixels that have a value of 0 the generator effect is invisible, for the pixels that have a value of 1 the intensity is unchanged.

Outputs:

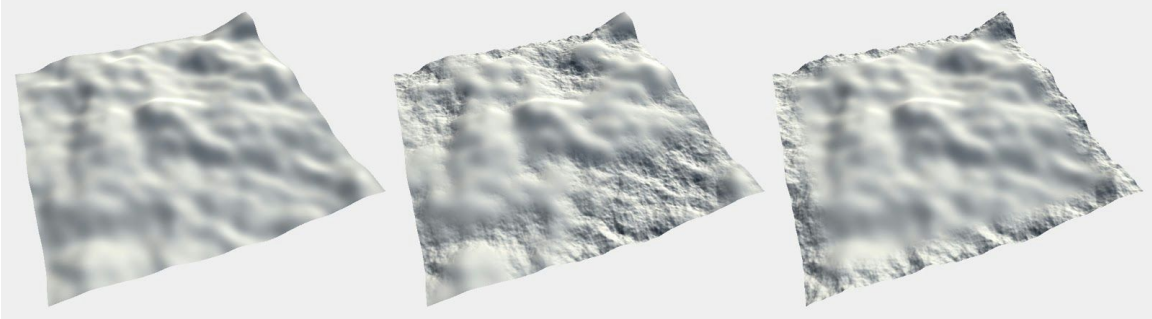
- Output - stores the generator's processing result.

Properties:

- Intensity - the amount of influence of the generator on the input map. When the value is set to zero the generator effect is not visible. In other words, it is the generator's effective opacity.

Noise Amount (when applied over Voronoi): 0 0.1 0.5

- Safe Border - masks terrain borders so that the generator effect on the borders is zero, and increases with distance from the border. This parameter sets the amount (in pixels) until the effect's full strength. When set to 0 border masking is off. This parameter is essential for better terrain welding, so always enable it if this generator outputs to height.



Blur Generator *Using Mask* *Safe Borders: 50*

- Seed is a number used to initialize pseudo-random noise generator. If two generators use equal seed numbers the resulting pattern will be the same.

Constant

Creates a map filled with a given value. This is the most simple generator, used to create transparent masks or flat heightmaps.

Output : the planar at a certain level.

Properties:

- 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).

Noise

The Noise Generator is one of the most basic generators in MapMagic. It generates a fractal Perlin noise map that is widely used as a base for various map creation algorithms.

Inputs:

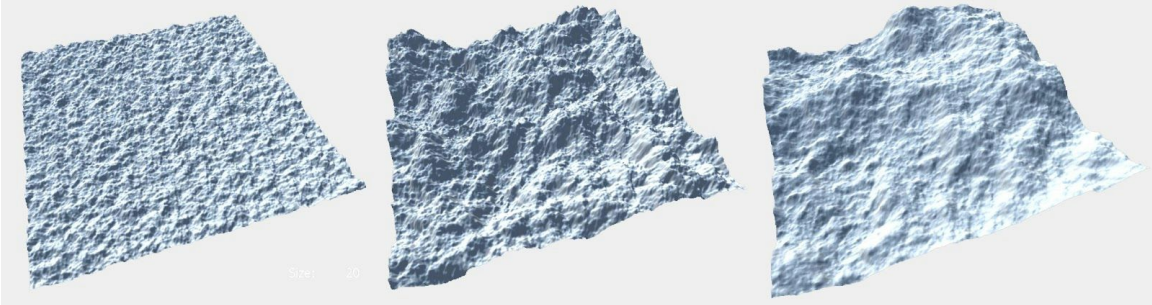
- Input - if no input is specified, noise generator returns the pure noise. If input is specified, noise is added (+) above the input map.
- Mask - a map that controls a generator's level of intensity.

Outputs:

- Output - a noise map (if no input specified) or an input map with a noise added (if input specified).

Properties:

- 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.

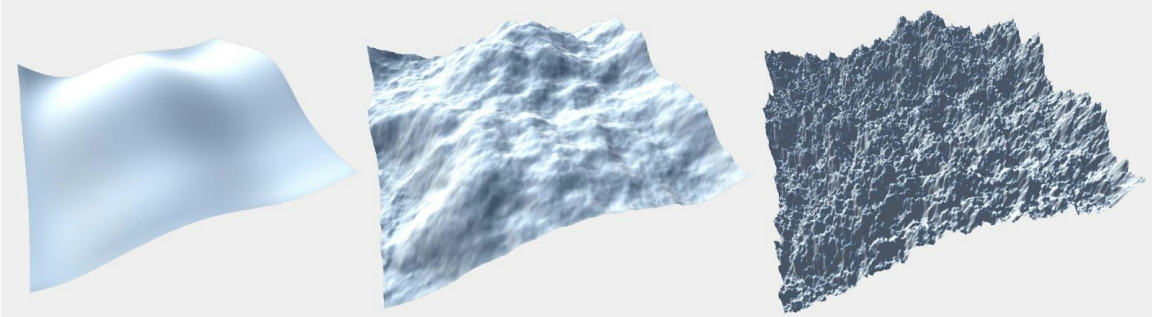


Size: 20
Amount: 0.1

100
0.5

400
1

- 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 more smooth noise.



Detail: 0

0.4

0.6

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

Voronoi

Uses a Voronoi diagram to create maps. Splits the map into cells, generates a random point for each cell, and fills the map using evaluated distances to the closest point and the second closest point. A Voronoi map looks like a mosaic composed of irregular convex polygons.

Inputs:

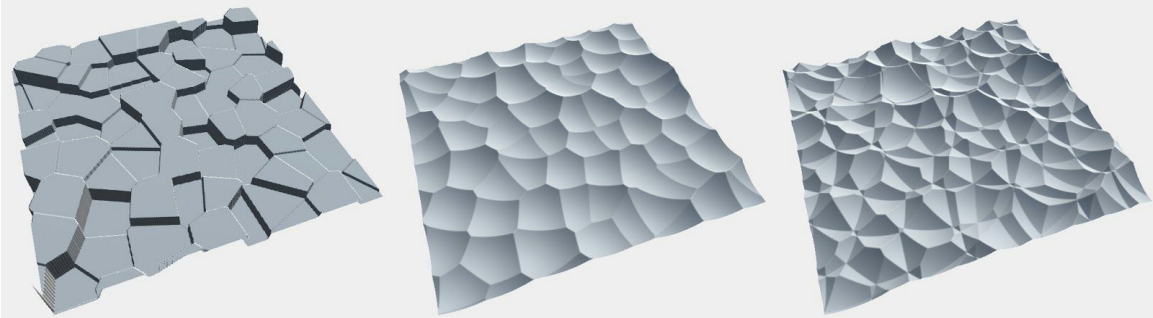
- Input - if no input is specified Voronoi generator returns the pure Voronoi map. If an input is specified, the Voronoi effect is added (+) above the input map.
- Mask - a map that controls a generator's level of intensity.

Outputs:

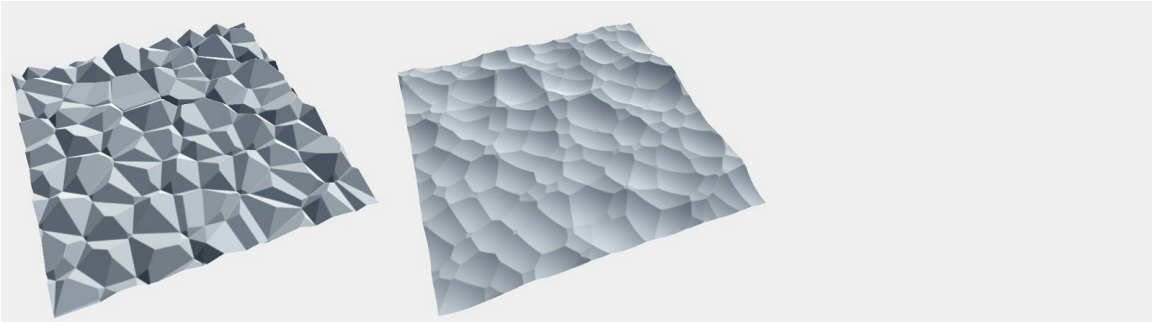
- Output - a voronoi pattern (if no input specified) or an input map with a voronoi pattern added (if input specified).

Properties:

- Type: determines how exactly the final height is calculated:
    - Flat: fills all of the closest pixels with cell point's value
    - Closest: fills pixel with distance to the closest point value
    - Second Closest: same as Closest, except the distance to the second closest point is calculated
    - Cellular: distance to the second closest point minus the distance to the closest point.
    - Organic: distance to the second closest point plus the distance to the closest point.

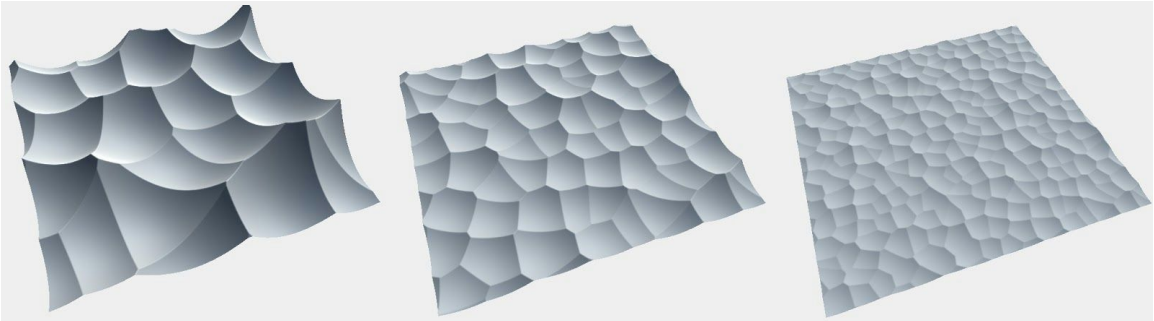


Flat *Closest* *Second Closest*



Cellular *Organic*

- Cell count: the quantity of cells. This parameter sets the size of the Voronoi pattern: The larger the count, the smaller the polygon size. For better terrain welding it is recommended to use a power of two cell count, however it is not mandatory.



Cell Count: 4 *8* *16*

- Uniformity: determines the offset of the cell's point, and as a consequence, the diversity of the polygons. With a value of 1 all polygons are equal, and they get more unique with the decrease in Uniformity value.

- Seed: a number used to initialize pseudo-random Voronoi generator.

Curve

Adjusts map 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.

MapMagic uses Unity's animation curve interface for curve editing, so working with curve 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.

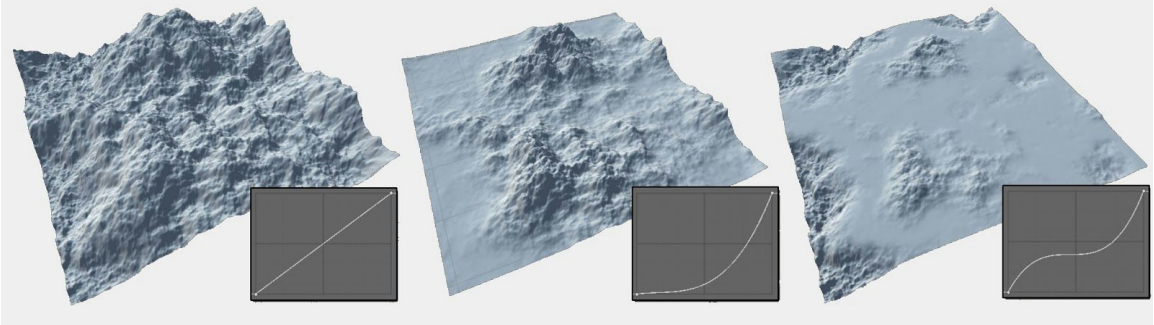
This generator provides plenty of opportunities for map editing: it can be used to invert maps, adjust minimum or maximum values, clamp map values or even make ladder slopes (although a terrace generator could be more handy for this).

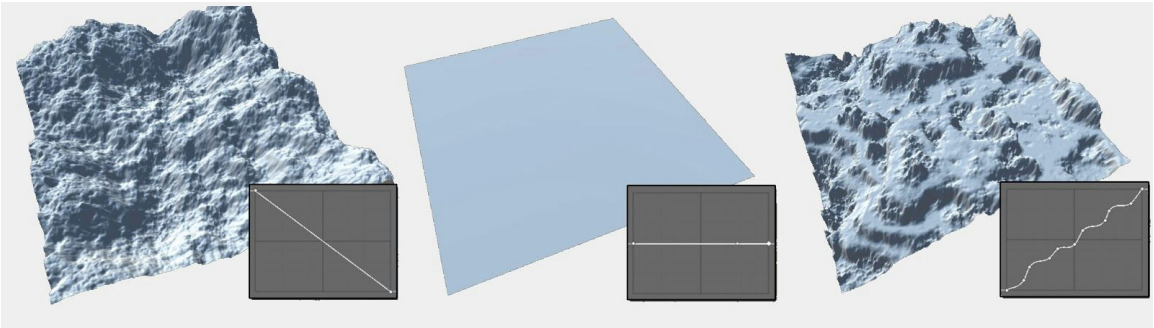
For example, to give the map more contrast just move left and right keys horizontally closer to the central line. To invert the map move the left key to the top and the right key to the bottom so that the diagonal line is inverted (note that key tangents should be set to auto to achieve a straight line).

Input (mandatory) - a source map
Output - a processed map

Properties:

- 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 same parameters could be used to create negative maps or maps with values that exceed 1.





Raw Input

Raw Input generator was designed to import .raw heightmaps as a base for MapMagic terrains. In order to use it the .raw file should be properly prepared. It should be:

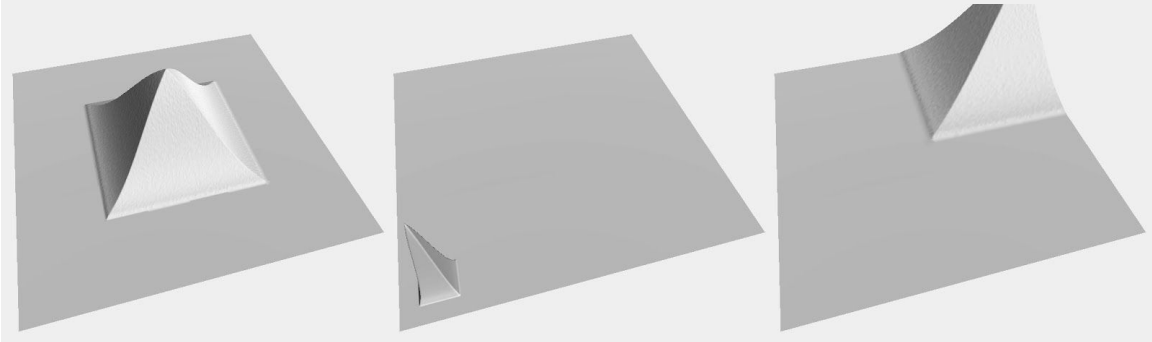
- square (width should be equal to length)
- grayscale
- 16 bits per channel
- and it should be saved as .raw file with PC byte order

It does not matter where the heightmap is saved, whether it is the Assets folder or not, it will be converted to matrix format on import and saved within the scene. A saved .raw file can be added with the help of the ‘Browse’ button. Please note that MapMagic does not keep track of heightmap changes, so the ‘Refresh’ button should be pressed after any change.

Output - the RAW image applied as a MapMagic map.

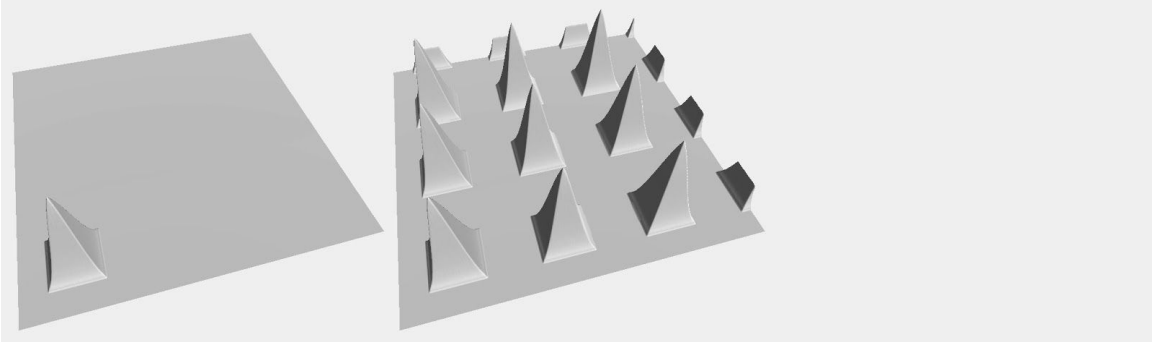
Properties:

- Amount: determines how strong the map’s influence will be on the terrain.
- Scale: sets the size of heightmap. By default size is equal to one terrain.



Scale: 1 *0.2* *2*

- Offset: moves heightmap along X or Z-axis.
- Tile: will repeat the heightmap so that every terrain point will not miss a map.



Tile off *Tile on*

Blend

Blends two maps together using the specified blend algorithm. This generator acts quite similar to the layer blending mode in Photoshop or other graphics editor. Think of the “Base” map input as an underlying layer, and the “Blend” map input is a blending layer.

Some generators like noise or voronoi have their own blend inputs: additive (Input) and multiplicative (Mask). In most cases it is enough to use them, but in some cases you might need the other blend type or need to blend generator chains or generators that do not have blend inputs. Blend generator was created to solve that.

Inputs:

- Base - the underlying map layer
- Blend - the blending layer

- Mask - a map that controls a generator's level of intensity.

Outputs:

- Output - the blended result.

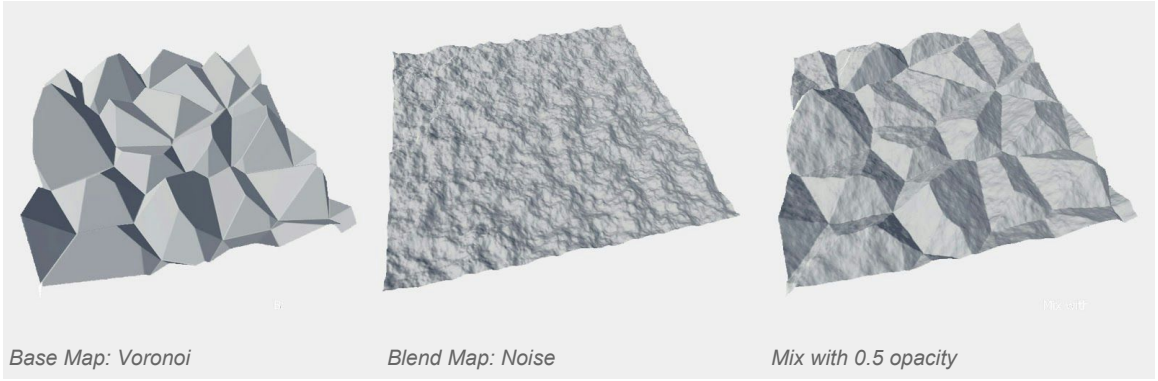
Properties:

- Algorithm - determines how exactly the layers are blended
- Opacity - the blending layer opaqueness factor. If the value is 0 the blending result is not visible.

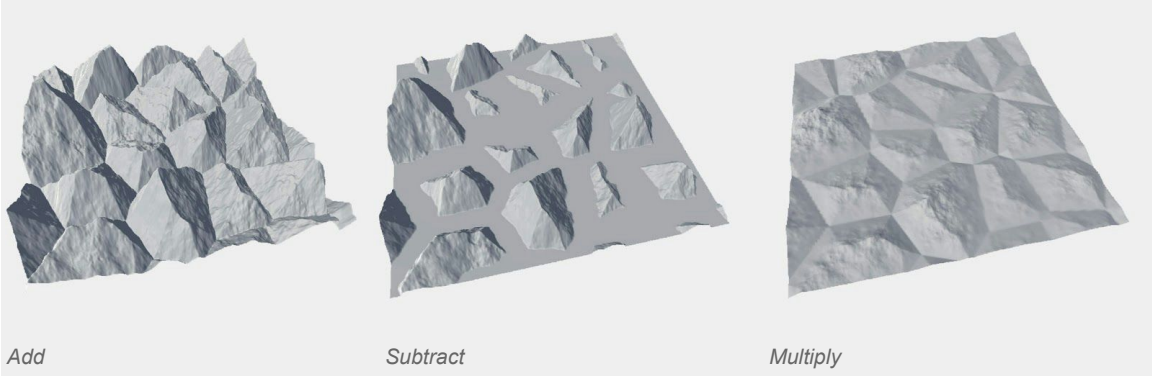
Here are the algorithms the blend generator offers:

- Mix: simply blends two maps using the opacity value Algorithm's formula is { return b };
- Add: adds the Blend layer to the Base layer, i.e. the sum of the layers { a+b; }
- Subtract: subtracts the Blend layer from Base layer { a-b; }
- Multiply: multiplies the two layers. Note that since input values in most cases are less than 1, the multiplied value would be lesser than any of the input ones. { a*b; }
- Divide: the opposite of Multiply. The result can often exceed 1, so use it with care. { a/b; }
- Difference: the delta value between two layers { Mathf.Abs(a-b); }
- Min: the minimum value between both layers. { Mathf.Min(a,b); }
- Max: the maximum value between both layers. Try experimenting with Min and Max blend algorithms as they can bring clean and useful results { Mathf.Max(a,b); }
- Overlay: this effect is often used to process images but the feasibility of using it for blending height or splat maps is questionable.

```
if (a > 0.5f) return 1 - 2*(1-a)*(1-b);
else return 2*a*b;
```



Base Map: Voronoi *Blend Map: Noise* *Mix with 0.5 opacity*



Add *Subtract* *Multiply*

Difference *Min* *Max*

Blur

Smooths the input map.

Inputs:

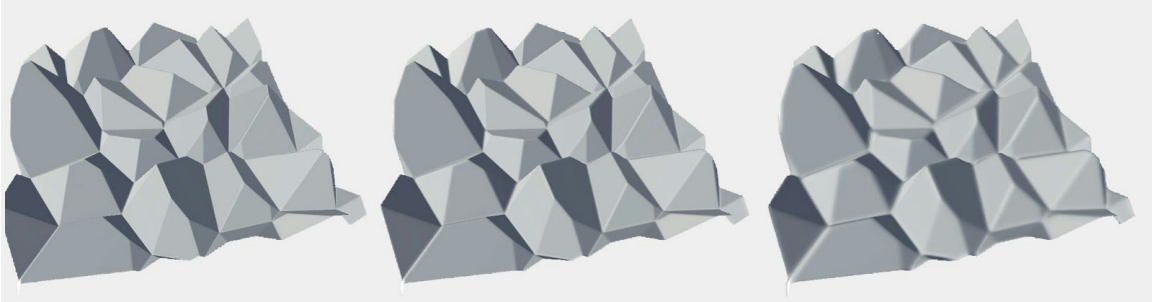
- Input - the default map to be processed by the generator.
- Mask - a map that controls a generator's level of intensity.

Outputs:

- Output - stores the generator's processing result.

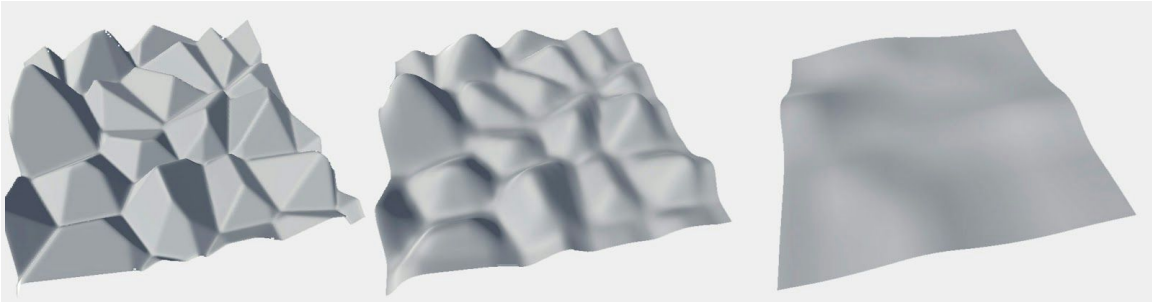
Properties:

- 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: 1 (10 iterations) *20* *60*

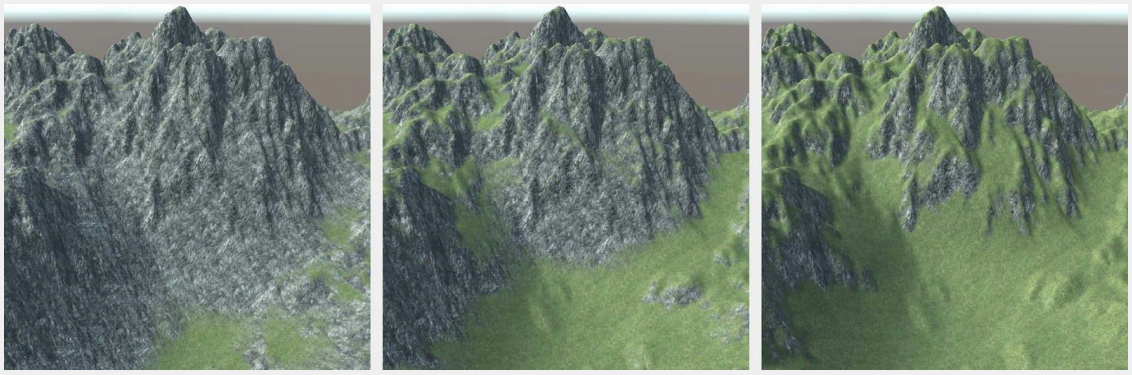
Slope

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).

Input - the default map to be processed by the generator.
Output - stores the generator’s processing result.

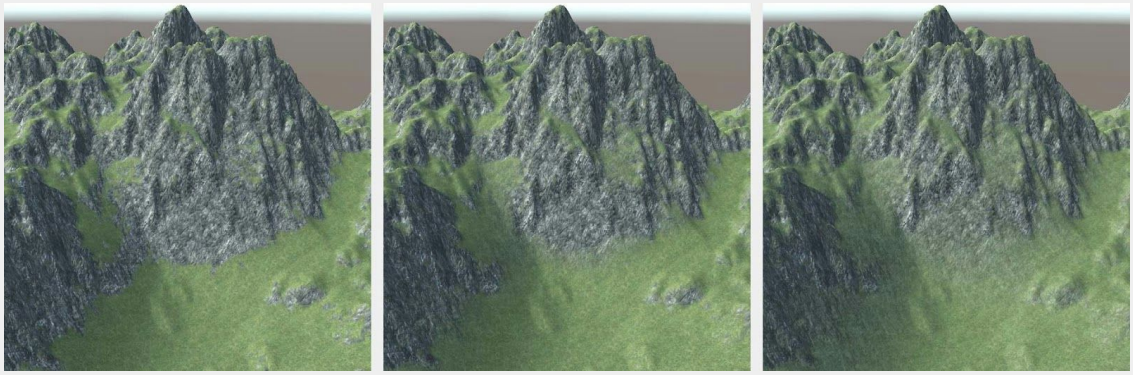
Properties:

- 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*

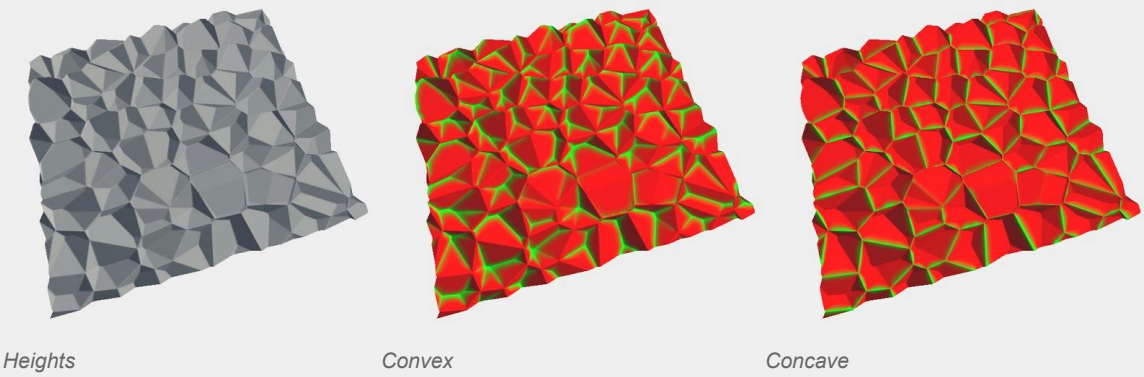
- Range: how gradual will be a slope transition (in degrees)



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.



Heights *Convex* *Concave*

Inputs:

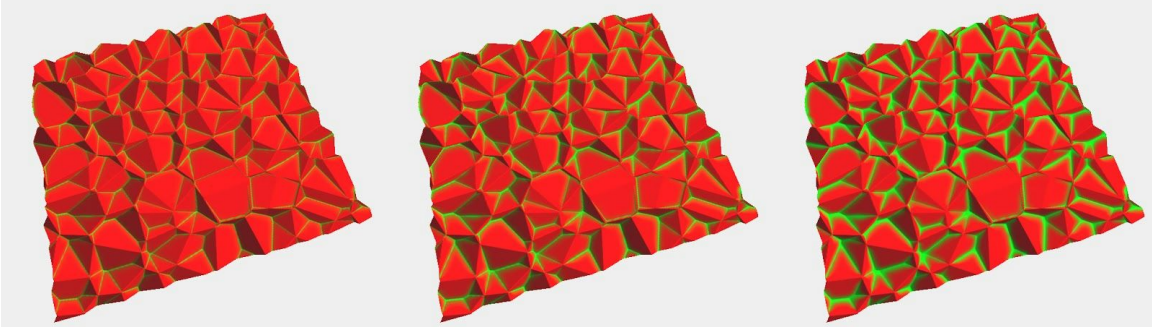
- Input - the heightmap to be processed by the generator.

Outputs:

- Output - stores the bulging or hollow areas (depending on a cavity type) and adjustment pixels.

Properties:

- Intensity
- Spread - since really curving terrain areas are rather rare, the Cavity Generator can expand resulting information on more level areas. Note that convex and concave maps still do not intersect.

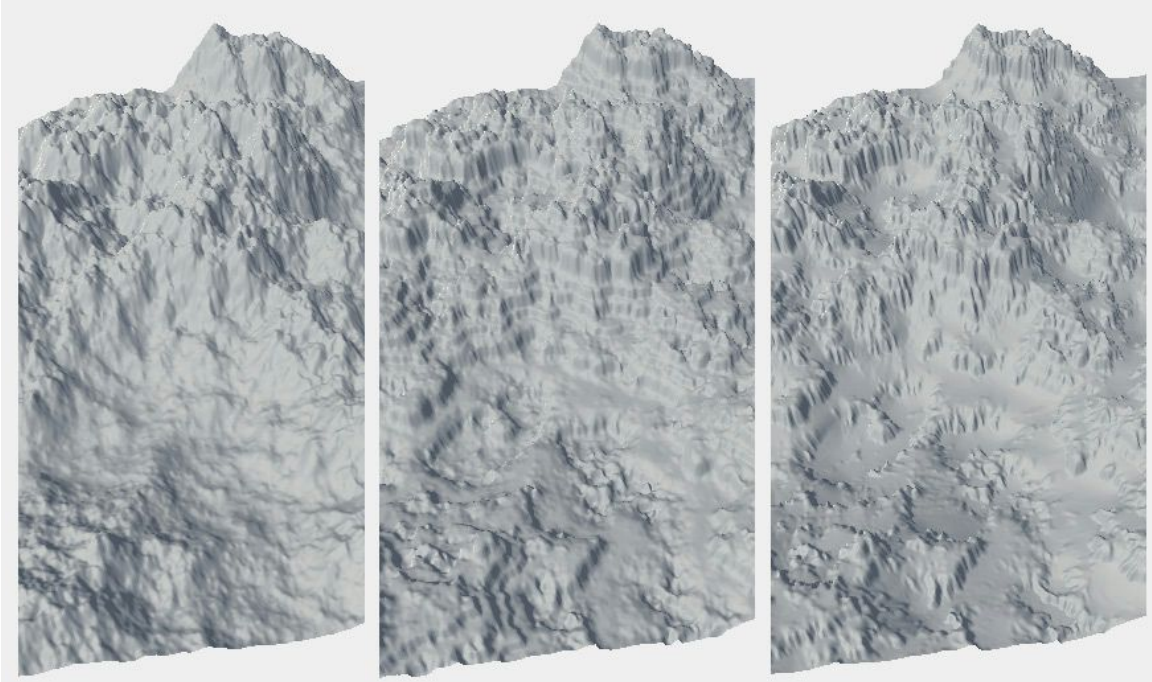


Spread: 1 *2.5* *5*

- 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.

Terrace

Produces step-like land forms on the slopes of hills. Used together with a Noise (as terrace's input) and Erosion (originates from terrace's output) Generator, it can produce fine landscapes. Don't underestimate the Terrace Generator - even if it seems that pure terrace output is not good enough looking, together with erosion it can make a terrain much more realistic and diverse.



Initial terrain *Terrace generator* *Terrace+Erosion*

Inputs:

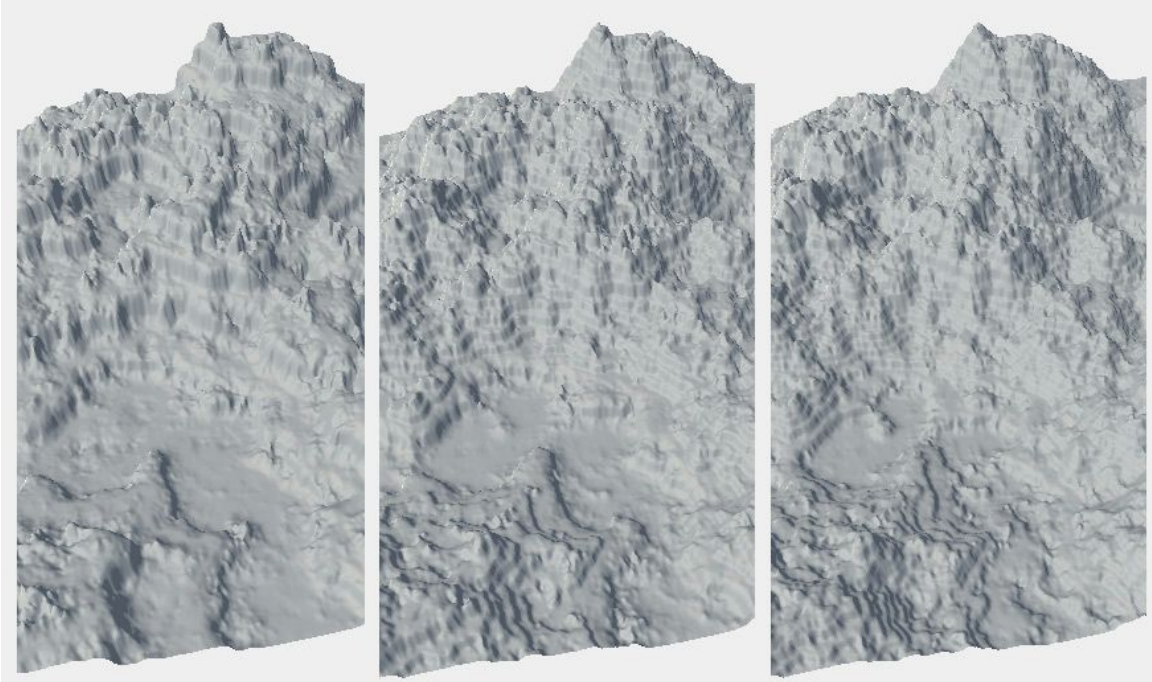
- Input - the default map to be processed by the generator.
- Mask - a map that controls a generator's level of intensity.

Outputs:

- Output - stores the generator's processing result.

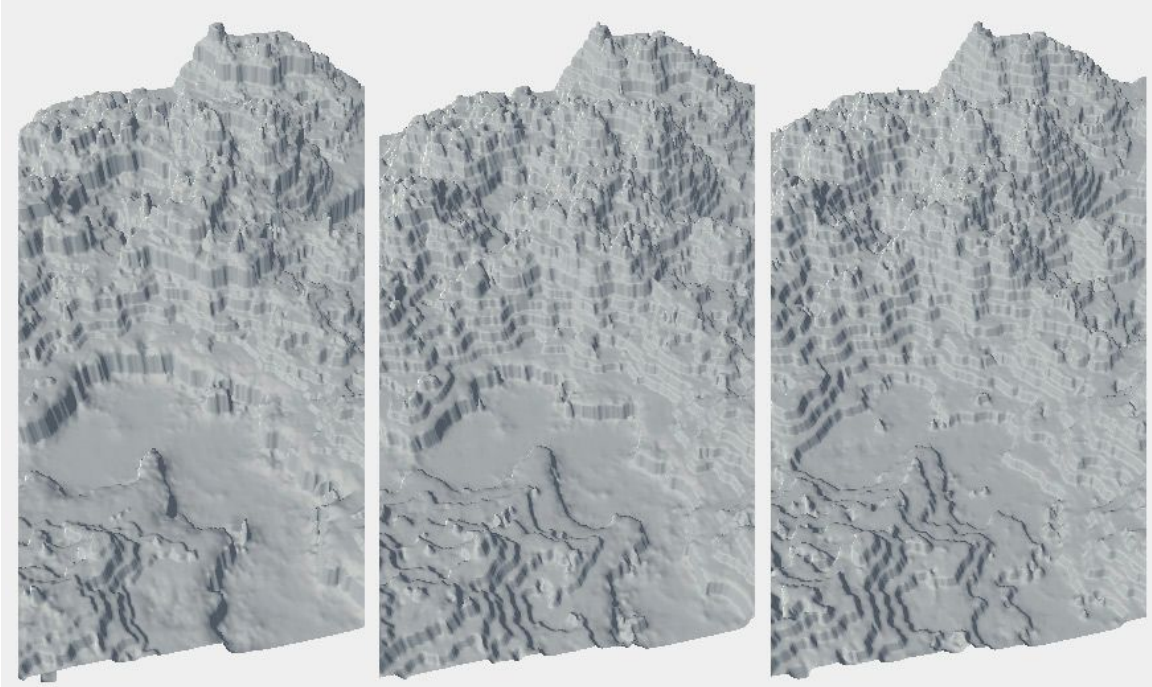
Properties:

- Tread count: each of the terrace's flat surfaces is called a tread. This parameter sets the number of treads - from 2 (at the top and bottom of terrain) to any reasonable number (note that making a terrace height lesser than a unit often does not make sense). This parameter affects the generator performance but since Terrace Generator is relatively fast every value below 100 will not have an appreciable influence on total the generation time.



Treads: 10 *20* *30*

- Uniformity: the terrace’s step’s height difference. When set to 1 all of the steps have the same height and when set to 0 the height may vary from zero to twice the uniform height.



Uniformity: 0 *0.5* *1*

- Steepness: determines how inclined the step's vertical plane(riser) is and how acute the bend between the tread and riser would be. A value of 1 will generate absolutely vertical risers with maximum tread length, while a value of 0 will set the tread's length to zero (but that does not mean that terracing would be turned off - to make the terrace effect even less noticeable use the Intensity parameter).

Steepness: 0 *0.5* *1*

Erosion

Reproduces water's flowing action on the terrain's surface. Flows erode cliff formation, transport eroded stratum to another location and settle it as a sediment. Note that all of the flows, erosion and sediment calculations are iterative and very resource intensive, this makes the Erosion Generator the slowest generator of all the MapMagic generators. It usually takes about 3 seconds to generate erosion with the default parameters on a modern computer - compared to any other generator which are nearly realtime. But the result is usually worth it.

Inputs:

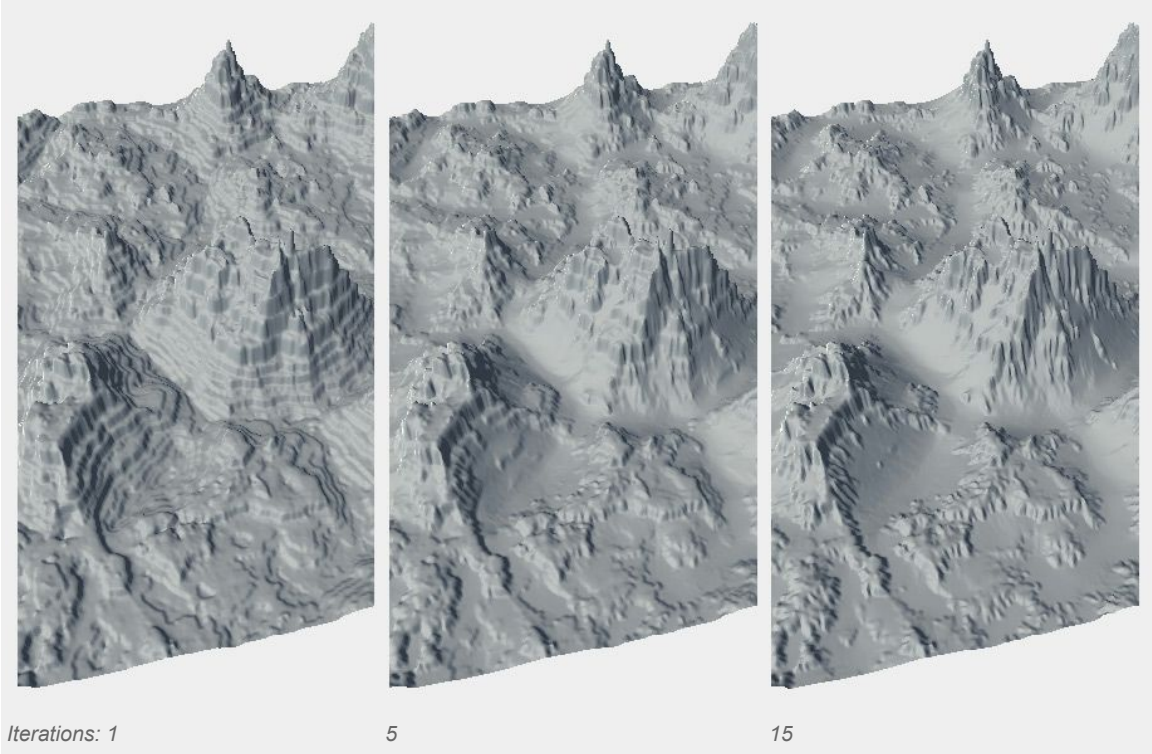
- Heights - the heightmap to be processed by the generator.
- Mask - a map that controls a generator's level of intensity.

Outputs:

- Heights - stores the generator's changed heightmap result
- Cliff - stores eroded depth information. The more soil that was eroded from the current pixel - the higher the map pixel value. This map is multiplied by Cliff Opacity value.
- Sediment - stores the sediment depth information. The more sediment that was brought to the map's pixel - the higher its value. This map is multiplied by Sediment Opacity value.

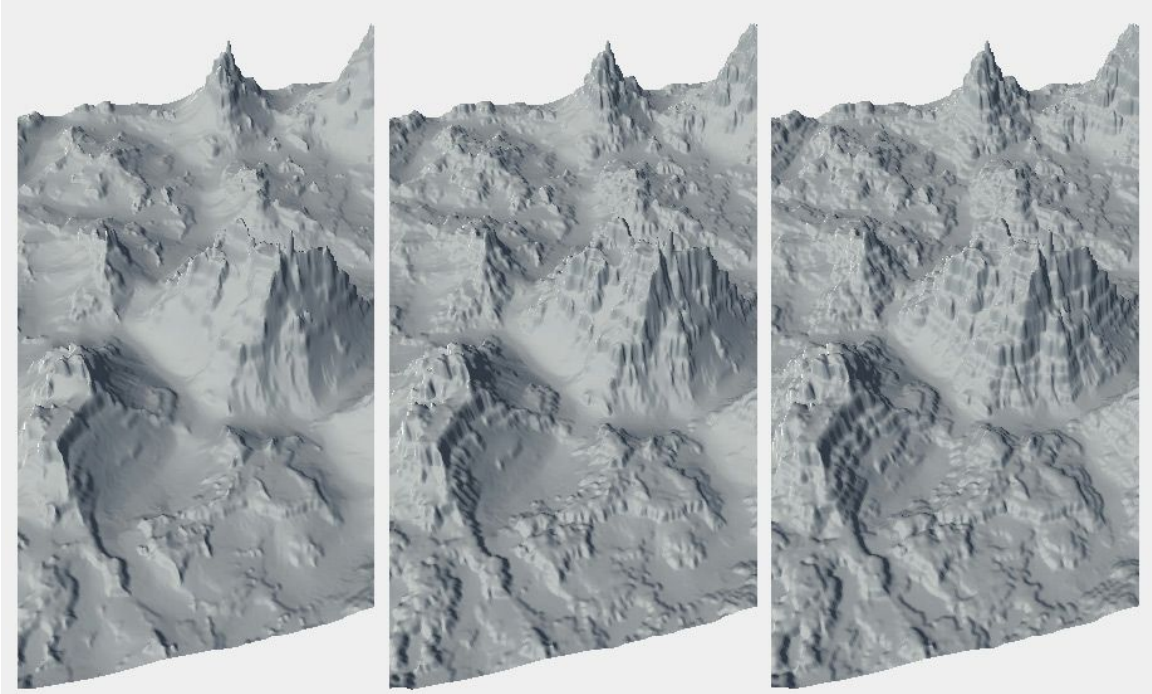
Properties:

- Iterations: number of passes the generator should perform. This parameter determines the performance directly. In most cases there is no need to set it more than 5-7 as further erosion becomes less noticeable.



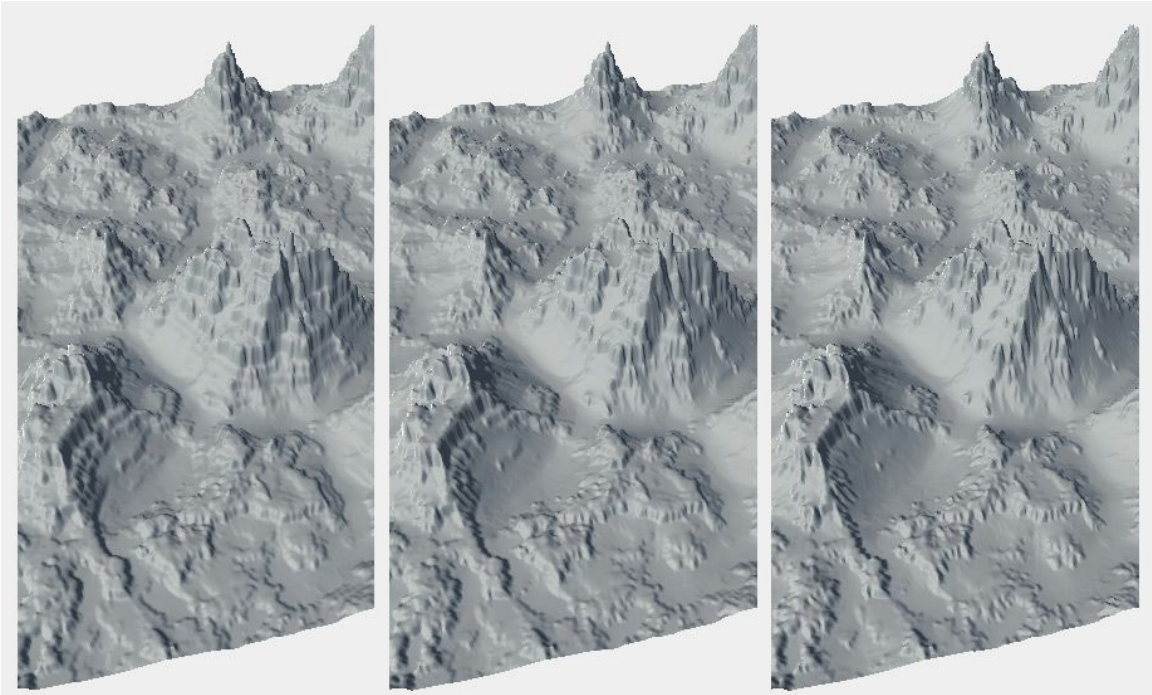
Iterations: 1 *5* *15*

- Durability: determines how durable the input terrain is - i.e. how much it is affected by the flow erosion. Lower values will erode the land more with each iteration but the overall result will be less accurate and predictable, so it is recommended to set this value to 0.8 or higher. If value is set to 1 then the land will not be eroded at all. NOTE: since this parameter acts similarly to Erosion value it will be probably removed in future versions.



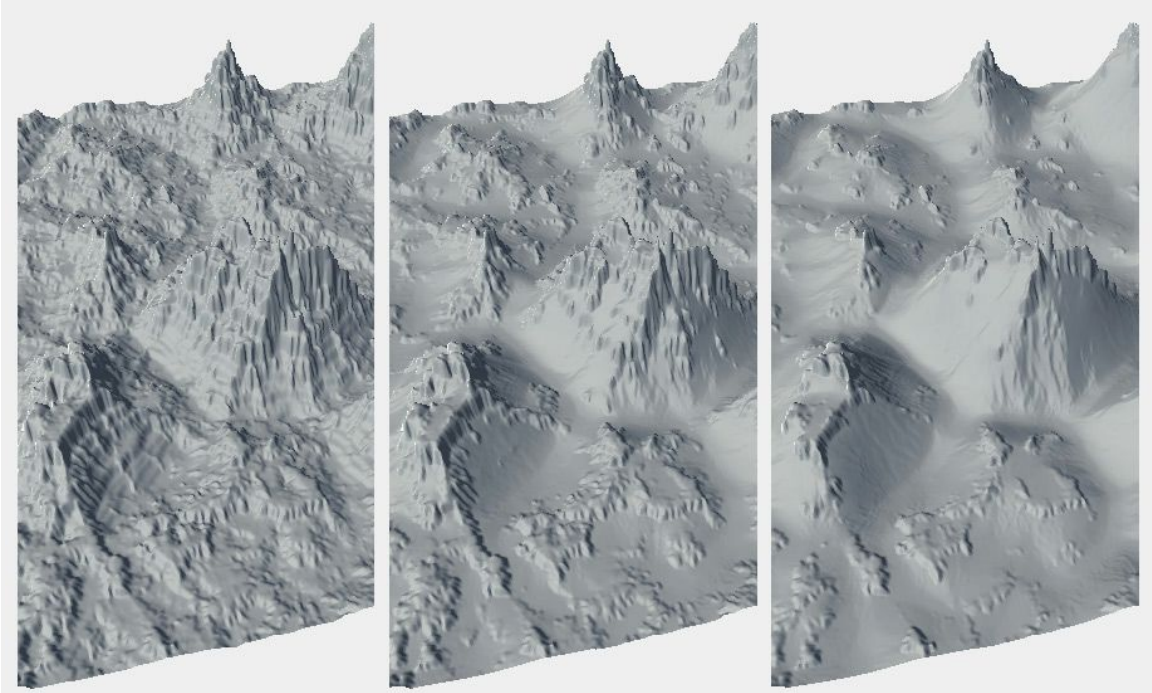
Durability: 0 *0.9* *0.98*

- Erosion: this factor multiplies the amount of stratum that will be raised by the flow to be transported to sediment.



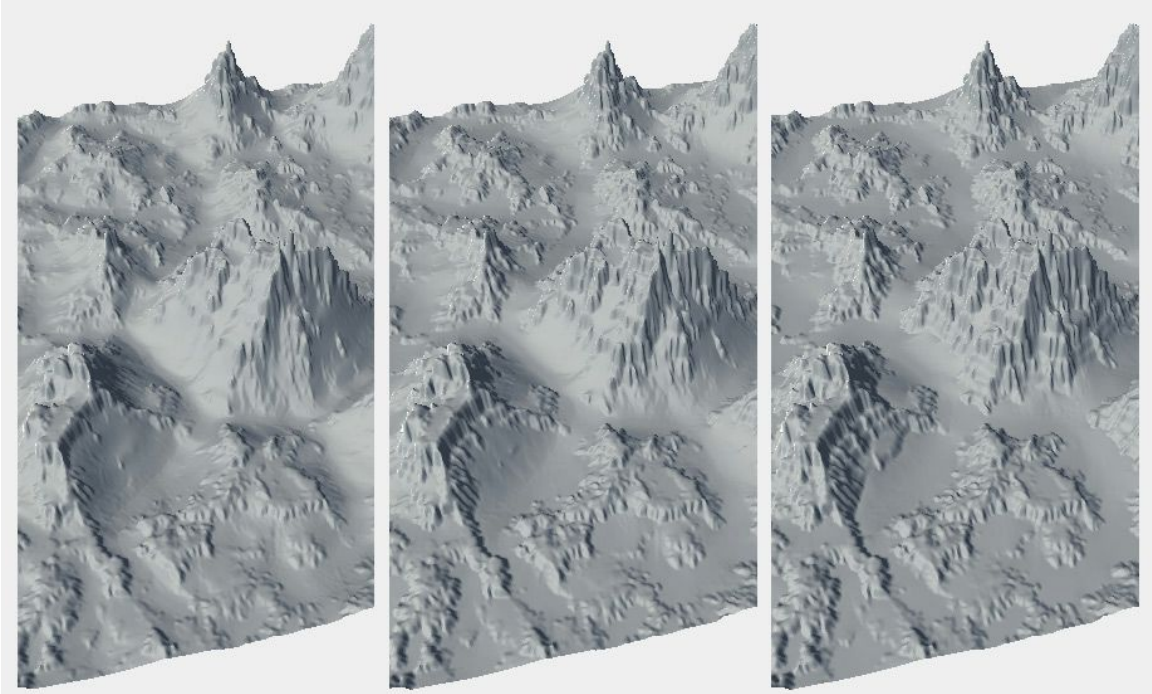
Erosion: 0.5 *1* *2*

- Sediment: this factor multiplies the amount of stratum that will be settled out of the flow. A value 0 means that all of the eroded land will be disintegrated or carried away somewhere so that no sediment will be left at all, just eroded land will remain. A value 2 means that the stratum amount will be magically doubled before it settles. A realistic value should be slightly below 1, but fine results can be achieved within all of the 0-2 value range.



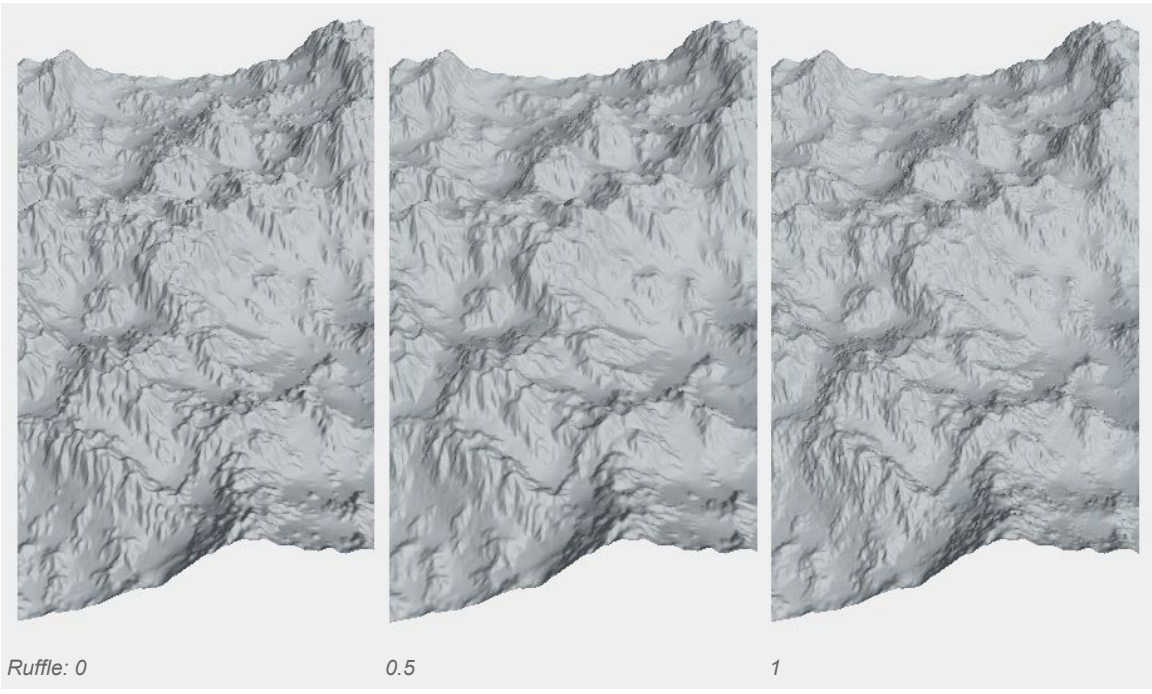
Sediment: 0 *1* *3*

- Fluidity iterations: the amount of passes used to calculate flow runs. A higher value means the farther sediment will travel from where the stratum was originally eroded. Together with the Iterations parameter it affects performance greatly.



Fluidity: 1 *5* *15*

- Ruffle: adds some randomness to the amount of stratum that is being eroded. This results in a more noisy cliff look and a bit lesser erosion depth.



Ruffle: 0 *0.5* *1*

- Cliff and Sediment opacity: factors that multiply Cliff and Sediment outputs to make them more visible.

Shore

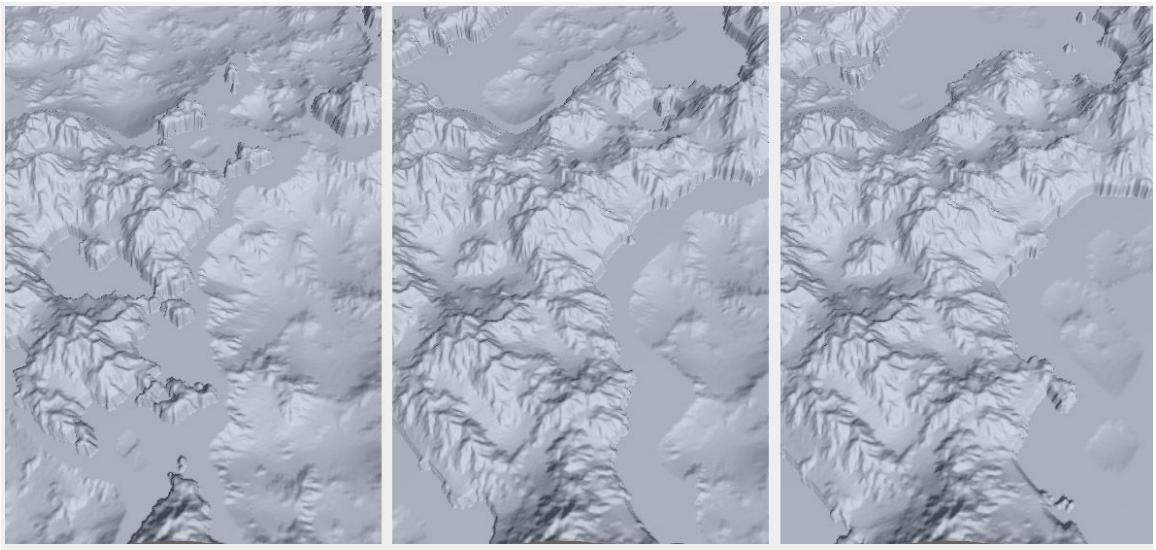
Creates a shore line with a beach and a ridge.

Inputs:

- Input - the heightmap to be processed by the generator.
- Mask - a map that controls a generator's level of intensity.
- Ridge Noise - a map to control diversity of the ridge. Usually just a noise map used here just to make a ridge less homogeneous.

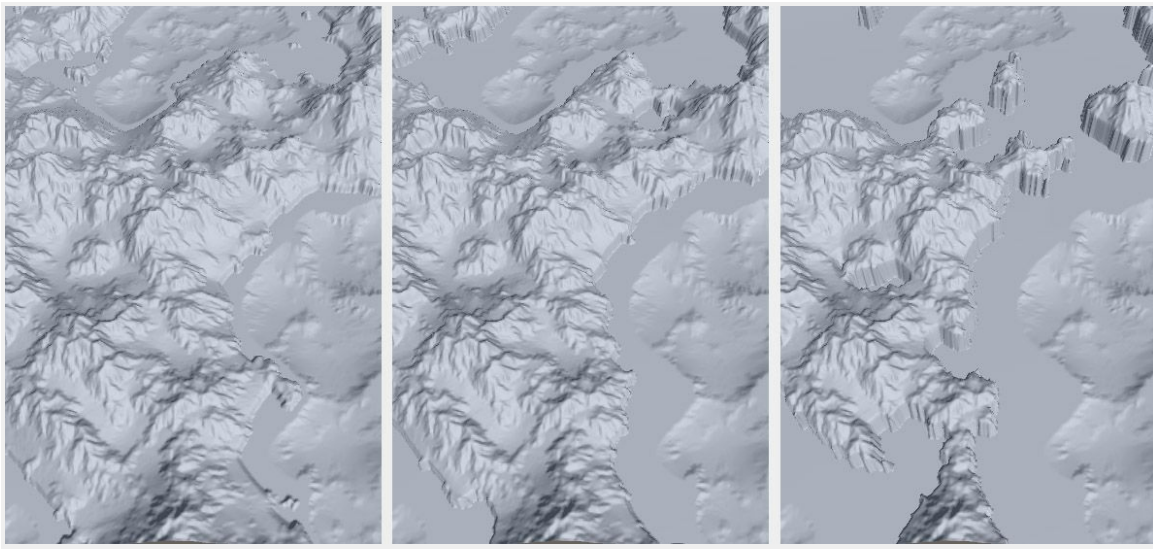
Properties:

- Intensity: the influence of the generator. Hint: this parameter can also modify the beach incline (setting the lesser value makes the beach more inclined).
- Water Level: the level of the beach from the MM object pivot, in units.



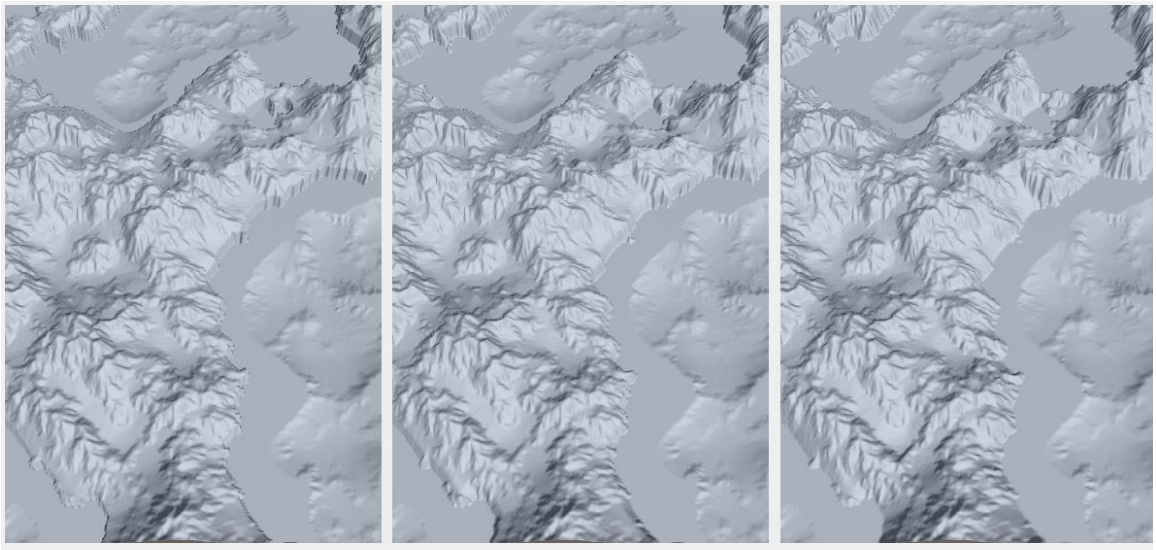
Water Level: 100 *50* *10*

- Beach Size: the size of the beach



Beach Size: 10 *20* *40*

- Ridge Step: the steepness of the ridge. When the Ridge Noise map is assigned it is used as a ridge minimum steepness



Ridge Step: 0 *10* *20*

- Ridge Step Max: the maximum ridge steepness when the Ridge Noise map is assigned.

Object Generators

Object Generators do all the calculations related to objects. They have at least one Object Input or Output(they can also have Map Inputs or Outputs).

Even though “Object” typically refers to “Game Object”, it does not contain a mesh or prefab - it is a special class, equivalent to the “Transform” component. It has the following exposed properties: position, height, size(uniform scale) and rotation(along the Y axis). The real objects are assigned to these object coordinate records in Object Output Generators.

Scatter

Scatters objects in the terrain area. Please note that scattered objects do not lay on the terrain surface but at the zero level - use the Floor Generator to align them according to the terrain height.

Probability input determines the chance of the object to appear on a certain map pixel (for the value of 0 the object will be never appear, for 1 it will always be created according to the generator’s settings).

Note that scattered objects are placed at the zero level. Use Floor Generator to set them to the ground.

Inputs:

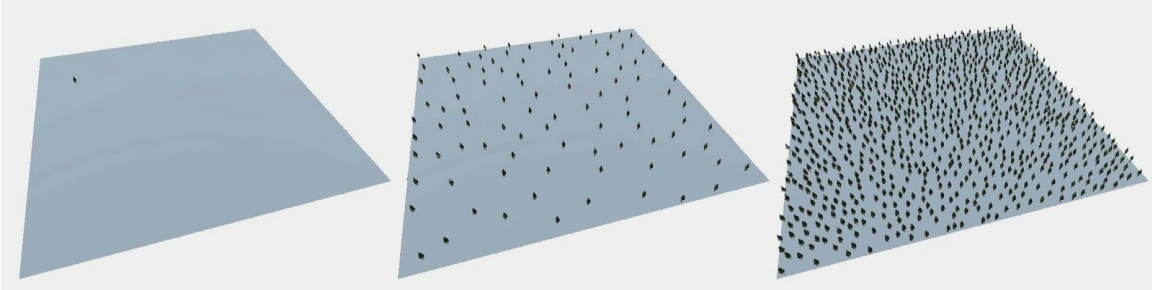
- Probability - the chance to spawn an object at the current map pixel. The higher the pixel values in some of the map area - the more objects will be spawned there.

Outputs:

- Output - scattered objects

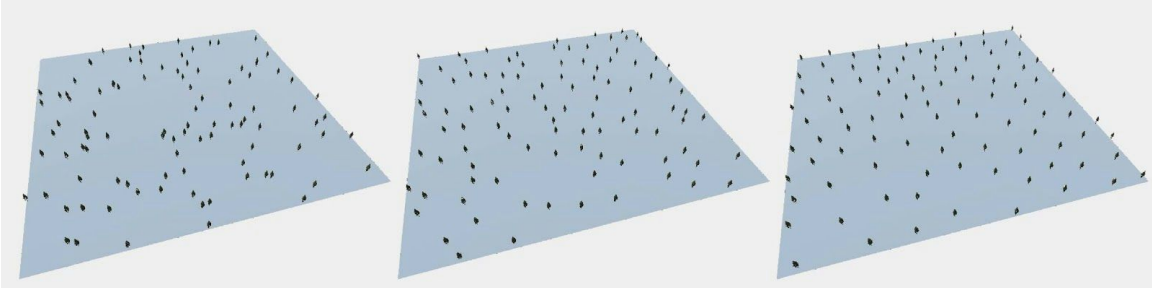
Properties:

- Count: 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.



Count: 1 (terrain size:1000) *Count: 100* *Count: 1000*

- Uniformity: determines how evenly objects are distributed along the terrain. When the value is set to 0 objects are distributed using the pure random algorithm. This can cause creation of object bunches as well as vast empty spaces. Otherwise, the generator skips coordinates that are located close to already scattered once and rolls the dice once more. The higher the value the more even object distribution. Note that the Scatter Generator will never achieve an even “cellular” distribution.



Uniformity: 0 *0.1* *1*

Adjust

Modifies the height, rotation and size parameters of the objects. Note that object rotation and scale parameters are not taken into account by Object Output unless it has “Rotate” and “Scale” checkboxes turned on.

Inputs:

- Input - the default object’s hash to be processed by the generator.
- Intensity - a map that controls a generator’s level of intensity. If some object is placed at the map pixel with value of 0, no adjust will be applied to it. If the pixel value is 1 then it will be adjusted using full intensity.

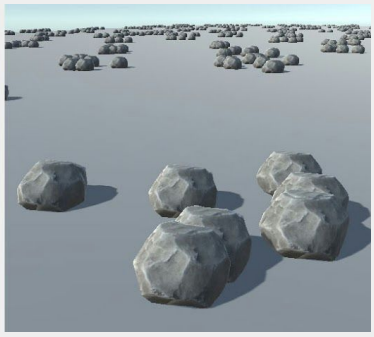
Outputs:

- Output - stores the generator's processing result.

Properties:

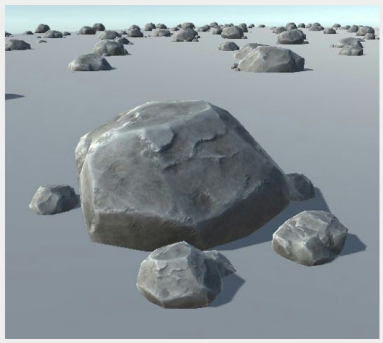
- Type: when the "Relative" type is selected all of the adjustment changes are applied to the existing object's height, rotation and size. The "Absolute" type resets the old values and then places (and scales) the object as if it was placed on zero level, having no rotation and having a scale of 1.



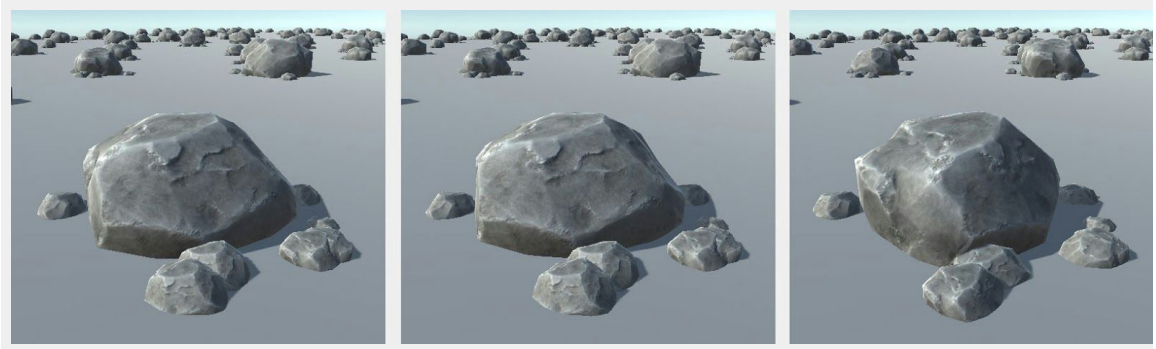


Initial Scene *Relative Size Adjustment (1.5x)* *Absolute Size Adjustment (1.5x)*

- Height, Rotation, Size parameters: all of these have minimum and maximum values, The resulting value will be selected at random and will lie within the minimum and maximum.


No height adjustment *Height: -5, 0* *Height: -5, -5*

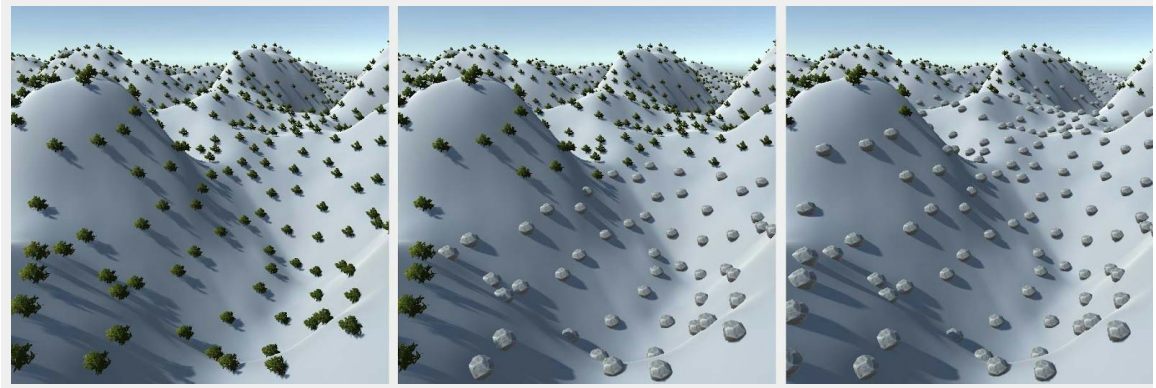


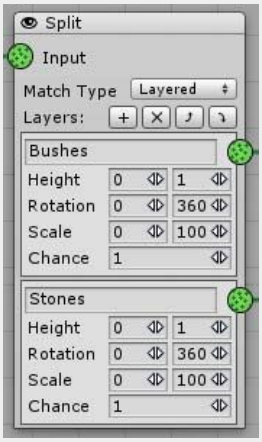
No rotation adjustment *Rotation: 0-10* *Rotation: 0-360*

- Size Factor: multiplies the changes applied by this generator depending on the initial object size. For example, if the object's size is 2 then all of the adjustment values will be doubled if the factor is equal to 1. This is useful for proportional object changes: if the object should be lowered to half of it's height, for instance. When the value is 0 all of the objects are adjusted the same regardless of their size.

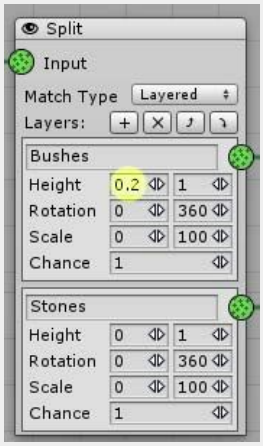
Split

Creates several new outputs and fills them with input's objects using certain conditions. Split Generator has a layered structure. Each layer has its own output as well as minimum and maximum range parameters. For each of the input objects, Split Generator finds a layer that matches the object properties and writes this object to the layer output.





Two layers with the same conditions
Using the top one (bushes)



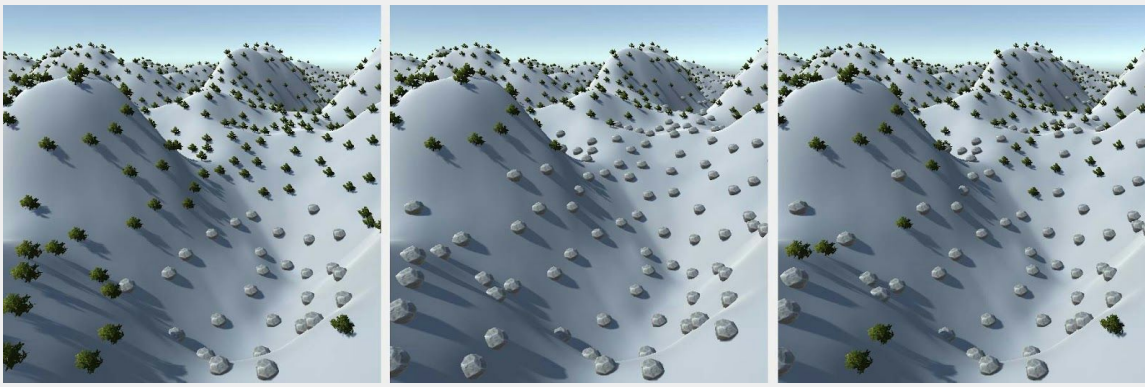
Bushes height starts from 0.2
For every object below Stones layer is used

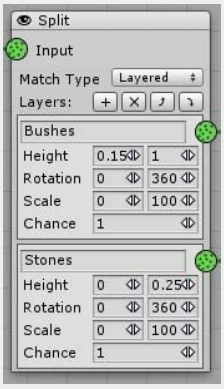
Bushes height starts from 0.3

Input - the default object's hash to be processed by the generator.

Each of the layers has an Output, which stores only those objects from an initial hash that passed the layer conditions filter.

Match Type: Note that the input object will be passed to only one layer output. If the object's properties matches several layers' conditions it will use the top layer if the "Layered" Match Type is selected or it will select a layer at random in the case of "Random" Match Type.





Stones Height: 0 - 0.25
Bushes Height: 0.15 - 1
Bushes atop

Stones atop

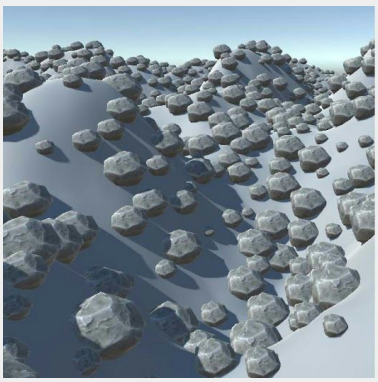
Random match type
Note that in range 0.15 - 0.25 bushes and stones are mixed

Split Generator uses the layered system, similar to the system used in Output Generators. A new layer can be added(with the “Add” button), the selected layer can be removed(with the “Remove” button). Layers order can be organized(with the “Up” and “Down” buttons).

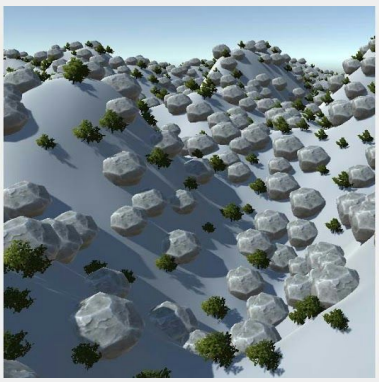
The selected layer can be renamed. Just type the new name in the field next to its output icon.

Each of the layers has the following properties:

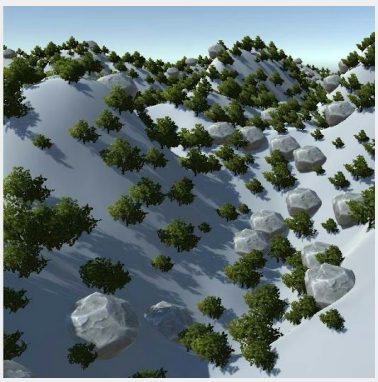
- Height, Rotation, Size conditions: these parameters have minimum and maximum values. If the input object’s parameters lay within the range of **all** layer conditions then this layer will be used(if not overlapped by an upper layer).



Stones Scale Condition: 1- 4

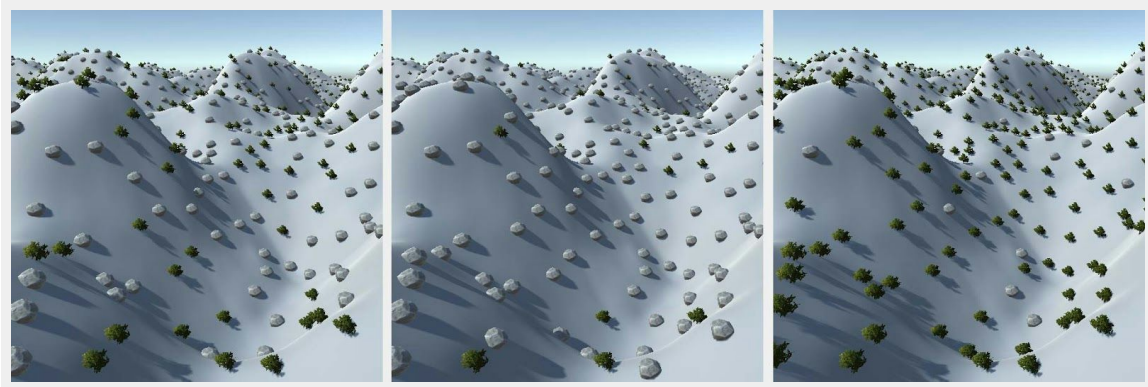


2 - 4



3 - 4

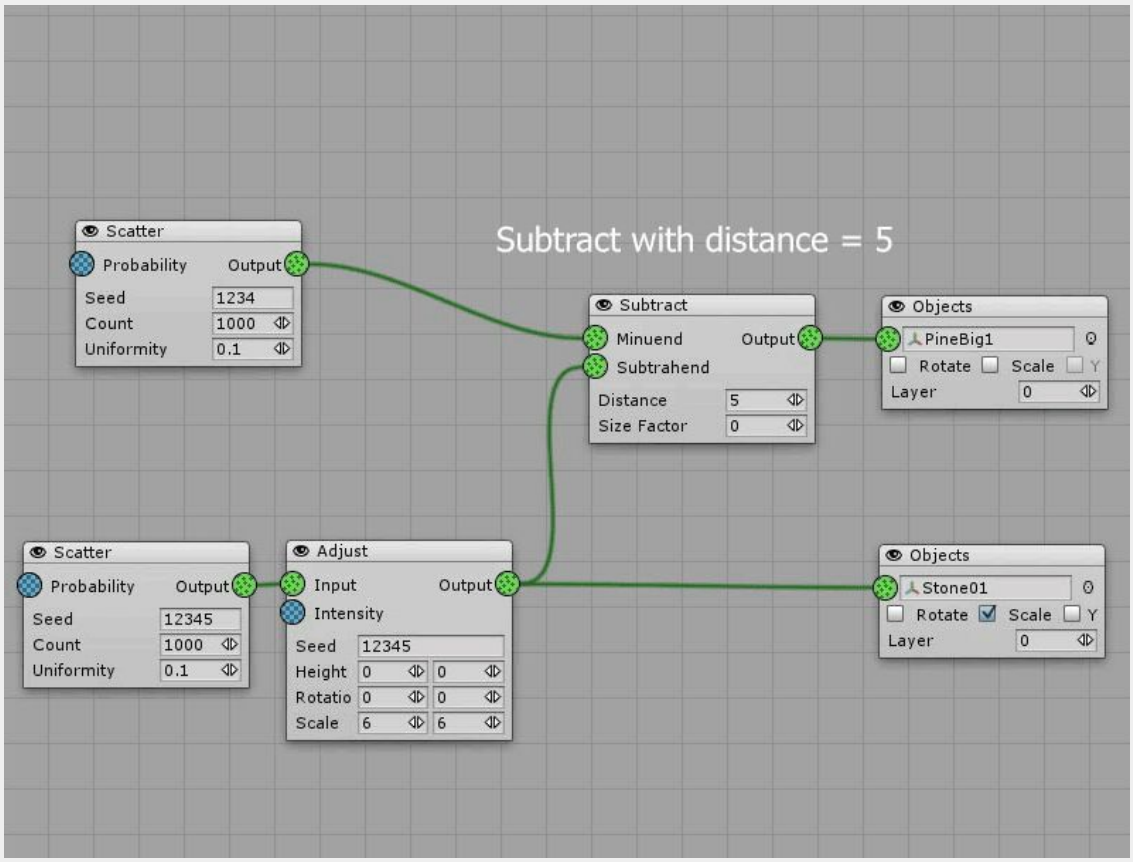
- Chance: used if Match Type is set to Random. If an object matches two or more layers then it is selected at random using the Chance parameter. It determines the probability of using the layer - higher values will give more chances.



Bushes Chance: 1	*0.1*	*10*
Stones Chance: 1	*1*	*1*

Subtract

Returns the modified Minuend Input so that all objects positioned at a certain distance from Subtrahend Input objects are removed.



Subtracting stones from pines removes all pine objects within range of 5

Inputs:

- Minuend - the object hash whose objects will be removed
- Subtrahend - a reference hash that defines which minuend object will be removed. This object hash is not changing and is not included in output.

Outputs:

- Output - a minuend object hash with the defined objects removed

Properties:

- Distance: if the distance between the minuend and any of the subtrahend objects is closer than this value, the object will not be included in the output(i.e. it will be removed). Note that the distance is measured in map resolution units, not in world units.
- Size Factor: makes a distance dependable from closest **subtrahend** object's size. Useful for clearing areas around objects whose size may vary(like stones).

Combine

Object sum: returns the combined objects from the generator's inputs. Output will have all of the objects from all of the inputs connected. The number of inputs can be set with the Inputs Count parameter.

Two or more Inputs - the hashes with the objects that will be combined in output

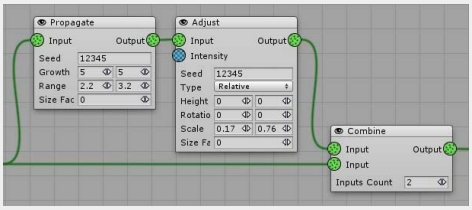
Output - a hash that has all the objects from all the input hashes combined.

Propagate

For each of the input objects Propagate Generator will create a certain number of clones and will offset the created clones from the object's position by a certain distance in a random direction on a plane.

Original object

Propagate with 5 clones



Propagate + Size Adjust + Combine with original

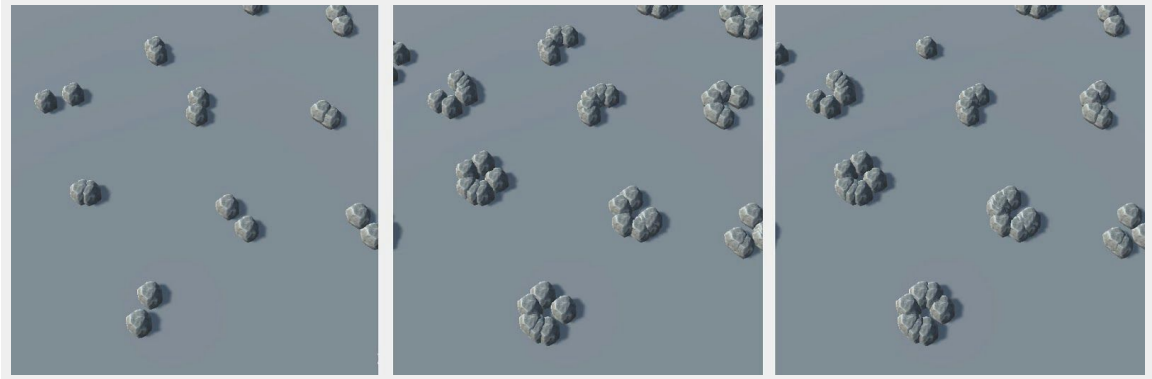
Note that this generator will not return the input objects themselves, it will just output their offset clones.

Two or more Inputs - the hashes with the objects that will be combined in output

Output - a hash that has all the objects from all the input hashes combined.

Properties:

- Growth: minimum and maximum number of clones created for each object by the Generator. This value is a float. For example, when the growth range is set to 5 - 5.5 the Generator will create 5 clones for 75% of the objects and 6 clones for 25%, so the average number of clones per object will be 5.25.



Growth: 2 - 2 *6 - 6* *1 - 8*

- Distance: minimum and maximum distance for clone's offset from the object's original position.

Distance: 1 - 1 *4 - 4* *1 - 4*

- Size Factor: impact factor of the original object's size on the Growth and Range parameters. When set to 0 the object size is not taken into account, when set to 1 Growth and Range parameters are multiplied by object size. Keep in mind that big objects will generate many more clones that can end up overflowing the terrain with cloned objects.

Stamp

Stamp Generator draws circles on the map map in places where objects are located. It can be used both for painting maps under objects and for levelling areas where the are objects placed.

If the Canvas Input is available it will return the modified canvas map, otherwise it will create a new map and apply the stamps to it.

Inputs:

- Input - the default map to be processed by the generator.
- Mask - a map that controls a generator's level of intensity.

Outputs:

- Output - stores the generator's processing result.

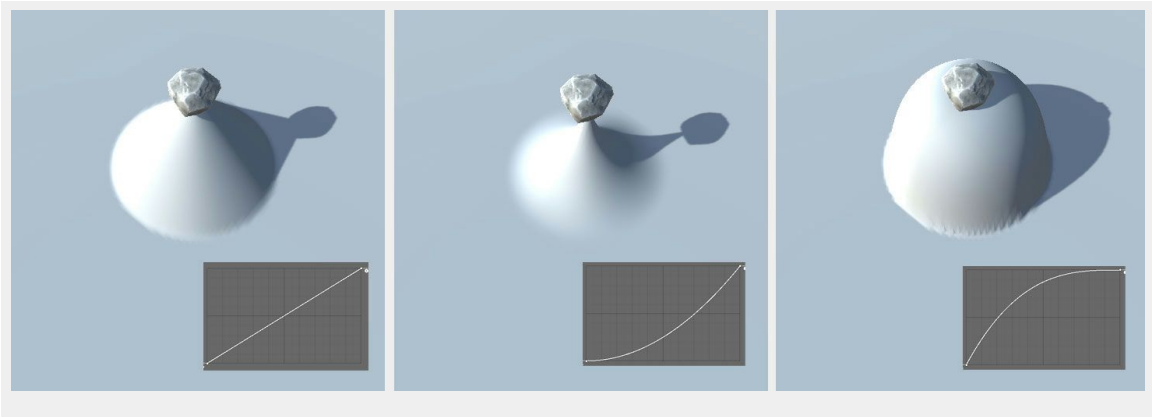
Properties:

- Radius: the size of the painted stroke

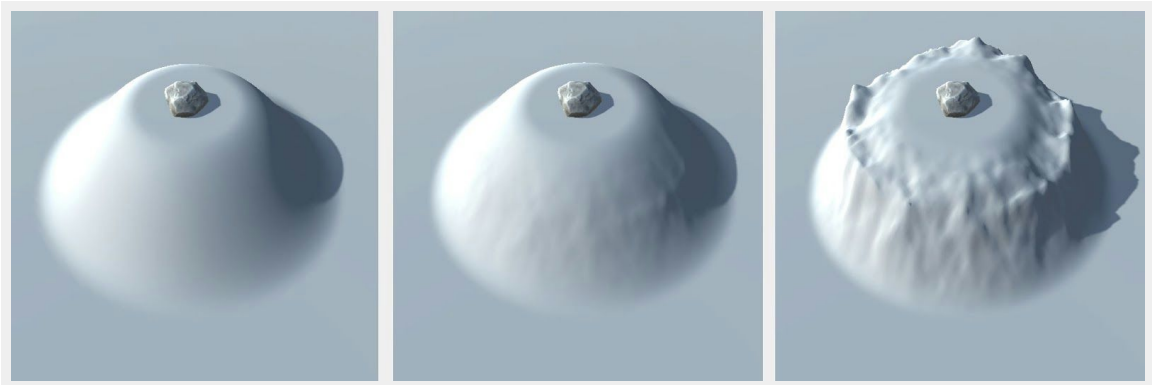
Radius: 1 *2* *4*

- Falloff:

- Curve defines the stroke profile - starting from the edge(curve X = 0, left of the curve field) to the center(X = 1, right of the curve field).

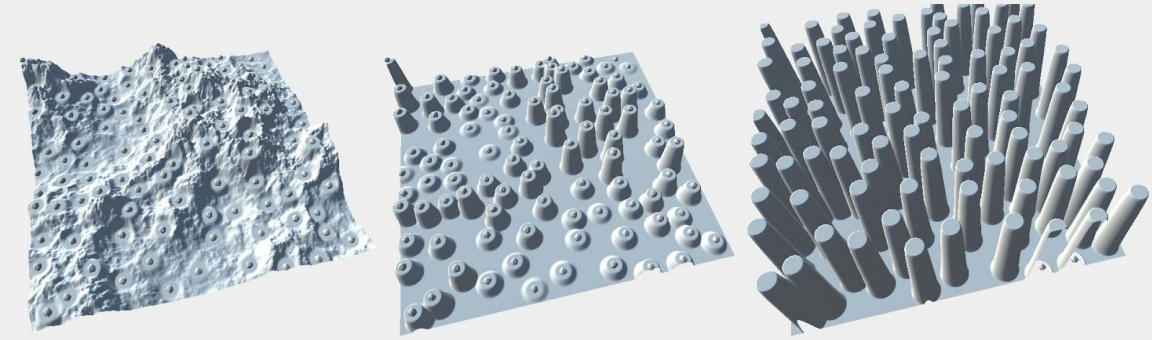


- Noise: applies a noise on the falloff. Note that noise is applied only on the gradient part and not expressed when the stroke amount is equal to 0 or 1. Noise amount and size can be set with “A” and “S” parameters respectively.



Noise: 0 *0.1* *0.5*

- Use Maximum Height. When enabled the Generator will always fill the center(maximum value) of the circle with value of 1(unless it is modified by falloff curve). This is useful for creating new maps using Stamp(creating splats under objects, for instance). When the parameter is off, the Generator will fill the center(maximum value) with the height of the object. This is useful for modifying the heightmap(drawing flat planes under objects, for instance).



Canvas: enabled
Maximum Height: off

Canvas: disabled
Maximum Height: off

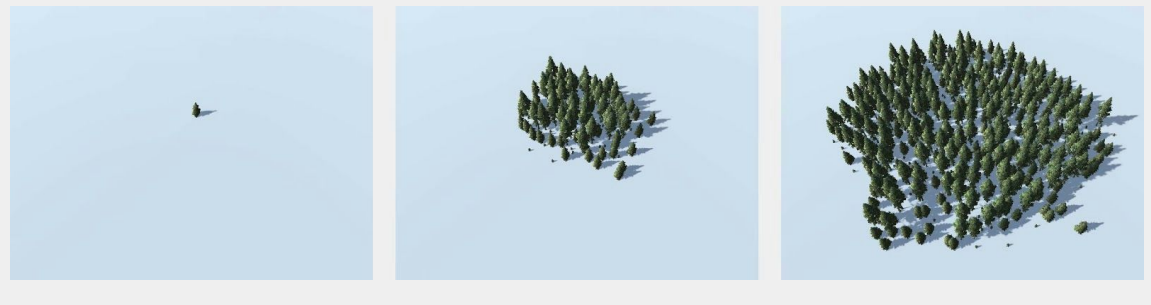
Canvas: disabled
Maximum Height: on

Thus, the Stamp Generator could be used for two distinct purposes:

- creating new maps with recorded object placement(Canvas Input is not required, turn Maximum Height off).
- creating flat planes under objects(Canvas Input is required, turn Maximum Height on).

Forest

Emulates a natural forest growth: seed dispersal, the growth of trees, the adequacy of lighting, and soil quality. Forest Generator can generate several forests of one tree type. As forests grow and expand they will be joint together into a single whole. Each forest starts with a single tree - a seedling.



Along with Erosion Generator this Generator takes some time to compute.

Keep in mind that a forest is a system with negative feedback. Every little change in the beginning causes an absolutely different result in the end. So generating a forest with slightly different inputs can cause a very different final picture. For example, changing the seedling's position can cause the entire forest to die, while other forest arrays would appear in different places.

Inputs:

- Seedlings - the initial trees. Each of the seedlings starts a new forest.
- Extra Shade - additional tree objects that shade the trees. For example, when planting a birch forest it could be more long-lived trees like oaks. This object hash prevents the forest from growing in the other forest's area.
- Soil - a map that controls the chance of the tree to live and to produce seeds. Poor quality (low map values) raises the tree's chances of dying. Think of it like soil quality but this also could be a height or slope factor or their multiplication. This parameter can control not only the soil quality, but other aspects like height or slope factor.

Outputs:

- Trees - living trees object hash. The age of each tree in a Tree's output is recorded as an object size. One year corresponds to 1 unit in size: the first year's sapling's size would be 1, while a hundred year old tree's size would be 100. Use Split Generator to sort trees, with the conditions properly set: for example for young trees the Size Condition should be 0-10, for medium trees 10-30, for big ones 30-200.
- Dead - stumps, deadwood and dead trees. When a mature tree dies(because of age, the lack of lighting, poor soil, etc.) it is sent to the Touchwood output.

Properties:

- Years: number of years passed since the first seedling started to grow. In most cases this parameter determines the size of the forest, but in some cases the forest can shrink after it matures and even die - this often happens for tall trees (i.e. high Shade Dist value) and poor soil.

Years: 50 *100* *150*

- Density: the maximum number of trees in an area of 10*10 units.

Density: 10 *3* *1*

- Max Age: the maximum tree life time. The tree will die when it reaches this age, but it can die earlier because of bad conditions.
- Reproductive Age: the age at which the tree starts to produce seedlings around it.
- Fecundity: how many seedlings the tree produces per year.
- Seed Dist: how far a tree can throw a seed. Increasing this parameter can make a forest spread fast, but for more realistic results it is recommended to set this parameter a little higher than Shade Dist.

Seed Dist: 5 *10* *15*

Slide

Pulls objects downhill. Returns objects that were moved down according to the terrain normals.

Inputs:

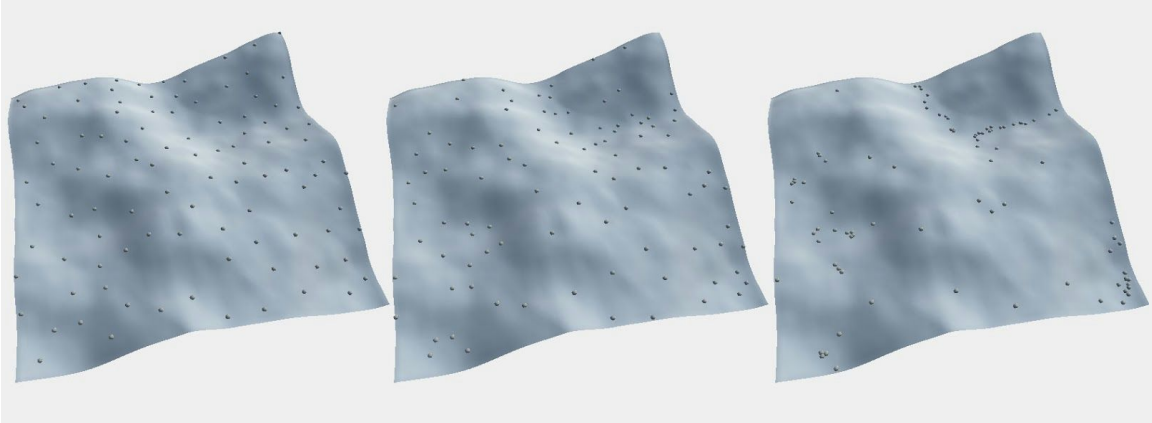
- Input - the object hash to be processed by the generator.
- Heigh - a terrain heightmap. The objects use it to determine their movement direction.

Outputs:

- Output - stores the generator's processing result.

Properties:

- Iterations: object move direction is evaluated every iteration. Increasing iteration count will result in objects travelling further, preserving a generator fidelity but increasing generation time.

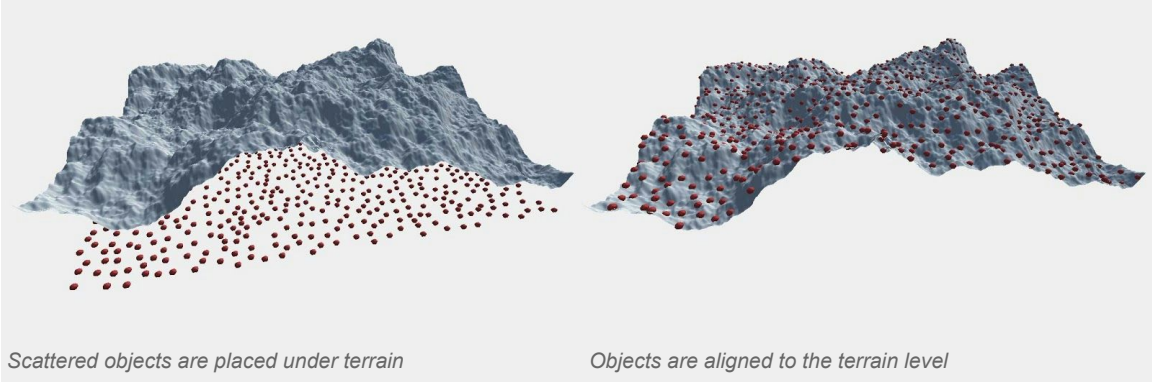


Iterations: 0 *10* *50*

- Move Factor: The distance an object travels per iteration. Increasing this value will move objects further, but very far distances can reduce generation quality: it can make an object move upward or bounce above narrow hollows.

Floor

When objects were just scattered or adjusted(using absolute values) they have no information about terrain height at their position. The Floor generator aligns objects according to the Stratum Input map.



Scattered objects are placed under terrain *Objects are aligned to the terrain level*

Inputs:

- Input - the object hash to be processed by the generator.

- Height - a terrain heightmap. The objects use it to determine height at their position.

Outputs:

- Output - stores the generator’s processing result.

Output Generators

Brings all the map and object inputs to life: converts maps to terrain heightmap, splatmap or detailmap and places real objects on terrain. All of the generator chains should end up in one (or more) Output Generators - otherwise they will not be generated at all.

If the Graph does not have an output of a certain type it will clear corresponding terrain information. For example, a graph with no Texture Output will fill terrain with gray color, removing all the terrain textures.

Layers

A MapMagic Graph should have only one Output Generator of a certain type: only one Height Output, only one Textures Output Generator, only one Objects Output Generator, etc. To output several textures or several objects using the single Output Generator a layer structure is used. Each layer represents the output object type: the Texture Output has the layers number equal to number of Splat (Alphamap) Prototypes and each layer contains a Splat Prototype properties (like texture and bump map); each of the Objects Output layers contains it's own unique object prefab to instantiate its instances on terrain; and so on.

All of the layers[1] have their own input connection, so each of the layers could be considered a separate Output Generator, grouped together with outputs of the same type for technical reasons.

- Clicking the layer name (next to its input) will select a layer and display the layer properties and settings.
- Add button: will add new layer atop of the selected one.
- Remove button: removes currently selected layer
- Up and Down buttons: change the layers' order by moving the selected layer up or down.

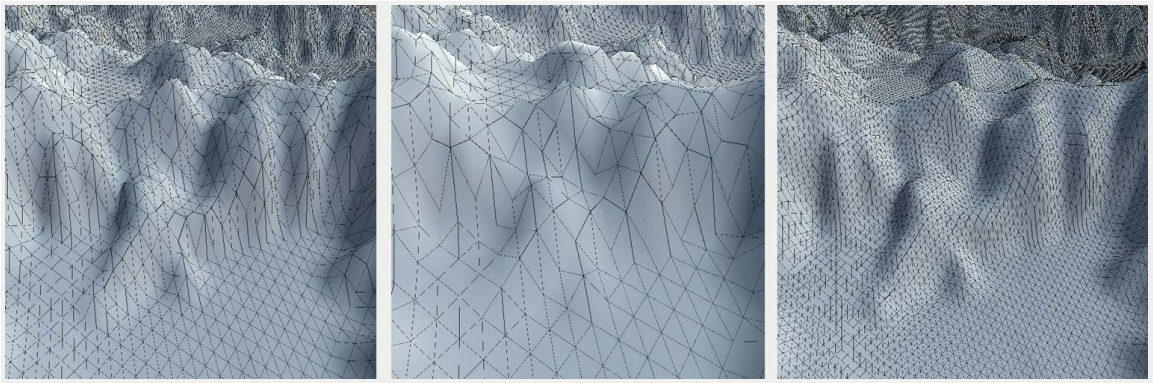
[1] except Texture Output background layer

Height Output

Applies the Input map as a terrain height.

This is the only Output Generator that has no layers, because obviously there are no different objects to apply.

A scale parameter resizes the final heightmap before apply: when set to 0.5 the terrain heightmap resolution will be the half of the map resolution that is set in Settings. When the parameter is equal to 2 it will upscale the map twice with bilinear filtering.



Scale: 1 *0.5* *2*

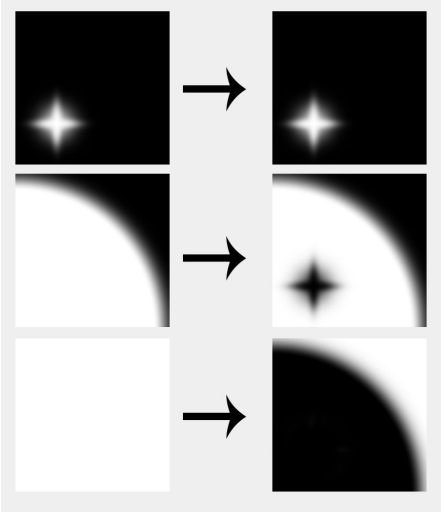
Texture Output

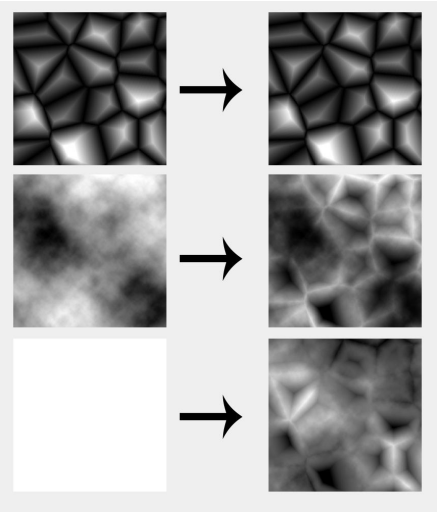
Applies the terrain texturing information, i.e. “colors” the terrain with textures.

Texture Output final result depends on the layer order. Layers are blended together similarly to the layer system in Photoshop and other graphics editors: Each of the upper layers overlaps the lower ones. A mathematical algorithm for each layer generator multiplies all underlying layer values with the inverse value (1-value) of the current layer.

The Background layer does not require an input since it is regarded as completely filled (a constant of 1).

Texture Output layers can have output connections. These connections store a processed and blended layer mask. The sum of the output map values is always equal to 1. These outputs could be used for further map processing (for example, for planting grass using the grass map so it really gets on the terrain).





All the layer properties are similar to the standard terrain texture parameters (splat prototypes). Note that due to better terrain welding splat prototypes the size parameter should be equal to a power of 2.

Objects Output

Applies objects to terrain. For each layer it instantiates the required number of objects and places them in the position and height prescribed by layer's input. Object size and rotation are used too, if the object's layer "Rotate" and "Scale" parameters are checked.

Instantiated objects are grouped as terrain's child transforms.

Objects Output uses the object pool to instantiate objects faster: when the terrain gets out of generate distance its objects are not destroyed, but used to generate new terrain that appears within generate range. So be careful changing objects that are placed on terrain: a changed object will appear here and there as the player walks across the land. Use a separate prefab to make such unique objects.

In editor mode new objects will be created maintaining the prefab connection, in playmode objects are instantiating using prefab clones.

Each layer has these properties:

- Prefab field: a prefab that will be instantiated for each input object
- Rotate: will rotate an object around the Y axis according to its rotation value set in the input object's hash.
- Scale: will scale an object. If the Y parameter is turned on then the object will be scaled along the Y-axis (height) only. If it is off then the object will be scaled uniformly.

- Use Object Pool: will re-use objects instead of their removing and instantiating. This will greatly increase the objects apply time, but could not be used with an objects that change in-game (for mob such as animals, monsters and NPC, for interactive objects such as choppable trees). Using pools with a changeable objects can make the already modified objects appear in the new lands.

Trees Output

Tree Output works similarly to Objects Output: for each layer it instantiates several trees of a given type at the positions prescribed by layer's input. The only difference is that trees in this case are not the Transforms but the terrain Tree Instances.

Note that prefab trees that do not have LODs can not be rotated or scaled. It's not a MapMagic bug, it is the way Unity works.

If some layer's tree prefab is not assigned, Tree Output will report an error in the console but it will not stop or fail generating process. Unassigned trees just will not be displayed.

Each layer has the properties:

- Prefab field: a prefab that will be instantiated for each input object
- Rotate: will rotate an object around the Y-axis according to its rotation value set in the input object's hash.
- Scale Width, Scale Height: the X/Z and Y scale axes can be toggled independently. Switching Width on and Height off will make the trees scale along X and Z axises, while Y scale will always be equal to 1, and vice versa. To scale uniform check both toggles, to disable scaling uncheck both.
- Color Tint: multiplies tree type with a given color.
- Bend Factor: the influence of the tree to the wind zone.

Grass Output

Paints grass on terrain using layer's maps as a mask. This generator can place detail meshes as well.

- Mask - the overall map of grass density. For the mask pixels that have a value of 0 the grass is not planted, for the pixels that have a value of 1 grass is planted using the full density.

Obscure Layers toggle makes the grass behave the same way as Texture Output layers. If turned on for each new layer the grass quantity is blended using the layer opacity. If turned off all grass layers are set up independently.

Each layer has the following properties:

- Mode: determines whether it will be a standard grass, a billboard grass or a detail mesh
    - Grass is the default method to display grass using the static planes.
    - Billboard grass images will rotate so that they always face the camera.
    - Vertex Lit mode is used to place a detail meshes.
- Texture field (for the grass modes): a grass albedo (diffuse) texture
- Object field (for the Vertex Lit mode): a detail mesh prefab

All of the grass parameters are similar to the standard terrain grass and detail properties:

- The *Noise Spread* value controls the approximate size of the alternating patches, with higher values indicating more variation within a given area. (Tech note: the noise is actually generated using *Perlin noise*; the noise spread refers to the scaling applied between the x,y position on the terrain and the noise image.) The alternating patches of grass are considered more “healthy” at the centres than at the edges and the *Healthy/Dry Color* settings show the health of grass clumps by their color.
- Width and Height values specify the upper and lower limits of the size of the clumps of grass that are generated.

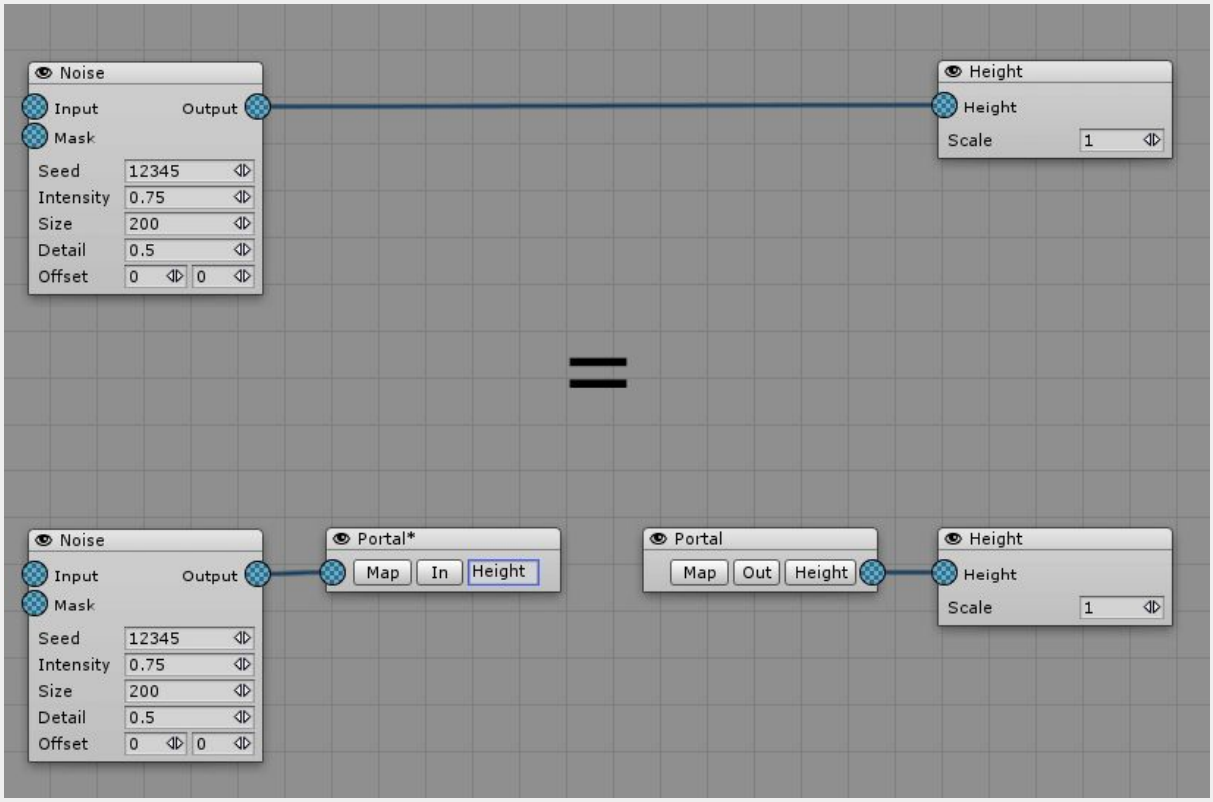
Moreover, there is a patch res parameter that applies to all of the layers. It specifies the size in pixels of each individually rendered grass patch. A larger number reduces draw calls, but might increase triangle count since detail patches are culled on a per batch basis. A recommended value is 16. If you use a very large detail object distance and your grass is very sparse, it makes sense to increase the value.

Portals

Portals are the special generators used to organize the graph in a more convenient way. Sometimes it is necessary to connect some inputs and outputs on opposite sides of the graph. It is not easy to connect generators using the furthest zoom and makes it hard to read the graph.

Portals are the mean to solve the problem. One portal (an input form) has an Input connection and the other portal (an output form) has an Output connection. On generate the input portal just passes the object to the output one, connecting the two generators together.

There is no functional difference between a portal connection and the standard one.

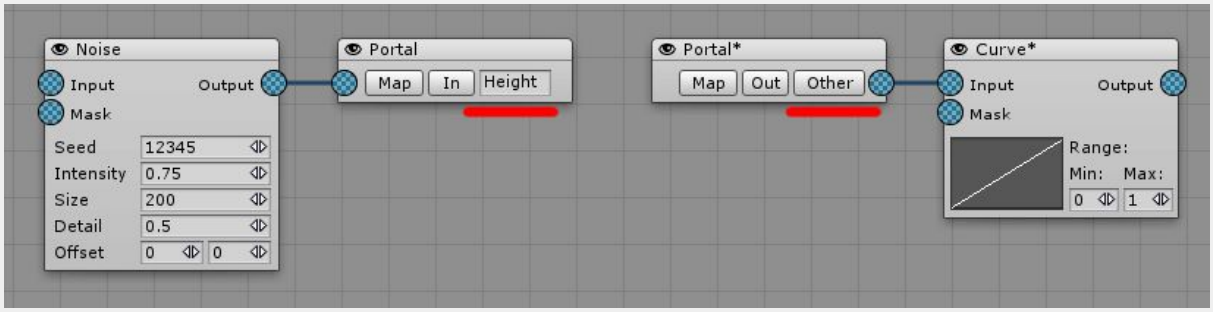


Think of portal connection as a standard connection with no line displayed.

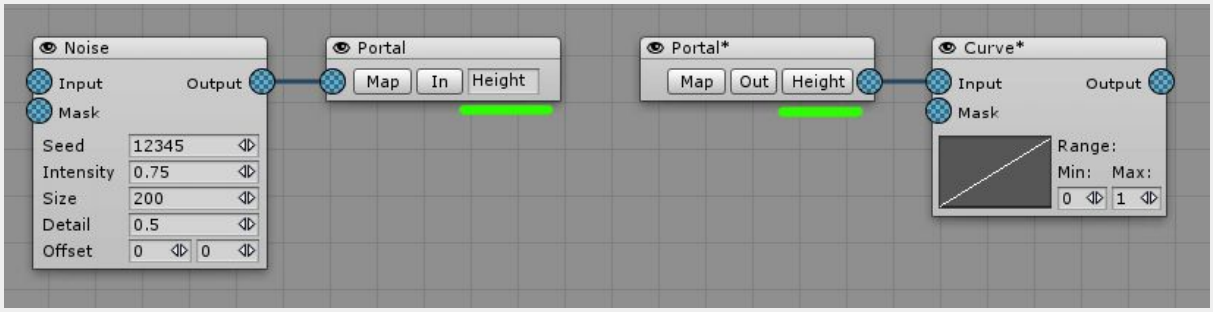
To switch between Input Portal and Output portal use the In/Out button in the middle.

To change the portal In/Out type (Map or Object) use the Map/Obj button to the left.

One more thing that's important to know about portals in order to use them is the portal article. Only those portals that are linked share the same article - in the example above, the article is called "Height". So to link the portals you've got to type portal article in an Input Portal and then select a typed name in an Output Portal. An article could be any name you like and which describes portal map the best in your opinion.

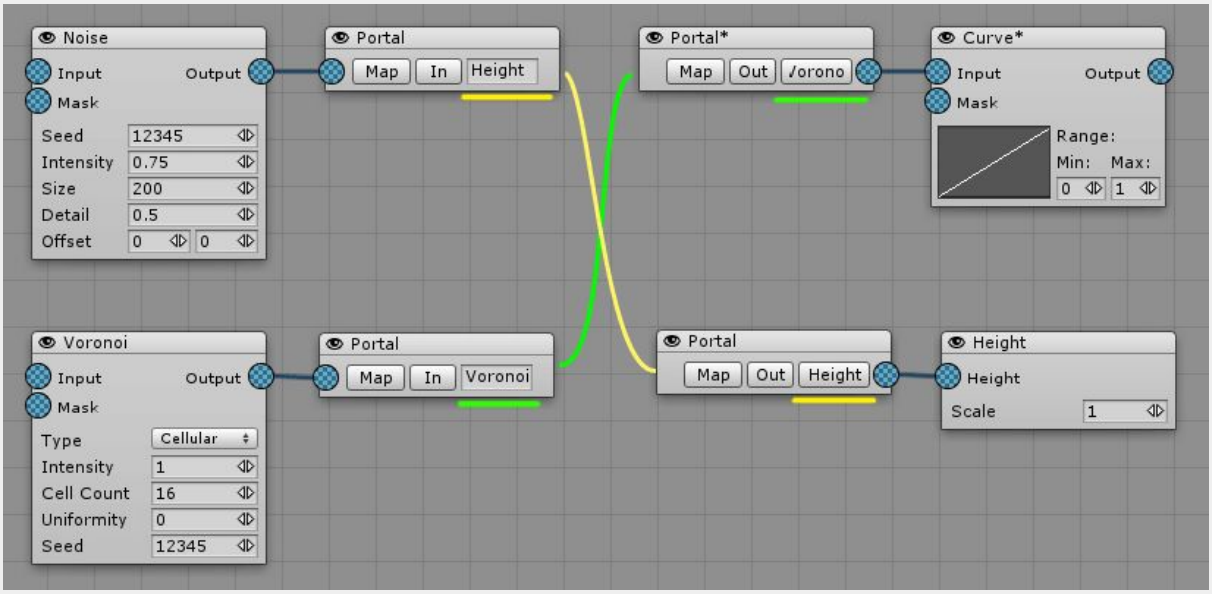


Portals are not linked because of an article mismatch



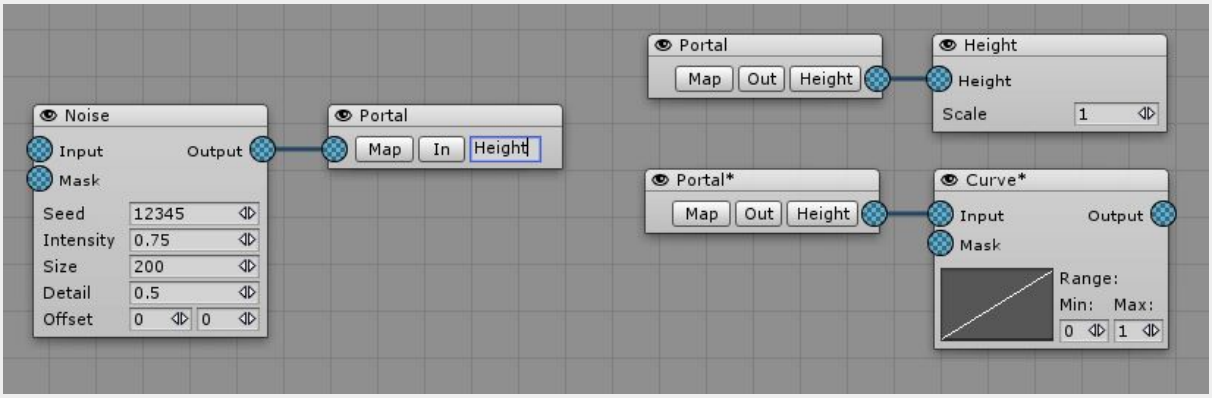
Portals are linked because they share the same article

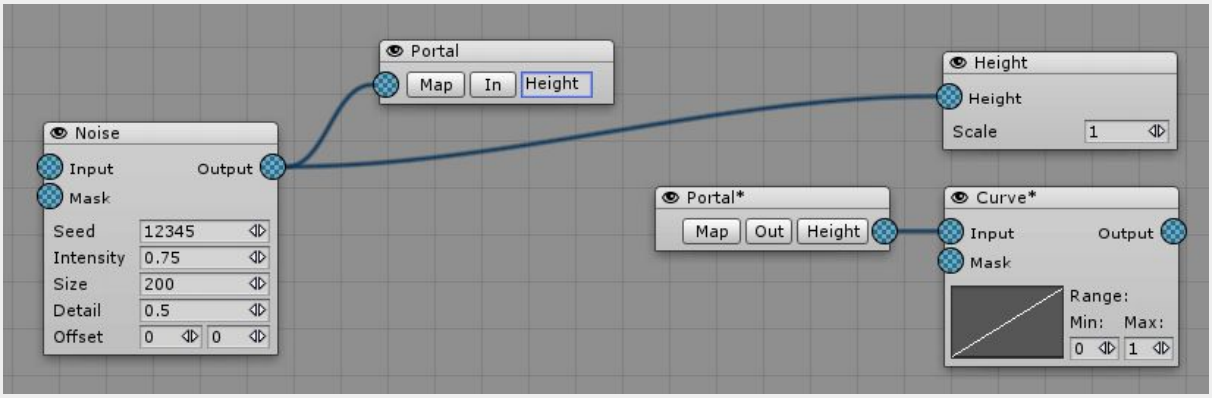
In a complex graph you would like to use portals with a different articles. Keep in mind that only those portals that share the same name are connected.



Portals are cross-connected using the article principle

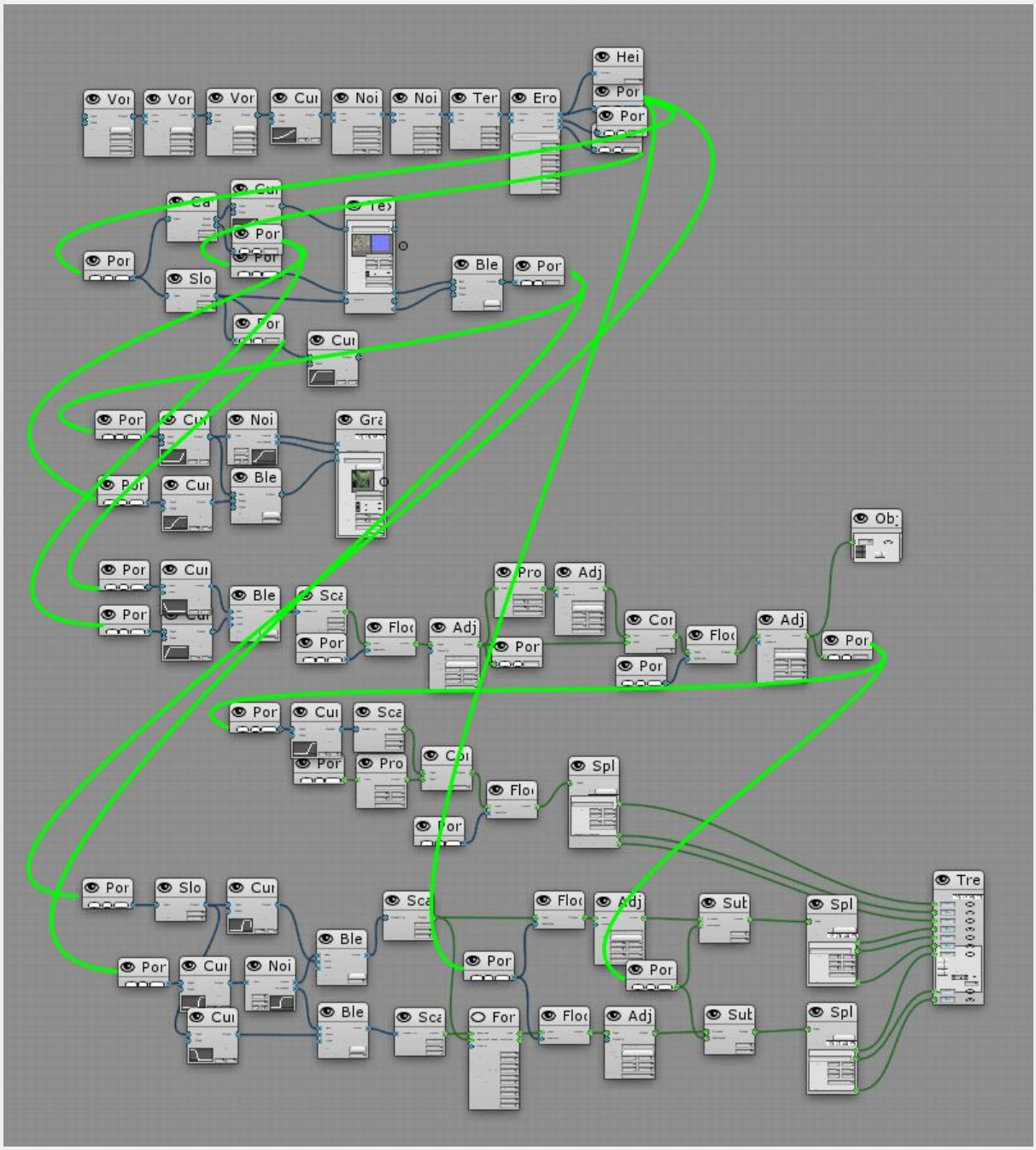
Multiple portal connections are also possible. To create them, use one Input Portal and multiple Output Portals. More complex combinations are also possible.





Possible portals connections

One might ask why do these portals are if they could be replaced with direct connections. It is even more confusing when using a simple graph when learning Map Magic. But I'd like to show a portal connection from a rather complex graph:



Demo scene portal system

So portals should not be used with every connection and not even on every fourth connection but they could be used from time-to-time to make a graph a bit more pretty and readable. Most likely even a complex graph will not have more than a dozen different portal articles.

Creating a custom generator

MapMagic can be extended with custom generators. All generators are made as modules, so a custom generator can be created without changing MapMagic's code and without the need to re-write it on every MapMagic upgrade.

A generator template can be found in the SampleGenerator.cs file. You can copy this file and store it anywhere in your project. But for a better understanding an example of gradual generator creating will be given.

First of all let's start with an empty class. All the generators derive from a base Generator class. Custom generators should override two abstract functions: Generate and OnGUI.

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    public override void Generate (MapMagic.MapMagic.Chunk chunk) { }
    public override void OnGUI (Layout layout) { }
}
```

This class will have two attributes: one that makes the class serializable and another one that adds the current generator to the MapMagic right-click context menu, for example this one could be created with Create > MyGenerators > MyGenerator. The last attribute also has a "disengageable" parameter that allows the generator to be turned off by clicking the eye icon in the upper left corner.

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    public override void Generate (MapMagic.MapMagic.Chunk chunk) { }
    public override void OnGUI (Layout layout) { }
}
```

Now let's add the generator's inputs and outputs. To work properly, all the generator's inputs and outputs should:

- be properly initialized
- included in the inputs or outputs enumerator
- have a graphical representation in the generator's OnGUI function

So let's add the Input and Output variables. Inputs and outputs constructors generally take two arguments: the first is the name of the input/output as it is displayed in the node and the second is its type in the form of InoutType enum (InoutType.Map or InoutType.Objects). In addition, Input can have an optional "mandatory" property: if set to true it will display a red mark for unconnected input.

Warning: remove MyGenerator from a graph before applying the following code and create it again when scripts are compiled. Otherwise it will have a null input/output.

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{

    //input and output vars
    public Input input = new Input("Input", InoutType.Map, mandatory:true);
    public Output output = new Output("Output", InoutType.Map);

    public override void Generate (MapMagic.MapMagic.Chunk chunk) { }
    public override void OnGUI (Layout layout) { }
}
```

On generating MapMagic iterates a generator's inputs and outputs to process each of them. Enumerables are used to do this, just adding the variables will not make them accessible to MapMagic.

Each input should be included with "yield return" keyword in an overridden Inputs enumerable, while each output should be included in an overridden outputs enumerable.

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    //input and output vars
    public Input input = new Input("Input", InoutType.Map, mandatory:true);
    public Output output = new Output("Output", InoutType.Map);

    //including in enumerator
    public override IEnumerable<Input> Inputs() { yield return input; }
    public override IEnumerable<Output> Outputs() { yield return output; }

    public override void Generate (MapMagic.MapMagic.Chunk chunk) { }
    public override void OnGUI (Layout layout) { }
}
```

If the generator has, let's say, three inputs and two outputs it should start with something like this:

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
```

```
    public Input firstIn = new Input("First", InoutType.Map);
    public Input secondIn = new Input("Second", InoutType.Map);
    public Input thirdIn = new Input("Third", InoutType.Map);
    public Output firstOut = new Output("First", InoutType.Map);
    public Output secondOut = new Output("Second", InoutType.Map);
    public override IEnumerable<Input> Inputs()
        { yield return firstIn; yield return secondIn; yield return thirdIn; }

    public override IEnumerable<Output> Outputs()
        { yield return firstOut; yield return secondOut; }

    public override void Generate (MapMagic.MapMagic.Chunk chunk) { }
    public override void OnGUI (Layout layout) { }
}
```

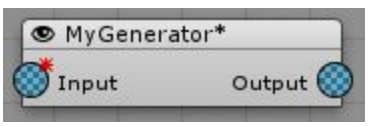
Now we have a correct and properly set-up generator. It has only two drawbacks: it does nothing and it has no displayed parameters in the GUI - not even input and output connection nodes.

We will fix the last one by exposing the input and output in the OnGUI function. This function is called when the generator node is rendered on the graph and uses a Layout wrapper, instead of UnityEditor.EditorGUILayout, which allows element scrolling, zooming and dragging.

Right now we will not dig into Layout, we will just add a new line with a 20-pixel height(with zoom=100%) layout.Par function and then draw the input and output (a circle with an icon and a label with input/output name) at that line by calling their own OnGUI functions.

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    //input and output vars
    public Input input = new Input("Input", InoutType.Map, mandatory:true);
    public Output output = new Output("Output", InoutType.Map);

    //including in enumerator
    public override IEnumerable<Input> Inputs() { yield return input; }
    public override IEnumerable<Output> Outputs() { yield return output; }

    public override void Generate (MapMagic.MapMagic.Chunk chunk) { }

    public override void OnGUI (Layout layout)
    {
        layout.Par(20);
        input.DrawIcon(layout);
        output.DrawIcon(layout);
    }

}
```

Now you can see that MyGenerator has two map connections, and the input connection is displayed as mandatory.



Our 3 inputs and 2 outputs generator will have 3 lines, one input and one (or none) output per line:

```
    public override void OnGUI (Layout layout)
    {
        layout.Par(20); firstIn.DrawIcon(layout); firstOut.DrawIcon(layout);
        layout.Par(20); secondIn.DrawIcon(layout); secondOut.DrawIcon(layout);
        layout.Par(20); thirdIn.DrawIcon(layout);
    }
```

Now let's add a generate function. Let's say it will do a simple function - it will invert an input map's value(1-value). The same effect could be achieved with curves but sometimes it is handy to use special generator.

Here is the body of the Generate function. This function overrides the base Generator's function. It takes a chunk as an argument and it is void - does not return anything. Later you will see that it loads maps and object hashes from the chunk and saves the processed result back to the chunk.

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    //input and output properties
    public Input input = new Input("Input", InoutType.Map, mandatory:true);
    public Output output = new Output("Output", InoutType.Map);

    //including in enumerator
    public override IEnumerable<Input> Inputs() { yield return input; }
    public override IEnumerable<Output> Outputs() { yield return output; }

    public override void Generate (MapMagic.MapMagic.Chunk chunk) { }

    public override void OnGUI (Layout layout)
    {
        layout.Par(20); input.DrawIcon(layout); output.DrawIcon(layout);
    }
}
```

Now we will load inputs using the input.GetObject function. This function returns an object type, so we have to cast it to the map's type, which is internally called "Matrix". This matrix will be called **src** (source).

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    //input and output properties
    public Input input = new Input("Input", InoutType.Map, mandatory:true);
    public Output output = new Output("Output", InoutType.Map);
```

```
    //including in enumerator
    public override IEnumerable<Input> Inputs() { yield return input; }
    public override IEnumerable<Output> Outputs() { yield return output; }

    public override void Generate (MapMagic.MapMagic.Chunk chunk)
    {

        Matrix src = (Matrix)input.GetObject(chunk);

    }

    public override void OnGUI (Layout layout)
    {
        layout.Par(20); input.DrawIcon(layout); output.DrawIcon(layout);
    }
}
```

Now let's add three guard clauses to the Generate function. They will exit the function before executing its body to prevent errors or to save computing time.

The first one occurs in the case where an input gets a null object - if it is not connected or the previous generator returned null. In this case this generator should not set any objects.

By the way, you can check if the generate thread is still running by checking the chunk.stop boolean value. If it is set to true all generating processes should end as soon as possible. It's a good idea to check if the chunk has not stopped regularly in a function body, because due to the multithreaded approach it can change its value at any time, and often it does - **during the generate function execution** from the main thread.

The third way the function can exit early is if the generator is not enabled. But in this case it should pass the input object to the output. So it uses output.SetObject function and returns.

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    //input and output properties
    public Input input = new Input("Input", InoutType.Map, mandatory:true);
    public Output output = new Output("Output", InoutType.Map);

    //including in enumerator
    public override IEnumerable<Input> Inputs() { yield return input; }
    public override IEnumerable<Output> Outputs() { yield return output; }

    public override void Generate (MapMagic.MapMagic.Chunk chunk)
    {
        Matrix src = (Matrix)input.GetObject(chunk);

        if (src==null || chunk.stop) return;
        if (!enabled) { output.SetObject(chunk, src); return; }

    }

    public override void OnGUI (Layout layout)
    {
```

```
        layout.Par(20); input.DrawIcon(layout); output.DrawIcon(layout);
    }
}
```

We're now at the most interesting part. We will do some simple calculations to invert a map.

But before doing that let's make sure that it will not modify our original "src" matrix. This matrix could be used by other generators and, more importantly, it will be used by this generator each time it's parameter changes if "save intermediate results" is checked in the MapMagic settings. So do not forget this simple rule: do not change the input maps or object hashes, use new matrices or hashes to write values.

So we will create a new matrix to write our changed src values and call it **dst** (destination). This matrix should have the same size and the same coordinates as the source matrix. The size and offset information is kept in a matrix.rect struct. We will use src.rect in a new matrix constructor.

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    //input and output properties
    public Input input = new Input("Input", InoutType.Map, mandatory:true);
    public Output output = new Output("Output", InoutType.Map);

    //including in enumerator
    public override IEnumerable<Input> Inputs() { yield return input; }
    public override IEnumerable<Output> Outputs() { yield return output; }

    public override void Generate (MapMagic.MapMagic.Chunk chunk)
    {
        Matrix src = (Matrix)input.GetObject(chunk);

        if (src==null || chunk.stop) return;
        if (!enabled) { output.SetObject(chunk, src); return; }

        Matrix dst = new Matrix(src.rect);

    }

    public override void OnGUI (Layout layout)
    {
        layout.Par(20); input.DrawIcon(layout); output.DrawIcon(layout);
    }
}
```

In order to invert the matrix we have to iterate all the src pixels and set them to *dst = 1 - src.* It's very simple to do this using matrices. Matrix is a wrapper of a standard single-dimensional array of floats, with an interface that of a 2D array but makes it a bit faster.

And it takes matrix offset into account - so that matrix coordinates start not from zero, but from the offset values.

For example, if we want to get some pixel at a coordinates, let's say [768,256] we will call matrix[768,256]. If we want to iterate a matrix that has an offset [1024,0] and a size [512,512] we should use the following code:

```
for (int x=1024; x<1024+512; x++)
   for (int y=0; y<0+512; y++)
      float val = matrix[x,y];
```

By the way, since the matrix is used for 3D objects like terrain, and it is applied horizontally, it's better to use the Z-axis name instead of Y(which usually indicates height).

So, the simple invert generator code should look like this:

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    //input and output properties
    public Input input = new Input("Input", InoutType.Map, mandatory:true);
    public Output output = new Output("Output", InoutType.Map);

    //including in enumerator
    public override IEnumerable<Input> Inputs() { yield return input; }
    public override IEnumerable<Output> Outputs() { yield return output; }

    public override void Generate (MapMagic.MapMagic.Chunk chunk)
    {
        Matrix src = (Matrix)input.GetObject(chunk);

        if (src==null || chunk.stop) return;
        if (!enabled) { output.SetObject(chunk, src); return; }

        Matrix dst = new Matrix(src.rect);

        for (int x=src.rect.offset.x; x<src.rect.offset.x+src.rect.size.x; x++)
           for (int z=src.rect.offset.z; z<src.rect.offset.z+src.rect.size.z; z++)
               dst[x,z] = 1 - src[x,z];

    }

    public override void OnGUI (Layout layout)
    {
        layout.Par(20); input.DrawIcon(layout); output.DrawIcon(layout);
    }
}
```

We can speed up the code a bit to avoid summing of rect with offset using the Coord struct. Think of Coord like a Vector2, except that it uses ints instead of floats.

```
Coord min = src.rect.Min; Coord max = src.rect.Max;
for (int x=min.x; x<max.x; x++)
   for (int z=min.z; z<max.z; z++)
```

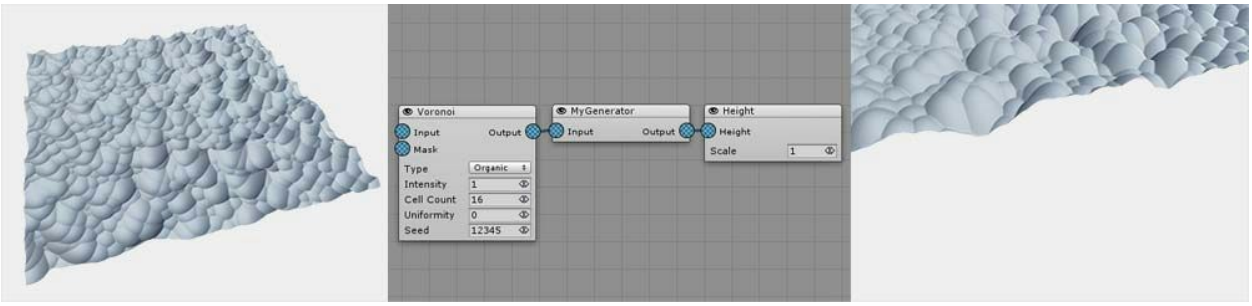
```
        dst[x,z] = 1 - src[x,z];
```

And the final step to make a generator work - it should store the generated result using the output.SetObject function. This function takes two arguments: the first one is a chunk to store an object and the second one is the result object itself.

But before storing an object check that the thread is not stopped. Just in case.

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    //input and output properties
    public Input input = new Input("Input", InoutType.Map, mandatory:true);
    public Output output = new Output("Output", InoutType.Map);

    //including in enumerator
    public override IEnumerable<Input> Inputs() { yield return input; }
    public override IEnumerable<Output> Outputs() { yield return output; }

    public override void Generate (MapMagic.MapMagic.Chunk chunk)
    {
        Matrix src = (Matrix)input.GetObject(chunk);

        if (src==null || chunk.stop) return;
        if (!enabled) { output.SetObject(chunk, src); return; }

        Matrix dst = new Matrix(src.rect);

        Coord min = src.rect.Min; Coord max = src.rect.Max;
        for (int x=min.x; x<max.x; x++)
           for (int z=min.z; z<max.z; z++)
              dst[x,z] = 1 - src[x,z];

        if (chunk.stop) return;
        output.SetObject(chunk, dst);

    }

    public override void OnGUI (Layout layout)
    {
        layout.Par(20); input.DrawIcon(layout); output.DrawIcon(layout);
    }
}
```

Now if we create this generator and connect it to the organic Voronoi Generator you will see that it produces bubbles instead of caverns - so it works the way we expect.



It has one drawback though - it creates a map somewhere at the top of the terrain which makes it hard to edit or blend the map with the other ones. This happens because we subtracted src values from 1 - the top terrain level. So if it was replaced with a custom value we will get a more adequate result. And note that we should monitor the dst value to prevent it from going below zero.

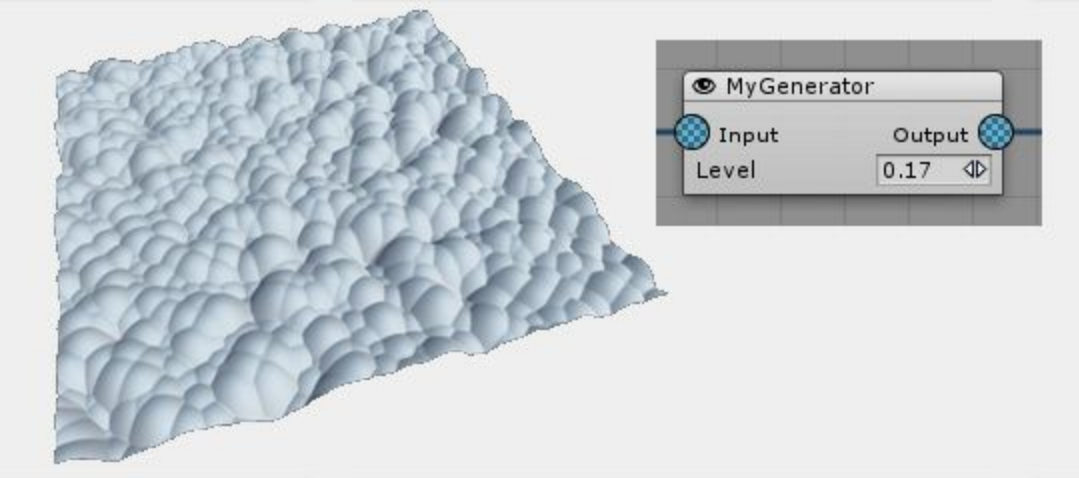
So let's add a "level" variable and expose it on the GUI using the layout.SmartField. A smart field is a common MapMagic field with a text input and a draggable icon to the right. The SmartField function takes two required arguments: the ref of a value we want to expose and the text that should be displayed in front of it. And we will use two additional parameters: minimum value (min:0) and maximum value (max:1) because, obviously, the level could not be less than zero or higher than 1.

By default there is no need to create a new line using layout.Par() because this function is already called in a smart field unless the newLine parameter is set to false.

```
[System.Serializable]
[GeneratorMenu (menu="MyGenerators", name ="MyGenerator", disengageable = true)]
public class MyCustomGenerator : Generator
{
    //input and output properties
    public Input input = new Input("Input", InoutType.Map, mandatory:true);
    public Output output = new Output("Output", InoutType.Map);

    //including in enumerator
    public override IEnumerable<Input> Inputs() { yield return input; }
    public override IEnumerable<Output> Outputs() { yield return output; }

    public float level = 1;

    public override void Generate (MapMagic.MapMagic.Chunk chunk)
    {
        Matrix src = (Matrix)input.GetObject(chunk);

        if (src==null || chunk.stop) return;
        if (!enabled) { output.SetObject(chunk, src); return; }

        Matrix dst = new Matrix(src.rect);

        Coord min = src.rect.Min; Coord max = src.rect.Max;
        for (int x=min.x; x<max.x; x++)
           for (int z=min.z; z<max.z; z++)
```

```
                float val = level - src[x,z];
                dst[x,z] = val>0? val : 0;


            if (chunk.stop) return;
            output.SetObject(chunk, dst);
        }

        public override void OnGUI (Layout layout)
        {
            layout.Par(20); input.DrawIcon(layout); output.DrawIcon(layout);

            layout.Field(ref level, "Level", min:0, max:1);

        }
}
```

Now we've got a nice looking generator, which has an input, output, and an adjustable parameter. It's also seamless: It appears as if it's been in MapMagic forever, but the really cool thing is that it is literally a plugin for MapMagic, we've done it without touching the main code. It could be removed by just deleting the file, sent to a colleague or published online.



Event system

OnApply

OnApply is a static event before applying any type of the output to the terrain. With it's help you can pre-process terrains and what is going to be applied with your scripts.
Delegate: ApplyCallback (Terrain terrain, object obj)
Usage example (your custom script):

```
public void OnEnable ()
{
    MapMagic.OnApply -= MyPreProcess; //just in case it was not called on disable
    MapMagic.OnApply += MyPreProcess;
}

public void OnDisable ()
{
    MapMagic.OnApply -= MyPreProcess;
}

public void MyPreProcess (Terrain terrain, object obj)
{
    //processing heightmap
    if (obj is float[,]) //terrain.terrainData.SetHeights recieve float[,] as heightmap
    {
        float[,] heightmap = (float[,])obj;
        //do something with heightmap
        //or set the terrain layer
        //or assign some script to terrain
    }

    //processing splatmap
    if (obj is float[,,]) //alphamap is float[,,]
    {
        float[,,] splatmap = (float[,,])obj;
        //do something with splatmap
        //don't assign a script here if it is already assigned in heightmap
    }

    //processing grass
    if (obj is int[][,]) {}

    //etc
}
```

Generate Change Events

You can use the static generate change events to monitor current generate state and pre- or postprocess terrains before (or after) they have been modified with MapMagic:

- OnGenerateStarted - called before the terrain is started to generate
- OnGenerateCompleted - called when the terrain generate is complete, but before the generate result is applied
- OnApplyCompleted - called when the generate change is applied to the terrain and it will not be modified with MM further.

Delegate: ChangeEvent (Terrain terrain)

Usage example (your custom script):

```
public void OnEnable ()
{
```

```
    MapMagic.OnGenerateStarted  -= MyGenerateNotify;
    MapMagic.OnGenerateStarted  += MyGenerateNotify;
}

public void OnDisable ()
{
    MapMagic.OnGenerateStarted -= MyGenerateNotify;
}

public void MyGenerateNotify (Terrain terrain)
{
    //notifying script that generate has started
}
```