Basic Edition User Guide (v1.1.0)

The #1 Low Poly Terrain Engine for Unity

Create stunning stylized environment with least effort, in no time!

Online version: https://docs.pinwheel.studio/polaris/

Homepage: http://pinwheel.studio

Join the community on Facebook Page, Facebook User Group and Forum

Polaris is a hybrid procedural low-poly terrain engine, a complete solution for creating stunning low-poly environment, especially built for Unity developer. This is a must-have asset which saves you thousands hours of work and tons of effort, give you more time and resources to focus on other aspects of you game

This User Guide is created to help you get started with the tool, including the most basic information about its features and workflow, as well as best practices, tips and tricks from its creator and showcasing amazing work from other successful developers.

To see what's new in this version, please visit this page.

For business, please contact: hello@pinwheel.studio

For support, report a bug or request a feature, please contact: support@pinwheel.studio

TABLE OF CONTENTS

OVERVIEW

What's new in this release (v1.1.0)

- Even more useful with the Geometry, Color and Foliage painter from Pro version.
- Many bugs fixing.
- Performance optimization.

What is Polaris Low Poly Terrain Engine?

Polaris is a user friendly terrain engine dedicated to help you create gorgeous landscapes, deeply focused on low-poly and stylized world, that can run well on both Mobile and Desktop applications, save you a lot of time and effort!

Polaris is carefully designed to totally eliminate the frustration of complexity, giving you the fastest and easiest experience of exploring and creating, empowered by the latest techniques from the inside, giving the highest quality of achievement. Even if you are a beginner or an advanced creator, it is just the right tool for you!

Polaris cleverly combines the traditional Sculpting method with the modern Procedural one, offer you the most flexible workflow, feel free to unleash your power of creativity!

Polaris also comes with sample assets and example scenes for you to play with. You can even use them in your commercial projects. Have fun!

Why Polaris?

Low Poly environment has become the most popular trending in game development, especially for small and medium size projects. However, creating a gorgeous looking low poly landscape is NOT an easy task. It DOES require a lot of skills and efforts. You can spend several hours in some modelling applications to sculpt the terrain, then import it to Unity, and repeat until satisfied. It's hard, it's rough, it's inconvenient, it's time consuming, and that's the point, time is money, save time is save money!

Polaris is created to simplify the process. With Polaris, you are SAVING, NOT SPENDING!

You can discover other great terrain engines out there, but they are dedicated for AAA environment, while Polaris is deeply focused on low poly and stylized. Choose the right tool, wisely!

Key features[1]

- Super easy to use, clean and tidy interface, in-depth documentation.
- Support 2 types of terrain, one for small & stylized, with simple setup, one for larger scale with more advanced features.
- Procedurally generating terrain base shape, with various settings, cleverly combines between Height Map, Noise, Wave and Curve, also support for clipping terrain shape based on vertex height or snap elevation value to create blocky terrain.
- Import height map data from a regular texture, no RAW file needed.
- Advanced Geometry painter for further polishing terrain shape, with various painting modes and multiple built-in brushes.
- Advanced Foliage painter for spawning objects into the scene, with various spawning rules set and multiple built-in brushes.
- Advanced Color painter for further polishing terrain color, or creating hand painted world, with handy color picker and swatches, multiple built-in brushes.
- Advanced Splat painter for painting texture onto the surface, easily creating roads, grasses, with multiple built-in brushes.
- Advanced Mask painter for taking action on a specific terrain region, such as lock it out from editing.
- Ability to create your own brush gallery.
- Advanced optimization techniques, including LODs management, automatic billboard asset creation, automatic collider fit.

[1] Some features only available in Pro version

- Other utility features like saving procedural mesh, extract data from terrain, etc.
- Work perfectly with other world building tools like Gaia, Map Magic.
- High quality, fully documented C# source code included.

Key benefits

- Fully satisfied result, gorgeous terrain achieved within clicks.
- Save a ton of time and money!
- Easy to migrate existing work from other world building tools.
- Continuous development, more interesting features ahead.
- Responsive customer support.

External Resources

The following sections will describe basic concepts of how to use Polaris in Unity Editor, for more information such as tutorials, best practices, development logs or newest announcements, please visit the following links:

- Pinwheel Studio on Facebook.
- Pinwheel Studio on Twitter.
- Polaris page on Facebook.
- Polaris User group on Facebook.
- Unity Forum Thread.

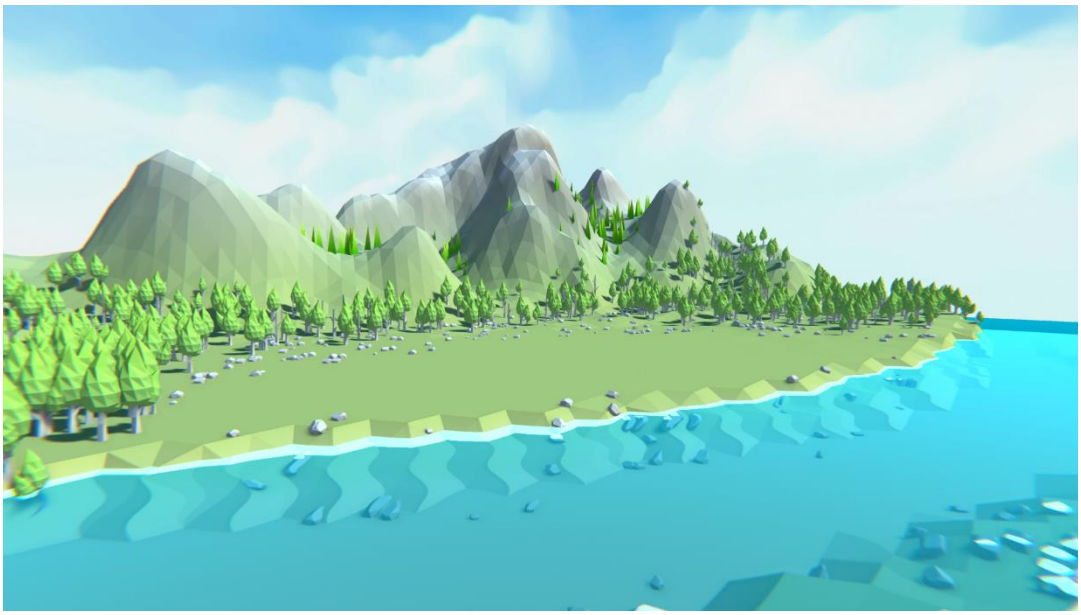
SIMPLE TERRAIN SYSTEM

Concept

The Simple Terrain System is dedicated for creating small terrain with simple geometry, with a few trees and other objects.

This terrain uses a single mesh to represent the terrain geometry, thus the terrain size is limited somewhere around 100x100 based on your settings and use simple optimization techniques like mesh combination.

This system is best suit for Mobile device or Low-End devices.

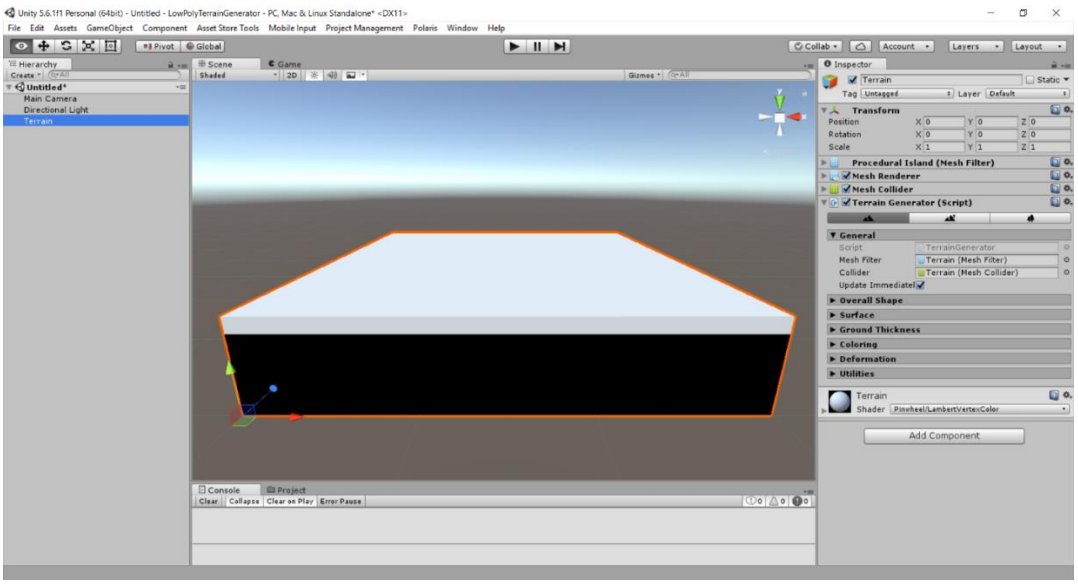


Creating a basic terrain

Create new terrain

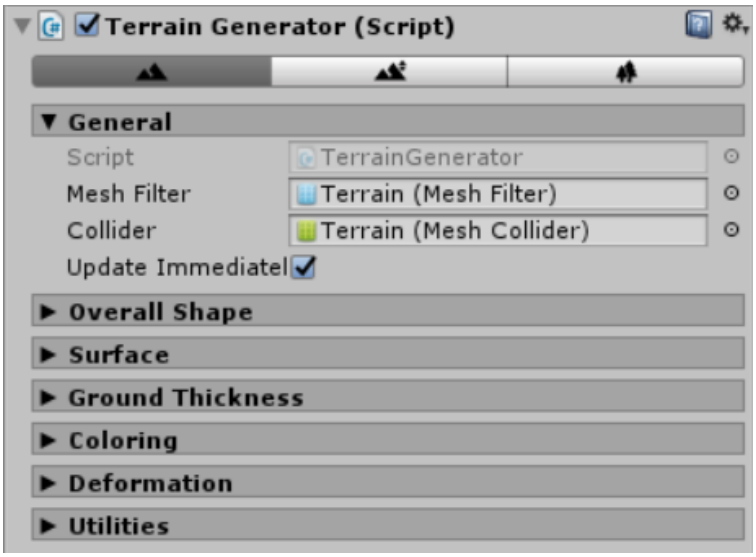
In the current scene, right-click on the Hierarchy window, then select Polaris > Simple Low-Poly Terrain, a new terrain will be created at the world origin point, with components all set.

You will see something like this:



General settings

In the Inspector window, click on the General fold out to open it, you will see the properties as listed below:

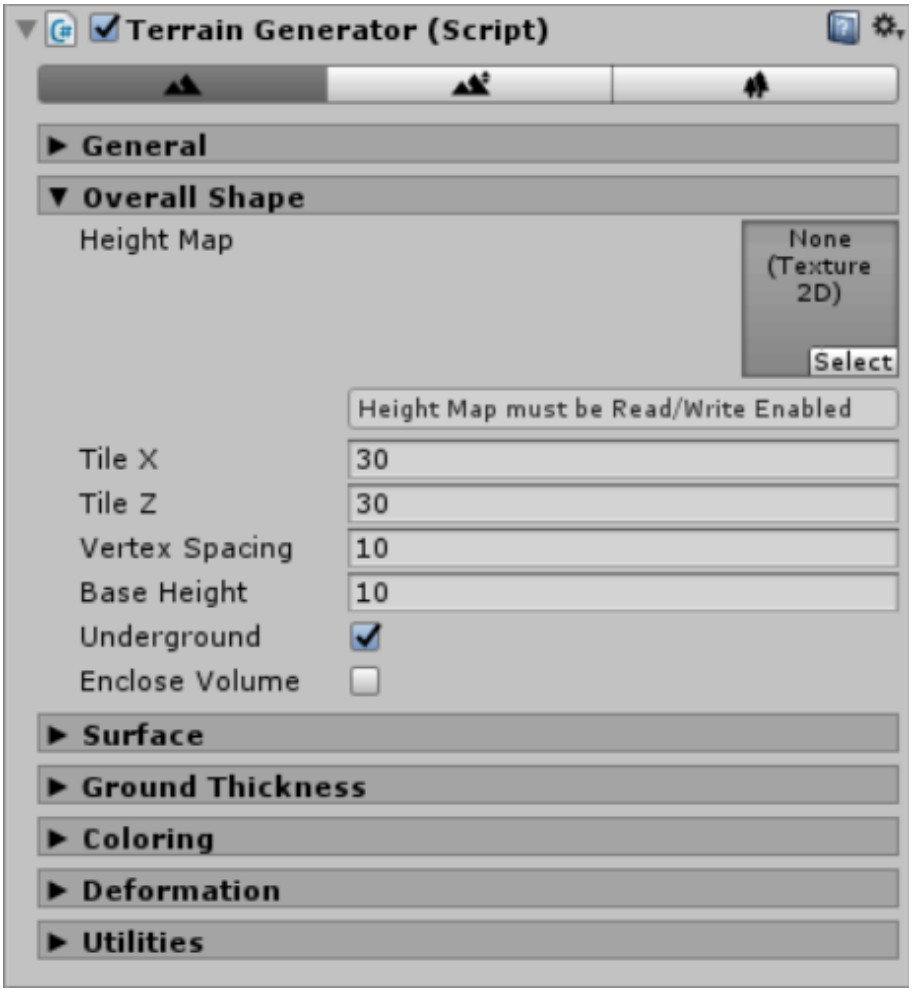


- Script: The MonoScript associate with this component, double click to open the script if you want to see what's inside.

- Mesh Filter: The Mesh Filter component to assign the mesh to. If null, the terrain will not be rendered but instead stored in the GeneratedMesh property.
- Collider: The Mesh Collider component for this terrain. Assign this if you want the player to walk on it. This property is assigned automatically if you use Geometry Painter of Environmental Painter.
- Update Immediately: Should the terrain update right after Inspector changed? Uncheck this to save some processing power on low end machine.

Overall Shape settings

In the Inspector window, click on the Overall Shape foldout to open it, you will see the properties as listed below:



- Height Map: A grayscale map to determine terrain elevation (see below).
- Tile X: Number of tiles on X-axis.
- Tile Z: Number of tiles on Z-axis.
- Vertex Spacing: Distance between 2 vertices.
- Base Height: Height of the base (underground part)
- Underground: Should it generate the underground & ground thickness layer?

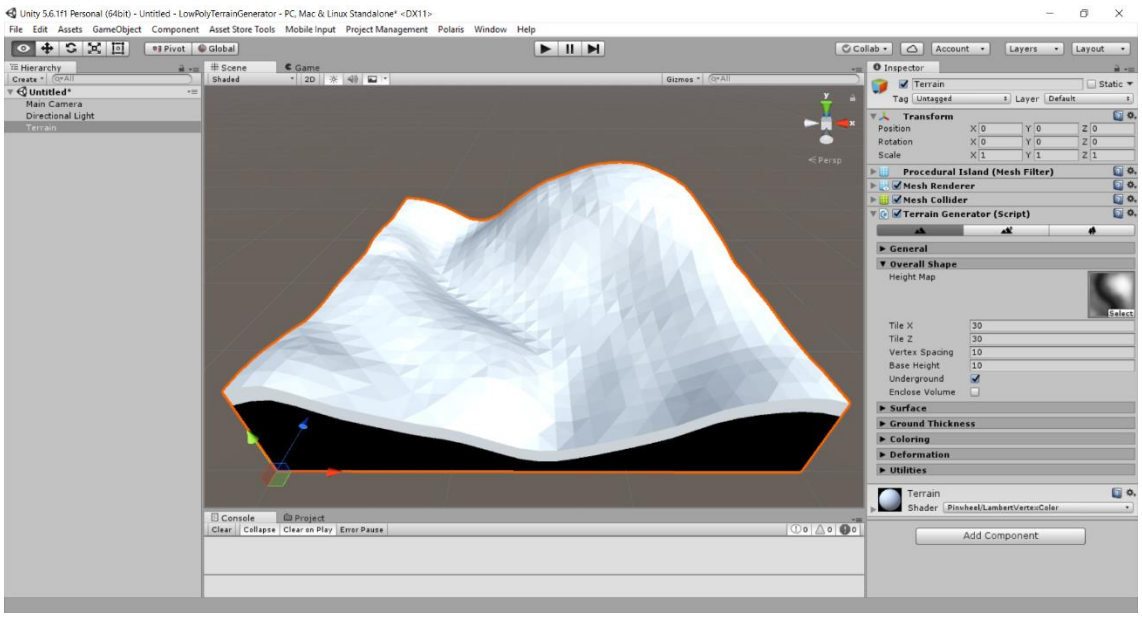
- Enclose Volume: Should it generate some extra triangles to close the terrain volume? Look at the terrain from the bottom up to see the effect.

The height of your terrain can be determined by a height map, which only contain grayscale pixels, where black is the lowest and white is the highest altitude. For example, we have this map:

After sketching your height map, import it into Unity and set its importing options as follow:

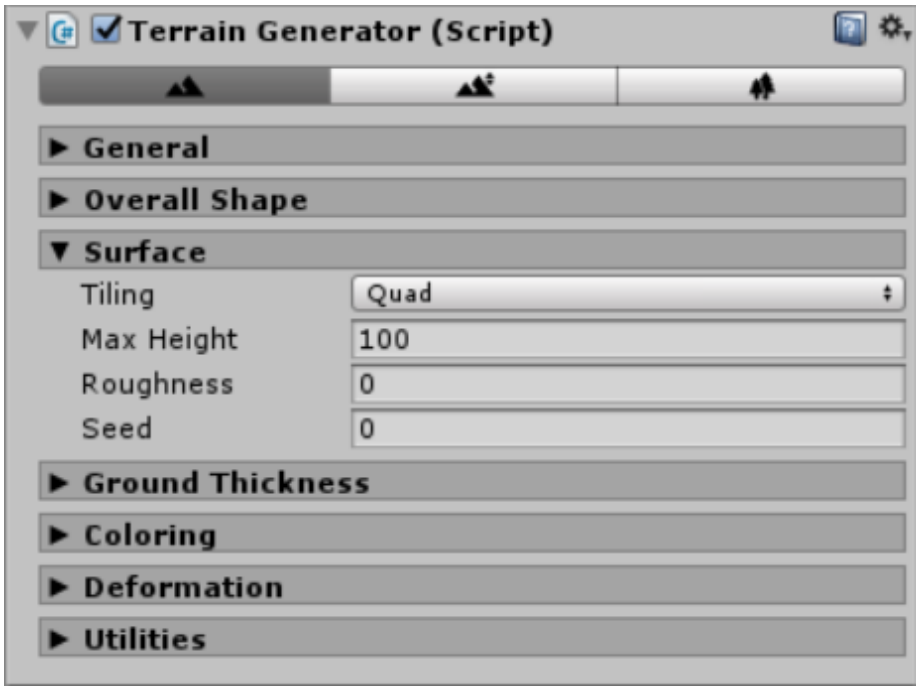
- Alpha Source: set to **From Gray Scale**
- Read/Write Enable: set to **True**
- Wrap Mode: set to **Clamp**

Then, assign it to the Height Map slot and see the result:



Surface settings

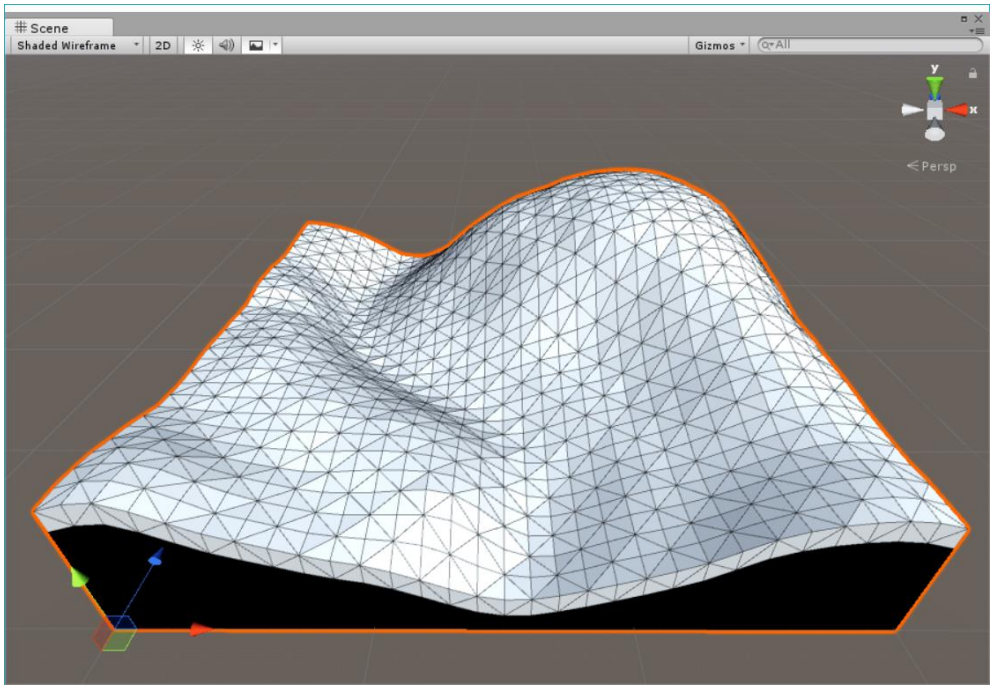
In the Inspector window, click on the Surface foldout to open it, you will see the properties as listed below:



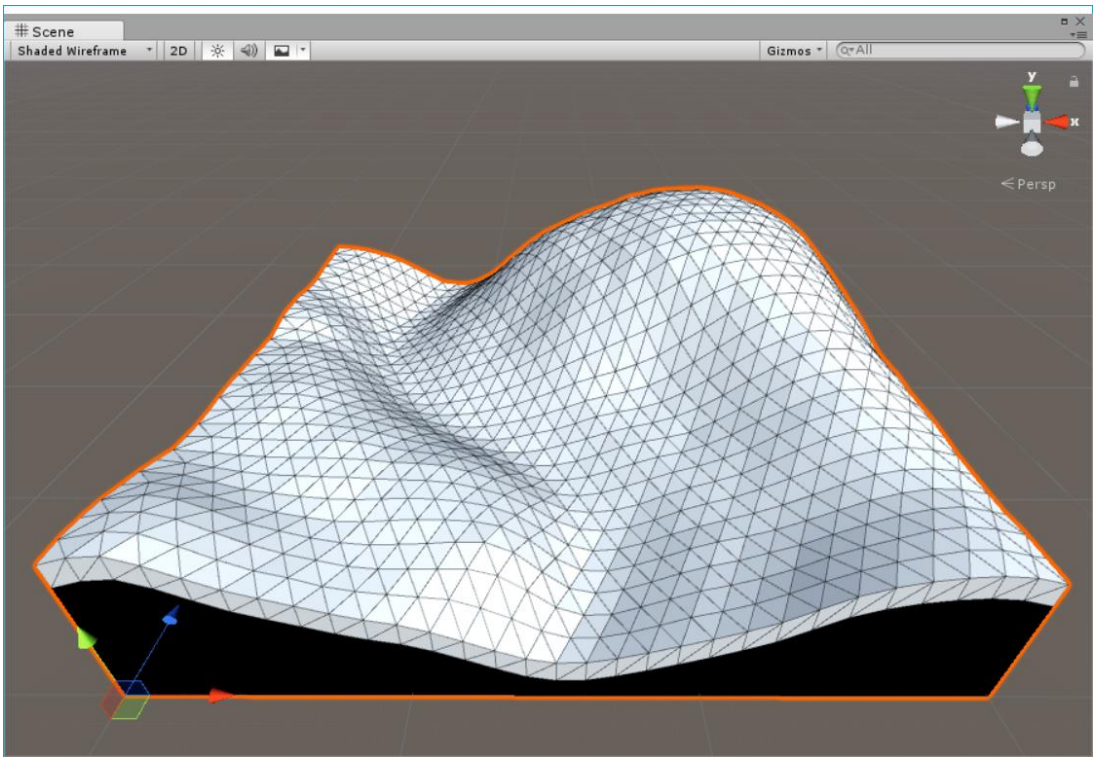
- Tiling: Which tiling pattern to use on the surface? Quad or Hexagon?

- Max Height: The surface maximum altitude.
- Roughness: Should we add some randomness to the surface using Perlin noise?
- Seed: A number used to generate random noise, different seed generate different noise pattern.

Note: Vertex displacement using Perlin noise is a multiply and un-normalized operation, that means it is clamped by the Height Map and noise value at a point will not change if you resize the terrain dimension (keep Roughness and Seed unchanged, of course)



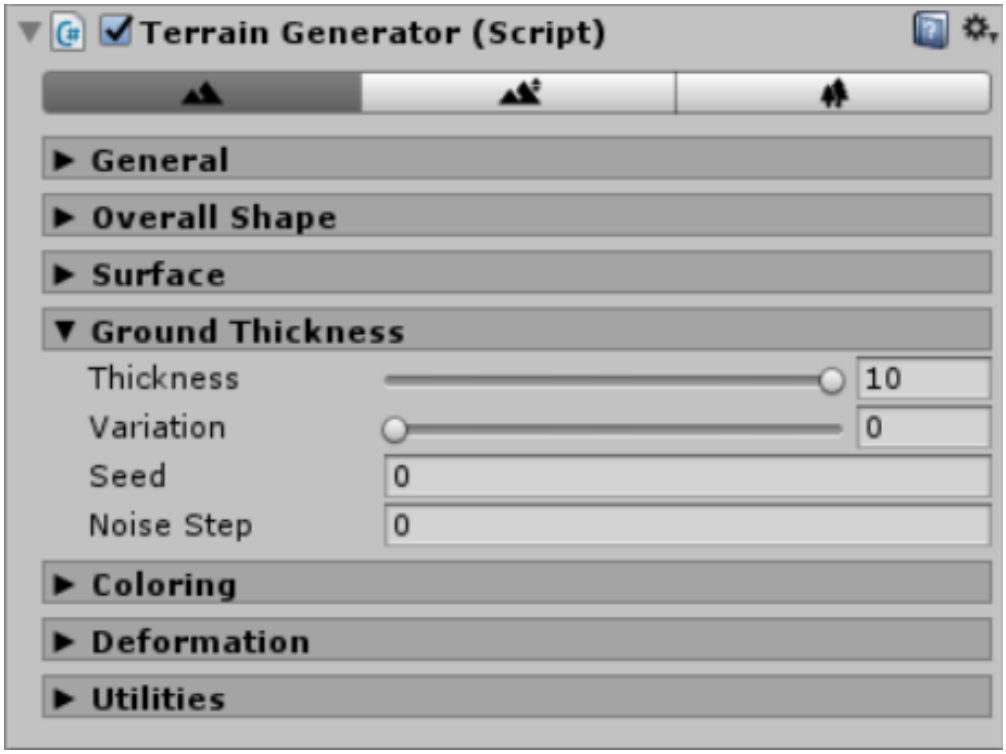
Quad Tiling



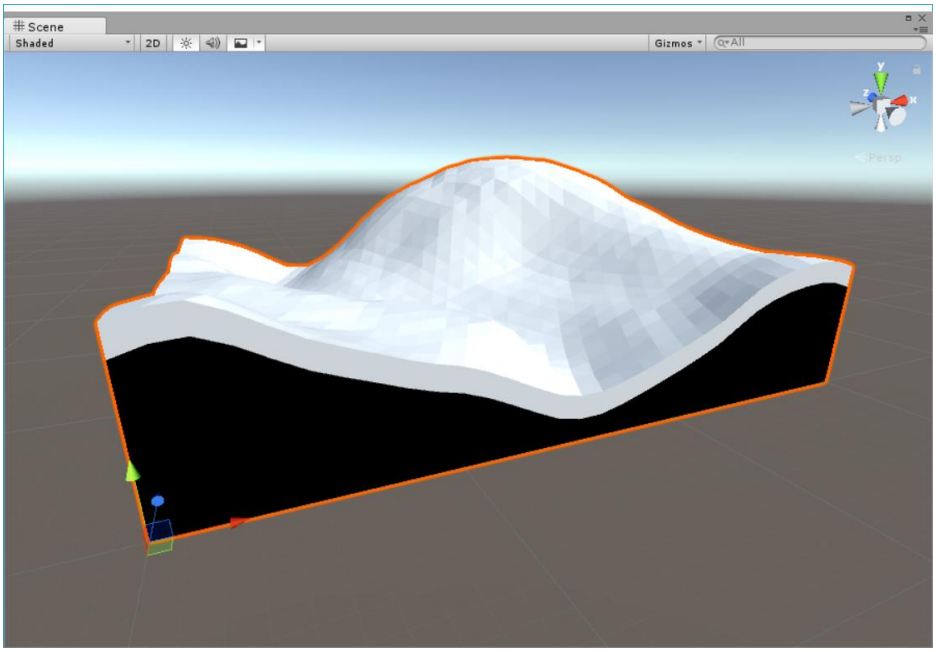
Hex Tiling

Ground Thickness settings

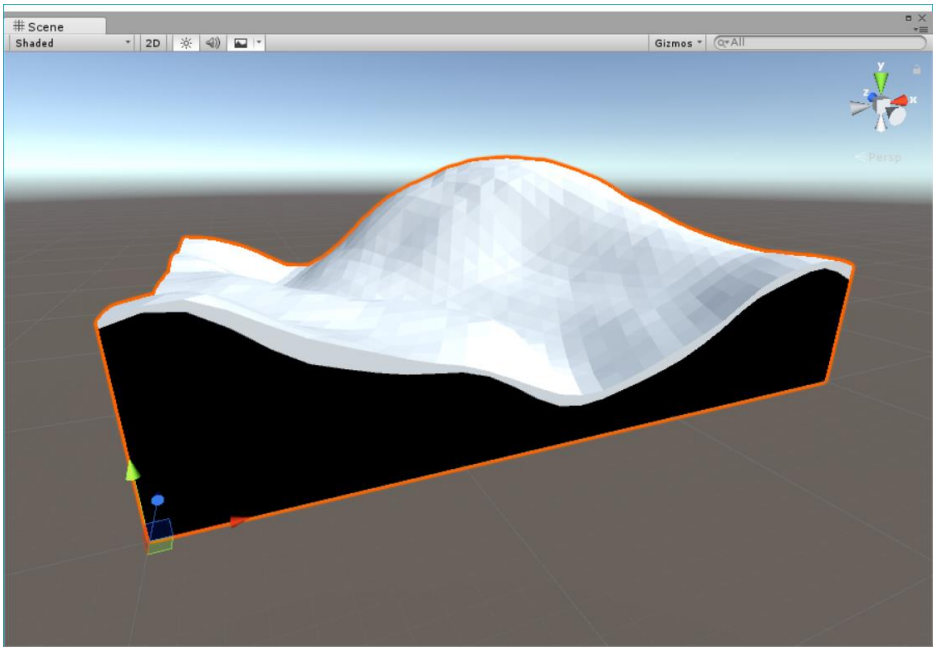
In the Inspector window, click on the Ground Thickness foldout to open it, you will see the properties as listed below:



- Thickness: The thickness of surface layer. This value is clamp between zero and Base Height in Overall Shape settings.
- Variation: The variation rate of ground thickness across the surface.
- Seed: A number to sample noise, different seed generate different ground thickness variation.
- Noise Step: Control thickness variation smoothness/roughness.



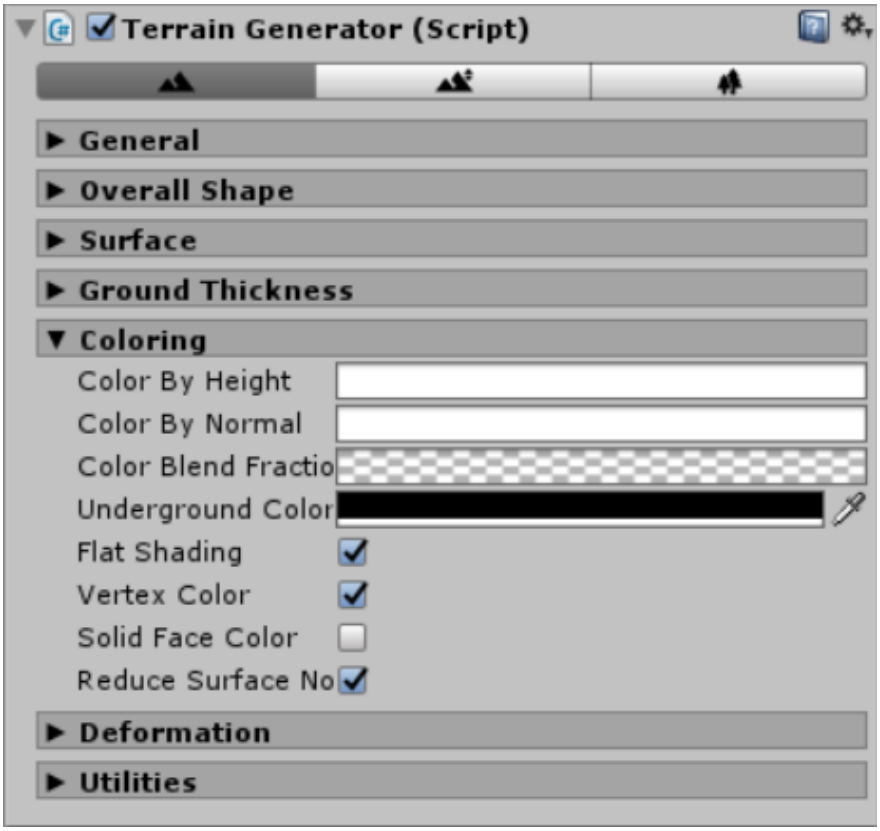
Terrain with no Thickness Variation



Terrain with Thickness Variation

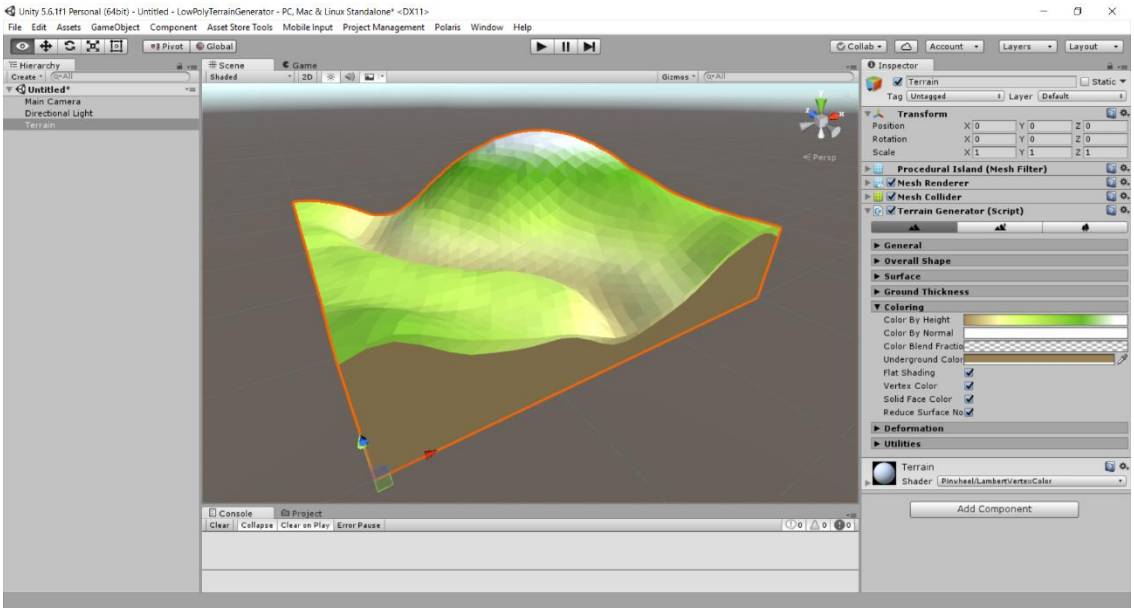
Coloring settings

In the Inspector window, click on the Coloring foldout to open it, you will see the properties listed as follow:

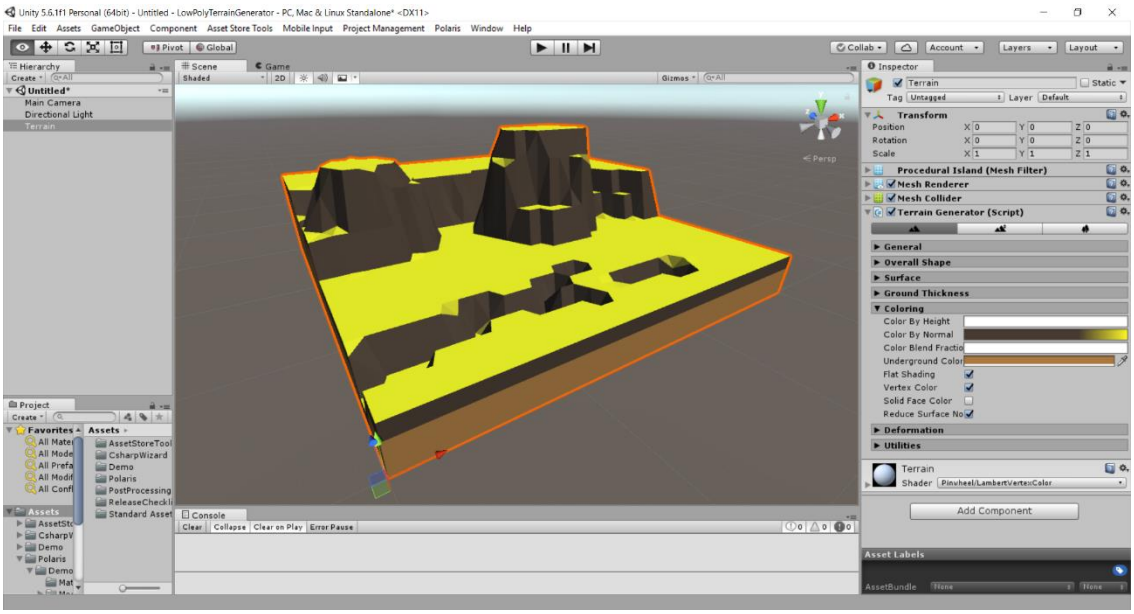


- Color By Height: Determine color of each vertex by its height.
- Color By Normal: Determine color of each vertex by its normal vector.
- Color Blend Fraction: A fraction to blend between Color By Height and Color By Normal, where alpha=0 is fully by height, and alpha=1 is fully by normal.
- Underground Color: Color of the underground part.
- Flat Shading: Toggle between flat and smooth shading, the number of vertices will be tripled when turning on.

- Vertex Color: Should it generate vertex color? Uncheck if you use a diffuse map for its material.
- Solid Face Color: If turn on, vertex color will not be blended across the triangle.
- Reduce Surface Noise: Use bilinear instead of point filter when sampling height map.



Color by height example

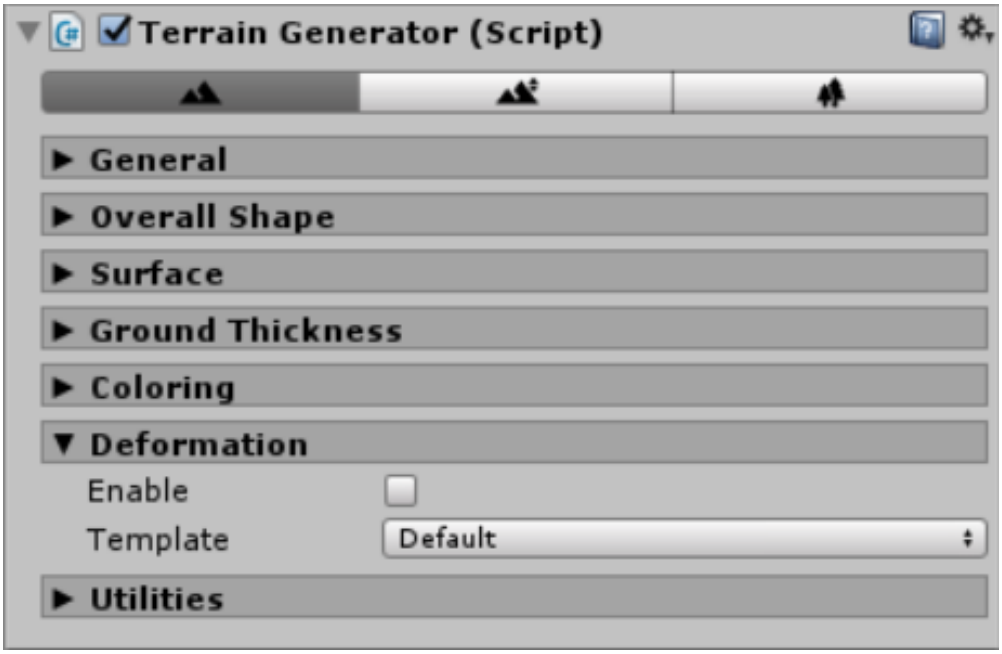


Color by normal example

Deformation settings

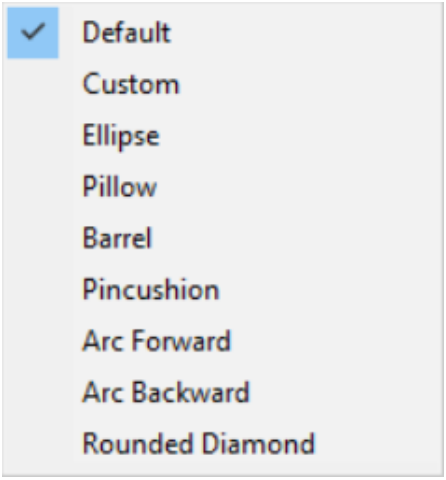
Traditional terrain engine often generates a mesh with rectangular mesh. But since Polaris is a stylized terrain engine, you can use it Bezier Based Vertex Deformation feature to create terrain with custom shape. Unleash your creativity!

In the Inspector window, click on the Deformation foldout to open it, you will see the properties as listed below:

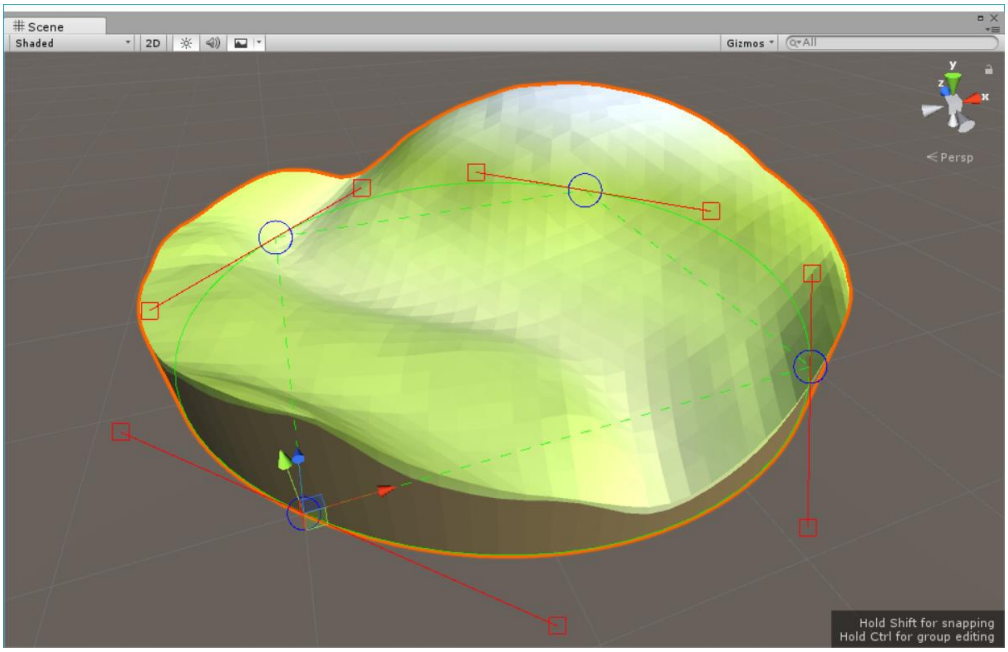


- Enable: Should the terrain be deformed?
- Template: Bezier template to choose from, switch to Custom to create your own.

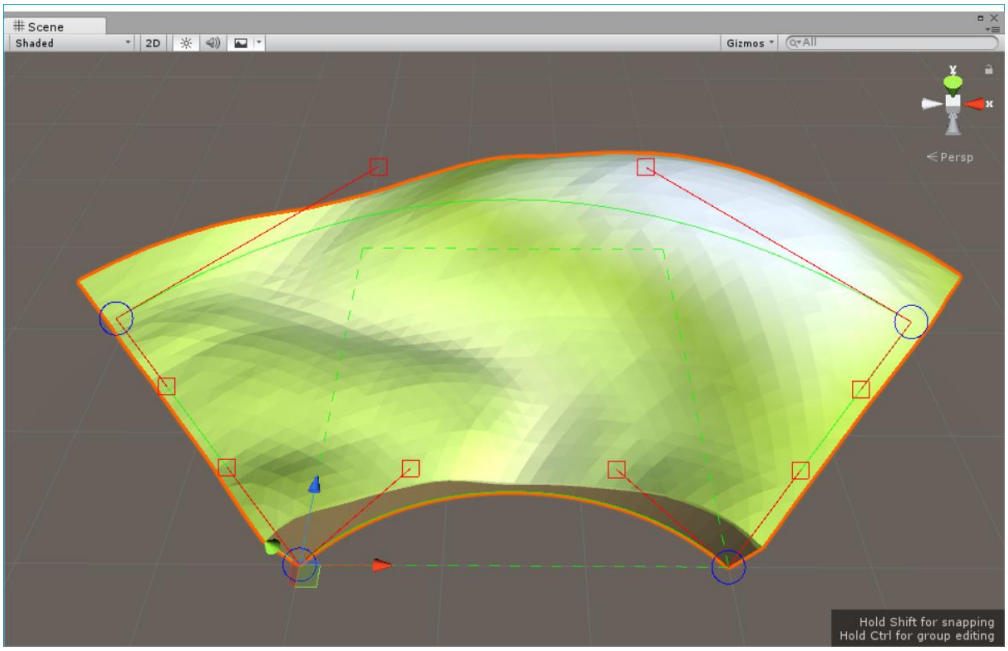
There are several templates for you to choose, more will be added in future updates:



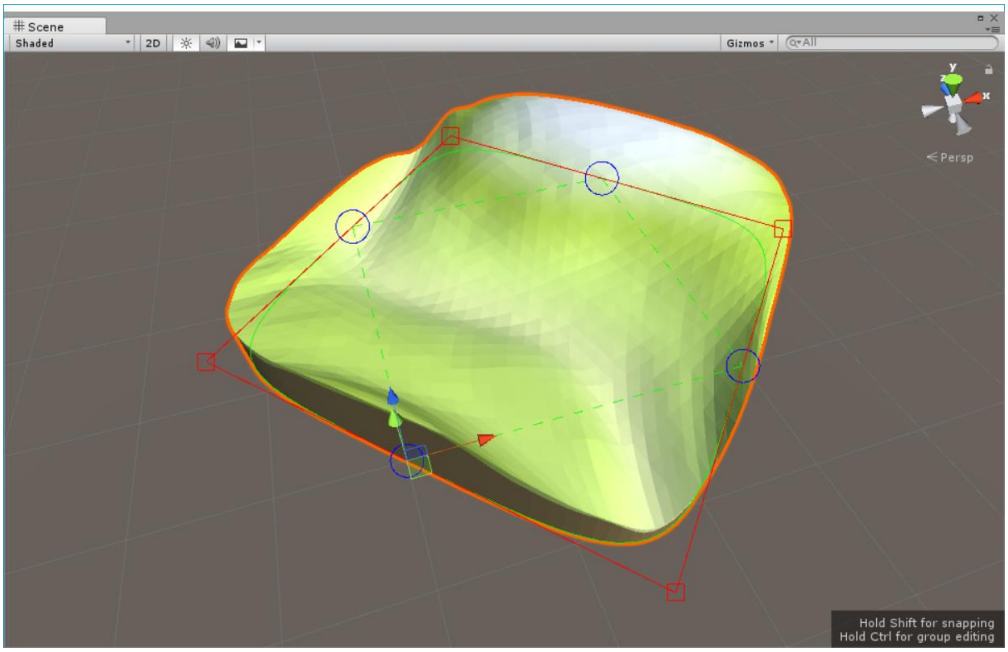
Once this features enabled, you will see the Bezier handles appear in the Scene view, try dragging them to create custom shapes.



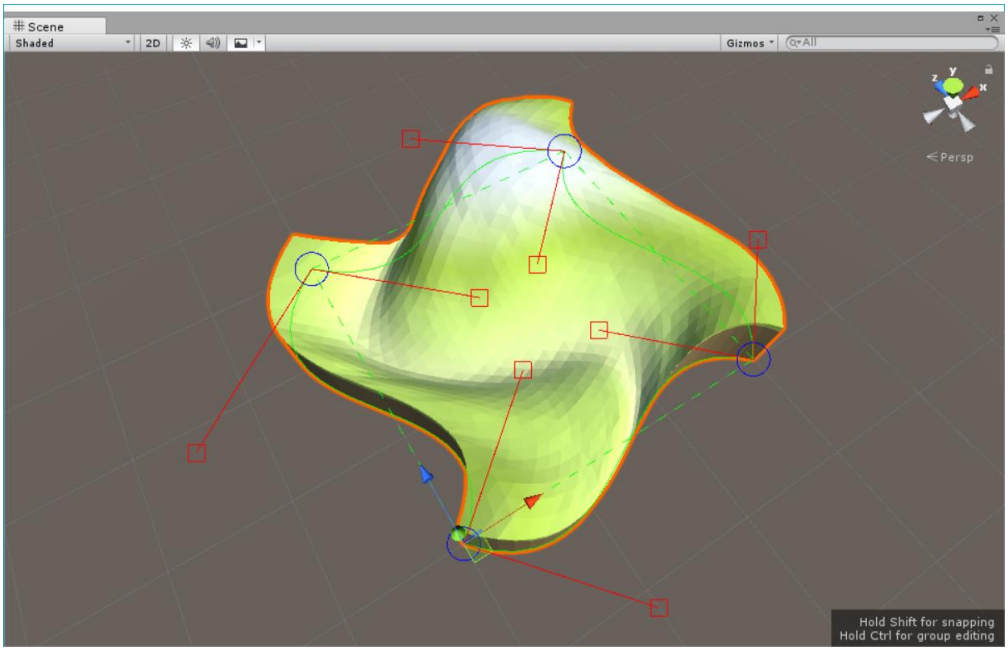
Ellipse deformation



Arc forward deformation



Rounded diamond deformation



Custom deformation

Note: Bezier Scene view handles are hidden when you collapse the Deformation foldout.

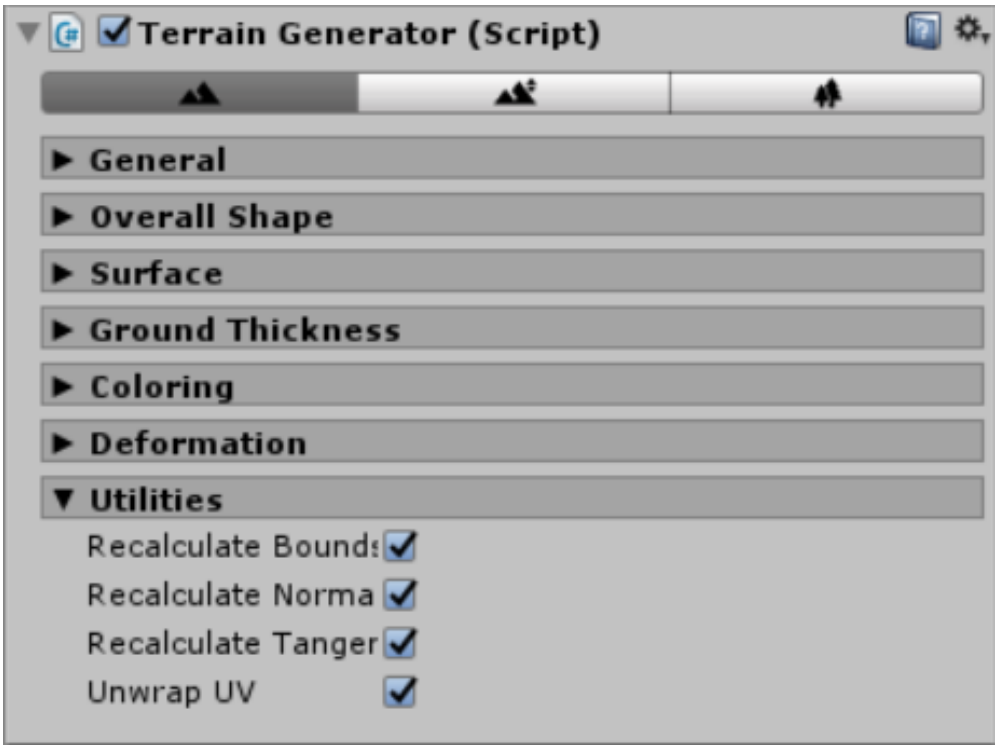
Keyboard shortcuts

Below are some keyboard shortcuts when working with the deformation tool in Scene view:

- Hold Shift: Snap the Bezier control points to an even coordinate in world space, or snap the handles to the 15, 30, 45, 60, etc. degree vectors.
- Hold Ctrl: Enable group editing. Move the 2 handles along with the control point, or reflect the other handle by the control point.

Utilities settings

In the Inspector window, click on the Utilities foldout to open it, you will see the properties as listed below:

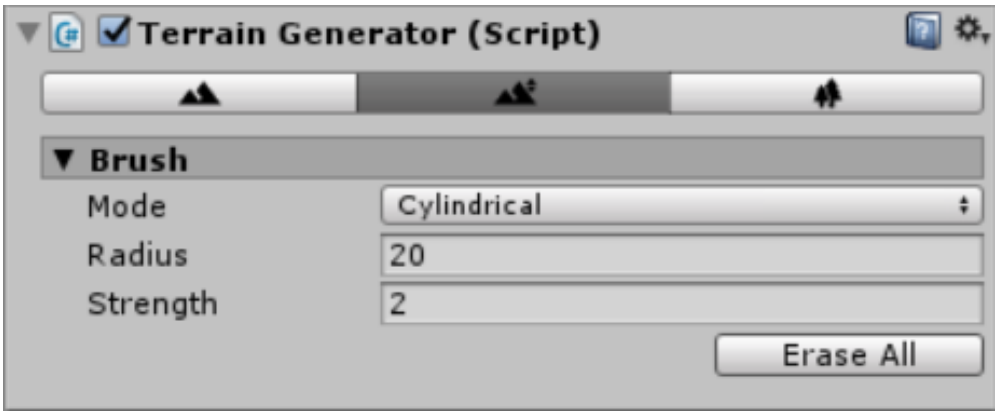


- Recalculate Bounds: Should it recalculate mesh bounds after generated? Turn this on to avoid culling issues.
- Recalculate Normals: Should it recalculate mesh normal vectors after generated? Turn this on if your shader use normal vectors.
- Recalcualte Tangents: Should it recalculate mesh tangent vectors after generated? Turn this on if your shader use tangent vectors.
- Unwrap UV: Should it generate a UV layout for the mesh after generated? Must be on for Texture Exporter to work correctly.

Geometry painter

Brush settings

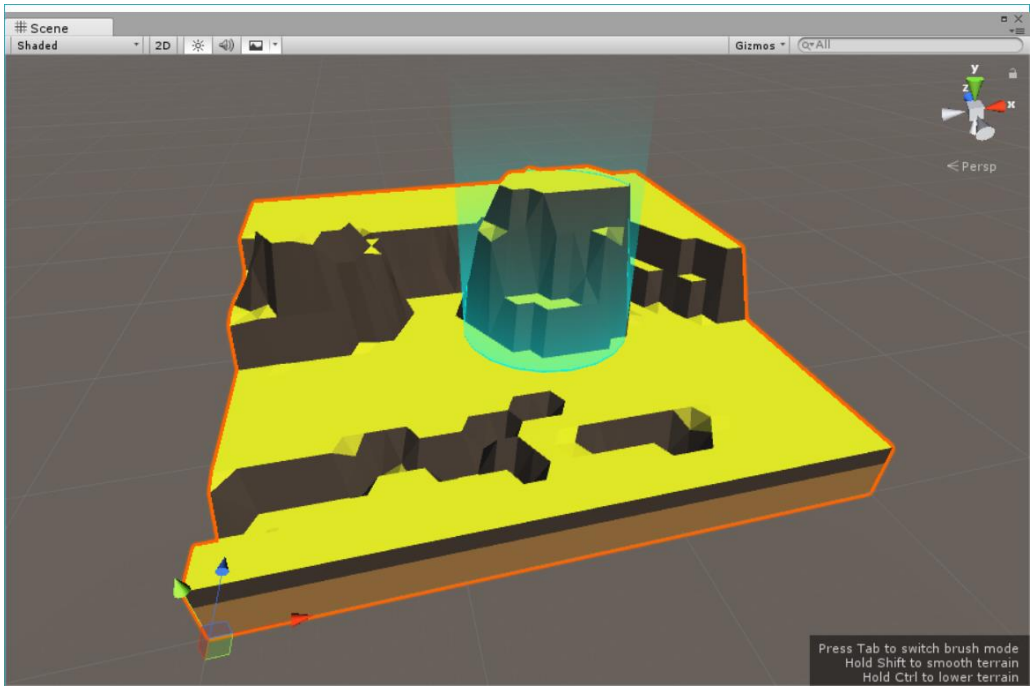
In the Inspector window, switch to the second tab, then open the Brush foldout, you will see the properties as listed below:



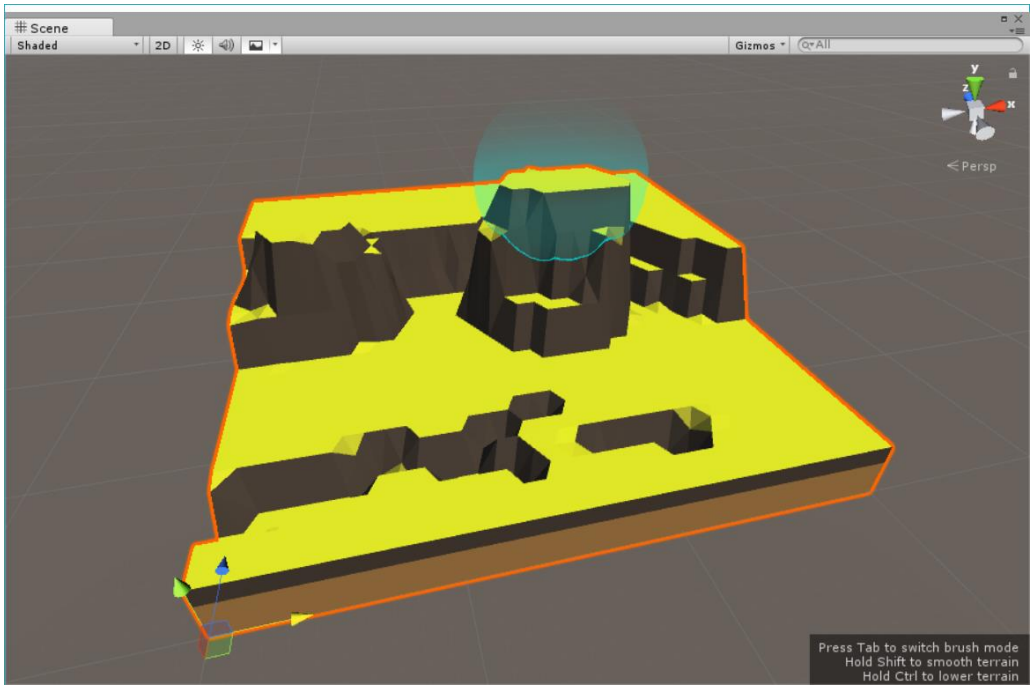
- Mode: Brush mode, determine which vertices are affected when painting (see below).
- Radius: Radius of the brush.
- Strength: How fast the terrain elevates when painting.
- Erase All: Click this button to flush painting data from the terrain.

Currently there are 2 brush mode supported in this version:

- Cylindrical: Use a cylinder with infinite height to determine which vertex to modify, that means the Y-coordinate of vertex position will be ignore when calculate distance to the painting point.



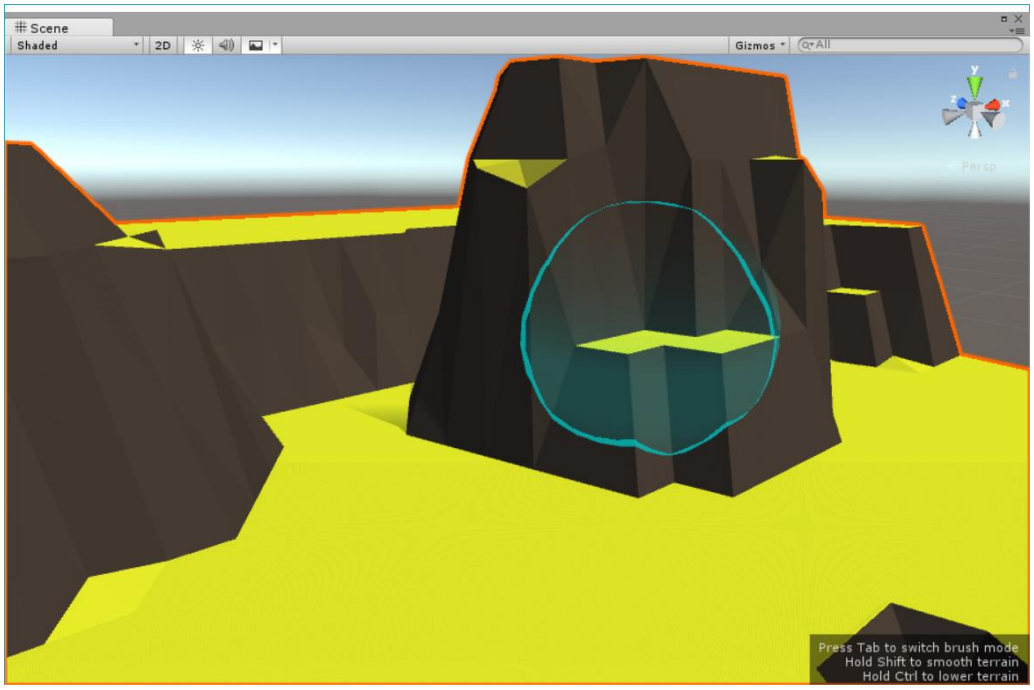
- Spherical: Use a sphere to determine which vertex to modify, any vertex inside the sphere will be affected on painting.



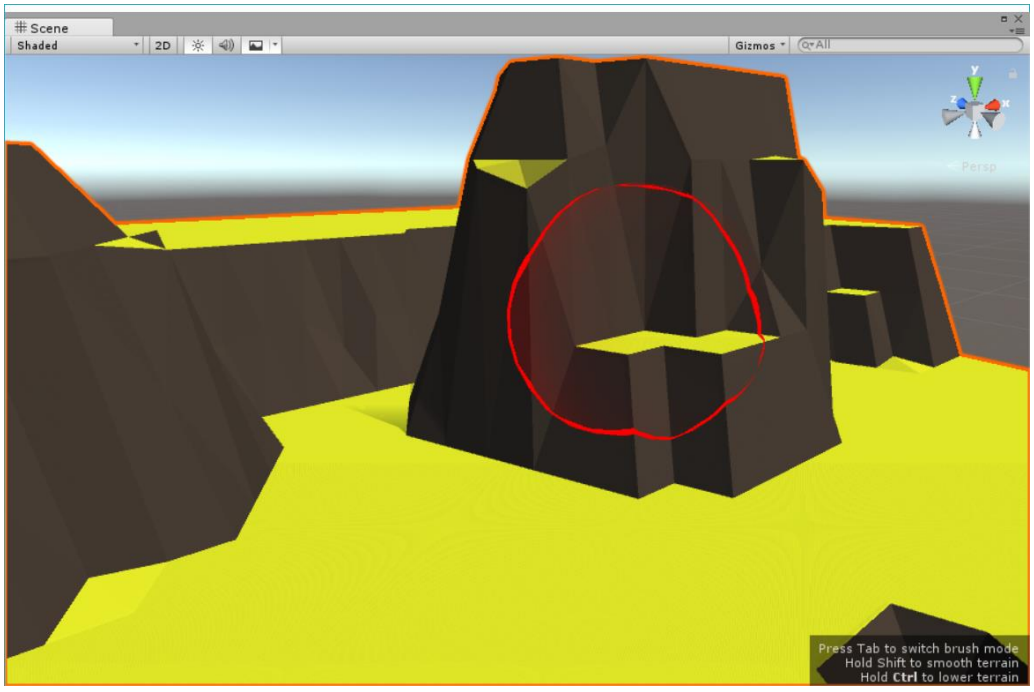
Edit terrain in Scene view

Hover your mouse over the terrain in the Scene view and you left mouse to paint on it. Currently, there are 3 paint mode supported in this version: Raise, Lower and Smooth.

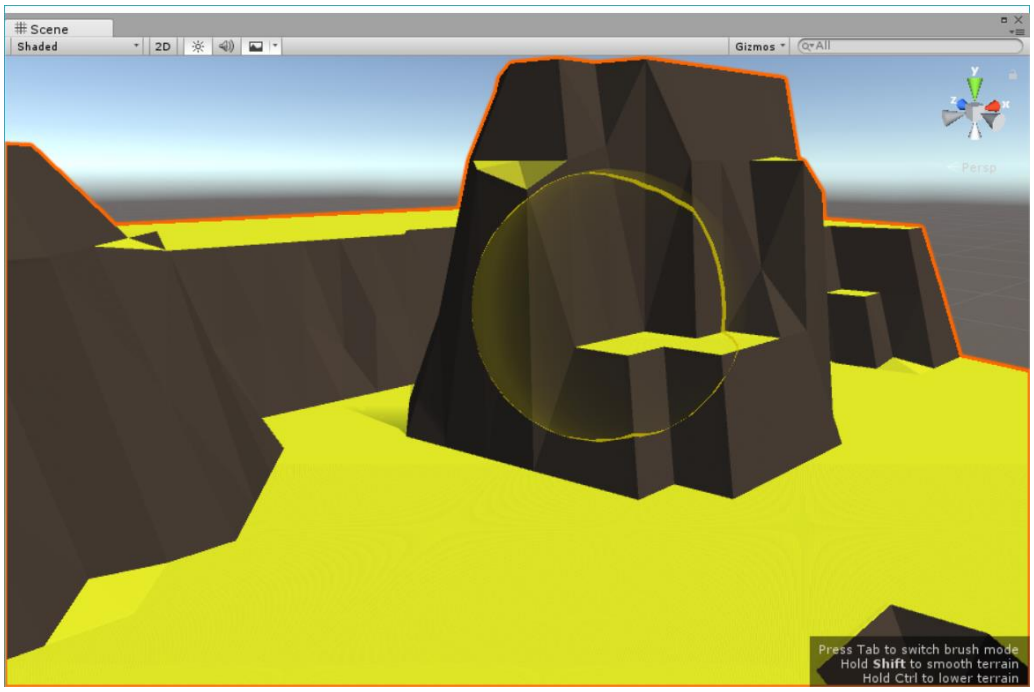
- Raise: This is the default mode, surface vertices will be move up and not exceed Surface Max Height property. In this mode, brush cursor will have cyan color.



- Lower: You have to hold Ctrl key to use this brush, surface vertices will be move down and not exceed Base Height property. In this mode, brush cursor will have red color.



- Smooth: You have to hold Shift to use this brush, this will apply a Laplacian smooth function to the vertices in range. In this mode, brush cursor will have yellow color.



Note: Vertex displacement using brush (hand painted method) is an additive and un-normalize operation. That means it is **not clamped** by the height map and painting data at a position will **not change** when you resize terrain dimensions.

Note: When painting, the terrain will display a quick preview version, which may have incorrect color, some case it disables flat shading due to large dimension. Don't worry, everything will be correct when you release the left mouse.

Keyboard shortcuts

Below are some keyboard shortcuts when working with Geometry painter in the Scene view:

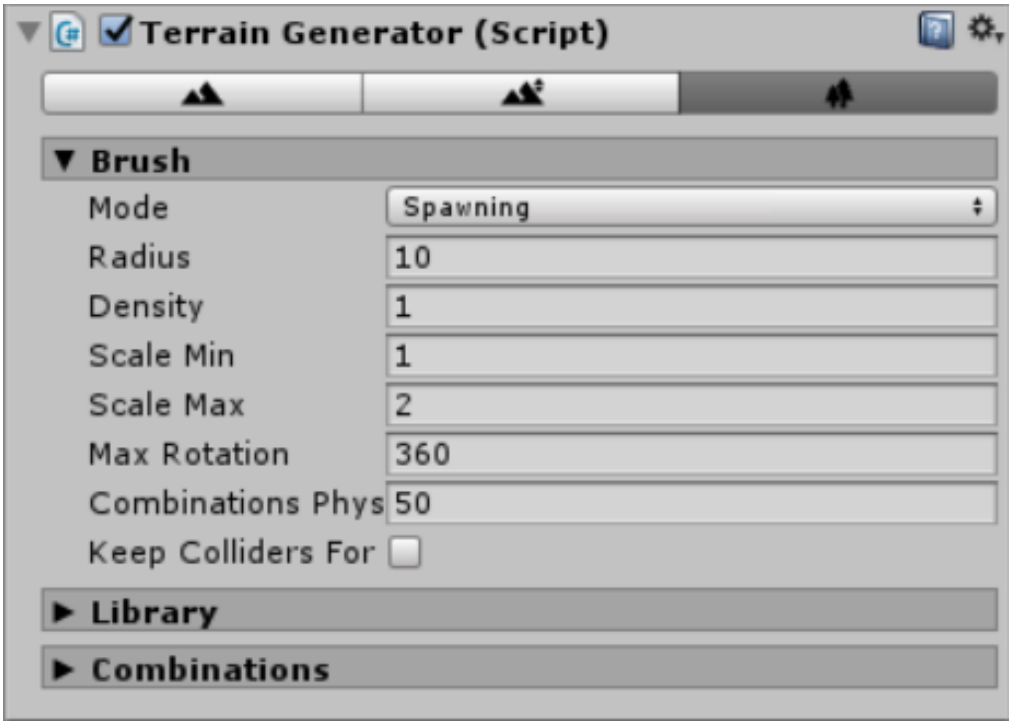
- Tab: Switch between brush shapes.
- Hold Shift: Use Smooth brush.
- Hold Ctrl: Use Lower brush.
- Plus (+): Increase brush radius.
- Minus (-): Decrease brush radius.
- Shift Plus: Increase brush strength.
- Shift Minus: Decrease brush strength.

Note: Scene view must have focus to use these shortcuts.

Environmental painter

Brush settings

In the Inspector window, switch to the third tab, then open the Brush foldout, you will see the properties as listed below:

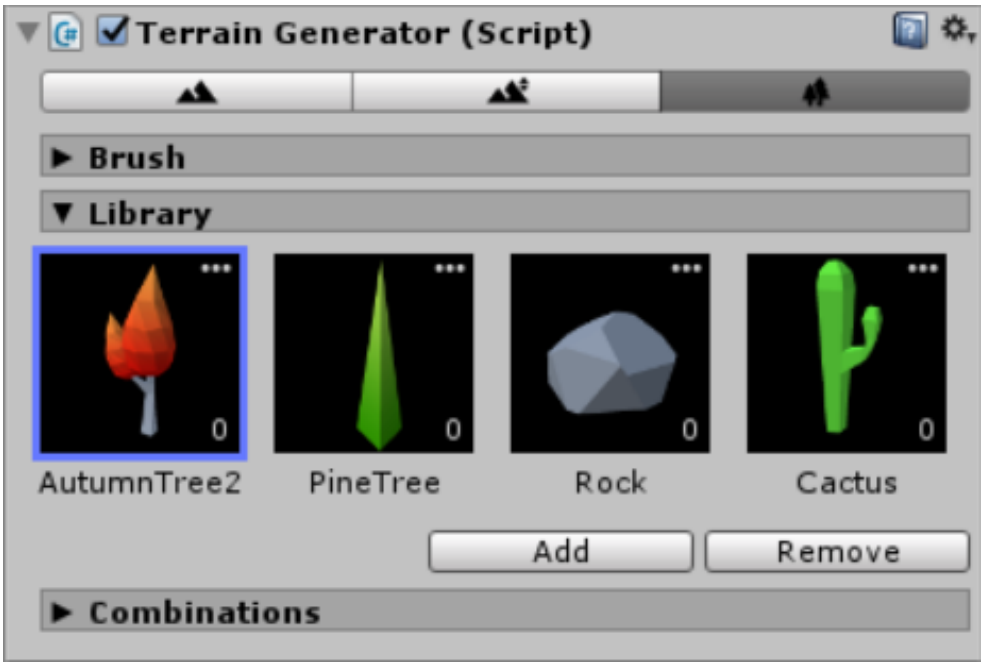


- Mode: Which mode to use, Spawning or Masking?
- Radius: Radius of the brush.
- Density: Determine how dense to spawn objects.
- Scale Min: Minimum scale of spawned objects.
- Scale Max: Maximum scale of spawned objects.
- Max Rotation: Maximum rotation of spawned objects.

- Combination Physical Size: size of an object combination. Larger value allows more objects to be grouped into one combination. Choose it wisely, too small value requires more draw calls to render the scene, while too large value loses the advantage of culling.
- Keep Colliders For Groups: If turned on, grouped objects will have colliders like its prefab, otherwise its colliders will be removed.

Library settings

In the Inspector window, switch to the third tab, open the Library foldout, this is where you create and manage your environmental objects collection:

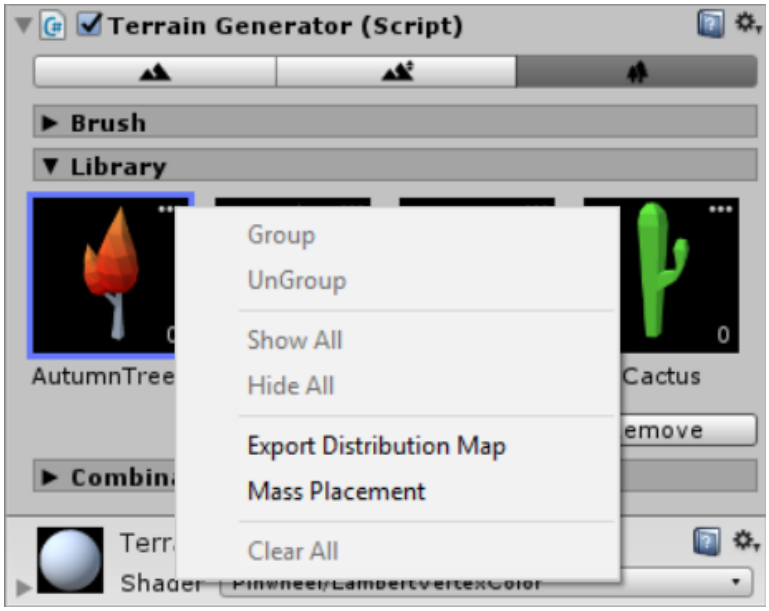


To add a prefab to the library, click on the Add button, then select the prefab you want to add. In some case, if the preview image cannot be loaded, an asset re-import is required. But don't worry, this process happens automatically and asynchronously, you have to do nothing!

To remove a prefab from the collection, select the prefab and hit Remove, then OK. Caution: this action cannot be undone at the moment, and all prefab instances, including the grouped one, will be flush out of the terrain.

Each prefab in the collection will be displayed as a tile, with its name shown below. The blue border indicates the currently selected prefab.

In each tile, the number at bottom-right corner indicate how many instances has been spawned to the scene. When click on the three-dot button at the top-right corner will show a context menu with additional actions:

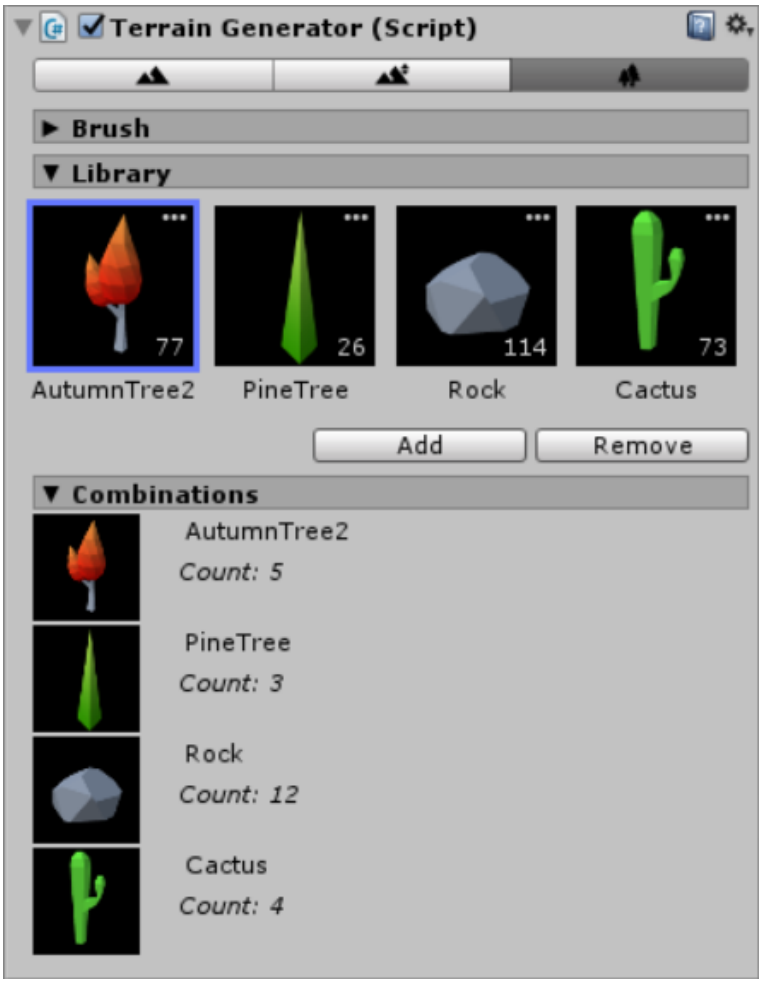


- Group: Group nearby instances into one larger object, size of the group depends on the Combination Physical Size property.
- UnGroup: Un-group all instances.
- Show All: Show all instances of a prefab.
- Hide All: Hide all instances of a prefab.

- Export Distribution Map: Create a grayscale map base on instances position.
- Mass Placement: Randomly spawn object using the distribution map and other parameters.
- Clear All: Destroy all prefab instances as well as its groups.

Combinations settings

In the Inspector window, switch to the third tab, open the Combinations foldout, this is where you manage your environmental objects groups:



Each prefab will have an entry if its instances have been grouped. The Count property indicate how many group created from its instances, this number depends on the value of Combination Physical Size property.

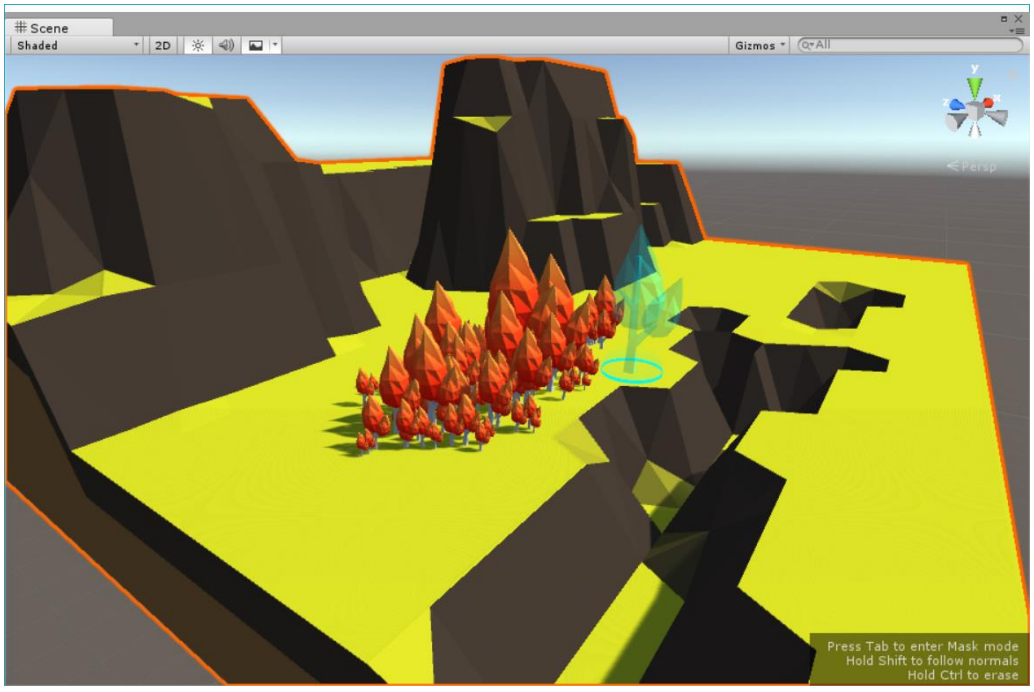
For example, in the image above, 77 instances of the AutumnTree2 prefab have been grouped into 5 larger objects.

Edit environment in the Scene view

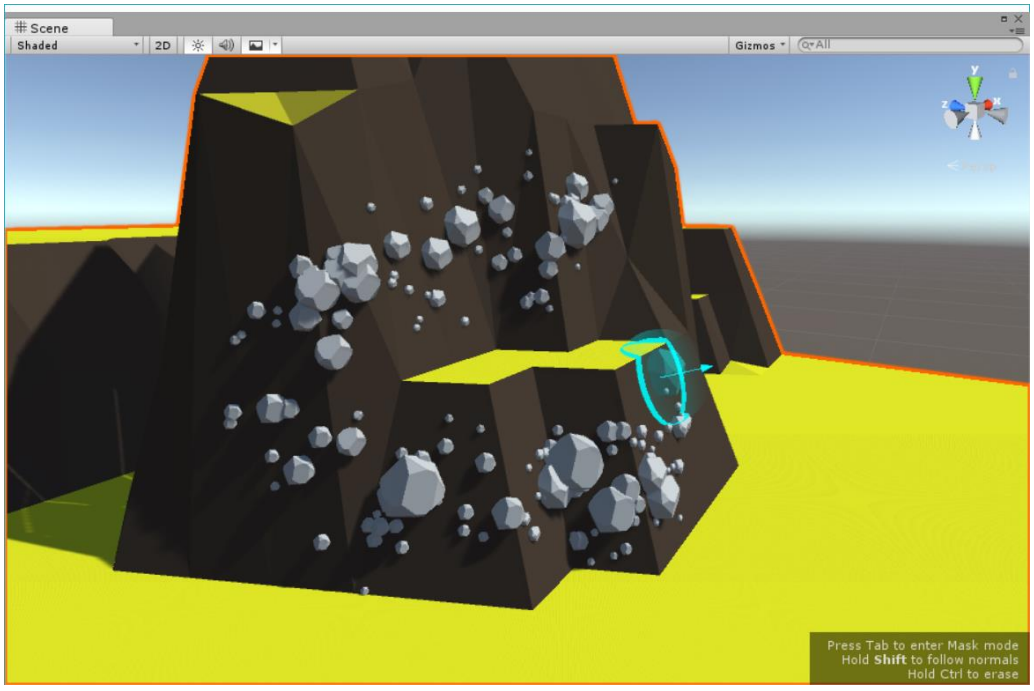
To spawn object into the terrain, you first have to add some prefabs into the Library. If you don't know how to do it, please see this page first.

Select the prefab you want to spawn, then hover the mouse over the terrain and use left mouse to paint. Currently, there are 2 paint mode in this version: Spawn and Remove.

- Spawn: This is the default mode, simply hold the left mouse to spawn objects. When you paint over the previously created instance, it's likely scale that instance up instead of spawning a new one. This mechanic gives the environment a more natural look since trees in the middle of a jungle often grow higher than the one on the edge. In some case, you want the spawned object to follow the surface normal vector, for example: rock on a mountain cliff, simply hold Shift and paint. In this mode, brush color will have cyan color.

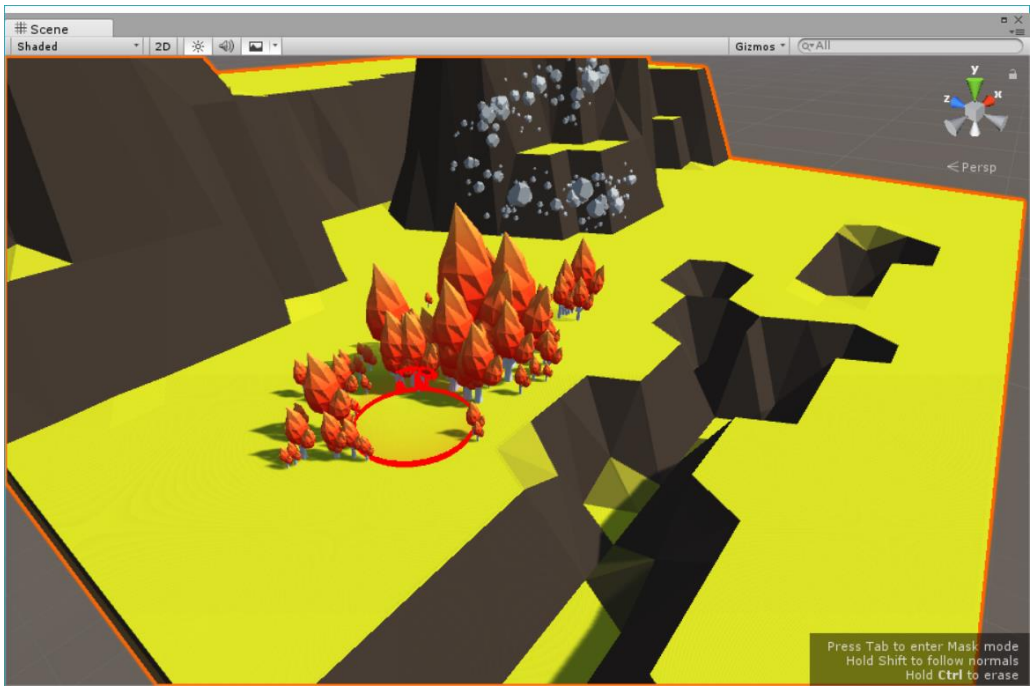


Default Spawn brush, noticing the tree in the middle are higher



Spawn brush with Follow normal option, the small arrow indicates surface normal vector

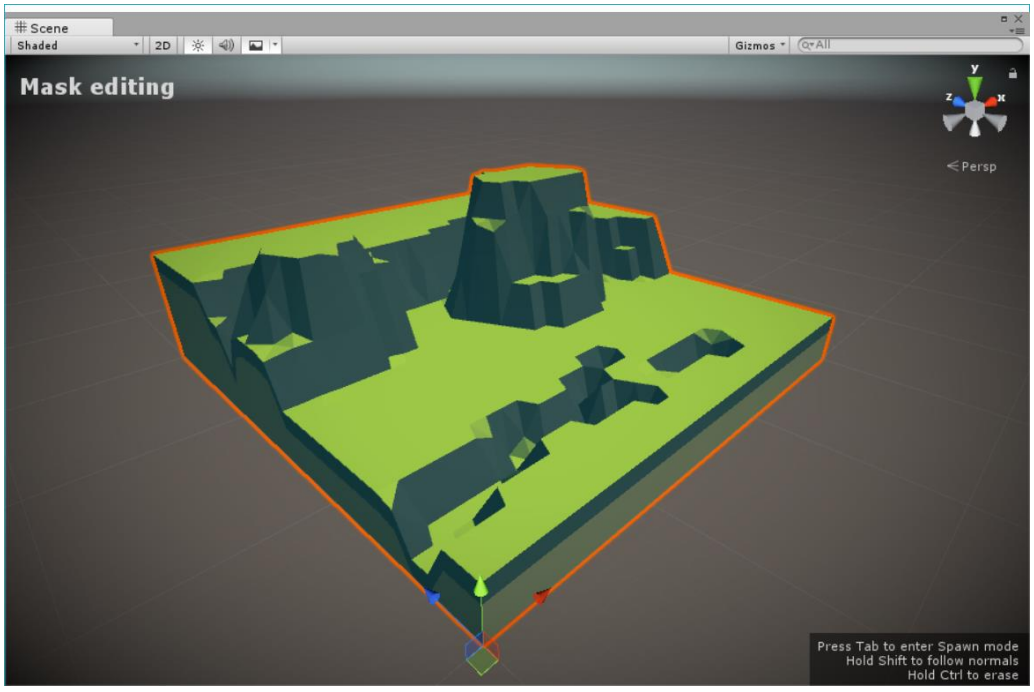
- Erase: Hold Ctrl and use left mouse to erase prefab instances. Note that it only erase instances of the currently selected prefab, and that instance must not be in any object group. In this mode, brush cursor will have red color.



Mask editing

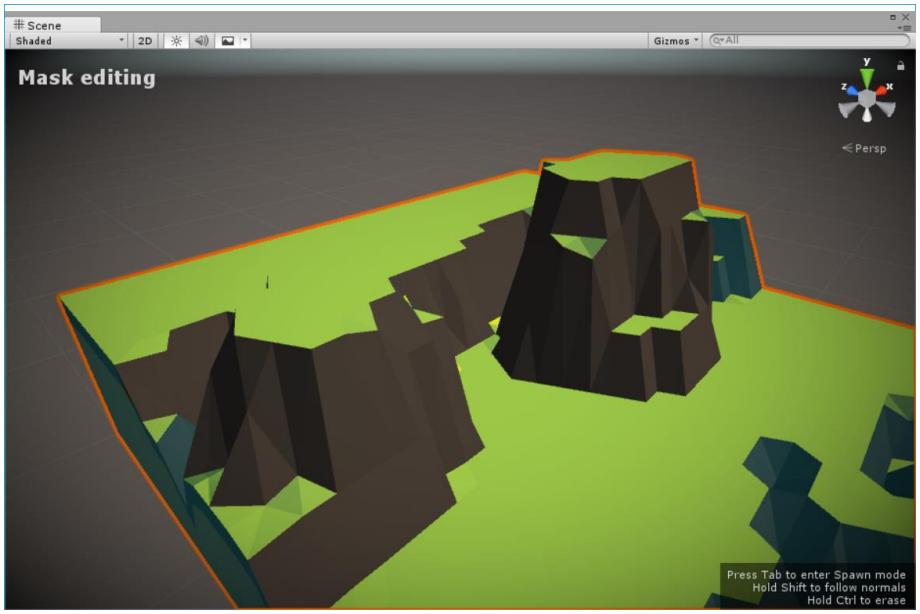
In some case, you want to restrict the spawning region to specific location on the terrain, say, trees are not allowed to grow on a mountain cliff, Mask can be used to serve for this purpose.

In the Inspector window, switch to the third tab, then open the Brush foldout, set the Mode property to Masking, you can see the Scene view switch to Mask editing mode.

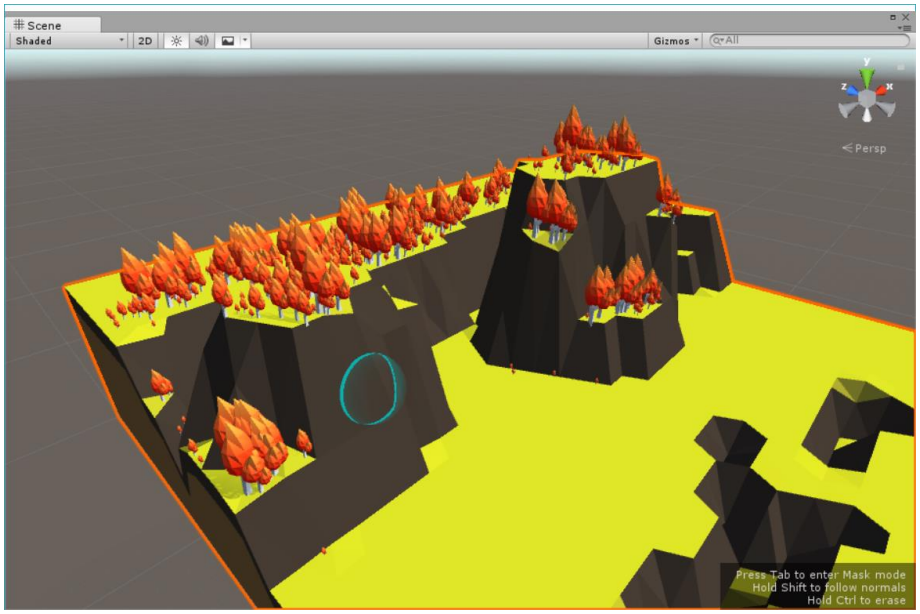


The cyan tinted region is where you can spawn objects, and the region with true color is where you can't. Similar to Spawning mode, use left mouse to paint, and hold Ctrl with left mouse to erase.

Below are examples for the effect of Masking:



Mask on the cliffs has been erased



Trees spawned with mask, noting the cursor doesn't display preview when hovering over restricted area

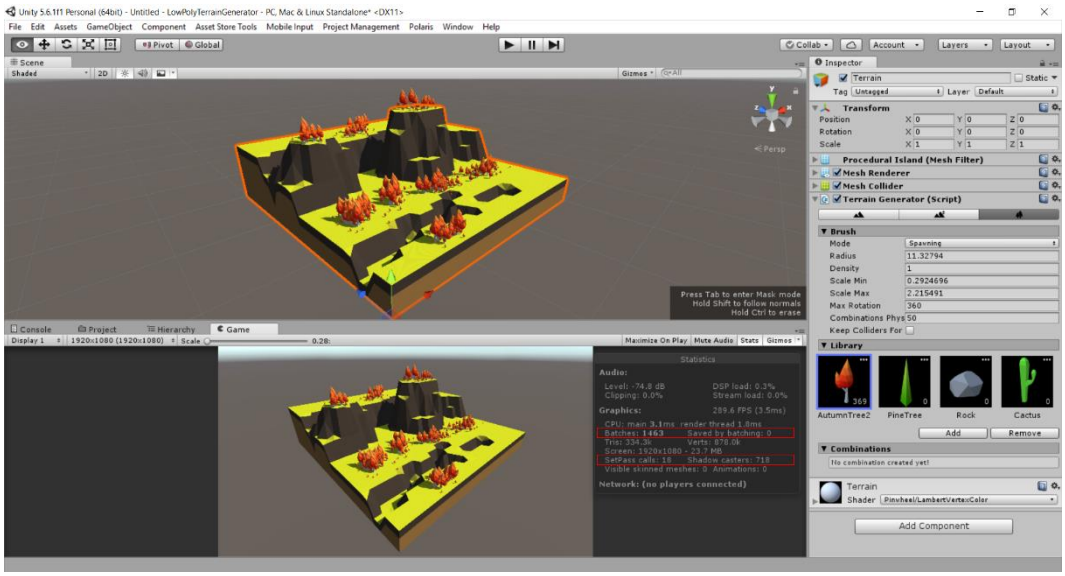
Note: If you make some specific changed to the terrain, like resizing, the mask will be reset to its initial state.

Optimize the environment

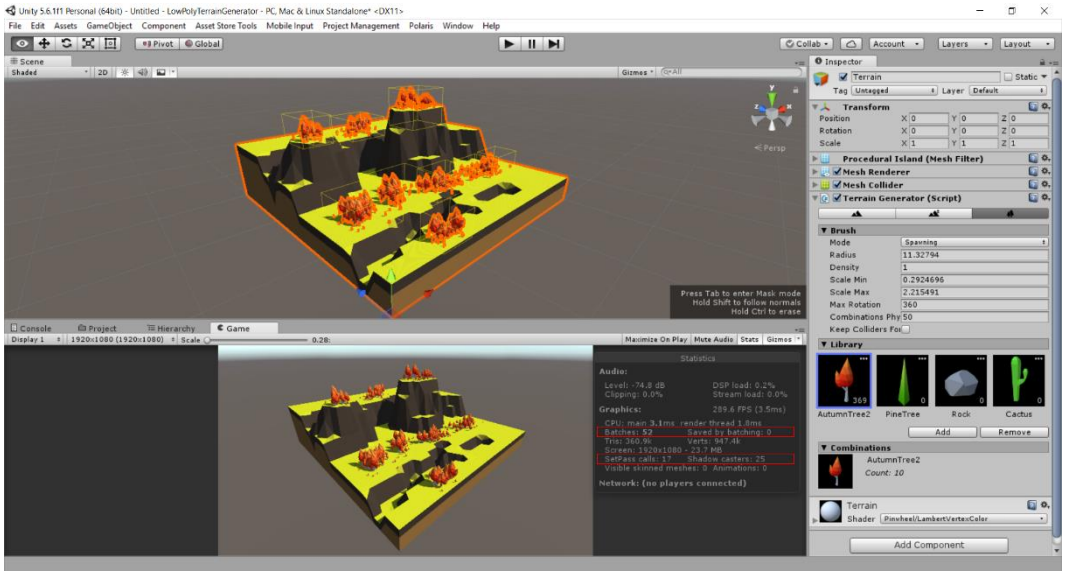
When it come to a large and complex environment, performance may be affected if there's so many objects spawned to the terrain. There are many methods can be used to optimize the scene, one of them are using the Group features.

Basically, the more objects in the scene, the more draw calls it need to render, the situation even worst if each of them can cast and receive shadow. So we need to combine objects of the same type into a larger object and render it once, to reduce rendering resources.

After spawning objects into the scene, click on the three-dot button at the top-right corner of the prefab tile in the Inspector window, then select Group. Done!



Scene before Grouping



Scene after Grouping, note the change of scene stats

You should try different Combination Physical Size values to maximize performance. Small value produces more group, thus more batches, draw calls and

shadow casters, while large value produce less group, but loose the advantages of culling, since the bounds of that group is too large and being considered visible all the time!

Other technique use can use to optimize the scene:

- Reduce object detail, especially number of vertices
- Disable shadow if it's not important.
- When modelling grass, make a bush instead of a single leaf.
- Remove colliders and other component if they're not important.

Keyboard shortcuts

Below are the keyboard shortcuts when using the Environmental painter in the Scene view:

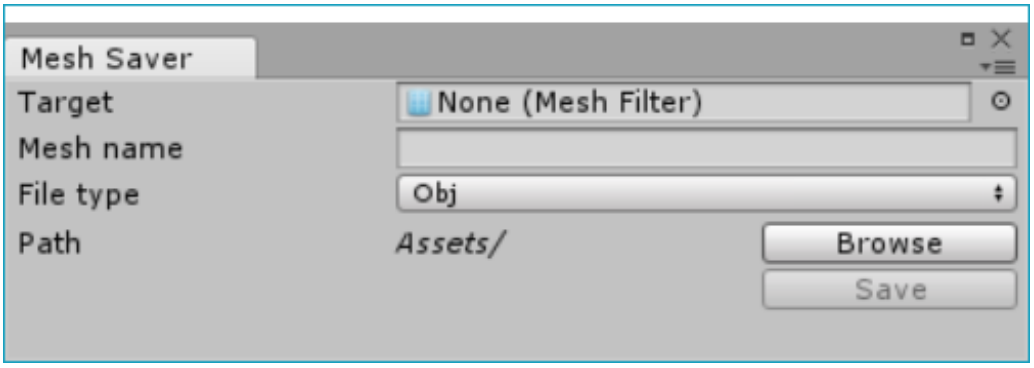
- Tab: Switch between Spawning and Masking mode.
- Hold Shift: Enable follow surface normal spawning.
- Hold Ctrl: Enable erasing brush.
- Plus (+): Increase brush size.
- Minus (-): Decrease brush size.
- Shift Plus: Increase object scale.
- Shift Minus: Decrease object scale.

Note: Scene view must have focus to use these shortcuts

Extract data from terrain

Save procedural mesh

There is a built-in tool to help you save meshes into 3D file. Select Polaris/Mesh Saver menu, you will see a window appears as below:



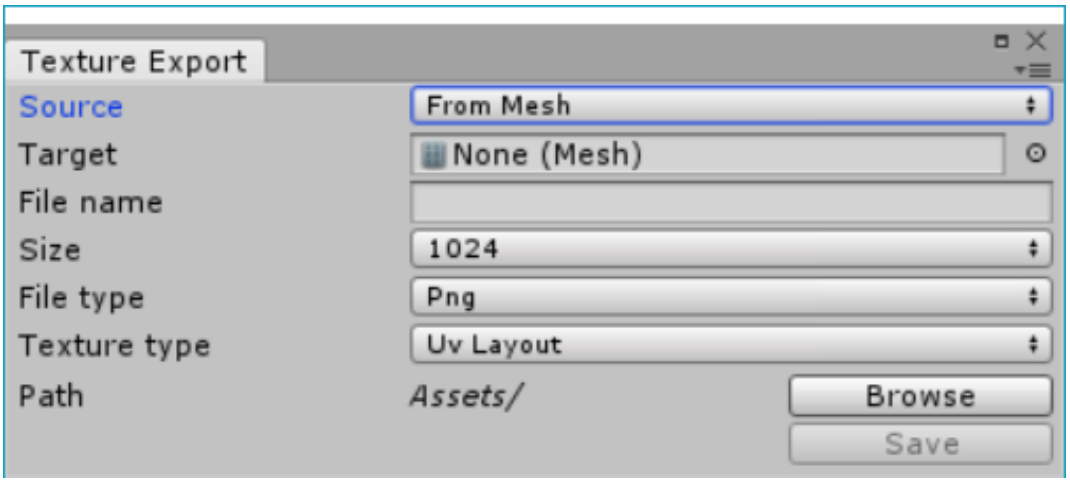
Fill in appropriate parameter, then hit save, a new 3D file will appear in selected directory.

Currently support file types: Obj, Fbx ASCII

More file types will be added in future updates.

Export maps from general mesh

You can export several kind of map from mesh within Unity using the Texture Exporter. Select the Polaris/Texture Exporter menu, you will see a window appears as below:

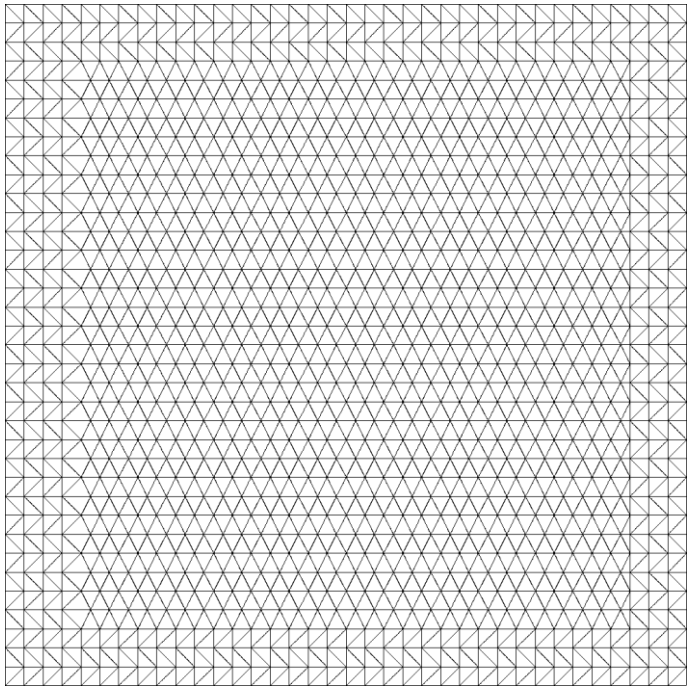


Fill in appropriate parameters, then hit Save, you will see the map appear in the selected directory.

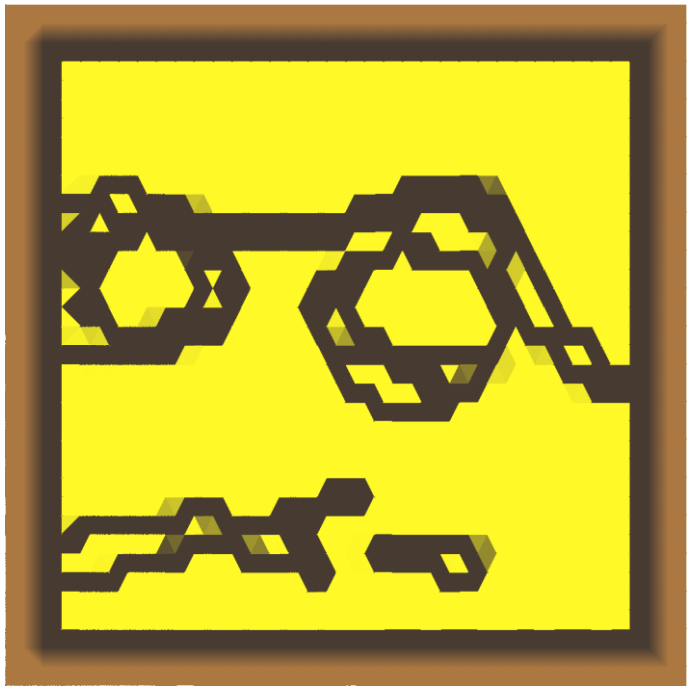
This tool can export map from a Mesh or a Mesh Filter.

Currently support texture types: Uv Layout, Vertex color.

More texture type will be added in future updates.



Uv layout exported from a terrain



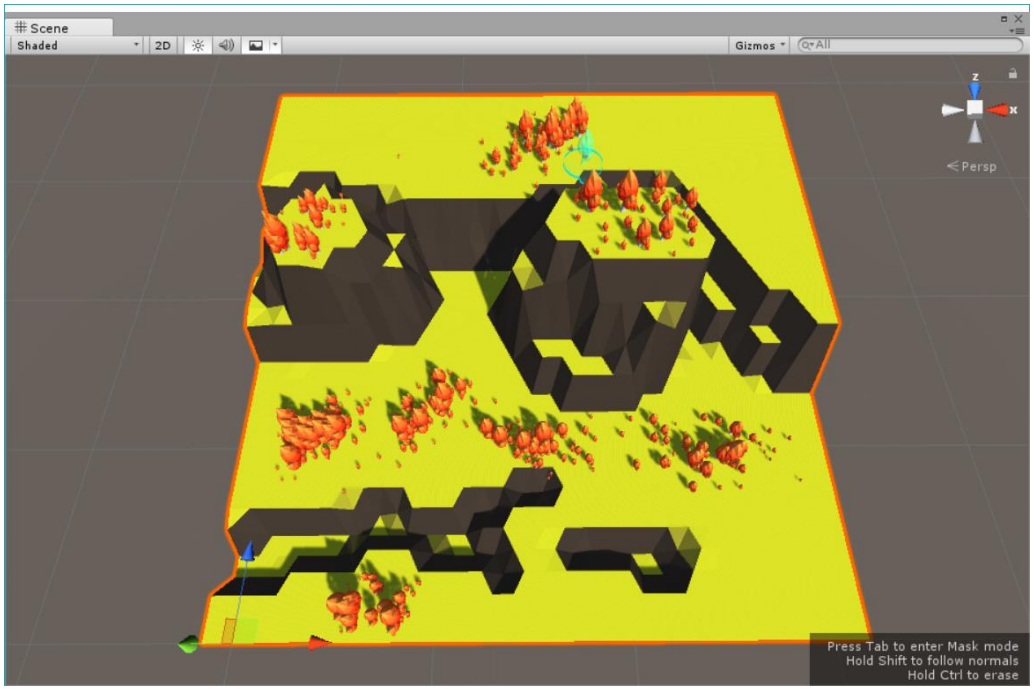
Vertex color map exported from a terrain

Export terrain specific maps

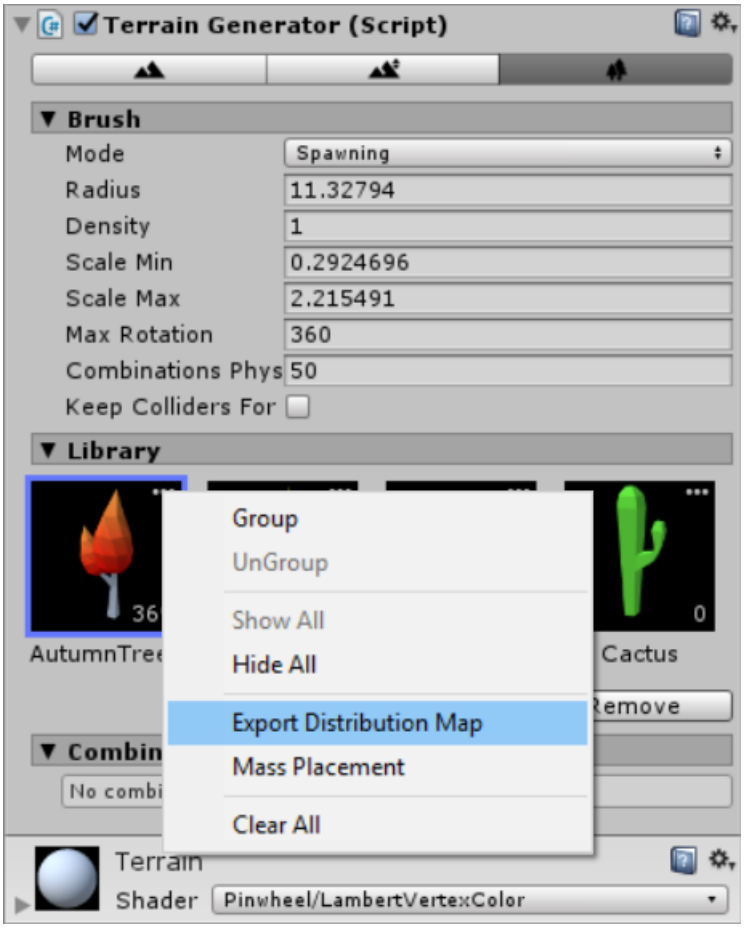
Some data map is specific to the terrain, and they are placed in different UI location, see the sub sections for more information.

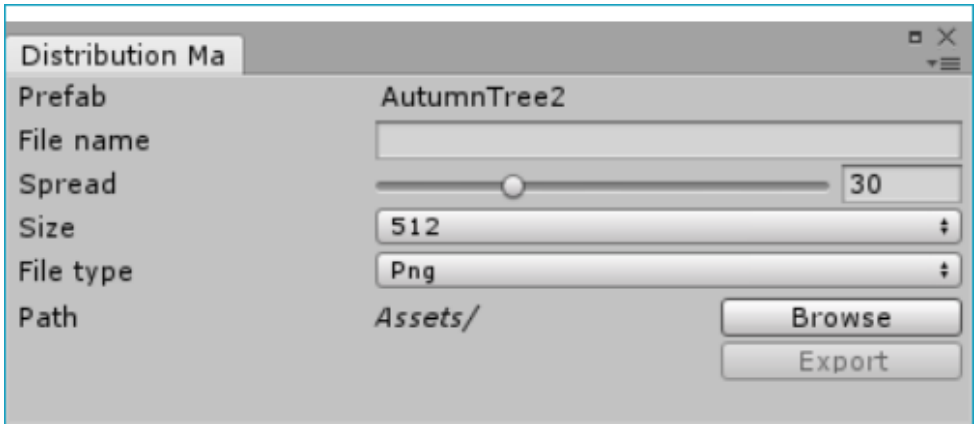
Distribution map

Distribution map is used as a parameter for Mass Placement feature. To export this map, open the Environmental Painter tab (the third one), select a prefab and paint some object directly onto the terrain as samples.

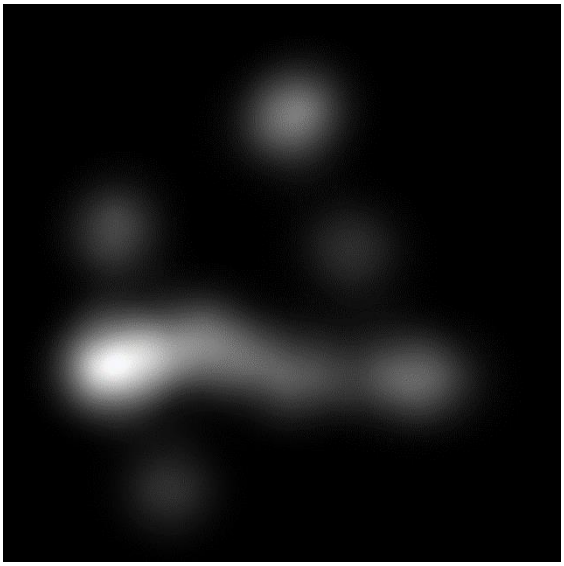


Then click on the three-dot icon on the prefab tile in the Inspector, select Export Distribution Map, a window will appears as follow:





Fill in appropriate parameters, then hit Export, you map will appeared in the selected directory.



APPENDIX

Anatomy of a Mesh

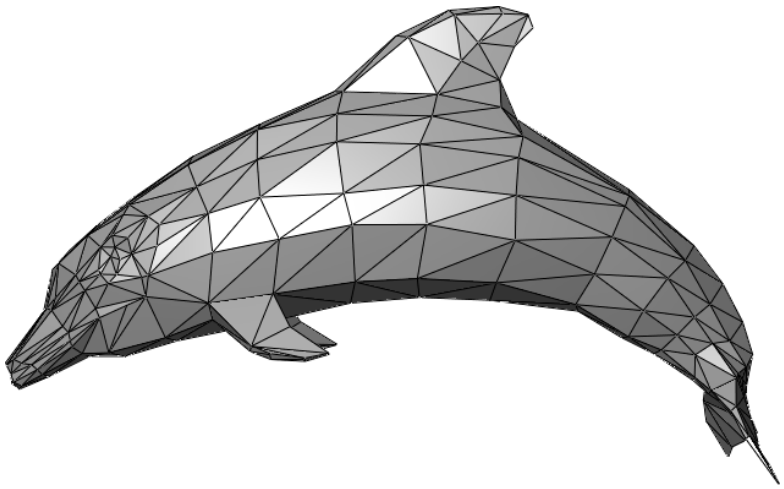
Since the terrain generated by this engine is actually a mesh, we should understand the basic of its anatomy to create better production.

What is Mesh

Basically, mesh is a collection of vertices, edges and faces represent the shape of an object in 3D space, which is heavily used in both real life and computer graphics.

For a real world example, imagine that you have a building, then the mesh would be its steel frame structure.

In computer graphics, you may have seen a lot of 3D model used in gaming project or in film. They are all constructed from mesh:



There are 2 type of mesh: Polygonal and Volumetric. Polygonal meshes only represent the object surface, such as simple geometry like triangle, rectangle or a plane, while Volumetric meshes represent both the surface and the volume of the object.

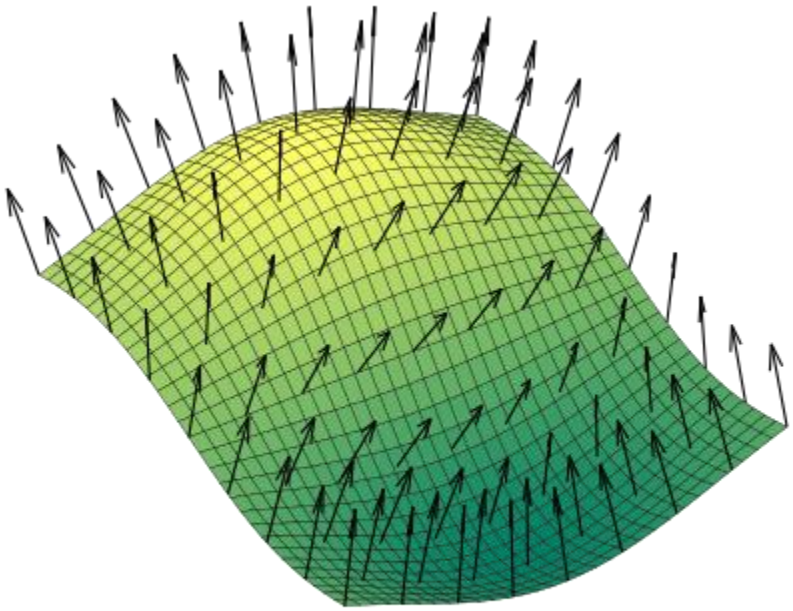
Vertices

A vertex, simply a point in 3D space, is the most basic element of the mesh anatomy. The more vertices, the more power it took to render the object on to the screen. In Unity, these vertices position are relative to their parent game object, e.g. in local space, and number of vertices in a single mesh cannot exceed 65000.

Beside of position data, a vertex also contains some other information like normal vectors, tangent vectors, color and uv-coordinate, they are used for specific purposes.

Normal is a vector which is perpendicular to a given object in space, in this case, the vertex. Actually, vertex is just a point, so the normal vector should be perpendicular to the surface contains the vertex at its position. The normal vectors give us an impression of where the surface faces to, which is useful to determine

how it react to the light. For more information about normal vectors, please see here.



Tangent is a vector which is "just touch" a curve or a surface at a given point, in this context, at a vertex in the mesh. Tangent vectors are often used in pair with normal vectors (bi-normal vectors as well) and very useful in some case such as normal mapping. For more information about tangent vectors, please see here.

Each vertex in a mesh can optionally store a RGBA color value, this is called vertex color. This piece of information can be used for many purposes. For example, you can use vertex colors to paint a mesh without bothering with texture and uv-mapping. Essentially, when the computer renders a triangle of the mesh using vertex colors, it will try to interpolate colors of each vertex of the triangle on every pixel across the surface. That's why this method often achieve a smoother look,

with less noise than using a texture with solid colors due to texture compression and bad uv-mapping. For that reason, we will use this technique to color our terrain. Beside of painting the mesh, vertex colors are also used for custom effect like ambient occlusion, wind effect for foliage, etc.

Vertex coloring is nice, but when it comes to a more complex model with lot of details, it's not the good choice. This is when texturing and uv-mapping join the game. Each vertex in a mesh can hold a piece of information call uv-coordinate which is used for projecting, or flattening a 3D mesh onto a 2D image. The U and V letter denote the horizontal and vertical axes of the image, to avoid ambiguity with XYZ in object position. This technique is used to bake a large amount of vertex information like color, direction and custom data into a single image, for ease of editing. For more information about uv-coordinate as well as uv-mapping, please see here.

Unity engine doesn't have a specific data type to store vertex information. Instead of creating one, they just store them in several arrays:

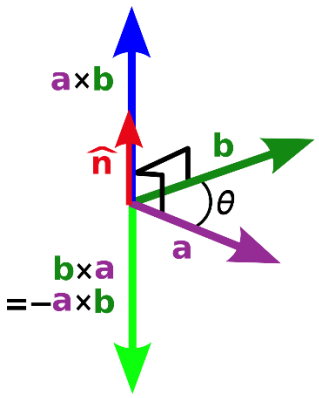
- Vertex positions are stored in a Vector3 array: vertices.
- Normal vectors are stored in a Vector3 array: normals.
- Tangent vectors are stored in a Vector4 array: tangents.
- Vertex colors are stored in a Color array: colors.
- Uv-coordinates are stored in a Vector2 array: uv.

When making changes to these arrays, we need to copy them into a temporary array then do the work on it, after that, reassign it to the mesh, because there are a lot of behind the scene stuffs happen when accessing these arrays.

Edges

An edge is simply a line connects two vertices together. Using the two vertices, you can easily find the edge's formula, as well as its direction, and other properties. Since the edges don't contain special data and they're used not so often, Unity doesn't store it anywhere, but you can refer to them indirectly via the vertices and triangles arrays.

We almost don't care about edges during the mesh generation process, but it doesn't mean that they are useless. One example for their application is to manually calculate the normal vector of a specific face, by using cross product of the two edge's direction.

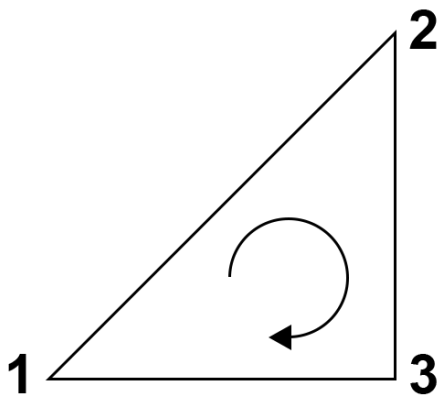


Faces

Face is a closed set of coplanar edges. A single face often contains three or four edges, but in some case it can be more, depend on the render engine you use. A face with three edges is called a triangle, and the one with four edges is called a quad. In Unity, most of faces are triangles.

Face is the element makes the mesh visible. In Unity, the process of rendering a face is broken down to 2 steps: Render front face and back face. So

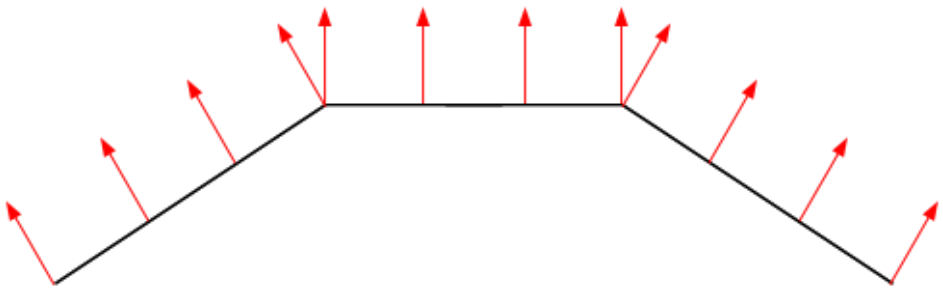
how can we determine which side is front face and which side is back? To form a triangle, you have to pick 3 vertices into a group, the order matter! When looking down to the mesh from the camera, front face is the side which its vertices order goes clockwise, and back face is the side which its vertices order goes counter-clockwise. However, it doesn't matter which vertex you start with.



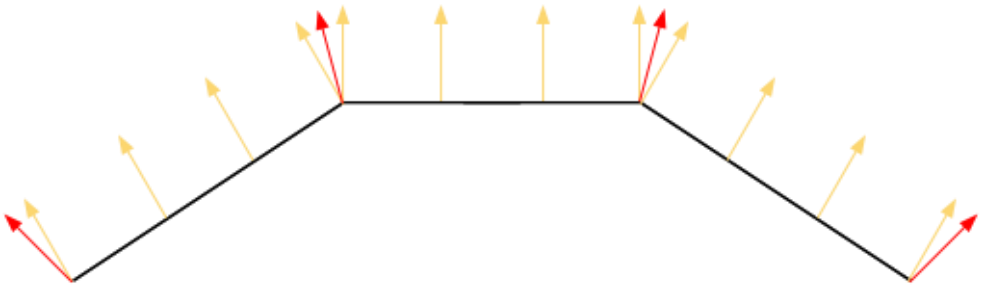
In Unity, triangles are simply stored as an integer array, where each element is an index to the vertices array. Each triangle will take 3 indices; thus the array length will be a triple of number of vertices.

Vertex data interpolation

As you have known, mesh data like normal vectors, colors, uv-coordinate, etc. are stored per-vertex (e.g. corresponding to each vertex), but a single vertex is shared between multiple faces, so the data should be interpolated based on these faces. The clearest example is normal vectors blending. Below is a 2D slice of a mesh surface, where the arrow is the direction of normal vectors on each face:



But since the normal vectors are stored per-vertex, the actual vector should be blended between the directions of the surrounding faces, like this:



This way, we can create an impression of a smooth surface even if the true nature of the mesh is flat and hard edged. For more information about vertex data interpolation, please see here.

Rules to follow when assigning mesh data

When generating meshes from code, there are some strict rules to remember:

- Don't modify its data arrays directly, instead copy them to a temporary one, do the work, then re-assign it back to the mesh.
- The vertices array must be assigned first, other arrays later. Once again, the number of vertices cannot exceed 65000.

- Number of indices in the triangles array must equal to a triple of number of vertices.
- Number of elements in the normals, tangents, colors and uv array must be equal to number of vertices, if assigned.