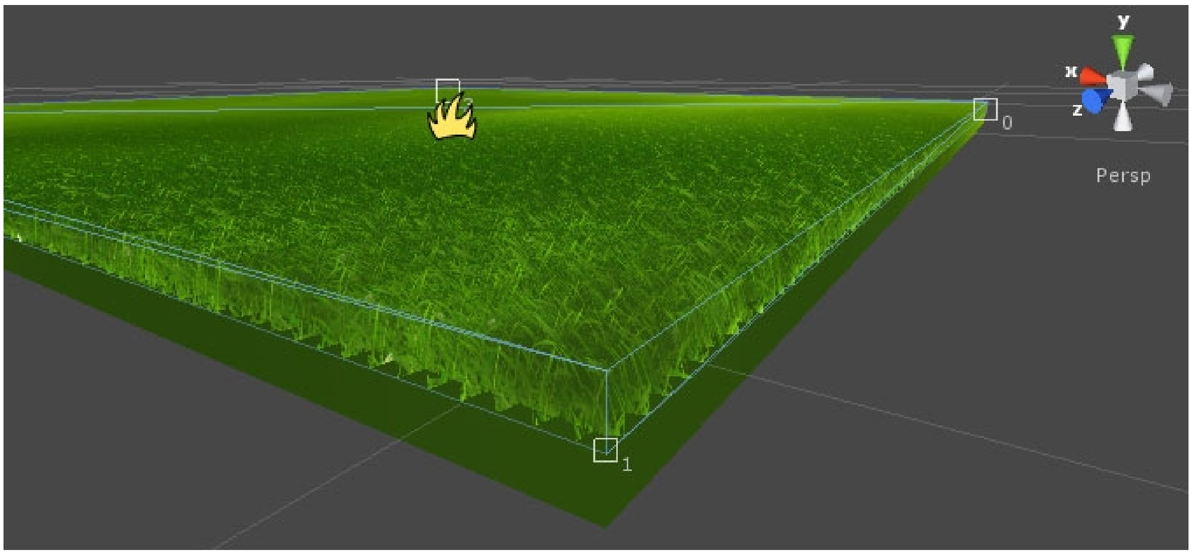
VolumeGrass (v. 1.1)

documentation & workflow

Overview / requirements

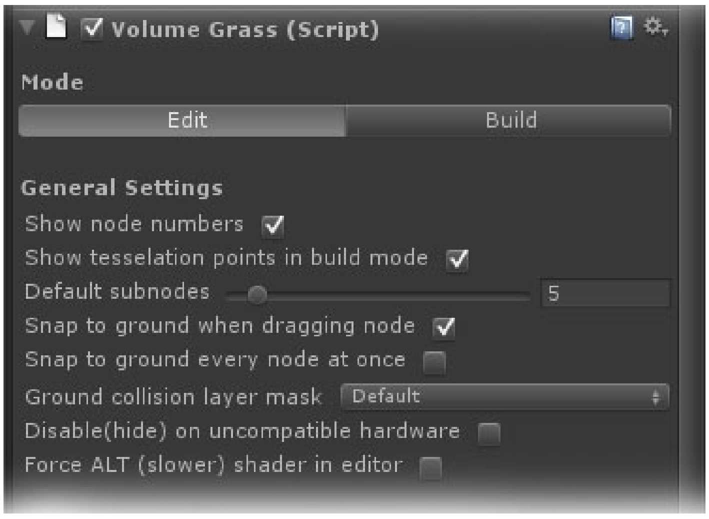
VolumeGrass is basically an editor extension which helps in quick creation good looking grass. It consists of editor script that's responsible for building proper mesh and setting up necessary parameters for shader. The heart of grass system is special ray-tracing shader that gives an illusion of grass being rendered “inside” the mesh volume:



The mesh above consists of only a few polygons, while the shader makes the whole job. Generally it's quite possible to use it with any mesh, but dedicated mesh editor makes it very easy to build grass area of any shape. While grass mesh may be very simple, the shader itself is not... It's GPU intensive and works efficiently on hardware of mid-high range only. The reasonable usage threshold lays somewhere at Radeon HD 4650 / GeForce8600 level. Slower GPUs may not be able to handle such complicated shader at sensible framerates and resolutions. The shader is, however, highly configurable, so user can make it work on slower machines at cost of quality. The grass shader requires at last **SM3.0** GPU and Unity3 (surface shaders). Has been tested on Unity3.1 / 3.2. Can theoretically work on Unity Indie but without any z-testing functionality described below as it requires render textures.

Building the grass mesh

To start work place an empty object in scene. Go into _Scripts folder in your grass package and attach script named *VolumeGrass* to the empty object. You should see such set of properties in the inspector:



Show node numbers ✓
Show tesselation points in build mode ✓
Default subnodes 5
Snap to ground when dragging node ✓
Snap to ground every node at once
Ground collision layer mask Default
Disable(hide) on uncompatible hardware
Force ALT (slower) shader in editor

Fig.1

By default we're in ***Edit*** mode which allows us adjusting all aspects of desired mesh. At this moment the only option we should care about is ***Ground collision layer mask***. It's set to *Default* layer. That's the layer the grass will be “sticked to” while editing. You could for example first put terrain into your scene and then place grass on it. It's important to remember that object to which grass will be adjusted should share the same layer as ***Ground collision layer mask*** so mesh could be placed on desired object. The object should also have collider component. If grass mesh editor can't find ground under the grass mesh it snaps everything to plane at y=0.

Now we're ready to build grass mesh. Choose Hand Tool from Transform Tools palette to disable multiselection when adding nodes. In scene view place grass mesh nodes by clicking while holding shift. After adding a few nodes you can build simple rectangle shape like this:

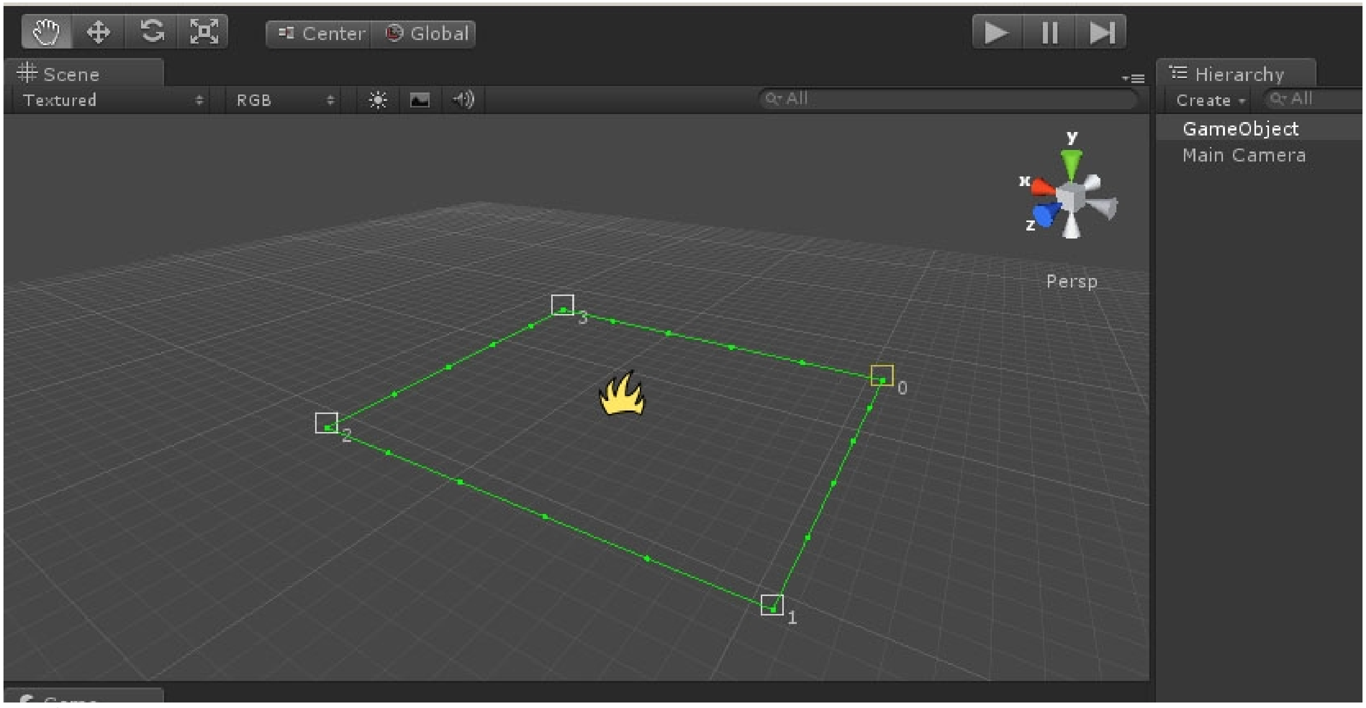


Fig.2

Numbered boxes mean nodes, small green points on the borders mean subnodes. We don't have any underlying geometry (terrain) so nodes were simply placed on y=0 plane. By default we've got 5 subnodes between each nodes pair. We can change it by adjusting ***Default subnodes*** slider in general settings (fig.1). Subsequent param ***Snap to ground when dragging node*** means that when you drag a node it will be snapped to the ground (its vertical / y position will be adjusted). ***Snap to ground every node at once*** means that all nodes will be snapped when you drag the grass object transform. To add node between two other nodes **shift+click** on green border. ***Show node numbers*** toggle is self-explanatory - allows hiding node numbers in scene view.

If you select node additional parameters appear on the bottom of VolumeGrass inspector:

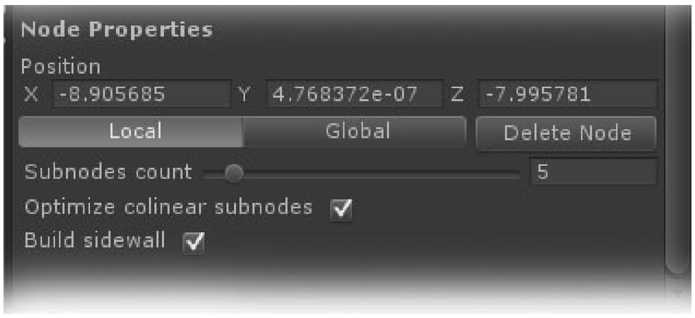
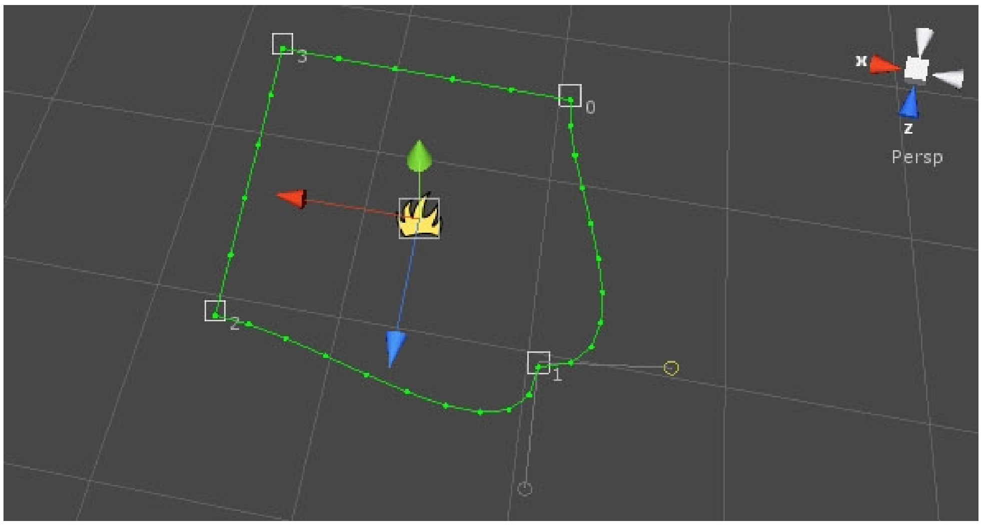

Node Properties
Position
X -8.905685 Y 4.768372e-07 Z -7.995781
Local | Global | Delete Node
Subnodes count 5
Optimize colinear subnodes ✓
Build sidewall ✓

Fig.3

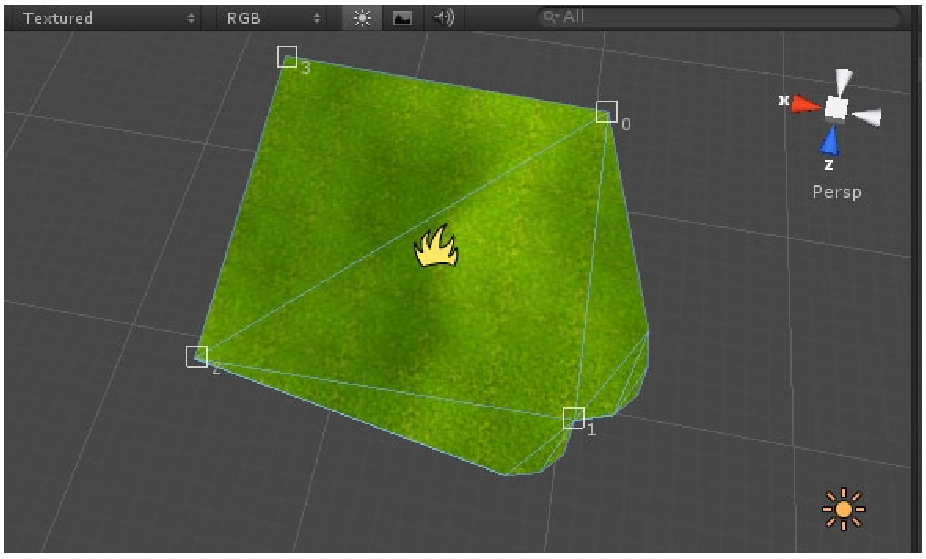
Here you can adjust node position numerically in local/global space. You can also adjust subnodes count so there will be different than default number of subnodes on border linking selected node N and next node (N+1).

One may ask – what are these subnodes for ? Answer is simple. They outline bezier curve that can be modeled between nodes rather than simple straight segment. **Alt+click** on node to add bezier handle. You can now model shape of grass area better:



Note that for nodes 0 and 1 **Subnodes count** (fig.3) has been set to 10 which gives us smoother shape.

We are ready to build the grass. Click **Build** button in inspector toolbar (top of inspector – fig.1). What you see is our grass viewed from the top:



It has a few polygons as can be seen in inspector:

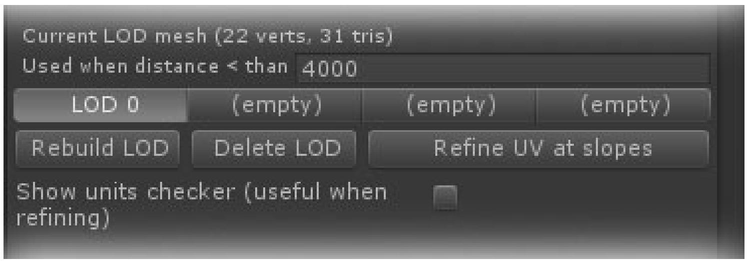
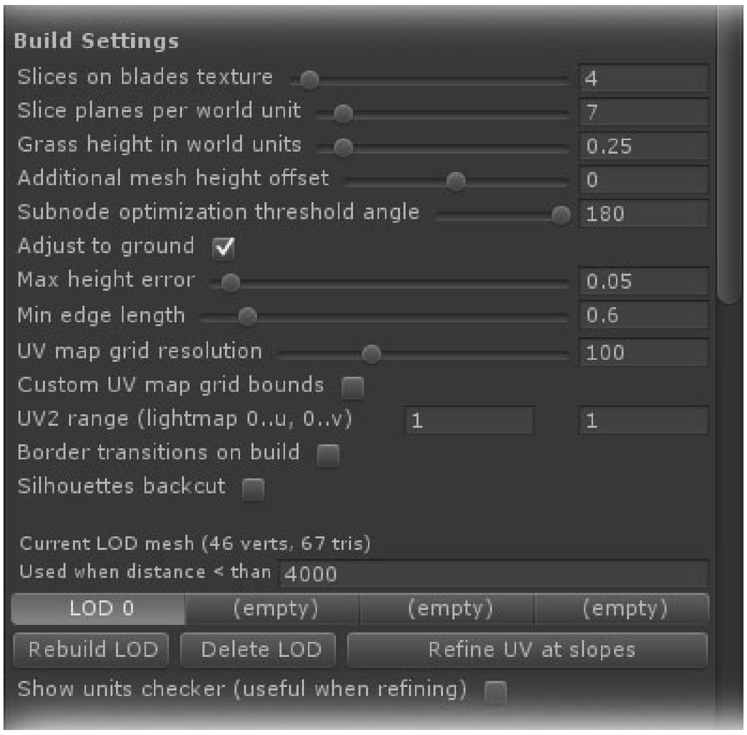

Current LOD mesh (22 verts, 31 tris)
Used when distance < than 4000
LOD 0 | (empty) | (empty) | (empty)
Rebuild LOD | Delete LOD | Refine UV at slopes
Show units checker (useful when refining) ☐

Fig.4

We may notice that only a few subnodes constituted mesh vertices – only that near the node 1. That's because mesh builder optimizes number of polygons so you don't have to care about it at all ! You may, however, not be much happy about optimization. You can then disable optimization on the borders by unchecking **Optimize collinear subnodes** (figure 3 - node properties). You may also enlarge ***Subnode optimization threshold angle*** to 180, so only "really" collinear subnodes will be optimized:

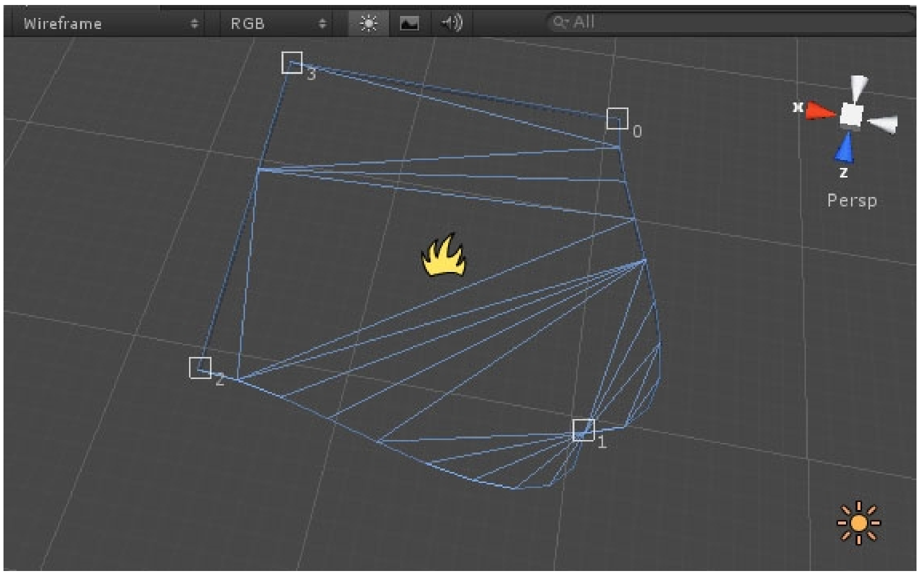

Build Settings

Slices on blades texture	4
Slice planes per world unit	7
Grass height in world units	0.25
Additional mesh height offset	0
Subnode optimization threshold angle	180
Adjust to ground	✓
Max height error	0.05
Min edge length	0.6
UV map grid resolution	100
Custom UV map grid bounds	☐
UV2 range (lightmap 0..u, 0..v)	1 1
Border transitions on build	☐
Silhouettes backcut	☐

Current LOD mesh (46 verts, 67 tris)
Used when distance < than 4000
LOD 0 | (empty) | (empty) | (empty)
Rebuild LOD | Delete LOD | Refine UV at slopes
Show units checker (useful when refining) ☐

Fig.5

Click ***Rebuild LOD*** and the result should look like this (switched to wireframe view):



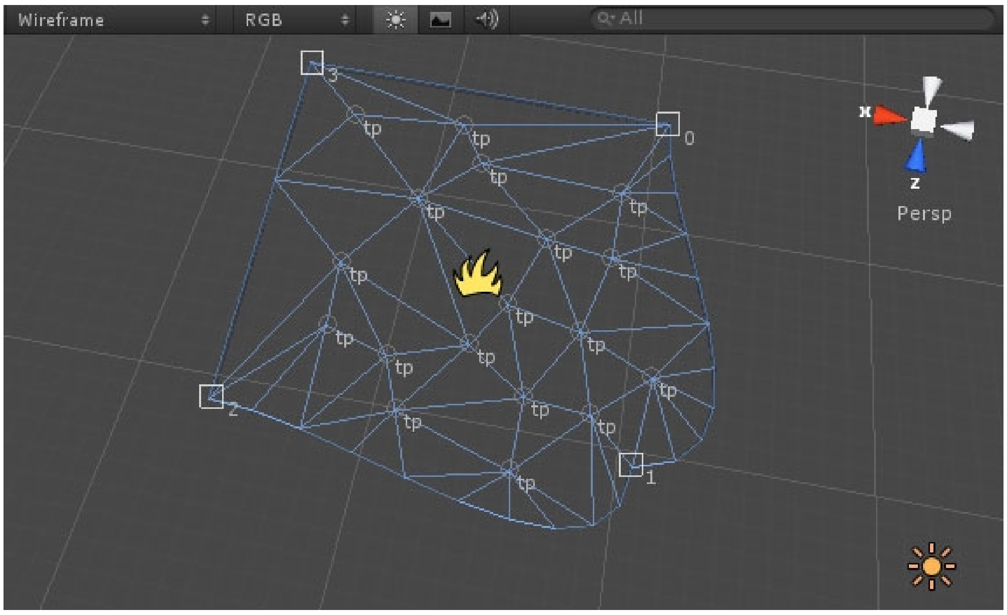
Build parameters 1

That's the moment we could introduce other parameters from grass inspector *Build Settings:*

Slices on blades texture	Number of slices on texture. Refer to *_Textures/grass_blades.png*. Blades texture should be always square, but can consists of any number of slices. By default in blades texture (512x512) there are 4 slices (512x128 each). When changing texture, remember adjusting this setting and ***Rebuild*** the mesh that makes changes take effect.
Slice planes per world unit	How dense grass planes are placed inside the mesh volume. Lower number reveals grid structure at sharp viewing angles. To hide grid structure you can also adjust ***Bending for sharp view angles*** described in shader settings section below.
Grass height in world units	Height of grass in world units. Changes take effect after rebuild.
Additional mesh height offset	Offset that will be added to all mesh vertices in direction normal to the ground (underlying collider or plane y=0). Changes take effect after rebuild.
Subnode optimization threshold angle	If angle between subnodes (previous / actual / next) is higher, subnode is treated as “collinear” and optimized when mesh is built unless **Optimize collinear subnodes** is unchecked for node the subnode belongs to.
Adjust to ground	Typically this should be always checked unless you don't want mesh to be adjusted to the ground, but build “as is” only by positions of nodes/subnodes.
Max height error	Number of polygons in resultant grass mesh depends not only on nodes/subnodes count and optimization settings. When mesh is built grass editor checks if it “fits to ground” at the center of every edge. If not – it puts additional vertex here. The whole process is repeated till the moment such condition is satisfied for every edge. For example if underlying terrain is wavy, mesh builder will add necessary polys to hold maximum difference of distance between ***Grass height in world units*** and mesh vertices below this error threshold. So – the lower number here the more complex mesh could be (but more accurate / smooth).
Min edge length	Here we can limit length of mesh edges so too much mesh tessellation does not take place and resultant mesh is not too complex.

Typically you'll adjust ***Max height error*** and ***Min edge length*** params and rebuild mesh till mesh quality vs. its complexity is OK. We can use up to 4 LOD (level of detail) meshes per grass object, each of them can be used at different distances from camera. See fig. 5 – we use LOD 0, but we can choose 3 other (empty) slots and build meshes with different number of polys. If you want to use LODs, you can attach *_Scripts / VolumeGrassLODAdjuster.cs* script to the grass object. It'll automatically switch between LODs basing on distance between grass object and main camera (Camera.main). You can also use different material for very distant grass objects by checking ***Use simple material beyond given distance*** in *VolumeGrassLODAdjuster* script.

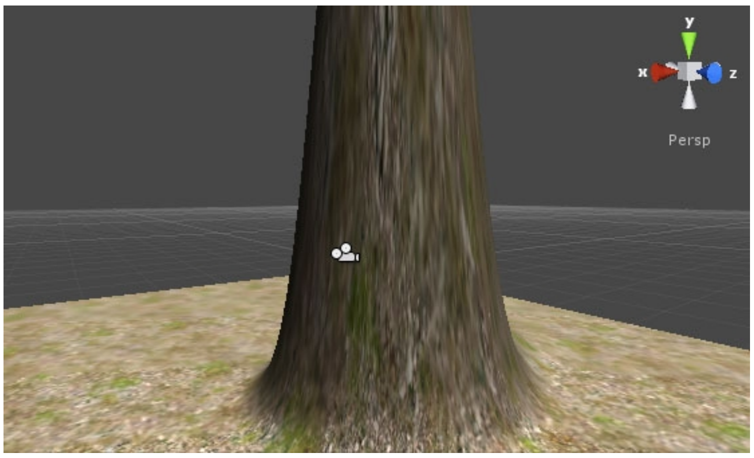
Sometimes there is necessity to tessellate mesh a bit different way than mesh builder suggests. Great help are additional tessellation points that can be placed anywhere inside grass shape. They always constitute mesh vertices (they are not optimized by any mean). They can be edited either in Edit or Build mode. **Shift+Alt+click** to add some tessellation points and rebuild current LOD mesh. Result could be something like this:



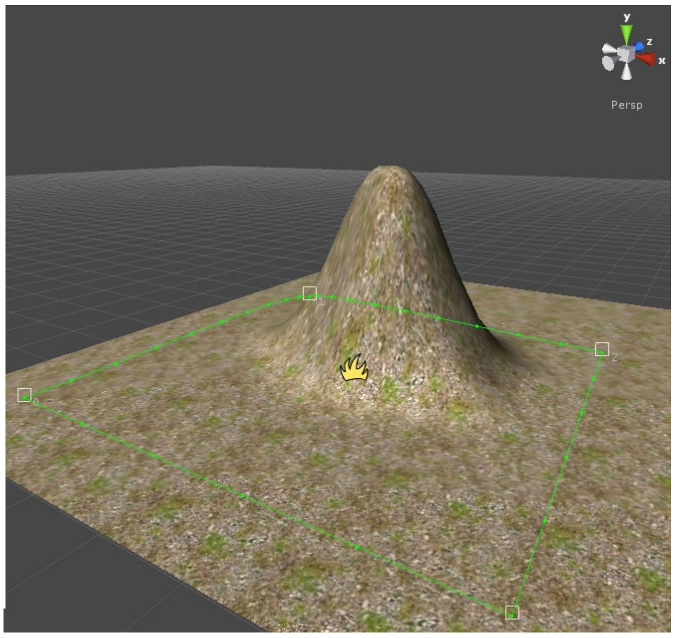
Tessellation points have the same properties as nodes in edit mode – select a tessellation point and you can adjust numerically its properties (position) in inspector (below LOD slots). If there are too many tessellation points and mesh becomes illegible in scene view you can turn them off from displaying in build mode by unchecking ***Show tessellation points in build mode*** (fig.1).

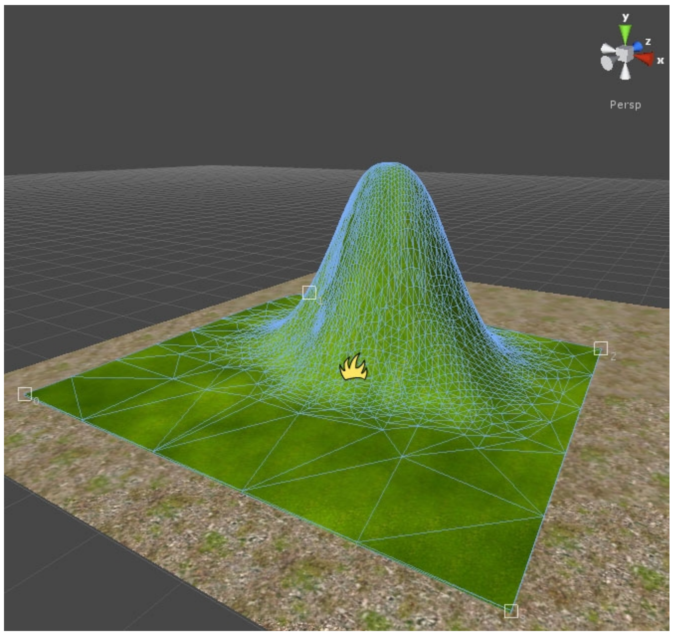
Build parameters 2

Grass mesh should have proper UV coordinates that map inner space inside the mesh volume. Ideally UV space shouldn't be stretched and every part of grass should have the same UV proportions. To give you an idea when stretching occurs build Unity terrain and place very steep slope there. Next paint it with texture and see what happen – UV mapping is realized by simple projection along y axis so slopes look like that:

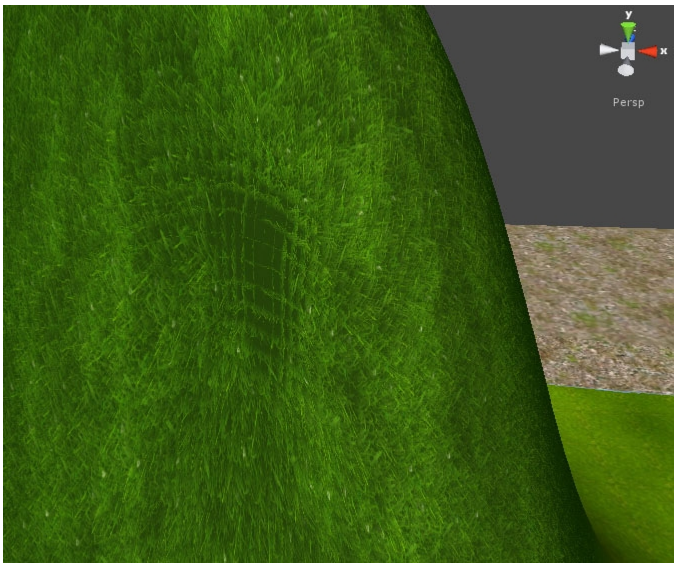


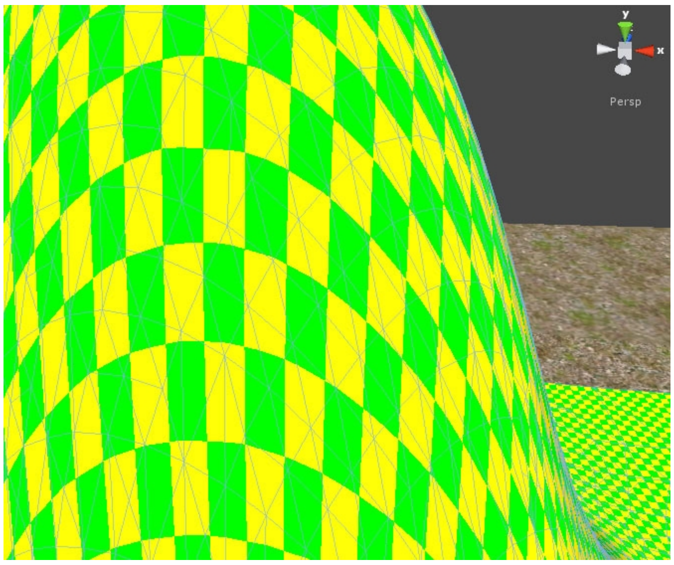
Now consider building grass on a terrain shaped such way:





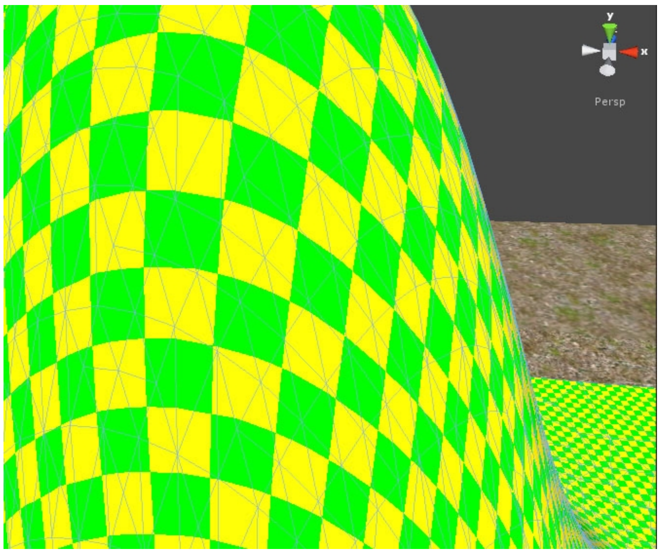
Mesh builder correctly sticked grass to the terrain collider, but when we take a closer look on grass it's stretched on hill:





You may not notice it instantly, but if you check ***Show units checker (useful when refining)*** (refer to fig. 5 – bottom) you'll see that slopes are not filled by squares but rather rectangles. This will result not only in worse look but also prevents the shader from correct distance calculations inside volume which are necessary to execute correct z-testing (described on pages 9/10).

Mesh builder can try to fix it. Click ***Refine UV at slopes*** a few times and you should get much better UV mapping:



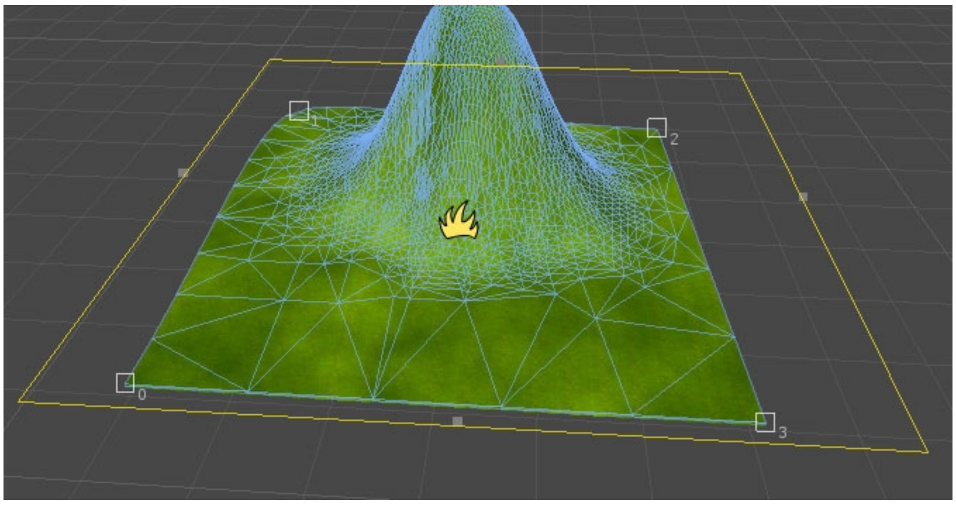
Now you can uncheck ***Show units checker (useful when refining)*** and grass at slopes should not be stretched that bad. Note that on very wavy / complicated surface it's impossible to resolve UV mapping and mesh builder do it approximately only.

UV map grid resolution

When resolving UV mapping mesh builder makes internal grid of UV coordinates. Resolution of this grid is set by ***UV map grid resolution*** setting. For large grass areas or more complex terrain it's better setting it higher to get better accuracy, but it also cause the UV resolver works slower and we'll need more steps (clicks on ***Refine UV at slopes***) to get better results. Generally default value of 100 works fine unless you've got very large grass area or trying split one grass objects into few smaller.

Custom UV grid bounds

Sometimes we may need to split one big grass mesh into a few smaller. That's wise in terms of performance since occlusion engine can do better job on many small static objects rather than one very big object. First - at edges where grass objects touch each other you should uncheck ***Build sidewalls*** in node properties (fig.3) so adjanced grass meshes don't have sidewalls at common edges. Second – check ***Custom UV grid bounds*** for both neighbor meshes and set exactly the same bounds. Bounds can be edited by dragging handles placed in the center of each bound edge (little gray boxes on yellow edges):



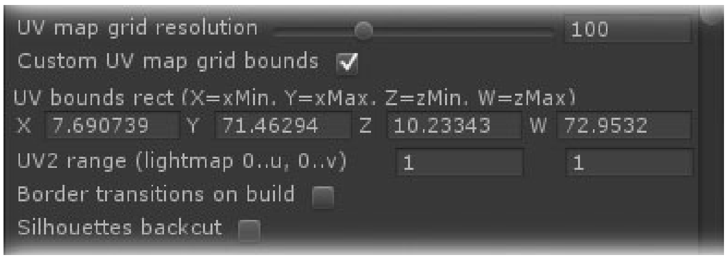

UV map grid resolution 100
Custom UV map grid bounds ✓
UV bounds rect (X=xMin, Y=xMax, Z=zMin, W=zMax)
X 7.690739 Y 71.46294 Z 10.23343 W 72.9532
UV2 range (lightmap 0..u, 0..v) 1 1
Border transitions on build ☐
Silhouettes backcut ☐

Fig.6

To be sure that adjanced grass objects (on the previous picture we've got only one grass object however), shares the same UV bounds, adjust their numerical values in inspector (***UV bounds rect***). Then two grass objects should fit to each other seamlessly. Take into account that build settings for adjanced objects should be the same so that mesh builder give the same set of vertices on seam.

UV2 range

This is used for lightmapper. Mesh builder automatically fills uv2 coordinates so lightmapper can process grass geometry. **Remember** that marking grass object "static" for lightmapper, sidewalls child object **may not be** marked as static as it's used only internally for resolving sidewalls z-test cutout. Sidewalls subobject doesn't have any uv coords and lightmapper will throw errors attempting to lightmap grass.

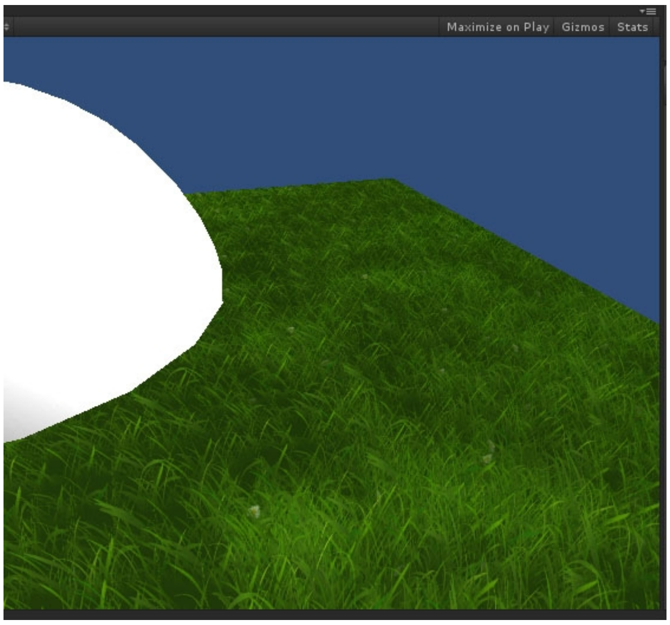
Border transitions on build

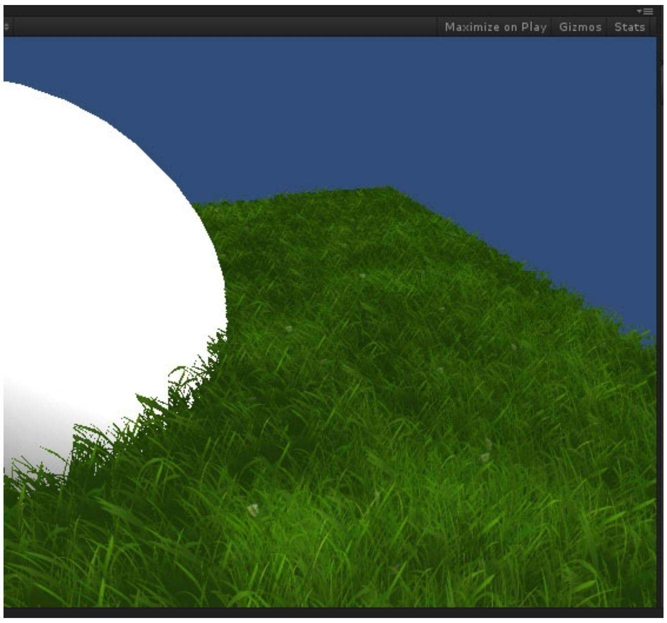
When this is checked borders of grass will be automatically marked as "zero-height" (refer to ***Modifying grass height / coverage*** section below). **Remember** that if border vertices are the only ones (there are no inner vertices, for example like in soccer field sample scene) checking this will cause grass "disappear" - whole volume will be "pushed down" below ground level by shader.

Silhouettes backcut

Before this setting is described, attach *_Scripts / SetupForGrassRendering.cs* to camera. This script renders geometry that should be "cut inside grass" into depth texture so grass shader knows where cut "holes" while rendering. Proper z-test works only in play mode and is not visible via camera of editor scene view. Choose layer at which you'll place objects that intersect the grass (***Culling Mask***) in camera inspector for added script.

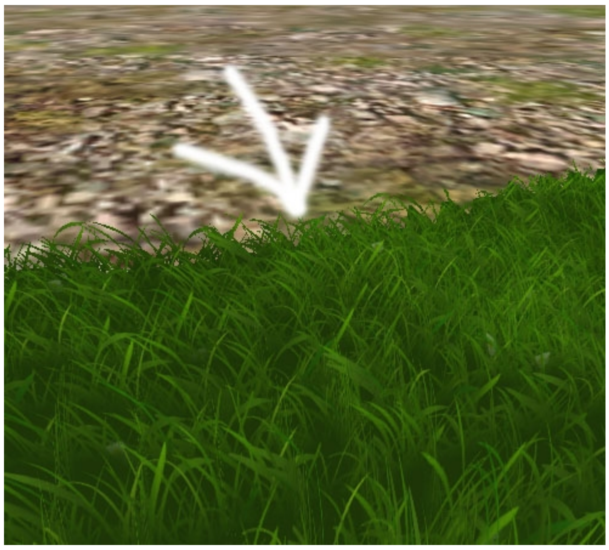
Every rendered geometry (unless it's for example transparent) writes into depth buffer so rendering pipeline knows what's "in the front" of the other. Grass mesh is no exception here. If we wouldn't resolve z-test internally inside grass shader, its mesh would be z-tested like any other solid geometry:

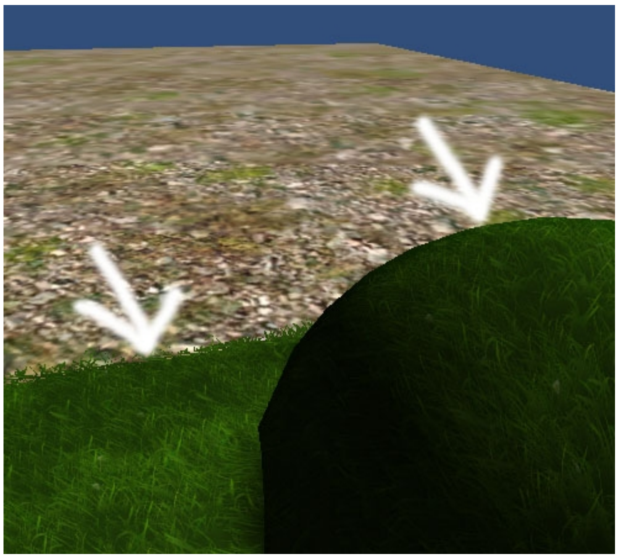




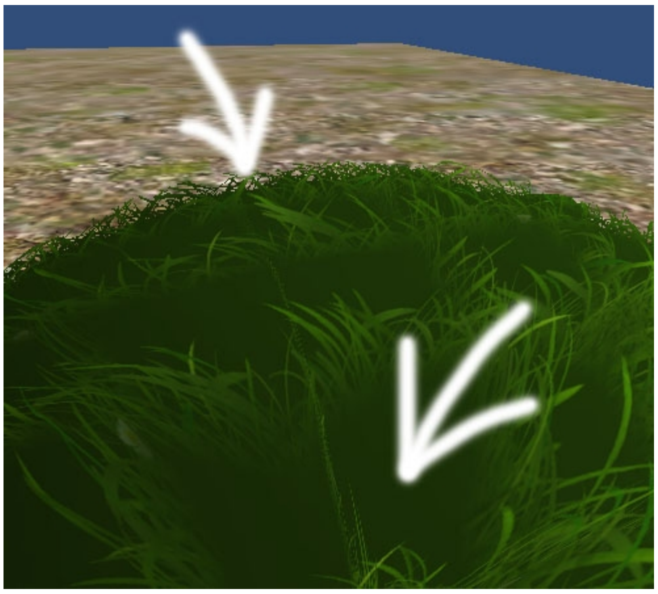
On the left no internal z-test is performed. On the right grass borders and sphere are z-tested properly. Note that sphere has to be placed on layer that is selected in *SetupForGrassRendering* culling mask script and every other object we need to intersect the grass should be placed on layer that will be rendered into aux depth buffer. We don't have to worry about z-testing sidewalls, they are always rendered into auxiliary depth buffer for grass shader so when we look at the borders from “the inside” to “the outside” grass looks fine. In fact inner sidewalls are placed on reserved layer that can be defined in *GrassRenderingReservedLayer* script. Camera that renders depth buffer renders everything from its culling mask (white sphere in example above) plus reserved layer (inner grass sidewalls).

So, now let's back to our silhouettes problem. Inner sidewalls (child object of grass object after build) are rendered only into auxiliary depth buffer. They resolve special case of grass silhouette problem when looking at the borders from “the inside” to “the outside”. By checking ***Silhouettes backcut*** you can force the whole back geometry of grass is rendered, not only inner sidewalls. On very wavy terrain it may improve silhouettes a bit. Unfortunately only “may” and only “a bit”. To show the problem, please look at the following pictures:





On the left we see grass border viewed from inside. Silhouette is properly resolved and separate grass blades are visible. On the right we see small grass area placed on sphere (our “hill”). And the bad news are that there is no simple method to render silhouette of grass on hill border. Ray which is traced inside shader assumes that it hits <u>flat surface</u> (*) and knows nothing about mesh curvature as it travels inside volume. It will break tracing only when hit something rendered into auxiliary depth buffer (like sidewalls). However if we check ***Silhouettes backcut*** ray has chance to hit back geometry of grass mesh so it can improve silhouette a bit:



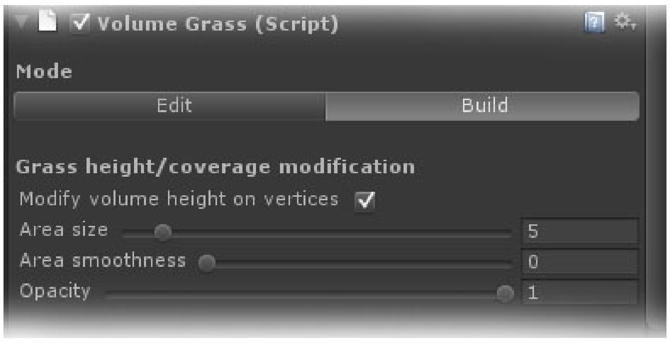
Note that silhouette on hill above is visible (top-left arrow marker) only because ***Slice planes per world unit*** has been reduced and the backdraw is that grass grid cells are visible (bottom-right arrow marker).

Conclusion is that silhouettes on very wavy terrain are <u>not</u> generally resolved so using ***Silhouettes backcut*** is not recommended as it engages full grass geometry rendered into auxiliary depth buffer and may introduce another artifacts.

**) As the author of the package tried not to trace ray along straights, but “bend” it along its travel path following geometry curvature, tested method wasn't much precise and caused shader to be much more complicated so this idea has been dropped.*

Modifying grass height / coverage

By default, after (re)building grass has its nominal height as entered in ***Grass height in world units*** (fig.5). That's how we get straight borders on grass sidewalls. If we want more control over grass coverage we can modify its height at mesh vertices. Height modification is realized in vertex program of shader (so original mesh model isn't changed). In **Build** mode we've got additional switch that allows “height painting” over the mesh:



Volume Grass (Script)

Mode: Edit | Build

Grass height/coverage modification

Modify volume height on vertices ✓

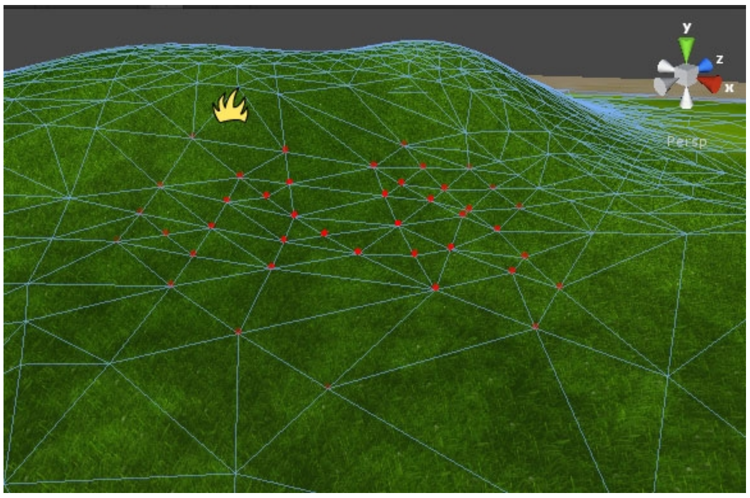
Area size 5

Area smoothness 0

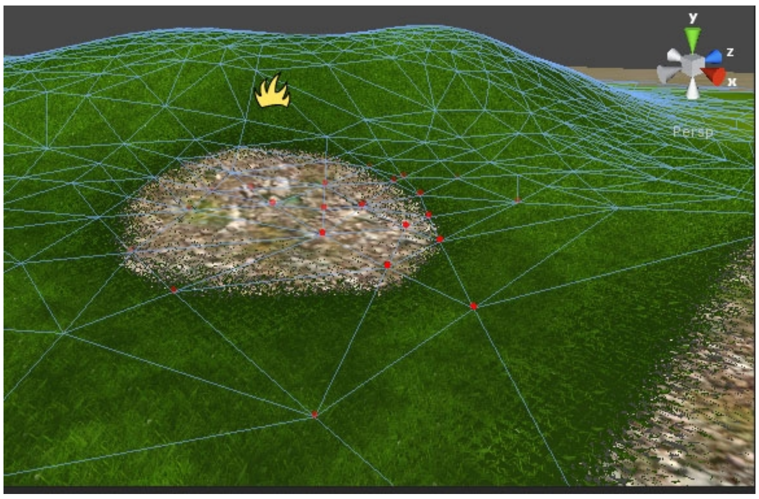
Opacity 1

After checking ***Modify volume height on vertices***, 3 additional properties appear: ***Area size*** which is simply our “brush” radius, ***Area radius*** which is brush smoothness and ***Opacity*** – strength of brush height modificator.

Now we can move mouse cursor over grass mesh in scene view and we'll see red dots that defines which vertices, and how much will be modified when “painting”:



On this example brush smoothness has been set to 1, so points on the radius are almost transparent. Clicking/dragging mouse allows reducing grass height at selected area. If we hold **shift** when painting dots become green and we can paint in the opposite “up” direction to the original height. Holding **ctrl** while in “Hand tool” mode we'll be able to drag camera in scene view without painting.



Result of modifying grass height could look like on the left picture. That's how grass can be modeled to get any shape / coverage on ground. In addition ***Border transitions on build*** (property at fig.5) was chosen so mesh sidewalls have been automatically reduced to ground level on build.

Note that when modifying height we don't see nodes/tessellation points and they can not be modified then.

Grass shader and its parameters

When you look at grass shader attached to built grass you'll quickly get an idea how grass is rendered. It needs at last one texture - "***Grass blades (RGBA)***" with slices of grass rendered on grid inside mesh volume.

Some of functionalities below DO NOT WORK unless we explicitly enable them in shader code. They may be disabled due to performance adjustments (less functionality for more performance).
It's best idea to look into _Shaders and Materials/Resources/GrassShader (GrassShader ALT) code first and find defines section where full discussion about functionality, performance and its factors takes place. Also, look at *Release history notes* section below where some new customized params have been introduced.

The following shader parameters are:

Ground Color (RGB)	Color of ground (multiply blend with texture) where ground texture is visible and for far distance
Ground texture (RGB)	Texture used to fill the ground (if appropriate setting is defined in shader code) and used for far distance
Texture tiling at far distances	Tiling of ground texture viewed at far distances
Far distance	Distance at which grass starts fade into flat ground texture. At far distances shader works faster (exits all unnecessary calculations). **) It won't work faster for ALT version of shader (refer to compatibility issues section below)*
Far distance transition length	Length of above fade
Blades Color (RGB)	Color of grass blades blended multiplicative with blades texture
Grass blades (RGBA)	Texture of grass blades slices – main factor of grass look.
Grass blades back color (RGB)	Color used when ray hits iteration limit without resolving pixel color completely (some opacity remains).
Grass blades back texture (RGB)	Texture used when ray hits iteration limit without resolving pixel color completely (some opacity remains).
Bending for sharp view angles	Setting that allows "bending" grass blades when viewed at sharp angles. That's how we can hide grid structure without setting ***Slice planes per world unit*** very high.
Blades blurring distance	At distances close to the observer shader uses MIP0 level of grass blades texture. Without this option grass areas placed far from us would look "noisy". Here is the distance at which shader starts using higher MIP levels from grass blades texture.
Blades blurring transition length	Here's how fast shader uses consequent MIP levels. Low transition length cause very blurry look near ***Blades blurring distance***
Blades blurring MIP distance limit	Here we can limit max MIP level used for blurring so for example MIP level 8 is not used at all (MIP level 8 for 512x512 texture reduces to 1x1 so no details are visible).
Glow to contrast ballance	If we decide to use it, here's where we set how much "glow" add to the grass. Refer to shader defines section for further info.
Slice bottom coloring (RGBA)	Color that's used for coloring bottom of the grass.
Slice bottom coloring noise tiling	If bottom coloring is noise texture dependent here you can define tiling of noise texture used for coloring.

Slice bottom coloring distance fade	Distance at which bottom coloring fades to zero.
Slice bottom coloring for far distance	Factor that allows reducing bottom coloring for far distances.
Slice bottom coloring border damp	Factor that allows reducing bottom coloring for borders (mesh sidewalls observed ”from outside” and grass of reduced height).
Noise coloring (RGBA)	Color used for additional grass colorization
Noise coloring tiling	Tiling of noise texture used for colorization
Noise	Noise texture 1 (RGBA) – in shader code you can bind any of noise texture color channel to different grass parameters (like wind, coloring, etc.)
Noise2	Noise texture 2
Noise for slices hash	Texture used to randomly move slices along grid and prevent “patternizated” appearance. In shaders originally placed in the package it's not used. Custom hash function is used instead as it gives reasonably good “random” values and works faster than texture lookup.
Wind amplitude	Amplitude of wind.
Wind speed	Speed at which offset of noise texture is changed.
Wind frequency (NoiseTex tiling)	Frequency of wind changes.
Border fray strength	Setting used to “fray” grass sidewalls viewed from outside so they don't look that “straight”.
Border fray noise tiling	Border fraying is realized by a channel of noise texture. Here is tiling for the fraying texture.
Alpha cutoff	Shader is realized as transparent cutoff. Here is cutoff parameter used as output from surface fragment program.
Linear motion blur like effect	If “quasi” motion blur is enabled in shader and in *SetupForGrassRendering,* this property will be adjusted automatically as camera will move. Effect may be not much convincing for general use, but for camera running very fast in one direction (for example in racing game) it may be OK'ish. ***Motion Blur Grass Objects*** in *SetupForGrassRendering* is array of grass objects that will be adjusted by this script. You can fill this array automatically pressing ***Fill Motion Blur Grass objects array***. If you need for some reason (post process) your camera renders depth (into _CameraDepthTexture) you can check ***Render depth***.
Motion blur distance	Distance at which “motion blur” starts fade to zero.
Motion blur distance transition	Transition length of above fade.
Motion blur direction - (cam steered) (xyz)	Direction of camera movement. If you want further explanations about it refer to *SetupForGrassRendering* script code where “motion blur” effect is implemented.

Note that disabling some feature by setting its property to zero (for example wind amplitude) is not effective in terms of performance. If you don't want wind, disable it finally in shader defines section rather than via material properties to make shader simplier (and faster).

*Shader should work well in **forward** and **deferred** rendering paths, however in **forward** you can experience some trouble with shadows on grass borders. Beware that in **deferred** there are always TWO passes. In forward we can render it with ONE pass if only number of dynamic (not lightmapped) lights is small. So, the fastest possible rendering path is **forward**. As every grass shader pass is expensive, consider using this path.*

Grass shader compatibility issues

Shader has been originally implemented on PC, nVidia GT240 GPU. As it need texture lookup operations inside conditionals it turned out that this is very problematic on openGL machines (Macs). To put texture lookup operation inside conditional we can't use tex2D(), but tex2Dlod() instead. This compiles for openGL only if #pragma glsl is used. The problem is that even if it compiles, it DOES NOT work for many Mac configurations. Trial and error method leads into conclusion that only nVidia GF8000 series and above are “safe” on Macs and other GPUs are typically problematic. We can use tex2Dlod() in fragment program on such hardware, but NOT INSIDE conditionals. For example on OSX10.6 and Radeon X1600 it renders “black hole”. On other GPUs it may **crash** Unity... :/

But don't worry too much :) *VolumeGrass* script takes care about it. If it detects problematic card on Mac it switches shader to “Grass Shader ALT” version. Alternative version has almost the same functionality, but renders slower as it doesn't benefit from optimizations (for example skipping calculations when pixels are far enough). If you have Mac you may manually change shader on grass material to “Grass Shader” but ON YOUR OWN RISK :). On PC grass always uses “Grass Shader” version.

When adjusting defines section in shader code **remember** to adjust it in **both** shaders so that PC and Mac users get similar results of grass appearance. You can of course disable some functionalities in ALT version of shader because it's slower, but be aware of the fact that there are **2 shaders**. On PC you will be able to test ALT shader in editor by checking ***Force ALT (slower) shader in editor*** (fig.1). You can also direct script to hide grass mesh on incompatible hardware by checking ***Disable(hide) on incompatible hardware***. Incompatibility test is performed in *VolumeGrass* script (shader model <3.0). You can change it by overriding function exposed at the beginning of script (oldHardwareDefTest()). If grass is not hidden but can't be run on user's hardware it falls back into “Diffuse” shader.

Another compatibility issue is z-testing buffer rendered by custom replacement shader (*_Shaders and Materials/Resources/GrassRenderDepth).* This shader does not write into color channels at all (relying on native z-buffer only). It may not work for some hardware that is compatible and efficient. For example on PC, Win XP, GF7900GS z-testing don't work properly with this custom depth shader. In such case you may either switch off **Use CustomDepth Shader** in *SetupForGrassRendering* camera script or tweak *GrassRenderDepth* shader your own way that will be less “hackish” and more reliable (at cost of a little performance loss). I strongly encourage you to customize your own depth shader for grass z-testing as it may benefit from tag replacement (for example we don't need tree leaves to be rendered as it won't intersect grass for sure, the only part of tree we need is bark with its bottom part).

Release history notes

Changes in v.1.1 (mowing expansion) are:

- 2 new parameters for dedicated SubShader (LOD 698 – LawnMower scene SubShader) - _AuxTexture with grass height and tiling param of this texture. For proper setup of params look at dedicated SubShader code and Lawnmower scene ***(VolumeGrass / VolumeGrass Sample Assets / LawnMower)***. Known issue is that mowing render texture buffer resets at resolution change (for example going fullscreen in webplayer). Possible workaround is copying this buffer into other custom texture every frame, so original mowing buffer can be restored after res change.
- small tweak in shader code (all subshader versions) that allow better hidding slices grid when viewed at sharp (near 90 degress) angles (esp. when **FILL_GROUND_BY_TEX** is active). Look for 2 additional defines:
  BOTTOM_COLORING_SHARP_ANGLE_RADIUS_MULTIPLIER
  BOTTOM_COLORING_SHARP_ANGLE_LOW_THRESHOLD
  in shader code for explanations.

Summary

For further information look for VolumeGrass Unity forum threads where eventual problems could be discussed. You can also refer to ready made sample scenes *“meadow”, “soccer” and “lawnmower”* inside package to look at the grass at work. All scripts and shaders inside the package are richly commented where they should be, so look into them first if you've got any doubts. On *“lawnmower”* user will probably notice cam error thrown. This is only Unity editor bug and does not affect built players. For reference one may follow forum thread:

http://forum.unity3d.com/threads/78259