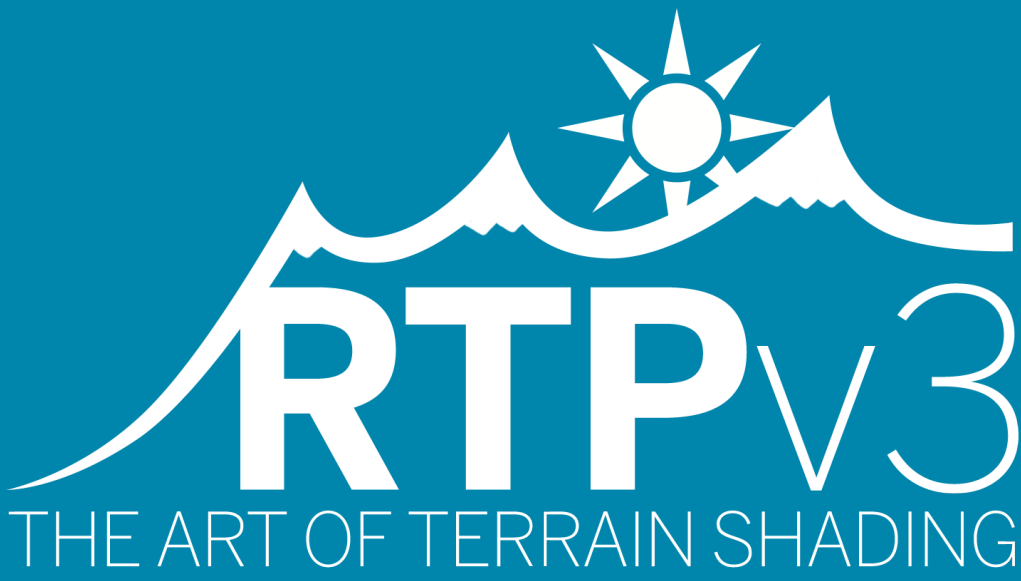


V3.3 Documentation

January 25th 2016

RTP
unity

Preface

Thank you for buying RTP. At the moment of release this solution gives unmatched terrain quality that will bring your ideas to life at AAA level. It consists of brand new terrain replacement shaders that can be optionally used on arbitrary mesh (not necessarily terrain) and toolset script for seamlessly blending static geometry against the terrain. Shaders require SM3.0 GPU card which benefits from conditional branching. The shader itself is quite fast on mid-end hardware, even when using up to 8 parallax occlusion mapped detail textures. Note that such quality is reserved to AAA engines like CryEngine only on ultra levels which are handled well enough on very strong machines only.

Although we need power to get all the features at time, RTP is quite performance wise. It skips unnecessary calculations where applicable and manages rendering basing on distance. In U4 RTP can use separate shader to handle far distance. In combination with shader option to render only global color map far from camera, your large environments can work almost as fast as built-in Unity terrain shader (bump mapped) without compromising the quality at close distance. Moreover special shader LOD manager lets you easily adjust and recompile shader code to tailor it to your needs, so you don't need to deal with RTP shader settings by hand, but only if you want. Number of options that can help adjusting quality against performance is quite large, but even with basic set of features like global color map, perlin normal map and 4 layers your terrain will look considerably better than anything you can get now considering not only existing Unity terrain solutions, but even other existing hi-end 3D engines. And this not only will look better, but also run decent even though we'd use parallax mapping on every layer.

RTP3.3 update is dedicated to Unity5.3 only (with RTP3.2 to work on lower Unity versions). Most pronounced improvements are both DX11 and openGL4 (Macs) tessellation, Unity PBR model, simplified features.

I would love to thank Peter Rößl (aka Becoming) for his help making RTP3 demo scene. His artistic support and technical inspirations given were invaluable for this release. Many thanks also to Nathaniel Doldersum (aka eagle555) for his support and cooperation making our products work so nice together.

Table of Contents

The docs below are organized in a few main sections where you'll find out more about:

1. RTP features list
2. First steps working with the system
    - Grouping terrain objects
3. Basic ideas behind
    - Layers, passes & shader LOD levels
    - Heightblending
    - Actual shaders used
    - Shader mode, its material & multiple terrains
    - RTP_LODmanager
4. All settings explained one by one
    - Settings - Main
        - Tiling, MIP adjust & lighting
        - Shading color adjustements
        - Heightblend – AO
        - Hot air refraction on emissive layers
        - Lightmapping your terrain
        - Presets
    - Settings – Parallax
    - Settings – UV blend
    - Settings – Vertical texture
    - Settings – Global maps
        - Global color map with holes feature
        - Global normal map / tessellation height&normals map
        - Global trees /shadow map
        - Grayscale detail textures
    - Settings – Snow
    - Settings – Perlin normal
    - Settings – Water & caustics
    - Settings – Super detail
    - Settings – Glitter
    - Layer properties
        - IBL / PBL properties
        - PBL – gloss by normal variance baking utility
        - Layer adjustment
5. Coverage
    - Compose
    - Acquire
    - Control maps
6. Combined textures
    - Atlasing
    - Height maps
    - Normal maps
    - Special
7. Geometry blend features
    - Terrain blending with a mesh
    - Terrain blending with sticked mesh surface
8. Advanced topics
    - Using RTP with Shadow Softener
    - RTP shaders – tweaking features manually
    - RTP properties
    - Custom geom blend compliant shaders

9. Standalone RTP shaders for voxel terrains, planets, any objects that might use RTP shading
10. RTP brief glossary
11. RTP3.2 & 3.3 change log
12. Troubleshooting

1. RTP features list

Main shader features of RTP 3.3 release:

- 8 POM textures with self shadowing in 1 pass
- Hard, well defined blending between detail textures (using height maps)
- Unity's PBR (physically based) rendering pipeline
- Unity5 realtime GI (globall illumination)
- Tessellation features with DX11/openGL4 (smoothing terrain as well as displacing detail heightmaps)
- Terrain holes - paint terrain holes, seamlessly cover with additional mesh (cavern), collisions resolved
- Remove tiling with UV blend feature to mix layer color with any other at different scale
- Global color map with adjustable filtering, saturation - separate blending at different distances
- Perlin normal map used to add more detail at adjustable distance
- Dynamic snow – coverage by mesh slope, normal map, height, randomly by perlin
- Glitter – the same model as in my UBER shaders – most usable with snow
- Dynamic water – adjusted per layer (reflective, refractive, slope direction flow + flowmaps, rain drops)
- Caustics for realistic underwater environments
- Emissive features – terrain itself and dynamic water over it can be emissive (lava, fantasy glowing parts)
- New triplanar texturing using PM shading and now correct perlin mapping
- Vertical map (global) texturing to reproduce realistic geological structures at slopes w/o stretching
- Superdetail at close distance – additional perlin normal map to boost surface details
- MIP adjustment to sharpen details when needed

Triplanar shader version prepared for usage with meshes that uses vertex color to control coverage (simply use it on meshes coming with existing voxel terrain solutions). Tested on any mesh, RTP can render properly Planetary Terrain meshes with unparalleled quality.

Other RTP 3 features are:

- Compose terrain coverage right inside RTP inspector (RGBA control maps merged with built-in editor)
- Runtime shader LOD / feature set adjustment (POM self-shadowing / POM / PM / SIMPLE / CLASSIC)
- New, powerful geometry blending to seamlessly melt any mesh with terrain – quickly resolve slope stretching w/o triplanar mode, add overhangs, caves, paths, hide geometry intersecting parts of environment objects like rocks, tree roots against the terrain surface. New features allow painting polygons on the terrain – you can quickly put virtually unlimited number of surface elements. Package includes a lot of additional shaders that might use RTP features and can be used on any scene or as geom blend.
- RTP can be used on Unity terrain or any arbitrary mesh (with mesh vs. geometry blend working)
- Brand new UI – user friendly – navigation between layer properties and any settings is very quick now
- Quick Substances acquisition. Put procedural material on a layer slot and click the button – color, normal map and height map will be automatically retrieved and saved on disk. Your workflow using substances will be a breeze now !
- Presets – you can quickly save/restore terrain setup. You can also easily interpolate between presets with a few lines of code (i.e. for example dynamic summer/winter change)
- Convenient multiple terrain handling. Tailored to work well with Terrain Composer and World Composer
- RTP shading can be achieved on any mesh with standalone versions (material based w/o RTP scripting)
- Configurable shaders now have powerful custom inspectors that allow you to adjust/recompile shaders right from editor (no more need to tweak any shader code manualy).

System itself **doesn't necessarily need** any Unity PRO feature.

2. First steps working with the system

When you import the package you can choose components (shaders, textures) needed, but for example scenes to work it's advisable to import the whole package. It will be placed inside *ReliefPack* subfolder in your *Assets* folder. If you upgrade from version 3.2 – remove it before installing RTP3.3. You should take this into account and preferably backup your project before upgrading. Remember that it's necessary to install the package into Assets/ReliefPack folder.

Generally RTP for terrains can be used in 2 scenarios – on Unity's terrain or arbitrary mesh, so:

1. When you need RTP using Unity terrain follow the steps below

- Prepare your terrain settings like its position, resolutions, height map;
- Add number of layers you'll use (either 4 or 8). Use Unity's terrain script to make this. It doesn't matter which detail textures you'll place – you can easily change them later, but it's important that all layer color textures used are of the same size;
- Look for *ReliefTerrain.cs* script and put it on your terrain object (you can use Component menu – Relief Terrain/Engine... component for this);
- You'll find that _RTP_LODmanager object is added to your scene;
- Get into the inspector of above object manager and choose the features you'll need for your project

Fig1

Clicking on *Recompile shaders for given features set* button will require some time to recompile shaders. After that you're ready to start dealing with the script inspector placed on your terrain.

This is important to look at hint boxes placed next to inspector features.
You'll find many useful informations there so you don't have to look into docs.

2. When you need RTP using arbitrary mesh follow the steps below

- Prepare your mesh so that it has UV coords (typically 0..1 x 0..1) spanned over the model
- Look for *ReliefTerrain.cs* script and put it on your terrain object (you can use Component menu – Relief Terrain/Engine... component for this);
- You'll find that _RTP_LODmanager object is added to your scene;
- Get into the inspector of above manager and choose the options you'll need for your project. Refer to the screenshot below regarding terrains (there are terrain shaders enabled – you need to enabled it on mesh).

Grouping terrain objects

In presence of many terrain objects take care about objects grouping. When you put terrains in different place in hierarchy (lets say – terrain A and terrain B has different anchor game objects) they will use different object (*ReliefTerrainGlobalSettingsHolder* instance) that stores RTP shader state, textures and other parameters. Such ungrouped terrains behaves independently.

When you need different setup (different set of detail textures for separate terrain objects) use grouping described above. Put terrain objects inside different parents in scene hierarchy before attaching ReliefTerrain script. Recompile shaders in LOD manager if needed.

3. Basic ideas behind

Layers, passes & shader LOD levels

RTP has been developed to improve quality of your environments. It's basically a system that replaces shading pipeline of terrain surface. It's not "new terrain" system as it works on the top of Unity's built-in terrain. It's worth mentioning here how classic Unity terrain engine works like. Terrain detail color coverage is taken from RGBA texture (called splat control) where R channel level means coverage of 1st layer, G channel of 2nd layer and so on. This way we've got an option to use 4 layers. So how Unity can handle more ? It just uses next splat control texture and next set of 4 layers is rendered again (in next rendering pass) on the top of previous layers using additive blending. Just unfold your terrain object in the project view and you'll see control texture(s). The number of splat controls is *number of layers / 4* rounded to the next closest integer value. This means that whenwe have terrain with, let's say, 100k polys with 12 layers used, Unity would need to render it 3 times. More – when you're using more lights you'll have to multiply the number of rendered triangles by the number of lights that affect them (in forward rendering path). In deferred rendering path number of rendering passes are constant and the object is always rendered once (3 passes in our case – 300k polys). This is why shader coders try to handle as many layers as possible in single pass. RTP can handle 4 or 8 layers in one (first) rendering pass. Due to changes in U5 I removed possibility to handle 12 layers. Visually we can use another RTP feature to get virtually unlimited number of materials placed on the terrain surface. This will be described in chapter Geometry blend features. Although you can't use all the features in 8 layers mode, it's advisable to use it for performance reasons. The main disadvantage is 8 layers mode needs to use 2 atlases for layer color textures and won't allow you to use some features together as we'd run out of texture samplers in the resultant shader. Generally in 8 layers mode we've got 1 additional texture sampler for chosen feature (disable lightmaps to get 3) – water raindrops, caustics or vertical texture. In 4 layers mode we've got more additional textures and *_RTP_LODmanager* takes care about it. For example in 4 layers mode you can use both – water raindrops and caustics together. Even more options are available at the same time when you check *Use color atlas* option in 4 layers mode (3 textures more to use). The mode selection is shown below:

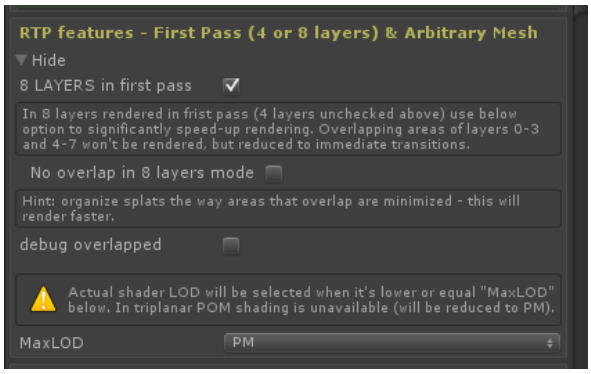
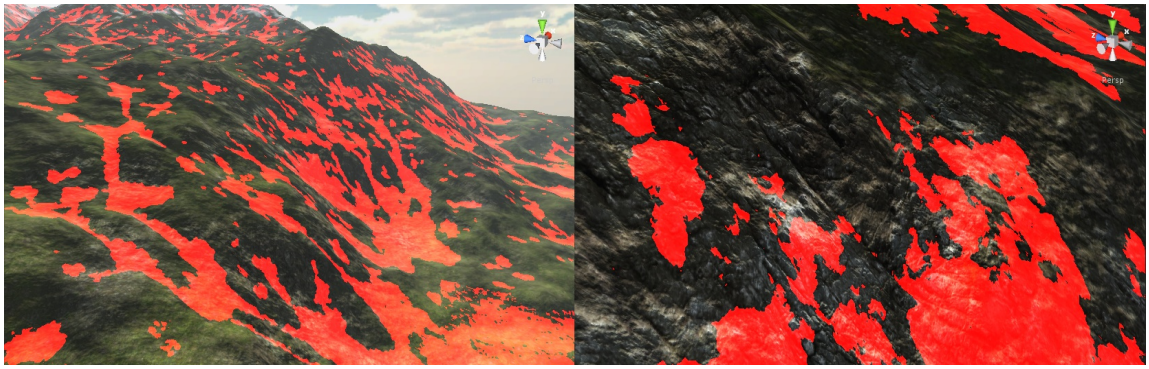


Fig 3

When we're in 8 layers mode we've got some additional options available. *No overlap in 8 layers mode* is very useful when we care much about the performance and we agree compromising a little bit of quality on the areas where any of layer 1-4 blends with layer 5-8. Although shader can handle it, this costs much more than rendering single set (1-4 or 5-8) layers – conditional branch inside the shader chooses the most expensive part where we have to resolve all values for all 8 layers to get the final color. When using *no overlap* mode shader judges which set of layers are "more visible" and uses it. This results in sharp transition between layer sets, but in most situations is visually acceptable. New in RTP3.1 actual transition edge is covered by global colormap pixel value thus edge is much softer. If you don't like the effect (because your colormap has values that don't fit to the detail color) you can disable this effect tweaking code of **.cginc** shader include (described in advanced topics section).

If you anyway decide to not use *no overlap* mode it's strongly recommended to arrange terrain coverage the way overlapping is minimized. You can enable *debug overlapped* to see where it occurs and try to rearrange your layers different way.

No overlap in 8 layers mode - off *No overlap in 8 layers mode – on (runs faster !)*



Debug overlapped – red areas are most expensive – shader manage 8 layers there

RTP shaders can adjust their quality and features runtime. It can be realized right inside the quality settings in your application or from the editor via LOD manager:

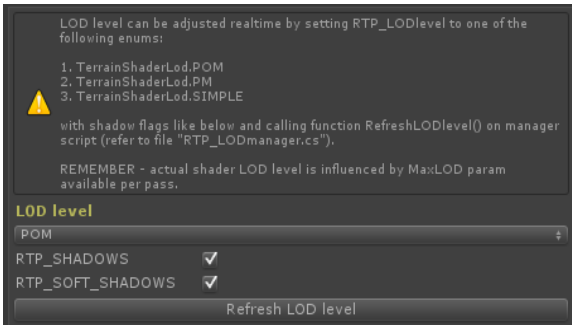


Fig 4

You'll have such options:

- POM (Parallax Occlusion Mapping). This is the most advanced and expensive shading level. You can enable self-shadowing here, too with optional self-shadow smoothing.
- PM (Parallax Mapping). Much less demanding than POM, but still good looking
- SIMPLE (no flowing water, superdetail, nor parallax, but heightblended)

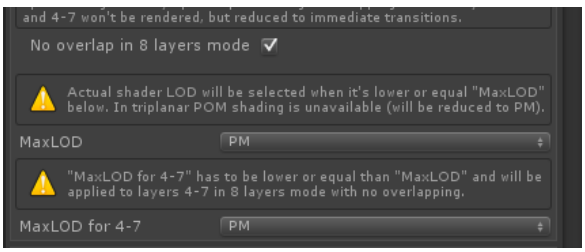


Fig 5

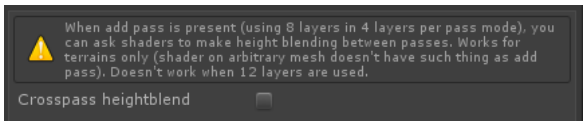
This is very important that actual shader LOD level used will be constrained by settings shown on *figure 5*. By default (after importing package) maximum shader LOD is set to PM.

RTP terrain/mesh shader can work in 2 modes, so you can render either 8 or 4 layers in first pass. When you've got less than 5 layers on your object, don't check 8 layers mode in RTP_LODmanager as result will be broken.

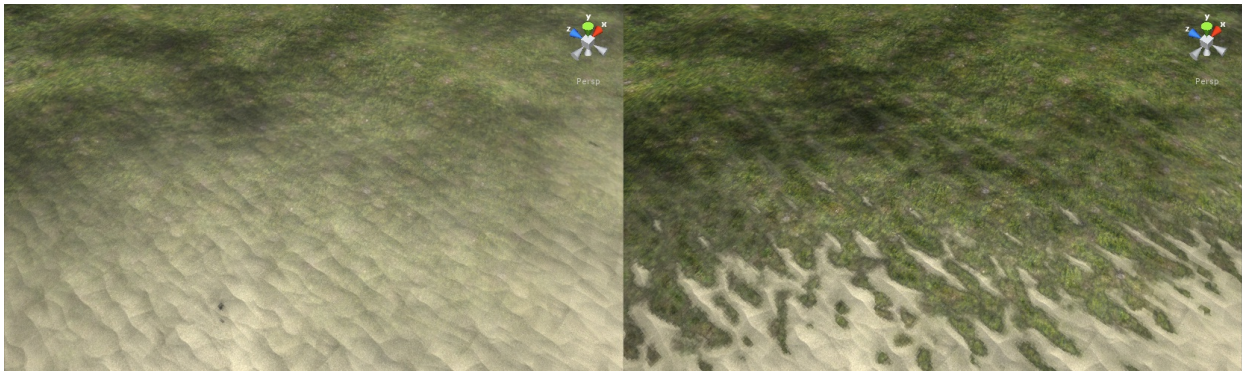
Heightblending

One of the main RTP principles is new layer blending method called heightblending. Classically amount of layer color used on terrain depends only on splat control coverage texture and it's realized like "alpha". For example %50 of sand and %50 of grass results we see sand blended with grass which doesn't look much realistic. With height map blending we can command shader to decide where to show sand and where grass depending on their height maps which are set per layer. Now if these 2 textures are blended 50/50, the brightest height map wins, that means if you have a gray height map for your grass and a white height map for your rocks, only the rocks would show up as they are higher. Of course having just a plain grayscale value on the whole height map would be a bad idea in most cases. We need more variation here so some rocks, or sand patches pop over the other blended layer. The other nice advantage of heightblending is that you can now reduce the resolution of your splat control maps and still get nice transition between layers (well defined, not "alpha").

One might ask if heightblend is available in scope of separate pass only. When rendering 8 layers we can achieve height blending between any layer when we don't use *No overlap in 8 layers mode*. However – sometimes you'd like to use 2 passes, for example when more features are needed (those which use dedicated textures). Fortunately (new in RTP3) we can make heightblend between passes ! If you use 2 passes (4+4), you can check *Crosspass heightblend* in AddPass settings (RTP LOD manager):



Crosspass heightblend results are like this:



Layers 0-3 have grass and rocks, layers 4 -7 – sand. On the right screen we use Crosspass heightblend.

Note that crosspass heightblend isn't "free" as it also uses 4 texture samplers in AddPass.

Height maps are also used for achieving parallax effects which brings an illusion that terrain is not flat but "extruded" (this is how parallax mapping works – for more info refer to any web resource looking for parallax mapping or parallax occlusion mapping techniques description). When RTP is set to LOD level SIMPLE it doesn't use parallax effects, but still we've got heightblending.

Actual shaders used

You might find useful to know where shaders that are part of RTP are used (placed in *Assets/ReliefPack/Shaders/ReliefTerrain/*):

1. *ReliefTerrain-FirstPass.shader* uses *RTP_Base.cginc*. This is used for rendering first pass of terrain and RTP used on meshes. Can render 4 or 8 layers at once.

2. *ReliefTerrain-AddPass.shader* uses *RTP_AddBase.cginc*. This is used for rendering additional pass for terrain (only). We can use only 1 additional pass, thus max number of layers handled by RTP is 12 (8 in 1st pass plus 4 in 2nd pass).

3. *ReliefTerrain-FarOnly.shader* uses *RTP_Base.cginc* and depending on number of layers also *RTP_AddBase.cginc*. This shader is replacement for Unity's basemap shader and is considered for only far distance thus works faster.

4. *ReliefTerrain2Geometry.shader* uses the same as above. Used for materials used on RTP placed on mesh objects.

5. *Internal* folder stores shaders that are automatically put on auxiliary underlying mesh for geometry blend feature (refer to Geometry blend features).

6. *Bonus shaders* folder consists of shaders that can be used alone w/o RTP. These are for example shaders that handles water (the same as RTP uses on terrain) or snow. However for snow on objects that are on the scene w/o RTP you'll have to provide some global shader params for snow to work as snow shader is meant to be synced with snow properties (for example global snow strength) of whole scene. Most shaders included are "legacy" (RTP3.0) one. But we've got new ones stored in CustomizableShader1Layer(PBL).zip and CustomizableShader2Layer(PBL).zip. These are new to RTP3.1 and are highly configurable to get most of RTP features available for 1 or 2 layers (vertex color blended versions).

7. *GeomBlendCompliant* folder holds shaders that can be used with geometry blend. Also you'll find there an example shader with instructions how to simply make your custom shader "geom blend compliant" which means it then can be placed on objects that will be blended with underlying terrain w/o seams.
New RTP3.1 configurable shaders are packed in files named accordingly:
CustomizableShader1Layer(PBL)_geomblend.zip and CustomizableShader2Layer(PBL)_geomblend.zip
New RTP3.3 they use new custom inspectors that make them very easy to use, without necessity to configure anything in shader code.

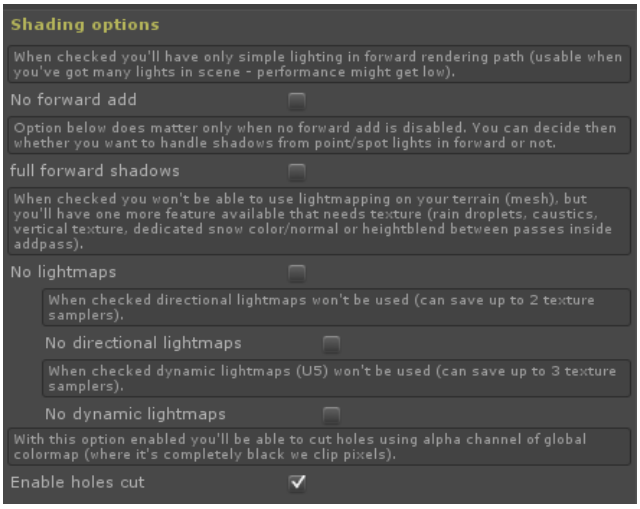
Shader mode, its material & multiple terrains

Most textures and settings in RTP are global for terrains placed under common parent game object. These are for example detail colors, bump maps, height maps. It's good to know how RTP handles textures that are – in opposite – terrain tile specific (not global). For example global color map (spanned over whole terrain tile). The most desired scenario is to setup your terrain with right number of layers then assigning *ReliefTerrain.cs*. You should know that RTP can "collaborate" closely with other high-end terrain solution available – Terrain Composer and World Composer made by Nathaniel Doldersum. When you setup your scene with TC or WC you can easily hit <RTP> button there and composers will take care about the rest.

RTP_LODmanager

This is basically an object that helps manage RTP shaders features. As in most cases we don't need them all at once (which is also impossible to use) manager helps to use only the required shader components. This results in shaders code tweaks and their recompilation. You could adjust options needed in *RTP_Base.cginc* and *RTP_AddBase.cginc* commenting/uncommenting feature #define keywords, but manager do it for you, so you don't need to have any coding experience to work with RTP. However manager doesn't take care about everything regarding shaders used in RTP package. Some of them needs adjustement in code which is only looking into code (at its top, right after properties of material) and finding the feature to be adjusted. Don't worry - there are comments in shaders that explains which define keyword is used for, so you won't have problems in tweaking them. The most "tweakable" shaders are standalone (standalone means - material only, w/o ReliefTerrain script) shaders that can be used on any mesh.

Main section of *RTP_LODmanager* checkboxes are shown below:



No forward add helps hold better performance in presence of more lights in forward rendering. This assures that only the most important light is used for rendering and we won't have additional passes for additional lights which will be then realized using spherical harmonics lighting (refer to Unity lighting docs). Note, that with this option checked point and spot lights will not work properly.

Full forward shadows - when *No forward add* is disabled, you can ask to handle shadows in forward for other light types (point/spot).

No lightmaps makes shader smaller and we won't have an option to use lightmapping. In reward we'll be able to use more options in shader that requires dedicated textures (like caustics, water dropletes, etc.).

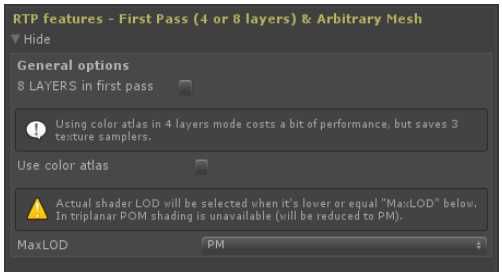
Enable holes cut means that you can paint holes in global color map section. It works the way black alpha in global colormap texture constitute holes, everything above black reduces parallax effects like before (in RTP3.0).

Below options of RTP LOD manager are set separately for First Pass and Add Pass.

RTP LOD manager can compile different set of features for First Pass (4/8 layers) and Add Pass (4 layers). You should always hold only features needed for given pass to get better performance.

General options section:

When 8 layers mode (described above) is not checked, we've got 3 additional options:



Use color atlas – new in RTP3.1 we can ask shader to get detail color from atlas rather than from original layer detail texture. As atlases are always used in 8 layers mode, in 4 layers mode we can also benefit – 3 additional texture slots will be available for additional features. This way in 4 layers mode we can turn on almost every RTP option and shader will still compile without exceeding texture resource limit. However, you should not use this option when you don't need it for sake of such limit. Atlas texture calls are always slower than regular one and atlasing is kind of "necessary evil" here.

UV blend / distance replacement section:

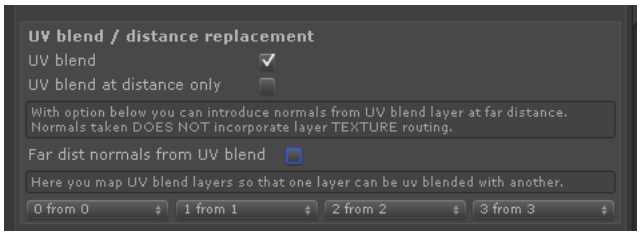


Fig 6a

UV blend – this feature helps hiding layer detail tiles. It works the way it use a layer color and mix it (multiplying) with blended layer at different scale. So blended layer can be spanned over more layer color tiles. Beside of color multiplication (new in RTP3.1) we can also gradually replace close distance detail color with its blended texture at far distance. The proof it works nice (together with globally applied perlin normal map) might be below screenshots:

UV blend at distance only – works faster, but applies (gradually via distance) UV blending only far from camera.

Far dist normals from UV blend – by default RTP doesn't sample layer detail normalmaps at far distance. The only bumpmap detail you can get relies on global normalmap (or global color map when lighting is baked there) and perlin normalmaps. In RTP3.1 detail normalmap sampling has been introduced as an option. They are sampled at UV blend scale (matching UV blend for color).

These dropdown boxes below Far dist normals from UV blend option are meant to serve as "layer router". Any layer can use any other layer (from its pass) for uv blend. In *figure 6a* above we see that all layers are blended with themselves. You will find such setup natural, esp. when you need to use replace color mixing. Grass texture would look weird when fully replaced with rocks texture at distance. So you could ask – what we need routing for ?
During production, you'll often find some textures doesn't serve nice for UV blend – flat sand detail texture won't introduce any additional variation when zoomed in. Magnified "big rocks" texture might also look weird (we perceive then very big rocks at distance instead of uniform, but varied rock areas). In case of RTP terrain example scene and textures included I'd personally say that 4th layer texture – Cliff (Grassy)_Diffuse_Spec – works very nice for UV blend purpose, although w/o strong replacement factors that would make all our terrain looks like 4th layer magnified and put all over terrain.

The other reason for routing is performance factor. It's desirable to use low number of routed layers. For example 0 from 3, 1 from 3 , 2 from 3 , 3 from 3. In such case this would be just one – 4th layer (counted from 0 so it's named 3) and all 4 layers would be blended with 4th layer. This can save some GPU performance by optimizing texture lookups in shader. As every layer would use one texture for UV blending (4th), shader would refer to just one texture when fetching it from GPU RAM. It's probable that this would be already present in GPU cache thus would work faster.

Global maps features section:

Fig 6b

No detail colors at far distance – as it says we we'll use layer detail colors only at close distance (what does it mean "close" will be explained later in section describing Settings/main settings). Note that in this mode only global color map will be used far so we won't have such freedom in setting its blending values (refer to Settings/Global maps – global color map settings). This option works well for large scenes with many terrain tiles and helps save some GPU power needed.

Global color blend multiplicative – this is default. If unchecked global color map will be blended classical way (replacing detail colors depending on blend value). Remember that using such option can cause that you won't see your detail layer textures when blending values will be put high.

Advanced color map blending – this option allows to blend global colormap with extra control over where and how it's applied. It's based on detail heightmaps. So you can ask shader to mix detail texture with global colormap only where it's low, or ask shader to colorize pebbles that are punched higher in detail heightmap. You'll find it possible to fully control the blending process – height thresholds, brightness and saturation values – all can be set separately for each layer.

Global normal map – instead of using normal from mesh this option allows using your own normal map. For current RTP users this might be confusing because now it's "true" global normal map – spanned over whole terrain tile. In RTP2 there was a feature named global normal but now it's "perlin normal" which fits best the usage. In RTP3 global normal map means texture used for whole terrain while perlin normal map is used tiled relatively to color detail tiles. Global normal map allows to get detailed look of terrain with realtime lighting while actual terrain mesh might be simple (high pixelError value in Unity terrain settings).

Global trees map – feature to use dedicated texture at far distance that will take precedence over global color map. Where global trees map pixel isn't black (RGB near zero) it replaces global color map. It's intended to put "pixel trees" far from camera where billboard trees reduces to size close to a few pixels (so needn't be rendered) but trees still should be visible. Alpha channel of the texture is used to show baked shadows. This way we can get rid of terrain lightmapping but use baked shadows. Terrain Composer will have such an option to generate/handle this texture. For more info – contact TC Author (Nathaniel Doldersum – eagle555 on Unity's forum).

Snow features section:

Fig 6c

Dynamic snow – enables dynamic snow features. You can set a layer that would serve for snow color (instead of constant snow color set in Settings/Snow) and dedicated layer which normal map will serve as normal map used for deep snow.

Water & Caustics features section:

Fig 6d

Water/wetness – water that's placed globally on the terrain. Use this when you intend to handle water on larger parts (or whole) terrain. If you need a few patches of water here and there (like streams) it's better to use geom blend option (water shader) and stick them to terrain. When simple water is not checked you can use animated droplets on areas where we've got water.
Caustics – useful for scenes with "more water" like ocean, lake. We can command RTP shader to render caustics below absolute world height (y coordinate). It uses caustics texture which has been prepared for you in the package. In bonus shaders folder you'll find shader that uses the same caustics algorithm as RTP terrain shader (bumped specular & caustics). Use it for meshes that are put underwater like fishes and so – they'll receive caustics without necessity to solve it using projectors.
Vertical texture from caustics – this checkbox will be available only when you turn on Caustics and Vertical texture map (described below) together. As caustics are monochromatic in most cases this is a way to save one texture sampler resource for shader compilation. Vertical map and caustics will share the same texture (look at TriplanarVerticalTexture+Caustics.png file placed in ReliefTerrain/Textures – RGB stores vertical map info while A channel is caustics mask).

Additional features section:

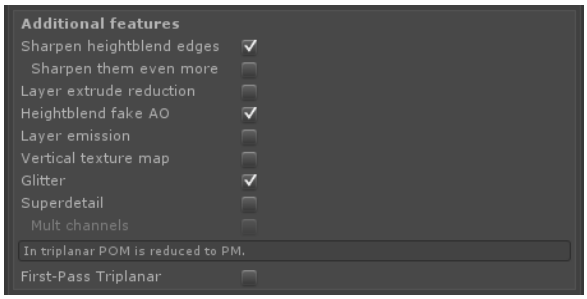


Fig 6f

Sharpen heightblend edges – check this when you'd like to have more pronounced edges between different detail layers blended basing on their detail height maps.

Layer extrude reduction – extrude reduction is basically doing the same thing as you can achieve

manipulating the brightness of your detail height maps. It ramps the dark parts of you height maps brightening them. When you use you extrude reduction value in layer setting maximum you'll cancel parallax effect for this texture and height map will bet treat as white. You can use layer reduction to see what effect brightening your height map would have. If you know more about the system you can live perfectly without extrude reduction because you know how to setup your height maps correctly. Its still good to have this feature to "debug" your height maps. In the end if you are satisfied with your heighmaps and are able to leave the extrude reduction on all layers at 0 you should disable the feature in the manager to save some performance.

Heightblend fake AO – edges where one detail layer meets another can be darkened in RTP. Amount of such outlining can be controlled globally here - Heightblend – AO – and per layer.

Layer emission – you can ask RTP to render layers either normal or emissive way. When we use emissive feature and set a layer (by per layer slider) to be emissive then A channel of detail texture is not considered to be glossmap any loner, but will be considered as emission mask. Emissive features affects also water/wetness rendering (fluids on surface can be then set to be emissive, too). When the option is used, you'll be also able to choose hot air refraction feature which is nothing more than simulating animated refraction effect on the emissive parts (think of lava flowing that might be hot so we can see animated “mirrage” effect).

Vertical texture map – enables vertical texture mapping. This is intended, but not limited, to make terrain slopes more "geologically interesting".

Glitter – introduced in RTP3.3 in place of deprecated Reflections – it works the same as in UBER shaders and is esp. useful for snow.

Superdetail – this option put additional perlin noise normal map at very close distance. This helps push more detail look at very close distance. Esp. for layers that are rough like rocks.

Mult channels (works with superdetail) – allows to mix up 2 additional grayscale textures. It is intended to put even more variations close on the terrain via color multiplication. This option will described in detail later on.

First (Add) pass Triplanar – enables hi-end triplanar texturing. This will resolve your mountain slopes texture stretching. Triplanar works only when rendering 4 layers in first pass (for add pass we can't render more) and can use shader LOD level up to PM (no POM shading with triplanar). Although quite well optimized it might be considered as one of more complex options in terms of performance. If you don't use triplanar on large areas, but need to resolve stretching only on some parts of your scene it's better to use geom blend features for this (described below). Also - using triplanar, don't set it for both first and add pass as your stretched areas can be probably managed to use only 4 layers. This saves performance. It's unlikely a case that all your layers need to be unstretched, for example sand or grass appear mostly on flat parts of a terrain. Place them on set of layers (that belong either to first or add pass) that don't use triplanar texturing. In RTP3.1 we can ask shader to use triplanar in 8 layers mode, but such texturing will be applied to layers 1-4 only, and not for parts where layers 1-4 and 5-8 overlap.

4. All settings explained one by one

Settings – Main

Tiling, MIP adjust & lighting

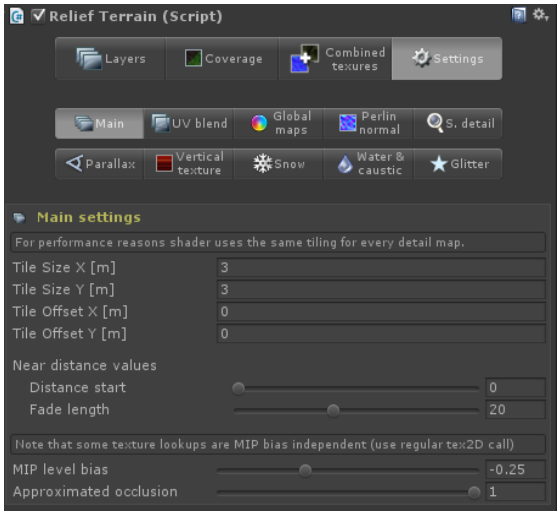


Fig 9

Before you actually play with your terrain it's good to look into main settings. This is where you define size and offset of your detail (layer) tiles like in Unity terrain script, but in RTP values are shared across every layer. Very important are *Near distance values* which define how far shader will render everything full detailed. RTP shaders use conditional branching and decide when render at full quality and where not. As parallax effects or detail bump maps are not distinct far from camera we need not render them there. At *Distance start* RTP starts fading from close distance. At *Distance start+Fade length* we don't have parallax effects nor bumpmapping. In case of other detailed options like water we won't see water flow there, too. *MIP level bias* is helpful when we want to adjust textures crispness. Default value of -0.2 works nice (close distance have quite good detailed look). *Approximated occlusion* is ambient/reflections light realted. RTP doesn't have seaprate textures for occlusion, but it can be approximated from detail heightmaps and normalmaps. It's very quick formula, but works pretty fine in most cases (heavily sloped parts of detail normalmap are occluded when they are low on the detail heightmaps).

When you setup multiple terrain tiles hold your detail Tile Size at value that fit evenly on whole terrain. For example, terrain tile which is 600x600m might use detail tile size of 3x3m. You'll have exactly 200 tiles spanned over terrain (it's shown in the brackets next to tile size label).

Shading color adjustements section of main settings is like this:

Fig 10a

Here you can tweak a bit the global hue and brightness of your terrain object independent from light setup of your scene. To take more control over color coming from detail color of layers, we can adjust here its brightness and saturation globally.

Gloss for deferred AddPass – this slider demands more explanation. It refers to deferred lighting only

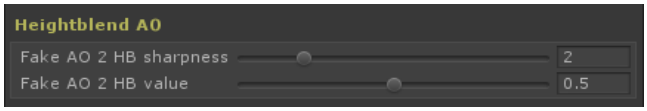
and direct PBL (physically based lighting) specularity. IBL specularity is not influenced by the issue. As already described terrain shading is often split into separate passes to render subsequent sets of layers (1-4, 5-8 in 4 layers mode, or 1-8, 9-12 in 8 layers mode). Additional pass is blended as decal. The same for geom blend features (described later). Unfortunately we can't use specular decals in deferred, because the same screen space buffer A channel is used for both – storing glossiness and amount of blending applied. That's why alpha blended decals in deferred have not their own, independent glossiness (shininess in terms of classical Unity shaders naming), but will "inherit" glossiness from underlying opaque geometry. For example, if we render glossy surface and then we put decal on top of it, no matter how hard we try to control decal glossiness (shininess), it will be always glossy the same amount as underlying mesh rendered first. And the opposite, if underlying opaque surface will be dull (rough, not glossy), we won't be able to make decal shiny. The only thing we can do to control this is damping its specularity strength (can be set per layer). This way decals will be just dull. You could use textures that represent diffuse (rough) materials for AddPass. If you need direct light PBL specularity there, this can only be constant value all over AddPass areas. For example in 4+4 setup, only first 4 layers in deferred will have glossiness controlled independently. Layers 5-8 (and gradually areas where they overlaps with first 4 layers) will have constant glossiness set by the slider explained here.

In deferred lighting path we can't control AddPass glossiness nor materials using geom blend compliant shaders via their glossmaps (A channel of detail color textures), because they are realized as alpha blended decals. It's better to use 8 layers mode in deferred when we need 8 layers (not 4+4 setup). Then we'll have full control over glossiness of all layers because they are realized in 1 pass. If we dramatically need triplanar shading, it's possible in RTP3.1 even in 8 layers mode, but only for first 4 layers.

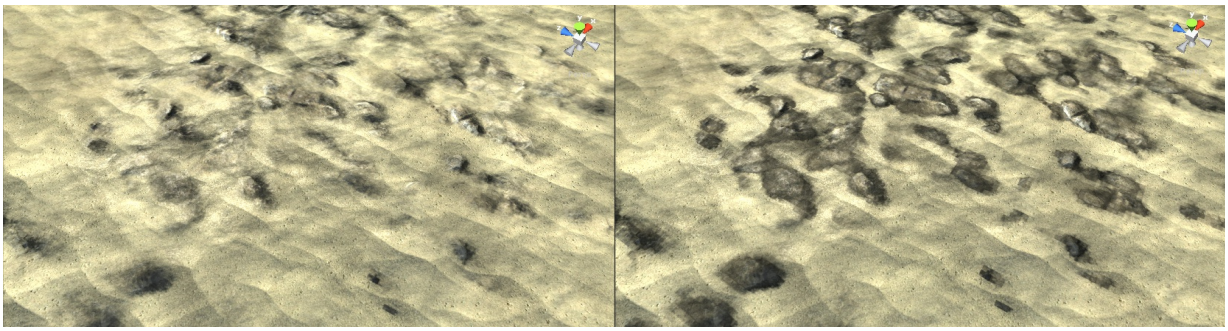
Using 8 layers mode we can assure all is rendered in one pass (no decal blending between layers 1-4 and 5-8), but it still doesn't resolve glossiness for geom blend objects (they are decals, too). They will also use constant glossiness set per object. Constant glossiness means the same size of specular highlight lobe, but its power can be masked, so it doesn't actually have to be the same shiny all over surface. That's how it's now realized in BlinnPhong lighting in Unity shaders (they set constant shininess/glossiness for material, but its strength can be modulated by spec masks). Conclusion is that decals in deferred reduces to classical shading models which don't apply to physically based multi-material paradigm (where some areas can be shiny while some can be dull depending on roughness map - in RTP stored in A channel of detail color texture). As we see deferred can introduce its own drawbacks and it's not true in all situations deferred is just "better". Forward lighting pass is superior to deferred in case of PBL lighting and using decals. Many modern titles uses forward rendering, but uses only one realtime directional light (objects are rendered once then). Rest lighting is baked in lightmaps and light probes (simple ones or high-resolution IBL cubemaps spread around an environment).

Did you know deferred and forward rendering paths are often used both even for single camera setup ? It's possible when using deferred for camera rendering path. Materials that can not be rendered using deferred are rendered in forward after all deferred rendering is finished. Natural example are transparent materials. They are rendered using forward. We can also force any surface shader to render as forward by adding **exclude_path:deferred** directive to the #pragma sufrace... line in shader code. This might be the way to render terrain or geom blend shader using forward even if all scene is rendered in deferred. Sometimes it's helpful to remove forward variants from compiled shader, because of texture resources beening exceeded. Use deferred rendering and add **exclude_path:forward** to be able to compile shaders.

Heightblend - AO – although this is named AO, but what this option do is tracing where heightblended layers have transition between each other and darkening these areas.



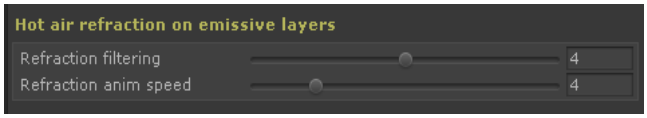
Below is how it works, however right screenshot uses a bit exaggerated values (sharpness=1,value=1)



Transition between heightblended sand and stones layers is outlined with fake AO on the right screenshot.

Amount of outline application can be also set per layer.

Hot air refraction on emissive layers – available only when emissive features are enabled in LOD manager:



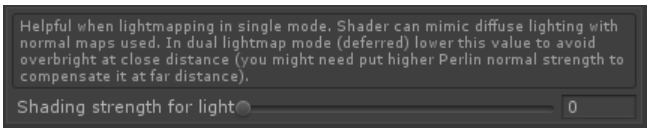
We'll distort texture offsets by animated noise taken from perlin normalmap (2 perlin textures superimposed and offset thru time variable).

Refraction filtering – perlin texture MIP level taken for animation (higher means smoother distorition).
Refraction anim speed – higher value means quicker distortion movements. Depends also on filtering – with high filter values you'll probably need to enlarge speed slider accordingly.

Refraction animation amount is also influenced by layer emissiveness, so look at further section for more info.

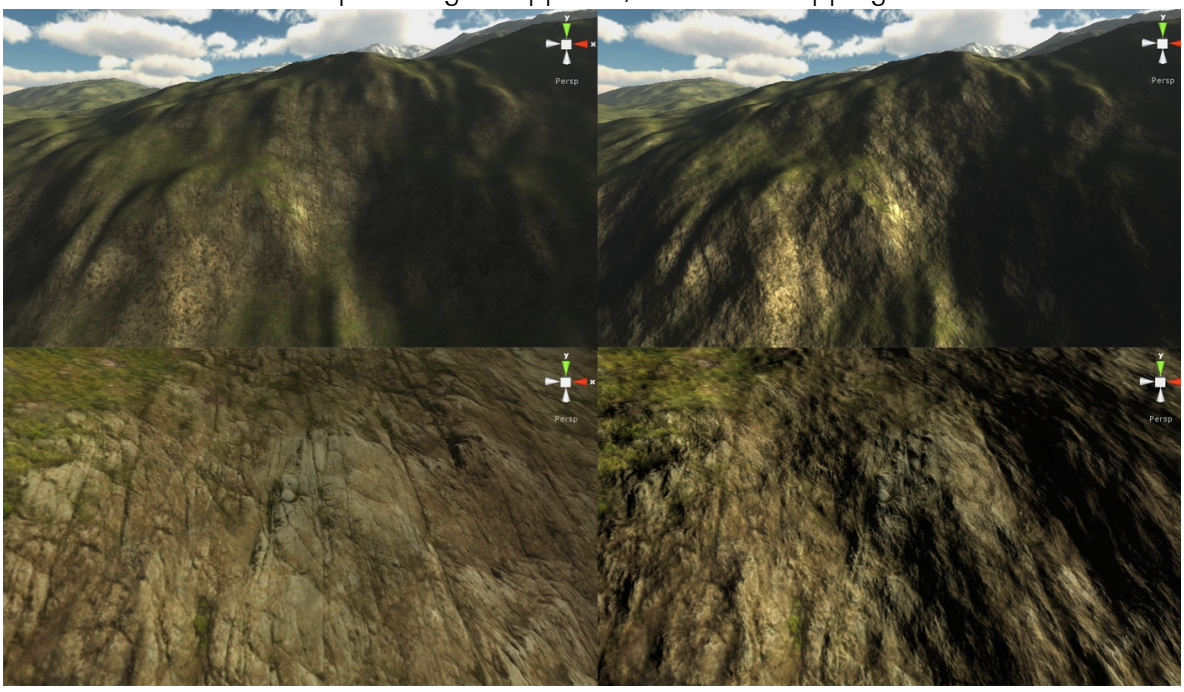
Lightmapping your terrain

When you lightmap your scene there will be additional option available in main settings:



As you might know – using single lightmaps you loose normal mapping. Dual lightmaps will also damp perlin normal mapping at far distance for your RTP terrain. Directional lightmaps doesn't work (terrain lacks of tangents – you may however try to export terrain, import calculating tangents, lightmap this using directional lighmapping and use this lightmap for terrain). When using lightmapper RTP allows you to rebuild one diffuse light normal map info via above slider. Works best with single lightmaps.

Results could be like this – left part is lightmapped w/o normal mapping restoration:

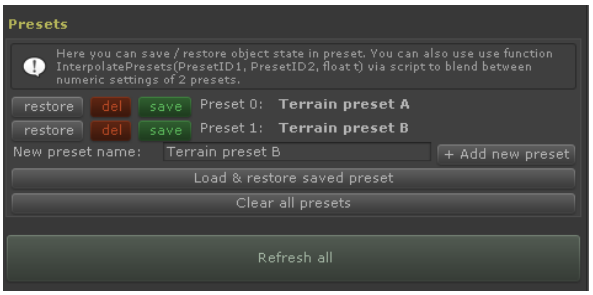


Lightmapped terrain at far and close perspective. Right screenshots show normal mapping restoration.

You should know that lightmapping is not the only one technique that might be considered. Maybe

even not prefered as Unity terrain lightmaps are often very slow to compute. You can use RTP global normal map feature to gain detail at far distance holding high pixelError (reducing rendered number of triangles) or global color map with baked lighting info. Both global maps can give similar results with advantage of specular lighting (as lightmap normal restoration is 1 light diffuse only). More bumpiness & detail at far distance is also well achieved by using perlin normal described later.
If you'd also like to bake shadows info for far distance that would allow you shortening realtime shadow distance for better performance use another option available.

Presets



Presets are meant to help artists trying different setups on terrain and saving them. A preset is defined by its name (don't need to be unique, but it's strongly recommended). Type this into *New preset name* text field and click at *+Add new preset*. Then you'll be able to restore to this preset settings at any moment (esp. desired when you go too far playing with params). If you'd like to share preset between different scenes, it's possible when you save them to .asset files using *save* and *Load & restore save preset* buttons. Remember, that presets sharing will work for terrains that has compatible configuartion, esp. number of layers used. You cant apply preset that stores different number of layers comparing to the terrain you'd like to load and apply it to.

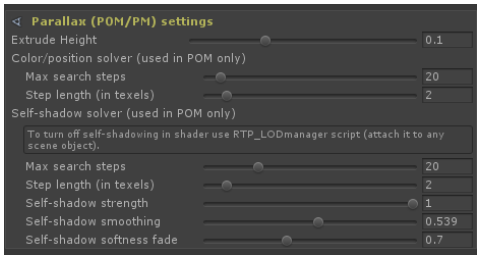
Presets can be considered to be kind of "RTP prefabs" that stores all parameters and references. That's important to keep in mind that preset file might loose texture reference to those existing as scene objects only. Save textures used by RTP to disk before you save preset to make sure they will be correctly restored.

The second important application for presets is that we've got the option to interpolate RTP settings between two presets. Let's say you've got 2 preset setups for terrain – one for winter and one for summer, or for different weather conditions. If you'd look inside *_appControlerTerrain.cs* script that is used for example scene (placed on camera game object) you can go to lines 161-173 which are commented. Uncommenting them will allow you to test how preset interpolation works via slider. Analysing this code isn't difficult so custom implementation interpolation should be easy for anybody with basic coding experience. *ReliefTerrain* object instance (let's say variable named *script*) has function that retrieve preset by its name – *script.GetPresetByName("preset name...")*. When you've got 2 presets (stored in variables named *holderA* and *holderB*) you can call:
script.InterpolatePresets(holderA.PresetID, holderB.PresetID, t);
where t is value from 0..1 like in classical Lerp().
The last thing is calling *script.globalSettingsHolder.Refresh()* which will apply changes (injecting params from scripts to shaders). As you might expect interpolation is realized only on values that are either numerical (for example global snow strength) or color (for example ambient color correction). We can not interpolate textures.

At the end of Settings – main section you'll find a big green button - **Reset all** – which applies needed params to every object on the scene that has ReliefTerrain script component. Might be useful when your scene consists of quite many terrain tiles and you'd like to refresh them in case then don't look like they should after global changes (for example scene that is setup via Terrain Composer).

Additional buttons that are present at the bottom of ReliefTerrain inspector are for reseting values to their defaults. It's useful when you overplayed setting and try to restore reasonable look of your terrain.

Settings - Parallax



This settings are applied only in shader LOD levels POM and PM.

Extrude Height – this is main setting that controls parallax extrusion level. In case of PM (parallax mapping) setting this too high might result in artifacts so try to not overuse it as results might be bad.

Color/position solver params - this is used only in POM (Parallax Occlusion Mapping). Unlike in PM (Parallax Mapping) we search for solution in multiple steps which number is not constant, but depends on angle at which we observe extruded surface. For small angles (view direction almost parallel to surface) we would need more steps and shader will make them smaller for better precision.

Max search steps defines maximum number of steps in which shader looks for solution. If still not found shader takes last tested position. This causes extrusion flattening at small angles with insufficient number of steps, but is often acceptable – it will just look like extruded surface bottom is "pushed up".

Step length is the smallest search step length taken. Actual step length depends on viewing angle like in steep parallax mapping technique. It is defined in texels of your height map. Thus height maps should be of smallest acceptable size because this texture is what we actually sample in shader when calculating POM (and we sample this *Max search steps* times in pessimistic case). Making height maps 1024x1024 doesn't make sense in most cases because POM calculations will need 4 times more steps (of the same texel length) needed to find solutions comparing to 256x256. RTP example scene uses 256x256 height maps and this looks fine despite of what you could expect from such low-res texture – just because we don't see height maps directly. We use them for parallax calculations and for heightblend only. If you'd like heightblend to be more "hard" and better defined you may, however, try to increase height maps resolution and the same time increase *Step length*, so POM calculations need the same number of steps to find solution. Anyway – remember that small height maps give more chance they can be placed whole in GPU cache (means faster access !).

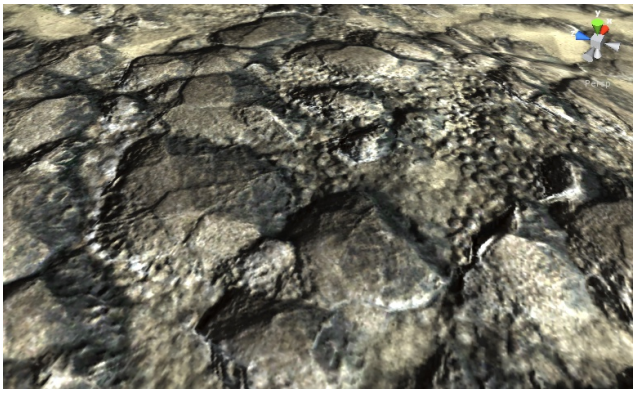
High step length means that POM solver will probably find the solution in most cases, but the solution might be imprecise (expect edges of your extruded bumps look like they're "sliced"). You should try to find right balance between number of steps (lower means faster but we could not find solution – flattening will occur) and their length (shorter means more precision, but solver may not find solution in max steps, in opposite – too long steps means artifacts) to get reasonable performance and quality.

Setting *Max search steps* to maximum and *Step length* to minimum will look good but will be also slow. And vice versa. Optimal solution in terms of performance vs. quality is placed somewhere in-between. Default values should work fine with 256x256 height maps.

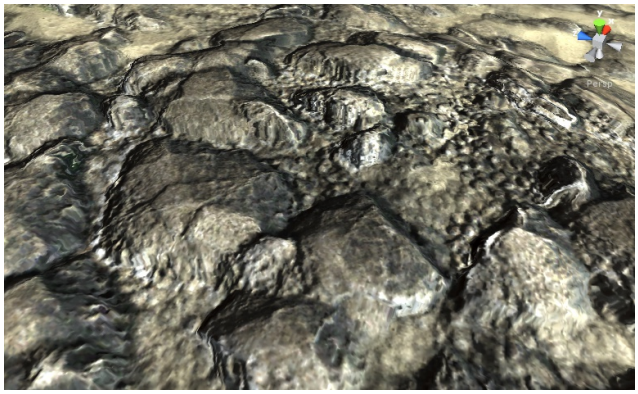
Take care about your mesh triangles size. For very big triangles parallax effects are often badly distorted because of wrong viewing vector interpolation. Make tris smaller to solve this.

Self-shadow strength is self-explained (how dark the self shadowed surface is).

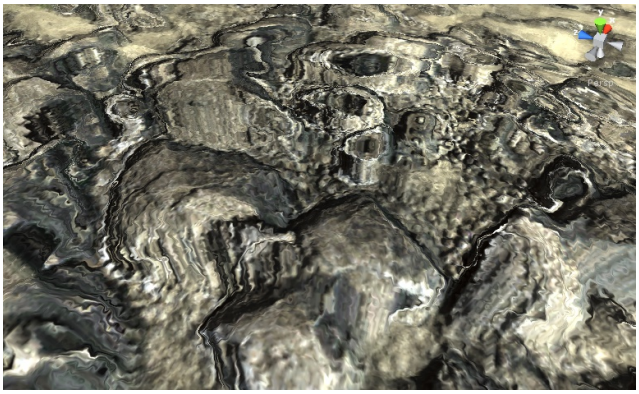
Below screenshots show difference between shading w/o parallax, with PM and with POM using different settings:



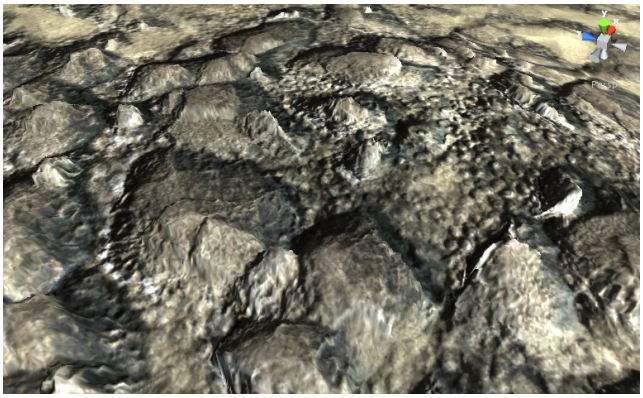
No Parallax



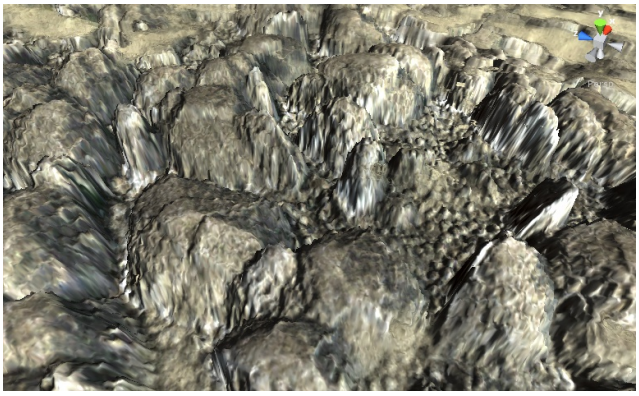
Parallax Mapping (PM) with moderate extrusion (0.06)



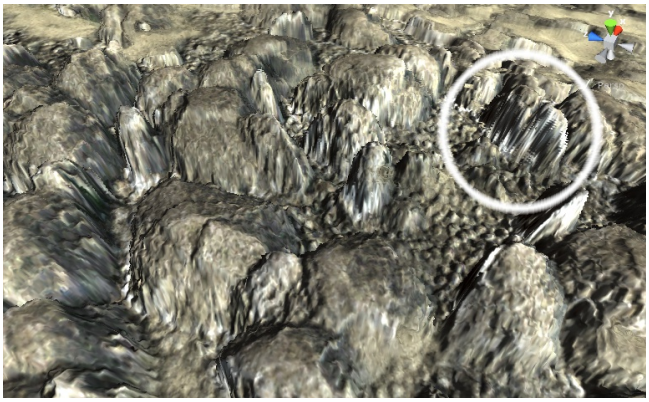
Parallax Mapping (PM) with 0.2 extrusion level – artifacts



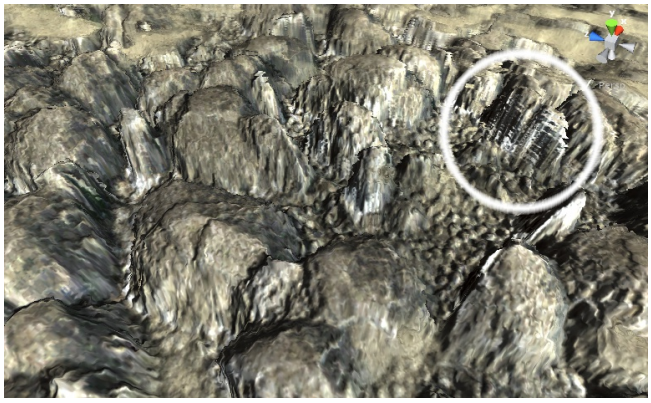
Parallax Occlusion Mapping (POM) extrusion level 0.16
max steps 10, step length 0.5
– solver can't reach the bottom (flattening)



max steps 120, step length 0.5
– full extrusion realized with high quality, but slow (too many steps needed)



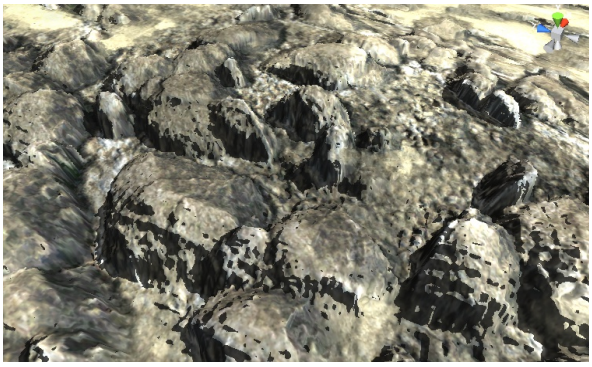
max steps 20, step length 2
reasonable quality vs. performance



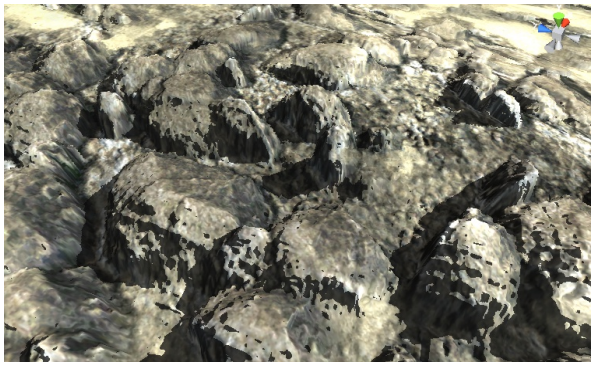
max steps 10, step length 5
although works faster, noticeable slicing artifacts

Self-shadow solver – used when shader LOD level is POM with self shadowing. Shader realizes the same raytracing like above (solving detail texture offset) but from different start point and in different direction. We search solution from the point where color solver stopped (the texture pixel we actually see) towards light position. If we hit something on the way – pixel is in shadow. Insufficient number of steps causes shadows "shortening" – if shadow solver can't reach the top it assumes that pixel is not shadowed. Like for parallax above you can compensate lower number of steps with higher *Step length*. Although this reduces detail of shadow silhouettes, higher step length is desirable as detailed height map sampling may result in tiny shadowed patches very distinct when shadow smoothing is off.
When smooth shadows are used you can decide about overlapping additional, 3 constant steps shader should perform to render soft shadows prenumbra. This is *Soft-shadows shadow smoothing*. *Self-shadow smoothing* does what's expected – reduces small shadow patches and makes shadow edges smoother.

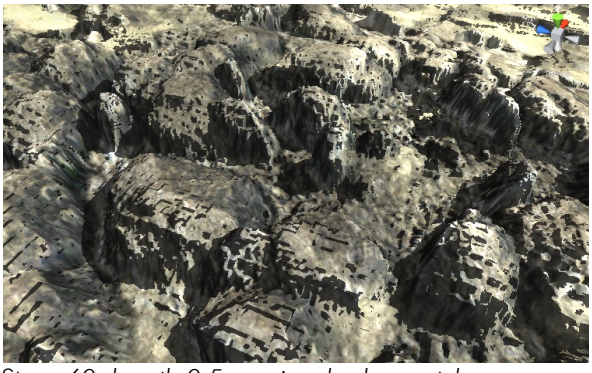
RTP calculates soft shadows only by approximation which is meant to trade imperfect quality for less GPU usage. Generally self shadowing is one of the most expensive shader features, so suggested usage is for hi-end GPUs only.



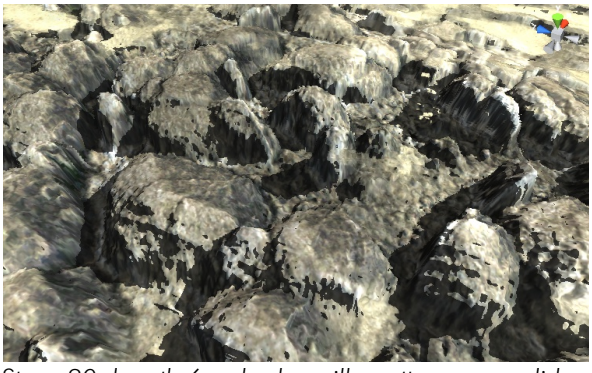
No smoothing, steps 4, length 4 – shortened shadows



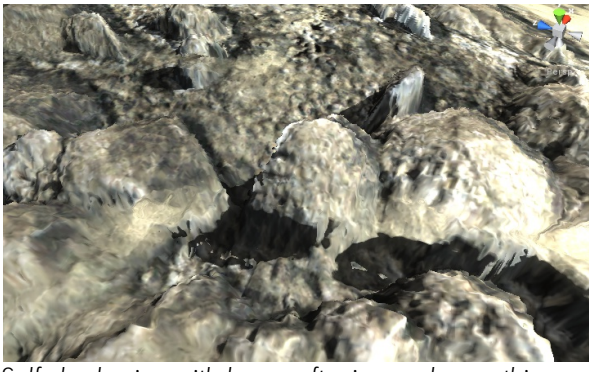
Steps 40, length 4 – shadows get longer



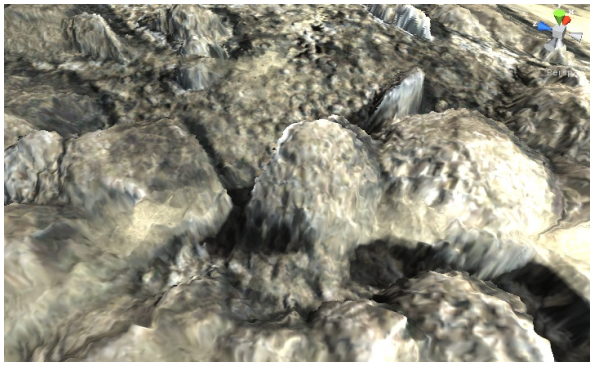
Steps 60, length 0.5 – noisy shadow patches



Steps 30, length 6 – shadow silhouettes more solid

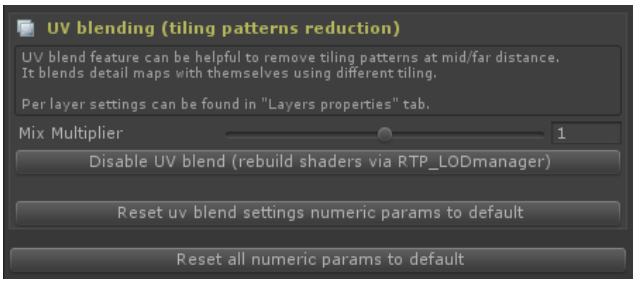


Self-shadowing with lowersoftening and smoothing



Self-shadowing with higher softening and smoothing

Settings – UV blend



Here you can globally control UV blend for all layers (*Mix Multiplier*). As for every feature that is optional in RTP you can enable/disable it either here (Disable UV blend button) or in RTP LOD manager.

To adjust UV blending for a distinct layer - choose Layers tab and click a layer thumbnail:

In this case chosen layer is sand and its UV blend settings are like this:

Fig 11

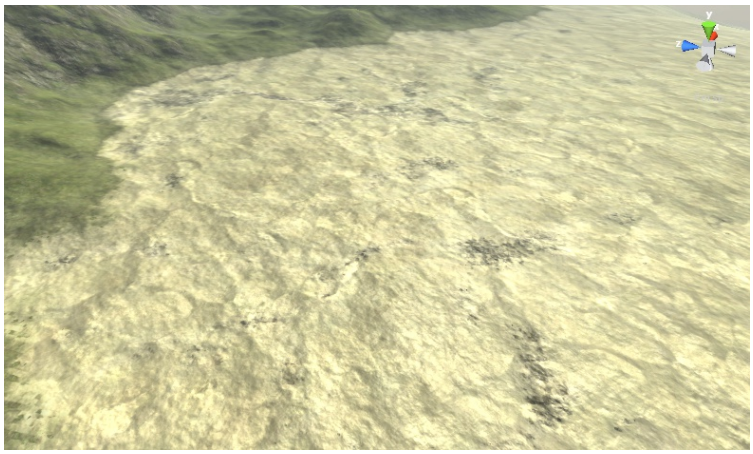
UV blend scale – set to 0.1 means that 1 UV blended tile is spanned over 10 detail tiles (1/0.1=10). Small A button helps aligning UV blend tiling so it fits evenly to whole terrain.

UV blend val – set to 0.3 means that blending strength is 30%.
As UV blend works via multiplication (detail texture color is multiplied with blended texture) with high blending value result might be oversaturated if both detail and blended textures have saturated colors. That's why we need UV blend saturation – set to 0.2 means that blended texture is first desaturated to 20%, then multiplied.

UV blend brightness – resultant color is calculated via multiplication and brightness is additional multiplier (detail color x UV blended color x brightness). By default it's 2. You can adjust its value when you find your blended results are too dark or bright.

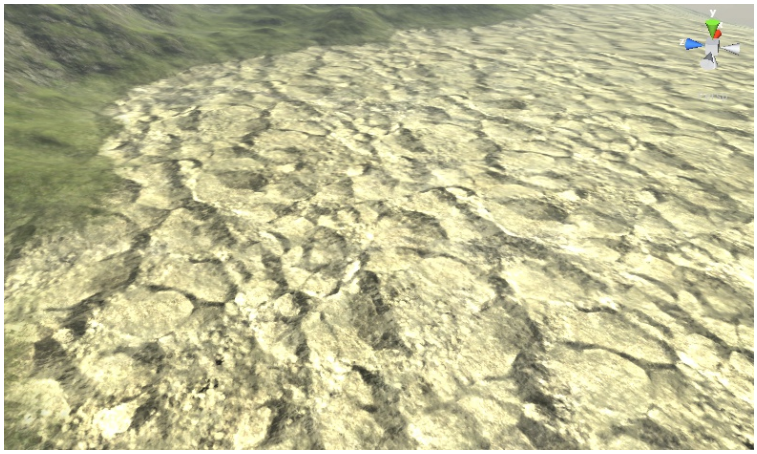
Far blend replace – we can command shader to make additional blending operation at far distance. While close to the camera UV blend works using multiplication, at far distance we can additionaly alpha blend UV blend texture replacing close distance multiplicative result. So – when replace slider is set to max. you'll see only UV blend texture at far distance. Use this with care when you route UV blend source layers. Overusing replacement feature might look weird when you completely replace grass layer with rocks at far distance. Anyway – applying just a little bit of replacement even for UV blend sources that differ from layer affected works fine and additionally remove tiling patterns.

UV blend multiplication results in such effect which introduces a bit of brightness variation over sand:



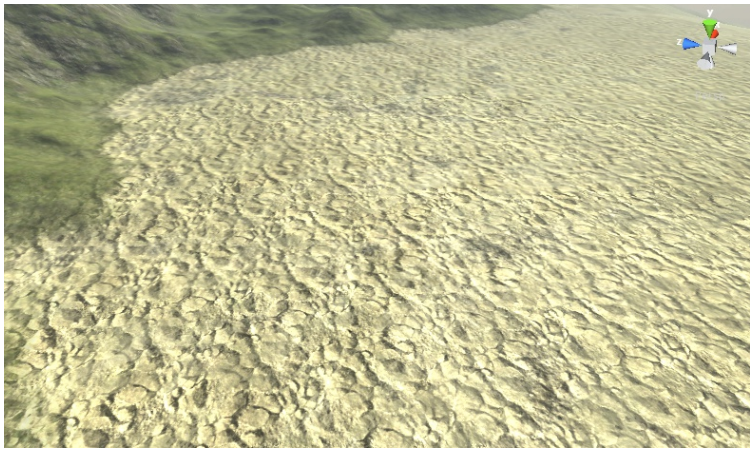
Blended texture from rocks layer gently visible over sand layer

Here's how it would look like for different params – scale 0.1, but exaggerated blend value 0.9:



Blended texture from rocks layer strongly visible over sand layer

And with different scale (0.4 – uv blend tile spanned over 2.5 detail tiles 1/0.4=2.5):



Blended texture from rocks layer with more tight tiling

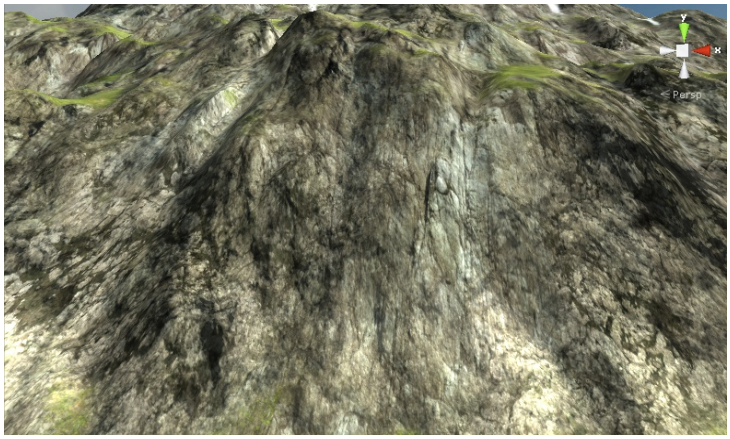
For sand untiling is not that crucial, but there are often detail textures that are a lot more prone to noticeable tiling, especially at farther distance. Here we've got slope, mixed with 2 types of textures – layers 3,4 in FirstPass:

Visible tiling on mountain slope, let's do something...

Let's set strong blending for these layers by setting max 0.9. Routing is set to (2 from 3), (3 from 3). This causes 3rd layer is UV blended with itself, 4th layer is UV blended with 3rd layer:

Strong UV blend on rocks looks quite fine in example scene

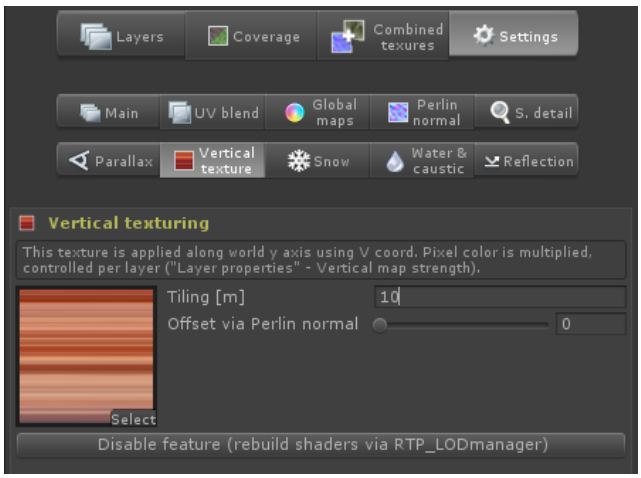
As we see tiling has been resolved using UV blend technique alone. If we want to benefit from replacement feature, here's how it looks like. Notice that we've got UV blend source visible only at far distance for rocks texture (I replaced both layers 3rd and 4th layers with themselves as UV blend routing has been set to 0 from 0, 1 from 1, 2 from 2, 3 from 3):



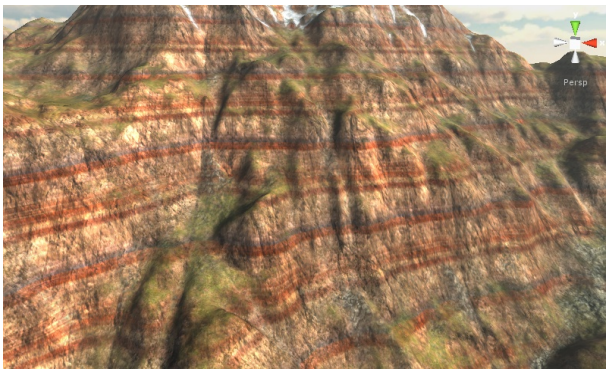
Rock layers replaced with them selves at far distance

When using high replacement values, keep higher UV blend value, too. This will ensure better, gradual transition from very close to far distance.

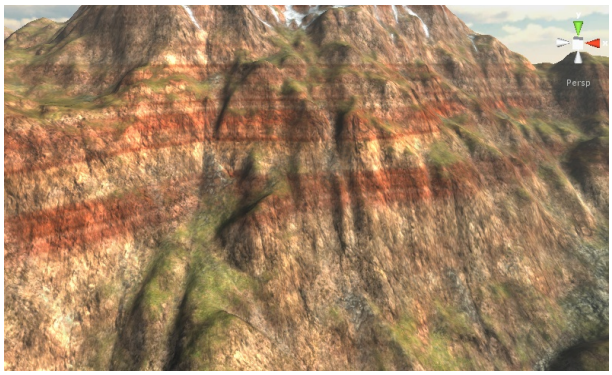
Settings – Vertical texture



Vertical texturing is meant to allow introduce more variation on slopes regardless of its steepness. It is blended via color multiplication. Generally the texture is sampled only at its first column, so it can be very small – 1xN pixels. Tiling set at 10 means that texture will be repeated in vertical axis (world Y axis) every 10 world space units.



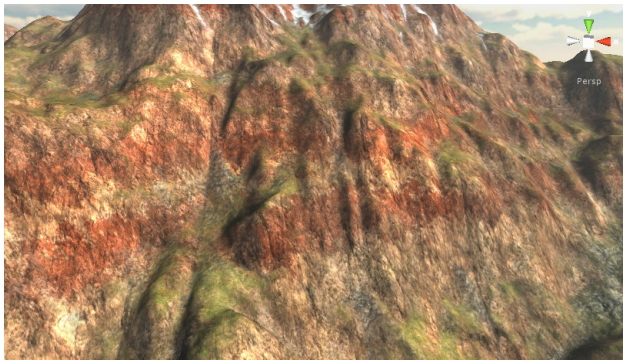
Vertical texturing – tiling 10 m, perlin offset 0



Vertical texturing – tiling 60 m, perlin offset 0

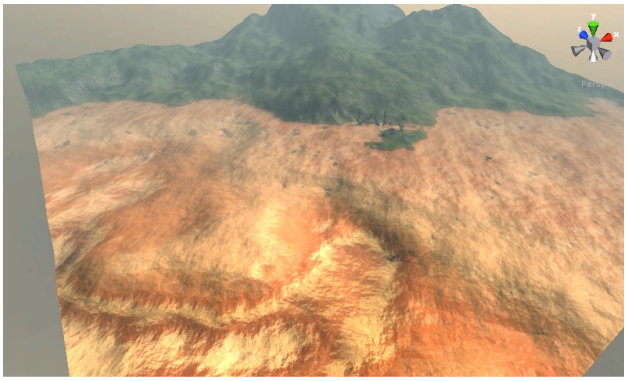
The feature is applied per layer (you'll find it in layers settings). For layers 3,4 (rocks on slopes) I set it 1 for above screenshots, while grass layer has this lower – thus we don't see stripes on grass.

Offset via Perlin normal is slider that allows to perturb uv coords when sampling vertical texture – this way you can get stripes that are not that straight:

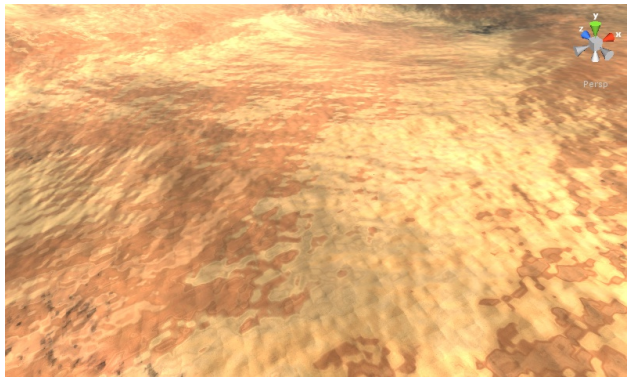


Vertical texturing – tiling 60 m, offset 0.1

Although it works great on slopes allowing to easily make "Grand Canyon" geological structures, no slope factor is taken into account by shader. With high offset perturbation values you can also introduce surprising variations on quite flat parts of your terrain, too.



Vertical texture on flat sand layer



Vertical texture on flat sand layer - close-up

On above screenshots we see how it might work, however texture is quite strong saturated resulting in heavy red colorization on sand.

Settings – Global maps

Global color map with holes feature

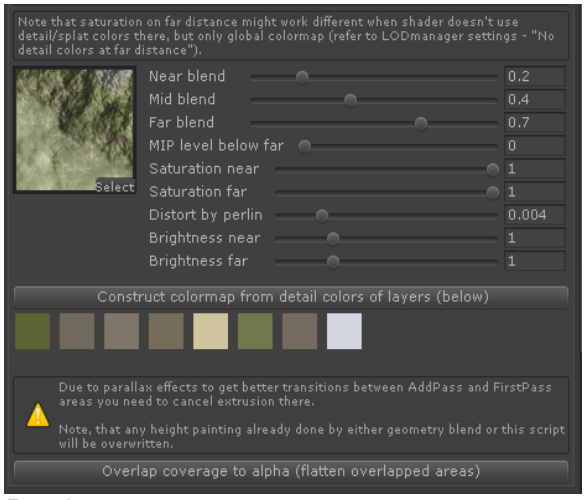


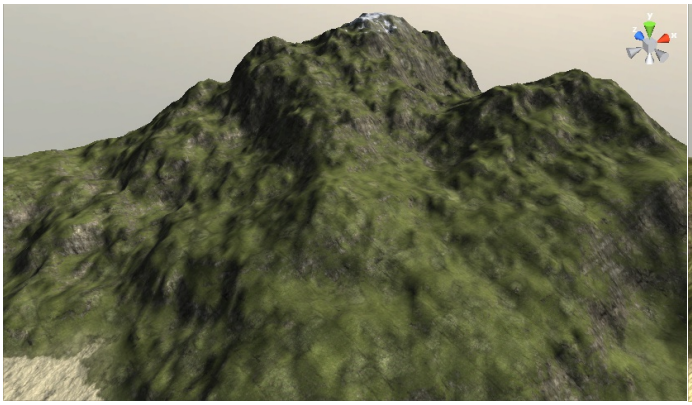
Fig 12

Global color map is texture spanned over whole terrain tile. By default its color is blended via multiplication with rest of color components (i.e. detail textures). Level of blending is controlled separately for different distances (*Near blend*, *Mid blend*, *Far blend*). Near distance is defined in Settings – Main. Far distance is defined in Settings – Perlin normal. Mid distance stays somewhere in-between near and far.
MIP level below far is the texture MIP level that will be taken for close distance. This is helpful when you'd like to low-pass filter global color maps taken from satellite images. Far away they might look very good on terrain and may constitute whole color seen – as referred to *this* option. Beware when we don't use layer detail colors at far distance *Far blend* has little effect. The same time global color map from satellite image might look bad when applied on near distance and taking only a coarse texture value (high MIP level) looks better.
As global color is blended via multiplication your terrain might get oversaturated – *Saturation near/far* sliders helps fixing the issue. The same when you'd like to adjust a bit its brightness – use *Brightness near/far*. *Distort by perlin* perturbs global color map uv coords using perlin normal map. It's useful to artificially enhance color map resolution. It works for shading LOD greater than SIMPLE.

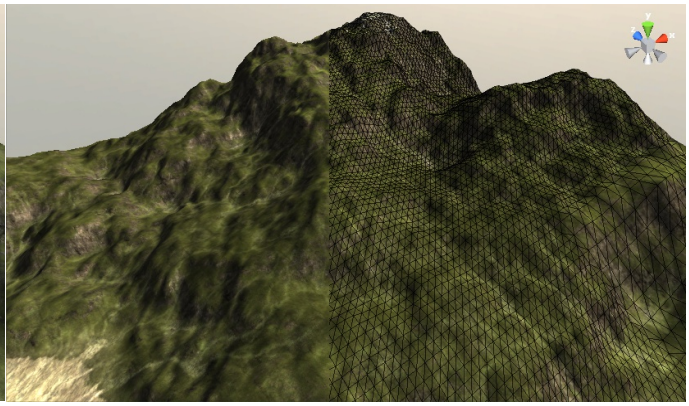
It's best to use global color map that reflects terrain conditions – good results can be achieved using for example WorldMachine. If you don't have it you can eventually compute color map basing on detail color coverage (*Construct colormap from detail colors* button). Each layer has its "average color" which is pixel color at the highest MIP level. These colors are reflected on thumbnails – figure 12.

Mixing together external color map with constructed texture which follow detail layers coverage is also an option (to be done in any image editor).

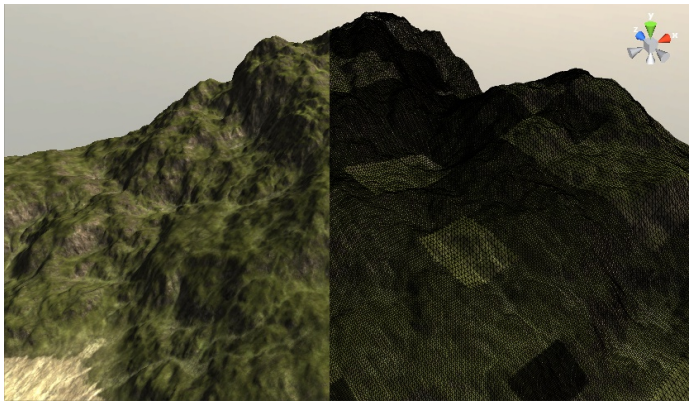
For static light conditions global color map may serve as lightmap and AO map which both can be baked into. To show the difference global color map can make let's look at below screenshots:



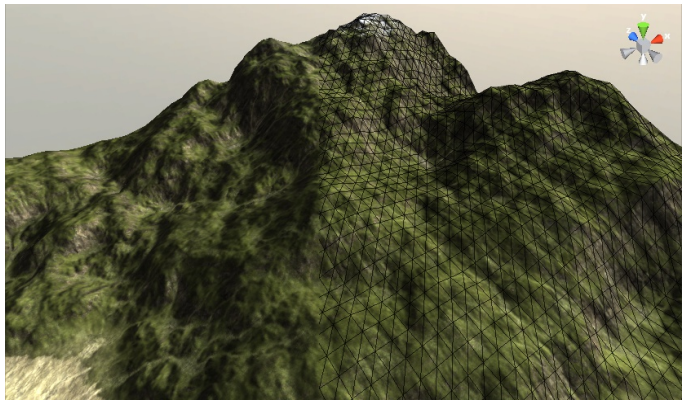
No global color map, pixelError 5



Global color map from World Machine, pixelError 5
Notice nice color variations due to WM flowmap

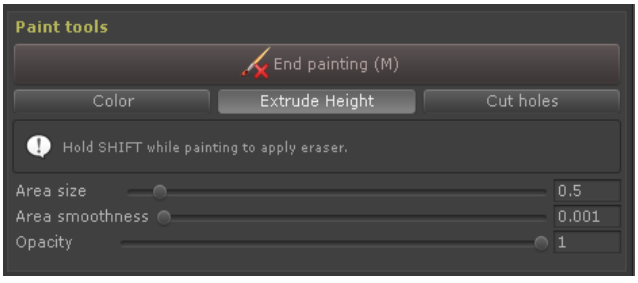


Color map from World Machine, pixelError 1 - we've got hi-res lighting, but mesh is unacceptably tessellated



Color map from World Machine with lighting info baked. PixelError is 10 so terrain is low-poly, but still quite detailed

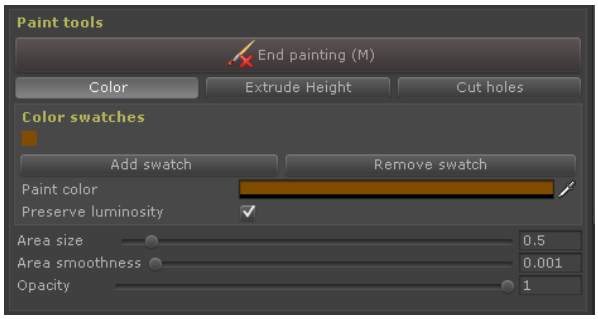
If your color map has been modified you'll have an option to save it on the disk as png file – *Colormap uncompressed & readable* – save to file button. It's the case when you construct it, or modify its alpha channel using *Overlap coverage to alpha* button (figure 12). If you use a texture compressed w/o alpha (DXT1 for example) it will be treated in shader as 1 (white alpha mask). In most cases you won't see any difference, but alpha channel of global color map plays important role in cancellation of parallax effects. It's necessary to make surface "flat" in terms of parallax extrusion height in some places because of artifacts you might experience where different passes overlap (in 4+4 or 8+4 layers shading setup) or at the edges your geom blended meshes intersect terrain (feature described later). When you find parallax looks weird you can put black on the alpha channel of your global color map to flatten extrusion. Finding where passes overlaps might be tedious between passes, so RTP can make this job for you using *Overlap coverage to alpha* button presented above (texture have to be present, it won't work when you have global color map empty). For geom blend you'll have to edit this by hand using painting tool:



Hitting green *Begin paint* button turns it into this red one shown above. You can also turn on/off painting pressing M. Option chosen is *Extrude Height* so we'll paint on alpha channel of global color map in scene view. Meaning of sliders that control paint size and opacity are obvious. *Area smoothness* is level of gradual opacity applied to painting brush at its edges.

Dark alpha channel of global color map may be the reason you can't get parallax shading working.

When painting - choose option *Color* (instead of *Extrude Height* above):



Now you can tweak colors of your global color map. When choosing a color needed you can store it inside swatch (in U3 we don't have it for built-in color picker). You can also choose color using detail average color values (figure 12).

Preserve luminosity is an option that will hold brightness of global color map while painting. It's good when you modify texture that has already lighting info baked, so that we won't affect darker, shadowed parts.

Global color map value is controlled per layer:

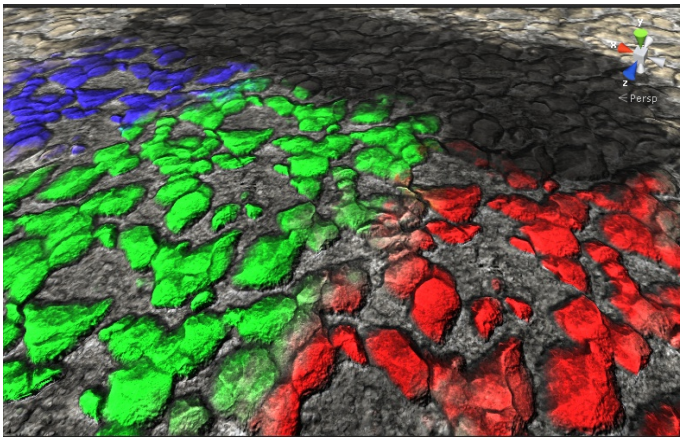
If you can't see any difference on your terrain check if above slider in layer settings isn't damped to 0.

Check your global color map so it's set to wrap mode **clamp** in its settings. You will experience seams between terrain tiles with wrap mode set to repeat.

When *No detail color at far distance shader setting* is used in LOD manager you've got no option but use global color map – otherwise you'll see gray (nothing) at far distance.

Simply a single slider (above) is present when you use basic global colormap blending functionality which is default. When you check Advanced color map blending LOD manager you've got many more options available:

Advanced color map blending is meant to apply independent blending values basing on detail heightmaps. For example we might need to affect only high or low parts of a detail texture - pebbles popping out from the soil or soil alone without affecting pebbles color. In an example inspector velues presented above I used test colormap with 3 color stripes (RGB) and black part at the top:



Advanced global color mp blending - only upper parts of detail heightmaps are blended with saturated values

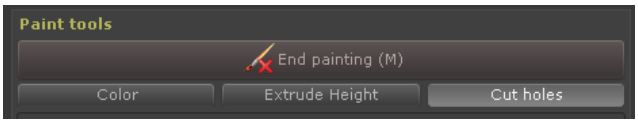
Height levels slider defines range where transition between parts considered to be low and high are placed. In our case everything below 1/3 is low. Everything above 2/3 is considered to be high. Range inside is transitional area. To make transition sharper set this area smaller.

As *colormap saturation LO* slider is zeroed – bottom parts of the detail texture is influenced by colormap (see – it's all dark at the top of screenshot), but desaturated. Only big rocks that are high parts of detail heightmap are colorized (*colormap saturation HI* slider set to 1).

When we blend detail color with color map we might also need to control saturation of detail as blended result might be oversaturated or tinted undesired way. Let's say we've got detail color green while color map is red. If we would like to make mixed result more red, we couldn't as detail color influences output. However, if we first desaturate detail color and then mix it with global color we get desired red tint (instead of dark yellow). So - sliders *layer saturation LO* and *layer saturation HI* are meant to control detail saturation in process of mixing with color map.

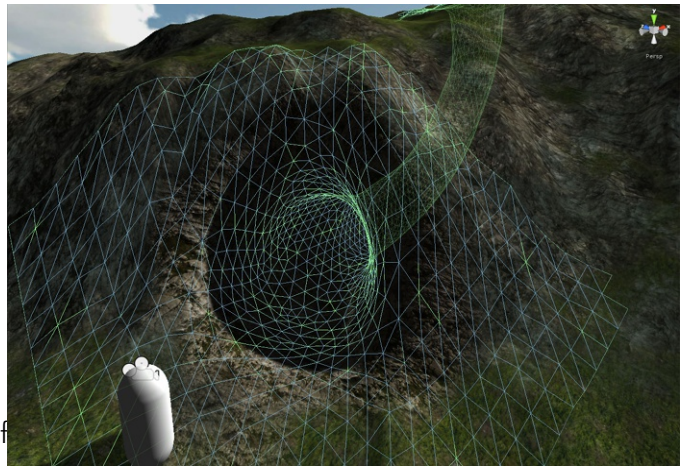
Blending for LO and *Blending for HI* control overall amount of blended result used – separately for low and high parts (we might need to apply color map blending only for one part).

Last painting option available here is *Cut holes* functionality (to be enabled in LOD manager), it has the same brush settings like in case of height extrusion.



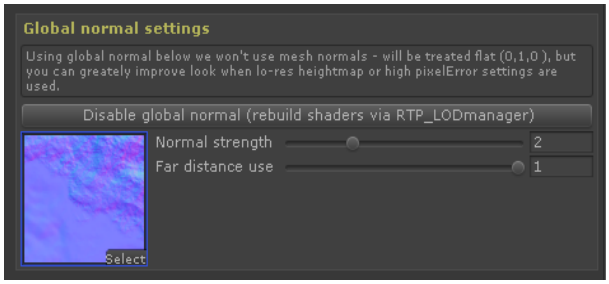
Hole cut in terrain(border fray due by color map distortion) *… and covered by geom blend mesh (100% seamless)*

When cut holes functionality is enabled we can just paint holes in terrain to place there caverns,



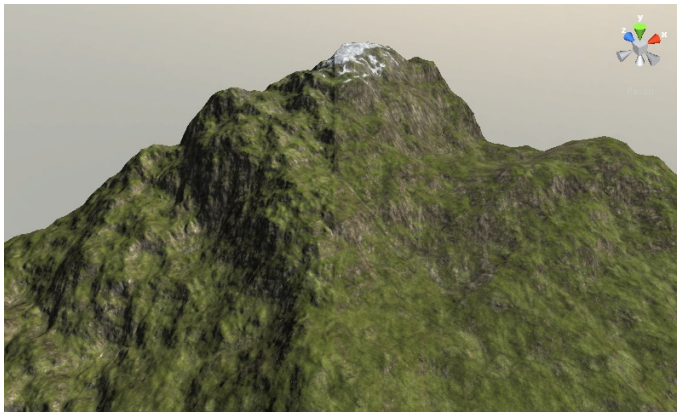
tunnels or large overhangs. The reason holes belongs to global color map is that they are just black values in alpha channel of color map (anything above zero reduces extrusion height only). Holes are placed using alpha cutout shading and are influenced by color map perlin distortion (reduce it to have not that frayed hole edges). While visual hole is one thing, enabling objects to pass thru holes is another issue. This is resolved by placing hole entrance triggers (simple collider like cube is fine) and applying *ResolveHoleCollision.cs* script (available also from ReliefTerrain component menu – helpers) to objects that need to pass thru. You define list of entrance triggers and terrains that are hollow. Additionally you can check *Start Below Ground Surface* option when your object is placed inside cave at start. *Check offset* is used (raycasting from object pivot downward) to determine whether object is inside cave (under terrain ground collider level) or not. To check how it works refer to example TerrainScene included.

Global normal map / tessellation height&normal map

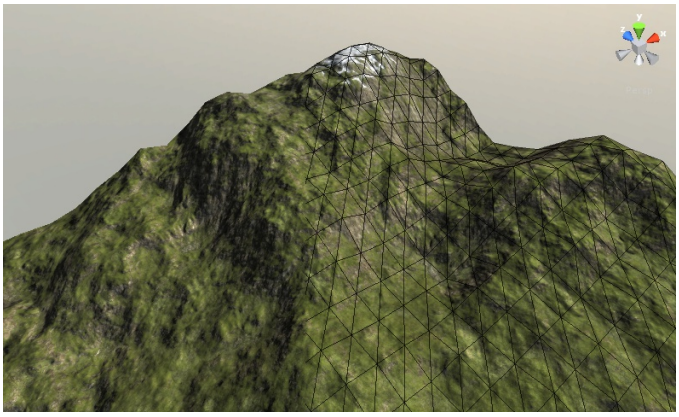


Option to be enabled in LOD manager. As described in inspector hint box your terrain mesh normals won't be used, but taken from the normal map. It's good when you'd like to use lower resolution of the mesh (high PixelError), w/o compromising lighting detail, esp. at far distance. Global normal map can be produced by 3rd party tools, eventually you can use terrain height map for this and make normal map out of this. We can adjust *Normal strength* via slider shown above.

And here's how does it work:



Left side of terrain has applied global normal map with strength set to 2, right side uses 0.5 (almost none)



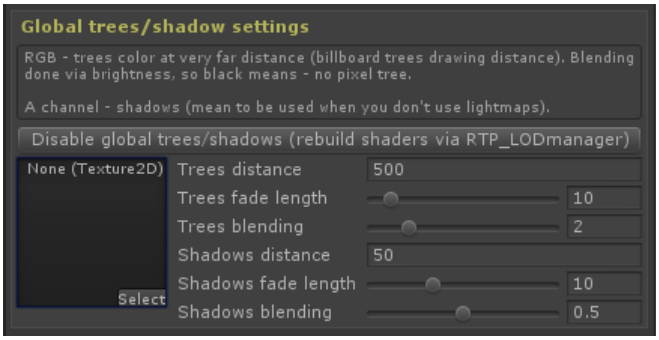
pixelError=30 - very low-poly mesh but we still have far distance terrain shadowed very detailed (1k normal map)

Note that unlike global color map can have – global normal map doesn't operate on static lighting, but can serve for dynamic light directions.

When your terrain mesh fits exactly global normal map slope values (in world space), you'll probably use value of 1 for *Normal strength*. When you are not sure the value is right or you just want to use original terrain mesh normals at close distance – turn *Far distance use* slider to max. When you come close to the terrain and the slider is turned up, close area normals will be used from mesh, not from global normal map. This way you can also "calibrate" *Normal strength* value. Adjust it until you minimize distinct difference in lighting at close distance while playing with *Far distance use* slider.

RTP3.2 – here we've got tessellation height/normalmap if tessellation is enabled. New tessellation features are described and explained in this video (global normalmap is then part of tessellation functionality).

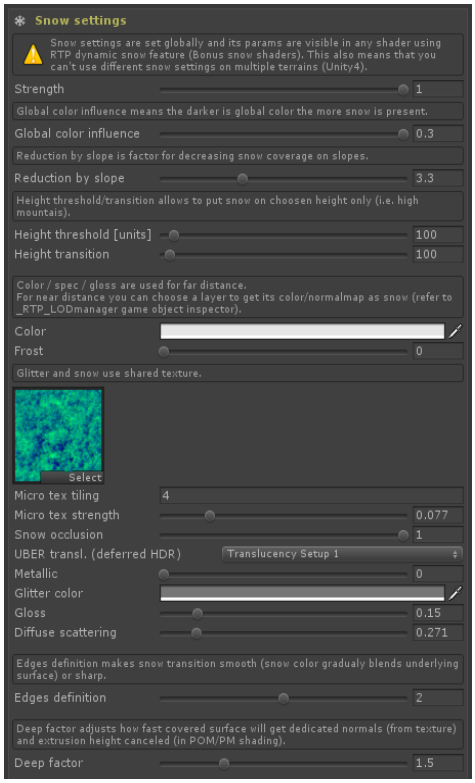
Global trees / shadow map



Although not used in example scene of RTP package this option might be useful when dealing with big environments that has high number of trees. At far distance Unity trees billboard are rendered using billboarding. At very far (for example when rendering flight simulator scenes) they can reduce to size of pixels and need not be rendered as billboards, but as a part of terrain texture. Where the texture RGB color isn't black it will replace global color texture above *Trees Distance* gradually using *Trees fade length*. Strength of pixel trees blending is controlled by *Trees blending* slider.
The same distance principles are used for alpha channel of the texture which defines shadows (black alpha channel will damp shader diffuse Albedo output). When you decide to not use lightmapper to bake dual lightmaps, you may use this option and place precalculated shadows info here (on alpha channel of the texture). Then set *Shadows distance* to the distance of your realtime shadows, adjusting *Shadows fade length*.
Such technique is suggested by Terrain Composer Author. In fact it's been implemented in RTP upon his request, so for more info – refer to his tools.

(RTP3.2 and up – trees map can't be used with conjuction to tessellation)

Settings – Snow



Snow coverage is controlled globally by *Strength* above and per layer in its settings:

As you can see most of slider are already described in inspector hint boxes, so there's no need to explain it here again. What you should remember is that snow color can be also taken from dedicated layer - just reserve one of your layers as snow texture/bump map – and define number of this layer in LOD manager. If you don't use it your snow will have constant color from snow settings. If reflections feature is enabled, your snow can also use this. Remember that snow coverage is controlled by following factors:

1. global snow strength
2. snow strength set per layer
3. global color map via Global color influence
4. terrain mesh and detail normal map slope steepness
5. by world Y position via Height threshold and Height transition

Color is self-explanatory

Frost has the same meaning like in UBER shader. Increases diffuse scattering and tints diffuse color of surface with snowcolor without actually placing snow on the surface.

Micro texture with tiling and strength is used to define finest normal detail of snow (for separate snoflakes that can appear on the surface). We can ask RTP to use the same data for refl/ambient occlusion. The texture is shared with Glitter feature (one of the texture channels store glitter mask).

In deferred rendering path, when UBER shaders are installed we can specify translucency setup for snow.

Metallic controls metalness of snow (RTP 3.3 has metallic PBR setup)

Glitter color controls makes it possible to use separate color/strength of glitter for snow and underlying surfaces (for more info look at Glitter feature).

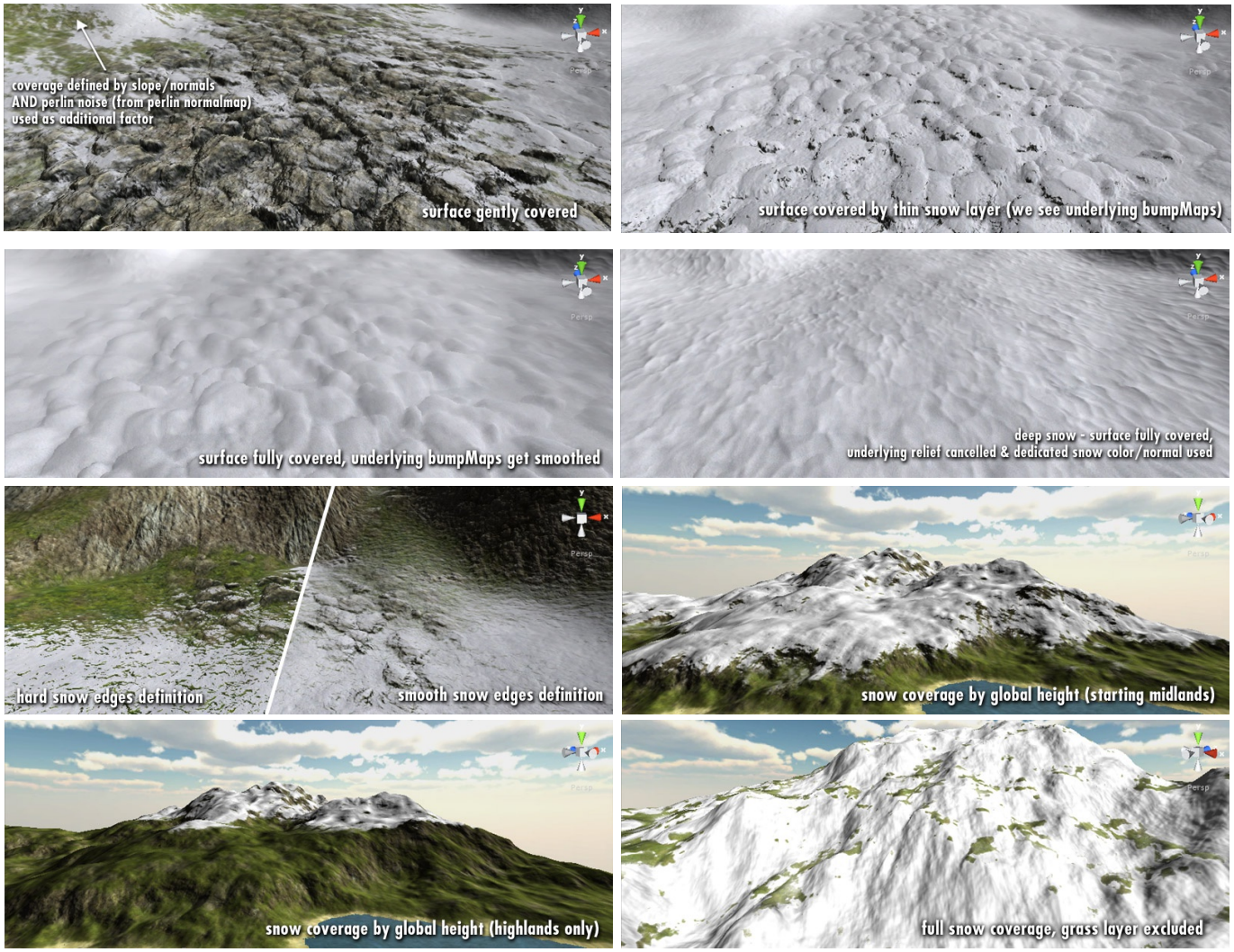
Gloss controls specularity of snow (1 – mirror, 0 – dull)

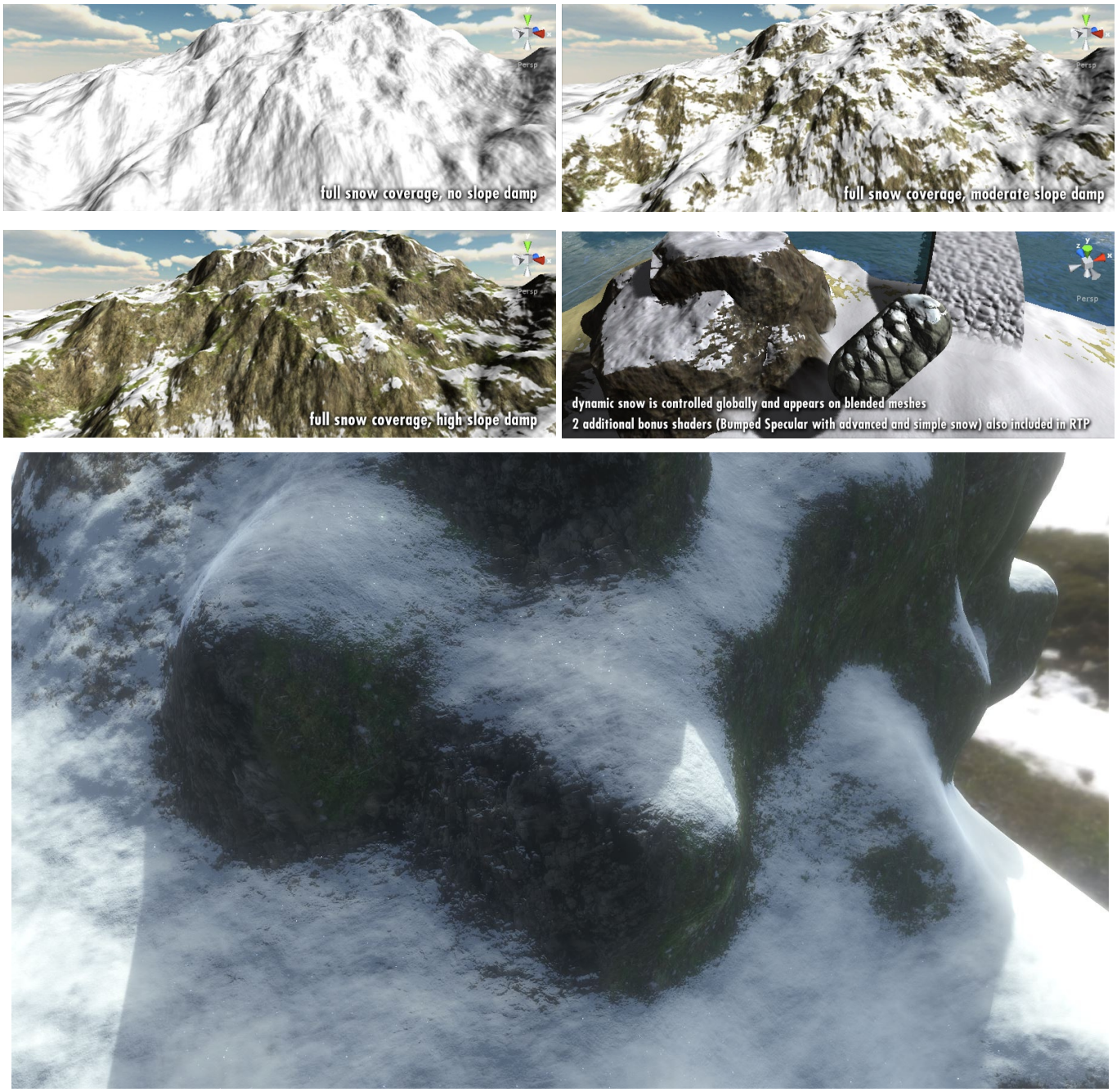
Diffuse scattering – snow can appear brighter at grazing angles apart from lighting conditions. This is kind of rim lighting (values around 0.1-0.2 works fine) that makes the look more natural.

Edges definition controls how hard will be transition between parts fully and only partially covered. Smoother transition allows to get gently visible snow "fluff" which makes underlying surface brighter.

Deep factor slider adjusts which coverage level will be considered as "deep snow". When snow first appear it only affect surface color, when the snow cover gets deeper is starts to filter normal textures (taking higher MIP levels) of underlying surfaces. When the snow get even more deep – dedicated texture/normal map appears (when this feature is used).

As an additional factor which introduces more variation shader uses also perlin normal value, so that exact level of snowed parts is not much obvious. A set of screenshots visualising different snow features are presented as follows:





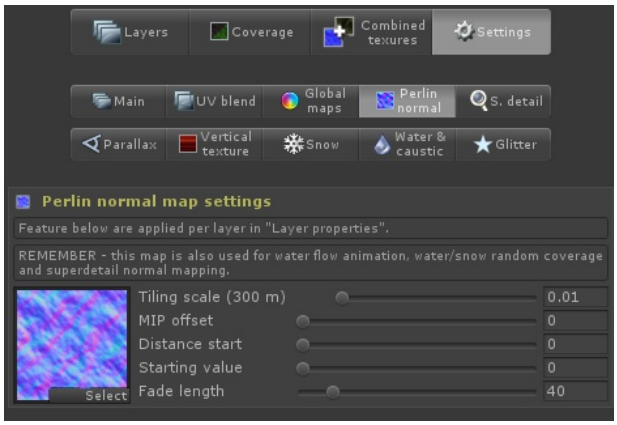
Snow placed on a mesh (available on one of example scenes) with glitter

There is also a bit "hidden" feature in shaders that allows to control snow coverage by global texture mask like we can do for terrain water. In *RTP_Base.cginc* and *RTP_AddBase.cginc* you can look for:

#define RTP_SNW_COVERAGE_FROM_WETNESS

Uncommenting this line and recompiling shaders you'll be able to use mask from wetness (enable this feature first to be able to use it in RTP script inspector) for snow coverage. When you finish working on your scene you can disable back water features as you can't use both – this hackish snow coverage and water together.

Settings – Perlin normal



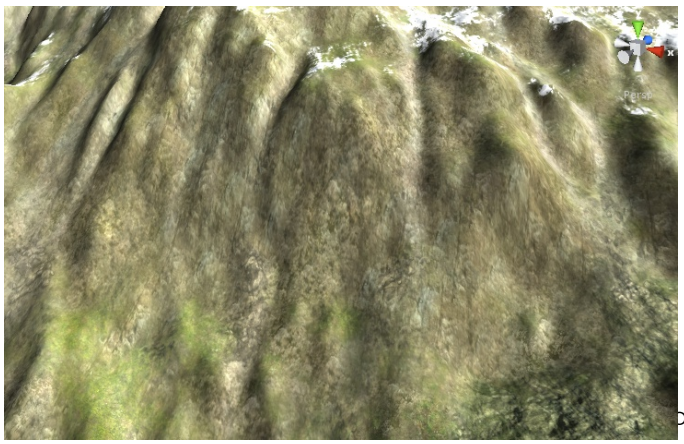
This texture is used in number of cases. First of all this normal map is mixed together with detail normal maps but at different scale (*Tiling scale*). It works like for UV blend, so 0.1 means than one perlin normal tile will be spanned over 10 detail tiles (1/0.1=10).

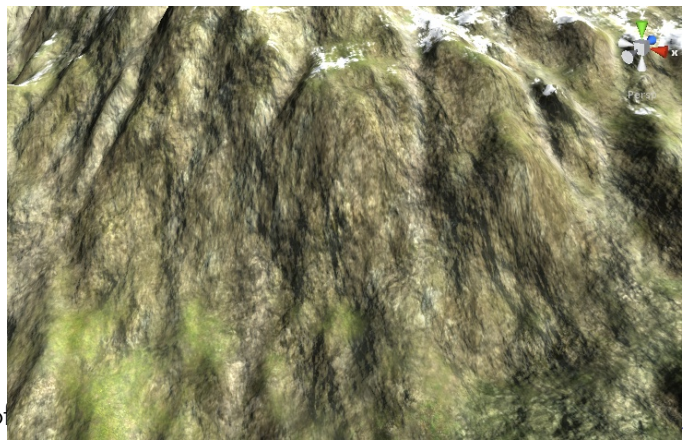
MIP offset allows using higher MIP levels of perlin texture (low-pass filter) – might work a bit faster for perlin textures of hi-res (like 2048).
Distance start and *Fade length* are very important as they define where "far distance" begins. At far distance we can have different global color map blend values or see only the global color map (depending on *settings*), but basically "far" means distance where perlin normal map will start appear.
Notice that close distance may overlap with far distance, but far distance can not begin closer than close distance start (described in Settings/main settings). Eventually, when close distance start is 0, far distance can also begin just from 0. In this case we can tell the level of perlin normal mapping applied from the beginning of viewing distance (very close to the camera). It will then gradually fade into full perlin at fade length. If you put *Starting value* to 1 you'll have no gradient, but whole viewing distance will present equal, full perlin bump map applied.

Perlin normal map texture is taken by default from a texture that is already present in RTP package, so you don't need to make it yourself (it's made in Photoshop using perlin noise clouds, then imported into Unity as normal map). If you'll try to leave it blank (no perlin tex) RTP will try to reapply default texture. The moment you apply texture here special combined texture is built (the texture actually used by shader). It's described later.

Far normal damp makes gradually normal vector that is taken from mesh biasing it to flat. At far distance with damp value set to 1 your terrain will be shaded as it would be "flat" mesh. Helpful when lighting info is baked into global color map (as lighting from mesh and color map superimpose).

This is how nice perlin mapping could look like, esp. on rocky surfaces. Notice, like it also helps removing tiling patterns from detail textures:





No perlin normal at rocks on slopes *With perlin normal used*

How much we apply perlin per layer is set in layer's properties:

Perlin normal is also used for making more variations on snow / water coverage, for superdetail (which is also additional normal mapping, but at small scale and very close distance) and for modeling wrinkles of flowing water (when the feature is enabled).

When you use UV blend at distance only far distance should overlap with close as small as possible or you'll experience visible outlined transition at the end of close distance.

Settings – Water & Caustics

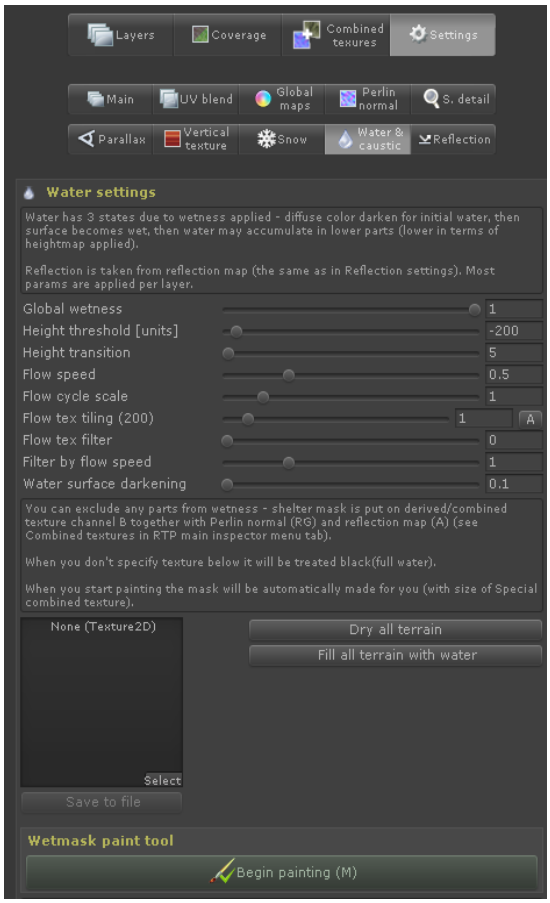


Fig 13

This is quite new feature section introduced in RTP3. It helps you making large environments affected by water, for example upon rainy weather conditions. We can wet or dry the whole terrain using *Global wetness* slider.

Height threshold and Height transition meaning is the same as for snow – we can decide where wetness on the ground begins in world space Y coordinate. That's helpful when we've got another "water" models in the scene like lakes, ocean. Then we might need to remove wetness under sea level without necessity of using wetmask texture. Another application could be simulating lava flowing down the mountain. In RTP3.1 wet surface can be also emissive. You could animate Height threshold runtime (decreasing) getting an effect of lava appearing gradually down the hill.

Flow speed controls how fast water will flow depending on mesh slope. For flat terrain where you'd like to make water flow faster adjust this setting higher, but remember that overshooting flow speed value will cause visible flow wrinkles stretching at peaks where water runs in opposite directions down the hill. Wrinkles of running water are taken from global perlin map using tiling from *Flow tex tiling* slider. Tiling = 1 means that one perlin texture is applied over one detail texture tile (remember about terrain alignment). Note that flow speed might be also set to 0 – then your water will just freeze – nice for ice simulation as it refracts and reflects nicely.

Flow cycle scale – you can try to use as low value as possible, because this will make your flow less repetitive (animation cycle will be longer), but it has consequence for areas of varying flow direction. Let's consider a tiny hill (better say bigger bump) on the terrain. When water flows, the direction of flowing isn't constant but different for all parts of the bump. After a while of animation noticeable stretching artifacts appear, esp. when flow speed is higher. To solve stretching we'll need higher Flow cycle scale values, but the same time animation will be also more repetitive. By this param adjustment you should be able to find right balance between reducing repetitiveness and stretching artifacts.

Flow tex filter allows taking higher perlin texture MIP levels (water wrinkles will get smoother).

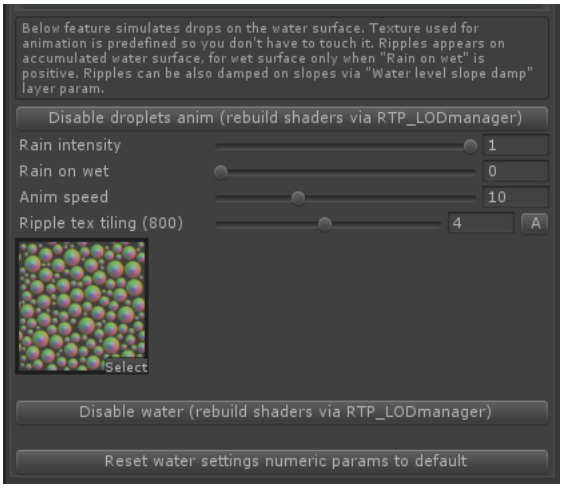
Filter by flow speed controls above filtering strength by water flow speed. Where water runs faster (for example down steep slopes) it will be less filtered - more wrinkles will look like water flow is more turbulent. When water runs slower we'll apply more filtering with high *Filter by flow speed* values (thus making smooth flow).

Generally idea of RTP water is that it has 3 states. First, when small amount of water is present on the surface, it gets darker and more saturated (true for most mineral objects). We assume at all materials has some porosity, which is not always true (white plastic doesn't get darker when wet), but terrain is specific case. This is controlled by *Water surface darkening* slider. With value of 1 surface will get completely black when it's wet, so beware. Values of about 0.4 works OK (in gamma color space). You should be also aware that reaction to water is influenced by layer glossiness. This is subtle, yet might be noticeable comparing very glossy to rough materials (they react more to darkening – as we assume they're porous).
Second water state is when surface isn't just darkened, but actually wet. In this state water doesn't have its own color nor use height map info.
Third state is when surface is wet and water level is high enough to accumulate/flow on the surface. Where such accumulated water is present depends on layer height map. On lower parts of surface (where its layer detail height map is dark) water will appear sooner with the same wetness strength level.

Whole terrain object wetness has its mask which helps exclude some parts from being wet - for example under trees or another covering water blockers (buildings, roofs, tents, etc). Masking texture is considered to be Alpha8 compressed (because we use only alpha channel of this texture). It's wise to prepare this texture in some external tool so you don't have to paint dry/wet areas by hand. You can, however make whole terrain dry or wet by one click using buttons next to water mask texture slot. Then you can use painting tool quite similar to what you've got on global color map to manually exclude some areas from being wet or cover them with water.

Remember that terrain wet mask needs to be of perlin normal map size because they're both combined into one texture (perlin will be RG, wet mask will be B) for shader. If you'd like to have more detailed water mask – increase resolution of your perlin normal map. If you don't use water nor reflections (which reflection map is also part of combined texture as A component) you can hold perlin normal map size lower – 512x512 is quite enough.

Water on surface can be also affected by rain droplets. This is additional feature to be enabled in RTP LOD manager. It works in presence of water only. It's inaccessible when we set water feature to "Simple water" in LOD manager. The same for feature of flowing water – in "Simple water" mode we don't have flowing water wrinkles.



The right texture that's used for animation is already prepared and assigned when feature is enabled. Rain intensity controls intensity of water droplets. You can adjust it runtime for changing weather conditions. All you have to do is getting any instance of ReliefTerrain script object (by getting its component from any terrain tile) – let's say you named this script instance variable *reliefTerrainScriptInstance*. Then you can adjust rain intensity by this variable:

reliefTerrainScriptInstance.globalSettingsHolder.TERRAIN_RainIntensity=avalue; // 0..1

and call:

reliefTerrainScriptInstance.globalSettingsHolder.Refresh();

The same way you can runtime adjust global terrain wetness:

reliefTerrainScriptInstance.globalSettingsHolder.TERRAIN_GlobalWetness=avalue; // 0..1

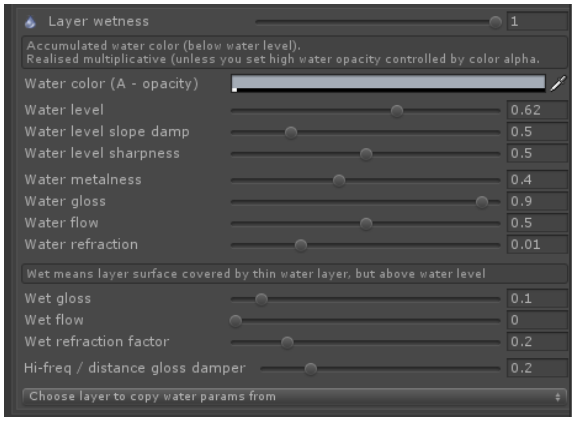
By default droplets will appear only on surfaces where we've got accumulated flowing water (water "3rd state"). If you'd like droplets to be visible on wet surfaces (these w/o deep water present) – use Rain on wet slider.

Anim speed slider usage is obvious.

Ripple tex tiling makes our droplets smaller / bigger. Default value spans 4 tiles of droplets texture over a single tile of detail texture. As usual – number in bracket means how many droplets tile we'll have over terrain (keep it integer to avoid seams).

Water flow/droplets applies for shader LOD levels above SIMPLE

Water itself is extremely adjustable per layer:



Using Layer wetness we can exclude some layers from being wet, or making them more/less wet.

Water color is color of water in its "3rd state" – when its accumulated. This is what is named "Water" in RTP settings above, in opposite to "Wet" which is "2nd state" (wet has no own color). For special effects you can put some extreme color values here like red, yellow water. In most cases gently saturated blue or cyan works fine. Alpha channel of water color controls its opacity.

Water emission – available only when emission feature is enabled in LOD manager. It controls how emissive the fuild is. For water it's zero, but for lava it's higher (preferably one). This is option available

when emissiveness is turned on in LOD manager. Emissive color depends on water color and its emissive glow described later.

Water level controls level of accumulated water. With high water level all surface will get covered by this "colorized" flowing water.

Water level slope damp can control water level by mesh slope. The higher is this setting and steeper slope is – the less accumulated (3rd state) water will be present.

Water level sharpness makes transition between 2nd state and 3rd state more smooth or hard (introducing harder edges of accumulated water level).

Water metalness overrides underlying surface metalness PBR value where deep water (3rd state) is present. This can allow you make tricks like flowing metals (set water color to light yellow, opacity to high value and dial up this metalness value).

Water gloss is simply water smoothness PBR value (it's controllable here because muddy water on soil/dirt surface can have lower smoothness value).

Water flow is amount of flow (wrinkles strength) applied.

Water refraction measures refraction distortion produced by flowing water wrinkles.

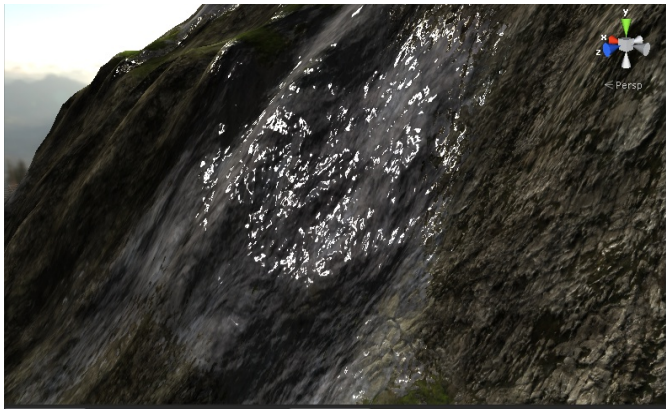
Wet gloss have the same smoothness meaning as above, but are applied to surfaces in 2nd state (w/o colored water present).

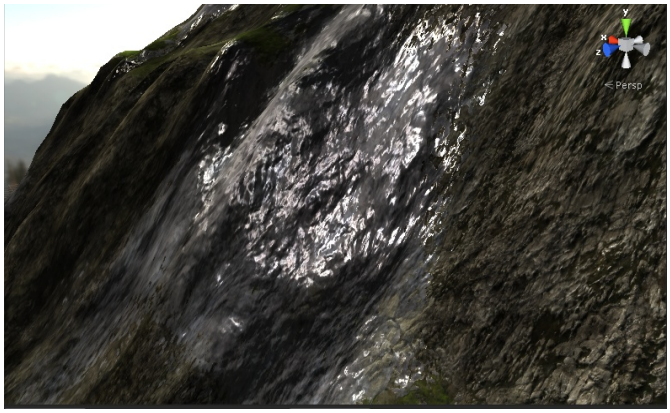
Wet flow controls amount of flowing wrinkles appearing at wet surfaces (2nd state).

Wet refraction factor is amount of refraction applied to wet surfaces comparing to water. With low *Water refraction*, for wet it will be also dampened (no wet gloss means there is no refraction – surface looks dull). If you put *Water refraction* to zero – wet surfaces won't be refractive neither. With *Wet refraction factor* set to 1 wet surfaces will be as refractive as water covered ones.

IBL spec / Refl - wet is the same parameter, but for 2nd state wet surfaces (described above for 3rd state water).

Hi-freq / distance gloss damper is helpful when your flowing water has hi-freq (small) wrinkles. With high gloss values such bumpy surface can present jittering artifacts of small specular spots. This slider helps reducing glossiness when we are farther from water:



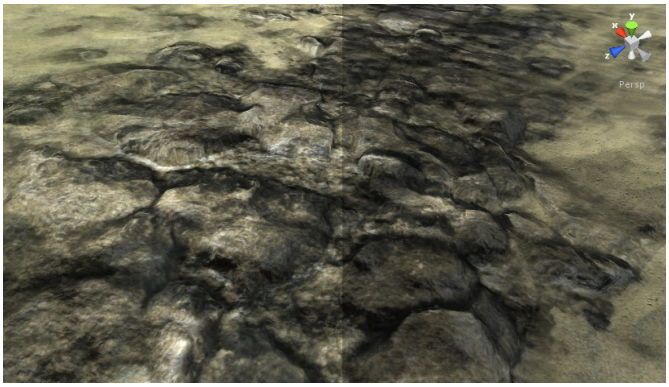


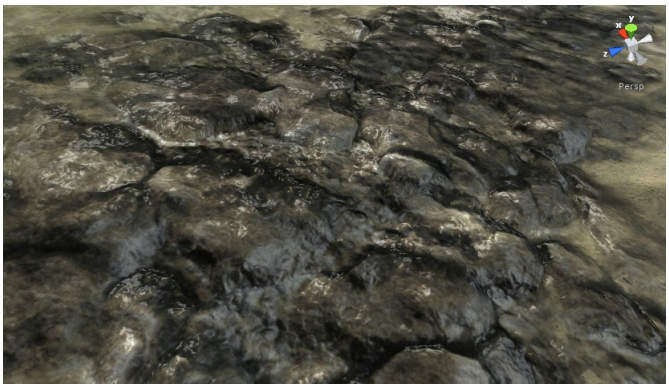
Left screen shot presents glossy water at mid distance. As camera moves (and water flows) you'll see small/pixel size specular spots blinking. For quickly flowing water that's maybe not big problem (in real

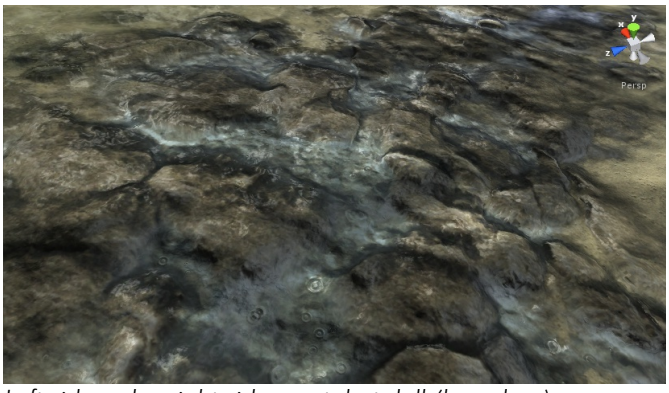
world we also see quickly blinking sparks reflections from the water), but you might find it better to reduce glossiness at the distance, so specular spots get bigger and softer. This gives an impression of anti-alias smoothening. For right screenshot above we've got gloss damper param set to 0.5 and water spec boost set to -0.7. You might ask – why do I need to reduce spec strength by 0.7 ? This is because what we see is LDR output while PBL give very high dynamic range results. Specular spots of very glossy materials (gloss=1) have very high specular power (up to 2000 !) while they are clamped to white (1). Reducing gloss makes spots wider but they are still very bright (even gloss of around 0.35 can imply LDR white color specularity). Without reducing spec strength, we would have smoother spots, but still white, which would look unnaturally overbrighten). Only when you would render your camera with HDR and normalize output to 0..1 range you would see everything dark except for glossy spots from the left screenshot.

As every layer has great number of independent water parameters to be set RTP introduces dropdown selector which helps quickly copying water parameters between layers. You can simply adjust water params for one layer and then apply it to the others.

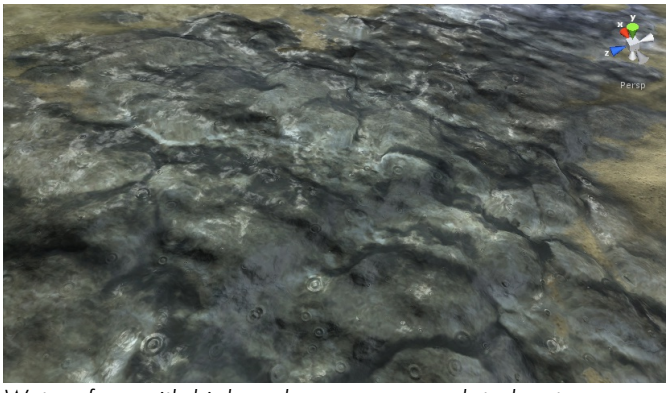
Below are screenshots presenting different state of water:





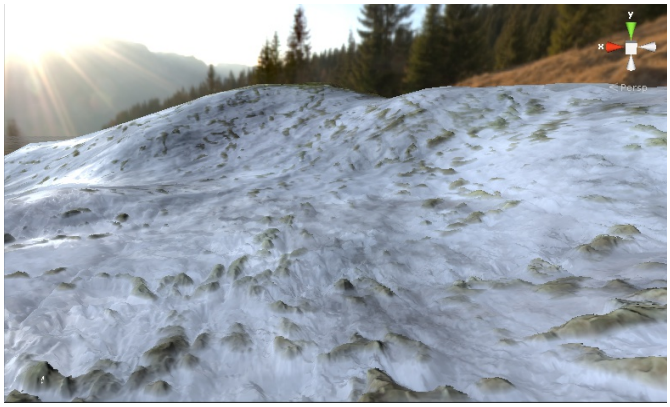


Left side – dry, right side - wet, but dull (low gloss)
Lower water level - accumulated water on the bottom only

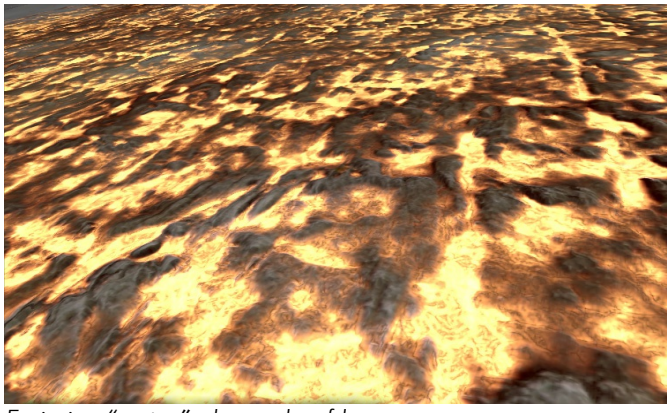


Wet surface with higher gloss, no accumulated water
High water level - rocks covered by colored flowing water

PBR reflections on the water (overexposured to present them obviously, point light shows direct PBR specularity)



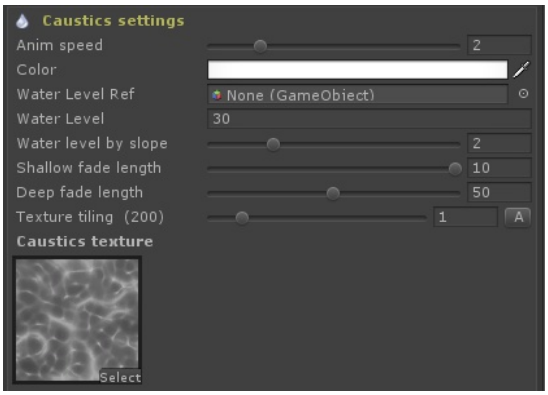
IBL reflections on opaque water (colored bright blue) with flow speed set to zero acts like an icy surface



Emissive "water" plays role of lava

Caustics

Feature helpful for deep water environments (lake, sea, islands). With caustics you don't have to use projectors to simulate these lovely animated light streaks at the bottom of your water reservoir.



Basically all you need is defining global water level where we render caustics below. This will cause that all terrain geometry that stays below desired level (in world Y axis coordinate) has this applied. This also implies that you can not simulate sea depressions w/o caustics or high mountain lake, because all your geometry below has caustics rendered, not only area of high placed lake.

There are 2 ready made caustics textures inside package (*ReliefPack/Textures/ReliefTerrain*). You can also find a version where caustics is present in A channel while RGB is vertical texture. Such solution is used when we've got Vertical texture from caustics option enabled in LOD manager.

Anim speed is obvious.

Color helps adjusting intensity and hue of caustics (which can be colorized texture, but in RT it's just grayscale texture used).

Water Level Ref is reference object which Y world position defines water level – it's useful when we'd like to control water level using some animation

Water Level – if above reference object is left blank, we can define water level here.

Water level by slope allows to put caustics higher on steep slopes of terrain (with high level of this param). So - on flat beach caustics can stop like defined by water level while on steep slopes of rocks that are sunk under the water we can raise caustics level a bit – this could resemble effect of light actually reflected by water surface and casted on rock slopes close to the water.

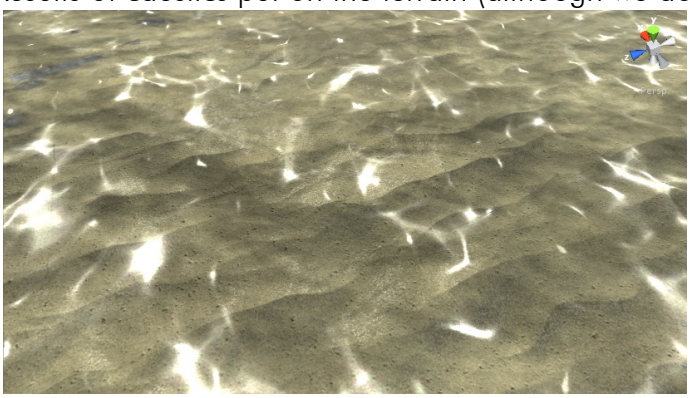
Shallow fade length – height of gradual caustics fade close to the water level.

Deep fade length – as caustics are not visible on very deep water we can define how deep they should

reach. With for example 15 you won't see caustics below water level minus 15 world units.

Texture tiling – here you can define tiling of animated texture in reference to detail texture tile size (keep number of caustics tiles over terrain integer – shown in brackets, use A button to align).

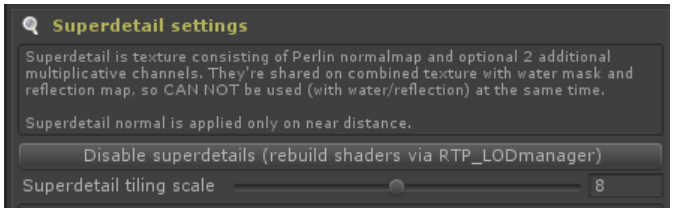
Results of caustics put on the terrain (although we don't render any water nor its effects here):



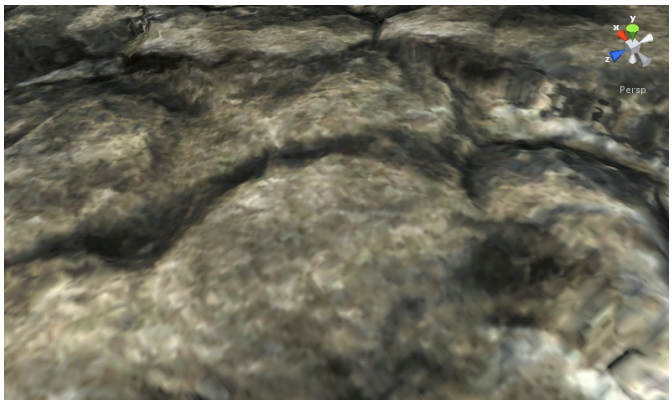
You should also notice that RTP has 2 bonus caustics shaders that you can use for your underwater scenery (fishes, corals, etc.) - they have the same caustics params like RTP shader has.

ReliefPack/Scripts/ReliefTerrain/SyncCausticsWaterLevel.cs is useful script you can put on any object that will update caustics water level for shader by its world Y position (available also from component menu – (*Relief Terrain/Helpers/Sync Caustics Water Level*). For example you can have an object that will follow level of water changing due to waves or tides – then caustics level should be also modified runtime to not let caustics be visible over the water. It's done globally (by Shader.SetGlobalFloat() call). If you put GameObject as a param for the script it will update caustics water level for shader that is attached to referenced object.

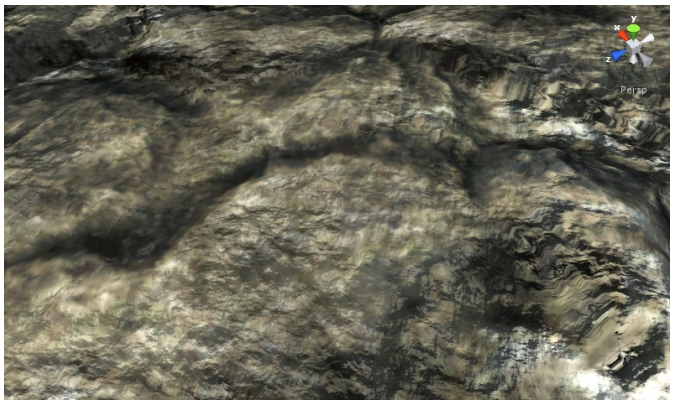
Settings – Superdetail



In most cases global settings of superdetail feature is quite simple – reduced only to tiling measured in reference to layer detail texture tiling (default value of 8 means that 8 tiles of superdetail are spanned over 1 detail tile). As in case of UV blend or perlin normal remember to hold tiling evenly otherwise you could experience seams between multiple terrain objects. Superdetail is just additional perlin normal map applied at close distance (the same as perlin normal map feature at far distance) – up to the middle of close distance length (start+fade from main settings).
It can bring more detail over some surfaces, esp. rough like rocks:

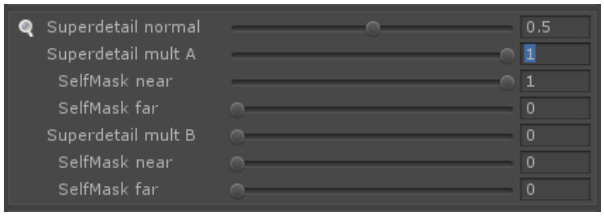


No superdetail – low resolution of detail texture revealed



Superdetail normal map – lack of detail resolution covered by additional perlin normal.

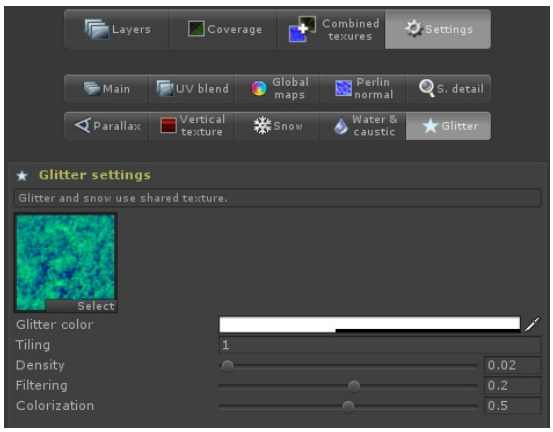
Like most feature settings, superdetail is applied per layer:



With superdetail mult channels enabled, you can additionaly use 2 grayscale masks which are multiplied with detail texture color (causing local darkening/brightening) for more variation on the surface. These masks are combined with perlin normal texture and placed on channels B,A – so they need to be of the same size as perlin normal texture is. As they occupy the same channels that are used for wet mask (B) and reflection mask (A) they can not be used simultaneously with wet/reflections features.

SelfMask near applies mult channel additionally masked by itself. With 0 you'll see mult channel at close distance – everywhere. With 1 it will be first masked by itself (selfmasking), but at scale of global perlin map (its tiling). SelfMask far does the same – selfmasking of mult channel, but applied at far distance. With 0 you won't see any mult channel applied at far, with 1 you'll see some variation on terrain (depending on channel value).
Channel B works the same as channel A and is applied after (so color is first multiplied by channel A).
Example textures for superdetail mult feature are placed in package *(ReliefPack/Textures/ReliefTerrain)*.

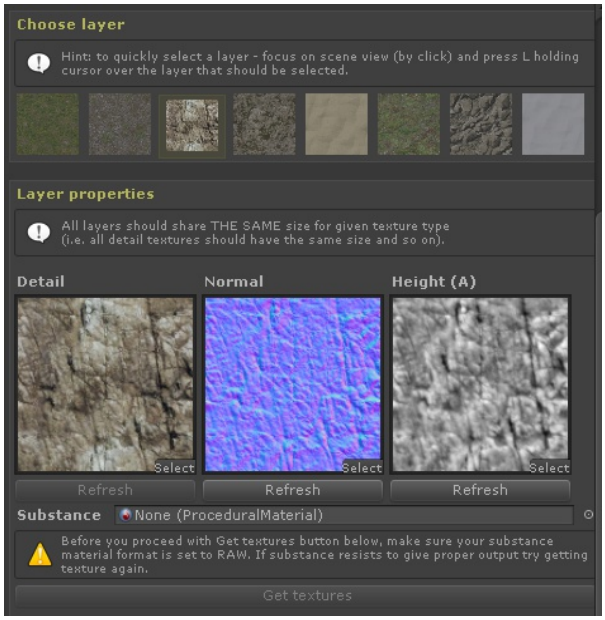
Settings – Glitter



The texture used is shared with snow. On R channel it has glitter mask (noise spreaded points) that's used for this feature. Glitter color RGB gives tint and strength of glitter "sparkles" on the surface. Glitter effect itself influences specular output (highly boosted in HDR range). It's not emissive. So – no ambient/reflection light means no glitter. Color alpha controls influence over smoothness of sparkle (higher alpha means increased smoothness of a sparkle). Tiling applies to sparkles texture. Density is sparkles coverage. Filtering is actually MIP offset taken when sampling sparkle texture. With high low value of filtering we would see random single sparkles faraway. With high filtering sparkles will get damped (filtered) beyond close-up distance. Colorization is pseudo random (controlled by pixel world position) aberration of sparkle color (for example when observing glitter on snow we might be able to see that sparkles are not monochromatic because light gets broken up into its constituent spectral colors.

When glitter feature is turned on, you can control glitter strength per layer - dry grass doesn't present glitter, you could use it for water droplets there. For mineral materials you can add glitter (granite). The feature itself is most useful for modelling snow where glitter effect is very obvious.

Layer properties



There are many options available in layers properties that haven't been described yet. First is active layer selector - similar (but totally independent !) to what we've got in Unity's built in terrain engine inspector when managing detail textures.

Two important hint boxes are refering to how to quickly select layer in scene view (**L** shortcut) and detail textures size constraints. Detail textures should be all of the same size. For convenience you can put detail texture here in detail slot (instead of in Unity's terrain script). Its alpha channel controls glossiness of the layer.

Detail texture w/o alpha channel (DXT1 compressed) means that alpha is considered to be white – your layer will have maximum gloss. To put right alpha channel into texture you can use attached tool which can combine channels across different textures. It's accessible from Main Unity Menu – **Window/Relief Tools/4 to 1 ...**

Normal map and height map slots might be left blank but it's not recommended as shader won't run any faster then. As detail normal maps are combined in pairs (layers 1+2, 3+4 and so on) they should be of the same size in pairs (layer 1 normal map must be of the same size as normal map of layer 2). More – to avoid MIP selection issues (normal map might seem to be too blurry or crisp) ALL your normal maps should be of the same size. The same for height maps – they are combined in quads (two combined RGBA textures for 8 layers - 1+2+3+4, 5+6+7+8) – they should hold the same resolution (preferably 256x256 due to better performance).

You can also specify substance that can be automatically grabbed into right slots of layer textures. Note that substance compression format should be set to RAW or you won't be able to grab it.
Important note is that shader actually uses combined textures that are calculated when textures in layer slots changes. This means that if you change texture properties somewhere in project folder or edit texture outside Unity - RTP knows nothing about it. In such case **Refresh** buttons below texture slots are meant to quickly rebuild combined textures (color for atlas, normal for combined normals pair, height map for combined 4 height maps).

Use Refresh button at the bottom of each layer texture in case it changed outside Unity. RTP need to rebuild combined textures to reflected changes.

Depending on shader mode used (4 or 8 layers in first pass) you'll be prompted for doing atlases, too

as they are necessary for 8 layers mode (actually they're used in shader, only in the lowest shader LOD level – classic shader will use original source textures).

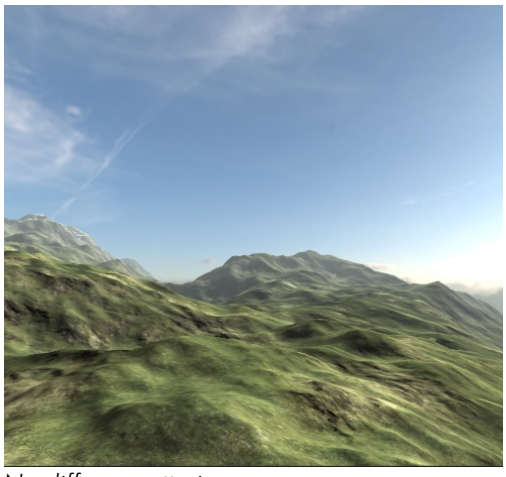
PBL / IBL properties section is the control center of PBL rendering pipeline in RTP:

Layer N metalness controls metallic quality of layer. In most cases it will be set to zero (most terrain materials are dielectric, non metal).

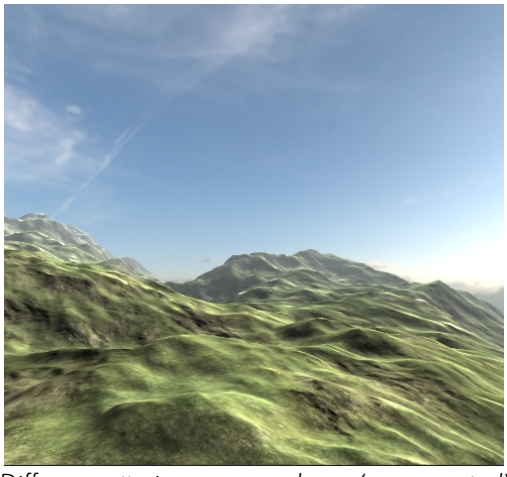
gloss min/max – you can range smoothness that's taken from diffuse texture alpha channel here - (0..1) values from texture will be mapped/lerped to (gloss min..gloss max) range

far gloss correction – if you experience specularity of terrain at far distance need to be corrected you can adjust it here.

diffuse scattering is useful feature when you'd like to model fresnel like effect but on the albedo diffuse part of your terrain layer (has nothing to do with other specular and IBL terms). Good example might be grass which scatters light inside – it's diffused between millions of separate blades so at distance grass surface could look a bit like fluffy medium. Another analogy related to 3d art might be rim lighting – that's what basically the parameter controls:

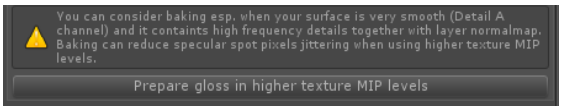


No diffuse scattering



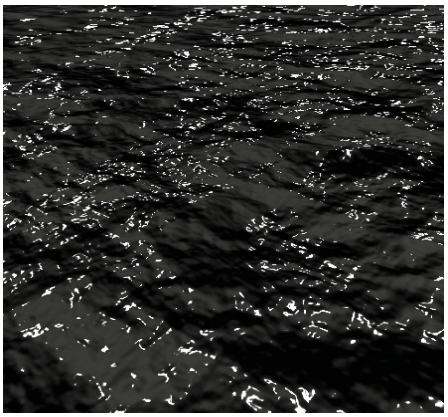
Diffuse scattering on grass layer (exaggerated)

PBL – gloss by normal variance baking utility section

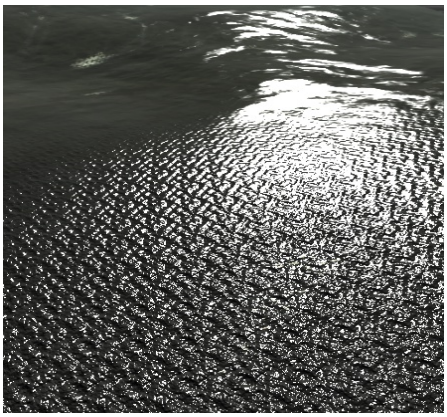


This utility you might not use at all, but only in some specific cases when you find your detail texture specularity causes problems. It reimports diffuse texture with subsequent MIP levels of alpha channel specialy processed to take into account energy conservation and normalmap variance of layer (high frequency bumpiness of detail bumpmap).

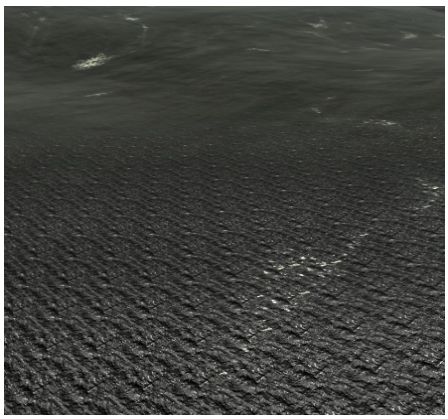
Let's consider a layer that has normalmap texture of high frequency. For sake of presentation I used perlin noise texture here. Detail color let be blank white texture with gloss set to 0.7. Below I'm presenting specular term alone.



Hi-freq bumpy layer of high glossiness (observed up-close)



Bottom parts jitter when moving camera while distant parts are "giant mirrors"



With baked normal variance data, gloss is corrected for higher MIP levels used

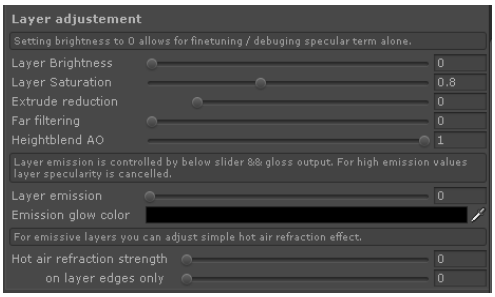
When observed up-close surface looks fine (left screenshot), but when we come farther problems arise. First one is that for lower normalmap MIP levels (0-2) variance present is still high (it's still bumpy, not smoothed by usage of higher MIP levels of normalmap). As PBR uses very high dynamic range we'll experience nasty pixel sized tiny shiny artifacts (look at the bottom of mid screenshot) that jitter as we move camera. The same time higher MIP levels of normalmap are used at far distance so they constitute flat surface. Material has constant, high gloss so we shouldn't be surprised we see giant mirrors there ! Simply we don't consider for specularity calculations that normalmap has high frequency details that should be diffused at distance... To solve this we can include such information for higher MIP levels of gloss map. So, for presented example gloss taken at distance should be diminished to compensate bumpmap variance. That's what I've done for 3rd screenshot. Clicked Bake MIP gloss data button and texture with glossmap corrected for higher MIP levels has been prepared and used instead of original texture.

There is also the other consideration that might be important. When your gloss map is also of high-frequency, its specularity might be realized incorrectly when using higher glossmap MIP levels. Let's consider a situation when at MIP level 0 we have 4 adjanced textels forming 2x2 chunk. First texel has full gloss, while texels 1-3 are dull. This means first texel can bounce light very much - with PBL high dynamic range glossy highlight spot there might have brightness of, let's say, 20. Adjanced dull texels will absorb and diffuse light so they are almost black. Our 2x2 chunk rendered in screenspace will have average brightness (energy) = 5 (20+0+0+0 / 4). When shader takes MIP level 1 of gloss map, it doesn't average energy, but actually average gloss value ! Average gloss for chunk is 0.25 (1+0+0+0/4) and energy bounced from such MIP1 texel is now quite weak (below 1) instead of what we would need in oversampled "ground true" situation (brightness = 5 is correct solution).

What RTP gloss baker tool does, apart from processing variance data, is also properly aligning gloss for subsequent MIP levels which is not simple averaging formula (a+b+c+d)/4 (!), so that actual specular power on the shader output remains constant. Without this feature your hi-freq material gloss map would quickly diminish its specular power at distance.

Generally gloss baking adjustment functionality (w/o actually baking anything) you've got also when playing with water. Actual baking there is, however, not possible because all water params are set dynamic (tiling, flitering, flow strength and so on). The way we adjust gloss for higher MIP levels used is realized by Hi-freq / distance gloss damper slider).

Layer adjustment section



Here you can adjust such options as:

Layer Brightness - controls brightness color (albedo) output of your layer. It's useful when aligning brightness of different layers.

Layer Saturation is rather obvious. Instead of tweaking such values in image editor (N x number of layers times) you can quickly do it here. In the production process you will experience textures often don't fit each other in terms of their saturation. This slider can save much time when using textures of different quality and origination.

Extrude reduction is the same as you would clamp dark parts of your height map to be greater or equal this slider value. This is separate feature that can be disabled (shader becomes a bit simplier/faster then).

Far filtering means additional low-pass filtering of detail textures at far distance. It just biases selected MIP levels in shader so higher are used causing more blurry look there.

Heightblend AO means adjustment of this setting per layer. It's useful when some of your detail textures don't look good with high border darkening.

Layer emission can turn your layer into emissive material. With slider set to max (in most cases you'll use it as 0 or 1) albedo color will turn into emission color which does not depend on any lighting conditions. This quality is, however, not constant for all your layer, but you can decide some parts of your detail texture might be emissive while other need not. Full emissive layer means your glossmap (A channel) becomes emission mask. Where it's white texture is emissive (with its RGB color), where it's black texture behaves "normal", but you don't have control over it's specularity anymore as A channel is now taken as emissive mask.

Emission glow color will be used in 2 situations. First to colorize emissive water. Second – to use this glow color at layer edges (where it heightblends with other layer). Emissive layers don't use heightblend AO feature and we actually don't darken edges, but set emissive quality there. The simple example might be static emissive lava texture. It will be bright yellow/red with white alpha channel (all layer we need to be emissive). Then you can cover it by rocks that pop up over lava due to heightblending. Where lava to rock edge is present we can set blending edges to glow (being emissive, too). This sounds like postprocess glow functionality but is realized here as a part of RTP shading.

Additional (optional to be enabled in LOD manager), "postprocess like" effect is hot air refraction feature which will animate terrain surface look:

Hot air refraction strength controls amount of effect per layer, which is adjusted globally in one of the Setting/Main sections.

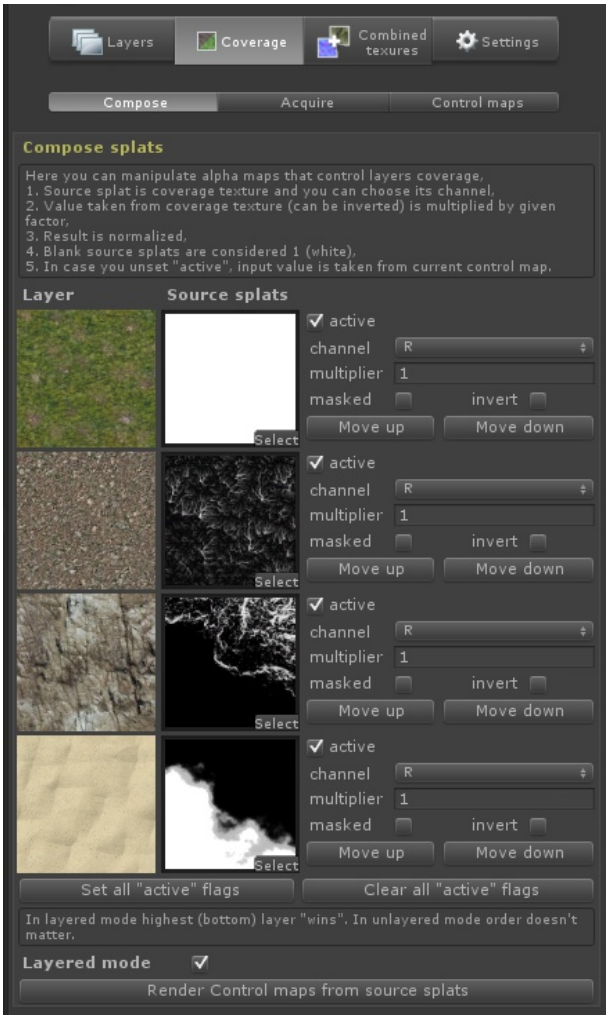
on layer edges only - we decide if effect is visible on the whole emissive layer or only where it heightblends with another layer.

It's worth mentioning that we can go to any settings tab just right from layer properties. For example:

Clicking on this tiny bluish icon representing Perlin normal settings will lead you right into the perlin settings. To go back – click on the scene view area occupied by an interesting layer and press L on your keyboard to quickly show the layer properties as described in hint box above.

5. Coverage

Compose



However it looks complex at first glance, compose tool is quite simple. It's briefly described in its hint box. The tool allows composing your own control coverage maps out of source grayscale source splats. They should be all of the same size as current control map resolution (Unity's control texture resolution setting). For RTP used on mesh if there is no controlmap already present resolution will be considered to be the same as source textures. Each layer of terrain is represented by one row with detail texture on the left for reference.
First check *active* checkbox to set desired params. Used source splat textures don't have to be grayscale – you can specify *channel* which will be taken as coverage value.
Source texture may left white if you don't specify any – this is the case in example above for the 1st layer (grass).

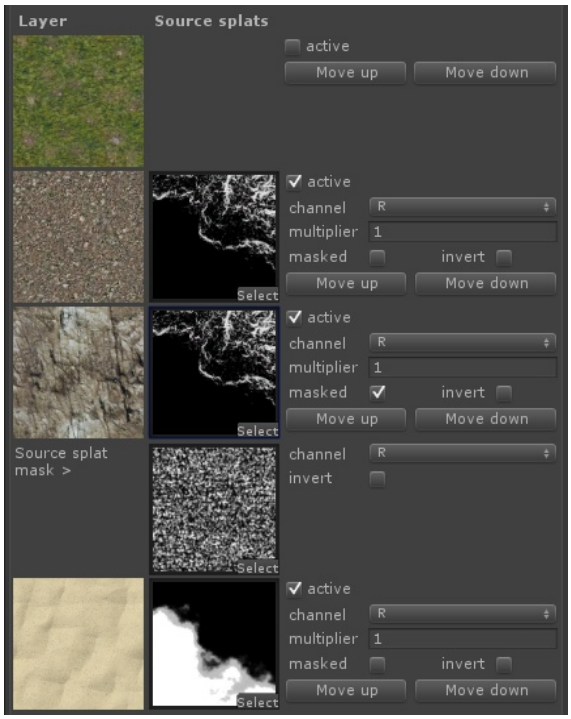
The most important is understanding of how coverage is calculated. It has 2 modes:

1. Unordered – when *Layered mode* checkbox is not set. Order of layers that can be modified using *Move up* and *Move down* buttons doesn't matter. Coverage strength is just taken from source texture and amplified by *multiplier*. To make a layer coverage stronger – put multiplier *higher*. Of course – where your source texture is black, amplifying won't help.

2. Ordered – when Layered mode checkbox is set. Layered mode is probably more convenient and intuitive way of composing – your source textures can overlap with their white areas, but coverage is calculated as you would stack layers. For this mode you'll probably set your first (top) layer source blank (will be white – empty like in example above). This way the first layer will "initially" cover whole terrain. Then subsequent layers according to their order will cover underlying ones. Here you can also amplify a layer if you'd like its coverage to be more pronounced.

Note that actually nothing happens until you hit *Render Control maps...* button at the bottom of compose tool inspector.

If you'd like a source texture black areas should define coverage – use *invert* checkbox to flip texture values. Sometimes you would like to use source texture but premultiplied with a mask. When you check *masked* additional texture slot will appear:



Source texture will be first multiplied with its mask and the result will be taken as coverage value (for amplifying by *multiplier*). Note that we used the same source texture for layers 2 and 3, but 3rd layer is masked. This way it will cover 2nd layer randomly (as 3rd layer mask is set noise). The same method has been used for example scene where I put 2 types of rocks on slopes. Note that you can use such approach to remove tiling for some "difficult" layers (which detailmaps reveal very noticeable pattern over larger areas).

Check layer multiplier values if you see your results are just black – for all layers amplification set to 0 would be just the case... Also if in some parts (the same mask tex coords) all your source textures are black – results of coverage calculation will be broken there. That's why you should set first layer blank in layered mode – doing so you make sure that all your terrain is covered by "something" (at least by this top layer).

In case you'd like to modify coverage of a selected layer only – set it active, and the rest – inactive. Current coverage value will be taken from layer if *active* checkbox isn't set. So inactive layer would be the same as if you just put currently calculated control map as source, set multiplier to 1 and check right channel associated with the layer. For first layer this will be R, for 2nd layer G, and so on. Inactivating all layers and recalculating control map is the nice method of doing ... nothing.

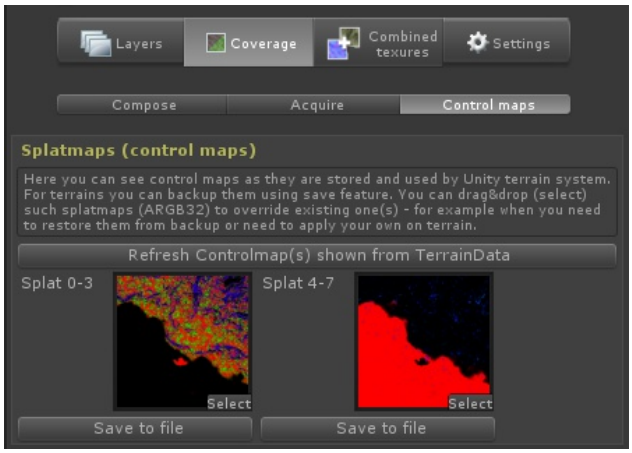
Acquire

As composing tool described above uses grayscale source textures, here you can get them. Wisely using steepness map with height map you can cover your terrain natural way (as grass doesn't grow at steep cliffs). You'll probably have to work over textures acquired here in some 3rd party editor (for example to adjust contrast of steepness texture). In case of example scenes I haven't used acquire tools, except for *Save direction*. WorldMachine made nice coverage textures better (you can find them in package together with WM2 project file) – not only steep map, but also flowmap, erosion, deposition or snow coverage maps. Also – Terrain Composer is very versatile and powerful tool to manage your coverage masks – right in Unity (and number of tricks TC can do is a lot more than only composing coverage control maps).

As steepness and height are rather obvious, *Save direction* can be described more. For each pixel it takes terrain normal vector + *Direction reference* game object local z axis and calculates dot product. If you take directional light as reference object the texture produced will be something like your terrain diffuse lightmap (where terrain surface normal facing light dir will be white). Such texture has been used in example scene with global color map – I baked WorldMachine color map output with such lighting info – example scene terrain looks detailed even with low poly mesh.

Another application could be when you'd like to get areas facing for example only north – turn your reference object heading desired direction and check *Skip y component*. You can optionally check Flip direction, when you'd like to invert result.

Control maps



Control maps purpose have been already described previously in chapter 3. Following layers coverage is controlled by subsequent RGBA color channels. For more than 4 layers we need more control maps (2 for 8, 3 for 12). For terrain control maps are automatically generated by Unity's engine when a layer is added/removed. If you'd like to provide your own – you can put them here directly, note that the size of provided maps must match the size of the current Control texture resolution. The maps shown in this section of RTP inspector are used by the RTP shaders. If for some reason you feel that they don't reflect actual Unity's terrain controlmaps (while they should in most cases) click *Refresh Controlmap(s)* button.

In case RTP is used on arbitrary mesh you must provide these maps. You can either compose them yourself or use the tool described above.

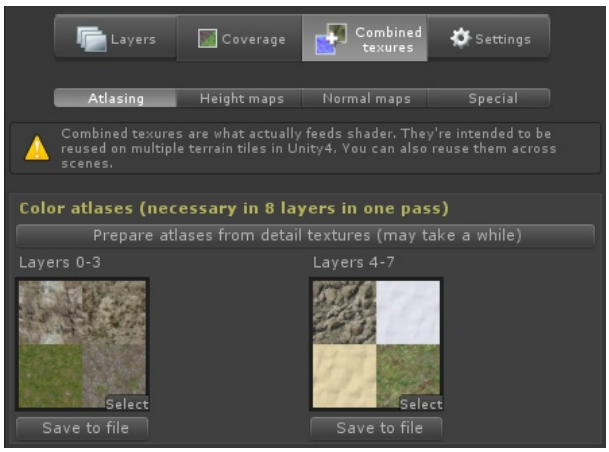
Make sure that control maps you'd like to put are normalized. As RTP composing tool takes care of normalization (like Unity terrain engine do when you paint coverage on it), it's up to you to normalize custom control maps. Normalization means that all layers coverage values sum to 1. For example a pixel defining coverage of values RGBA (0, 0.25, 0.5, 0.25) is normalized and coverage of 3rd layer is 50%, and layers 2,4 – 25%. It doesn't necessarily mean that in RTP this pixel will be covered most by 3rd layer because of heightblending factor. On the other side RGBA (1,1,1,1) is not normalized because it sums to 4. Normalization of such RGBA texture would be dividing each component by the sum (4). It would result in (0.25, 0.25, 0.25, 0.25) normalized value. For more than 4 layers (more control maps used) normalization should apply to sum of components from every control map. For example for 8 layers RGBA (0, 0, 0, 0) + RGBA (0, 0, 0.2, 0.8) value is normalized – coverage of 7th layer is 20%, 8th layer – 80%.

Not normalized controlmaps cause coverage problems – either dark, uncovered areas (when sum of components is <1) or overbrighted (sum >1).

6. Combined textures

Atlasing

For 8 layers in first pass rendering mode we need detail textures to be atlased. It's done automatically, or you can do it manually here:

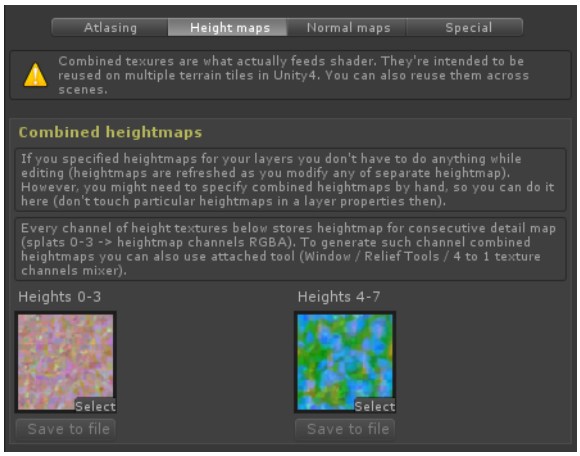


For 4 layers mode, atlasing is an option which helps you saving texture resources for additional features. Above textures are used for all shader LOD levels except for CLASSIC which doesn't use atlasing but renders 2 passes (4+4). It's wise to save atlases when you decide your detail textures configuration
is final. You'll be able to reuse atlases on different scenes. Atlases are also stored in presents and can loose their references as they are scene objects before saving. To scale down a little bit detail textures RTP uses hi-end resampling services – even better than Photoshop's bicubic – for atlasing to deliver it as good as possible, but if you'd like – you can provide your own atlases, but remember that textures should be padded/wrapped 16 pixels around.

As atlases are twice as large as detail textures upper limit for detail size is 2048 (atlas will be 4096). However - it's advisable to use detail textures not larger than 1024 (atlas 2048) for performance reasons – in case of atlases GPU have to "jump" over 4 distant texture pixels for sampling values of layers which may cause caching issues. As you can figure – one atlas are made out of 4 detail textures, so they should be all of the same size. Even if not atlased – when using detail textures of different sizes – expect MIP selection problems – your textures might look too blurry/crisp.

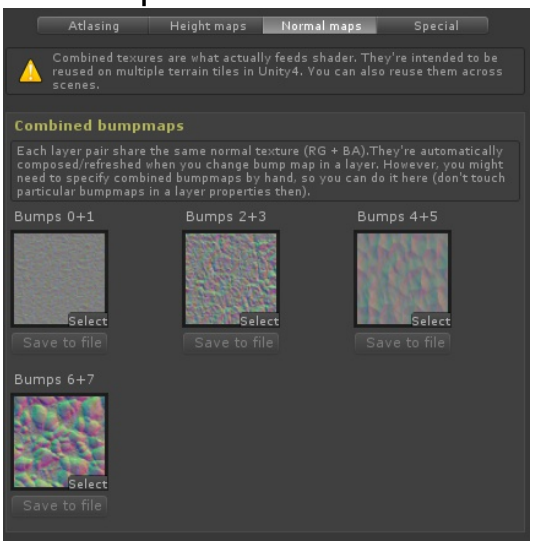
When you can't see your detail maps in 8 layer mode reason might be atlases are not prepared yet (or global color map blending is not multiplicative while global color texture itself is empty). If your atlases are present but you still can't see them used you can eventually use ***Refresh all*** button in *Settings/Main*.

Height maps



Here you can look at textures that are produced out of 4 layers height maps (RGBA values correspond to layers 1-4, 5-8 or 9-12). In our case we use 8 layers so we've got 2 height maps. Combined height textures are produced the moment you put something on height texture slot in layer properties. Here they are already saved (inactive *Save to file* buttons), so they're not stored in RTP object (saved in scene) but are separate files in the project and could be reused in other shaders or scenes. If you'd change any of height texture in layer settings – combined texture will be redone (thus not saved).
Here's also the first place you should look into when your heightblend or POM/PM shading seems not to work (also – check your global color map alpha channel isn't 0 nor extrude reduction per layer isn't 1). Note, that height maps that you've got specified in layer properties are not the textures used by shader, but above combined textures are ! Inspecting values of combined height maps will make sure that everything is OK on this side.

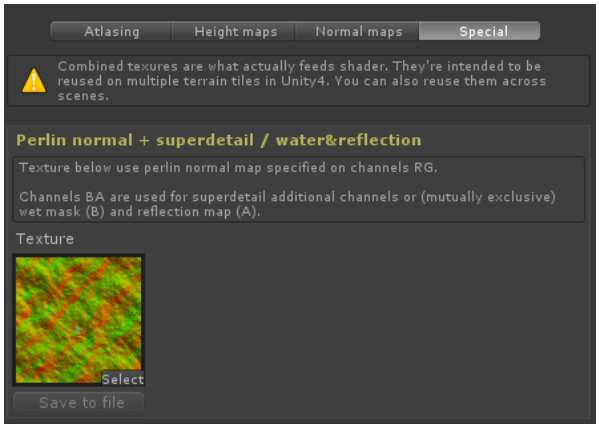
Normal maps



As in case of height maps – here you've got combined textures produced out of normal maps pairs (layers 1+2, 3+4, 5+6, and so on). These combined textures are actually used by the shader. normal maps are not needed on your build (and are automatically "detached" from RTP script when building). This cleanup process is realized by *ClearAuxTerrainTextures.cs* script. Save combined normals as your layer textures setup is ready, but remember that they are redone each time you put something (new) on normal map slot in layer properties.

The reason your detail bump maps not work is often that they are not marked as "**Normal map**" texture type in its settings. In linear color space check *Bypass sRGB Sampling* in texture properties for all normal maps and height maps.

Special



This texture holds perlin normal texture on channels RG and either:

1. Water coverage (B), or
2. 1st superdetail mult channel (B), 2nd superdetail mult channel (A)

Hence you can't use superdetail mult channels with water/reflections.

Inspect this texture B channel if you notice your water coverage mask isn't working like it should.

In case water feature is not enabled, B channel of special combined texture isn't used. RTP try to reuse this texture then across multiple terrain tiles. If you use water – this combined texture is terrain tile specific and can not be reused for different tiles (you can, however, put a texture here manually to force it) as they probably need different water coverage.

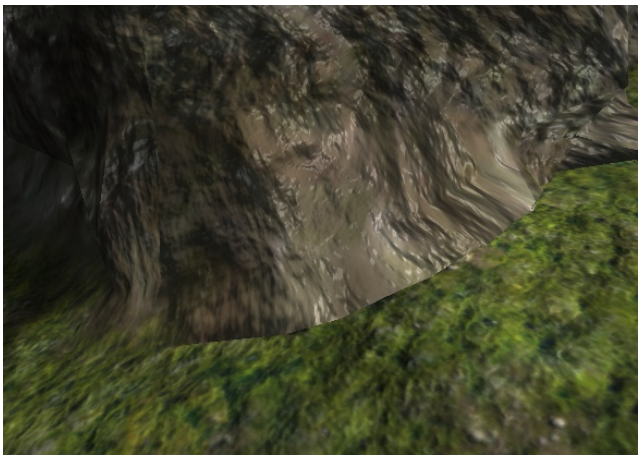
The moment this special combined texture is created after RTP script is attached to a terrain is when you "visit" Settings/Perlin normal tab, or change water coverage, or reflection map. So – at the beginning go to perlin normal map settings or put ready made special texture here manually for perlin mapping to work.

Note that this combined texture is also used for some additional shaders (water) present in RTP package – as they use both perlin feature and reflection map/water coverage the way RTP terrain shaders do. Look at material perlin texture description in its properties – if marked "special" – use this texture instead of regular normal map.

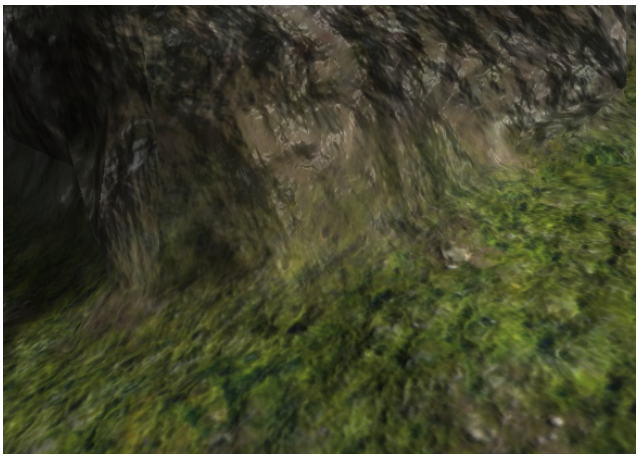
7. Geometry blend features

Terrain blending with a mesh

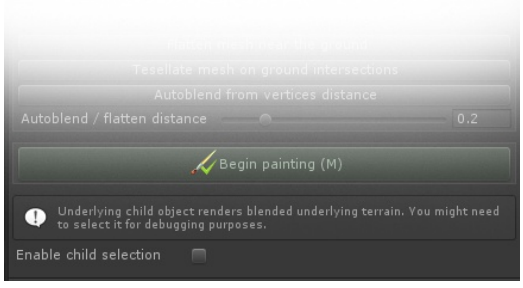
You've probably faced the problem many times – objects that are put on the ground has very distinct edge where both terrain and the object intersects:



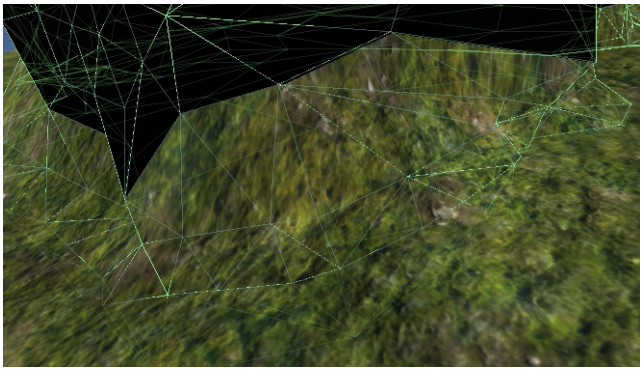
Top one is a rock mesh. Level designers hide it by adding some small elements like little rocks, plants and so on to reduce this effect. But with geom blend feature you can place a mesh, attach an RTP component called *Relief Terrain/Geometry Blend* from component menu, then choose geom blend compliant shader to be used on mesh, setup a few more params and results might be like this:



What happens when you actually set *GeometryVsTerrainBlend* script on the mesh blended ? It puts additional object with the same mesh as ours as the child with special shader that "mimic" terrain. Then main object is rendered with geom blend compliant shader – its blending mode is decal:blend in surface shader, so alpha value produced by the shader decides where underlying, child "terrain like" object is visible. To be able to select and look at this underlying object check *Enable child selection* checkbox at the bottom of geom blend script inspector:



Then, when you disable the main object renderer you'll see the "terrain child" alone:



Black parts are also rendered by "terrain like" shader (it's in *Internal/ReliefTerrainGeometryBlendBase* or *Internal/ReliefTerrain2GeometryBlendBase* depending on our terrain is unity one or arbitrary mesh), but they are just skipped for better performance since main mesh fully covers underlying object at this part of object. Even more – as the rocks occlude the terrain behind black parts that are drawn before – it causes that terrain expensive pixel shader is skipped. So – setting up scene with many big geom blend objects can actually make your scene will work faster.

Let's how to setup geom blend object step by step. After you attach it to an object you'll see something like this in object inspector:

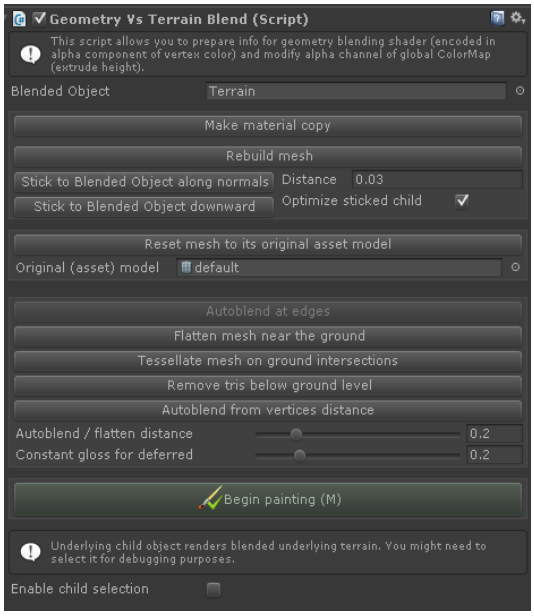


Figure 12

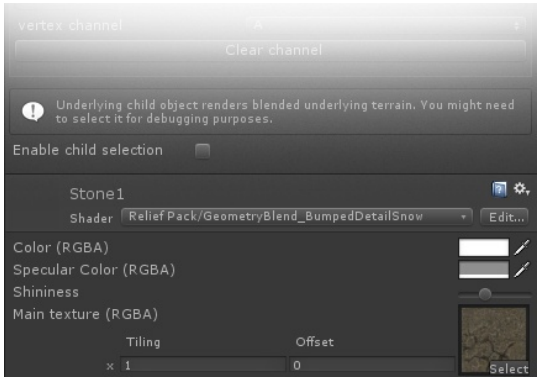
First thing is clicking *Rebuild mesh* which should be also done every time you modify transform of your object (moving, rotating, and so on). As you see the script has to know what's the terrain object that we're blending with – *Blended Object* field is setup automatically or you can put here an instance of your terrain game object manually. You can also make material copy to have your object use separate one. Working with geom blend makes your object mesh to be rebuilt and is no longer the mesh of the object before, but its reference is still hold in *Original (asset) model* field. You can then restore the model to its original mesh pressing *Reset mesh to its original asset model*. New mesh used use vertex color alpha channel to control blending coverage, so your model can not use it.

As mentioned – main object can not use any shader, but "geom blend compliant". It means that shader should meet some special requirements to work properly. There are many ready made shaders inside RTP package that are geom blend compliant (either ready to use or zipped to reduce package import time). They are placed in *Relief Pack/Shaders/Relief Terrain/GeomBlendCompliant* folder. For example scene rocks I use "bumpmapped specular detail & snow" shader. In your case you can use any of these shaders or make your own "geom blend compliant" shader which is quite simple in most cases and will

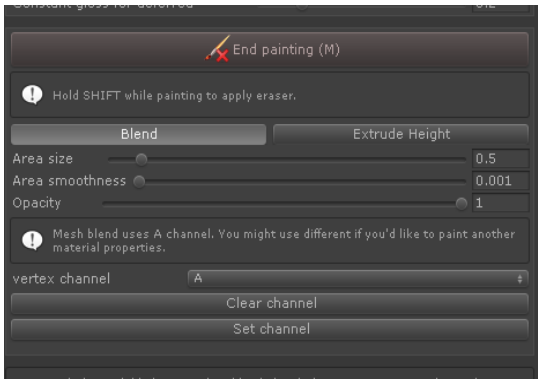
be described later. Note that I also put a shader variation of Michael O. (Manufactura K4) – Bumped Diffuse Overlay which he uses on his rocks stuff (don't worry about the intellectual property - his shader has been made by me :)). With this shader you'll be able to blend his rocks with RTP terrains. New PBL shaders that you can configure to use most of RTP features are placed in **CustomizableShader1Layer(PBL)_geomblend.zip**, **CustomizableShader2Layer(PBL)_geomblend.zip**, **ReliefTerrainPMTriplanarStandalone_geomblend.zip**, **ReliefTerrainPMStandalone_geomblend.zip**.

Most shaders placed in RTP package has extensive comments inside – please refer to them for more info about shader you'd like to use. Because they are adjustable - inside you'll find many instructions how to do it.

After you put your object, add RTP geom blend script component, rebuilt mesh and selected geom blend shader like here (*GeometryBlend_BumpedDetailSnow* shader used on the example scene rock):



You can try to blend it with terrain using painting tool (similar what you have on terrain script – painting global color map or water coverage mask) – *Begin painting (M)* button (figure 12):



The button turns into *End painting (M)*. With your keyboard **M** shortcut you can turn it on/off quickly.

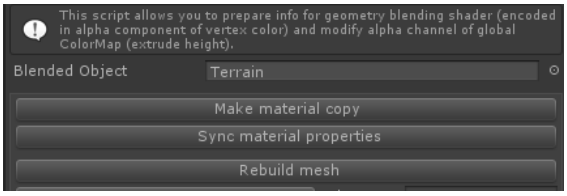
Manual painting needs collider to be present on the object (only for painting) !

By default you can paint *Blend* which means that *vertex channel* will be affected. By default this is alpha channel which controls blending, but for versatility you can choose it here from droplist as some shaders use other vertex color components. *Clear channel* button zeroes instantly all vertices color for selected channel (in case of blending this means your object isn't blended at all). This implies that shaders that uses alpha vertex color component can not be made "geom blend compliant" that simple. *Set channel* makes the opposite (setting vertex color channel to 1).
The 2nd tab when painting is *Extrude Height* which is the same as when painting global color map for terrain – using this you can draw over alpha channel of global color map causing reduction of parallax effect in selected areas.

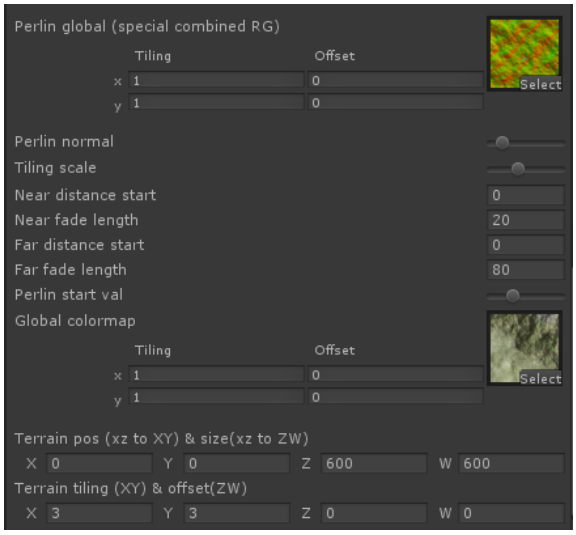
Extrude Height painting is useful and often necessary when you blend your object with terrain area which is heavily extruded in terms of parallax height – blending them w/o reducing parallax would reveal disturbing artifacts at blended edges.

Manual painting isn't the only method to blend intersecting edges of mesh. Refer to figure 12 where *Autoblend from vertices distance* button is present. Actual distance used is set by *Autoblend/flatten distance* slider. You'll see that every vertex that is close enough (below distance threshold) to terrain surface will be blended automatically then. When your mesh has many vertices that are placed under the ground level you might find good option to remove them as they are invisible anyway. This can be done using Remove tris below ground level button.

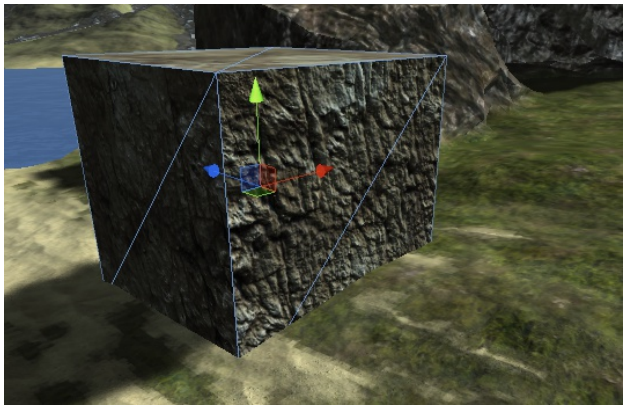
As underlying "terrain like" object mimics terrain it's also – like terrain – textured top-planar. This means that surfaces of blended object that are orthogonal to terrain surface will reveal stretching. Let's consider a case. I've just put a cube on the terrain and applied geom blend script to it - simple PM shader will then used automatically. After initialization using *Rebuild mesh* button for this specific shader RTP detects that the shader used can be also integrated with terrain better – its material uses combined perlin normal map and even terrain global color map so your object can be better blended. Note that in case of initializing such shaders with *Rebuild mesh* another button appears – right above this button – it's *Sync material properties* which automatically looks for the RTP terrain area under blended object pivot, and when detected it gets terrain params like global color map and so and puts it automatically on geom blend object material:



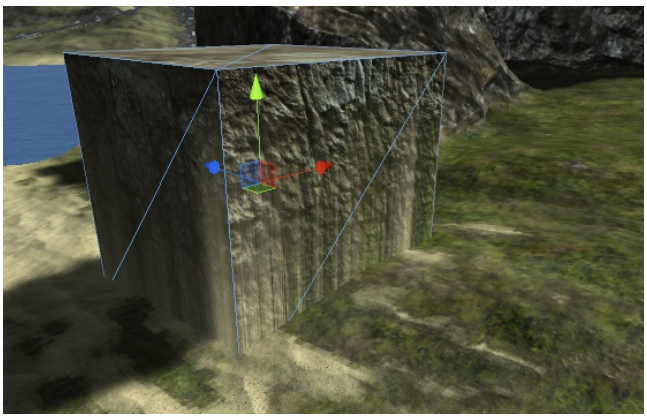
You don't have to attach underlying properties manually:



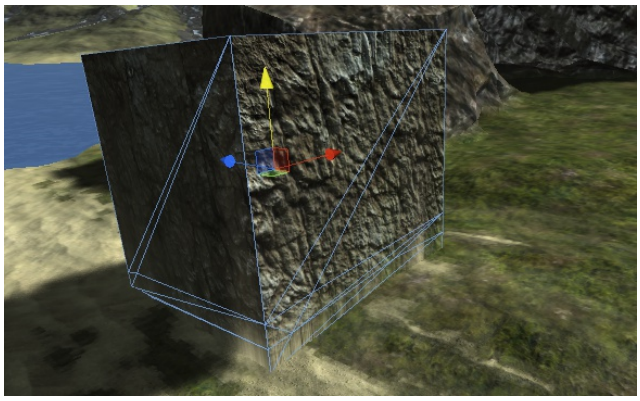
All the settings like position, tiling and its textures are synchronized with blended terrain. Our cube looks like this:



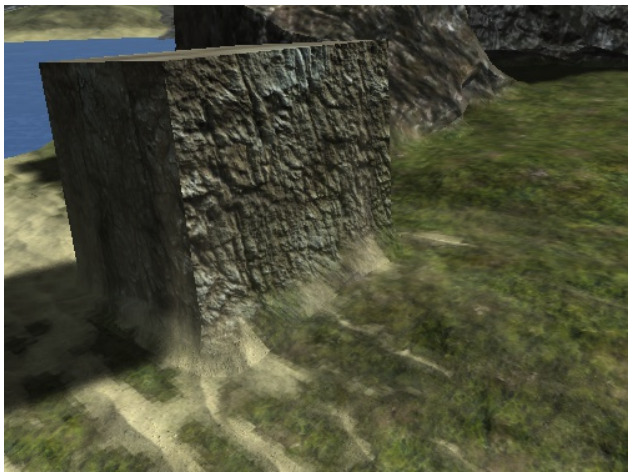
Now let's try to blend it with underlying terrain (either manually or using autoblend feature):



Hmmm, looks really bad because cube walls are almost exactly orthogonal to the terrain. Now lets use *Tessellate mesh on ground intersections*. Then moved the cube a little bit up and tessellated it again. It looks like this:



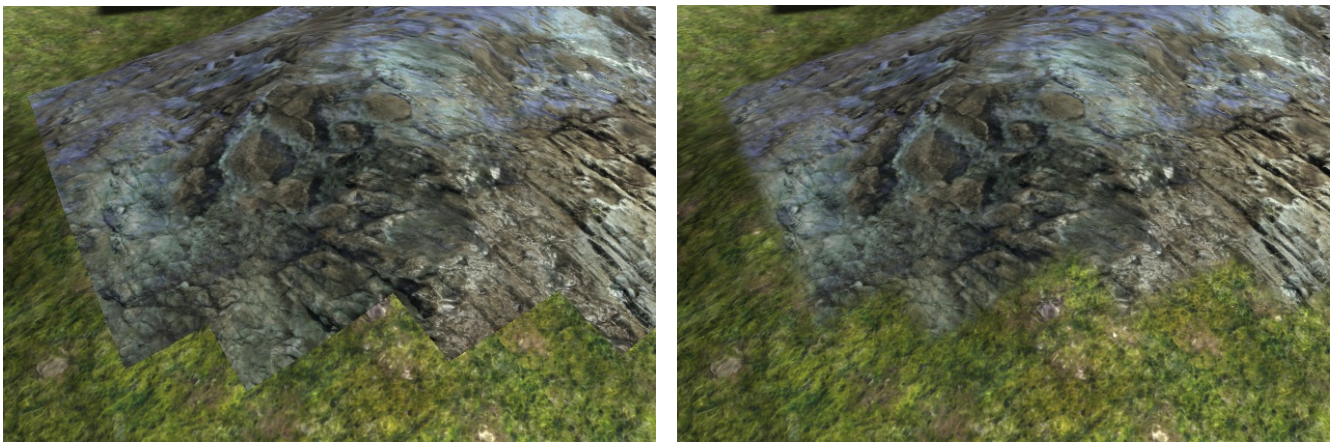
I also rotated a little bit the cube to better fit terrain slope. Note additional vertices and faces introduced. Then I auto blended it the way only the bottom, tessellated parts are blended near the ground. Now using *Flatten mesh near the ground* button (clicked a few times) I pushed problematic orthogonal parts of mesh so they are gradually moved along terrain surface:



Yes, now it looks much better. Maybe not perfect due to some "terrain like" object tangents interpolation inaccuracy, but edge has been nicely hidden. With "easier" objects (which doesn't have that difficult triangles initially) effect would be better. You can make it the same on any of your geom blend objects. For example you just put a rock, tessellate its bottom twice (or more for smoother mesh result), flatten bottom and voila – it can be seamlessly blended ! Note that flattened are only vertices that are blended. This way you can control which vertices should be pushed and which left alone (even if they are actually problematic - orthogonal).

Terrain blending with sticked mesh surface

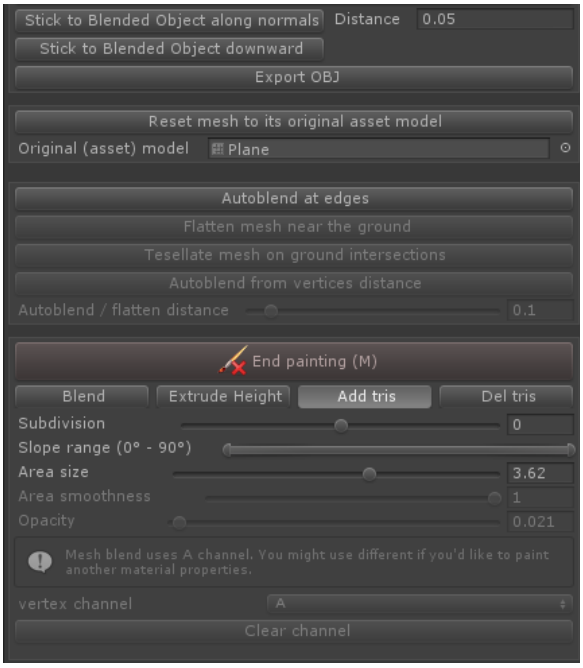
The other type of blending is when you have a flat mesh – for example an outline for a road and you'd like to "stick it" to terrain with smooth blending at edges:



In our case it's fragment of terrain example scene where I sticked a mesh with water shader. Left screenshot is sticked but not blended, right one is blended using *Autoblend at edges* which does it exactly like named – RTP attempts to find edges of mesh and put blend 0 value there (zero alpha for vertex color).
And this is how it's setup. You can either stick any ready made mesh ("your medieval path") or initially use anything - like a plane. In above case I put plane, attached geom blend component, setup water shader and sticked using *Stick to blended object along normals* button. The distance inner vertices will be put above ground is controlled by *Distance* value. Edges won't be offset with the distance. You can also choose to stick downward which in some cases works more stable when sticked mesh is complex.

Offsetting a mesh when sticking is wise because terrain mesh isn't static – it changes depending on viewing angle, distance and pixelError value – that's why it could "z-fight" with our sticked mesh. To avoid it – put higher offset distance when sticking or, after sticking you can edit your terrain pushing it a bit downward to avoid fighting.

When mesh is sticked it reveals some other options in script:

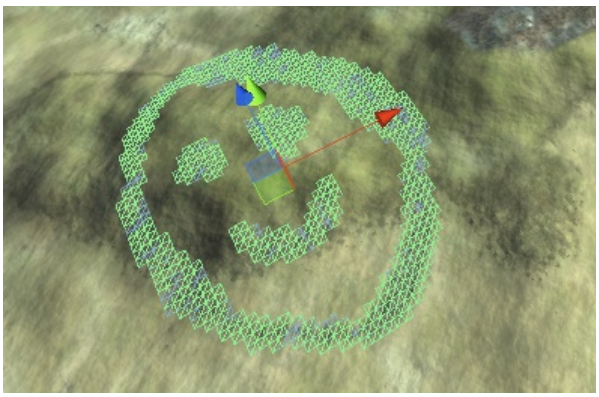


In paint mode we can actually remove existing sticked geometry using *Del tris* option or add new triangles (*Add tris*) which will perfectly match underlying terrain mesh – this helps avoid fighting with terrain at close distance ("painted" new triangles are assured to be right above ground ones).
Subdivision slider helps us to tessellate less or more new geometry. Default 0 value means added triangles are of the size and match underlying terrain triangles which depends on height map resolution.
You should know that RTP provides you with tweaked version of terrain to obj exporter. This script is available from main Unity menu (*Terrain/Export to OBJ by steepness*) and allows you to export whole terrain mesh with selected steepness range so you don't need to paint every cliff manually.

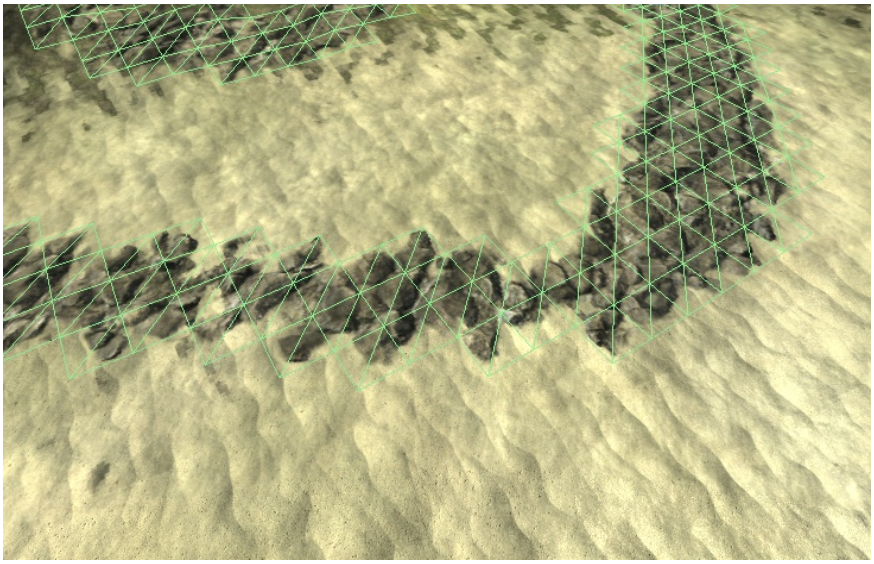
Slope range controls the triangles being added are in selected steepness range – it's useful when you'd like to paint geom blend sticked geometry on some stretched slope (assuming you don't use triplanar). Then you can export created mesh using *Export to OBJ* button. In 3D modeling software you can then unwrap uv coords so they're not stretched any longer and import back to Unity. This way of removing stretching at slopes is used on example scene – look at terrain scene and this steep hill in the corner – introduced to give you an idea how it might work.

Sticked geom blend also uses underlying child object, but it no longer "mimics terrain". It's just transparent and on fully covered parts it does something looking like holes cut, but this is only half a truth. Although it doesn't render to color buffer (we see holes), it writes into depth buffer. As it's drawn first - it actually covers underlying terrain, so it's not processed by pixel shader – large areas covered by sticked geometry and using cheap shader (like simple bumped specular geom blend version for example) would render very fast (although will need additional triangles to be rendered, they won't lift drawcalls higher). This child shader is *Internal/ReliefTerrainBlendBaseCutout*. Keep in mind that for large sticking offsets you'll notice parallax errors – sticked geometry might look "hollow at edges" – it's matter of finding right balance between avoiding surfaces fight and quality by using as small stick *Distance* as possible.

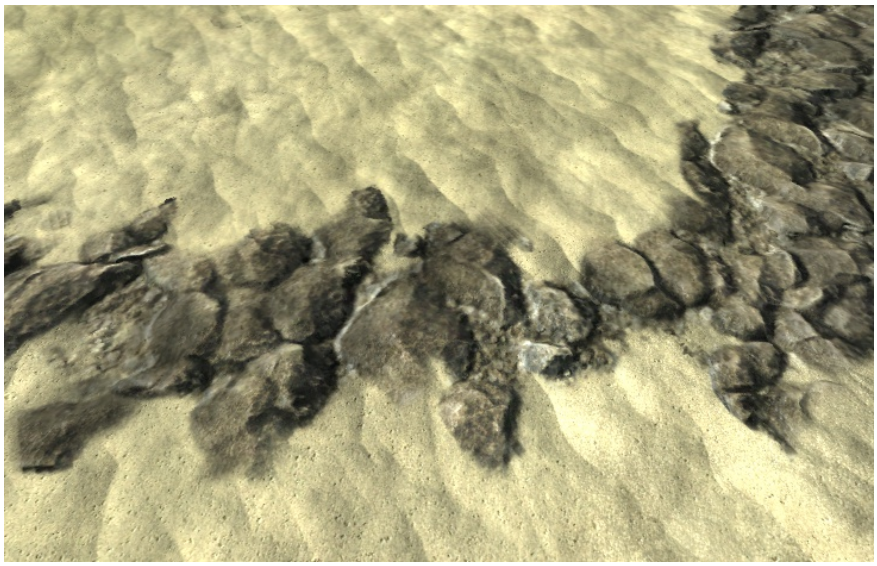
As you may realized – adding some sticked geometry sets allows you to use only minimal number of layers for RTP terrain shader which will be good for performance and then decorate your scene with virtually unlimited number of sticked surfaces that will look just like part of terrain. You don't have to worry anymore how to use best your layers. In case of 4 it's always problem like: "*Should I use grass, sand, rocks and dirt or 2 types of rocks w/o dirt? And what about this superb medieval stones path I'd like to use in the village ?*". Now it's limited only by your imagination how detailed and interesting your terrain surface will be. It can always look like terrain – not a separate object placed on.



Some geom blend geometry painted on the terrain



A close – up



*Shader used - **Relief Pack/GeometryBlend_PM_HB_COMPLEX***

Although it dosen't show all possibilities this shader can do (_COMPLEX suffix), we see at least it can realize heightblend with terrain sand layer (that's why it's marked _HB_ in the name). Cool, isn't it ? I only had to manually replace controlmap and height map combined to make it heightblends with layers 5-8 (because blended object pivot was placed over layers 0-3 and using "sync to terrain" set heightblend with them while sand is 5th layer).

Some of geom blend water shaders can control its wetness with terrain wet mask. Even if terrain doesn't use water feature – like in example scene we've got streams, while terrain doesn't have water. It might be confusing, but actually you can paint wet mask on terrain with water disabled – enable feature under RTP script (not LOD manager), but DON'T go to RTP LOD manager. Then you can paint.
Another option is to use switch in geom blend shader to take water coverage from vertex B channel instead of terrain wet mask (which is baked in special combined texture together with perlin normal and reflection map). You can also ask for AO to be taken from vertex color.

You can edit multiple geom blend objects at once – select them in project / scene view. Put 20 rocks and align them on the ground. Then you can adjust their values together.
RTP3.2b – you can unify material used on the object and combine objects for static batching.

8. Advanced topics

RTP shaders – tweaking features manually

Basically what RTP_LODmanager does is modifying shaders code so you don't have to even look into it. However, if you'd like here is the list of defines that can be found in *RTP_Base.cginc* and *RTP_AddBase.cginc*. They are included into shaders. When you'd like to change shader mode (4/8 layers in one pass) it's advised to not doing this manually as it needs more steps than changing _4LAYERS define. In most cases you don't need to look into shaders anyway (however you might trace below underlined parts which are tunable only here):

```
// layer count switch (undefined in RTP_Base in 8 layers in first pass mode)
#define _4LAYERS

// ATLASING in 4 layers mode to save 4 texture samplers
#define RTP_USE_COLOR_ATLAS

// if you're using this shader on arbitrary mesh you can control splat coverage via vertices
// colors, note that you won't be able to blend objects when VERTEX_COLOR_CONTROL is defined
// this option is available only here (unavailable in LOD manager)
#define VERTEX_COLOR_CONTROL

// to compute far color basing only on global color map
// it's - No detail color at far distance - in LOD manager
#define SIMPLE_FAR

// uv blending
#define RTP_UV_BLEND
// uv blending only at far distance
#define RTP_DISTANCE_ONLY_UV_BLEND
// usage of normals from blended layer at far distance
#define RTP_NORMALS_FOR_REPLACE_UV_BLEND

// super detail perlin normal & additional mult channels
#define RTP_SUPER_DETAIL
#define RTP_SUPER_DTL_MULTS

// dynamic snow feature
#define RTP_SNOW
// layer number taken as snow normal for near distance (for deep snow cover)
// if undefined - dedicated snow normal won't be used
#define RTP_SNW_CHOOSEN_LAYER_NORM_0
// layer number taken as snow color/gloss for near distance
// if undefined - dedicated snow color from detail texture won't be used
#define RTP_SNW_CHOOSEN_LAYER_COLOR_0

// global color map (can't be disabled in LOD manager)
#define COLOR_MAP
// if not defined global color map will be blended (lerp)
#define COLOR_MAP_BLEND_MULTIPLY
// advanced colormap blending per layer
#define ADV_COLOR_MAP_BLENDING

// global normal map (and we will treat normals from mesh as flat (0,1,0))
#define RTP_NORMALGLOBAL

// global trees/shadow map - used with Terrain Composer / World Composer
#define RTP_TREESGLOBAL

// heightblend fake AO
#define RTP_HEIGHTBLEND_AO

// layer emissiveness
#define RTP_EMISSION
```

```
// when wetness is defined and fuild on surface is emissive we can mod its emissiveness by
// output normal (wrinkles of flowing "water")
// below define change the way we treat output normals
// (works fine for "lava" like emissive fuilds)
#define RTP_FUILD_EMISSION_WRAP
// with optional reafractive distortion to emulate hot air turbulence
#define RTP_HOTAIR_EMISSION

// when defined you can see where layers 0-3 overlap layers 4-7 in 8 per pass mode.
// These areas costs higher.
// (note that when RTP_HARD_CROSSPASS is defined you won't see any overlapping areas)
#define RTP_SHOW_OVERLAPPED

// when defined we don't calculate overlapping 0-3 vs 4-7 layers in 8 layers mode,
// but take "higher". It's recommended to use this define for significantly better performance
// undef it only when you really need smooth transitions between overlapping groups
#define RTP_HARD_CROSSPASS
// below setting isn't reflected in LOD manager
// it's only available in RTP_Base.cginc and is turned on by default
// when hard crosspass above is active, we can blend passing edges with global colormap
#define RTP_HIDE_EDGE_HARD_CROSSPASS

// define for harder heightblend edges
#define SHARPEN_HEIGHTBLEND_EDGES_PASS1
#define SHARPEN_HEIGHTBLEND_EDGES_PASS2

// triplanar (works only with _4LAYERS)
#define RTP_TRIPLANAR

// vertical texture
#define RTP_VERTICAL_TEXTURE

// we use wet (can't be used with superdetail as globalnormal texture BA channels are shared)
#define RTP_WETNESS
// water droplets
#define RTP_WET_RIPPLE_TEXTURE
// if defined water won't handle flow nor refractions
#define SIMPLE_WATER

// caustics
#define RTP_CAUSTICS
// when we use caustics and vertical texture - with below defined we will store
// vertical texture and caustics together (RGB - vertical texture, A - caustics)
// to save texture sampler
#define RTP_VERTALPHA_CAUSTICS

// if you don't use extrude reduction in layers properties (equals 0 everywhere)
// you can comment below - POM will run a bit faster
#define USE_EXTRUDE_REDUCTION

// cutting holes functionality (make global colormap alpha channel completely black to cut)
#define RTP_CUT_HOLES


//
// below setting isn't reflected in LOD manager,
// it's only available here (and in RTP_ADDBase.cginc)
// you can use it to control snow coverage from wet mask (special combined texture channel B)
//
#define RTP_SNW_COVERAGE_FROM_WETNESS

// helper for cross layer specularity / IBL / Refl bleeding
#define NOSPEC_BLEED
```

```
//
// in 8 layers mode we can use simpler shading for not overplapped
// (RTP_HARD_CROSSPASS must be defined) 4-7 layers
// available options are:
// RTP_47SHADING_SIMPLE
// RTP_47SHADING_PM
// RTP_47SHADING_POM_LO (POM w/o shadows)
// RTP_47SHADING_POM_MED (POM with hard shadows)
// RTP_47SHADING_POM_HI (POM with soft shadows)
//
#define RTP_47SHADING_PM
```

In AddBase include we've got 2 other options (note that _4LAYERS is also defined in add pass, but you CAN'T specify 8 layers there – it won't work):

```
// in presence of 2 passes we can do heightblend between passes
#define RTP_CROSSPASS_HEIGHTBLEND
```

Not every option works together with each other. When you manipulate shaders manually you have to take care about available resources or shader won't compile.

RTP properties

There might be a situation you'd like to control some RTP properties of terrain runtime. One method is to use Presets and their interpolation features described earlier. Other option is to make it yourself to have more control. Below table describes RTP script properties (internal public variables) in their relation to RTP editor inspector. Then you can modify it from your script and apply changes using Refresh() functions.

The list below represents set of properties that are likely to be changed:

RTP property name	Inspector name / description
Texture2D NormalGlobal	Global normal map
Texture2D TreesGlobal	Global pixel trees/shadow map
Texture2D AmbientEmissiveMapGlobal	Global ambient emissive map
Texture2D ColorGlobal	Global color map
Texture2D BumpGlobalCombined	Special combined texture (RG – perlin, BA – wet mask, reflection map or superdetail mult channels)
Texture2D controlA	Coverage controlmap RGBA texture for layers 0-3
Texture2D controlB	Coverage controlmap RGBA texture for layers 4-7
Texture2D controlC	Coverage controlmap RGBA texture for layers 8-11

They are placed in a ReliefTerrain script instance of a terrain tile. This means that they are set locally for given terrain.

Example usage (C#) for given **go** (GameObject) where RTP terrain is placed:

```
ReliefTerrain rt=go.GetComponent(typeof(ReliefTerrain)) as ReliefTerrain;
rt.ColorGlobal=yourTexture2D_Instance;
rt.RefreshTextures();
```

This will change global color map texture for your terrain tile.

The list below consists of shared values that are common to all terrain tiles that has the same anchor – they are stored in GlobalSettingsHolder object and every terrain tile has the reference for this unique object (unique in scope of terrain objects grouped by their anchors).

RTP property name	Inspector name / description
Settings/Main	
float distance_start	Distance start (close distance)
float distance_transition	Fade length
float RTP_MIP_BIAS	MIP level bias
Color _SpecColor	Specular color
Color rtp_customAmbientCorrection	Ambient correction
float MasterLayerBrightness	Layers detail brightness
float MasterLayerSaturation	Layers detail saturation

Vector4 RTP_LightDefVector (x component)	Negative light power
Vector4 RTP_LightDefVector (y component)	Reflex diffuse softness
Color RTP_ReflexLightDiffuseColor	Reflex light A diffuse
Color RTP_ReflexLightDiffuseColor (alpha)	Reflex light A diffuse strength
Color RTP_ReflexLightDiffuseColor2	Reflex light B diffuse
Color RTP_ReflexLightDiffuseColor2 (alpha)	Reflex light B diffuse strength
Vector4 RTP_LightDefVector (w component)	Reflex specularity
Color RTP_ReflexLightSpecColor	Reflex light specular
Color RTP_ReflexLightSpecColor (alpha)	Specular strength
float RTP_AOsharpness	Fake AO 2 HB sharpness
float RTP_AOamp	Fake AO 2 HB value
float EmissionRefractFiltering	Refraction filtering
float EmissionRefractAnimSpeed	Refraction anim speed
Cubemap _CubemapDiff	Custom cubemap for diffuse IBL
Cubemap _CubemapSpec	Custom cubemap for specular IBL
float TERRAIN_IBL_DiffAO_Damp	Diffuse IBL AO damp
float TERRAIN_IBLRefl_SpecAO_Damp	Spec IBL / Refl AO damp
Settings/UV blend	
float blendMultiplier	Mix Multiplier
Settings/Vertical texture	
float VerticalTextureTiling	Tiling [m]
float VerticalTextureGlobalBumpInfluence	Offset via Perlin normal
Settings/Global maps	
Global color settings	
Vector3 GlobalColorMapBlendValues (x comp)	Near blend
Vector3 GlobalColorMapBlendValues (y comp)	Mid blend
Vector3 GlobalColorMapBlendValues (z comp)	Far blend
float _GlobalColorMapNearMIP	MIP level below far
float GlobalColorMapSaturation	Saturation (near distance)
float GlobalColorMapBrightness	Brightness (near distance)
float GlobalColorMapSaturationFar	Saturation (far distance)
float GlobalColorMapBrightnessFar	Brightness (far distance)
float GlobalColorMapDistortByPerlin	Color map distortion strength using perlin normal
Global normal settings	
float global_normalMap_multiplier	Normal strength
Global trees/shadow settings	
float trees_pixel_distance_start	Trees distance
float trees_pixel_distance_transition	Trees fade length
float trees_pixel_blend_val	Trees blending
float trees_shadow_distance_start	Shadows distance

float trees_shadow_distance_transition	Shadows fade length
float trees_shadow_value	Shadows blending
Global ambient emissive map settings	
float AmbientEmissiveMultiplier	Emission brightness
float AmbientEmissiveRelief	Normal/height mod
float trees_shadow_distance_start	Shadows distance (the same as for pixel trees)
float trees_shadow_distance_transition	Shadows fade length (the same as for pixel trees)
float trees_shadow_value	Shadows blending (the same as for pixel trees)
Settings/Snow	
float _snow_strength	Strength
float _global_color_brightness_to_snow	Global color influence
float _snow_slope_factor	Reduction by slope
float _snow_height_treshold	Height threshold [units]
float _snow_height_transition	Height transition
Color _snow_color	Color
float _snow_metalness	Snow metalness
float _snow_gloss	Snpw smoothness
float _snow_edge_definition	Edges definition
float _snow_deep_factor	Deep factor
Settings/Perlin normal	
float BumpMapGlobalScale	Tiling scale
float rtp_mipoffset_globalnorm	MIP offset
float distance_start_bumpglobal	Distance start
float rtp_perlin_start_val	Starting value
float distance_transition_bumpglobal	Fade length
float _FarNormalDamp	Far normal damp
Settings/Water & caustics	
Water	
float TERRAIN_GlobalWetness	Global wetness
float TERRAIN_WetHeight_Treshold	Height threshold (where water starts in world Y axis)
float TERRAIN_WetHeight_Transition	Height transition
float TERRAIN_WaterSpecularity	Water specularity
float TERRAIN_FlowSpeed	Flow speed
float TERRAIN_FlowScale	Flow tex tiling
float TERRAIN_FlowCycleScale	Flow cycle scale
float TERRAIN_FlowMipOffset	Flow tex filter (real range 0 - 4)
float TERRAIN_mipoffset_flowSpeed	Filter by flow speed (real range 0 - 4)
float TERRAIN_WetDarkening	Water surface darkening
Rain droplets	
float TERRAIN_RainIntensity	Rain intensity

float TERRAIN_WetDropletsStrength	Rain on wet
float TERRAIN_DropletsSpeed	Anim speed
float TERRAIN_RippleScale	Ripple tex tiling
Caustics	
float TERRAIN_CausticsAnimSpeed	Anim speed
Color TERRAIN_CausticsColor	Color
float TERRAIN_CausticsWaterLevel	Water Level (Water Level Ref object has to be none)
float TERRAIN_CausticsWaterLevelByAngle	Water level by slope
float TERRAIN_CausticsWaterShallowFadeLength	Shallow fade length
float TERRAIN_CausticsWaterDeepFadeLength	Deep fade length
float TERRAIN_CausticsTilingScale	Texture tiling
Settings/Superdetail	
float _SuperDetailTiling	Superdetail tiling scale
Layer properties	
Textures	
Texture2D[numLayers] splats	Detail textures for subsequent layers
Texture2D[2] splat_atlases	2 atlases used in 8 layer in one pass mode
Texture2D[numLayers] Bumps	Normal map textures (used in edtior only, then combined)
Texture2D Bump01	Normal map combined for layers 0,1
Texture2D Bump23	Normal map combined for layers 2,3
Texture2D Bump45	Normal map combined for layers 4,5
Texture2D Bump67	Normal map combined for layers 6,7
Texture2D Bump89	Normal map combined for layers 8,9
Texture2D BumpAB	Normal map combined for layers 10,11
Texture2D[numLayers] Heights	Height map textures (used in edtior only, then combined)
Texture2D HeightMap	Height map combined for layers 0-3
Texture2D HeightMap2	Height map combined for layers 4-7
Texture2D HeightMap3	Height map combined for layers 8-11
ProceduralMaterial[numLayers] Substances	Substances (used in editor to grab splats, Bumps, Heights textures)
PBL / IBL properties	
float[numLayers] RTP_metallic	Layer N metalness
float[numLayers] RTP_glossMin	gloss min range
float[numLayers] RTP_glossMax	gloss max range
float[numLayers] FarSpecCorrection	far spec correction
float[numLayers] RTP_DiffFresnel	diffuse scattering
Layer adjustement	
float[numLayers] LayerBrightness	Layer brightness
float[numLayers] LayerSaturation	Layer saturation
float[numLayers] PER_LAYER_HEIGHT_MODIFIER	Extrude reduction

float[numLayers] MIPmult	Far filtering
Emissive features	
float[numLayers] LayerEmission	Amount of layer emissiveness (0..1, 1 – full emissive layer)
Color[numLayers] LayerEmissionColor	Color of layer's emissive glow
float[numLayers] LayerEmissionRefractStrength	Amout of hot air refraction applied o the layer
float[numLayers] LayerEmissionRefractHBedge	Whether refraction should be only present on heightblend edges (1) or for all layer (0)
Global color map	
float[numLayers] GlobalColorPerLayer	Global color map application per layer
float[numLayers] GlobalColorBottom	Low heightmap threshold value
float[numLayers] GlobalColorTop	High heightmap threshold value
float[numLayers] GlobalColorColormapLoSat	Color map saturation for low parts blending
float[numLayers] GlobalColorColormapHiSat	Global map saturation for high parts blending
float[numLayers] GlobalColorLayerLoSat	Detail color saturation low parts blending
float[numLayers] GlobalColorLayerHiSat	Detail color saturation high parts blending
float[numLayers] GlobalColorLoBlend	Amount of blending for low parts
float[numLayers] GlobalColorHiBlend	Amount of blending for high parts
UV blend	
float[numLayers] MixScale	UV blend scale (watch UV blend routing !)
float[numLayers] MixBlend	UV blend val
float[numLayers] MixSaturation	UV blend saturation
float[numLayers] MixBrightness	UV blend brightness
float[numLayers] MixReplace	UV replacement at far distance
Perlin normal	
float[numLayers] _BumpMapGlobalStrength	Perlin normal
Superdetail	
float[numLayers] _SuperDetailStrengthNormal	Superdetail normal
float[numLayers] _SuperDetailStrengthMultA	Superdetail mult A
float[numLayers] _SuperDetailStrengthMultASelfMaskNear	SelfMask near
float[numLayers] _SuperDetailStrengthMultASelfMaskFar	SelfMask far
float[numLayers] _SuperDetailStrengthMultB	Superdetail mult B
float[numLayers] _SuperDetailStrengthMultBSelfMaskNear	SelfMask near
float[numLayers] _SuperDetailStrengthMultBSelfMaskFar	SelfMask far
Vertical texture	
float[numLayers] VerticalTextureStrength	Vertical map strength
Snow	
float[numLayers] _snow_strength_per_layer	Snow strength
Water	

float[numLayers] TERRAIN_LayerWetStrength	Layer wetness
Color[numLayers] TERRAIN_WaterColor	Water color and opacity
float[numLayers] TERRAIN_WaterEmission	Water emission
float[numLayers] TERRAIN_WaterLevel	Water level (real range 0-2)
float[numLayers] TERRAIN_WaterLevelSlopeDamp	Water level slope damp
float[numLayers] TERRAIN_WaterEdge	Water level sharpness (real range 0.5-4)
float[numLayers] TERRAIN_WaterMetallic	Water metalness
float[numLayers] TERRAIN_WaterGloss	Water smoothness
float[numLayers] TERRAIN_Flow	Water flow
float[numLayers] TERRAIN_Refraction	Water refraction
float[numLayers] TERRAIN_WetGloss	Wet smoothness
float[numLayers] TERRAIN_WetFlow	Wet flow
float[numLayers] TERRAIN_WetRefraction	Wet refraction factor
float[numLayers] TERRAIN_WaterGlossDamper	Hi-freq / distance gloss damper

Example usage (C#) for any **go** (GameObject) where RTP terrain is placed:

```
ReliefTerrain rt=go.GetComponent(typeof(ReliefTerrain)) as ReliefTerrain;
rt.globalSettingsHolder._snow_strength=aValueForSnow;
rt.globalSettingsHolder.RefreshAll();
```

This will change global snow strength (hence its coverage). You can control runtime seasonal / weather conditions this way.

Changing some property per layer might be like this (modifying water level per 4th layer):

```
ReliefTerrain rt=go.GetComponent(typeof(ReliefTerrain)) as ReliefTerrain;
rt.globalSettingsHolder.TERRAIN_WaterLevel[3]=2;
rt.globalSettingsHolder.Refresh();
```

This will set maximum water level for this layer (remember that actual result depends also on global wetness, layer wetness and wetness mask).

Custom geom blend compliant shaders

Actually I put such info in a separate txt info file in package, but here I can describe that, too (for people that don't have RTP3 ...yet). In a basic situation to make shader GeomBlend compliant follow steps below:

1. find line "#pragma surface..."
2. put "decal:blend" at the end of line above (if not your shader doesn't have it already)
3. find "struct Input {"
4. add "fixed4 color:COLOR;" at the end of struct (before "}")
5. find "void surf" function
6. put

o.Alpha=1-IN.color.a;

at the end of it

Refer to example GeometryBlend_BumpedDiffuseOverlay.shader
(made from **BumpedDiffuseOverlay.shader**) - I put comments there that reflect above instructions.

As one may see – we're using vertex color A component to make blending, so basically you can't make any surface shader GeomBlend compliant, but only these which doesn't use vertex color A.

To know more about making GeomBlend shaders (for people with at least basic shader coding knownledge) – refer to the code of all shaders starting with "GeomBlend_"

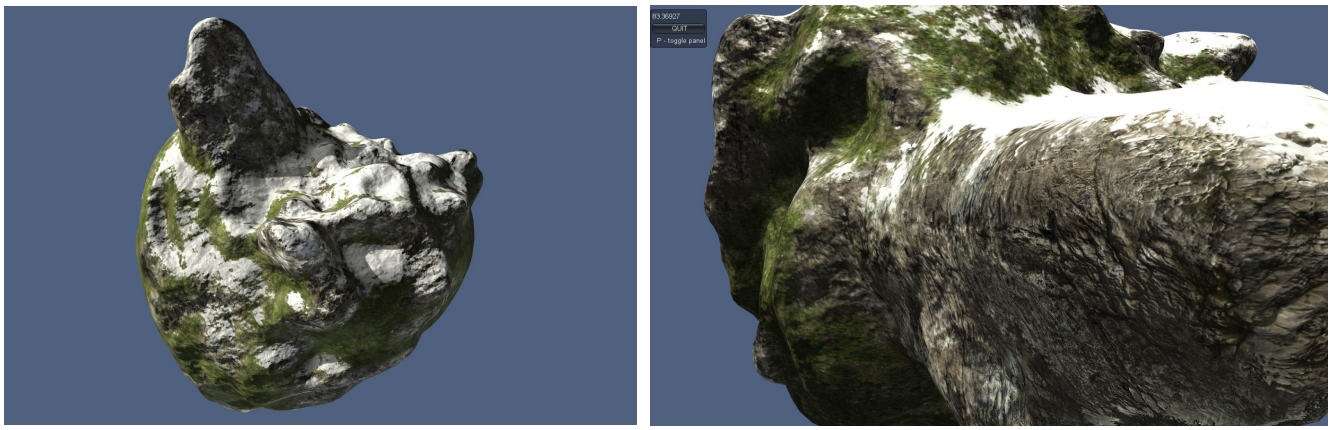
One of shaders in this folder (GeometryBlend_BumpedDiffuseOverlay.shader) is GeomBlend compliant version of a shader I've made for Michael O. aka Killstar aka Manufactura 4K – so – you might use it to blend his cool models with RTP.

9. Standalone RTP shaders

for voxel terrains, planets and any objects that might use RTP shading

As RTP shaders manage triplanar shading quite good and we actually can tell shader to use vertex colors for coverage instead of control splatmaps, it's good to use RTP shader with an existing voxel terrains that controls detail layer coverage this way (by vertex colors). You'll find the shader in *ReliefPack/Shaders/ReliefTerrain/VertexControl* folder. It's *ReliefTerrainPMTriplanarStandalone.shader*. Remember to put also *ReliefTerrainVertexBlendTriplanar.cs* script on the object.

This is how it looks like on the example scene (*ShaderOnTriplanarMesh.unity*):



Although it's not voxel terrain, but a mesh made in Blender it behaves the same. Although color can be taken from vertices colors explicitly (using for example free vertex painter tool from AssetStore, they are also defined by solutions like TerraVol), you can also ask standalone shader (using #define switch) to derive coverage from global normal. This is default option for this example scene and when you import RTP package. Such mesh does not need ANY additional data except for vertices and normals. This way such standalone shader can be used on any mesh you'd like. It's almost as powerful as RTP terrain shader itself. It can use PM shading and most RTP features, although in the example I haven't used all options, but basic like perlin normal, UV blend and dynamic snow.

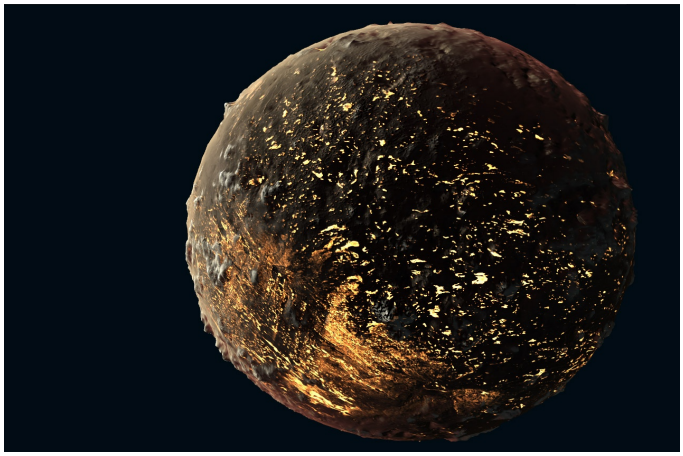
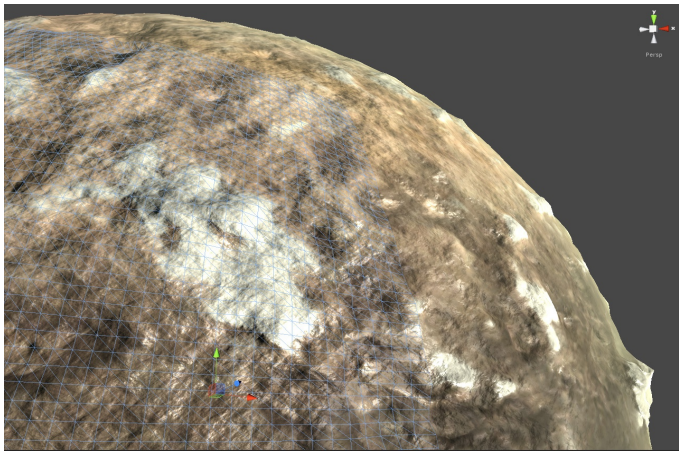
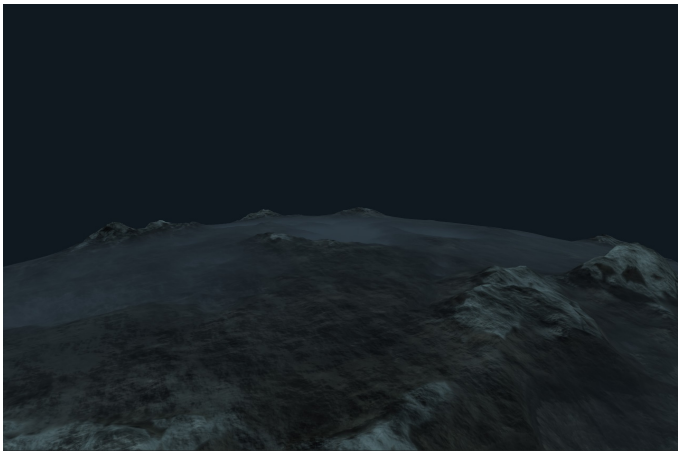
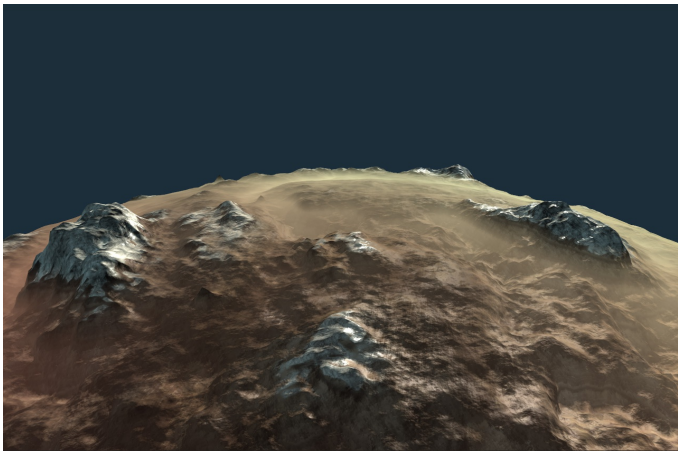
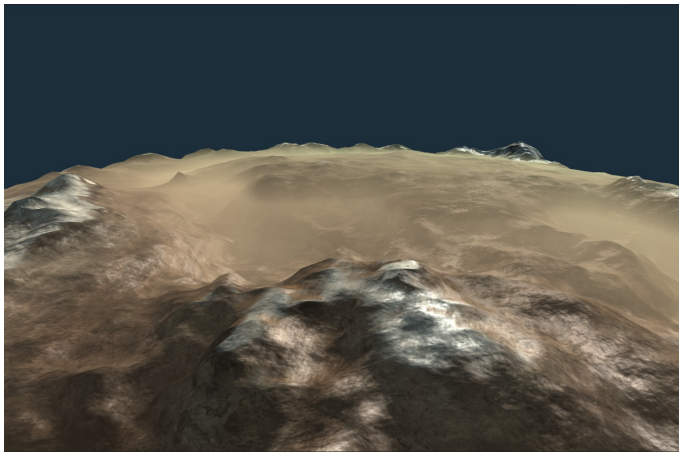
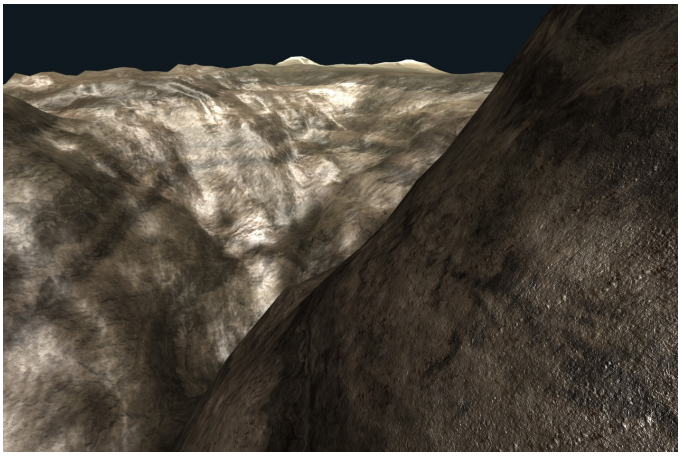
Voxel terrain usage is only an option. Realize how nicely looking could be your asteroids...

Shader features are all logically grouped in foldouts similar to RTP inspector.

Talking about space objects, RTP has powerful planet shader that has been specially prepared for usage with Planetary Terrain package. For your planet you'll need to adjust options used. You've got a mesh generated by PlanetaryTerrain in *ShaderOnPlanetModel.unity* example scene. Preconfigured shader / material used there is *ReliefTerrainTriplanarPlanet.shader* to be found in RTP shaders folder.

It uses many RTP features (global color map, global normal map for far distance, triplanar, parallax, snow, perlin, UV blend). One of main features that are specific to planet shader only is custom layered fog which is properly distributed over planet surface only and handles atmospheric scattering – you can choose different color of fog day/dusk/night time. You can also use emission mask (A channel of global color map – useful for "city lights" at night parts of planet).

Here are example screenshots presenting how good a planet can look at both – orbit distance and very close to the ground:

10. RTP brief glossary

4/8 layers mode
RTP can render either 4 or 8 layers in a render pass. 8 layers mode means that we render 8 of them at once (in one pass).

AO
Ambient Occlusion – in RTP we can use option that mimic AO in Settings/Main (but it's fake).

Arbitrary mesh
A mesh which is used instead of Unity's default terrain (which is mesh created dynamically out of height map).

Caustics
Effect made from a texture that is composed 4 times together using UV offset animation. Simulates light streaks on the bottom of water reservoirs (strongly light and shallow enough for caustics to be visible).

Combined texture
A texture used in RTP that is composed out of a few others. For example bump map combined in RTP stores information of normal maps from two subsequent layers. Special combined texture stores information for perlin normal map, wet mask and reflection map.

Control map
RGBA texture that controls coverage of 4 subsequent layers over terrain.

Detail texture
A texture used as main layer color for close distance. Detail textures are tiled over whole terrain. Tile size is set in Settings/Main.

Geom blend
Technique of blending a separate mesh object (like rock, tree, etc.) with terrain without visible seams where terrain and object intersects.

Global color map
A texture spanned over whole terrain to provide additional color (by default multiplied with detail texture color).

Global normal map
A normal map spanned over whole terrain to provide normals information instead of these taken from mesh (global normal map overwrites mesh normals).

HDR
High dynamic range. Indication that we use input/output render data (input textures, parameters and render output) of higher range and resolution. In opposite to LDR (low dynamic range) which works in range 0..1 and has 256 distinct parameter levels, HDR represents higher resolution and range. RTP can use HDR cubemap input (encoded RGBM – 0..8 range and higher resolution) and renders specular output in range ca. 0..50.

Heightblending
Blending method that takes into account not only layer coverage from control map but also a height texture associated with a separate layer. This way for 2 layers that have the same coverage value in control map more visible will be the one with „higher" (brighter) height map color.

Height map

Either a texture that controls terrain mesh height (defining terrain shape) or a texture that is associated with a layer for heightblending (defining detail bumps height).

IBL

Image based lighting. Method of taking light information coming to the rendered object from all surrounding environment using cubemaps that represents it. Unity makes usage of this when dealing with ambient diffuse light (from skyboxes) and for reflection probes.

LOD manager (RTP LOD manager)

_RTP_LODmanager game object added automatically to the scene when ReliefTerrain script is present on any terrain object (or arbitrary mesh). Script on LOD manger object provides options to adjust RTP shader features.

MIP

Refer to http://en.wikipedia.org/wiki/Mipmap.

Mult channels

These are related to superdetail. You can use 2 grayscale textures that will be multiplied with your detail textures.

Overlapping layers

When a layer coverage is %100 in splat control map the layer is considered to be not overlapped with any other.

PBL / PBR

Physically based lighting/rendering. Render methods that are meant to bring your models look more realistic using physical parameters and effects that we observe in real world (like energy conservation law, fresnel effect).

Perlin normal

Perlin noise grayscale texture turned into normal map. This is used as additional normal (bump) map put on terrain to complement layer normal map.

PM

Parallax Mapping

POM

Parallax Occlusion Mapping

Simple water

When chosen in LOD manager – water on the terrain won't use flowing capabilities nor rain droplets.

Splat control map

Control map

Stretching

As terrain is mapped top-planar stretching will occur on very steep slopes (cliffs).

Superdetail

Perlin normal map used up close to the camera to boost surface detail.

UV blend

Additional color mixed with detail texture color. UV blend value is also taken from a detail texture but at different scale. It helps removing noticeable tiling pattens on distant large terrain areas.

Vertical texturing
A texture mapped along world Y axis (V), it doesn't use horizontal (U) texture coordinate, so it's prone to stretching.

Wet mask
Grayscale texture used to control water coverage over terrain.

11. RTP3.2 change log

RTP3.2 change log

For users familiar with previous versions of RTP that already got thru this docs before it will be much quicker to follow introduced changes alone. They are as follows:

1. When DX11 platform is active in LOD manager we've got tessellation options available. They are described in this video. As tessellation is DX11 feature only, all other platforms are then unavailable in LOD manager. You'll find dedicated standalone triplanar shader that has tessellation enabled (for smoothening meshes, voxel stuff). Tessellation can't be introduced in planet shader (might be if you can compromise global normalmap though).

2. Details are tiled using world position coordinates. So - "A" aligning buttons are not necessary now and are removed in v3.2

3. In Unity5 you can't use 8 layers mode when not using materials. This is still possible for one terrain in Unity4 though.

4. Aniso filtering works by default in non atlased modes (4 layers w/o atlasing enabled). So you can have crisp detail color/normal textures even in triplanar.

5. In Unity5 we still use RTP's physical based shading pipeline which is quite OK. However if you'd like to use new Unity's PBR you can do it in new deferred mode. Remember that it doesn't handle any decal/addpasses to be rendered in deferred. So – addpass will be rendered in forward (always). Gloss (smoothness) value is what it is in RTP. Specular Color which affects reflectiveness and Fresnel behavior is taken from RTP's specular output (one spec color for all terrain in Settings/Main that can be influenced by a layer's spec multiplier and tint controller). You can improve blending of RTP's self-shadowing/AO with Unity's shadows in new deferred by using dedicated shader (can be selected in Unity5 settings). The shader is placed in Internal-DeferredShadingRTPU5.zip

6. You'll find water shaders that uses Unity5 PBR (bonus shaders and geom blend ones). All shaders are unipped now (In U4.5 their compilation don't take that long so we don't need zipped stuff when importing).

7. Unity5 – you can enjoy dynamic lightmaps (realtime dynamic global illumination). Baked ones don't work much good for terrains (still – U5 beta 19). Surface shaders don't handle them right in current beta either.

8. Fog selector is not available in LOD manager under Unity5. Fog is handled by multi compile variant and switching between fog models is integrated back with the Unity engine.

RTP3.2b change log

1. Shaders in package come in precompiled state to use 8 layers in 4+4 setup. So – example scenes works out of the box. When you make your own scene it's enough to attach RTP component to an object (terrain/mesh), then refresh to add LOD manager object to the scene, then refresh again in RTP component inspector.

2. MIP Bias slider available in Settings/Main was inverted in RTP3.2. Now it works again as expected (negative values means more "crisp" look).

3. In Unity5 (from beta21) using materials and not using them seems having no impact on CPU load, so RTP always uses materials and you can't select non material mode.

4. When using tessellation heitghtmaps on meshes there is new Yoffset parameter introduced to modify placemend of tessellated terrain in world Y axis.

5. Geometry blend feature and its script have been improved. Now it works better (many small fixes introduced) and can be used with tessellated terrains. When you paint over tessellated terrain you might need enlarge subdivisions level though. If tessellated detail displacement is used remember to reduce detail extrusion locally under blended & sticked objects (can be done from geom blend paint inspector and from terrain global colormap painting inspector). Otherwise you can expect underlying detail bumps might clash with sticked geometry.

When you select multiple geom blend objects you can now unify their materials (to use the same). That's wise for batching, but make sure materials actually uses the same parameters and textures ! Materials that use different shaders are not unified. For multiple selection you can also combine your objects for static batching (thus – it's Unity Pro option only). This way you make sure they are groupped right way (the same mesh for underlying and overlying object gives no z-fight issues anymore).

6. For users that don't care about specularity and PBL there is new option available in LOD manager

RTP3.3 change log

1. U5.3 compatibility (tessellation for Macs)

2. Simplified features list - cropped a lot of stuff from LOD manager that's not necessary anymore

3. Unity PBR lighting model. Like in UBER - you've got gloss min/gloss max controller + one additional slider to control glossiness (adding/subtracting) at distance. This means forward and deferred it looks the same.

4. Matched UBER snow model (with snowflakes dissolve mask) + glitter

5. gloss baked feature - made it a lot simplier using asset texture importer (like in UBER), no need for additional objects and all this mess around it.

6. Bugs that has been reported should be now fixed.

Removed:
1. Custom RTP PBR model (mostly metallic - like on Unity terrain shader)

2. Complementary lighting (ambient lighting from Unity is like diffuse IBL using SH9 samplig so no need for this option)

3. custom reflections, IBL features

4. LOD manager platform selector and other options not used anymore.

12. Troubleshooting

Detail normal maps don't work

Check if they are set to *Normal* in *Texture Type* – texture settings inspector. Check in Combined textures/normal maps if they are present and look similar to those made from default RTP textures. In linear color space check ***Bypass sRGB Sampling*** in texture properties for all normal maps and height maps.

My terrain self-shadowing doesn't react realtime to light direction in deferred

Shaders uses light position/direction. In forward rendering path it's done automatically, however in deferred there is problem providing _WorldSpaceLightPos0 uniform variable into shader. When you light your scene in deferred, attach ***Scripts/Common/ReliefShaders_applyLightForDeferred.cs*** script to your main light which will fix light direction issue. You'll also need to use this script when using standalone RTP shaders (for example on any mesh, voxel terrains or planet).

My terrain detail tiling doesn't work like expected

RTP can handle multiple terrains on one scene, but they all should have the same size.

My terrain is all very glossy

Check your detail textures alpha channel – they hold glossiness information. If they don't have alpha channel it's considered to be white (full gloss). Check if your terrain isn't all wet and adjust it (water coverage and wet glossiness). Check ***far gloss correction*** param set per layer (make it lower or zero).

I change feature in RTP LOD manager, but see no difference

After adjusting features in manager you should recompile shaders (green button) to get effects.

My textures looks blurry, even at close distance

Try to use MIP bias offset in Settings/Main. The reason might be you're using textures of different sizes (for example your detail texture for 1st layer is 1024x1024, and for 2nd – 512x512). Also - take into account, that RTP doesn't use anisotropic filtering, so for far distance and small viewing angles blur may be present on detail textures.

At the edges where 2 layers blend each other I see weird dark stripe

It's Settings/Main – fake AO. For layers that has similar colors (two types of grass for example) it might not look good – there you can reduce the effect.

I'm changing fog in Render Settings, but terrain doesn't react to this

RTP need to refresh because it uses it's own (exponential only) fog implementation. To refresh refer to this tip.

I see only plain gray at far distance

You probably have *No detail color at far distance* option used in LOD manager and the same time don't have global color map defined.

I'm using bonus RTP snow shader w/o terrain and I can't make it working

It's because RTP terrain script sets snow params globally (so it syncs snow on your objects with snow on terrain). You have to manage global snow properties yourself – refer to RTP properties – snow section. Then find it in GlobalSettingsHolder.cs – Refresh() function where RTP variables are injected into shader (note that shader variable names sometimes differ from variable names in GlobalSettingsHolder). Set global shader values in your script. For example:

Shader.SetGlobalFloat("rtp_snow_strength", 1);

will set global value of snow strength (note that variable name in *GlobalSettingsHolder* is *_snow_strength* while name in shader is *rtp_snow_strength*).

My geom blend object jitters looking like z-fight of underlying black object against my mesh

Every geom blend object is rendered as 2 separate objects. I managed it this way instead of trying to use one mesh and 2 passes because now we can blend mesh vs. mesh. To be able to select underlying child object - called **RTP_blend_underlying** you'll have to set checkbox inside geom blend script inspector (*Enable child selection* at the bottom of inspector). Now you'll be able to select child object when you unfold hierarchy of main object.

Child object is rendered with shader that mimics terrain - it has texture coords, normals and tangents borrowed from terrain laying just under child object vertices. I don't use normal/tangent blending via vertices colors, only alpha of vertex color is used. Areas where blended terrain textures are not visible are skipped (so they're black and that's what you see black jittering on your object). This optimizes much rendering. This way we're sure that this child object looks 100% the same as terrain it mimics.

Main object is just transparent (surface shader decal:blend directive) one applied on the surface of child described. Theoretically main and child are exactly the same meshes (having same vertices) , but 2 issues might arise:
1. Sometimes child / main geom blend objects are merged in process of batching. After you run it - check if mesh of problematic object have been batched (merged togeter with another mesh) - it can be seen in MeshFilter inspector of an object in run mode. At this point child/parent object might be not exactly the same. Batching introduces some small differences in vertex positions in world space. **To get rid of this** - don't select geom blended objects to be batched automatically (set off batching in "static" property of gameObject) or make sure that child objects and geom blend objects are batched constituting exactly the same set of meshes (it's the case when using **Prepare static batch** button when multiple geom blend objects are selected).

2. In Open GL we've got (sometimes and I can't say why and when it occurs) even bigger problem. Even exactly the same mesh rendered in 2 passes can be subject of such z-fight issue. To get rid of this in Open GL you can offset a bit vertices positions so that we make sure main geom blend object is always rendered over **RTP_blend_underlying**. This can be done inside shaders where I put "Offset " (search for this phrase with following space character) directives. As for now they're commented, but can be used by users who can't solve z-fight problems another way. Offsetting meshes in shader just works, but has its cost. It might introduce another artifacts (not that dramatic like z-fighting) at steep angles and at the edges of objects.

Parallax effects not working...

First - make sure that you've compiled shaders using at least PM. Then check if current LOD level used in RTP LOD manager is at least PM. Then look at Combined textures/Height maps to make sure they are well prepared out of your source height maps placed in layer properties. Make sure that your global color map alpha channel isn't black for parts of terrain you'd like parallax effects to work.
Also - too large triangles of your mesh can cause parallax effects might be distorted.

I provided RTP with my own control maps but they don't work

They have to be normalized. So R+G+B+A value must sum to 1 - in case of 4 layers. In case of more layers sum of all components from all splat control maps must sum to 1. For more info look at control maps section.

My height map seems to be flipped comparing to rest of maps I'm trying to use

WorldMachine export height map flipped. Look at example WM project included in RTP package to see how I resolved it there.

I can't use default Unity terrain shaders and RTP together

In Unity3 it's impossible as RTP shaders replaces built-in ones. In Unity4 you can use different terrain shaders for different terrain tiles when you check *Use U4 terrain materials* in RTP LOD manager. Then for separate terrain tiles you can specify material with a shader you like.

Making terrain with RTP setup a prefab I lost references

Save all textures used (combined, global color map if modified, end so on) before you try to make it prefab.

I get very high CPU hit on my scene using RTP shaders

If you use terrain materials an have many terrain tiles your CPU hit will be quite big regardless of using RTP or not. To make it better – turn off *Use U4 terrain materials* in RTP LOD manager.

I get very high GPU hit on my scene using RTP shaders

In case of high GPU consumption it depends on GPU performance and RTP features used. Check your texture sizes. For example using too big textures (like 2048x2048 height maps with POM self shadowed) is often a bad idea when you'd like your game works reasonably fast. Most expensive GPU options are: POM self shadowed, triplanar, flowing water with rain droplets. For 8 layers mode try to use hard crosspass feature in LOD manager in increase performance.

Can't get global color map working

If you checked other options - check if it's not damped per layer (in layer properties)

My geom blend object works bad – I can't control its wetness or glossiness

Look into shader code where some settings may need to be set. Water geom blend shaders can have water coverage controlled by terrain wet mask (so they are synced to weather conditions on underlying terrain) or by vertex colors.

Where my geom blend object touches terrain it looks weird

Avoid blending mesh polys that are orthogonal o terrain – "flatten" then first. Sometimes parallax effects looks bad at blended edges – use extrude height painting tool to remove them.

My global color map blends different way in linear color space

In gamma color space RGB (128,128,128) gives gray and in sampler is evaluated as (0.5, 0.5, 0.5). This means such color is neutral in terms of blending. In linear space this pixel color will be evaluated as something a bit higher than 0.5. To make it the same as in gamma – set sampled texture „*Bypass sRGB sampling*" checkbox in its settings.

After shader recompilation in LOD manager it seems not to be used

In most cases RTP LOD manager changes *RTP_Base.cginc* and *RTP_AddBase.cginc*. They're include files for actual shaders. Unity should take care about shader recompilation in case any of its include file has been changed. However – what I experienced – sometimes it doesn't work. If you're not sure an RTP shader has been recompiled – find it in Shaders/ReliefTerrain folder and Reimport (RightClick+option from context menu) problematic shader manually.

Terrain detail tiling not working properly

Make sure all your terrain tiles are of the same size. At this moment RTP doesn't handle multiple terrain setup with different terrain sizes per scene.

After recompiling RTP shaders in LOD manager I see my terrain pink

Your shaders haven't compiled properly. Check if you use too many features that needs their own texture. LOD manager should warn you in hint box that shader won't compile with too many features enabled together. Otherwise – contact me reporting compilation bug (provide me with exact information of the LOD manager features that you're trying to achieve).

Using geom blend water shaders, but I actually can't get it covered by water

Check in the shader code if water coverage is taken from special combined texture (B component) or is driven by vertex color (color B component). It's set using define keyword at the beginning of shader file. In case you're using coverage from texture – take care wet mask inside is right. To make sure its synchronized with terrain use „Sync to terrain" button in geom blend inspector. Then you can paint wetmask on terrain and your geom blend object will get synchronized water coverage from it. You could also check global wetness value in RTP Settings/Water&Caustics.

Recompile shaders button seems to have no effect and terrain looks "default"

Make sure RTP is placed in default install location (Assets/ReliefPack). Don't move it to different location in your project folder after installation.

I'm using standalone shaders and but their look is odd

You need to consider that standalone shaders parameters are set via material, so parameters set are not checked and not constrained like it's the case when RTP script controls it. If, for example you set perlin normal tiling to unreal value like 100 in RTP standalone shader material property, you'll see bad texture jittering then. Perlin tiling should be set to value quite less than 1. Beside of taking care about right parameter values, in most situations standalone shaders need 2 scripts attached to game object using them: ReliefTerrainVertexBlendTriplanar.cs (accessible from Component menu as Relief Terrain/Helpers/MIP solver for standalone shaders) and ReliefShaders_applyLightForDeferred.cs. First one applies right MIP filtering for textures used in shader (the same limitation as on RTP script handled

detail textures – should be all of the same size). Second one provides shader with light location that's necessary for deferred.

I'm trying to use RTP planet shader version in deferred, but it looks bad.

For planet it's not possible to use deferred and specularity unless you install 3rd party PBL shading solution for deferred like Lux.

My terrain looks very bad at distance (rectangular mesh patches shaded like simple diffuse)

When you don't use terrain materials (check LOD manager for this option) terrain will be shaded using simple diffuse shader beyond basemap distance (Unity terrain script – settings/Base Map Dist.) By default it can be set up to 2000 units, but it's often not enough when terrain is large. RTP set this value very high (regardless of what you see as Base Map Dist.), but when you are in Unity settings this value will be limited back to 2000. You should simply leave Unity terrain settings to make it back working. Refreshing shader values (ReliefTerrain script – Settings/Main/RefreshAll button).

When I use spot lights on terrain they don't work properly

Uncheck "No forward add" in LOD manager and recompile shaders.

I'm trying to play with PBL specularity settings, but they seems not working at all

Check if your terrain isn't covered by water which takes precendence over "dry" layer properties.

I'm playing with settings in ReliefTerrain script inspector but I see it influences only one terrain object rather than all scene

When *Use U4 terrain materials* in LOD manager is set you'll see changes only for active terrain tile. To apply it for all go to Settings/Main and click Refresh all button. This will also apply changes to all geom blend objects and their child meshes that mimic underlying terrain.

Another factor is terrain objects grouping. When you put terrain in different place in hierarchy (lets say – terrain A and terrain B has different anchor game objects) they will use different GlobalSettingsHolder object that stores RTP shader state, textures and other parameters independent.

Copyright info

You may freely modify/adopt shaders & scripts in the package, but remember to not redistribute them alone nor exposing source code / ideas to the public without prior author's permission.

Graphical assets in package are free to use in your projects. They were free (with reference where they came from) or made basing on public domain resources.

Wish you make insanely good looking games using RTP

Tomasz Stobierski