Release Info - V1.0 (5 Dec 2012)

Thank you for buying RSP. The most comprehensive set of various parallax occlusion mapping techniques available. You will be able to not only experience classic steep parallax algorithms (well optimized) which are one of the most requested shaders nowadays, but also brand new, fast, "pioneer" approaches for relieving objects surface without need to render hi-poly geometries. Together with twin product RTP (Relief Terrain Pack), you can establish new level of detailed surface visualisation in Unity. Dare to compare RSP with similar products available. Most RSP shaders are full of extra features like smooth self-shadowing (as a standard), uv rotation, handling any lightmapping (not only directional lightmaps), some of them fast ambient occlusion too. Another important advantage is that RSP shaders carefully treat mip mapping and uv discontinuities caused by POM texture offseting while it's not the case in many other shader releases. Every shader is simply adjustable thru shader defines - please refer to any shader code where in-depth explanations are placed in comments. This is the best way to tailor shaders to your needs while not every option is always necessary. The simplier shader code is the faster it works.

As a bonus, RSP has replacement for RTP geometry blending shader which will allow you to render blended mesh using POM shading (instead of default bumped specular).

Contents

RSP is logically split into following shader groups:

- **Procedural relief** - very fast shaders that don't use heightmaps, they calculate POM effects in one step, analytically;

- **Distance map relief** – another brand new (author's) technique. We use distance maps which allow us to extrude surfaces in vertical direction at texel (max 256x256) precision. Only 2-3 search steps are usually needed to completely (and always 99.99% precisely) resolve POM calculations;

- **Extrude map relief** – technique close to classic POM shading, but we assume that extrusion takes 2 states only (ceiling or hole/floor). It allows extruding surfaces vertically which can be useful in many situations (and original POM algorithms can't handle it properly without modifications). Such shaders can render sidewalls of extruded surfaces which gain real 3d depth silhouette when viewed from side direction;

- **Steep Parallax mapping** – classic POM technique. It doesn't use cone maps, just pure heightmap, but is well optimized (dynamic number of steps with fast secant/interval ray hitpoint refinement). In addition – the feature you're probably waiting for a long time since first relief shaders appeared on Unity's forum – here is shader version available which can handle real silhouettes. No more constant values for curvature. It's now baked into vertices colors by dedicated tool script;
- **Steep Parallax Volume** – sounds mysterious, but this is special version of steep parallax mapping that allows you to handle meshes constituting real 3d volumes (might be viewed "from the side"). This is better known as lo-poly 2 hi-poly reconstruction (presented under this name at the forum). Think of it as of method for rendering lo-poly models that uses heightmap for mimic very hi-poly detail geometry. As this is quite new approach we couldn't say to what extent it could be used. Tests are however very promising. Fragment shader used is reasonably fast and polycount reduction in many cases can be as high as hundred to one (like in example "king kong faces" scene of the package).

Look at example scenes that will give you an idea how to use the product. Although system itself *doesn't use* any Unity PRO feature, example scenes uses fullscreen post FXes, so you may not be able to run it on indie installation.

### Usage

**Procedural relief**

As mentioned above this is one of the fastest shaders group. Most of them uses similar set of features like distance (fading calculations as camera gets far from object), extrude height and so on. The simpliest way to feel it is just playing with material settings. Depending on geometry used and parameters you can get a whole bunch of results – use your imagination. Play with sliders, relief tiling (Relief Transform param) to get surprised how many variations are hidden somewhere over here. *Sidewall Texture Offset* param is useful to remove texture stretching on extruded parts (inner sidewalls). Of course it's dedicated set. It's not intended for "bumpy rocks" renderings, but architectural, technical or sci-fi productions could benefit the most from this set of shaders:

Here is an example material which use "Accordion Cells 1D" (in attached scene) to visualise wooden paneling with exact 3d grooves with self-shadowing + AO realised by shader itself:
Distance map relief
This group of shaders uses distance map to compute texture offset (position in tangent space). Just prepare B&W (or grayscale) texture up to 256x256. Black (grayscale<128) pixels will be considered for extrusion. Then run dedicated tool Window/Relief Tools/Prepare Texture for DistanceMap shaders to convert it into RGBA32 texture (w/o mips – they don't apply the algorithm). You might optionally use normal map, but if you'd like your extruded surfaces to be just flat, you can disable it in shader code defines section. There are situations that extruded parts might look too synthetic (like brick wall grooves). For this reason there is additional feature added – UV Displacement map – which is used for perturbing surface of object (esp. around grooves). Refer to brickwall materials included to get idea how it works and how it could be used:

It's important to look at shaders defines section at the beginning of their code to adjust params/features you need (you might also need to uncomment material properties used for certain feature). Not every possible configuration is included (shaders number would be impractically high), but only a few for reference. Just go and customize it.

One of the most spectacular application of distance map shaders is space station example scene. It has only small number of polygons but looks like very detailed hi-poly model. Moreover - it's really fast as no heavy POM calculations are needed. It's so fast that even self-shadowing doesn't hurt. It seems to be perfect solution for sci-fi projects. Space station shader has been customized to handle emissive lighting on some parts depending on world space position (this way station/satellites parts glow on their surface when they're in shadow). All these shaders are rather simple to tweak this way. They calculate texture offset, then apply colors / normal maps / detail maps, in the end we apply self-shadowing. Thus finding place where to put additional features shouldn't be difficult even for not experienced shader programmer.
**Extrude map relief**

As distance map relief shaders can handle quantized (cubic) extrusion only, Extrude map relief shaders have been introduced. They don't use distance maps, but height maps alone. Heightmaps are still considered to take two states only so that extrusion will be vertical to the surface (like in distance map shaders). Ray marching algorithms are used so these shaders are not that fast as previous. Yet well optimized can be quite performant as long as "empty spaces" inside mesh in viewing direction are not too large. General rule is that the longer distance ray has to travel inside tangent space, the more steps are needed to precisely resolve texture offset. Many steps need many texture lookups and shader drops down the speed. In case ray hasn't hit surface in maximum number of declared steps we'll see effect of "flatten" extrusion, but it's often acceptable at steep viewing angles. The most important factors of performance and quality are number of search steps and number of binary search steps together with search precision. Whole process is like this:

- Number of steps is maximum number of steps allowed that ray marches to "pierce" extruded surface. When it hit surface next linear steps are conditionally skipped.
- Next resolver performs so called binary search that is intended to finetune solution. In case initial search steps gave no results, binary search takes the job and travels further in the ray direction. Actual binary refinement starts as soon as first piercing is detected (or floor/ceiling reached). Notice that number of steps (LINEAR STEPS) is maximum value and when look at object "from the top" only 1-3 steps will be probably executed and rest will be skipped. BINARY SEARCH STEPS however are always all executed. This way maximum number of steps in pessimistic case is LINEAR STEPS+BINARY SEARCH STEPS. In optimistic case number of steps performed is BINARY SEARCH STEPS + 1.
- Search steps length is defined in texels (SEARCH PRECISION parameter) of heightmap. For heightmaps with smooth shapes to be extruded, search precision might be bigger, but if we need high frequency details to be properly visualised, keep search precision lower. Remember that taking it too small (very short steps) improves extruded edges, but leads to waste of GPU power as you'll need to set very high number of search steps. Always test your objects at distances they will be placed relative to camera. There is no need to make them "super" precise if your objects are relatively small or far from observer. Note that shader itself adjusts search step length internally to sufficient value (so it might be longer for top viewing angles).
- There is also an optimization introduced which skips any ray marching when ray touches surface in first step (ceiling condition – the most optimistic case). In this situation no parallax offset is calculated. When you have heightmap with "holes" that cover relatively small area of whole surface things might work quite fast.

You have to deliver height maps in Alpha8 format (feature exposed in advanced texture options only). Also try to keep it not too big. In most cases 256x256 is enough so that the whole texture can be put in cache of almost every SM3.0 compatible graphic card, raising considerably shader speed.

Shaders of Extrude map relief group are sorted in 3 sub-cathegories:

- **Regular** – shaders that render floor (suitable for 3d volume depth extrusion)
- **Hollow** – shaders that cut floor (suitable for 3d volume depth extrusion)
- **2Sided** – shaders that render backface geometry

Refer to example scene materials and shaders:
Regular and Hollow shaders are suitable for 3d extrusion. This means that you can define flat, open surface, apply such material to it then put ExtrudeHeight script on the object. This script will extrude surface along mesh normals and setup correct sidewalls (using vertex colors). In the example scene disks in the center are prepared this way. Notice that we've got real 3d volume now that can be viewed from any direction. Take care about heightmap vs mesh shape. In the example disks are extruded because they outline height map well. For example if you would use plane with such heightmap there could be situations when ray has to travel long way before it pierce the surface:

To fix it you'll have to set number of steps to high value what affects performance unnecessarily. For such heightmap its much better to use disks as they constitute "tight" volumes. This is highly desirable. You can also think of surfaces that have holes inside in case their heightmaps also have empty (black) areas. It's always better to push a few polys more into the model helping fragment shader working faster. Objects with poly count of 50 and 1k will render the same for sure - by no mean GPU vertex shading unit suffers from such (small) number of triangles.

And here is a screenshot of curved disk extruded (320 triangles):
**Steep Parallax mapping**
This is the most general, but also the slowest technique. To get excellent performance on wide range of GPUs you'll have to tweak a bit quality settings (the same as in extrude map shaders). Still shader is quite well optimized and executes only necessary number of iterations, dynamically adjusts number of steps required with sufficient step length. There is no binary refinement here, but secant/interval mapping approach which needs only 1 additional tex fetch and a few math operations for good enough solution. Refer to shader code defines section for setup features.

There is also one quite special shader here. POM shader that can handle **real silhouettes**:

When applying such material shader on the object, run attached tool to bake curvature/uv scale information into vertex colors (Window/Relief Tools/Encode Curvature into Mesh vertices colors). **Enlarge area approx.** option works well for smoother, more tessellated meshes. When computing curvature at given vertex we look 1 step deeper so that not only adjacent neighbour vertices are taken into account but their neighbourhood too. To get better results you'll have to tessellate your mesh a bit. Tube mesh in example scene is maybe overshoot, but trying to make "cube looking round" with correct silhouette will just fail. Curvature values would get too high to be correctly represented as colors. More – viewing direction in tangent space will be also badly interpolated. If you intend to use such shader on Unity trees be aware that they're not enough tessellated in most cases. Using custom models seems to be better idea (there are lots of cool tree stuff on the Asset Store now).

**Steep Parallax Volume**
Have you ever wanted to sweeten your scenes with nice detailed geometry but polycount budget disallowed it ? The opportunity just arised. When hi-poly object can be represented as heightmap you can use the shader like in King Kong faces example scene (**POM_Volume.unity**).

This requires:
- height map (to constitute hi-def volume surface) + normal map (mesh won't have original normals)
- lo-poly "tight" mesh (look above what does it mean) that outlines desired 3d volume
- planar uv coords projection for such mesh (take care of uniform uv scaling)
- setting up correct normal/tangent direction params in POMVolume script (attach it to object)

The script prepares uniform normals (they all will be pointing in the same direction in which 3d heightmap extrusion is realised) and tangents (need to point the direction where tex u coords raise) + parameters needed by shader. The example has been originally prepared in Blender (to get heightmap and baked normalmap). Then hi-poly has been decimated from 65k to 650 tris and blowed along normals so that lo-poly mesh volume keeps always a bit outside hi-poly original.

Results are below:

Screenshot below shows shader in debug mode. We can see how many steps are actually performed in fragment shader. Dark green means 1 step. Bright red $\geq 10$ steps. Notice that shader doesn't have to be GPU heavy when we've got tight mesh. Only silhouette parts require more steps. Note that we need lo-poly geometry only. No more need to prepare separate LOD mesh levels as it scales via fragment shader. As object gets far - it's cheaper and cheaper (small number of pixels rendered). As object gets close we see hi-poly like silhouette at good framerate. Beware that there is no such thing like MSAA for this shader. To get antialiased edges we have to use FSAA instead (like in example scene). BTW it's property of any cutout shader (any RSP shader realising silhouettes).
Known issues that you may run into

**Alpha cutout issue**
The most problematic setup is forward rendering path + directional light + shadows + postFXes used with shader that discard pixels and doesn't handle it in shadow caster/collector passes. It's quite nasty Unity feature and hopefully will be fixed in a time. By now it can be resolved by not using any of mentioned features together or using MSAA (any - x2 is enough). More in-depth discussion about how to overcome it is placed in my VolumeGrass thread:

http://forum.unity3d.com/threads/80987/page6

**Shadow caster/collector in forward**
Deferred lighting is also better path for such custom fancy alpha cutout shaders because it can handle properly shadows that should be visible thru discarded object pixels.

**UV discontinuity**
At some stage you may also run into so called "uv discontinuity" issue. It reveals as GPU gets wrong mip levels for adjacent pixels in 2x2 square. Assumption that uv coords on adjacent pixels are continuous is no longer valid in POM shading. As computed texture offsets can get high values, virtually extrapolated "edge" looks bad:

To overcome the issue we could do at least 3 things:
1. don't use mip maps (quite bad idea for far distance)
2. use ddx/ddy functions to get correct mip thru tex2D() call
3. use tex2Dlod() call and specify mip level in fragment shader

Different shaders uses different approaches to fix the issue. RSP shaders have exact_mips define (look for #define exact_mips) which forces ddx/ddy function call so that right mip level is used. However explicit ddx/ddy calls are slower than pure tex2D() or tex2Dlod(), it removes problem completely. It's to your taste when results are acceptable and when not (use exact_mips then).

Sad news is that Unity 4 broke Cg -> GLSL converter. It concerns openGL platform. While in Unity 3.5.6 compiled GLSL code is fine, Unity4 resists converting tex2D(tex, uv, dx, dy) into GLSL texture2DGradARB() call. The only reasonable solution for now is to compile shaders with options needed under U3.5.6 and copy/paste compiled shader code to Unity4.

A few RSP shaders don't use ddx/ddy but use first mip level at near distance and fade parallax calculations far from camera (tex2D() is used then). This also works but user has to take care about setting distances in material parameter manually to get right results (mip level 0 looks bad far).
**Tangent space to world space ratio**

When using shaders keep in mind that we transform from world space into tangent/texture space when rendering parallax shaders. If you set uv coordinates which squeezes it the way u coord to world space and v coord to world space ratio is far from 1, then viewing direction will be interpolated wrong way. This is the issue of every parallax shader with Unity's built-in as no exception. When dealing with modelling software try not to "squeeze" uv coords in world space. To be well understood. It doesn't mean that your uv coords should map to meters in world space. It only means that when u texture coordinate distance covers 1 unit in world space, then v coord should also map to 1 unit. If your aspect ratio will be far from 1 you'll experience tangent space squeeze which is not desired.

This moment the only shader in RSP that is "squeezing resistant" (to reasonable extent) is Parallax Steep Mapping with correct silhouettes. When we bake curvature (rg color) we also bake uv / world ratio (ba color) for given vertex so you don’t have to worry about the issue using this shader. Unless uv ratio does get crazy big values that can't be precisely represented in 0..255 color range. Beware that script that prepares mesh colors works on object space vertex positions. When you apply not uniform scaling to prepared object tangent space squeeze may occur. In example scene tube with tree bark material has been scaled 3/1 so results from curvature preparator also had to be tweaked. In **Tangent to TexSpace...** vector material parameter, its y component has been manually corrected from 2 to 6 to follow scaling factor.

**Silhouette UV cut setting**

This is feature of every shader in the package (except for POM Volume). When enabled in shader defines, you’ll be able to cut silhouettes on requested UV borders:

When ray marches thru tangent/texture space and gets outside declared rectangular UV borders then pixel is discarded (alpha cutout). So this should not be confused with POM shader rendering silhouettes where we discard pixels that belong to curved surfaces.
Shading for lightmaps (probes) & deferred

Additional Shading for single/dual lightmaps (probes) is the feature of every RSP shader. When you mark your object static and lightmap scene or when you use light probes, your objects won’t lose normal mapping effects. Not only when using directional lightmaps, but any lightmap type. Shader computes light direction itself (as this is used for self-shadowing) which gives an opportunity to use it for mimic diffuse lighting on object surface when it's lightmapped.

Another problem might arise when you use deferred rendering path. Unity doesn't refresh _WorldSpaceLightPos0, so when your lighting is dynamic it's not reflected by self-shadowing on RSP shaded objects. To resolve the issue use ReliefShaders_applyLightForDeferred script (ReliefPack / Scripts / Common). Just drag&drop it on the light you'd like objects to follow or use it per object. Refer to comments inside the script for more info.

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/sound assets in package are free to use in your projects. They were free (with reference where they came from), made basing on public domain resources or made from scratch just for this package.

Roadmap, further development notes

Although RSP is quite large set of high quality shaders, it’s sold on reasonable price. To help further product development it’s important to remember that it depends strictly on package popularity. Please, if you only find my shaders a good solution, don’t hesitate to recommend it to your friend developers (on the forum or wherever else). The sooner the package sales compensate my work, the sooner it will be possible to prepare updates, improvements and additions.

Wish you make fabulous games using RSP

Tomasz Stobierski