Graduation report:
Procedural environmental design

Report on the subject of the procedural modelling of natural environments and creating procedural detailing for this environment.

Name: Freek Hoekstra
Supervisor: Kim Goossens
Student ID: 072130

Track: Visual Artist
Date: 22 December 2011
Period: block A & B
Number of ECTS: 39

NHTV International Game Architecture and Design
The Academy of Digital Entertainment
Breda University of Applied sciences

Abstract:

In this paper a volume based procedure is discussed for creating high detail fully controllable natural environments that allow for quick and efficient iteration, design, and detailing. The system is capable of automatically generating highly detailed game/film environments from any number pieces of rough input geometry, and is able to export the positions of any placed actors to the UDK.
The primary goals are to minimize the amount of input required for generating a high detail game environment, and minimizing the amount of work required to detail this environment, maintaining fast iteration and full control until the last moments in production.

Table of Content

1. Introduction

This Graduation project is about creating several procedural tools for modeling, designing, and decorating environments. The main effort was directed at achieving both visually high quality results while keeping the artist/designer in complete control over the end result, with the minimum amount of input required, allowing them to create levels faster leading to better levels overall.

The graduation period, seen from an educational point of view, is focused on applying all the skills learned during the specialization stage and expanding on that knowledge. Most of the effort was directed at creating realistic, controllable terrain generation and optimization, but effort was also spent creating realistic scattering of objects and smooth engine integration with the UDK.

In some ways this graduation project is a continuation of curve based procedural modeling, building on the specialisation product I have created before my graduation, (creating a procedural railroad). However, it has been expanded to allow for more forms of input and thus does not require curves specifically. Pre modelled sections or grids also work well, allowing designers the freedom to be creative in how they will set up a level and modify it. For example, if a pocket has to be moved the designer can literally cut out that object from the input geometry and position it where he wants it to be.

To keep the procedure easy to use for everyone the procedure is split up in several different sections, all focusing on a different portion of the process. This split also allows multiple people to work together on one level without the users having to worry about each other's work, being slowed down by details they do not need or having to recomputed things that are set, (locking unchanged nodes).
Designers can work on the layout (saving out the input geometry to be shared) while an artist is already working on the rock visuals for example, while another artists works on the vegetation parameters. If the designer commits an update the entire level updates for all 3 employees, making for a very flexible non-destructive environment.

On average, users require about 30-50 minutes of training to learn how to use this procedure in a rudimentary way without any prior knowledge of Houdini but confined to their area of expertise, such as a designer designing a level in it, but not concerned with customizing the detailing procedure.

To learn how to use and wire the tool please check out the video tutorial provided.

1.1 Previous work:

As mentioned this procedure builds on the work done for my specialization period, which was a heavily curve based procedural railroad automatically generating switches on intersections.
This graduation project also builds on the work done By Kim Goossens who is also my supervisor.

For the project a plugin was used created by Jan Pijpers. It exports attributes from Houdini to Unreal. However, the process of filling in the values is done by my procedures. Without it this graduation project would never have been possible.

The procedure was inspired by procedural tools like those used in the Cryengine and by a desire for good looking environments in games with the ability to sculpt tunnels, bridges, and have a lot less UV distortion. However, most other scientific research on procedural environments was irrelevant to this project, as they tend to focus on generating "random" environments, not providing artists with the control they require. For instance, diamond square or other noise algorithms may create visually stunning terrains but they are unusable (directly) for games as the end result is uncontrollable.

The procedure itself only relies in a small part on techniques I developed for my specialisation, but it does use plenty of tricks I have developed at that time. More info on it can be found here:

http://freek3d.com/downloads/Procedural_Railroad_Creation_Hoekstra_Freek.pdf

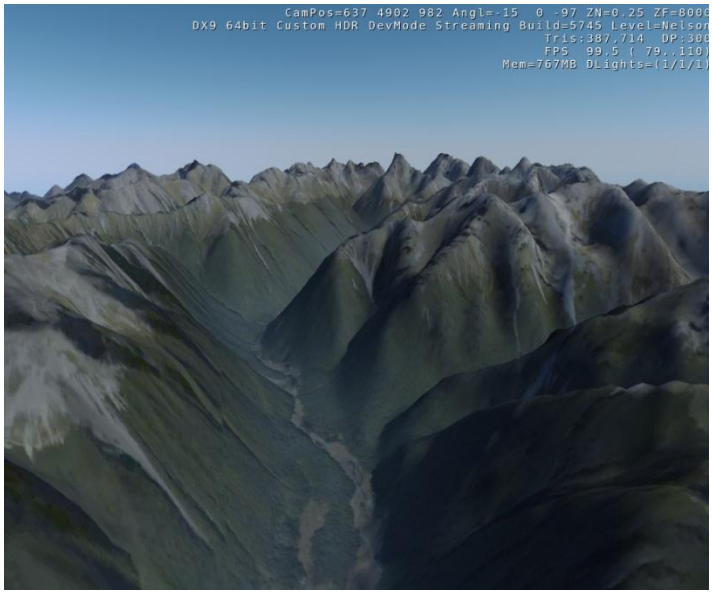
1.2 Goals and challenges

The project's primary objective was to create a tool to generate good looking external environments automatically, requiring no intervention from the artists/designers aside from determining the basic shape. This paves the way for a very controllable environment creation tool by allowing the user to modify and control the shape and look of the procedure very precisely, something which is not always possible in a random city generator for example. This enables a very fast workflow for creating procedural environments, but still allows for environments that very closely mimic the designers specifications without being too labour intensive or too random.

1.2.1 Heightmaps

At the moment there are 2 dominant methods of creating terrain; heightmaps and custom models.

Heightmaps do not natively support tunnels and natural bridges, forcing artists to make a part transparent and placing models through the edge, making a smooth transition hard to create. Heightmaps also cannot have overlapping or steep cliffs due to UV stretching that would take place, limiting the designers creativity.



To the right is a heightmap taken from the Cryengine with all detailing removed. One can see it features no overhangs, tunnels, or natural bridges forcing Crytek to find creative solutions to create such features in their game.

However, heightmaps are easy to modify and understand, texturing is baked in, and the time needed to create it is minimal.

1.2.2 Custom models

The alternative at the moment is to hand model the environments. Uncharted 2 features modeled mountainsides, showing much more realistic terrain and complete artistic freedom.
The trade-in however is the amount of time needed to create every environment, or worse update one.

Also this method becomes unpractical when dealing with larger terrains for a snowboarding or racing game for example.

This problem inspired me to try and create a system that is both easy in setup and iteration without the normal constraints, but would significantly beat the traditional modelling approach in terms of speed.

To the right, an icy environment in Uncharted 2. Notice the overhangs, suspended surfaces, and added realism, beauty and creative freedom.

2. Overview

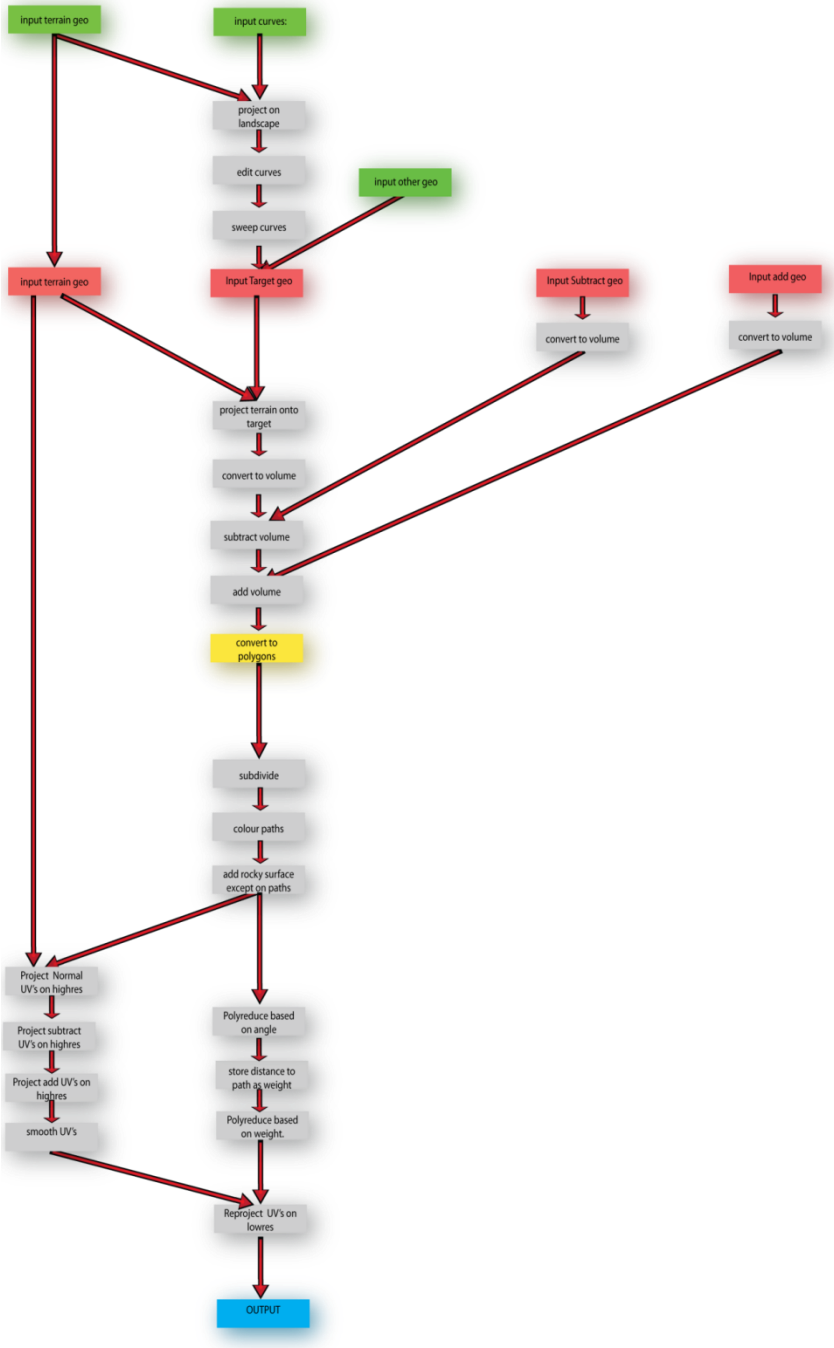
This chapter is to provide an overview of what the procedure does, how to use the procedure and how the procedure works and treats certain situations.
The procedure will be treated in depth in the chapters that follow.

2.1 The main graph:

In a “chronological” or “graph following” sense, the terrain generation procedure works as follows:

First the path geometry and the main terrain geometry that will be projected onto it are created by the user. Both can be a grid, a swept curve or anything that does not have overlapping vertices in Y.
Additionally the geometry that is to be subtracted or added to the normal geometry is created. These will together make up the four red nodes on the image.

2.1.1 The main layout geometry:



To create the main layout, the main terrain is projected vertically onto the path geometry. This can create canyons, heightened or flattened areas relative to the original terrain.
The input geometry can be anything the procedure can project on, although generally swept curves tend to be most user friendly.

After the projection the landscape is converted into a fog volume, and the subtracts are carved out of it, (to create tunnels for example) and extrusions added to it, (to create natural arches for example) working as a type of Boolean, creating a smooth transition from one to the other.

Finally the volume gets converted to polygons again, resulting in an even polygon distribution which gets cleaned up to be even more consistent in detail. We have now reached the Yellow node.

2.2 detailing the surface

The surface that comes out of the conversion process is very rudimentary. It has no features and is very low resolution, but it does have main desired shape and topology with consistent detail across the entire surface. This mesh is now suitable for sculpting or procedural detailing as described below.

To start the procedural detailing process a subdivision is done on the surface, (just like one would when sculpting the detail). The reason the model is subdivided and not simply sampled at a higher resolution is because a high resolution volume is very costly both memory and performance wise, thus the detail is attained via subdividing as the main features don't require such great detail.

To detail the surface, the areas that need to be displaced are coloured. This colour is used as a multiplier for the amount of displacement. The colour is applied by transferring it from the original target geometry onto the generated surface with a transfer and blend distance set, making sure the target surface is smooth, just like the designer wanted, while the rest of the geometry is displaced to look like rocks or other detail. One can also choose to detail the ground with another displacement if desired. However, as this is the gameplay track I personally feel it is better to leave this purely to normal maps at the moment.

After the detailed surface is generated the UVs are projected from the input geometry (the terrain) onto the generated mesh. This does not cover the tunnels and extrusions, so the UVs are also transferred from those inputs back on the geometry it has affected. After the UVs have been transferred they are smoothed to get rid of any stretching or application artefacts on the surface.

Finally the mesh is optimized to the desired level of detail (polycount) and the smoothed UVs are reprojected on the low-resolution mesh. This double projection prevents any UV distortion that would occur if the UVs were directly projected on the lowres mesh.
The detailed mesh is now complete and ready to be cut up into segments and be used in the game. Some minor manual tweaking might be desired for final quality.

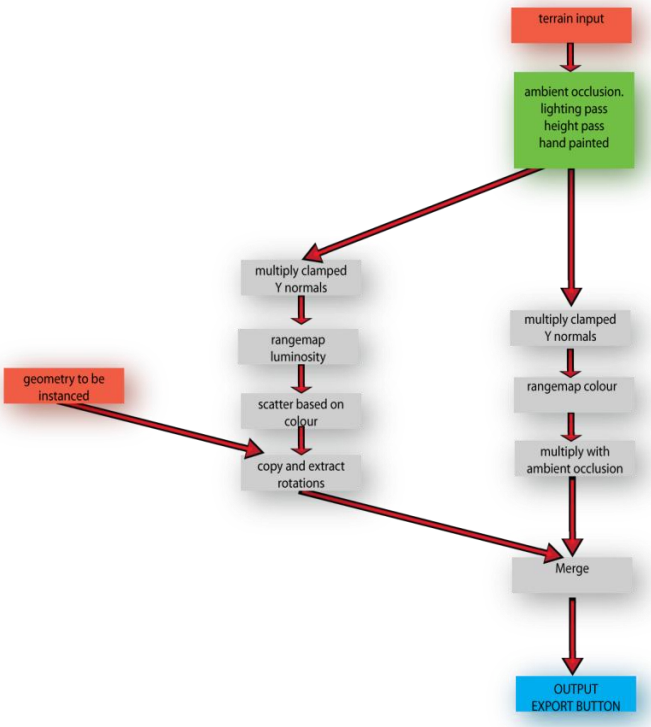
2.3 detail meshes

The terrain mesh is completed. However, it has not been decorated yet. Any number of operations (assets like fences, vegetation rock scattering etc.) can be run side by side to create procedural placement of objects/static meshes onto the surface.

This method also supports placement of gameplay items, like weapons, lights, pathnodes, spawn points etc. although I have not made massive use of these possibilities in this project.

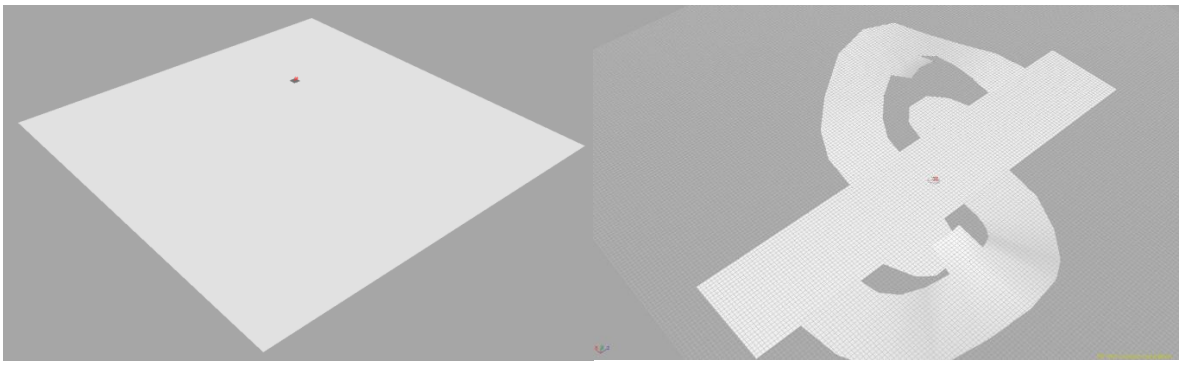
For example, to scatter vegetation I generate an ambient occlusion map, subtracting the Y normal (the walls) from the colours and scattering vegetation based on the various luminance values using range mapping.
One can also use parameters like height to grow specific types of vegetation at specific locations, or invert the normal based selection to select the walls to scatter objects upon like ivy or vines.

The same range mapping technique can also be used to create vertex colouring onto the surface (the right part of the graph). However, this technique is not used yet, as .obj export regrettably does not support vertex colours.

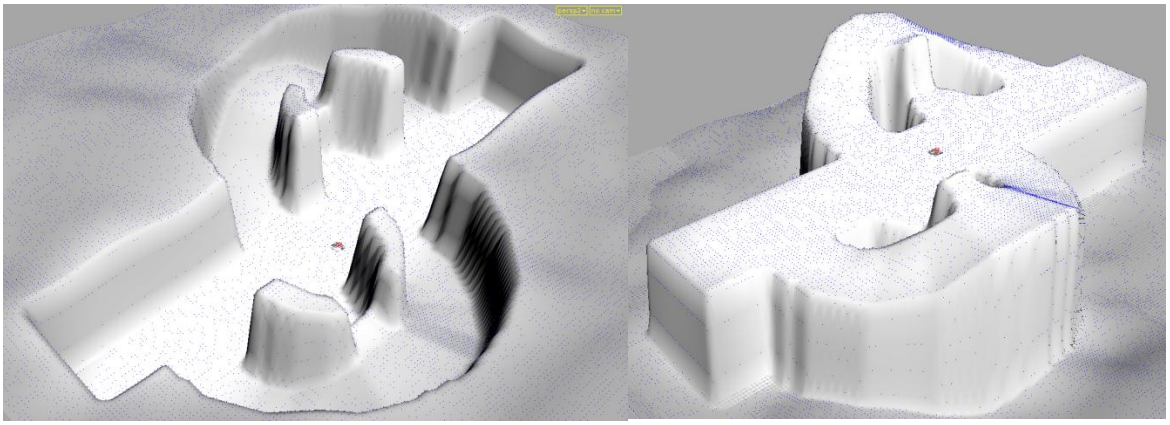


3: shaping the environment.

This chapter will discuss the process of creating the main gameplay surface(s).
The procedure projects the points of the original terrain onto the target surfaces as created by the user. I shall refer to this process as “stamping” throughout the rest of the paper.
All kinds of surfaces can be “stamped” onto the ground, Grids, premade models, swept curves. All non-overlapping geometry is supported. Below is an example of the “stamping” behaviour.

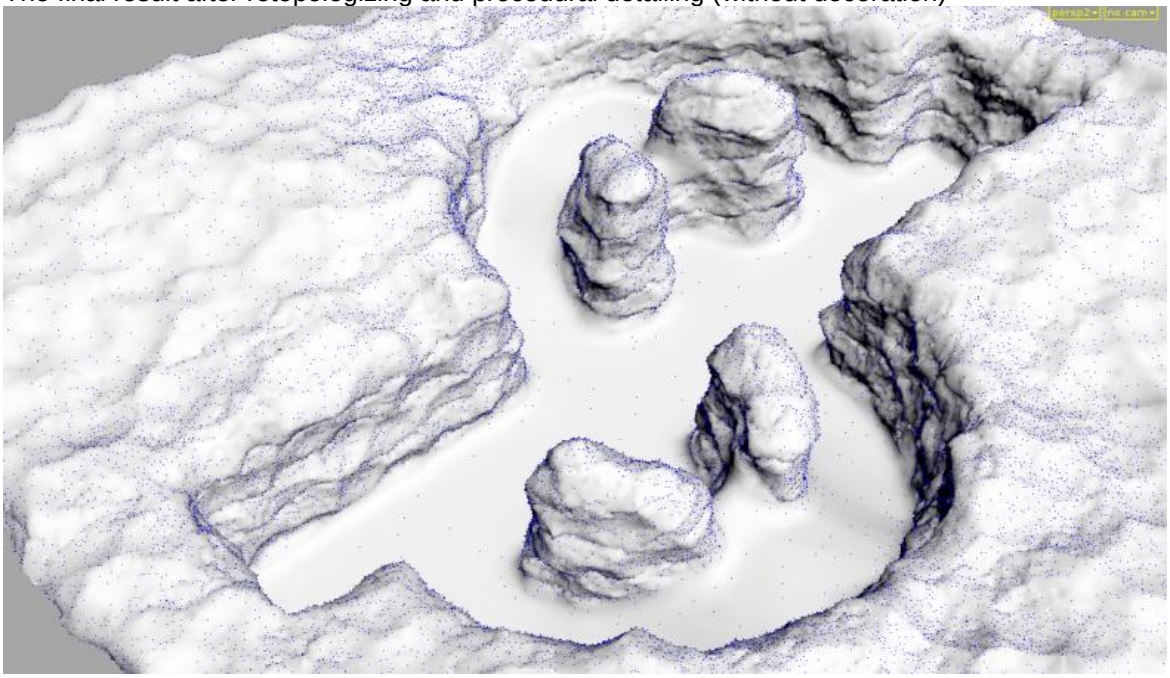


Terrain surface to be projected Target geometry to be projected upon.



Target geometry placed below the terrain Target geometry placed above the terrain.

The final result after retopologizing and procedural detailing (without decoration)



3.1 preparing the inputs:

At the start of the terrain generation 4 inputs are required.

- **Input 1: the original terrain**
  The first input is the original surface that is going to be displaced. It can be a grid, a side of a mountain, or a swept curve. It is not allowed to overlap in Y or converting to volume will cause errors. This surface will also be used to extract the main UVs from.

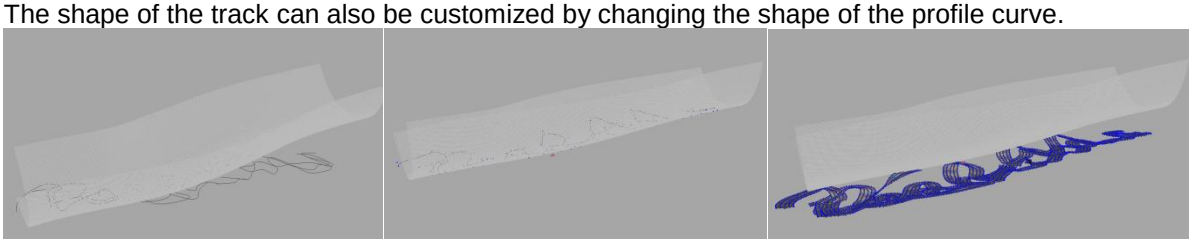
- **Input 2: the target geometry**
  The second surface is the target geometry. The original surface will be projected onto the target and the vertical difference will make up the shape of the terrain.
  These surfaces should also not overlap in Y. If they do the closest surface will be used. This can lead to errors and should be avoided.

- **Input 3: the subtraction shapes/volumes**
  The third input will be converted into a volume and subtracted from the normal geometry resulting in tunnels, alcoves or holes. This surface will also be used to extract the UVs from.

- **Input 4: the addition shapes/volumes**
  The fourth input is also converted to a volume and is used to add geometry, often used for overhangs or natural bridges. This surface will also be used to extract the UVs from.

3.1.1 Target surface creation:

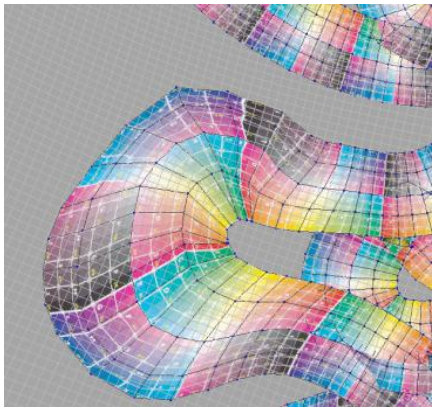
Most commonly target geometry is created by sweeping curves. This method allows for quick iteration and does not require any manual modelling by an artist, making it very user friendly.
An option is available to project the curves onto the landscape automatically to more easily follow the general contours of the original landscape which is especially convenient with very slanted surfaces like mountain sides. This projection can be offset by any amount in Y and after projection the curves can still be adjusted if desired. This method can be combined with curves that should not be projected (specific heights) or other shapes that are prebuilt or built from other shapes for instance a grid.
The shape of the track can also be customized by changing the shape of the profile curve.



Input curves | projected curves | offset and swept curves.

3.1.2 Final tweaks

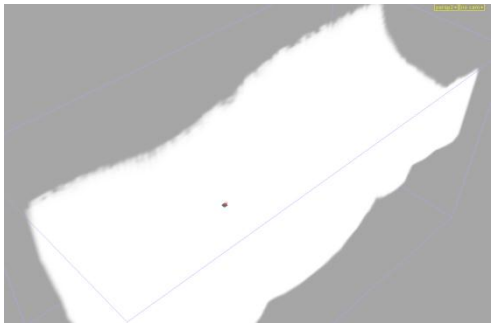
After the projections the curves are swept, but are not set. They can still be manually edited as with the image to the right. It demonstrates a widening of the path at certain areas, allowing the artist to control the path more precisely.



After this step is done the target surfaces are ready to be stamped onto the original surface.

Note: the distortion of this path does not affect the final UVs, except if a secondary channel is to be projected based on this input to for example overlay tire tracks or footsteps following the direction of the track. This is, however, not currently the case.

3.2 Converting the surface.



After the surface has been stamped it needs to be converted into a fog volume. This is done by extruding the surface downwards, making it a solid object and then converting it to a volume using the Iso-offset node.

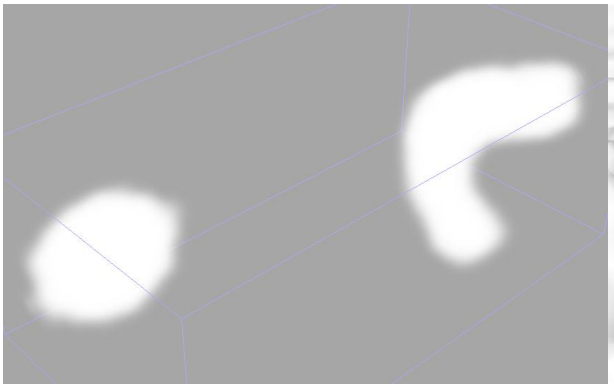
The image to the right displays an example of a volume generated from the extruded terrain mesh. This mesh will become the basis of this procedure.

3.3 Tunnels and extrusions:

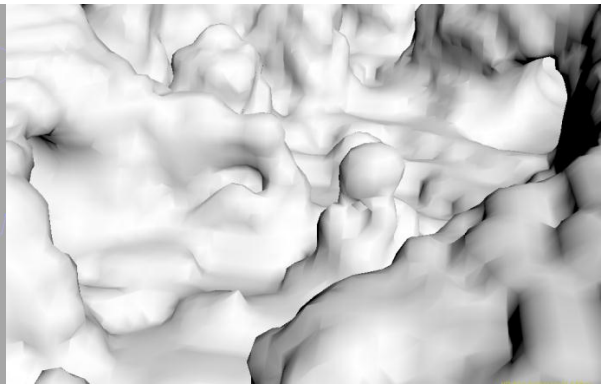
To create the tunnels or extrusions the user must create solid shapes and plug it into the third or fourth input respectively for subtraction or addition. That geometry is then converted to a volume and will be subtracted from the volume created from the main surface to create tunnels, or added to create the extrusions.

To create these input surfaces the user again has plenty of tools. The only restrictions are that the surfaces are closed, and it is preferable if the surface has UVs. I again tend to use curves with an extended sweep (Kim's sweep SOP with integrated UVs), cap it, and usually add some noise, but one can use pre-modeled shapes or simple primitive solids as well. It is, however, better to use objects with well created UVs*.

This setup allows for easy modification and decent control of the end result with smooth transitions from normal geometry to tunnels, bridges, or overhangs, something not possible with Booleans or intersecting models with heightmaps.



Subtraction volumes.



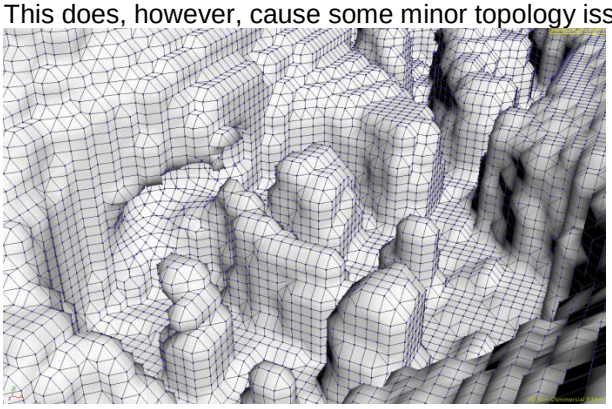
Converted volume with tunnels..

Currently the input meshes are not allowed to intersect with themselves, as this "confuses" the iso offset node. However it is possible to add geometry and then subtract a part of it, or change the order and subtract out a cave and add back in a natural arch.

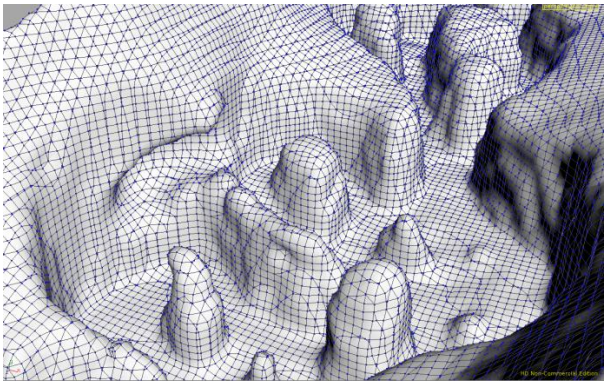
** Good UV's can be classified as little distortion but in this case it is mostly focused on having very little seams, as they can cause issues upon transferring to the final mesh. This problem may be addressed in future versions.*

3.4 Topology:

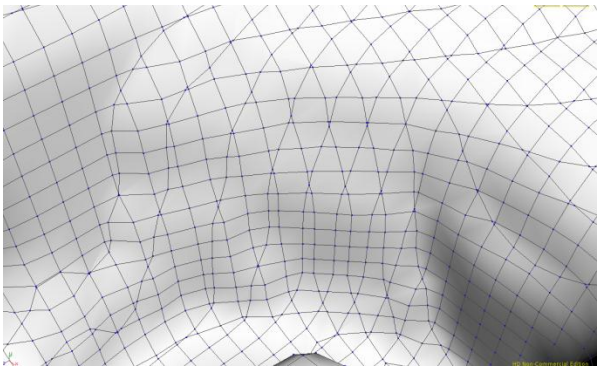
After the mixing is complete the volume is converted back into a polygonal surface using a slightly denser resolution than the original sampling, avoiding the typical staircase effect.
This does, however, cause some minor topology issues:



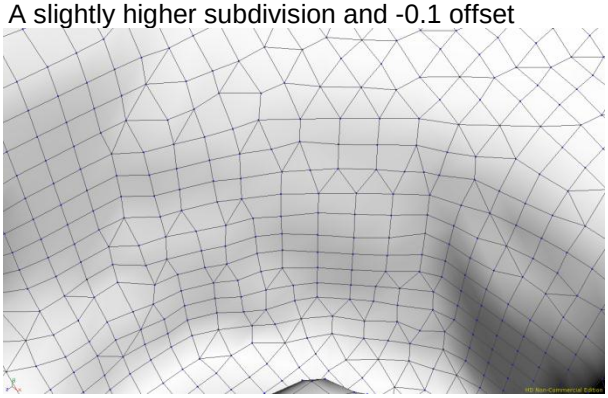
Offset set to same resolution and 0 offset



A slightly higher subdivision and -0.1 offset



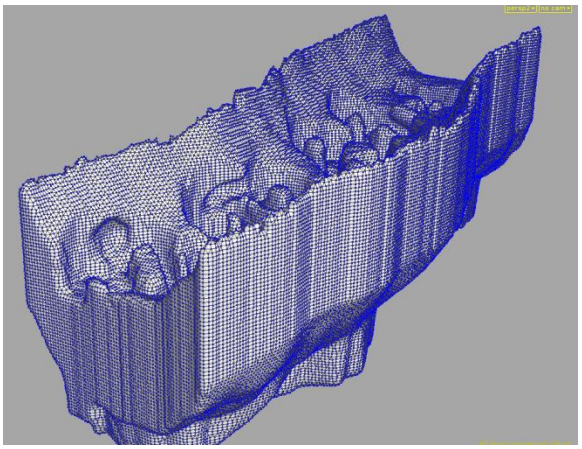
Original output mesh

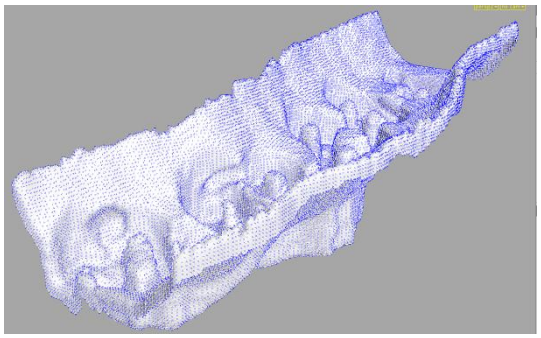


Fused points mesh

The result is brickered mesh. The mesh has polygons distributed evenly across the surface. However, it has unnecessary vertices along the “cutting lines”, making the topology unnecessarily cluttered.
To get rid of this a fuse is set to average the positions on half the conversions division size, merging all vertices that are closer than half an appropriate sized polygon together. This generates the second mesh on the bottom right. An area of improvement could be a proper quadrangulation node after the fuse (a simple detection of shared edges among connected triangles made the result worse). However, as the mesh will be subdivided and finally be polyreduced later I found it to be unnecessary for now.

The resulting mesh has an acceptable topology now, but the sides of the mesh are still present from the conversion from volume to polygonal mesh as visible below. To remove them one cannot simply remove all down facing polygons (as there are tunnels and such). For this reason colour is transferred from the *Input Mesh* and the input subtraction and adding inputs and all sections with no colour are deleted.





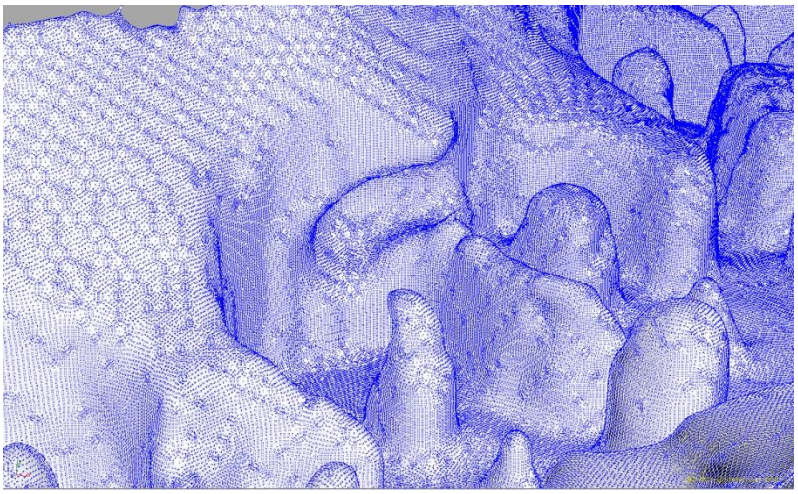
Left: original output
Above: after removing unnecessary points
I will refer to this mesh as the *Retopologized mesh.*

4: Creating surface detail:

Now the surface has all the major features desired by the designer. It can be decorated by the procedure or done manually if preferred. To do this a subdivision will be done on the surface to generate enough surface detail for the rock generation VOPSOP to displace.

4.1 preparing the input mesh:

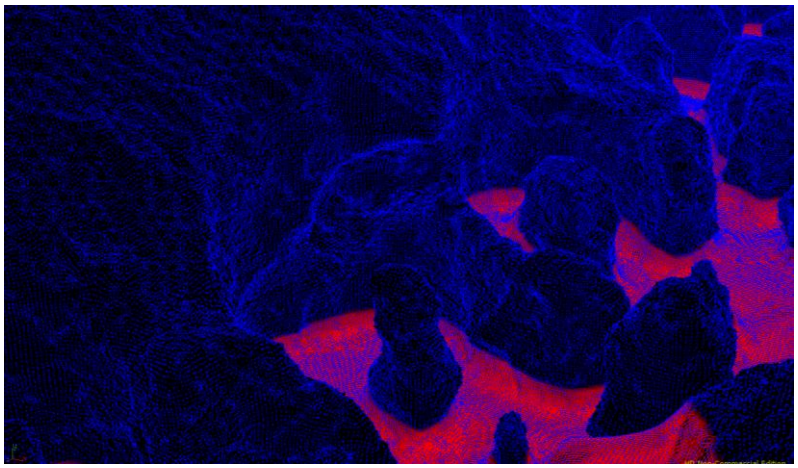
The mesh so far is very smooth and low res. To add detail the appropriate number of points must be added first. The *retopologized mesh* is now 35.000 triangles. By using the subdivide node (set to two iterations) in this case subdivided to around 500.000 triangles, enough to capture the surface detail required for the target resolution.



This generates the mesh shown to the right. It has enough points but is still very smooth.

To determine where the displacement has to be added a colour attribute is transferred from the target surface onto the subdivided mesh based on the distance to the track.

This colour attribute, shown to the right, will be used as a multiplier for the displacement scale of the rock generator and allows to determine what areas of the terrain is displaced by the VOPSOP.



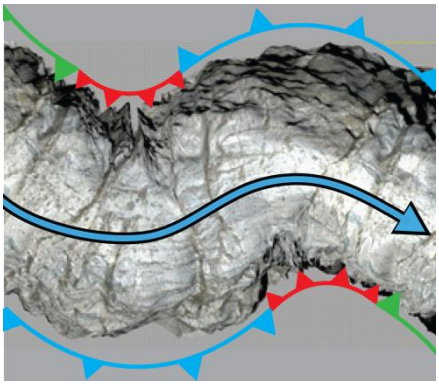
This allows for the creation of detailed and rocky walls, but also keeps smooth gameplay paths which won't obstruct the player.

4.2 sculpting the rocklike surface

To sculpt the rock surface two main types of detailing were tested. Displacement maps, using pre-generated textures applied by UV map onto the surface, and 3D noise patterns.
These showed the most promise out of several small tests that were done and were both efficient in terms of computation time.

4.2.1 Displacement maps:

The first tests were done with the mountain node. After those were discarded displacements were tested. The maps would line up nicely with the textures possibly adding more realism. However, the displacement maps suffered greatly from UV compression and distortion and seams showing up as tears across the surface (colour difference translates to displacement difference).



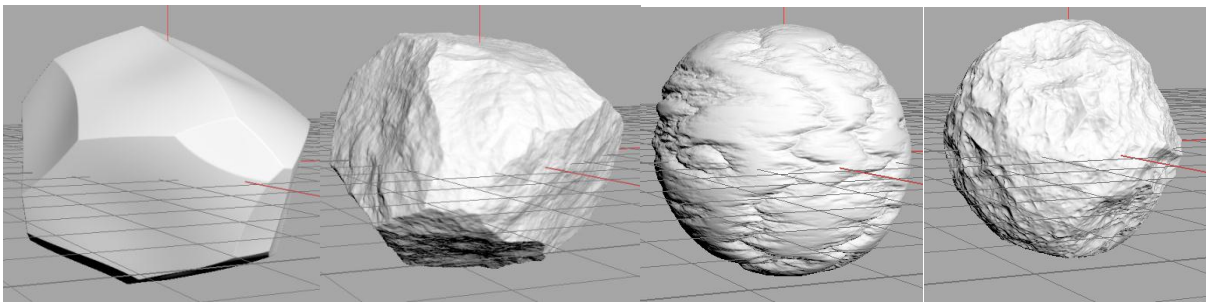
Even minor distortion in the displacement as seen on the right becomes very apparent in the model. Note the compression on the inside in red and stretch in the outside of the corners in blue. Finally a dark value (pushing in) could result in faces intersecting on a sharp ridge. For this reason the method was abandoned.

4.2.2 Noise algorithms:

As mentioned before the noises used are all 3D noises to eliminate any dependencies on UVs.
The main noise used for the main displacement is the Worley noise, a cellular type noise which is very useful to create either blobby or ridged surfaces (displace in or outwards).

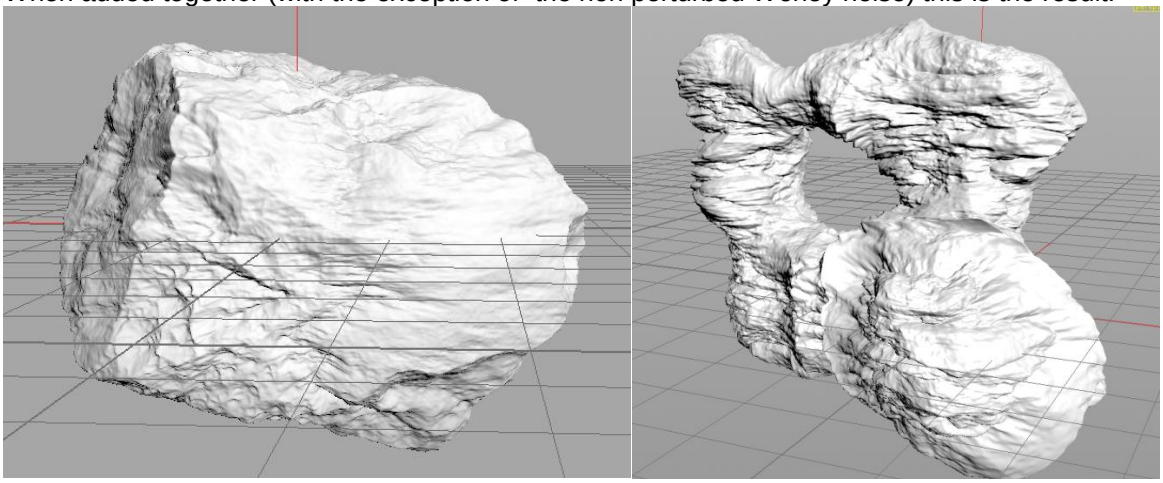
To add some cracks a Veins noise is used together with a fairly high noise value, and a frequency that is higher in the Y axis to create the illusion of layers of sediment, as the highres amount of erosion usually occurs between the layers in sedimentary rock.

Finally to create some more randomness, and to perturb, offsetting a noise functions application coordinates by texture or noise, the Worley noise, a turbulence noise is used.
Below are a couple of examples of what the noise does on a subdivided cube.:



Worley noise perturbed Worley Veins noise Perlin turbulent noise

When added together (with the exception of the non perturbed Worley noise) this is the result:

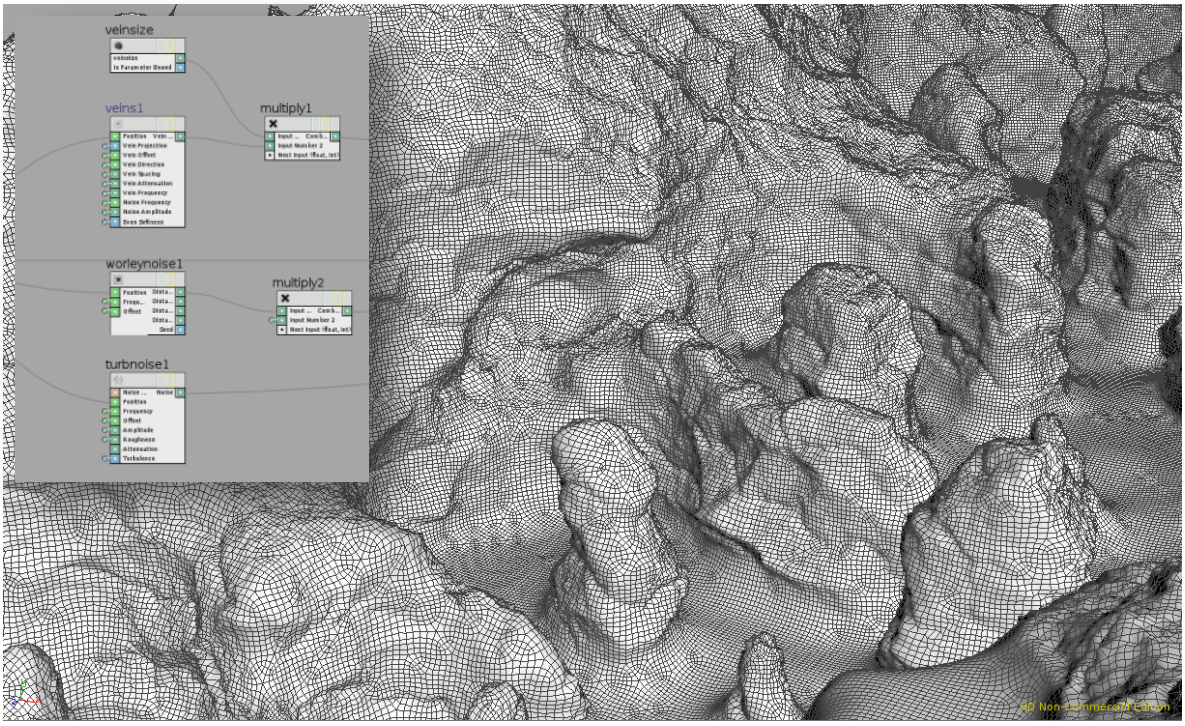


A rock sculpted from a subdivided Box or from a torus like shape.

4.2 Applying the noise to the terrain:

Having chosen the type of deformation similar noise patterns can be applied to the terrain.
With slightly different scale values and properties, which are all fully configurable parameters, the type of rock was made more appropriate for the lowered resolution of this mesh (level of detail per square meter).

The displacement itself takes only seconds to compute, allowing the artist to test various random displacements by playing with the offsets, scale, inversions, frequencies and random seeds.
The solution is resolution independent, so both cinematic or in-game uses are possible, as adding more detail will create more realistic rock, but is slower to calculate (although still fairly quick) but substantially heavier memory wise.



The solution works on several types of rock, blobby or carved (as above) by changing the parameters and frequencies of the several types of noise.

Above in the screenshot the clean paths and the rocky surface combination can be witnessed. The result is fairly convincing and normal maps and textures will be able to add in the fine detail that is missing on the mesh right now .

For next gen or cinematic slightly more detail might be useful, although real-time ingame tessellation might solve most of those issues as well. As the compression stretching issues should be less visible on a smaller scale tessellation that would be used in-game, making it a potentially viable solution to add it inside the engine with DirectX11 or openGL4, or at render time in any mainstream rendering package.

5: UV maps and polyreducing:

To apply distortion free UV maps on the polyreduced mesh a double stage projection is done. This is done so the UVs can be smoothed (on the high resolution mesh) without showing the topology of the polyreduced surface. Also care is taken to have the UV's not strecht on seams by using groups defined by the volumes used to create them.

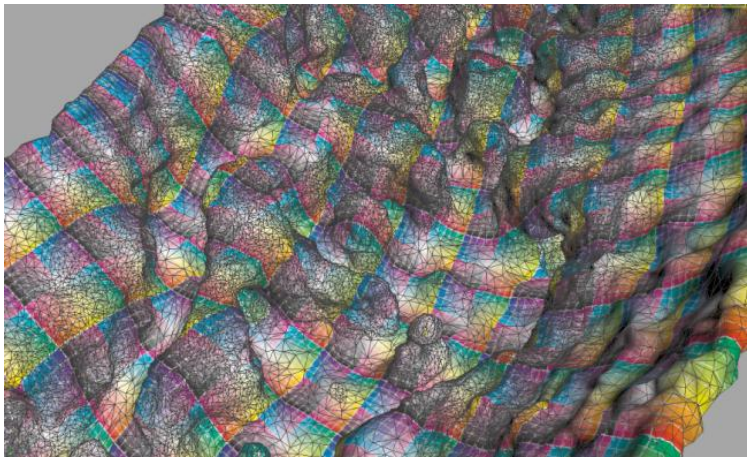
5.1 Primary UV projection

First the original input mesh, as discussed in chapter 3.1.2, is smoothed and the UVs are then projected onto the displaced surface. This reprojecting of UV's onto the landscape is required as all UVs are lost in the conversion process to a volume covered in section 3.2.

The UVs are first projected on the deformed high resolution mesh and smoothed. This is done on the high resolution mesh as it has very uniform primitive size, making sure the smoothing introduces no errors. An alternative would be relaxing. However, this proved too computationally intensive to do in Houdini, but is a possibly improvement for later.

After the smoothing operation has been done on the original terrain UVs the tunnel UVs are applied. A primitive group is made based on the input volume. Then the vertices inside the group are given the UV attributes from the original shell of the tunnel proxy geometry.

The same is done for the additives. The grouping is required to provide a clean cut across the UVs instead of a row of stretching across 1 primitive.



5.1.1 Secondary UV projection

The secondary UV projection is applied from the smoothed UVs on the highres model onto the polyreduced model, the production of which the process will be covered in the next section. (chapter 5.2)

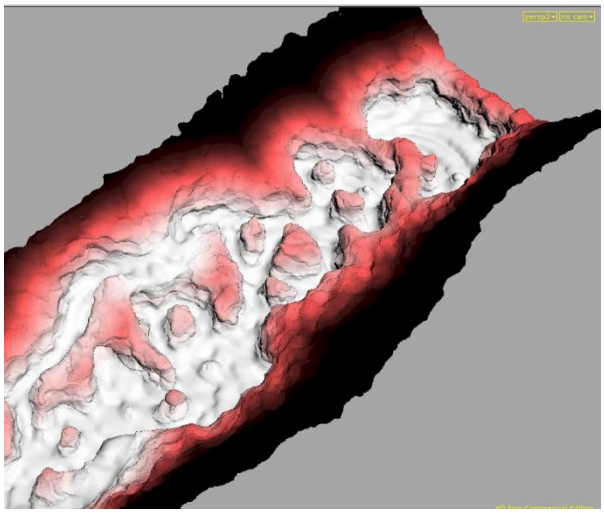
The groups are made for the low resolution mesh as well and the UVs are again applied in 3 steps: The normal UVs, the tunnels and finally the extrusions. This is again to reinforce the UV cut instead of getting a band of UV stretching, just as with the high poly model.

Note: There is still stretching along the original seam in the model where the UVs wrap around the cylinder, as they jump from 0-1, this is becouse the points of the polyreduced surface tend to go back and forth across the border, picking up UV coordinates on either side. A check could be built later to check how much UV space the polygons take to highlight or possibly automatically fix these issues. However, this is not present in the procedure as is.

5.2 Polyreducing efficiently:

As games have limited amount of triangles, of which a generally substantially lower amount is usually reserved for the terrain, procentually terrain often occupies the greatest deal of screenspace with the least amount of polygon. This means having a very efficient terrain mesh is very important for performance and visual fidelity. This is why a non-uniform polyreduction process was chosen, instead of just basing the polycount on the amount of curvature. The distance to the track is also stored as a weight, helping to keep more detail where it is visible while removing it where it is not.

5.2.1 Polyreducing efficiently:

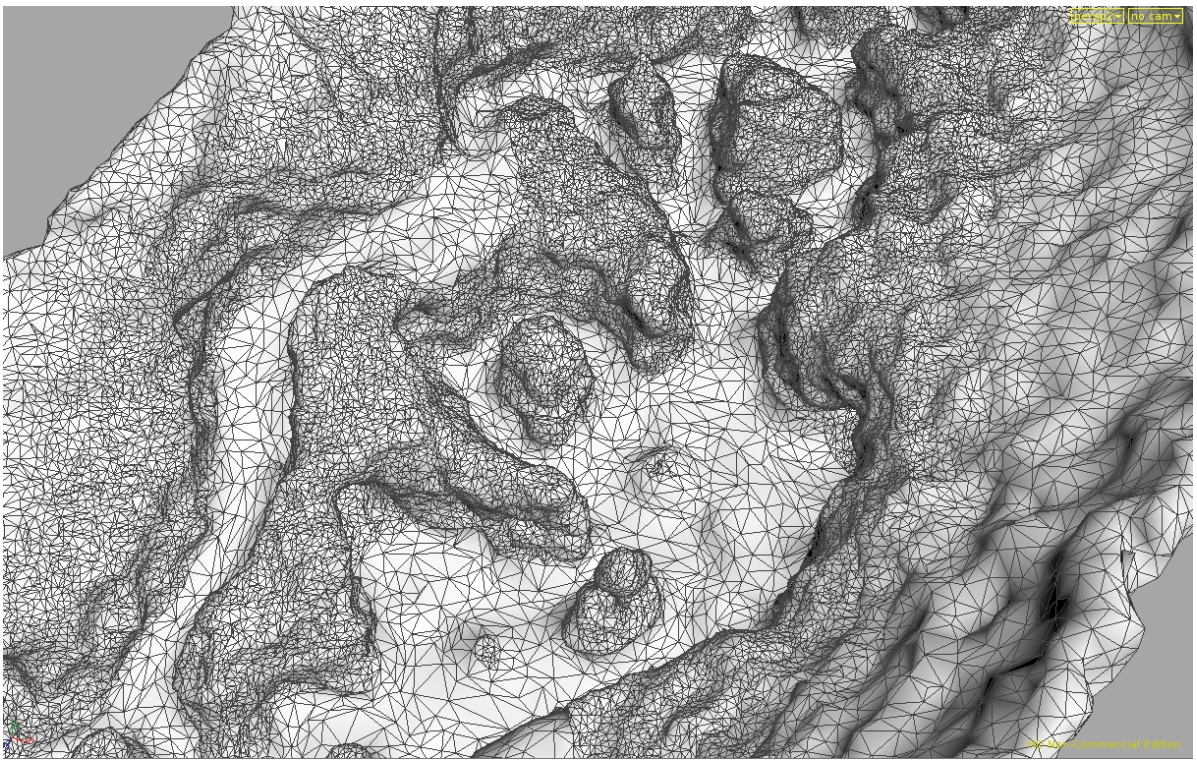


To extract this polyreduced mesh two polyreduce nodes are used.

The first node is set to keep 50% of the polygons, throwing out only the most unnecessary polygons and lowering the polycount for the coming attribute transfer, making it transfer colour faster.

The attribute transfer transfers colour from the target input onto the geometry around it. Using a blend with a fairly big distance this still turned out to be faster than smoothing.

This colour is converted to a weight attribute and used by the second polyreduce, set to reduce the polycount to the desired level based on the distance from the track and the complexity of the surface detail present.



The optimized resulting mesh

*Future work could include a multiplication of the colour with the visibility calculated from the path (using a dot product) or allowing the users to manually override the painted map.

6. Procedural object placement

The procedural placement of objects is done by points with a number of attributes that Unreal will use, and the normal will provide the orientation and if desired a (partial) scale. These points and normal can be created in many ways by using curves with the points aligned along it, or by using a colour biased scatter node in which the colour controls the semi random scattering of points across the surface. The colours can be generated many ways. They can be painted, based on the normal, calculated by a lighting or ambient occlusion pass or any combination of the above.

6.1 Terrain colouring

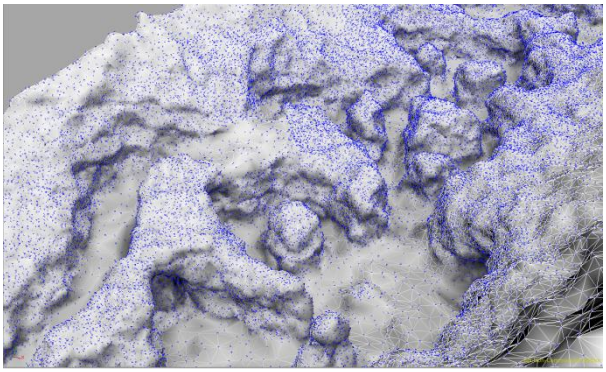
Note: In my procedure I have used two nodes that have not been made by me,
namely the VOPSOP AO node, which calculates ambient occlusion very nicel.:

A VOPSOP that recognizes sharp corners facing outwards (or inward depending on the values) was utilized to identify creases. This second VOPSOP is not present in the final version of the procedure, however, it was used in previous versions of the tool.

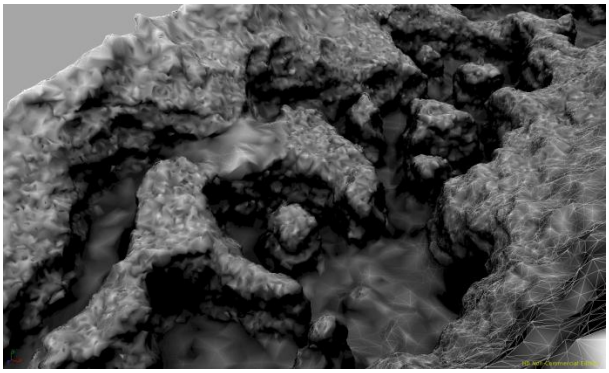
6.1.1 Ambient occlusion

For the terrain colouring I mostly use the ambient occlusion sop as a starting point. In real life objects have a tendency to either be in exposed areas with lots of light like plants, or “wanting” to be in dark areas and crevices such as rocks, and other objects that are subject to gravity.

I then proceed to multiply with the Y normal (clamped between 0 and 1) this way the colour remains only on the non-vertical surfaces where objects would normally be, which can also be inverted to work for objects that are supposed to be on the walls such as ivy, roots, or torches.
The aforementioned VOPSOP is used to calculate the AO pass which is stored in the mesh’s vertices: On this first pass some basic operations can be done to make the map more useful for example:

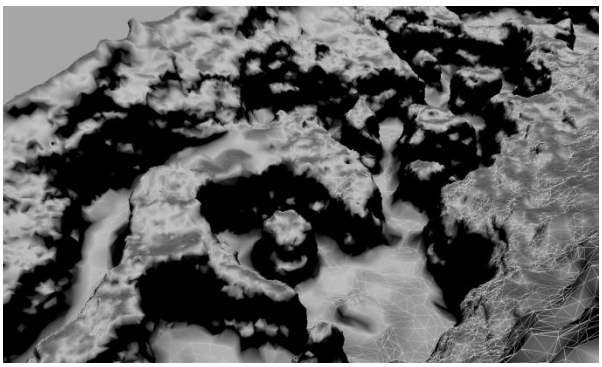


Normal AO pas



AO pass color to the power of 7 creating bigger colour differentiation

Smoothed pass multiplied with clamped Y normal. Used to scatter objects not residing on walls.



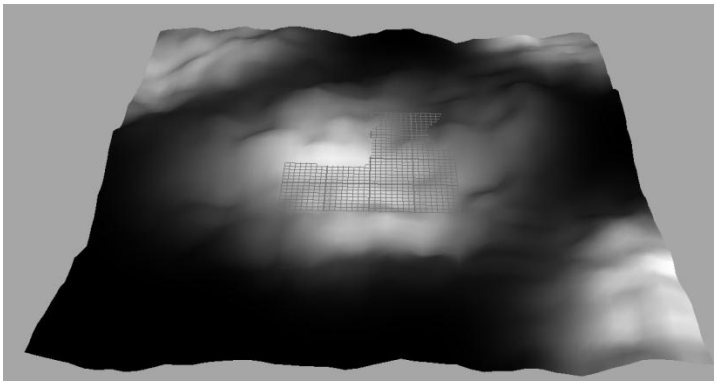
Range mapped AO pass minus the walls. Useful to scatter objects in a controlled area.

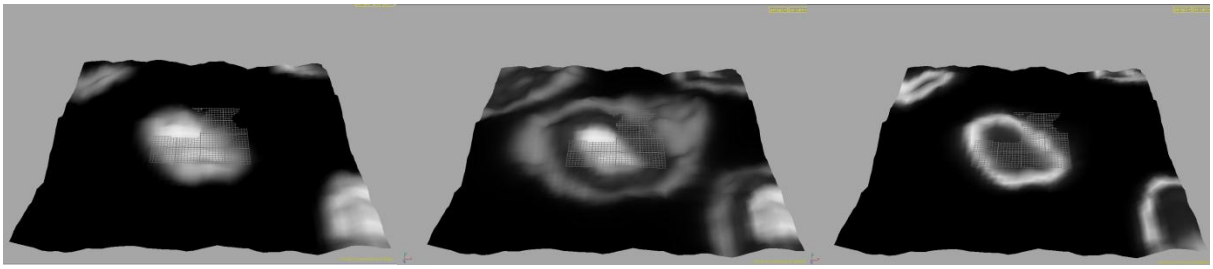
6.2 rangemapping and procedural object scattering:

Below there is an visual example of the colour based scattering:

To the right is the original input: in this case a simple manually painted map. The map is put through 3 range maps (one for each type of vegetation) and a separate colour range map for the terrain colours.

On the light area points are scattered and the stand-in geometry copied.





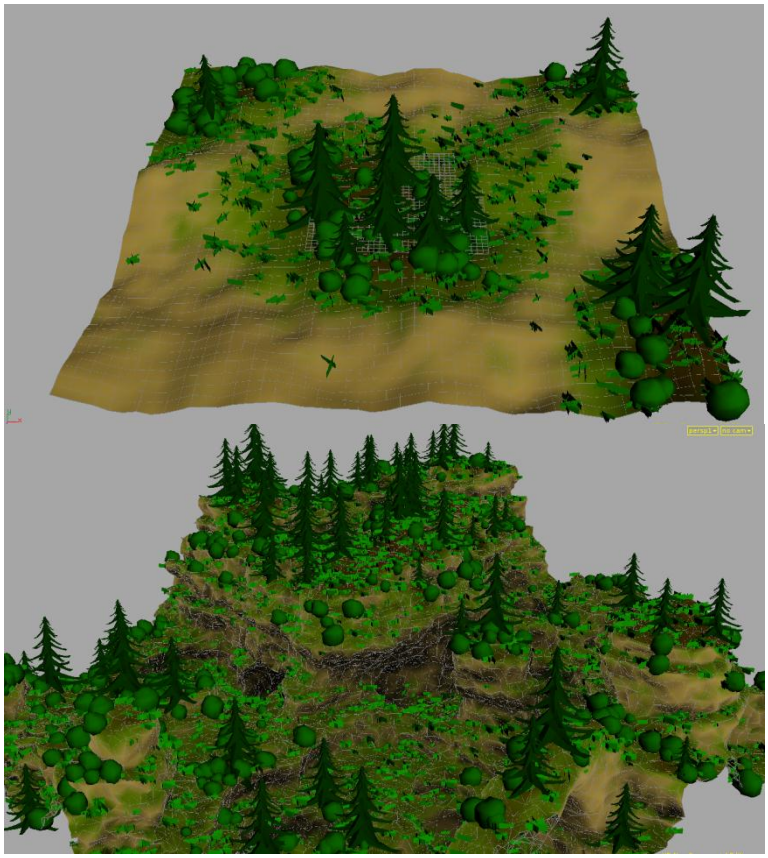
Tree rangemap Grass rangemap Shrub rangemap.

The final result of the hand painted map can be seen on the right:

The colours that were the brightest now have trees growing there. Slightly less bright areas are where the shrubs grow and finally the grass shows the start of the vegetation.

Plants want to grow in the light, the biggest most dominant vegetation taking the best places, and pushing the other vegetation away. Now the range map “rules” are defined in the top example. They can be used on the bottom terrain’s ambient occlusion pass as well.

The result is semi realistic plant scattering: plants are growing on tops of rocks, smaller brushes in the flat low lying areas and no vegetation in the cave system, nor is any vegetation growing on the walls, similar to what one would expect to see in real life.

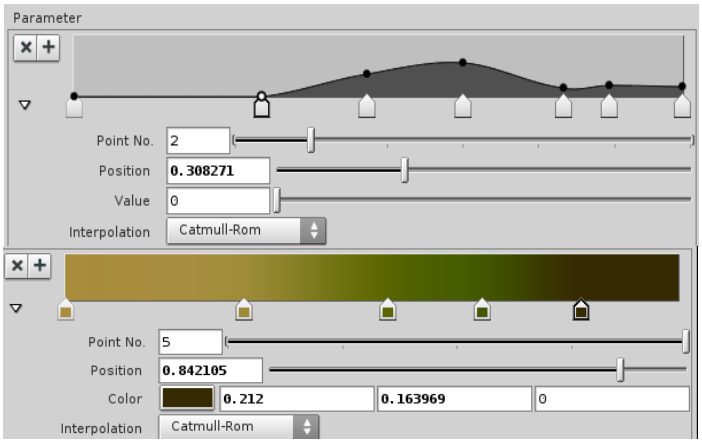


*To prevent trees growing on miniscule ridges a normal smooth can be applied in advance of the normal colour multiplication.

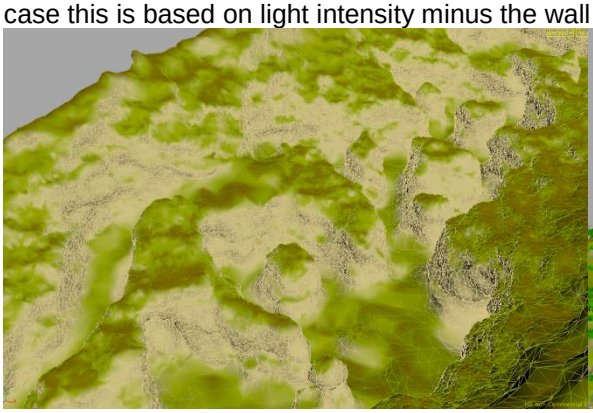
6.2.1 Vertex colouring

As mentioned range mapping is essentially a curve or colour range that is mapped to a value. The value is “remapped” to the value that is present in the range map.

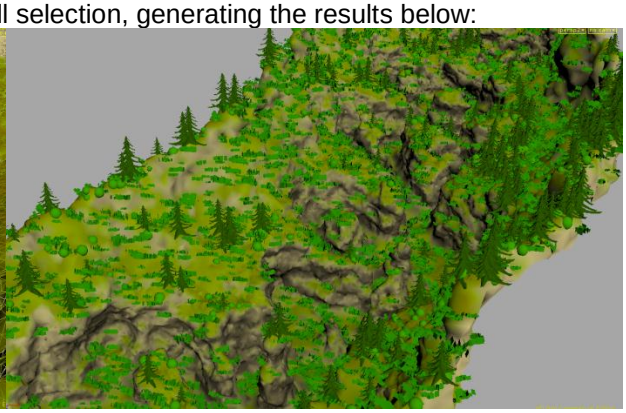
For instance, the range map depicted to the right remaps the colours from 0.4 to 1 to have a colour value. The brightest values exist around 0.6, and the curve lowers the colour intensity after that, providing the user with a selection of all the faces that were partially lit as they are now coloured the brightest.



However, as mentioned before the range map can also be a colour range. In this case vertices with a certain value are coloured to have a certain colour based on that position in the colour swatch. In this case this is based on light intensity minus the wall selection, generating the results below:



The vertex colours applied



The vertex colours* AO and vegetation scattering

6.2.2 Vertex colouring for texturing.

The vertex colours here are mainly to make the scene more readable and pleasant to look at, but it is also possible to map it to a red-green-blue gradient. Those vertex colours could theoretically also drive a shade in Unreal, selecting what texture/material to display based on the vertex colour.

However, this possibility was not pursued as the .obj export does not save vertex colours, making them impractical to get into the UDK. To compensate for the lack of vertex colour exporting capabilities a normal based shader was written in Unreal similar to a snow shader (See Chapter 8).

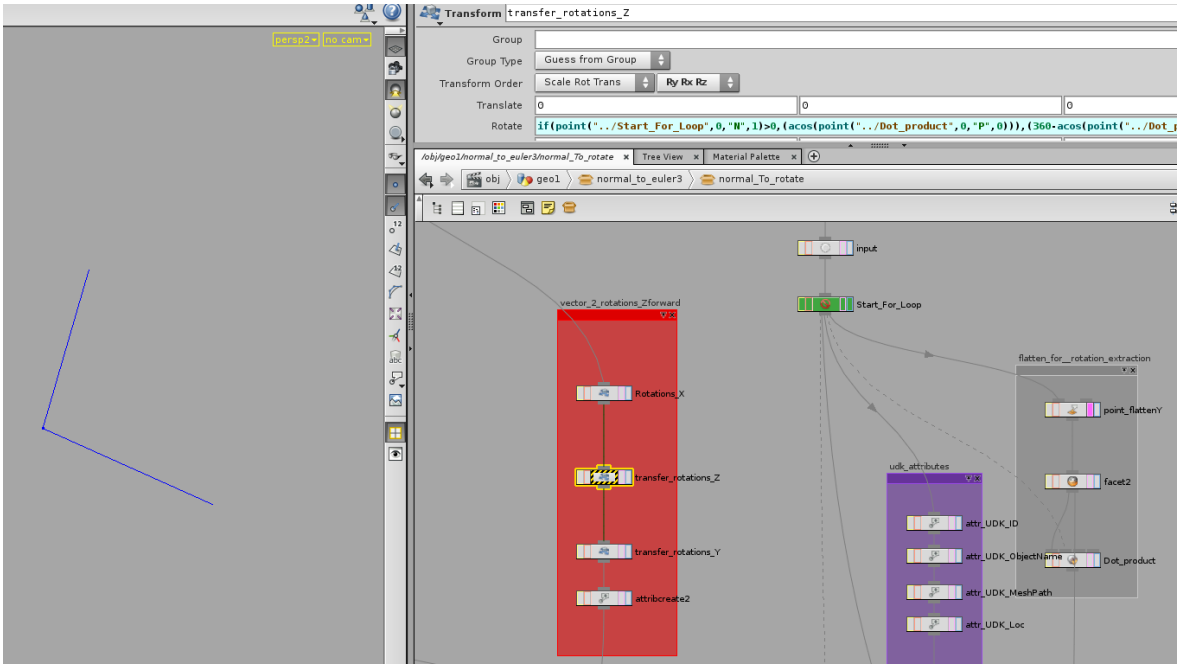
7. UDK connection

This connection has been a collaboration of several artists. The python code that exports attributes such as positions, rotations, scale, mesh-ID, as a .t3D file has been created by Jan Pijpers. It converts the Houdini attributes into a .t3D file with python script. I will do little explaining about this exporting procedure as I did not create it.

I have created a system that translates normals into Euler angels and converts them to the 16 bit integer values Unreal reads as rotations. I will cover this section of the exporter in depth, as it is a vital part of the pipeline.

The converting of the angles is very confusing as UDK works with a Z up world. This means that values may seem exchanged at first glance.

7.1 This is an overview of the most essential part of the network.



1- In the grey netbox a dot product is done between a flattened normal and the original. This will be necessary to extract the relative Z rotation of the normal. The angle compared to the flattened normal as comparing it to either of the X or Z flat axis would create a bigger, and incorrect angle then just the vertical rotation.

2- In the Red netbox the rotation values are calculated using the projected lengths of the normal.

rotation X: projection of the flattened normal on the X axis.
rotation Z: projection of the original normal onto the flattened normal.

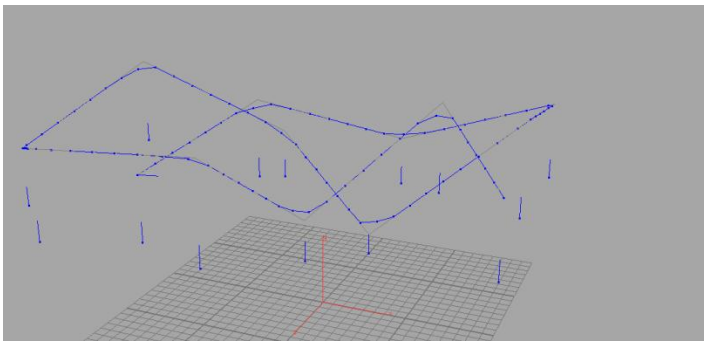
Y rotation (along the normal or roll as it is often called) is not computed, but can be input from outside the subnet, as it is impossible to extract the roll value from only one normal.

3- In the purple netbox the values are converted from Houdini values to values the UDK can use. For example, rotations are converted from 360 degree Euler angles to (positive)16 bit integer values and are stored as point attributes ready to be exported to a .t3d file.

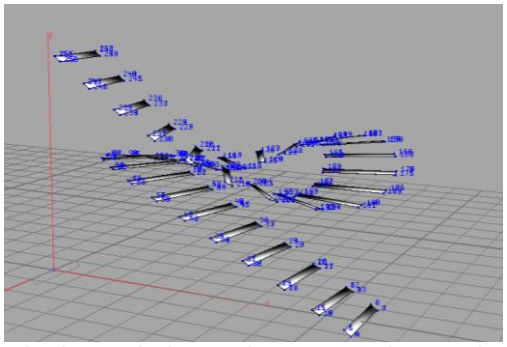
This is only a part of the network to convert the angles. The rest of the network takes care of additional rotation. This makes sure the procedure can work with models that need to be placed with Y or X forward instead of Z. An example could be trees. They align to the normal with Y instead of a wall which would align along Z so in short this prevents flat lying trees.

7.2 Normal to Euler angle conversion

The normal to Euler conversion subnet flattens one axis (Y) and projects the original vector onto the flattened vector using a dot product. Then by using an acosine function the rotation amount is extracted. After the Z axis rotation has been calculated the flattened vector is projected on the X axis (the Translation in X), and the angle between them is calculated the same way, giving the X rotation. If both rotations are applied an object built in the Z axis lines up perfectly with the normal.



Points with normals indicating direction and magnitude.



Planks copied onto the extracted normals.

7.2.1 Normal to Euler conversion in the Z axis

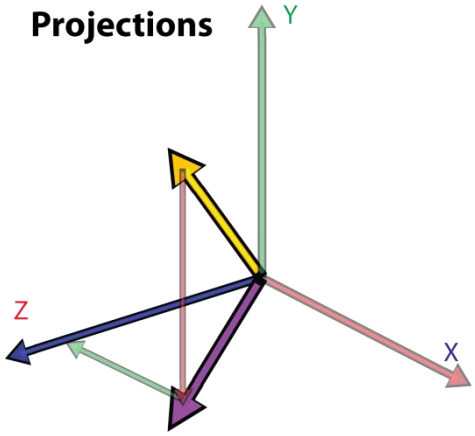
To start extrating the rotations a for loop is started, and 1 point (with one normal) is kept. The normal is flattened in Y, normalized, and dotted with the X Axis (the X position is used preventing an actual dot to be needed) This gives the projection length and is stored as a point position between 0-1.

To extract the Y axis rotation out of this the position is put into a point node and the following expression is run:

```
if(point("../facet2",0,"N",2)<0,
    acos(point("../facet2",0,"N",0)),
      360-acos(point("../facet2",0,"N",0)))
```

The acosine of the X value ,essentially the normal projected on the X axis, returns an angle. The resulting rotation value is the Yaw, the rotation in Y. displayed in green on the second image.

If the Y normal of the original normal is up use the acosine.
If the Y normal is negative invert the values (360- rotation is inverting for Euler angles)

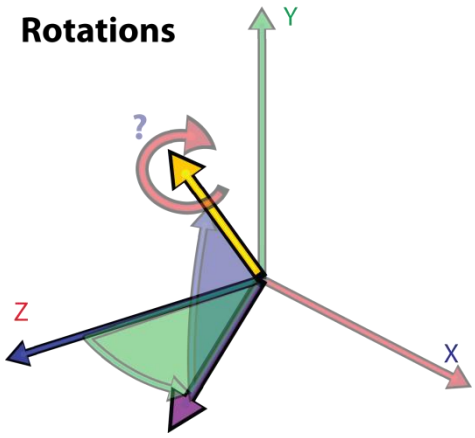


7.2.2 Normal to Euler conversion in the Z axis

To extract the relative Z rotation, along the local axis of of a crossproduct between the flattened normal with the original normal. The original normal is dotted against the flattened normal. (red rotation) and an acosine function calculates the angle.

```
If(point("../Start_For_Loop",0,"N",1)>0,
   (acos(point("../Dot_product",0,"P",0))),
     (360-acos(point("../Dot_product",0,"P",0))))
```

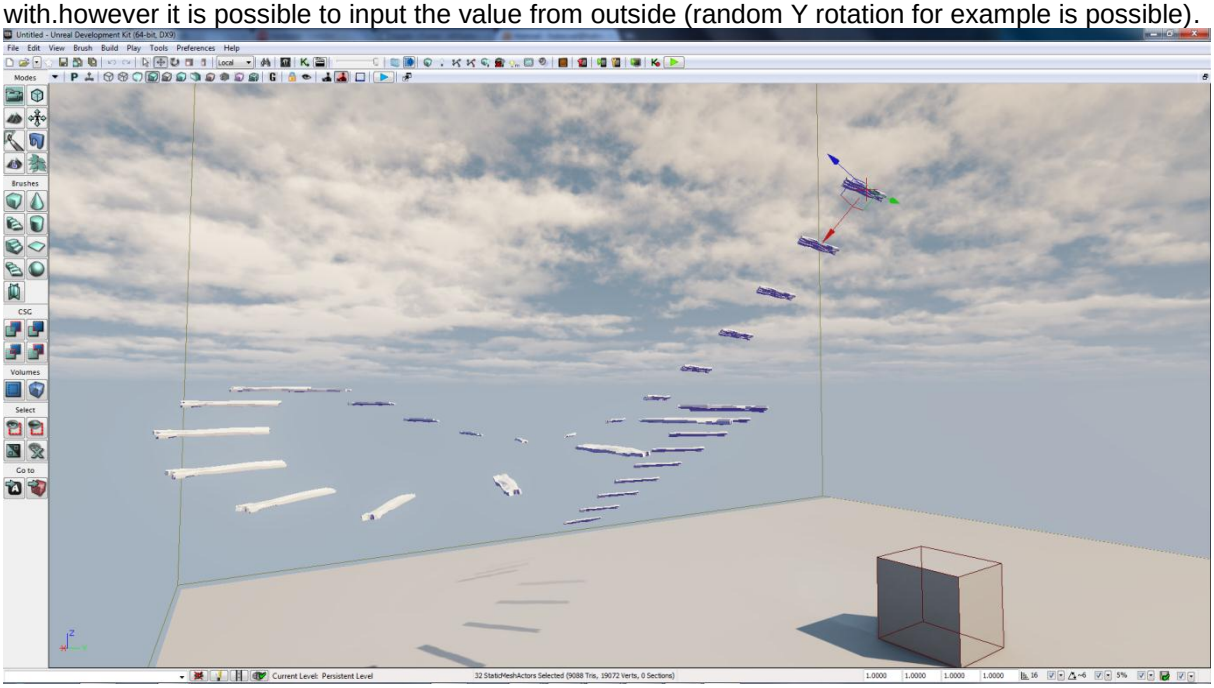
If an object originally aligns to Z and both rotations are added in the same order, the object aligns to the normal. objects aligned along another axis have 90% rotation added in the appropriate axis to accommodate.



yaw Green. pitch Blue. Roll Red

7.2.2 Normal to Euler conversion in the Y axis

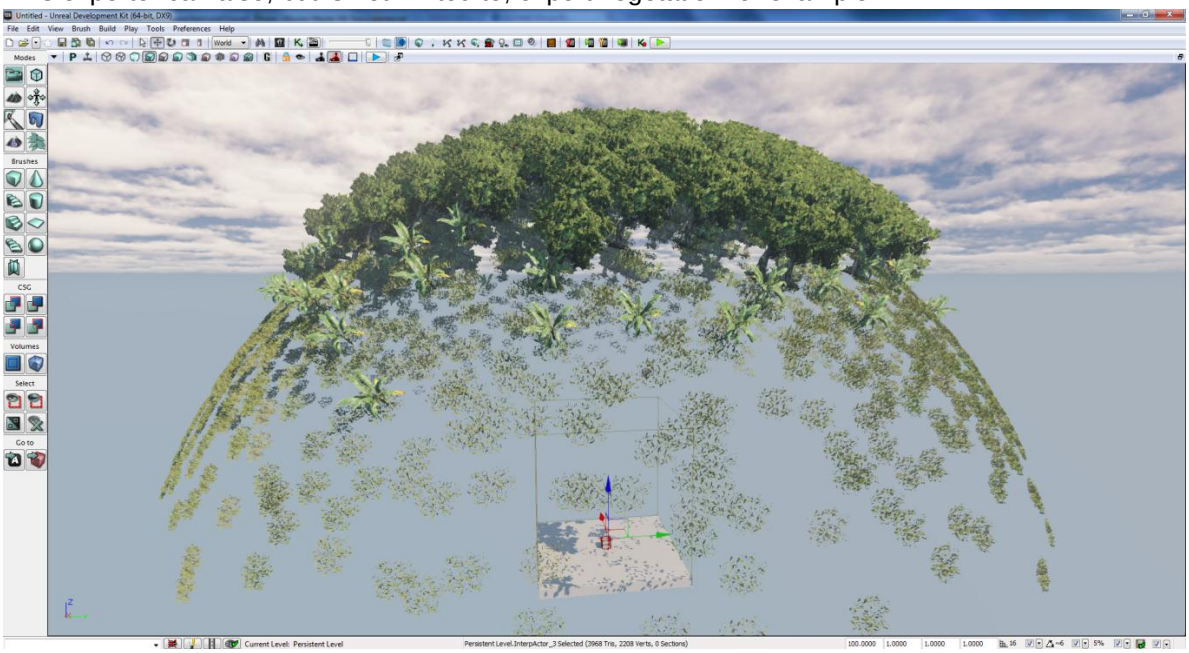
As mentioned before, the Y angle is not computed as the roll value cannot be computed on only a normal, as it is only a vector with a direction and without a secondary point to identify a roll value with.however it is possible to input the value from outside (random Y rotation for example is possible).



As one can see the exported positions have been translated successfully from Houdini to Unreal, with only the Unreal X rotation not known (and thus left to 0) resulting in a set of beams following the curve perfectly, but always looking up with Unreal Z.Although it is also possible to give it any rotation it cannot be extracted from the normal (input the point number * 36 into the rotation to get a full revolution every tenth beam for instance).

Y = UDK Z positive
Z = UDK X positive
X = UDK Y positive.

This exporter can also, but is not limited to, export vegetation for example:



An example of exported vegetation on a sphere, the vegetation is properly aligned along the normal, and follows the curvature of the sphere well. The input for the range map is the Y normal.

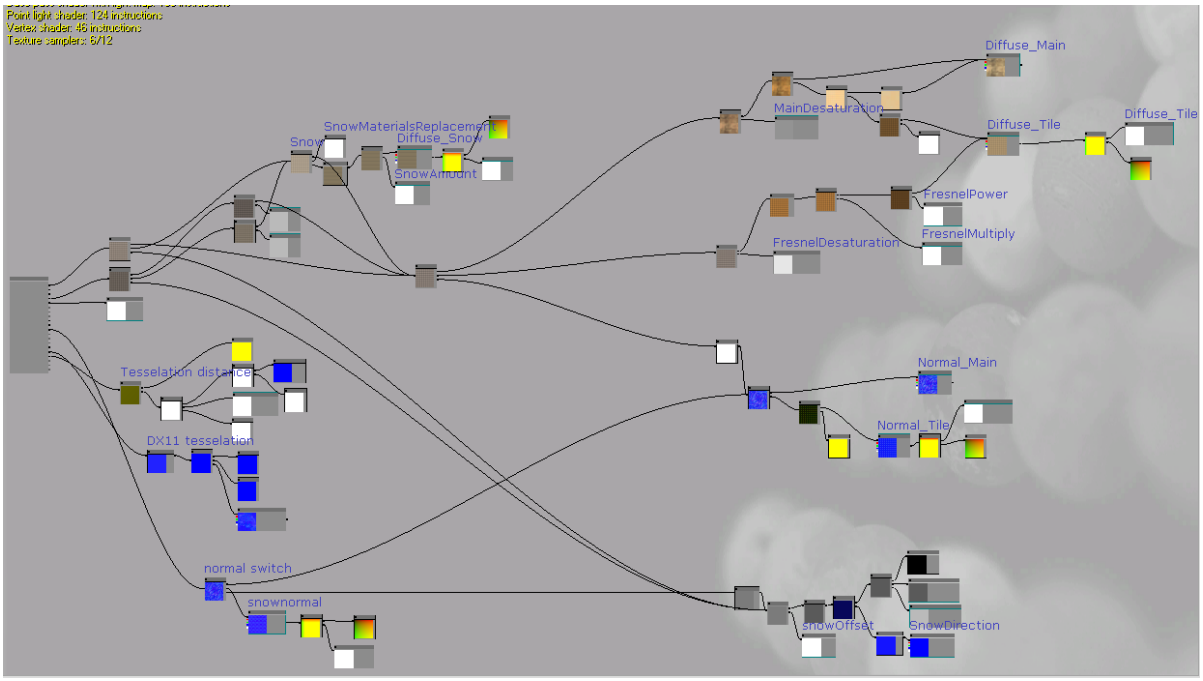
8. The UDK shader

As vertex colours cannot be exported into the unreal engine (as the apprentice HD version of Houdini does not export .fbx) I have created a backup option. The shader setup created is most commonly known as a snow shader and applies snow or any other material at the top of surfaces (based on the normal).

In this case I used it to make sandy paths with rocky walls. Furthermore the shader has several multiplications of textures in it to get high detail textures at any distance to the surface without showing tiling. The shader is finished off with a subtle Fresnel to “soften” the looks of the rocks and avoid the common “plastic” look in Unreal.

A benefit is that the shader can also be applied other objects, dynamically having sand on the top regardless of rotation. By changing the input colour the shader uses a blue input to define Z-up, the user can use the opposite sundirection worldnormal to automatically generate moss to the opposite side to the sun’s direction.

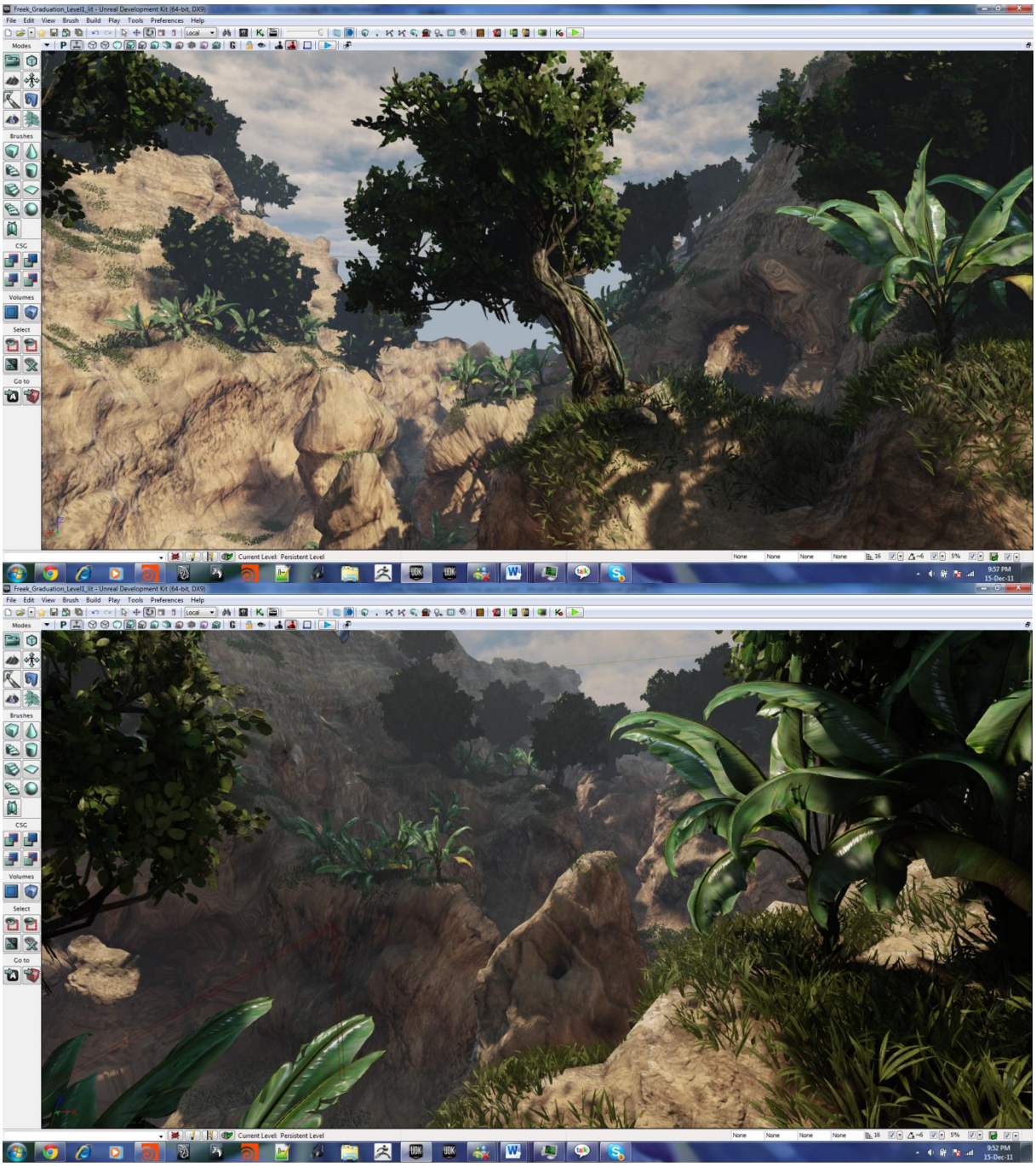
Regrettably the shader is quite heavy, mostly due to the realtime computation of the snow shader, which impacts the frame rate quite a lot. This could not be avoided as I could not work around the lack of .fbx exporting of this version of Houdini, nor did I want to seriously impact the visual quality at this time. Although the realtime nature of the shader is convenient for other objects which can be rotated, it would be beneficial for the frame rate if the vertex colours could be used on the terrain as it will not rotate and often occupies as much as 50% or more of the screenspace and it’s shader should therefore ideally be as light as possible.



A DX11 renderpass was also added to test realtime displacement’s results. However it is very computationally intensive as the shader is not aware of the camera frustum and apparently tessellates anything purely based on distance, even objects that are outside of the screenspace.

In DX9 mode it has no impact on the framerate so the implementation was left in the shader.

9. the final result:



Screenshots taken straight out of the UDK. No postprocessing or modification to either the image or level has been done after importing. These examples will also be delivered with the project in the UDK folder, although a fast PC is required to run at a decent frame rate.

10 Conclusion:

Design wise the method developed is indeed a lot faster than generating similar environments manually. Additionally adjustments and updates can be done in minutes regardless of the severity of the change. Multiple people can work on the same terrain without interfering with each other and the iteration is never held back by the decoration department. The procedure is not nearly as limiting as height maps are to the user, making it possible to integrate (overlapping) height levels, steep cliffs tunnels and overhangs.

Visually the true 3D nature of the geometry combined with decent UV's also makes for a much more compelling image, with a lot more creative freedom for the user to customize the look of the level. also the integration between floor and rock surface is a much more natural one then a blunt intersection as is commonly used now in games.

I truly believe that procedures like these will be the way forward especially for games with large environments that are specific in design look and feel. Games such as SSX or DIRT or GTA with a large environments that are very specific would benefit from this tool as it generates high quality terrain very fast, without sacrificing control or customizability.

Even on a more modest scale the tool might be useful to design levels in 3D instead of a 2D map, or to decorate handmade levels, retopologize and decorate levels made in other tools like the UDK BSP, or even just to sculpt several static mesh rocks as these operations take seconds instead of hours of sculpting and easily be modified.

10.1 : Known issues.

- At times the ISO offset node will fail to compute properly. Recomputing or turning laser scan off and on again fixes this issue.
- Making a canyon very close to the edge may create gaps in the wall. Making the level a bit wider or moving the path away from the edge solves this problem.
- No smooth terrain on the bottom of tunnels. It is possible to add this via the normal, but it was lacking control. For this reason I chose not to add it just yet. The same applies for extrusions.
- UV generation is very slow. I preferred good UVs over fast ones.This should therefore only be turned on as the last step before exporting.
- Very thin long pillars or deep gaps in the terrain will result in bad UV-stretching. And UV intersection might sometimes still result in some weird UV behaviour. An example can be found in the demo level. It was left as these would need manual modification or need to be avoided.

11. References:

Other peoples work used:

- **Kim Goossens**
  Extended Sweep as well as support.

- **Jan Pijpers,**
  the Python .t3d export script, allowing me to export positions of meshes into unreal.

- **AxeBeak** *(Username on odforce,)*
  For posting a great ambient occlusion VopSop.

- **Jwoelper** *(username on odforce)*
  For posting an example file for creating procedural stone.

11.1 Tutorials

Procedural road creation ~ by Kim Goossens
Website; Video Tutorial: 3 parts.
Publisher: CMIVFX (03-03-2010)
Language: English
Source: www.CMIVFX.com

Hip tricks ~ by Alvaro Castenada
Tutorial: 4 parts.
Publisher: CMIVFX (29-06-2011)
Language: English
Source: www.CMIVFX.com

Fractal clouds and terrains ~ Andreu Lucio
Tutorial 1 part.
Publisher: CMIVFX (23-02-2011)
Language: English
Source: www.CMIVFX.com

Houdini Tutorials ~ by Peter Quint
Website, Vimeo Channel
Publisher: -
Language: English
Source: http://vimeo.com/channels/54102

SIDEFX.com ~ by Various artists
Website; video tutorials
Publisher: SideFX Software
Language: English
Source: www.sidefx.com

11.2 papers

Realtime Procedural Terrain Generation ~ by Jacob Olsen
Paper, 20 pages
Publisher: University of Southern Denmark (31-10-2004)
Language: English
Source: http://oddlabs.com/download/terrain_generation.pdf

Special thanks go to:

Kim Goossens: for introducing me to Houdini, and never-ending support as my Supervisor.
Mata Haggis: for providing feedback on the process of level design and testing my procedure.
Andrew Paquette: for proofreading this dissertation and never being satisfied with my work.
Tessa el Miligi: for proofreading this dissertation, moral support and Pushing me to work harder.