Cloud function tutorial

By Martin Huisman

Table of Contents

Introduction

With the release of Terragen v2.2 several powerful new features are introduced. Among these new features are a couple of functions to modulate cloud positioning and appearance. This tutorial will explain these new cloud features, covered by example animations and images. Ultimately I will show you a way to make these new features work for every scene you work on.

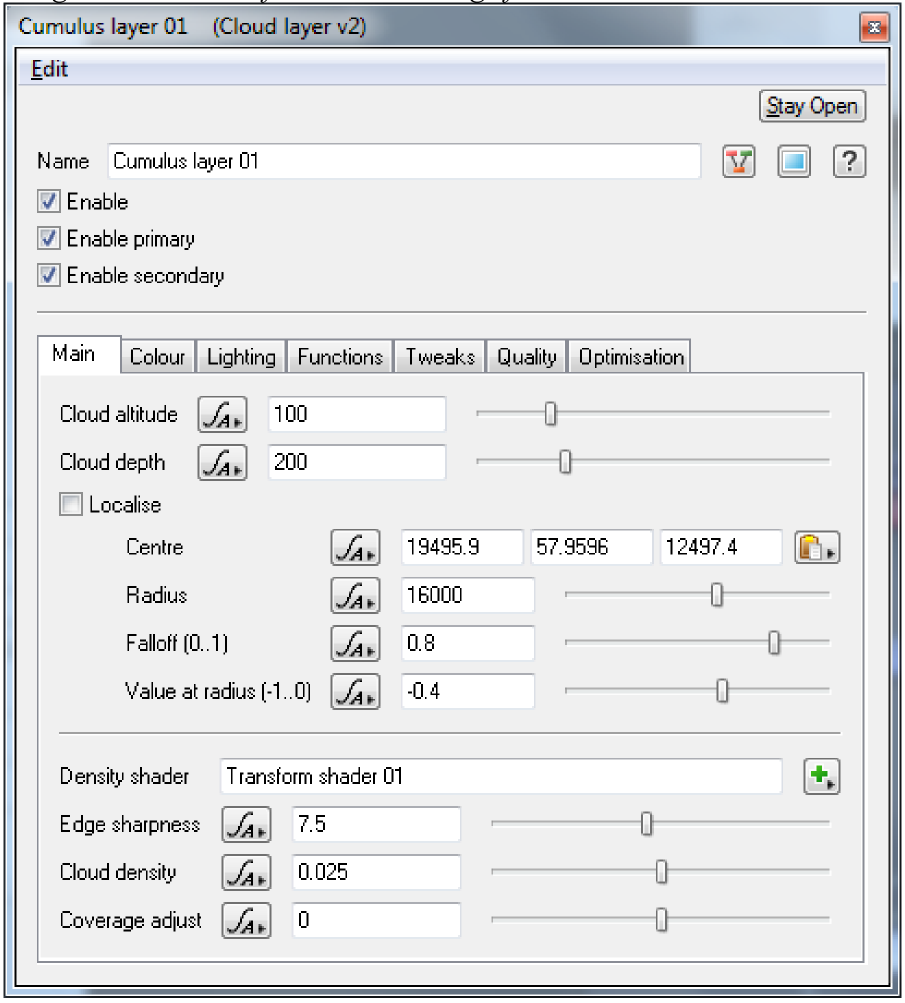
This tutorial is accompanied with a Terragen project file and terrain image file to reproduce the steps and examples I'll show you. Remember that I've been using Terragen Deep + Animation. So if you already have Terragen Deep + Animation you'll be able to see the keyed parameters, otherwise you can compare start and end values for example. The nicest thing is to experiment right away of course.

Ok, let's get started!

Part 1: Understanding the cloud shader node

The relation between cloud altitude and depth

Image 1: overview of the main settings for the cloud node included in the example scene.



Cumulus layer 01 (Cloud layer v2)

Edit

Stay Open

Name	Cumulus layer 01
☑ Enable	
☑ Enable primary	
☑ Enable secondary	

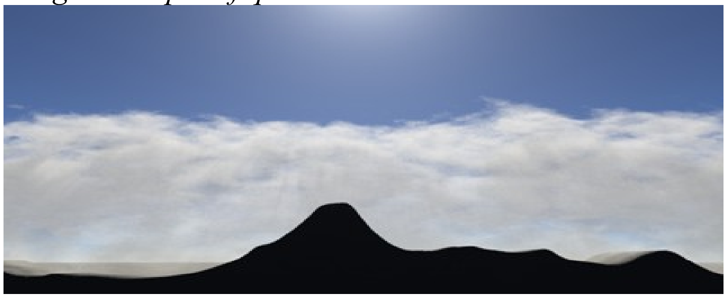
Main | Colour | Lighting | Functions | Tweaks | Quality | Optimisation

Cloud altitude	100		
Cloud depth	200		
☐ Localise			
Centre	19495.9	57.9596	12497.4
Radius	16000		
Falloff (0..1)	0.8		
Value at radius (-1..0)	-0.4		
Density shader	Transform shader 01		
Edge sharpness	7.5		
Cloud density	0.025		
Coverage adjust	0		

I will work through these settings from top to bottom. Cloud altitude and depth are already familiar to you, but in order to make the best use of the new functions it is critical to understand how the cloud depth and altitude work together.

Open the file “Cloud functions explained.tgd” in Terragen v2.2 and render the quick render. Your image should look similar to the following:

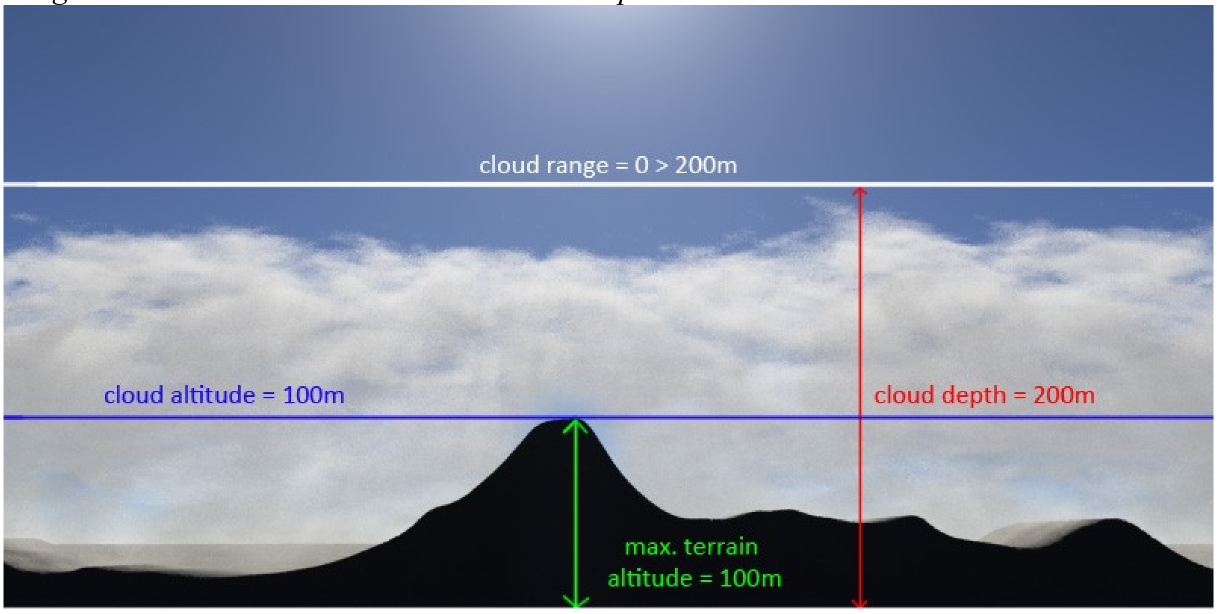
Image 2: output of quick render



The terrain is generated by a 1 x 1188px image. The 1px wide displacement is stretched in order to get a cross-section like terrain. The displacement factor is 100 which means that the maximum height of the terrain is 100m (if greyscale-value is 100% white). This way it is easy to see what's happening since you basically make the terrain kind of 2D.

So let's get back to the cloud node settings; how do cloud altitude and depth relate?

Image 3: relation between cloud altitude and depth.



Remember from "Image 1" that the cloud altitude is 100m and depth is 200m. The altitude defines the centre of the cloud. The depth defines the vertical range in which the cloud can exist in both directions from the cloud base. So in this case, with a depth of 200m the final depth will be:

Minimum cloud altitude = cloud altitude **-** (cloud depth / 2) = 100 – (200 / 2) = 0m
Maximum cloud altitude = cloud altitude **+** (cloud depth / 2) = 100 + (200 / 2) = 200m

So ultimately the range of the cloud layer is 0 to 200m with its centre at 100m. It is critical to remember that clouds are ALWAYS restricted to this defined range!

The new "Localise" feature

New in Terragen v2.2 is the "Localise" function for clouds. Usually when a user wanted to restrict clouds to a specific part of the terrain he/she had to use a blend shader of some kind, such as a Distance Shader, to mask the cloud's noise-function. When doing this the cloud layer is still "global", since we haven't restricted the cloud-layer itself, but only its noise function. The benefit of using a localised cloud is that the cloud layer will change from "global" to "local" and therefore limits the area where cloud calculations are required. This can result in much faster renders.

The size of this localised cloud is defined by the localise settings in the cloud node, where:

"Centre" defines the position of the clouds localisation. You can enter the XYZ coordinates manually or drag the cloud layer with the handles in the 3D preview.

The Y-coordinate of the Localise function is logically connect to the cloud altitude setting.

"Radius" is radius of the localised clouds in metres from its centre in all 3 dimensions.

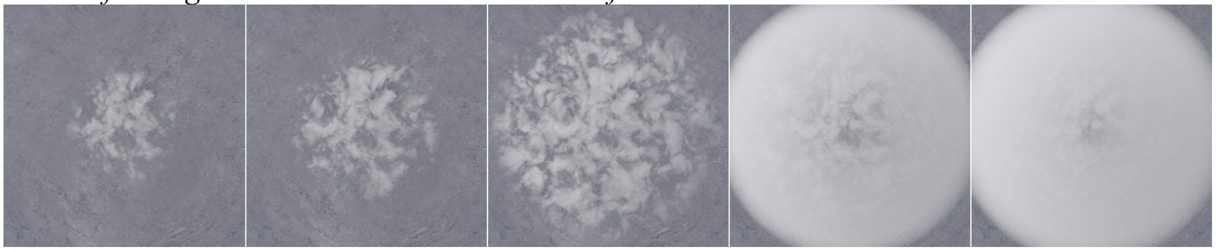
"Falloff (0..1)" defines the softness of the edge of the localised cloud layer.
0..1 indicates the ranges one could use, but as usual one can also use values > 1.

A value of "X" means that at 1 – X * radius the edge will start to fade out towards the edge of the cloud.

So with a radius of 1000m and Falloff @ 0.25 the edge will fade out from 1 – 0.25 * 1000 = 750m. So the falloff will be applied from 750m to 1000m from the centre. If the Falloff is 0.75 then the falloff will be from 250m to 1000m from the centre.

"Value at radius" replaces the value of the density shader at the perimeter of localisation.
In the falloff region, the Density Shader value is replaced by a blend between the original Density Shader value and the "value at radius". The blending is from the perimeter towards the localisation centre.

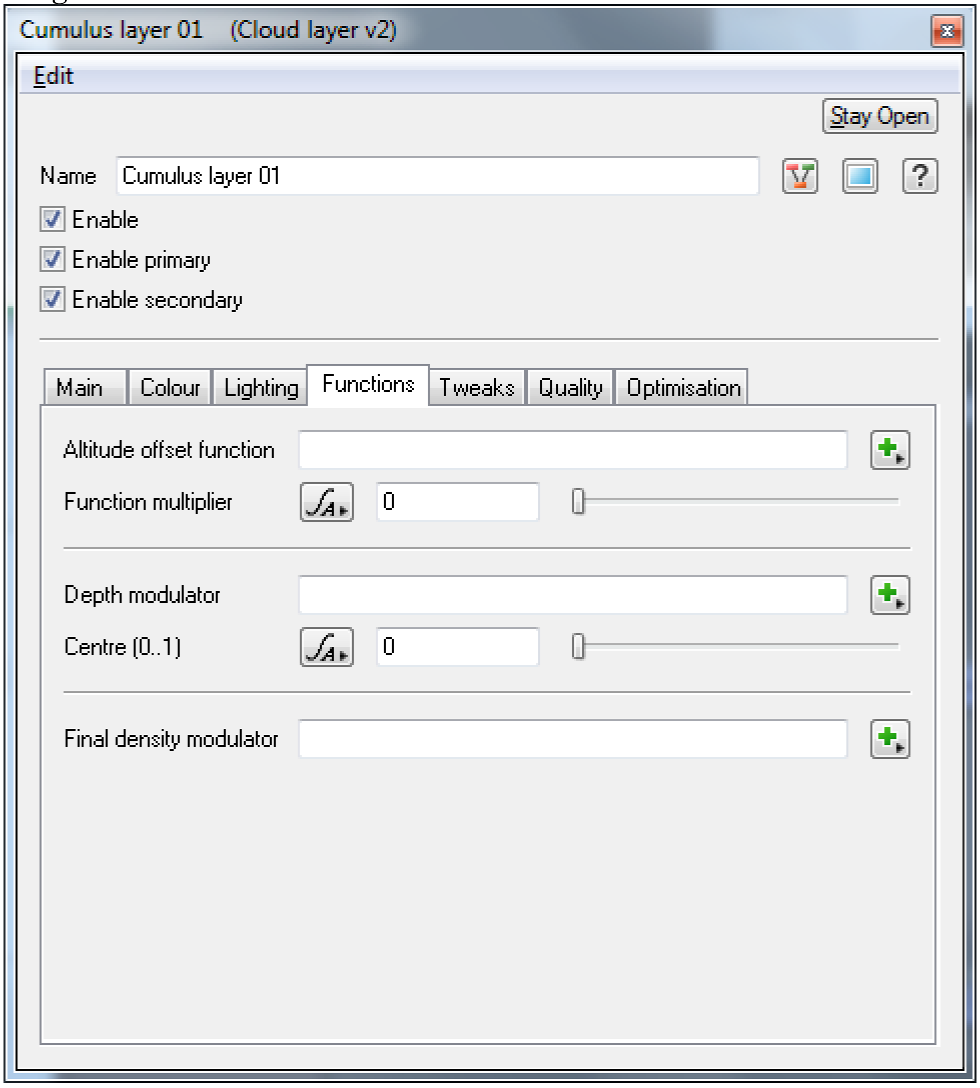
Image 4: The falloff is set at 1 for all images here.
From left to right the value at radius increases from -1 to 1 with 0.5 increments.



As you can see as With a value of 1, you will get cloud all the way up to the radius of localisation. Also, the empty spaces between clouds that you would normally get from the negative parts of the Density Shader are now completely filled with clouds. This is because the Density Shader value is completely replaced by the "value at radius" value. A value of 0 does not affect cloud coverage and provides a soft taper at the localisation perimeter, unless your clouds are too dense to make it noticeable. A negative value, 0.5 at default, will decrease cloud coverage towards the perimeter.

Part 2: The “Functions” tab

Image 4: The new “Functions” tab



Name: Cumulus layer 01

- [✓] Enable
- [✓] Enable primary
- [✓] Enable secondary

Altitude offset function:

Function multiplier: 0

Depth modulator:

Centre (0..1): 0

Final density modulator:

Again, let’s start from top to bottom, where:

“Altitude offset function” takes a function or colour shader which can vary in 3D space to offset the cloud's built-in vertical profile. A function could be an altitude based function made with the function nodes and a colour shader can be a powerfractal or a constant colour function node. The input-units are in metres. You can multiply the input by using the *“Function multiplier”* input-field.

For example, if you choose to use a constant colour function node as altitude offset function and set its value @ 1 then the cloud’s altitude will be offset with just 1 metre. It’s likely that you’ll not notice this in your renders. If you enter “100” in the “Function multiplier” input then the offset will be 100 metres.

Remember from Part 1 of this tutorial that the clouds cannot exist outside the boundaries defined by the cloud altitude and depth settings, so be careful with high altitude offset values!

The ”Depth modulator” takes a function or colour shader which can vary in 3D space to multiply or reduce the height of the built-in vertical profile. Basically you can use similar inputs here like in the altitude offset.

I find it easier to think about this function as a multiplier of the depth. So, for example, if you connect a constant colour function node to the “depth modulator” input port of a cloud node with a value of “X” to the Depth Modulator input then the depth of the cloud layer will be:

X * cloud depth.

If you use a value of 0.1 then the depth wil be 0.1 times, so 10 times less thick. In our project file the cloud layer would 0.1 * 200 = 20m thick.

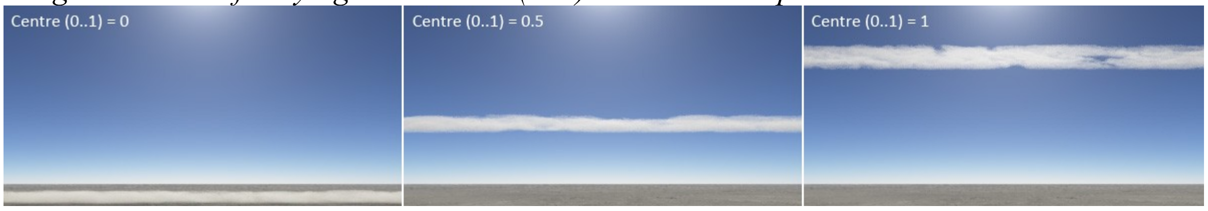
The setting for “Centre (0..1)” defines where this thinner cloud layer will appear within the defined cloud-range. “Centre (0..1)” defines the pivot about which the depth is multiplied.

Let’s get back to the project file included with this tutorial:

1. Disconnect the nodes which define the terrain and keep the compute terrain enabled.
2. Go to the cloud node and give “constant colour 01” a value of 0.1.
3. Inside the cloud node, in the functions tab, enter a value of 0 in the “Centre (0..1)” field.
4. Perform the quick render and change the value to 0.5 and 1, but keep the value of “constant colour 01” at 0.1!

The 3 renders should look like this:

Image 5: results of varying the “Centre (0..1) value when depth modulation = 0.1



If you compare the depth of the cloud layer with “Image 2” or “Image 3” you can see that the depth is 10 times less.The centre value allows you to control where the depth reduced cloud is positioned within the clouds boundaries. Bottom = 0, half-way = 0.5 and at the top = 1. Once again, remember that these boundaries are defined by cloud altitude and depth in the “Main” tab.

Click **HERE** to see how this looks animated.

Part 3: Combining altitude and depth modulation

Clouds following a terrain

Ok, so after all this boring reading we finally get to the most relevant part which is: how can we use this to make cool stuff? ☺

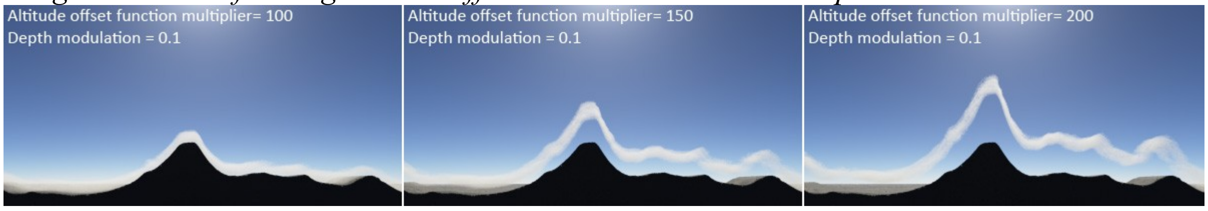
In the accompanied project file re-enable the nodes which generate the terrain. Take a look at "Image 3" to remember that the terrain has a max altitude of 100m.

Now let's connect the "Image map shader 01" node to the altitude offset function input of the cloud node and make sure that the "constant colour 01" is set at 0.1.

The image map defines the shape of the terrain using greyscale values ranging from 0 to 1. As you now know the input of the altitude offset function is in metres, so the input of this image map won't have a noticeable effect, unless we multiply it with a value.

Since the displacement factor is set to 100 in the displacement shader we know that the terrain is 100m high at maximum. So, consequently, set the altitude offset function multiplier to 100 as well and render the quick render. Repeat this for values of 150 and 200 for the multiplier. You should have results similar to this:

Image 6: results of testing altitude offset with a reduced cloud depth


Altitude offset function multiplier= 100
Depth modulation = 0.1
Altitude offset function multiplier= 150
Depth modulation = 0.1
Altitude offset function multiplier= 200
Depth modulation = 0.1

As you can see in the left example the 20m thick (remember, 0.1 * 200) cloud layer is exactly following the terrain, because it uses the same multiplier value as the displacement value. This matches exactly as well, since the source of the information for the displacement and the altitude offset comes from the image map.

In the right example you can see that the cloud layer almost reaches the boundaries of the cloud layer. Apparently the terrain max altitude is not exactly 100m and therefore the offset isn't +100m as well.

Take a look at this short animation where the cloud altitude offset starts at 100 and increases to 150. In the meantime also the depth modulation increases from 0.1 at the beginning to 0.5 at the end.

Click **HERE** to view the animation.

As you can see the cloud starts to hover above the terrain and becomes thicker!

Part 4: Limitations of these new functions

As you noticed I used an image map to generate the terrain and input for the cloud altitude offset. The reason for this is that it would be really slow to compute the terrain's altitude straight from your network and use that as input for you altitude offset. So, to have this work fast it is better to use an image-map. A dis-advantage of an image-map is that you can only locally apply the effect. You can export bigger areas of course, but like many things there are limitations.

If your image map contains very steep features it will look like your clouds aren't following the steep features, since the colour values are differing too much. This is most evident in animations as it will look like the clouds are coming out of the terrain.

A solution for this is to blur your image map to make the transitions less, but blurring large contrasts also has its limitations. In general you should blur your image map anyway in order to have the clouds follow the terrain more smoothly.

In this short animation the limitations are shown pretty well. In the background at the left of the centre the clouds follow the terrain pretty well, but to the right it looks like the clouds appear from the rocks. In fact, they don't, they are just quickly shifted down by the altitude offset! Nonetheless, it doesn't look too well.

Click **HERE** to view the animation.

Part 5: Generating an image map from your procedural terrain

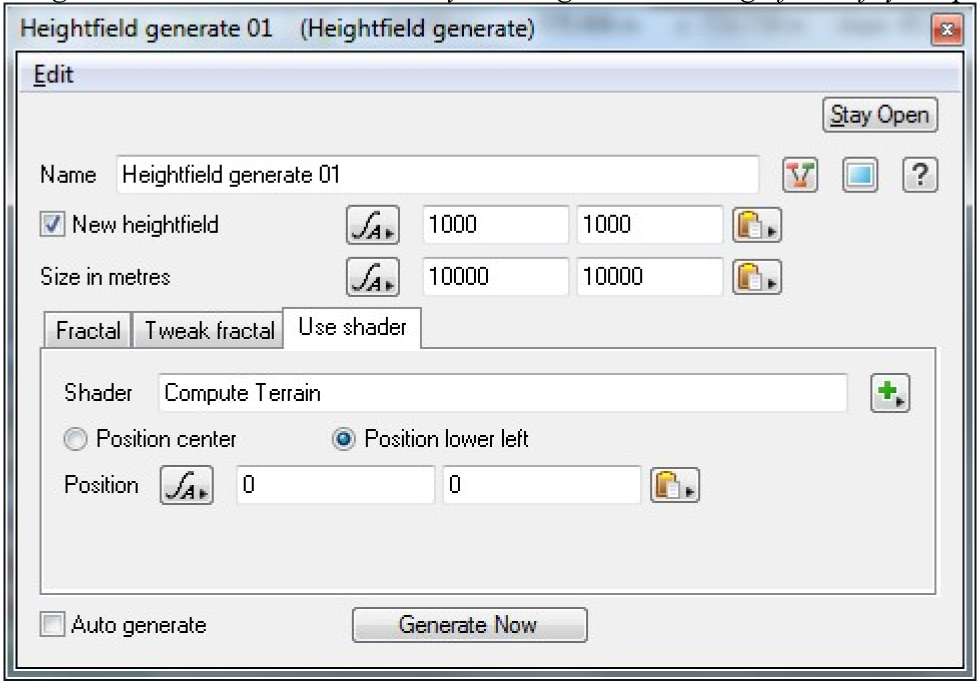
If you have World-Machine or Geocontrol generated terrain you can easily export your terrain as an image and use that straight away in TG2. However, it might need some research to find the right altitude offset multiplier values.To overcome this you could generate the displacement of the terrain by using the image map shader as input for a displacement shader. If you use the same altitude offset as the displacement value than you have it matched absolute correctly. An example is already in the project file with this tutorial.

But what if you do not have either of these two programs?

Open "image map tutorial.tgd" and in the node-network go to the terrain group. There you will find one fractal terrain and a heightfield generate shader.

Go to the heightfield generate shader and navigate to the "Use shader tab".

Image 7: "Use shader" tab where you can generate a heightfield of your procedural terrain.


Heightfield generate 01 (Heightfield generate)

Name: Heightfield generate 01

[✓] New heightfield: 1000, 1000

Size in metres: 10000, 10000

Use shader tab:

Shader: Compute Terrain

Position center [✓ Position lower left]

Position: 0, 0

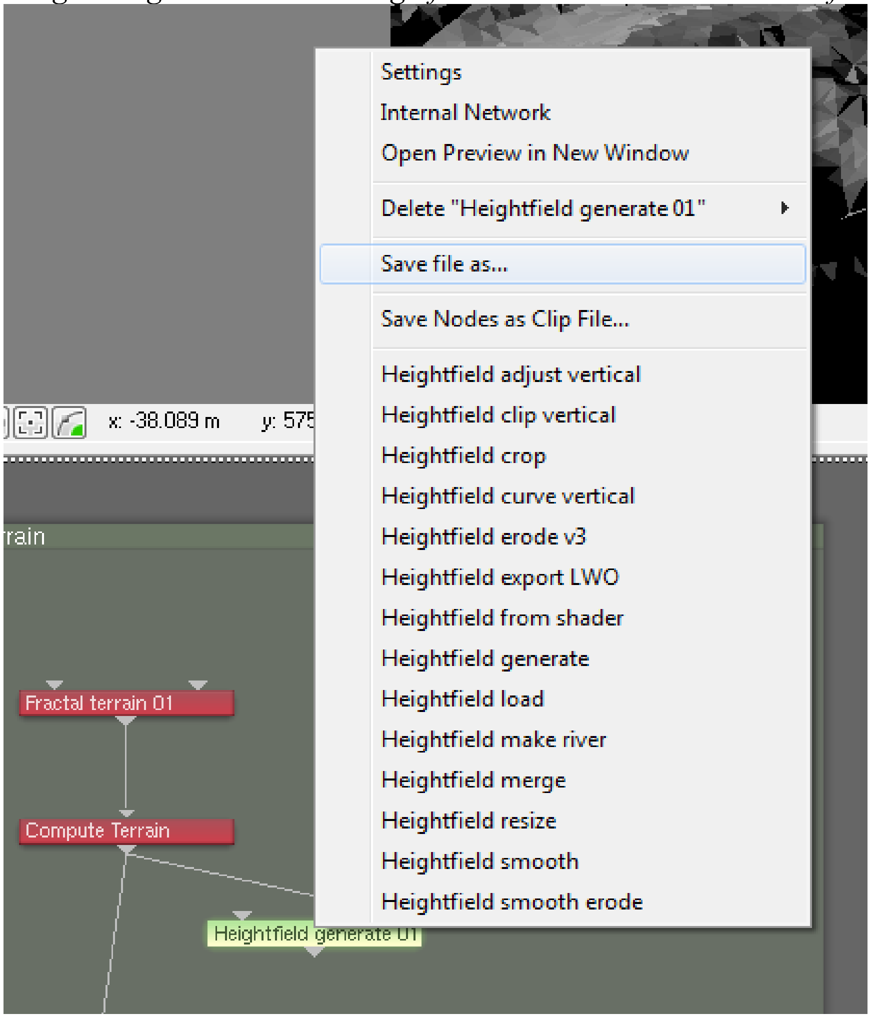
[ ] Auto generate

As you can see the heightfield generate will use the compute terrain node to generate a heightfield.

Very important is the coordinates and type of positioning you use. Later when you have your image map you should use the same coordinates and type of positioning.

Hit generate and after it's finished close the node. Then context-click (right mouse button) on the heightfield generate node and choose "Save file as…" and save your .ter file.

Image 8: right click on the heightfield shader and choose "Save file as…" to export.



Using Terraconv to convert your .ter file to an image

You can find TerraConv here:
http://koti.mbnet.fi/pkl/tg/TerraConv.htm

Start TerraConv and open your exported terrain. After loading click on Export and choose 16-bit .tiff export.

This .tiff file will not load into TG2's image map shader. For 16-bit accuracy you would require a SGI-image format. It's not necessary to use 16-bit, so you can also save the .tiff to a 8-bit .BMP.

You can find a SGI plugin for Photoshop here:
http://www.telegraphics.com.au/sw/#sgiformat

In Photoshop open the .tiff file and apply a little bit of blur. I wouldn't go beyond 8px of Gaussian blur. Usually I add around 4px. Save the file as a SGI file.

You can also use the free XnView image viewer and editor to perform some of these functions.
http://www.xnview.com/

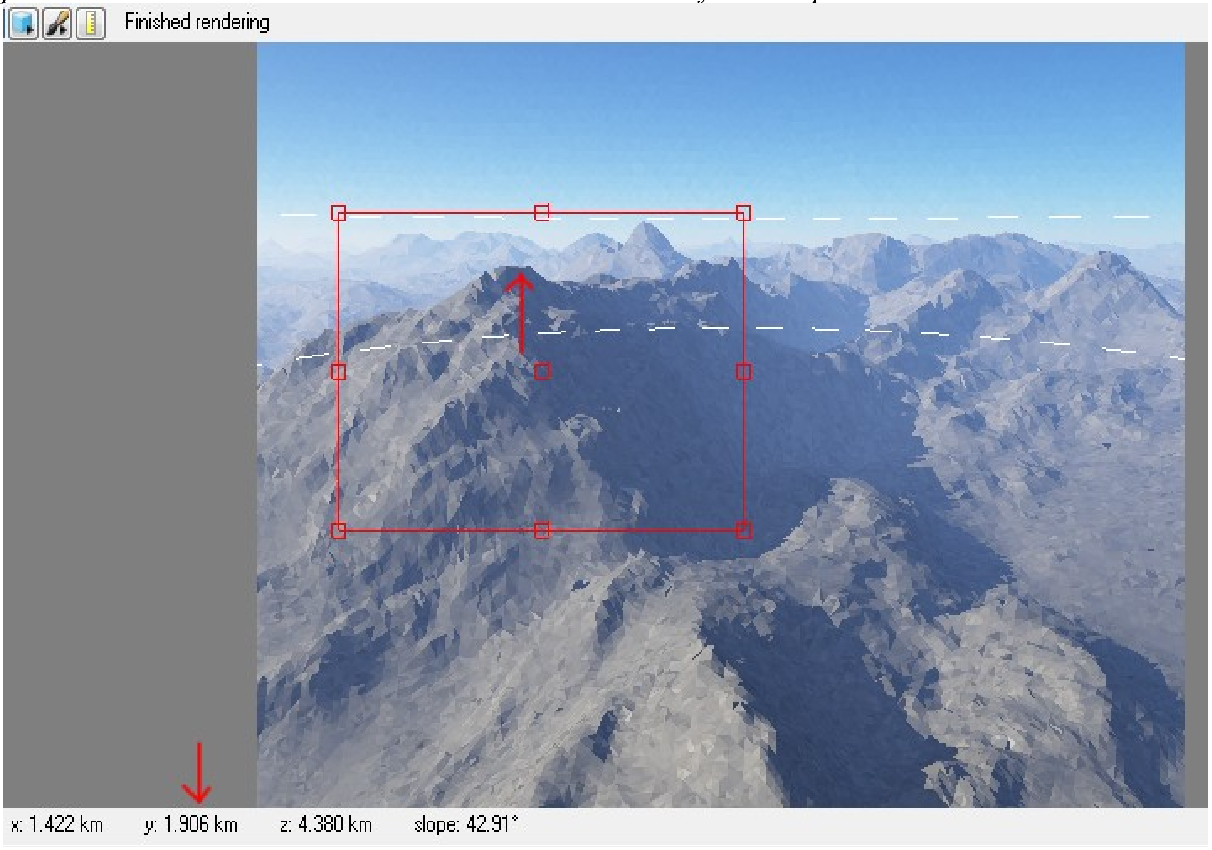
Importing and matching the image map in TG2

Create an image map shader and load the SGI or BMP.

Enter the same coordinates, coordinate type and size as you can find in the heightfield generate node. In our example file this is:
Coordinates: 0, 0, position lower left and 10000 x 10000m.

Create a cloud layer and match the cloud altitude and depth in such a way that the cloud layer covers the full height-range of the image. Let the preview render finish and pause it. Move your mouse to the highest mountain top and read it's altitude at the bottom of the 3d preview. You'll see it's around 1900m.

Image 9: at the top arrow I "measure" the altitude by hovering with my mouse over the 3D preview and I can read out the altitude at the bottom of the 3D preview



x: 1.422 km   y: 1.906 km   z: 4.380 km   slope: 42.91°

So set the cloud altitude at around 1000m and depth to 2000m. The cloud boundaries will just reach the highest peak of the terrain. To avoid clouds being cut off increase the cloud altitude and depth a bit further, to 1400 and 2800 respectively. Now we're sure the cloud can potentially cover all parts of the terrain.

Connect the image map shader to the altitude offset function input port of the cloud node.

Now comes the bit tricky part; finding the right altitude offset. This comes to a bit of trial and error, but there is a way to make it at least a bit easier. For starters, use a constant colour function node as density shader and set its value to 1. Now you have clouds everywhere. Increase the density of the cloud a bit to make it better visible.

As you should know by now we would also need a thinner cloud layer. So, create a constant colour function node, set its value to 0.15 for example, and connect it to the depth modulator input port.

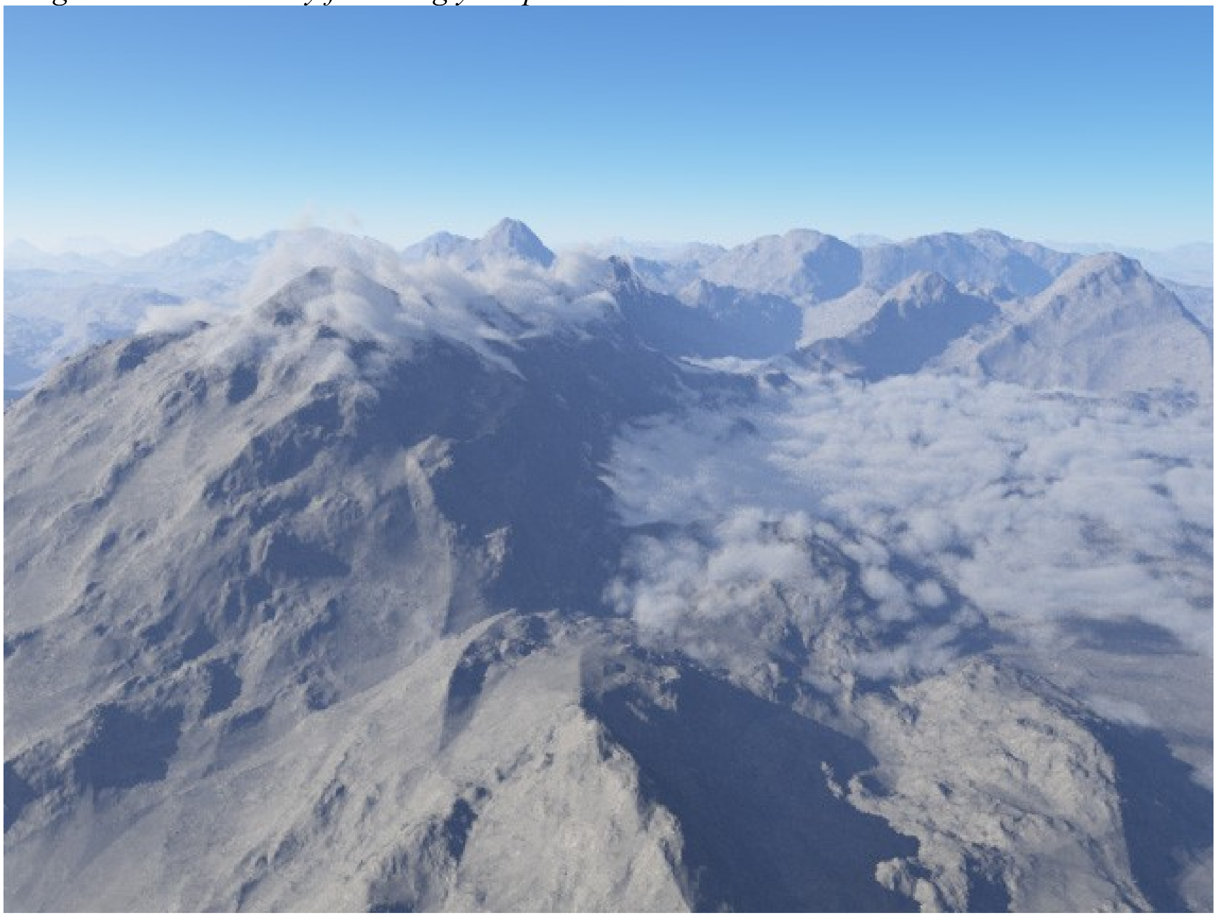
We know the mountain top is around 1900m so it would make sense to start with an altitude offset of around 2000. You'll notice that that offset isn't big enough. Increase it with 100m increments until you're satisfied.

Once the altitude is to your taste you can replace the constant colour which acts as density shader with the cloud fractal shader you disconnected.

If you want to change the thickness of the cloud layer then keep in mind you might need to re-adjust the altitude offset.

Here's how it looks after finishing:

Image 10: clouds locally following your procedural terrain



Part 6: The final density modulator

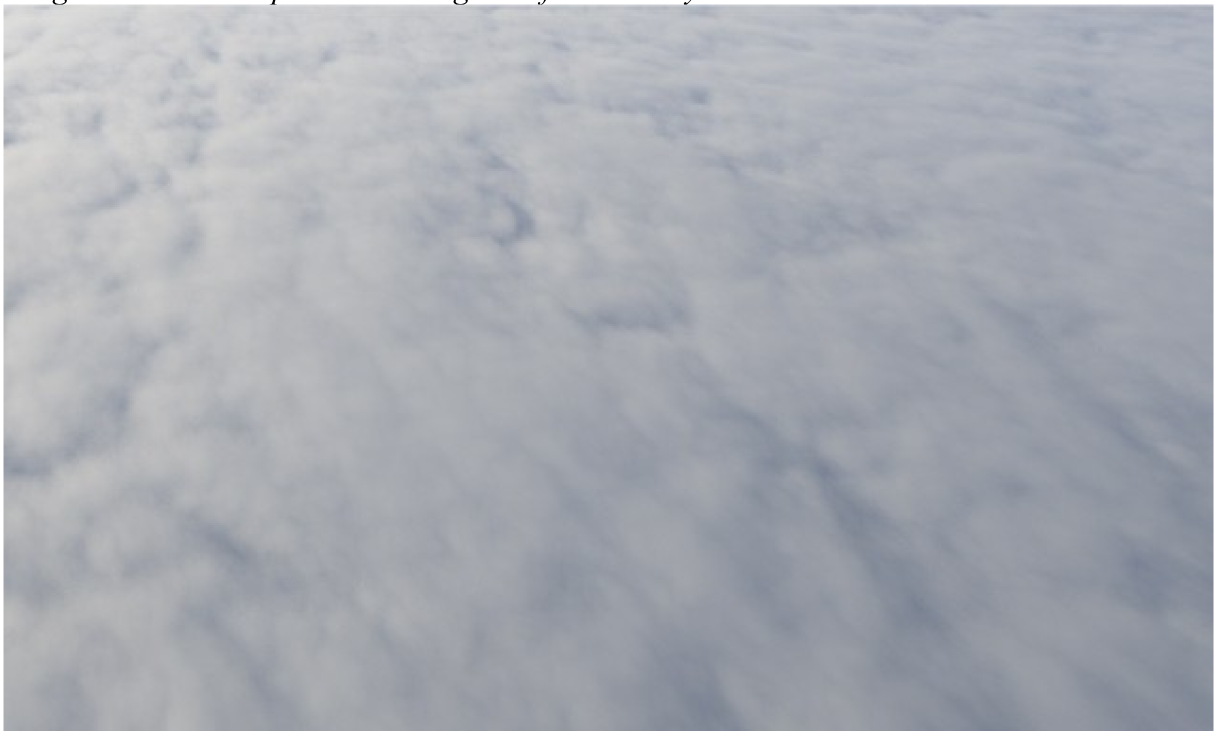
The final density modulator takes a function or colour shader which can vary in 3D space to multiply or reduce the final density of the cloud. This is being calculated after all other built-in clipping and cutoffs have been applied (by other cloud-functions, for instance).

This function allows you for example to have the cloud density change over altitude or distance. Also, you can add variation in density by feeding this function with a power-or cloud-fractal. In case of unclamped functions/fractals the final density can be modulated towards a density which is actually higher than the set density in the cloud node's main tab. So bear that in mind.

I will show you an example below. The project file for this example is included as well.

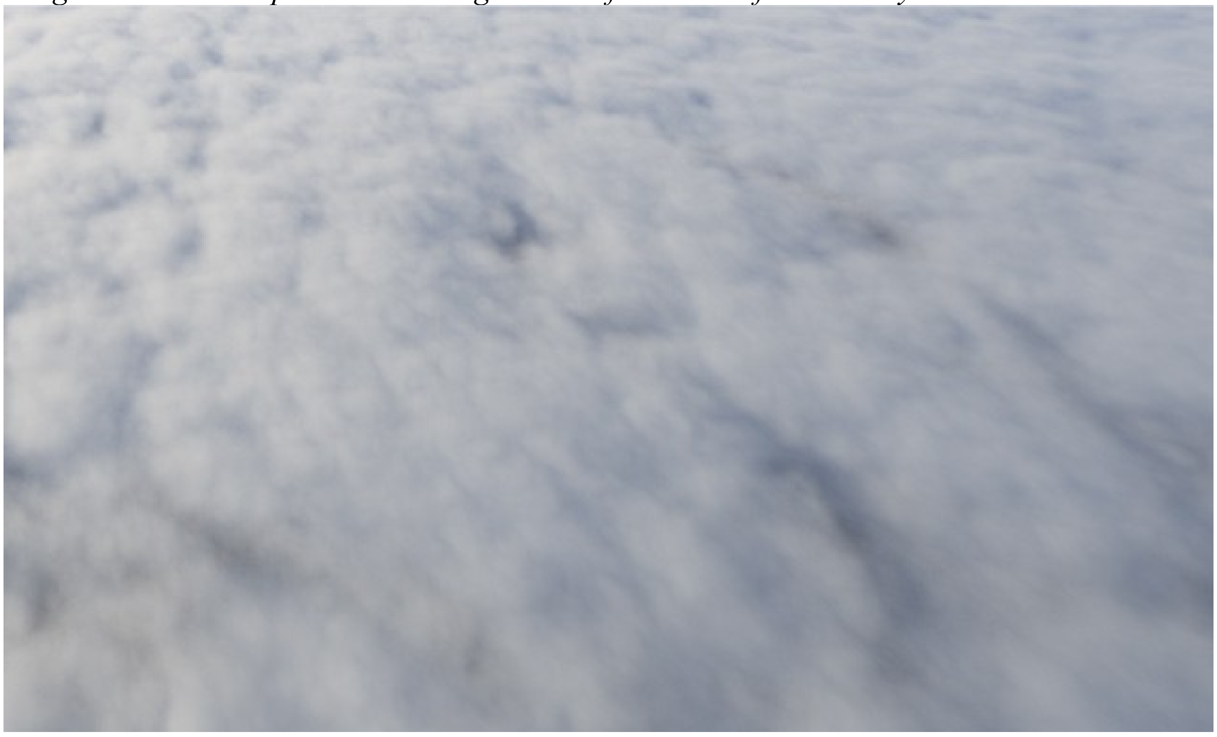
Here I've created a basic cumulus cloud layer at low elevation. The cloud fractal goes through a colour adjust shader which removes small floating bits of clouds (by increasing the blackpoint a bit) and adds some contrast (by reducing gamma). This setup functions as the density shader for the cloud:

Image 11: cloud setup without using the "final density modulator".



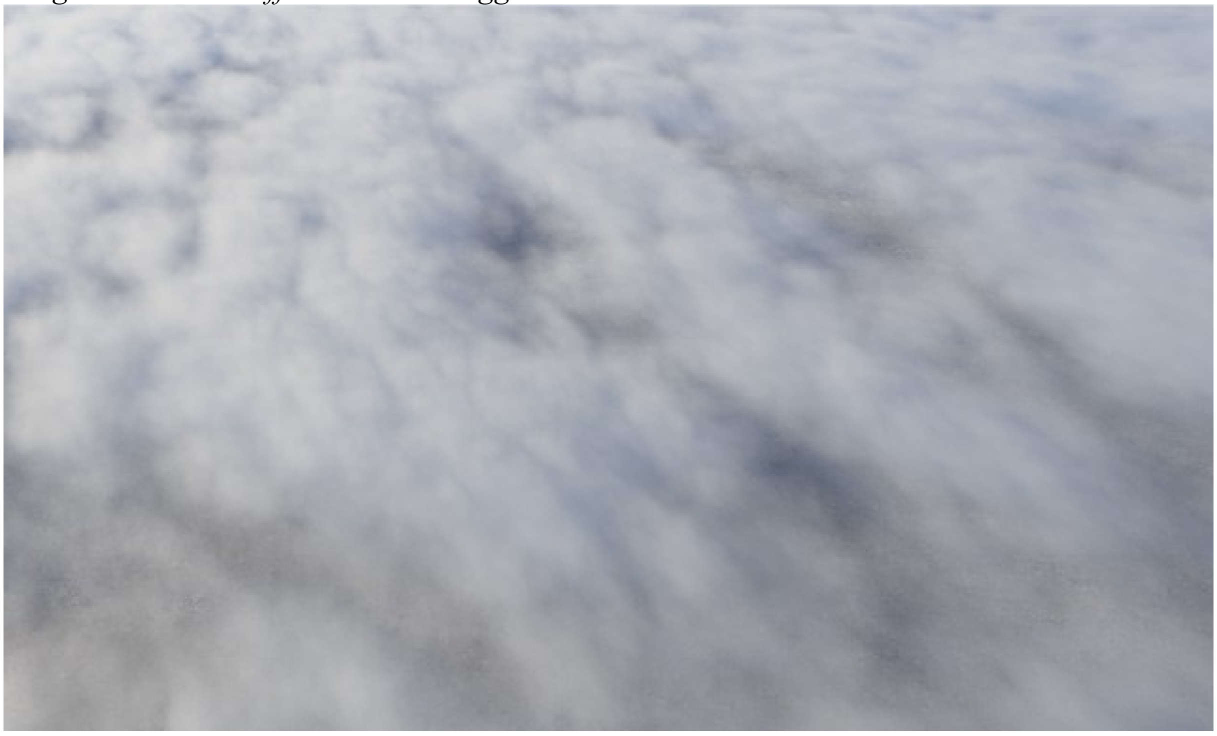
As you can see a basic cloud layer which lacks a bit of definition. A way to avoid this is to start tweaking your fractal by increasing contrast and adding some more roughness, but it is likely you will lose the original shapes. You can add extra interest by using the final density modulator function. In this particular case I use the cloud fractal itself as final density modulator. Except for that I reduced the colour range by increasing the white-point of a colour adjust shader and retained contrast by lower gamma a bit further.

Image 12: cloud setup now with using it's own fractal as "final density modulator"



The effect is quite subtle, but also effective. You could even take it a bit further, by further increasing the white-point a bit and reducing the gamma a bit as well.

Image 12: here the effect is more exaggerated



As you can see the overall coverage is still the same and only density is reduced in areas where the coverage is relatively low. Without using the final density modulator function you'd loose coverage of clouds if you'd only tweak the density fractal.

That concludes this tutorial on the new cloud control functions in Terragen 2.2. I hope you enjoyed learning about these powerful new functions and that you can put them to good use. If you have any questions you can visit the **Planetside Forums** for more information and help.

For more information on Martin Huisman's work and more great resources, visit his **Portfolio Gallery** and the main New World Digital Art site at **http://www.nwdanet.com/**