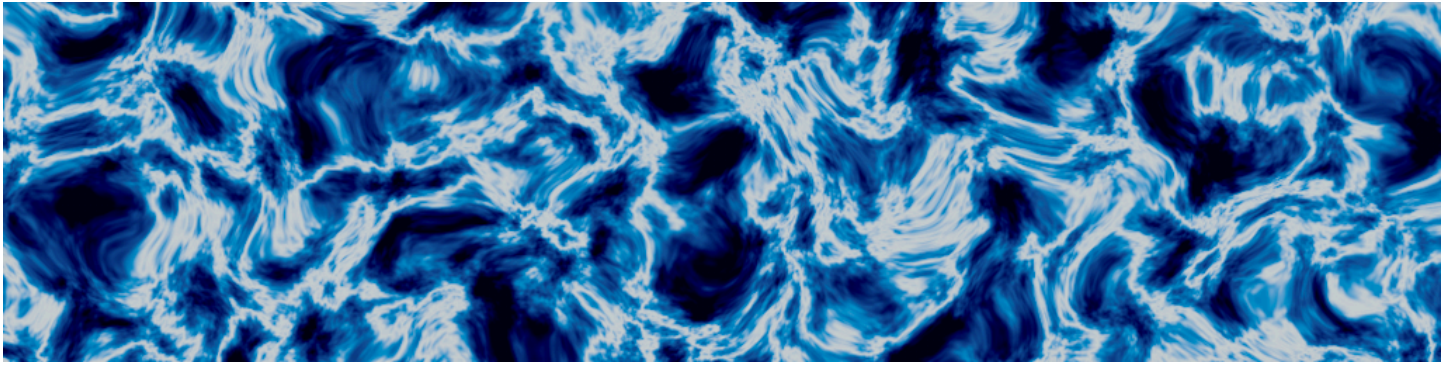


Microbion Libnoise

A shader plugin for Cinema 4D



Introduction

'Noise' in Cinema 4D can be used in shaders and various other operations to generate patterns or random variation. Cinema 4D contains one of the best noise shaders around. This was originally part of a suite of shaders and materials called 'Smells Like Almonds' from the company Bhodinut. Many or all of these were incorporated directly into C4D, so as well as the Noise shader there are several materials (not channel shaders) such as Cheen, Nukei and so on which are still present in the latest version of Cinema but which can't be used in Redshift, only in the standard renderer. There are other shaders too, such as the Falloff shader and Fusion, which also formed part of that suite.

However, there are few other noise shaders available for Cinema. There were some good noises in the old Darktree shader system and Biomekk produced a large set of shaders, including noise shaders, in a plugin called Enhance:C4D. Very sadly, neither of these are available now. Other render engines may have their own shader systems including noise shaders; for example, Cycles 4D from Insyrium has its own noise shaders. However, there are few, if any, other dedicated shader systems for any C4D user regardless of render engine.

Third-party noises

To expand the range of noises available, the only option would be to look for third-party noise libraries that could be incorporated into a Cinema plugin. A search for 'C++ noise library' on Google returns a lot of hits but the results fall into three categories:

- yet another implementation of Perlin noise
- specific noise algorithms, such as Worley noise
- actual libraries of noise functions

Looking at the last of those, most refer back to a library called 'libnoise'. This was developed some years ago - it seems that it stopped development around 2008 - but it is [still available](#) and still well regarded.

Libnoise

This may or may not have ever been incorporated into a shader for C4D. The Bhodinut noise either predates it or appeared at around the same time, but whether it is based partly on libnoise is impossible to say. The same is true for other shader systems or render engines. But if it hasn't, then possibly it might offer C4D users some additional noises, although the number of noise algorithms included is relatively small.

The original libnoise was intended to build a DLL which programmers would then load into their own software. This isn't useful here. For one thing, it was intended for use in Windows, and the pre-built binary is a 32-bit DLL at that (so might have to be rebuilt for 64-bit machines) and in any case the DLL could only be used by

someone writing a shader. However, the full source code for libnoise is available and could be ported to other operating systems, or - and this is the point - incorporated into a C4D shader directly.

Libnoise generates true 3D noise, so it can be applied to a 3D object and, like the inbuilt C4D noise, the result is seamless when rendered (but see the limitations section below). The number of noise algorithms is small though - Perlin, Billow, Voronoi and Ridged Multifractal are the main ones, though there are some interesting patterns such as Cylinders and Spheres. The usual range of parameters such as the seed value, octaves, lacunarity and so on are available, along with others such as persistence. These work very well in a C4D shader and there are some advantages over the inbuilt noises - for terrain generation, libnoise seems to generate more realistic results.

Libnoise can also produce 2D texture maps in addition to procedural 3D noise. These can be flat or spherical, and might be useful when a bitmap is needed rather than a procedural shader. This isn't something that should be done from a shader. To generate these maps really needs a separate, dialog-based plugin.

There is one disadvantage. Libnoise can combine multiple noise modules (those are the individual components that make up the library) to obtain more complex results. Take a look at the examples in the libnoise documentation and see the [complex planetary surface](#) that looks superb but uses over 100 modules to generate it!

Cinema can combine modules, to a lesser extent, by using a Layer shader and two or more libnoise shaders, but the original library was intended for use by programmers who would combine modules to produce individual, hand-tailored applications to generate the desired result. This isn't possible in a C4D shader. It might be possible if a material node for Redshift could be produced because that would allow any number of modules to be linked. Unfortunately that isn't possible now and may never be.

This shader contains the noise algorithms referred to above and their various parameters can be adjusted as required. However, this is not a complete implementation of libnoise. It doesn't contain the modifier modules which allow the noise to be moved or rotated, nor does it use the combiner modules which combine different noises into one. This isn't really a problem because the C4D Layer shader can do all that and more. The shader does support the turbulence modifier, however. What the shader does not do is to allow multiple modules to be linked to produce the final result, as in the planetary surface shader referred to above. This would require a node-based material to work.

A brief note on Perlin noise

Perlin noise is the basis of Billow and Ridged Multi as well as the basic Perlin noise itself. It might be helpful to review how it works to understand better what the various parameters do. At the end of this manual is an appendix giving more details. Please refer to this if you want to know more about why the parameters produce the results they do.

Using the shader

This is a channel shader so can be added to any channel in a standard material. It can be used in Redshift in a C4D Shader node, but there are limitations, as explained at the end of this manual.

Interface

The shader interface is shown in Figure 1 below.

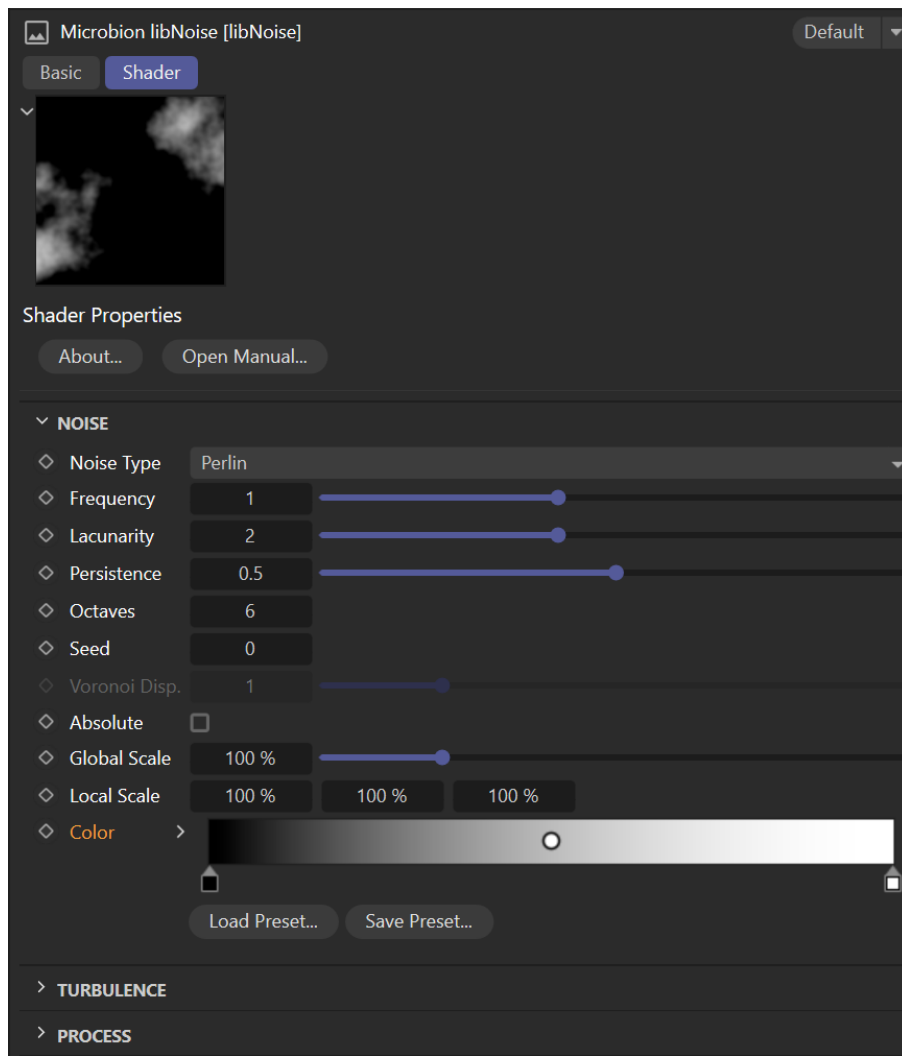


Figure 1. Libnoise user interface

Parameters

Noise Type

This menu allows the chosen noise type to be selected. The options are:

- Perlin - the classic Perlin noise
- Billow - a modified Perlin noise intended to give a more 'billowy' shape for such things as clouds
- Ridged Multi - another modified Perlin noise which gives the appearance of ridges, suitable for terrains
- Voronoi - classic Voronoi noise
- Cylinders - on a flat plane, this will give the appearance of a series of 3D cylinders
- Spheres - should really be called 'rings' because it generates a pattern of concentric rings
- Checkerboard - a standard checkerboard; included in this shader only for the sake of completeness, since there is already a perfectly good checkerboard shader in C4D

Figure 2 shows the different results from each type with all other parameters at their default (except for the colour gradient used).

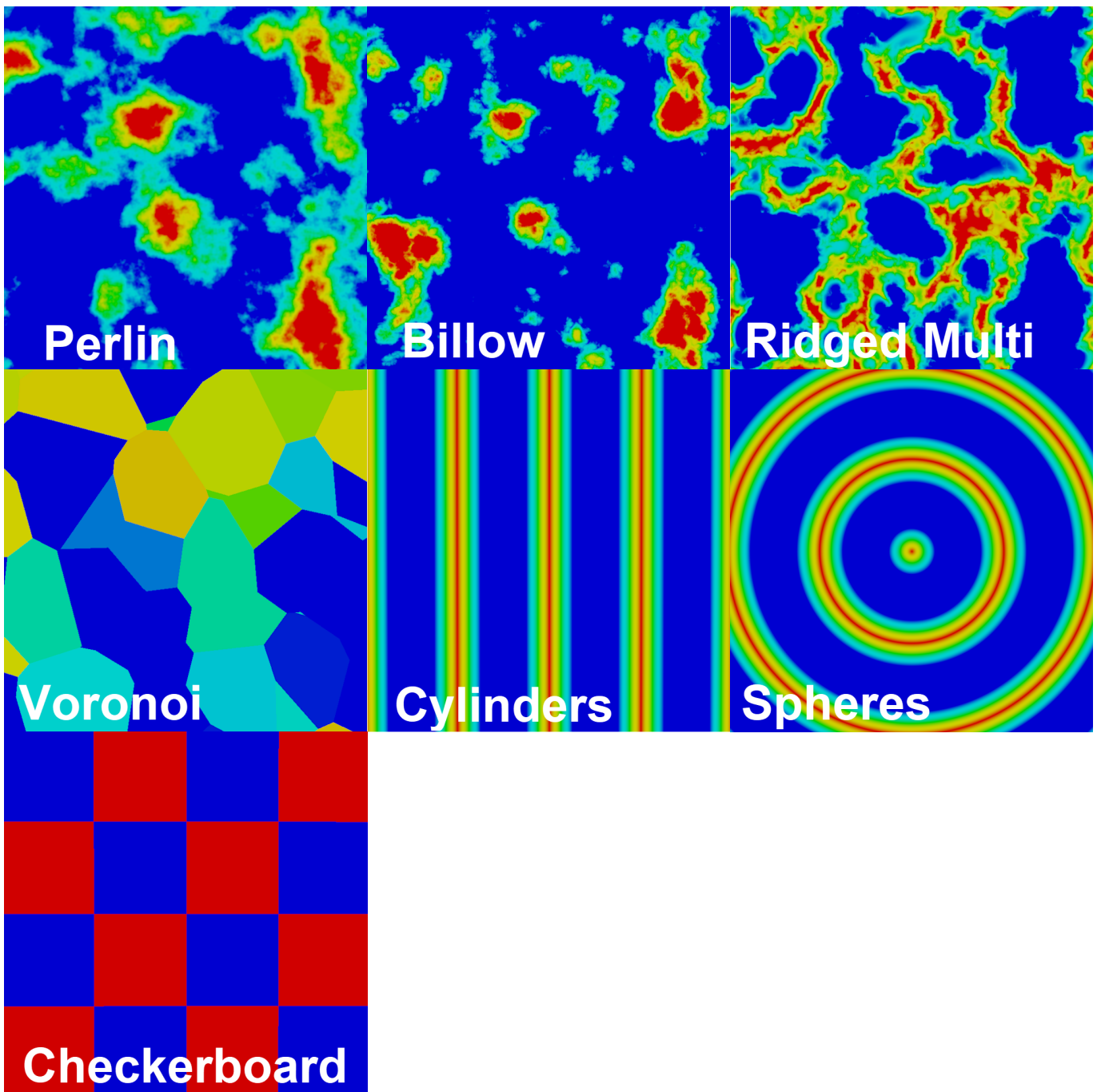


Figure 2. The different noise types

Frequency

This is the frequency used by the first iteration through the noise function (see the appendix for a further explanation). It is actually a scaling factor, and in fact a frequency of 1 and scale of 50% will produce the same result as a frequency of 2 and scale of 100%. However, this is only true in the absence of turbulence. Adding turbulence gives different results even if the scale is adjusted to match the frequency.

The effect of increasing frequency is to increase the amount of detail but what is actually happening is that the size of the generated pattern is reduced, letting you see more of the pattern over the same area.

For Cylinders and Spheres, increasing this value will increase the number of cylinders or spheres generated. For Checkerboard, increasing the frequency increases the number of squares generated.

Lacunarity

In Perlin-based noises, lacunarity changes the fine detail in the output. An increased lacunarity increases fine

detail, but if set too high, too much detail can appear, giving an almost pixelated appearance. Values in the region of 1.5 - 2.5 give the best results.

This setting is not used in the noise types which are not Perlin-based (i.e. Voronoi, Cylinders, Spheres and Checkerboard).

Persistence

Persistence (or 'gain') increases the roughness of the noise. It is only available for Perlin and Billow noises.

Octaves

The number of octaves in the noise. Increasing this value increases the detail in the noise. The minimum is 1, the maximum 30, but for most renders a value of 6 (the default setting) is fine. Only in very large renders would you see any difference between 6 and 30 octaves. Increasing the number of octaves also increases the render time.

The setting is only available for Perlin, Billow and Ridged Multi noises.

Seed

The seed value for the noise. Changing this will result in a different output. It is not used in Cylinders, Spheres or Checkerboard patterns.

Voronoi Disp.

This is a 'displacement' factor used in generating Voronoi noise. In Voronoi noise, each cell has a random value assigned to it. This value is changed by adding the displacement factor. The practical result is that lower values shift the colour selected from the gradient to the left edge, while larger values shift the selected colours to the right edge.

Absolute

The output from the noise generator is usually in the region -1 to 1 (but this is not guaranteed). If this switch is checked, all values are set to their absolute value - that is, positive values are unaffected but negative values are changed to positive. Any change in sign takes place before any other processing such as clipping, and can have a profound effect on the result. Consider this Perlin noise (using the default black-to-white gradient) showing the difference this switch can make:

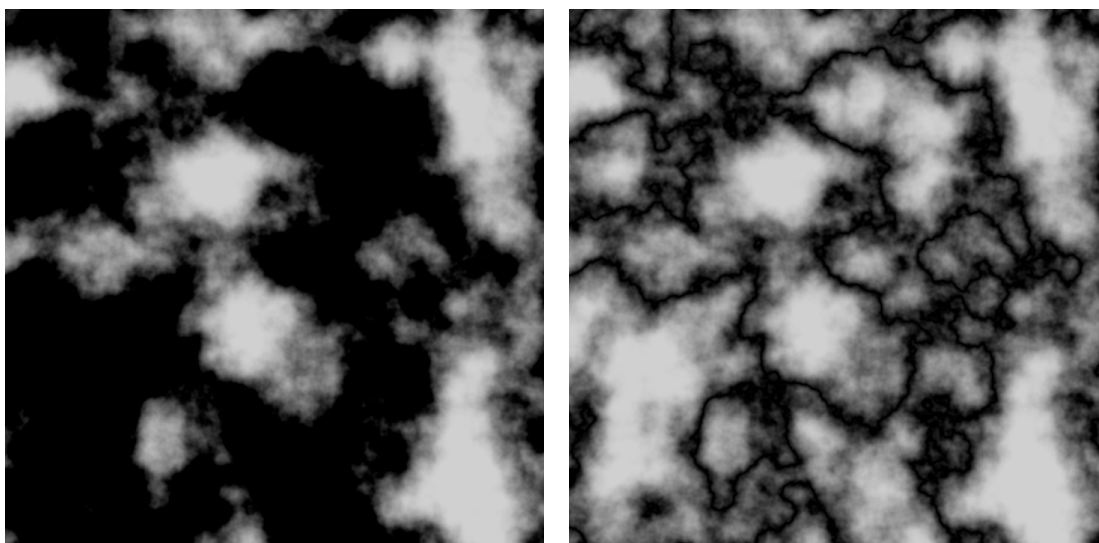


Figure 3. Perlin noise with 'Absolute' set to off (left) and on (right)

Global Scale

A simple scaling factor for the noise.

Local Scale

This works in the same way as the global scale but allows you to adjust the scale along the three axes separately.

Be aware that when rendering the global and local scales are multiplied together, so for example a global scale of 50% and a local scale of 50% will result in an overall scale value of 25%.

Color

The final output from the noise is used to select a colour from this gradient. Any gradient can be used; the default is a black to white gradient, but good results can be obtained with colour gradients.

Turbulence

All the noises in libnoise can be distorted by the use of turbulence, which is generated by three internal Perlin noises, one for each axis. These settings are contained in a separate group to reduce the amount of screen real estate required by the interface. Click the little arrow to the left of the 'Turbulence' heading to reveal these settings.

Turb. Power

This setting determines the strength of the turbulence; when set to zero, there is no effect. This is the result at 0% and 50% power:

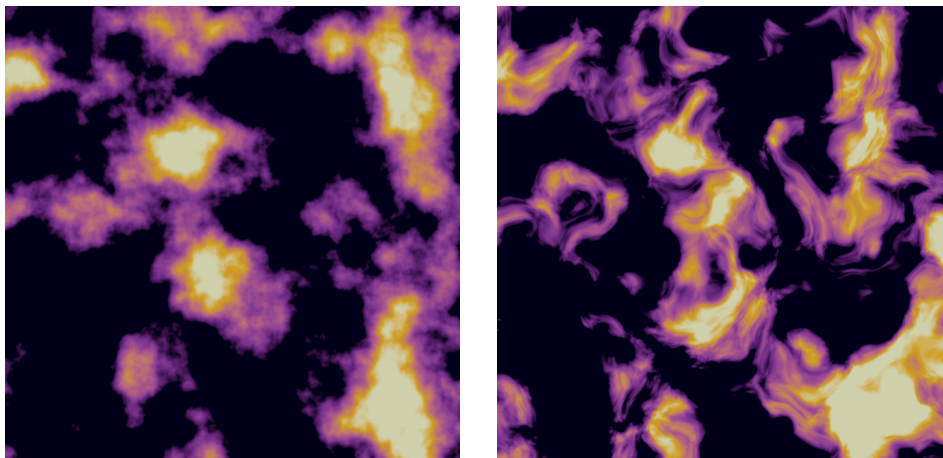


Figure 4. Perlin noise with 0% turbulence power (left) and 50% power (right)

Turb. Frequency

The initial frequency used by the internal Perlin noises. Increasing the value results in more detailed turbulence, as shown in Figure 5 with a setting of 2 (compare to the right-hand image in Figure 4).

Turb. Roughness

This is actually the number of octaves used by the internal Perlin noises. As with the Perlin noise type, the default setting of 6 is fine in most settings; decreasing this value will result in smoother turbulence. Because this is the octave setting for the three Perlin noises, values over about 6 have little visible effect as explained above.

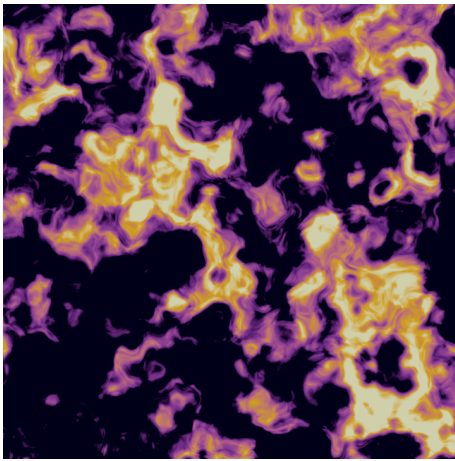


Figure 5. Turb. Frequency set to 2 (compare to Figure 4 above)

Turb. Seed

The seed value for the internal Perlin noises used to generate turbulence.

Process

The output values from the various algorithms will be in different ranges. For example, the Perlin implementation here will generally cause output values between -1 and 1, but this is not guaranteed and values can fall outside these nominal limits. The value is then processed further: it is either clipped or clamped, depending on this menu. The options are used in conjunction with the Low Range and High Range settings.

Click the little arrow to the left of the 'Process' heading to reveal these settings.

Method

This is the method the processing will use. The options are:

- Clip: a value below the Low Range setting will be set to zero. A value above the High Range will be set to 1.
- Clamp: a value below the Low Range setting will be set to the Low Range value. A value above the High Range will be set to the High Range value.

At the defaults of 0 and 100%, neither option will have any effect. The effect of clipping is to set more of the output to 0 or 1 depending on the range values. When used with a colour gradient, more of the colours at the left and right ends of the gradient will be seen. Figure 6 shows some examples using this gradient:



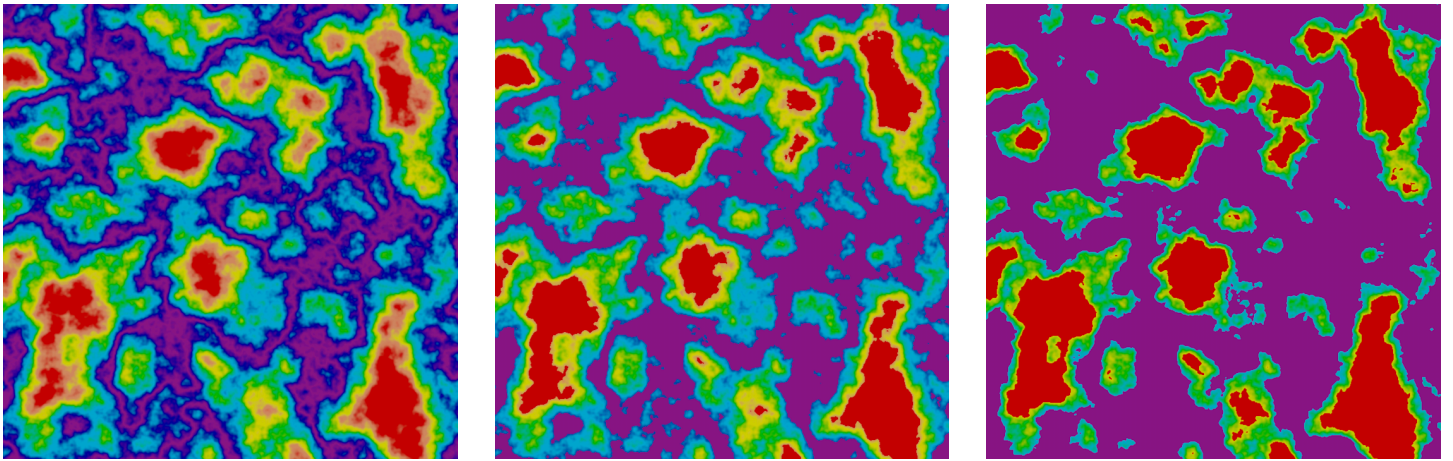


Figure 6. Effect of clipping. On the left, the default 0% and 100% for low and high range; all colours are seen. In the middle, the range values are 20% and 80% - blue and orange colours are lost because more of the noise values are set to 0% or 100%, so more of the colours at the ends of the gradient are shown. On the right, more extreme values of 35% and 65% lose even more of the colour range.

The effect of clamping is to reduce the number of output values at the extremes of 0 and 1. With a colour gradient, more is seen of the colours in the middle of the gradient and less from the ends, as shown here:

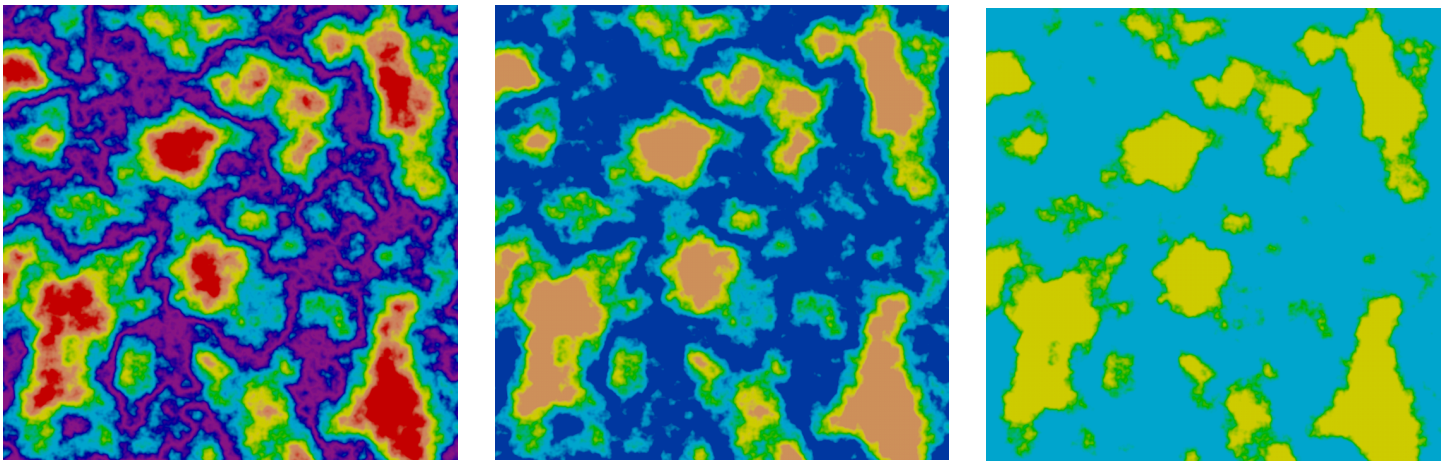


Figure 7. Effect of clamping. This is the opposite to clipping and shows the colours at the end of the gradient being lost. Range values are the same as in Figure 6.

Note: the 'Clip' option will have no effect on the Checkerboard pattern since it always produces output of either 0 or 1. The Clamp and Use Curve options do affect the result, however.

- Use Curve: with this option, the output value is adjusted by remapping it to a spline. The default spline is a linear curve and has no effect on the output. The value from the noise function is along the X-axis of the spline, while the new value is along the Y-axis. This curve produces very little coloured output because most of the noise values - all except those at the very right of the spline - are near zero:

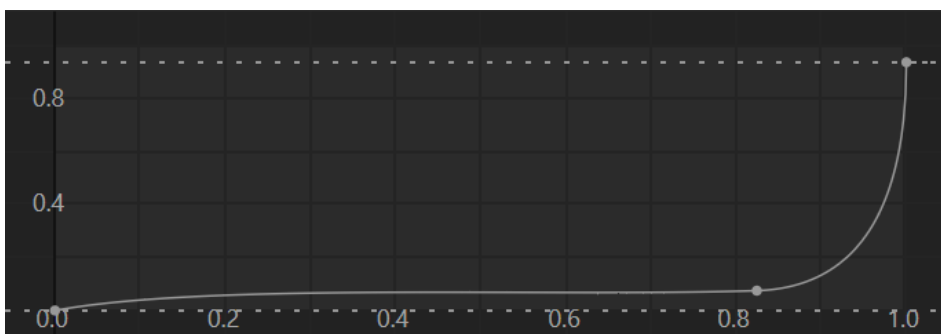
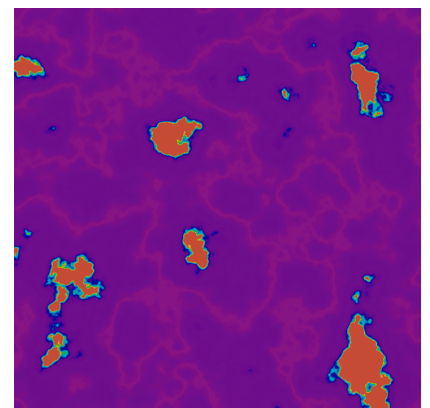


Figure 8. Using a spline curve to change the output noise values



Low Range, High Range

Range values used with clipping or clamping the output (see above).

Curve

The spline curve used when 'Process' is set to 'Use Curve'.

Invert

If checked, this switch will invert the final output, so that (for example) 0 becomes 1 and 1 becomes 0. This won't change the generated noise pattern but it will significantly effect the colours used from the gradient.

Limitations

There are some limitations in the use of this shader. It is not a complete implementation of the original libnoise. For example, it does not include the ability to rotate or offset the noise, nor the 'displace' function in libnoise. For these, you can use a Layer shader and its effects menu.

Libnoise also included the ability to combine noises in various ways, but again, you can do this with a Layer shader with more control and flexibility than libnoise offers.

The biggest limitation is using this shader in Redshift. It works fine in Redshift but only in 2D. That is, on a 3D object such as a sphere, it will not produce a seamless noise. On a Plane object, it works correctly. This is also a limitation of the C4D Noise shader when used in Redshift, presumably one reason why that excellent shader was converted to a native Redshift shader, which does work correctly in 3D.

Conclusion

Libnoise is a fast, high-quality noise library which adds some useful additional noise generation to Cinema 4D. One thing which can readily be done is to add more noise generators to the library, and this shader may be developed further to add some different noises.

The latest version can be downloaded from <https://microbion.co.uk/html/libnoise.htm>. If you have any comments or suggestions, or (hopefully not) bug reports, you can contact me at <https://microbion.co.uk/html/contact.htm>.

Steve Pedler
February 2026

Appendix: the basics of Perlin noise

All the images in this appendix use Perlin noise with the default settings apart from the parameter under discussion. The colour gradient is 'Heat 2' from the presets supplied with C4D. In each case the default value is highlighted in **bold** in the image caption..

Perlin noise is generated by calling an internal algorithm (see https://en.wikipedia.org/wiki/Perlin_noise for more details of this) one or more times. For each iteration (i.e. for each time the algorithm is called), the frequency of the noise is increased but the amplitude - that is, the influence of that iteration on the final result - is decreased. The results of each iteration are then added together to produce the final result. The 'octaves' parameter is simply the number of times the internal algorithm is called, so the more octaves there are, the more detailed the noise will be. However, the amplitude (that is, the influence on the final result) of each octave is reduced each time the algorithm is called, so very high octave counts produce little or no visible effect unless the render is very large (10,000 x 10,000 at least). These images show the same noise with octaves of 1, 2, and 6:

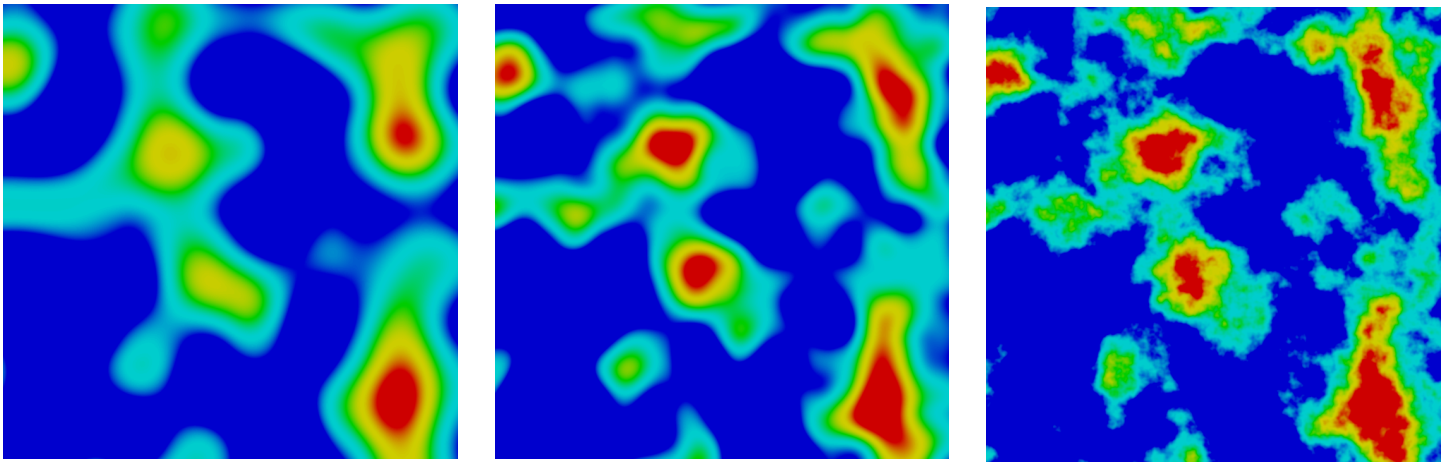


Figure 9. Perlin noise with octaves of 1, 2 and 6

The ‘frequency’ of the parameter is a value which multiplies the input coordinates of the point being sampled before calling the algorithm. You can see that in effect this is a scaling factor; if you render Perlin noise with two different frequencies, the original pattern is preserved but it will be smaller if the frequency is increased and larger if it is reduced. The effect is to increase the detail in the final noise. These images show the noise with a frequency of 0.5, 1, and 2; you can see that in each case the original pattern is still present, but is smaller at higher frequencies, which lets you see more of the noise on any given surface. The same area is highlighted on each image so you can see how the noise is the same but you can see more of it:

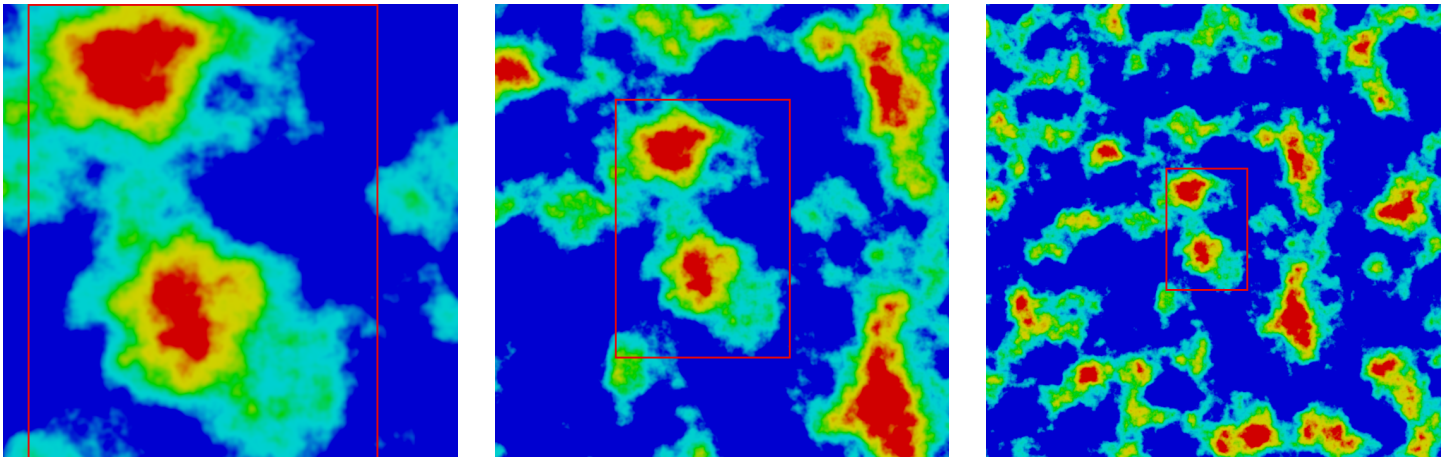


Figure 10. Perlin noise with frequency set to 0.5, 1, and 2

The frequency parameter in the shader is the frequency for the first octave, and is increased for each successive iteration. But by how much is the frequency increased? This is what the ‘lacunarity’ setting does. Increasing this will increase the frequency change for each octave (iteration). Therefore, increasing lacunarity will increase the detail in the final result. These images show the lacunarity with settings of 1, 2 and 3:

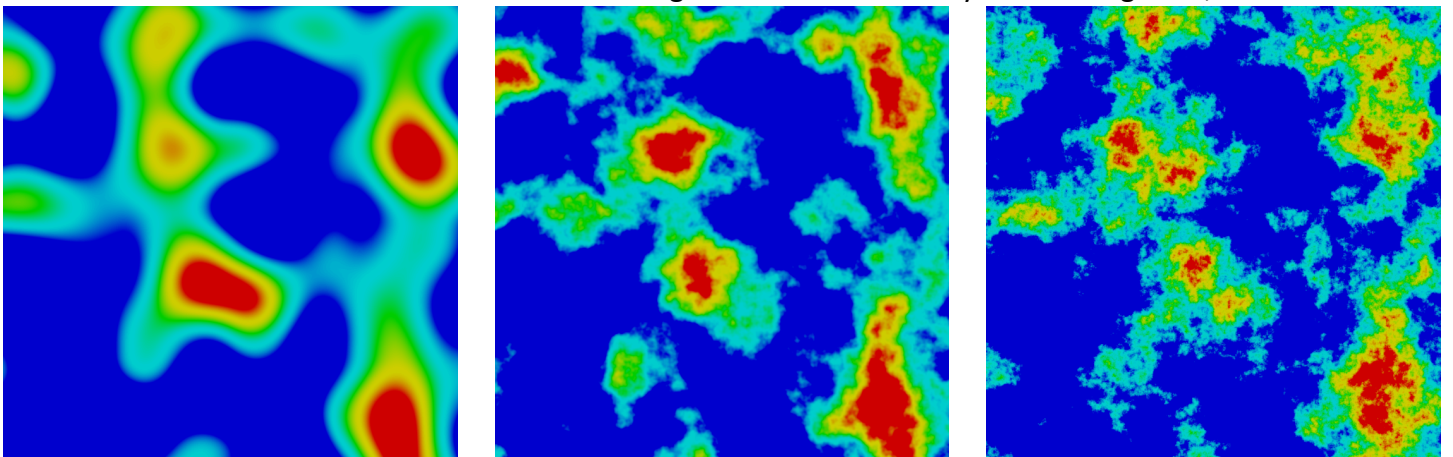


Figure 11. Perlin noise with lacunarity of 1, 2 and 3

It was mentioned above that the amplitude of each iteration is reduced for each iteration. This is done by multiplying the amplitude with the 'persistence' setting. The amount the amplitude is reduced by therefore depends on the persistence value, which may be termed 'gain' in some implementations. You can see that if the persistence setting is zero, the amplitude of each iteration after the first will be zero, so this will give the same effect as if the octave setting was 1. To see any change in the noise due to a change in persistence, you must have at least two octaves. The effect of increasing persistence is then to increase the roughness of the noise. These images show persistence at 0.4, 0.5 and 0.6:

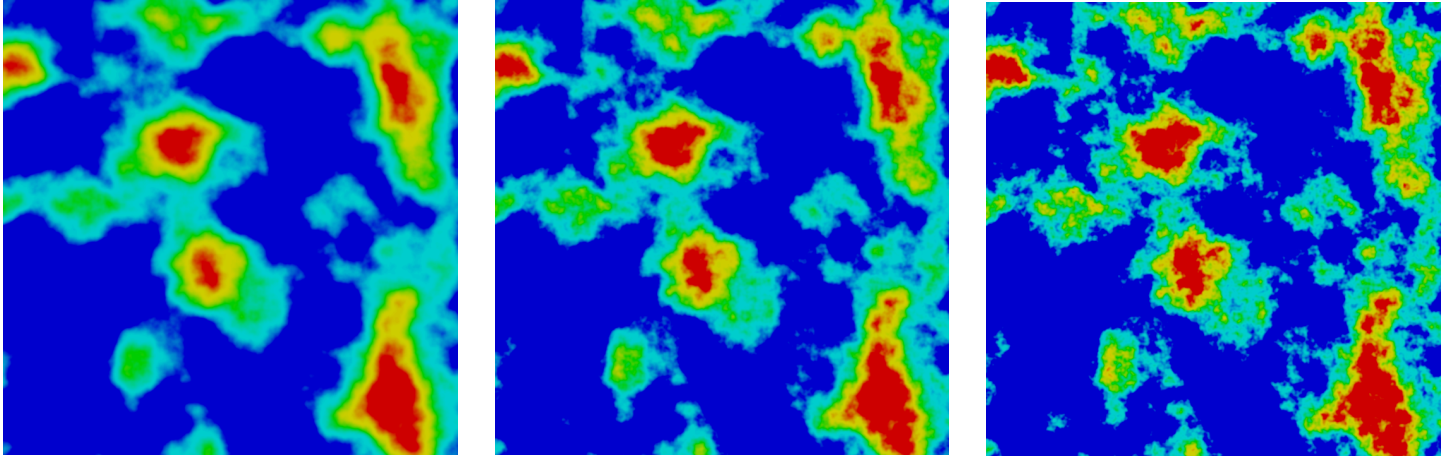


Figure 12. Perlin noise with persistence of 0.4, **0.5** and 0.6

Finally, the noise is generated from an initial value - the seed value. Altering the seed will cause a very different noise pattern to appear.

Acknowledgements

Libnoise was written by Jason Bevins who retains the copyright to the code. Thanks and acknowledgements are due to him for making the library and source code available. Libnoise in its original form is still available and the website can be found on Sourceforge at <https://libnoise.sourceforge.net/index.html>.