Using Perlin noise in sound synthesis

Artem POPOV
Gorno-Altaysk,
Russian Federation,
art@artfwo.net

Abstract

Perlin noise is a well known algorithm in computer graphics and one of the first algorithms for generating procedural textures. It has been very widely used in movies, games, demos, and landscape generators, but despite its popularity it has been seldom used for creative purposes in the fields outside computer graphics. This paper discusses using Perlin noise and fractional Brownian motion for sound synthesis applications.

Keywords

Perlin noise, Simplex noise, fractional Brownian motion, sound synthesis

1 Introduction

Perlin noise, first described by Ken Perlin in his ACM SIGGRAPH Computer Graphics article "An image Synthesizer" [Perlin, 1985] has been traditionally used for many applications in computer graphics. The two-dimensional version of Perlin noise is still widely used to generate textures resembling clouds, wood, and marble as well as procedural height maps.

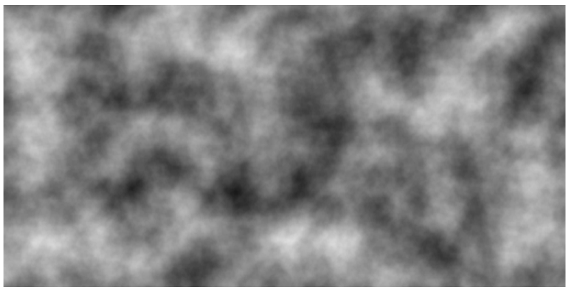


Figure 1: 2D Perlin noise as rendered by Gimp plugin "Solid noise"

Despite its popularity, Perlin noise has been seldom used for creative purposes in the fields outside the world of computer graphics. For music applications, Perlin noise has been occasionally used for creating stochastic melodies or as a modulation source.

This paper is focused on synthesizing single-cycle waveforms with Perlin noise and its successor, Simplex noise. An overview of both algorithms is given followed by a description of fractional Brownian motion and several techniques for adding variations to noise-based waveforms. Finally, the paper describes an implementation of a synthesizer plugin using Perlin noise to create musically useful timbres.

2 Perlin noise

Perlin noise is a *gradient noise* that is built from a set of pseudo-random gradient vectors of unit length evenly distributed in N-dimensional space. Noise value in a given point is calculated by computing the dot products of the surrounding vectors with corresponding distance vectors to the given point and interpolating between them using a smoothing function.

Sound is a one-dimensional signal, and for the purpose of sound synthesis Perlin noise of higher dimensions is not so interesting. While it is possible to scan Perlin noise in 2D or 3D space to get a 1-dimensional waveform, it's necessary to make sure the waveform can be seamlessly looped to produce a musically useful timbre with zero DC offset.

For one-dimensional Perlin noise, the noise value is interpolated between two values, namely the values that would have been the result if the closest linear slopes from the left and from the right had been extrapolated to the point in question [Gustavson, 2005]. Thus, the noise value will always be equal to zero on integer boundaries. By sampling the resulting 1-dimensional noise function, it's possible to generate a waveform that can be looped to produce a pitched tone (Figure 2).

3 Simplex noise

Simplex noise is an improvement to the original Perlin noise algorithm proposed by Ken Perlin

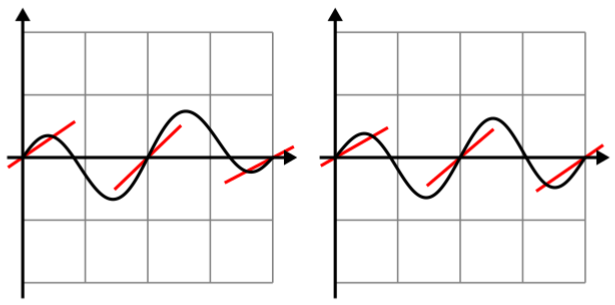


Figure 2: Perlin noise (left) and Simplex noise (right) with the gradients used for interpolation

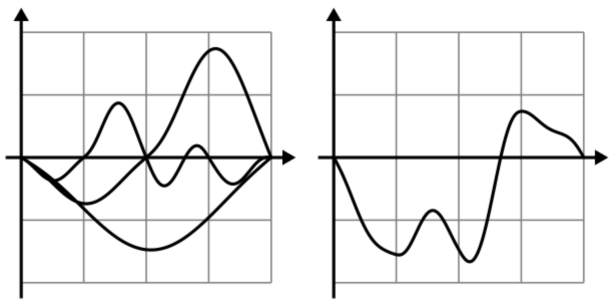


Figure 3: 3 octaves of Perlin noise (left) summed to generate a fBm waveform (right)

himself [Perlin, 2001]. The advantages of simplex noise over Perlin noise include lower computational complexity, no noticeable directional artifacts, and a well-defined analytical derivative.

Simplex noise is created by splitting an N-dimensional space into simplest shapes called simplices. The value of the noise function is a sum of contributions from each corner of the simplex surrounding a given point [Gustavson, 2005].

In one-dimensional space, simplex noise uses intervals of equal length as the simplices. For a point in an interval, the contribution of each surrounding vertex is determined using the equation:

$$(1 - d^2)^4 \cdot (g \cdot d) \tag{1}$$

Where g is the value of the gradient in a given vertex and d is the distance of the point to the vertex.

Both Perlin noise and Simplex noise produce very similar results (Fig. 2) and are basically interchangeable in a sound synthesizer[1]. For brevity, Perlin noise or noise will be used to refer to both algorithms for the scope of this paper, since Simplex noise is also invented by Ken Perlin.

4 Fractional Brownian motion

Fractional Brownian motion (fBm), also called fractal Brownian motion is a technique often used with Perlin noise to add complexity and detail to the generated textures.

Fractional Brownian motion is created by summing several iterations of noise (*octaves*), while successively incrementing their frequencies in regular steps by a factor called *lacunarity* and decreasing the amplitude of the octaves by a factor called *persistence* with each step [Vivo and Lowe, 2015].

$$fBm(x) = \sum_{i=0}^{n} p^i \cdot noise(2^i \cdot x) \tag{2}$$

Lacunarity can have any value greater than 1, but non-integral lacunarity values will result in non-zero fBm values on the integer boundaries. To keep the waveform seamless in a sound synthesizer, lacunarity has be an integer number. A reasonable choice for lacunarity is 2, since bigger values result in a very quick buildup of the upper harmonics (Eq. 2).

Fractional Brownian motion is often called Perlin noise, actually being a fractal sum of several octaves of noise. While typically the same noise function is used for every octave, different noise algorithms can be combined in the same fashion to create *multifractal* or *heterogeneous* fBm waveforms [Musgrave, 2002].

5 Waveform modifiers

5.1 Gradient rotation

One technique traditionally used to animate Perlin noise is gradient rotation [Perlin and Neyret, 2001]. When gradient vectors in 2- or more dimensional space are rotated the noise is varied while retaining its character and detail. This technique has been used for simulating advected flow and other effects. A similar technique can be applied to 1-dimensional noise to introduce subtle changes to the sound.

Rotating gradients is a computationally expensive operation and cannot be used with 1-dimensional noise, since the noise is built from linear gradients instead of directional vectors.

[1]In some cases Perlin noise adds additional low frequency harmonics to the sound which may or may not be desirable.

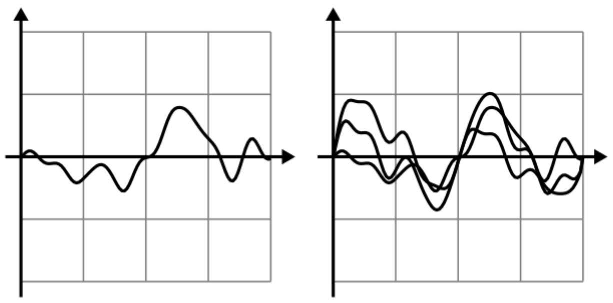


Figure 4: Gradient offsets applied to fBm (left) modify the waveform (right) while preserving the timbre

It is still possible to apply this technique to 1-dimensional noise by adding a variable offset value to the gradients and symmetrically wrapping it when the maximum allowed gradient value (1) is reached.

$$g' = \begin{cases} 2 - g, & g > 1 \\ -2 - g, & g < -1 \end{cases} \tag{3}$$

In a sound synthesizer, gradient rotation does not change the timbre significantly. It does alter the amplitudes of the upper harmonics slightly (Fig. 4), adding variations that can be used in a polyphonic (poly-oscillator) synthesizer.

5.2 Domain warping

Another classic technique for adding variation to Perlin noise is called domain warping. Warping simply means that the noise domain is distorted with another function $g(p)$ before the noise function is evaluated.

Basically, $noise(p)$ is replaced with $noise(g(p))$. While g can be any function, it's often desirable to distort the image of *noise* just a little bit with respect to its regular behavior.

Then, it makes sense to have $g(p)$ being just the identity plus a small arbitrary distortion $h(p)$ [Quílez, 2002]. In the most basic case the distortion can be the noise itself (Eq. 4).

$$f(p) = noise(p + noise(p)) \tag{4}$$

For the purpose of sound synthesis it is better to expose warping as an adjustable parameter. Warping modulation can be implemented by adding a coefficient that is used to control the warping depth (Eq. 5).

$$f(p) = noise(p + noise(p) \cdot w) \tag{5}$$

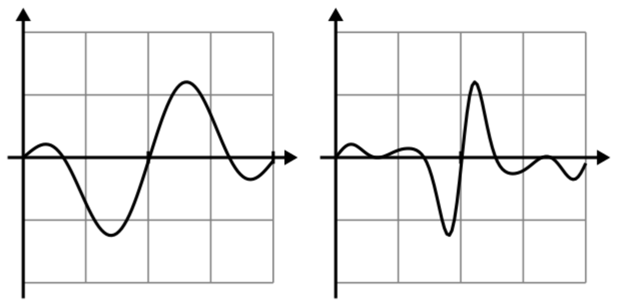
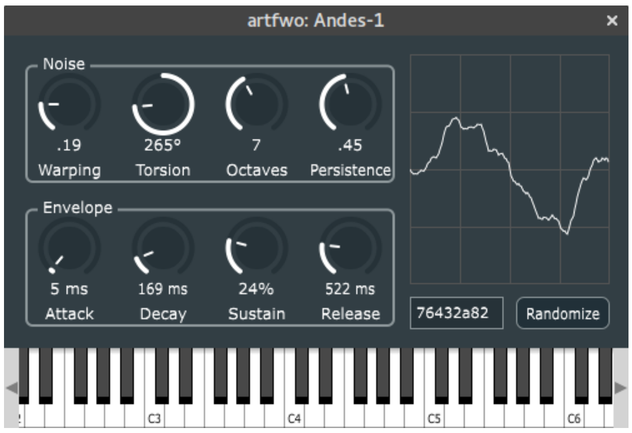


Figure 5: Simplex noise (left) with domain warping (right)



artfwo: Andes-1

Noise
.19 Warping
265° Torsion
7 Octaves
.45 Persistence

Envelope
5 ms Attack
169 ms Decay
24% Sustain
522 ms Release

76432a82

Figure 6: Andes, a JUCE-based synthesizer using Perlin noise

Since the domain of noise is distorted with the noise itself, the symmetry of the waveform will remain generally the same as seen on Fig. 5.

6 Implementation

The presented ideas have been implemented as a basic synthesizer plugin called *Andes* (Figure 6). The plugin has been developed using the JUCE[2] framework and is currently available in the form of VST, AU, and standalone program for Windows, MacOS, and Linux[3].

At the time of writing, Andes supports gradient rotation, basic warping, up to 16 octaves of noise, and adjustable persistence, which allows a usable range of unique sounds to be produced. The sound of noise is susceptible to aliasing at higher frequencies, but oversampling has not been implemented so far.

The resulting sounds resemble early digital synthesizers, but also have a unique character to them and can be described as "distinctively digital".

[2]`https://juce.com`
[3]`https://artfwo.github.io/andes/`

6.1 Predictable randomness

A synthesizer plugin cannot have completely randomly sounding timbres when the synthesizer is used in certain contexts such as a multi-track DAW project. The amplitude and timbre of the synthesizer cannot change in unpredictable ways to make sure the track won't break the mix.

Predictable randomness in Andes is achieved by saving the random seed for generating gradients in the plugin state (preset). The 32-bit Mersenne Twister 19937 generator from C++ standard library is used explicitly to make sure the random numbers generated from the same seed will stay the same across different architectures and platforms.

The set of gradients covering the entire allowed range of octaves is created and stored in memory every time the plugin is instantiated or when a new seed is created using the plugin UI.

The additional advantage of using precomputed set of gradients is that computationally expensive random number generation is moved out of the audio processing code.

6.2 Output level normalization

A big issue with Perlin noise is normalizing the output level to fixed values. This issue is currently not resolved in Andes, but a possible direction to explore is early computing of peak values during the stage of generating gradients.

6.3 Waveform symmetry

The symmetry of waveforms is another thing to consider when developing a noise-based synthesizer.

Current Andes implementation uses completely random gradients. The first noise octave is built from 3 gradients (at points 0, 1, and 2). Sometimes, this results in cusps and unwanted distortion when the both outermost gradients are either positive or negative. Alternating signs for even and odd gradients in the gradient table can further improve the synthesizer usability.

Setting signs for even and odd gradients explicitly can also help reduce the domain range for the noise function.

7 Conclusions

Perlin noise, Fractional Brownian motion and multifractal synthesis are interesting directions to explore for sound applications. Although Perlin noise can be used to make sounds, the approach still remains to be improved. Noise level normalization is one of the biggest issues yet to be resolved.

The general idea of using unconventional, i.e. graphics algorithms in sound and music presents a lot of challenges, but also opens many different possibilities in both the technical and aesthetic aspects.

8 Acknowledgments

The author would like to thank Maria Pankova for helping the idea of making a noise-based synthesizer to emerge and for assistance with maths in the early stages of Andes development. Thanks also goes to Alexey Durachenko for suggesting useful optimizations to the Simplex noise implementation.

References

Stefan Gustavson. 2005. Simplex noise demystified. `http://staffwww.itn.liu.se/~stegu/simplexnoise/simplexnoise.pdf`.

F. Kenton Musgrave. 2002. Procedural fractal terrains. In *Texturing and Modeling: A Procedural Approach*, chapter 9.

Ken Perlin and Fabrice Neyret. 2001. Flow noise. In *Siggraph Technical Sketches and Applications*, page 187, Aug.

Ken Perlin. 1985. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, July.

Ken Perlin. 2001. Noise hardware. In M. Olano, editor, *Real-Time Shading ACM-SIGGRAPH Course Notes*, chapter 2.

Íñigo Quílez. 2002. Domain warping. `http://www.iquilezles.org/www/articles/warp/warp.htm`.

Patricio Gonzalez Vivo and Jen Lowe. 2015. Fractal brownian motion. The Book of Shaders, `https://thebookofshaders.com/13/`.