# Setup-free Procedural Surface Noise

**Xiyao LI**

## Abstract

In computer graphics texturing, the most widely used method is to map a 2D image around a 3D object and define how light would affect it. However, texture mapping always requires texture parameterization and can introduce distortion and discontinuity. Here I present a procedural noise texturing without using texture coordinates. It is setup-free, which applies to the image and the surface. In two dimensions, the noise is generated by sparse convolution filtering with the Gabor kernel. And this method is also applicable in three-dimensional space with a 3D Poisson disk sampling. This noise offers accurate spectrum control with intuitive parameters, and it is robust to surface evolution and even topology change. It is relatively low-cost in computation time and gives a satisfying result. This approach is a good trade-off between the visual effect and the computation cost.
**Keywords:** texture, procedural noise, setup free

## 1 Introduction

Most existing texture techniques rely on texture mapping, which maps a 2D image onto the object's surface. Such texture methods always require a texture parameterization. However, texture mapping also introduces some distortion and discontinuity problems. One main problem with texture mapping is aliasing, which occurs when the texture has a lot of colour variation and a high resolution (lots of high-frequency information), and the amount of the pixel in the rendering target object is low (sampling frequency too low). To avoid aliasing, an oversampling is a possible but high-cost solution.

Here I present a procedural noise texture method without using texture coordinates. Noise has a wide range of applications in computer graphics. In particular, the Perlin Noise is a robust algorithm to generate random content. It is widely used in games and the special effects industry. Here I will use another procedural noise. In this implementation, the noise texture is based on sparse convolution and the Gabor kernel, which has the following properties:

- The noise is mapped to the surface without the need of texture coordinates and provides satisfying results with low-cost computation.
- The noise offers accurate spectrum control with intuitive parameters such as orientation, principal frequency and bandwidth.
- The noise is setup-free. It is robust to surface evolution and even topology change.

In the following sections, I will firstly present the 2D band-limited noise implementation from Ares Lagae, Sylvain Lefebvre, George Drettakis and Philip Dutré's paper [1] and the theories on which the algorithm depends. Then, I introduce the Sparse Gabor Convolution Noise implementation used as a surface noise on 3D objects.

## 2 Previous Work

This implementation is mainly based on the Procedural Noise using Sparse Gabor Convolution. So in this section, I present the basic notions and fundamental theories used in this paper.

### 2.1 Sparse Convolution Noise

The sparse convolution algorithm was proposed by Lewis in 1989[2]. Sparse convolution involves building a noise function by convolving a filter function with a set of randomly located impulses, a Poisson process. The scattered random impulses are a sparse form of white noise, hence the term *sparse convolution*. After filtering the white noise with

a low-pass filter function, the sparse convolution noise's power spectrum is derived from the power spectrum of the filter kernel, so modifying the filter lead to controlling the noise spectrum.

In this algorithm, a three-dimensional noise is synthesised by the convolution of a three-dimensional kernel $h(\rho)$ with a Poisson noise process $\gamma$:

$$\eta(\rho) = \int_{R^3} \gamma(\sigma) h(\rho - \sigma) d\sigma$$

where $\gamma$ is the Poisson process consists of impulses of uncorrelated intensity distributed at uncorrelated locations in space:

$$\gamma(\rho) = \sum a_k \delta(\rho - \rho_k)$$

Then a radially symmetric smooth cosine kernel with $|r| < 1$ is applied:

$$h(r) = \frac{1}{2} + \frac{1}{2}\cos(\pi r) \tag{1}$$

In the 2D implementation, sparse convolution noise evaluates by introducing a grid and generating the positions and weights of the kernels in each cell on the fly. The grid reduces the noise evaluation to the grid cells close to the point of evaluation. An example of sparse convolution noise is shown in this paper [1].

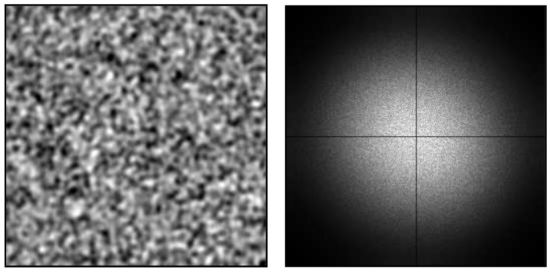


Figure 1: Isotropic noises and power spectra of sparse convolution

## 2.2 The Random Pulse Process

Sparse convolution noise is a random pulse process using the symmetric smooth cosine kernel. The random pulse process Y is the sum of randomly weighted and positioned pulses.

$$Y(x) = \sum_i w_i h(x - x_i) \tag{2}$$

where $h$ is the kernel, $x_i$ are the positions near $x$ sampled by a Poisson distribution with mean $\lambda$ called the impulse density, and the weights $w_i$ are realisations of a random variable $W$.

## 2.3 The Gabor Filter

To control the power spectrum of the random pulse process, it is essential to choose an appropriate pulse. The idea is to set up the pulse and using compact support in the spatial domain to enable efficient procedural evaluation. Therefore, a Gabor kernel is applied to filter the random pulse.

In image processing, the Gabor filter is a linear filter for texture analysis. It analyses whether an image has specific frequency content in a particular direction.It is particularly well suited to texture representation and discrimination.

In the spatial domain, a 2D Gabor filter is a Gaussian kernel function modulated by a sinusoidal plane wave. Its impulse response is defined by a sinusoidal wave multiplied by a Gaussian function. The filter has a real and an imaginary component. These two components can form a complex or be used separately. The Gabor filter can be expressed as follows:

$$g(x, y) = Ke^{-\pi a^2 (x^2 + y^2)} \cos[2\pi F_0 (x \cos\omega_0 + y \sin\omega_0)] \tag{3}$$

where $K$ is the magnitude, $a$ is the width of the Gaussian envelope, $(F_0, \omega_0)$ are the magnitude and orientation of the frequency.

The parameters of the Gabor kernel have an intuitive meaning. The width $a$ corresponds to the width of the Gaussian in the frequency domain. The frequency $(F_0, \omega_0)$ corresponds to the location of the Gaussian envelope. $(F_0, \omega_0)$ can therefore be interpreted as the principal frequency and the width $a$ as the bandwidth, the range of frequencies around the principal frequency. To make the parameters' meaning more intuitive, the following figures show the influence of changing only one parameter at one time.

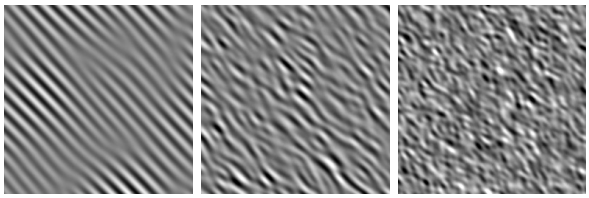


Figure 2: a = 0.01, 0.05 and 0.1

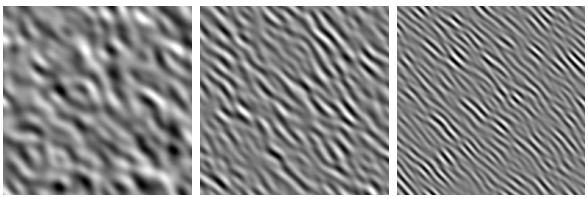


Figure 3: $F_0 = 0.0325$, 0.0625 and 0.1

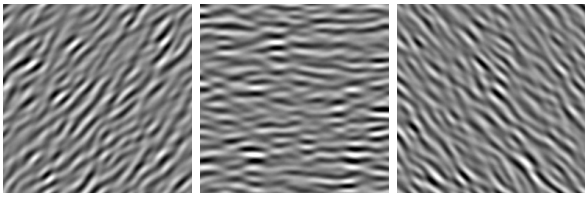


Figure 4: $\omega_0 = -\pi/4$, $\pi/2$ and $\pi/4$

### 2.4 Band-Limited Isotropic Noise

Now by combining the random pulse process with the Gabor filter as a pulse, the band-limited isotropic noise N can be expressed as:

$$N(x, y) = \sum_i w_i g(x - x_i, y - y_i) \qquad (4)$$

The rendering is shown in Figures 2, 4 and 5. 2D sparse convolution noise provides pleasant procedural effects with high-speed computation.

## 3 Setup-free Surface Noise

As the two-dimensional sparse convolution noise has good performance, it will be good to apply it to the 3D models.

In the previous section, the noise is computed by pixel on the image. And here, the aim is to generate noise on the surface with the same method. The idea is quite similar. Instead of working on a pixel, I compute the noise at a point on a surface $\mathbf{p}$ with the surface normal $\mathbf{n}$ by projecting a three-dimensional Poisson distribution on the plane determined by $\mathbf{p}$ and $\mathbf{n}$. Now, the method gets back to the evaluation of a two-dimensional noise in that tangent plane. Then, isotropic surface noise is obtained by filtering the projected random points on the plane with a Gabor kernel.

In the following, I will present how to generate a 3D Poisson disk at some length and the filters I used to get isotropic and anisotropic noise.

### 3.1 Poisson Disk Sampling

Poisson disk sampling is characterised by the fact that there are no two points too close. In the figure below, the points are respectively generated randomly and by Poisson disk sampling. The method here for sampling is Bridson's Fast Poisson Disk Sampling in Arbitrary Dimensions [4].

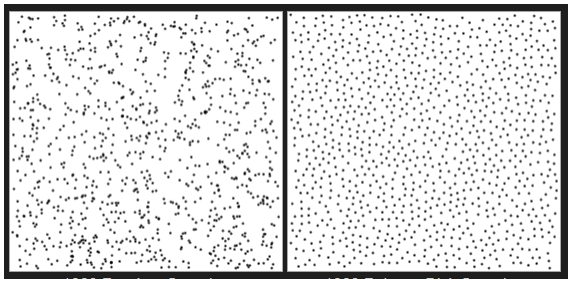


Figure 5: left: random sampling, right: Poisson disk sampling

The Poisson sampling is computed in a three-dimensional grid. The algorithm takes the features of the sample domain in 3D, the minimum distance $r$ between samples, and a constant $k$ as the limit of samples to choose before rejection in the algorithm (typically $k = 30$).

1. Initialise a three-dimensional background grid for storing samples and accelerating spatial searches. The cell size to be bounded is defined by $r/\sqrt{3}$, so that each grid cell will contain at most one sample.

2. Select the initial sample $x_0$, randomly chosen uniformly from the domain. Insert it into the background grid, and initialise the *active list*, an array of 3D samples.

3. While the *active list* is not empty, choose a random sample $x_i$ from it. Generate up to $k$ points chosen uniformly from the spherical annulus between radius $r$ and $2r$ around $x_i$. For each point in the turn, check if it is within distance $r$ of existing samples (using the background grid to only test nearby samples). If a point is adequately far from existing samples, save its position to the corresponding cell in the grid and add it to the *active list*. After $k$ attempts, if no such point is found, remove this sample from the *active list*.

Figure 6. shows points sampled in a cube with length, width and height of 6, 6 and 6, and the minimum radius between samples is 1. Much more

points are sampled in the real implementation, here is just an example to visualise this sampling. For more implementation details, the pseudo-code can be found below.

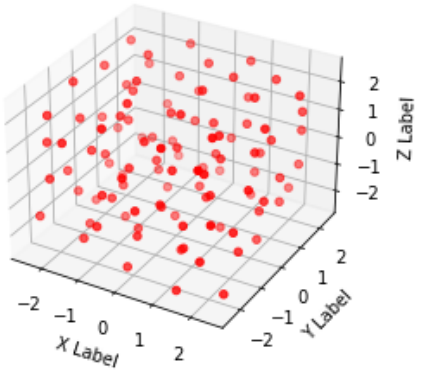



Figure 6: Poisson disk samples

**Algorithm 1** Fast Poisson Disk Sampling

```
if ActiveList is empty then
    p ← a random point in the cube
    Add p to ActiveList
    Save p in the grid
end if
while ActiveList is not empty do
    Choose x_i randomly from ActiveList
    found ← false
    for i < k do
        c ← from x_i with distance ∈ [r, 2r]
        if c is FAR ENOUGH from others then
            found ← true
            Add c to ActiveList
            Save c in the grid
        end if
    end for
    if not found then
        Remove x_i from ActiveList
    end if
end while
```

## 3.2 Isotropic Filter

The Poisson samplers are generated on the fly using a three-dimensional grid. The points are evenly distributed in a cube that encases the 3D object. After finishing the generation of samples, they are sent to the shader to compute the noise for each fragment. By projecting only the points near the tangent plane, I get random points with random weights for the equation4. The weight is defined as inversely proportional to its distance to the plane.

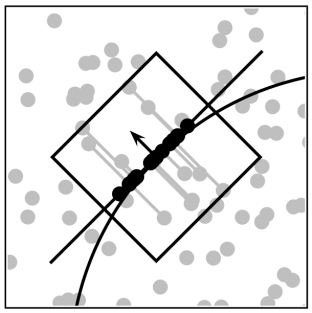


Figure 7: Project Poisson disk on the tangent plane

For one fragment, its tangent plane is defined by its position **p** and its normal **n**. The noise of this fragment is computed by Eq.4 with $x_i$ the projected points in this plane.

Then by applying a smooth cosine filter1, the isotropic surface noise is shown below:

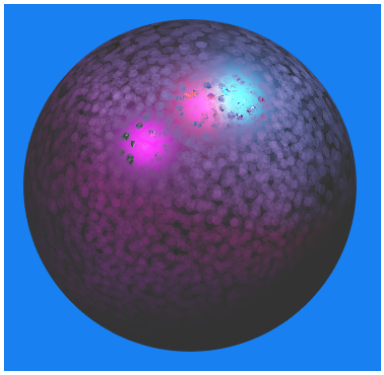


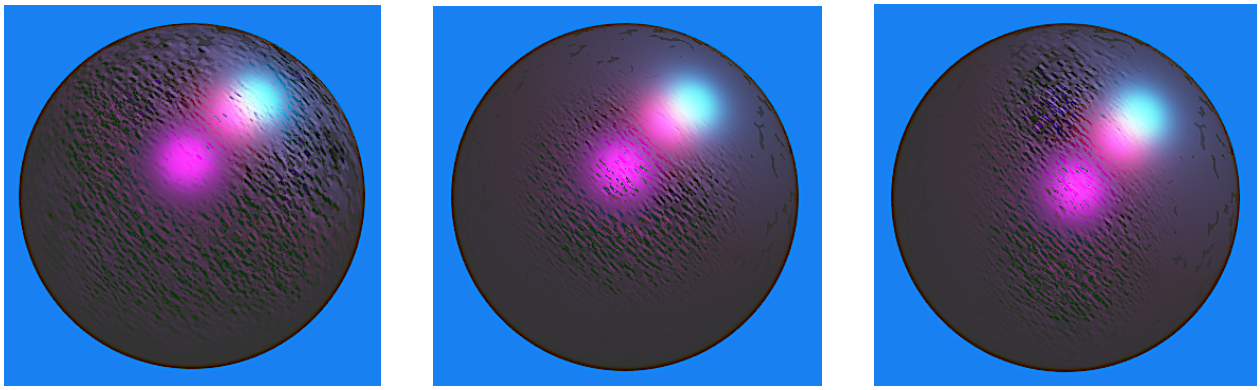
Figure 8: Smooth Cosine Kernel

And when applying a Gabor filter3, it gives an isotropic surface noise following one direction. The result is shown in the next section.

## 3.3 Anisotropic Filtering

The anisotropic surface noise relies on a vector field that guides the anisotropy procedurally by computing a local surface frame from the surface normal and a global direction. The detailed calculations can be found in section 5.2. of this paper[1].

There are several parameters of the anisotropically filtered Gabor kernel $K'$, $a'$, $F_0'$ and $w_0'$. $J$ is the Jacobian of the mapping from image to texture coordinates, and $\Sigma$ is the standard deviation of the Gaussian. The Gaussian Envelop is:



(a) Isotropic Gabor Kernel (b) Anisotropic filtering (c) Anisotropic Gaussian Kernel

Figure 9: Surface Noise with different filtering

$$N(f;\mu,\Sigma) = e^{-\frac{1}{2}(f-\mu)\Sigma^{-1}(f-\mu)^T} \tag{5}$$

The standard deviation in the frequency domain is:

$$\Sigma_F^{-1} = 4\pi^2\sigma^2(JJ^T) \tag{6}$$

and the standard deviation in the unfiltered Gabor kernel is:

$$\Sigma_G^{-1} = \frac{2\pi}{a^2}I \tag{7}$$

So the standard deviation in the anisotropically filtered Gabor kernel in the frequency domain can be expressed as:

$$\Sigma_{FG} = (\Sigma_F^{-1} + \Sigma_G^{-1})^{-1} \tag{8}$$

Then the parameters of the anisotropically filtered Gabor kernel are expressed as:

$$F_0' = F_0\cos(\omega_0) \tag{9}$$
$$\omega_0' = F_0\sin(\omega_0) \tag{10}$$
$$a' = 2\pi\sqrt{|\Sigma_{FG}|} \tag{11}$$
$$K' = K\frac{a'^2}{a^2}N(0;\mu_G,\Sigma_F+\Sigma_G) \tag{12}$$

It is also possible to avoid this calculation by using Gabor kernels with an anisotropic Gaussian envelope with the expression below. The results will be discussed in the next section.

$$G(x,y,\sigma_x,\sigma_y) = \frac{1}{2\pi\sigma_x\sigma_y}\exp[-(\frac{x^2}{2\sigma_x^2}+\frac{y^2}{2\sigma_y^2})] \tag{13}$$

# 4 Results and Discussion

This surface noise with isotropic and anisotropic filtering is shown in Figure 9. A procedural noise texture is created without using texture parametrization, and the computation speed is high enough. I have precise spectrum control on the noise. And the noise's features flow along the curvatures and naturally adapt to topology changes.

As anisotropic filtering is a compliment in this implementation, I did not have enough time to develop it more, so I chose a constant Jacobian $J$ for each fragment. As a result, the filter controls only the texture range on the 3D model but not the noise direction. If I had had time to create a vector field to guide the filter, the results would be more interesting.

However, one disadvantage of this implementation is that this surface noise is always small compared to the object's geometric detail. Moreover, a discontinuity in the surface normal results in a discontinuity in the noise texture.

# 5 Conclusion

The advantage of the procedural noise is that a precomputation for the texture mapping coordinates is not needed. Only a colour map combined with this implementation can give satisfying visual results. At the same time, it avoids the common problems associated with the texture mapping, such as distortions and seams. The users have control over the magnitude, bandwidth, frequency and orientation of the noise. It seems to be a good trade-off between the visual effect and the computation cost.

# References


[1] Lagae, A., Lefebvre, S., Drettakis, G., Dutré, P.. (2009) Procedural Noise using Sparse Gabor Convolution. ACM Transactions on Graphics, Association for Computing Machinery, 2009, 28 (3), pp.54-64. ⟨inria-00606821⟩

[2] Lewis, J.P.. (2001). Algorithms for Solid Noise Synthesis. Computer Graphics (SIGGRAPH '89 Proceedings). 23. 10.1145/74334.74360.

[3] Gabor, D.. (1946). Theory of communication. Journal of the Institute of Electrical Engineers, 93, 429–457.

[4] Bridson, R.. (2007).Fast Poisson Disk Sampling in Arbitrary Dimensions. SIGGRAPH '07: ACM SIGGRAPH 2007 sketchesAugust 2007 Pages 22–es.