INF584 - Synthèse d'Image

Yassine Mankai

Gabor Noise

Article reference: Lagae, Ares, Sylvain Lefebvre, George Drettakis, and Philip Dutré. 'Procedural Noise Using Sparse Gabor Convolution'. *ACM Transactions on Graphics* 28, no. 3 (2009): 54–64.

I/Paper Summary:

In this project, I try to implement some of the features discussed in the article. The paper presents a new technique to procedurally generate noise. This technique can be calculated on the fly with good time performance and a very low memory usage. It presents a high spectral control with intuitive parameters (orientation, principal frequency and bandwidth) which allowed the authors to create an interactive tool where the user can shape and design the intended noise in the spectral domain. In this paper, Lagae et al. proposed a use case in the form of texture generation. The noise values are used to sample on a color map. Controlling both the power spectrum control and the color map results in "visually interesting textures" in a simple manner.

1) Theory:

Gabor noise is expressed in the form of a random pulse process. It is the sum of "randomly weighted and positioned pulses using a gabor kernel g. The weights wi are sampled as the realizations of a uniform random variable W with a mean = 0. The positions are sampled as the realization of a poisson point process. In practice, we generate a random value N of the number of points using a Poisson distribution. And we generate N positions using a uniform random distribution.

$$N(X) = \sum_i \omega_i g(X - X_i)$$

The main idea of this article is the use of a gabor kernel in such process. This kernel is calculated as the product of a gaussian envelope (parameterized by a magnitude K and a width a) and harmonic (parameterized with a frequency magnitude F0 and a direction in the spectral space w).

$$g(X) = K \exp^{-\pi a^2 \|X\|^2} \cos\left[2\pi F_0 \langle X, \omega \rangle\right]$$

The advantage of such a kernel is that it can be approximated with a compact support in both the spectral and the spatial domain.

2) A generalized noise:

In the article, they present a way to have maximum control on the band-limits of the noise. Instead of generating all the values using a fixed F0 and w, we can randomly sample these to parameters inside a set of frequency regions:

$$\{[F_{j,min}, F_{j,max}] \times [\omega_{j,min}, \omega_{j,max}], j\}$$

The article investigates the different variations of this formulation to create a wide range of textures. The basic two types are:

- A band-limited isotropic noise: $[F_0, F_0] \times [0, 2pi]$
- A band-limited anisotropic noise: $[F_0, F_0] \times [\omega_0, \omega_0]$

The parameters of the Gabor kernel have an intuitive meaning in both domains. -> The frequency (F0, ω0) can be interpreted as the principal frequency, and the width a as the bandwidth (the range of frequencies around the principal frequency)

A second important feature of the presented method is the versatility in the type of generated noise. The base idea was to generate a 2D texture and apply it using a good parameterization of the mesh. This has obvious limits. For this reason, a solid version can be used for certain forms of meshes and certain types of intended material. In addition to these two, the article presents a setup-free version based on the projection of 3D sampled points onto the point tangent surface and then applying the 2D kernel.

3) Procedural Evaluation:

The paper discussed some subjects related to their implementation. The main one relates to the use of a grid to allow for an on-the-fly procedural generation. The sampling space is divided into cells of size equal to the kernel radius. The noise properties are generated using a pseudo-random generator with seeds calculated from the cell position and a global random offset. This way, we can have consistent results without the need to store the actual property values in each cell.

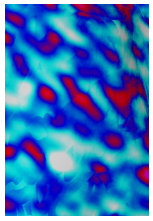
II/My Implementation:

A code was provided with the paper. I based my work on it and adapted it to make it work inside the already implemented renderer (both the rasterizer and raytracer versions). Before the scene is loaded, I run a texture generation step where I fill the data (float or vec3 array) attribute of a texture with values generated using the gabor noise. For a resolution of 256*256, It takes around 2.6 seconds to generate the texture on CPU and load it to GPU for further use. After these steps, a texture is attached to a material object and used exactly like the ones we load from disk. The main use was to generate albedo values. For this reason, I needed to convert a [0,1] value into a color in a way that allows enough control of the results. I implemented a simple color map feature where I define it as an array of n predefined colors and the [0,1] value alpha is used to linearly interpolate between the two colors having the two closest index to alpha * n.

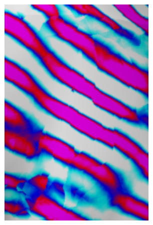
I also implemented for the raytracer the surface setup-free and the solid version of the noise. I faced an issue with the normalization of such values. For the moment, I use hard coded scaling factors to get values in the intended range but a correct way to calculate the variance should be explored.

III/Result:

To test the features implemented in the context of this project, I started by exploring the effect of some of the parameters. The orientation factor was the most obvious one as it decides how the stripes are oriented in the texture. The width of the gaussian envelope was harder for me to visualize. Here I present the result for two different values.

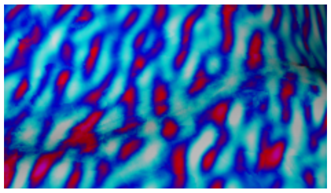


High value

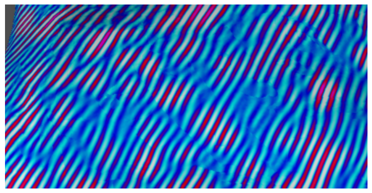


Low value

Small values of the width correspond to more regular texture as we have a smaller frequency bandwidth around the main frequency.For the frequency F0, I also presents the results for two different values



Low value



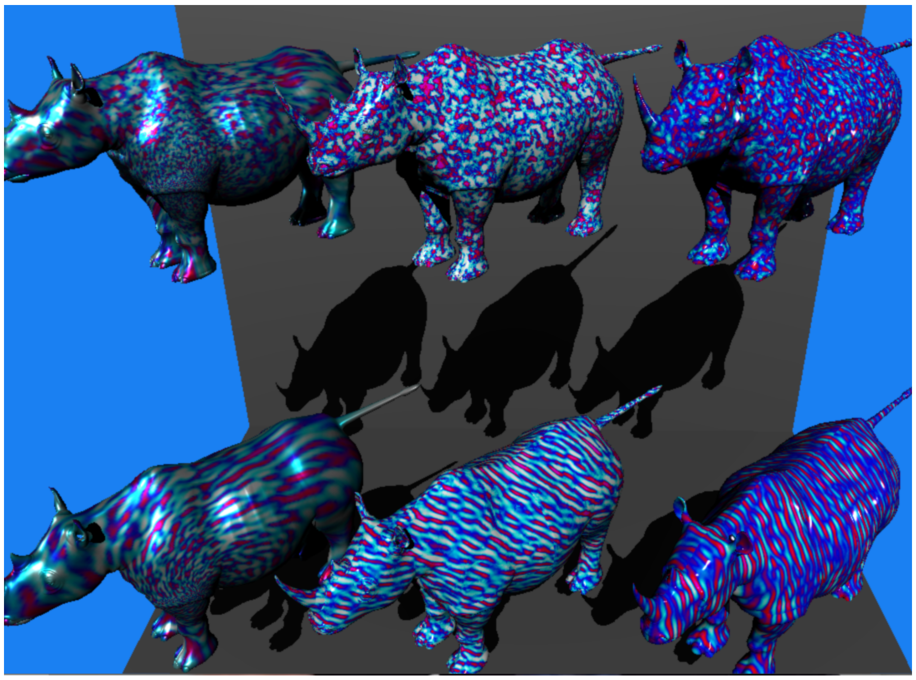
High value

We can clearly see that the frequency influences the number of stripes.

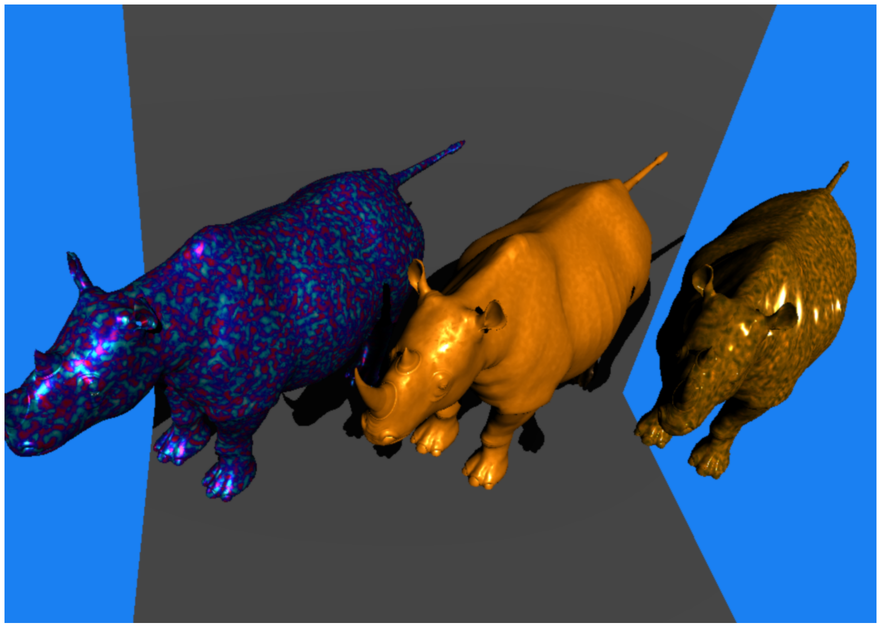
I should note that the results I presented are obtained using the setup-free surface noise. They suffer from issues related to bad normalization of the noise values but also form issues intrinsic to this technique (non smooth normals).

Finally, I created some scenes to visualize the results.

In the following one we have two rows of textured objects using gabor noise. The upper row is for isotropic noise and the lower one is for anisotropic noise. From Left to right, we have 2D texture (spherical uvs), setup free surface noise and solid noise. (Scene2.exe)



I then tried to explore how we can use this noise to generate not only albedo values but also roughness and metallicness. (Scene3.exe)



Some ideas that can be considered for the future may include the use of gabor noise for 3D modeling (GPU based displacement maps per pixel on a rough mesh).