Steerable Perlin Noise

Jacob Rice
Walt Disney Animation Studios
Burbank, USA
jake.rice@disneyanimation.com

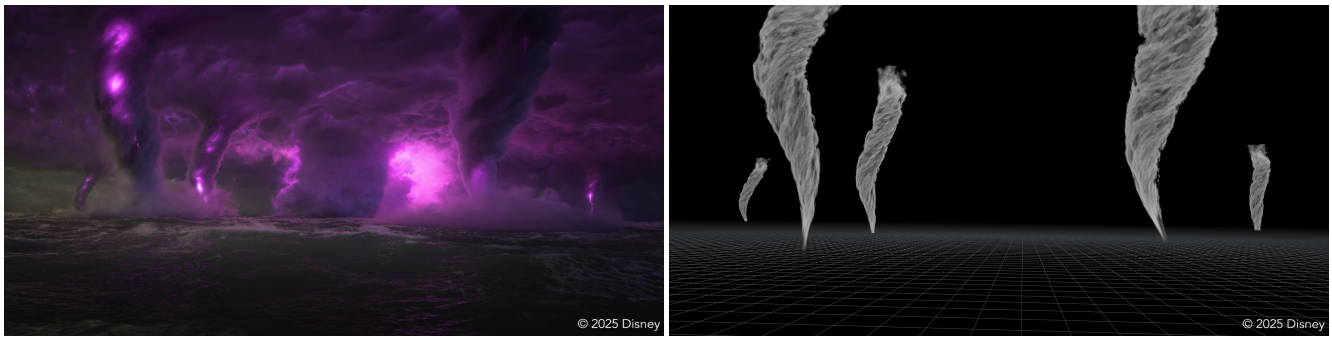


Figure 1: Tornadoes in "Moana 2" and their respective previs tornadoes textured purely with steerable Perlin noise.

Abstract

Perlin noise is an integral tool for generating procedural textures for computer graphics applications. It's widely used across multiple industries and applications due to its simplicity, speed and controllability. We present 2 simple changes to the Perlin noise algorithm that allows for the inclusion of anisotropy in the output noise. We believe these small adjustments open the door to new texturing workflows, as well as simple vector field visualizations in multiple dimensions.

ACM Reference Format:
Jacob Rice. 2025. Steerable Perlin Noise. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Talks (SIGGRAPH Talks '25), August 10-14, 2025.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3721239.3734101

1 Introduction

In the domain of noise based procedural texturing, there are a few different, important considerations. A noise that is set-up free and solid, is a type of 3D noise that does not require user intervention to prevent artifacting, which is of great importance for computer graphics artists and engineers alike. Solid noises are integral to any domain that is surface free, implicit or dynamic, in that they allow for the texturing and visualization of said domains without needing to re-discretize the domain.

Perlin [Perlin 1985] originally proposed a method for generating solid, set-up free, procedural noise. The design of his method amounts to taking the weighted average of angles and distances to random vectors stored at discrete nodes or vertices at every point in space. This approach, while incredibly efficient, is isotropic in nature, due to the uniform randomness of the gradient vectors and uniformity of the weights. This method has gone on to inspire numerous algorithms, however none of these methods offer user control for steering Perlin style noise, while maintaining the solid, and set-up free nature that makes it so powerful.

On Walt Disney Animation Studios' "Moana 2", we developed a technique to introduce steerability into Perlin noise, which we utilized as a mechanism for volumetric modeling during the production. For instance, the anisotropic qualities made it a great jumping off point when developing the tornadoes for the storm sequence during the climax of the film. While these tornadoes were ultimately replaced with simulated counterparts, the noise alone looked convincing enough for use in background tornadoes, when high fidelity wasn't needed. Furthermore, we found that the anisotropy gives the illusion of motion. We took advantage of this when developing looks for under sea volumetrics, which allowed for rapid generation without the need for costly advection-based methods, see figure (3).

2 Background

The Perlin noise algorithm can be described as a weighted sum of projections onto random vectors x_i at the vertices of some background grid v_i, from any given input sample point p. These projections are often described through the use of the dot product operator, $w_i * < p - v_i, x_i >$, where $<,>$ is the dot product. The weight w_i is a function of distance from our sample point p and any given background grid vertex v_i, typically described as a fade function.

3 Approach

Our contribution is two small adjustments:

SIGGRAPH Talks '25, Vancouver, BC, Canada

ACM ISBN 979-8-4007-1541-9/2025/08
https://doi.org/10.1145/3721239.3734101

Algorithm 1: 2D Steerable Perlin Noise

input : Sample Position P, Anisotropy Direction u, Anisotropy Strength s

$G \leftarrow MakeMetricTensor(u, s)$; // see supplemental

$gridCorner \leftarrow floor(P), p \leftarrow fract(P)$;

$n \leftarrow 0$;

for $i = 0$ **to** 1 **do**

  for $j = 0$ **to** 1 **do**

    $v_i = gridCorner + (i, j)$;

    $x_i = random(v_i)$;

    $d_i = \langle v_i - p, Gx_i \rangle$; // applies the metric

    $w_i = fade(p.x - v_i.x) * fade(p.y - v_i.y)$;

    $w_i \leftarrow w_i * fade(\langle v_i - p, G(v_i - p) \rangle)$; // aniso weights

    n += $w_i * d_i$;

  end

end

return n ;

- The incorporation of anisotropy into the projections onto the random vectors.
- The inclusion of anisotropy into the Perlin noise weights.

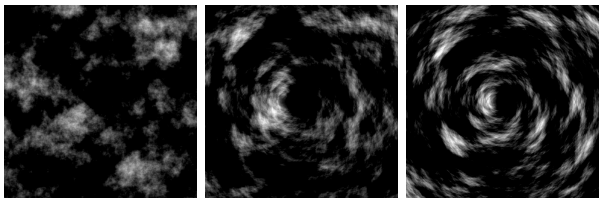
The anisotropy is applied through a metric tensor G of the users choice, which is to be supplied into the Perlin noise algorithm along with the sample position. For usability since defining metric tensors is a large topic in itself, we opt to have users instead supply a vector direction that internally is used to generate a metric tensor (see supplemental).

Incorporating anisotropy into the projection onto the random vectors x_i simply requires applying our metric to one of the vectors in the existing projection step: $< p - v_i, Gx_i >$. While this alone does create the appearance of anisotropy, the result is more subtle than one would hope (2b).

Our second contribution is to incorporate a second anisotropic weight along with the standard set of Perlin weights. In our case, it's simply a second fade function, that's nearly identical to the one from the original algorithm: $w_i * fade(< p - v_i, G(p - v_i) >)$ (2c).

3.1 Choice of Metric Tensor

The choice of metric tensor is extremely important in the success of this algorithm. In order to ensure the best results for our anisotropic weights our choice of metric tensor must satisfy the following:



(a) Original Perlin Noise (b) Metric in projection (c) Metric in proj & weights

Figure 2: Anisotropic weights
All images in Figure 2 are © 2025 Disney

Figure 3: Volumetric modeling via steerable Perlin noise

- The sum of weights at any given sample point, inside any grid cell, must be strictly positive (non zero).

This property is conveniently determined by the scaling of the eigenvalues.

Let G be some metric tensor with strictly positive eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_n$. The eigenvalues must satisfy the following constraint:

$$\sum_{i=1}^{n} \lambda_i < 4$$

By enforcing the eigenvalues sum to a value of less than 4, we're ensuring that at least one corner weight is covering the cell center (for proof, see supplemental). We also opt to specify a minimum value for our eigenvalues, which determines how slowly or quickly the distance under metric grows along the anisotropy direction:

$$\min(\lambda_1, \lambda_2, \ldots, \lambda_n) = .5$$

In practice these constraints are easy to apply, we simply need to ensure that the sum of our eigenvalues for any given metric add up to be less than 4. Since we know the smallest eigenvalue has to be .5 at the minimum in the single order case, we know the non smallest eigenvalues must sum to be less than 3.5.

See the supplemental doc for more discussion on the choice of minimum value, higher order noise and implementation details.

4 Conclusions and Future Work

Steerable Perlin Noise is an exciting tool we believe should be accessible to effects and look development artists the world over. It's simple in implementation and serves as a great jumping off point for many types of effects, both hard surface and volumetric alike. Its performance is fast enough for real time usage, and we believe in it's potential as a visualizer for general vector fields in many domains. We would also like to explore motion induced by anisotropic noise, as the output looks as if it should be moving along the anisotropy directions, however achieving motion that flows down those directions is not as simple as just offsetting the noise lookup position.

References

Ken Perlin. 1985. An image synthesizer. *SIGGRAPH Comput. Graph.* 19, 3 (July 1985), 287–296. https://doi.org/10.1145/325165.325247