Steerable Perlin Noise Supplementary

Jacob Rice

August 2025

1 Metric Tensor Discussion

1.1 Generating the Metric Tensor From an Input Vector

Given a vector $u \in \mathbb{R}^N$, we generate some initial symmetric matrix $H = uu^T$. This matrix is symmetric, but not strictly positive definite, so we must first apply a linear remapping of H's eigenvalues to ensure positive definiteness.

Let λ_i be the ith eigenvalue of H, and y_i the corresponding eigenvector of H.

The matrix H has only one non-zero eigenvalue, namely $\lambda_0 = ||u||^2$.

We wish to remap our eigenvalues, such that they're strictly positive, and such that our largest eigenvalue is remapped to be our smallest eigenvalue. This will give our final metric the property that distances measured along the direction of anisotropy are minimized. Further, we also wish to remap them such that they sum to equal a value less than 4 as a means of ensuring our weights are strictly positive over the grid cell.

Thus in order to satisfy the specified constraints we will use the following remapping process:

$$\mu_i = \frac{(k - \frac{4-k}{N-1} + \epsilon) * \lambda_i}{||u||^2} + \frac{4-k}{N-1} - \epsilon$$

where N is the number of dimensions, k is some arbitrary lower bound, and ϵ is some tiny value.

For our demos we set $k = .5$, however in practice any value of $0 < k \leq 1$ should function well. For more discussion on k see section 1.2. We also set ϵ to be a constant .0001 in our implementations.

Thus in order to convert H into a metric tensor G, we take our remapped eigenvalues μ_i and apply the following transform.

$$G = \sum_{i=1}^{n} \mu_i y_i y_i^T$$

1.2 Weight Coverage

Our goal in section 3.1 is to find the eigenvalues for G that satisfies

$$\sum_{i=1}^{n} fade(\langle p - v_i, G(p - v_i)\rangle) > 0$$

Where G is a given metric tensor, the corners of our grid cells are $v_i = (v^1, v^2, \ldots, v^N)$ where $v^i \in \{0, 1\}$ for each $i = 1, 2, \ldots, N$, and p is our sampling position.

Since our grid points v_i are stationary, in order to show coverage over the entirety of the cell, one most show coverage at the point which minimizes the distance under metric to all of the grid points for a given cell. As our sample point moves away from the minimizing position, the distance under metric to at least one of our grid corners must decrease, and therefore the corner weight will stay strictly positive.

We want to find the following

$$\min f(p)$$

where

$$f(p) = \sum_{i=1}^{n} (p - v_i)^T G(p - v_i)$$

Our objective function becomes

$$f(p) = np^T Gp - 2p^T G \sum_{i=1}^{n} v_i + \sum_{i=1}^{n} v_i^T G v_i$$

f reaches an extrema where $\nabla f(p) = 0$, so we deduce that:

$$\nabla f(p) = 2nGp - 2G \sum_{i=1}^{n} v_i$$

$$2nGp - 2G \sum_{i=1}^{n} v_i = 0$$

$$p = \frac{1}{n} \sum_{i=1}^{n} v_i$$

Since the minimizer is the centroid for the grid cell, we know this minimizer's coordinates will always be $p = \{.5, .5, \ldots, .5\}$ as our grid cell is a unit box in $\mathbb{R}^N$.

Let's write $x_i = p - v_i$; x_i can be expanded as $\{\pm.5, \pm.5, \ldots, \pm.5\}$

We want to find a property of G that guarantees that $x_i^T G x_i < 1$ for at least one grid corner v_i.

G can be decomposed as $Q^T AQ$ via the spectral theorem, where Q is an orthogonal matrix and A is a diagonal matrix of the eigenvalues of G where $A_{ii} > 0$ and $||Qu|| = ||u||$ for any vector u.

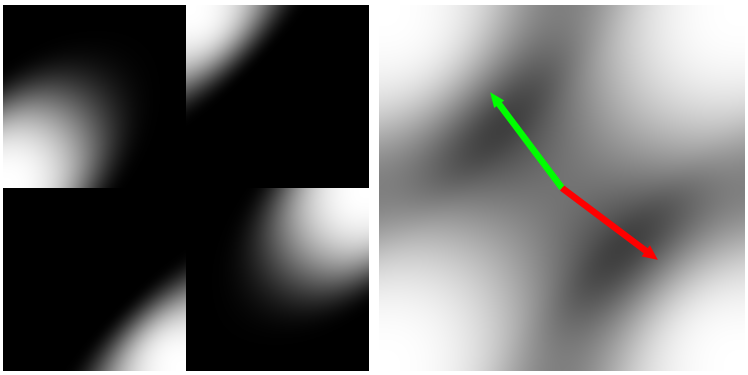
Consequently, we can work with A, the diagonal symmetric matrix without loss of generality. We have:

$$x_i^T Ax_i = \{\pm.5, \ldots, \pm.5\}^T A\{\pm.5, \ldots, \pm.5\} = .5^2 * A_{11} + .5^2 * A_{22} + \cdots + .5^2 * A_{nn}$$

Thus, $x_i^T Ax_i < 1$ becomes:

$$Tr(A) < \frac{1}{.5^2}$$
$$Tr(A) < 4$$

Which is what we set out to find.



(a) Corner Weights (b) Weights Sum and Tensor Viz

Figure 1: Anisotropic weights.
All images in Figure 1 are © 2025 Disney

1.3 Eigenvalue Discussion

The constraint that all eigenvalues of G sum to less than 4 is only required if you're interested in being able to normalize the noise with respect to the weights, that is if you want to ensure that dividing out the sum of weights will not lead to a division by 0. If that is not of interest, then you're free to set the top end values to be above 4, which will result in even greater amounts of anisotropy.

The value of k in section 1.1 of our supplementary specifies that our minimum eigenvalue of G is set to a value k. In our demos we've set this value to .5. the parameter k describes the length of the longest axis of the ellipsoid described by the levelset of $x_i^T Gx_i - 1 = 0$. When k is set to 1, this levelset will not intersect any grid cell walls. However because our method includes the Perlin weights, we're free to set k to be any value lower than 1. One is free to remove the Perlin weights, if they do set k to 1, leading to a slightly less computationally expensive algorithm.

2 Higher Order Noises

We allude to the fact that with this technique higher order noises (noises with larger sampling radii) are possible. These have the added benefit of elongating the anisotropic features.

In order to compute the anisotropic noise over a larger neighborhood of grid cells, the only 2 changes one needs to make, outside of increasing the size of the for loops to run over a larger area, is to scale the value of the smallest eigenvalue of our metric and scale the input vectors for the original (non anisotropic) Perlin weights, such that the fade function rolls off over a greater distance.

In general, we add a parameter to our noise we describe as *Quality* which is an integer from $0 \to N$. *Quality* controls how big of a neighborhood we should use for our steerable Perlin noise. Instead of looping from $0 \to 1$ along each axis, we will instead loop from $start \to end$, where start is defined as $start = -(Quality - 1)$, and $end = Quality$.

In fact, the minimum eigenvalue k should scale by $\frac{1}{Quality+1}$ as *Quality* increases. This allows for the ellipsoid our metric defines to grow in size as the search radius increases. For implementation details, see the GLSL code examples in section 5.

As for the scaling of the vector that feeds the original Perlin weights, they should be scaled such by the formula: $2/(|start| + |end| + 1)$.

However we do find that in general, single order noise is the correct balance of performance and anisotropy.

3 Higher Dimensional Noises

As dimensionality increases, the anisotropic effect of single order noise diminishes. This is due to the our eigenvalues of our metric G, needing to sum up to less than 4, regardless of dimension. As the number of eigenvalues increases, any given eigenvalue's contribution decreases. While the anisotropy is still quite noticeable in $\mathbb{R}^3$, we opt to use projected noise when applying our noise onto surfaces, as the anisotropic effects are more visible, and generating the eigen decomposition for our metric in $\mathbb{R}^2$ is more performant.

4 Comparison to Other Methods

Anisotropy in noise has been addressed in a wide range of research. van Wijk [1991] proposed Spot Noise, which outlined methods in which a convolution could be applied to existing noise algorithms, like Perlin's to allow for shaping and anisotropy in the output. Goldberg et al. [2008] describes mechanisms for baking anisotropy into the noise, in the frequency domain through oriented subbands, which gives the end user control over the directionality/steerability of the noise with the limitation of being not setup free. Galerne et al. [2012] describes a style of sparse convolution noise based off of the Gabor kernel, which is both setup free, and anisotropic. Tricard et al. [2019] proposes a reformulation

of Gabor noise, Phasor Noise, offering more flexibility in the look, sharpness and controllability of the final output.

Our noise is most similar to Spot noise in look, as the foundation for both noises is the same, however Spot noise relies on convolving the noise, which requires evaluating the noise at multiple sample points, increasing the cost of execution. Both Gabor and Phasor noise are general solid, set up free noises, with looks that differ greatly from traditional gradient noise. We believe our noise offers users a similar level of flexibility to that of Gabor or Phasor noise, while maintaining the classic gradient noise look.

5 GLSL Code Examples

5.1 Metric from Vector

```
mat2 outerprod(vec2 x, vec2 y){
    return mat2(x.x * y.x, x.y * y.x,
                x.x * y.y, x.y * y.y);
}

float fitrange(float x, float in_low,
               float in_high, float out_low,
               float out_high){

    float u = clamp(x, in_low, in_high);

    return ((out_high - out_low) * (u - in_low)) /
           (in_high - in_low) + out_low;
}

vec2 fitrange_2(vec2 x, float il, float ih, float ol, float oh){
    return vec2(fitrange(x.x, il, ih, ol, oh),
                fitrange(x.y, il, ih, ol, oh));
}

mat2 generate_metric(vec2 aniso_direction,
                     float aniso_strength,
                     int quality){

    mat2 m = outerprod(aniso_direction, aniso_direction);


    vec2 evals, evec0, evec1;
    solve_eig(m, evals, evec0, evec1);

    float k = .5 / (float(quality) + 1.0);
    int dimensions = 2;

    //this is 1, in 2d, but in higher dimensions
    //this enforces sum(evals) < 4.0
    float denom = 1. / float(dimensions - 1);

    float eval_max_old =  dot(aniso_direction, aniso_direction);
```

```
    float eval_max_new = (4.0 - k) * denom - 1e-4;
    vec2 mapped_evals = fitrange_2(evals, 0.0, eval_max_old,
                                                eval_max_new, k);


    return  mapped_evals.x * outerprod(evec0, evec0) +
            mapped_evals.y * outerprod(evec1, evec1);

}
```

5.2 Single Octave Steerable Perlin Noise

```
float steerable_perlin(vec2 p, mat2 metric){
    vec2 noise_p = floor(p);
    vec2 noise_f = fract(p);
    float out_val = 0.0;

    for(int i = 0; i <= 1; i++)
    for(int j = 0; j <= 1; j++){
        vec2 o = vec2(i,j);
        vec2 g = noise_p  + o;
        vec2 r = rand_dir(g); // random vector
        vec2 v = o - noise_f; //dir to corner
        vec2 metric_v = v * metric;
        float d = dot(r, metric_v); //inner product
        float w = interp(v.x) * interp(v.y); //perlin weights
        w *= interp(dot(v, metric_v)); //aniso weights
        out_val += d * w;
    }
    return out_val;

}
```

5.3 Single Octave Higher Order Steerable Perlin Noise

```
float steerable_perlin(vec2 p, mat2 metric, int quality){
    vec2 noise_p = floor(p);
    vec2 noise_f = fract(p);
    float out_val = 0.0;
    int start = -(quality);
    int end = quality + 1;
    float scale = 2. / (abs(start) + end + 1);

    for(int i = start; i <= end; i++)
    for(int j = start; j <= end; j++){
        vec2 o = vec2(i,j);
        vec2 g = noise_p  + o;
        vec2 r = rand_dir(g); // random vector
        vec2 v = o - noise_f; //dir to corner
        vec2 metric_v = v * metric;
        float d = dot(r, metric_v); //inner product
        float w = interp(v.x * scale) * interp(v.y * scale);
        w *= interp(dot(v, metric_v)); //aniso weights
        out_val += d * w;
```

```
    }
    return out_val;

}
```

6 Examples

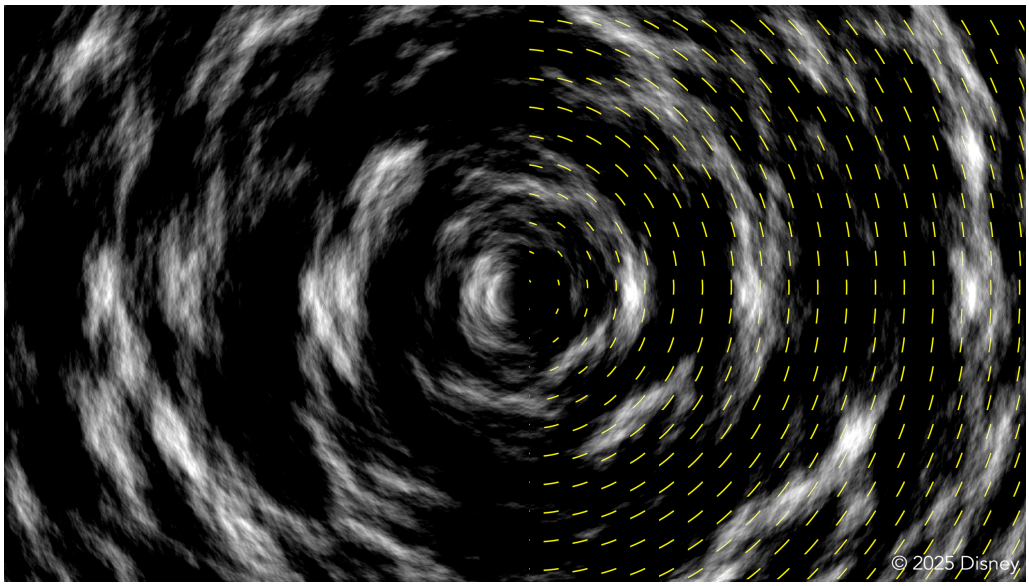


Figure 2: Circular Vector Field

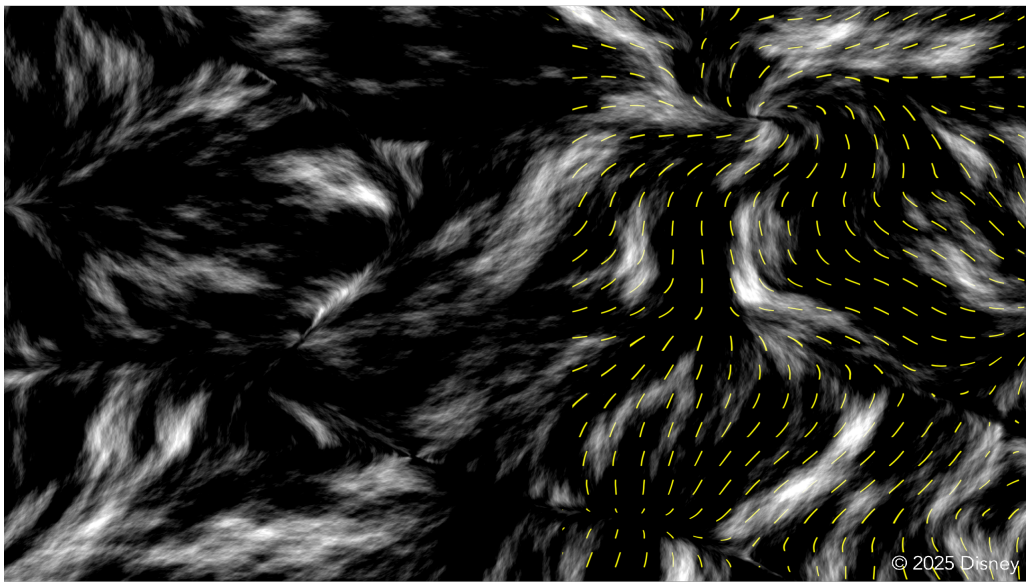


Figure 3: Image Driven Vector Field

References

Bruno Galerne, Ares Lagae, Sylvain Lefebvre, and George Drettakis. Gabor noise by example. *ACM Trans. Graph.*, 31(4), July 2012. ISSN 0730-0301. doi: 10.1145/2185520.2185569. URL https://doi.org/10.1145/2185520.2185569.

Alexander Goldberg, Matthias Zwicker, and Frédo Durand. Anisotropic noise. *ACM Trans. Graph.*, 27(3):1–8, August 2008. ISSN 0730-0301. doi: 10.1145/1360612.1360653. URL https://doi.org/10.1145/1360612.1360653.

Ken Perlin. An image synthesizer. *SIGGRAPH Comput. Graph.*, 19(3):287–296, July 1985. ISSN 0097-8930. doi: 10.1145/325165.325247. URL https://doi.org/10.1145/325165.325247.

Thibault Tricard, Semyon Efremov, Cédric Zanni, Fabrice Neyret, Jonàs Martínez, and Sylvain Lefebvre. Procedural phasor noise. *ACM Trans. Graph.*, 38(4), July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322990. URL https://doi.org/10.1145/3306346.3322990.

Jarke J. van Wijk. Spot noise texture synthesis for data visualization. *SIGGRAPH Comput. Graph.*, 25(4):309–318, July 1991. ISSN 0097-8930. doi: 10.1145/127719.122751. URL https://doi.org/10.1145/127719.122751.