Rotating Noise Vectors: the Full Story

Ian McEwan & Stefan Gustavson

November 18, 2021

About this document

This is the complete derivation of the algorithms we use to generate the rotating generating vectors, the "gradients", for the 3-D version of our simplex noise. It's heavy on mathematics, but it requires nothing beyond ordinary vector algebra to follow.

1 Introduction

For our 3-D version of "flow-noise capable" simplex noise, we want to generate pseudorandom "rotating vectors". By that, we mean a unit vector $\hat{g}_{\vec{x}}(\alpha)$ that, for some location $\vec{x}$, gives a random direction that can be smoothly rotated around some random axis $\hat{r}$ by a given angle α. One way to do this is to pick two unit vectors $\hat{p}$, $\hat{q}$ on a unit sphere and use them to define a plane of rotation. See Figure 1. The sections below present three algorithms for this. Section 2 presents a method of picking two vectors by two separate hashes, and then we proceed to show that is sufficient and equivalent to pick a single vector from a 2π-ball, using a single hash, to derive an equivalent rotation. Sections 3 and 4 present two alternative, faster methods which are equivalent in the limit of very large hash functions, but have their own properties in practical use. The first one is optimized for speed, while the second one is more well-behaved and allows for a very fast shortcut when no rotation is needed.

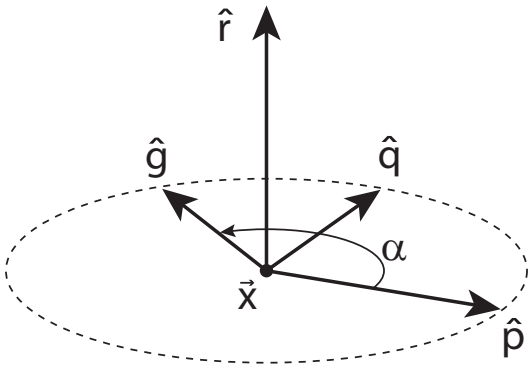

Figure 1: *Rotation of a vector $\hat{p}$ at point $\vec{x}$ around the axis $\hat{r}$ by an angle α towards another vector $\hat{q}$, to yield a rotating unit vector $\hat{g}$.*

2 Two-vector method

The two-vector method of generating $\hat{g}_{\vec{x}}(\alpha)$ uses two distinct hashes of $\vec{x}$ to compute two different unit direction vectors. These are then used to define a plane of rotation in which the first direction vector is rotated by α towards the direction of the second vector to produce a single, rotated unit direction vector.

Specifically, the two-vector algorithm proceeds as follows:

1. Generate two unit direction vectors from $\vec{x}$ as points on a sphere via two 1D hashes:

   $$\hat{p} = \hat{\mathfrak{g}}(\sigma_1(\vec{x})) \qquad \hat{q} = \hat{\mathfrak{g}}(\sigma_2(\vec{x}))$$

   Where $\hat{\mathfrak{g}}$ is a unit-vector selection function with a uniform distribution, and σ_1 and σ_2 are distinct hashing functions.
   Together these two vectors form a bi-vector which encodes four information components: a plane of rotation, an amount to rotate by, and a rotational offset that is usually discarded.

2. $\hat{q}$ is orthogonalized (and renormalized) with respect to $\hat{p}$ such that the plane remains unchanged.
   Note: This discards the amount to rotate by, as $\hat{p}$ and $\hat{q}$ are now always $\frac{\pi}{2}$ apart, but keeps the offset now encoded in both $\hat{p}$ and $\hat{q}$.

3. The plane of rotation can also be defined by its normal, the axis of rotation:

   $$\hat{r} = \hat{p} \times \hat{q}$$

   $\hat{p}$, $\hat{q}$ and $\hat{r}$ form an orthonormal basis.

4. The final vector is computed by rotating from $\hat{p}$ by a given angle α in the $\hat{p} \times \hat{q}$ plane. Using

CC BY NC ND

the orthogonality of $\hat{p}$ and $\hat{q}$, this can be neatly expressed as:

$$\hat{g}_{\vec{x}}(\alpha) = \mathrm{rot}(\hat{r}, \alpha) = \hat{p}\cos\alpha + \hat{q}\sin\alpha$$

3 Fast Method

We realize that the normal $\hat{r}$ is just a point on the sphere like $\hat{p}$ and $\hat{q}$, and so we could start with $\hat{r}$ and work backwards to get an equivalent $\hat{p}$ and $\hat{q}$:

1. Generate a unit direction vector using a single hash value:

$$\hat{r} = \hat{\mathfrak{g}}(\sigma_1(\vec{x}))$$

2. The plane which is normal to $\hat{r}$ (and thus parallel to $\hat{p}$ and $\hat{q}$) intersects any non-parallel great circle on the unit sphere at two points. These can be found by crossing the axis of rotation with the appropriate poles. We choose the equator and the $\vec{n} = (0, 0, 1)$ pole, but any unit-vector that does not collide with $\hat{\mathfrak{g}}$ is acceptable:

$$\hat{q}' = \widehat{\hat{r} \times \hat{n}}$$

3. We can find a vector orthonormal to $\hat{r}$ and $\hat{q}'$ by crossing again:

$$\hat{p}' = \hat{q}' \times \hat{r}$$

4. $\hat{p}'$ and $\hat{q}'$ are, by construction, orthogonal and in the same plane as $\hat{p}$ and $\hat{q}$ as defined in the preceding section, but rotated by some angle ψ. In the two-vector method, ψ could be computed from $\hat{p}$ and $\hat{q}$, but we observe that it is a uniformly distributed angle, and it influences only the rotation offset. As a consequence, its exact value is not of primary importance, and we choose to generate ψ directly from a hash instead of computing it. We choose to re-use the hash value from step 1 to feed into a scalar mapping function $\mathfrak{h}$ to get ψ:

$$\psi = \mathfrak{h}(\sigma_1(\vec{x}))$$

5. The final vector can be computed as before, but taking into account ψ:

$$\hat{g}_{\vec{x}}(\alpha) = \hat{p}'\cos(\alpha - \psi) + \hat{q}'\sin(\alpha - \psi)$$

This algorithm is stable and fast, but has two potential disadvantages:

1. The distribution of $\hat{g}$ is not a proper equal-area mapping for any value of α, and can be a little messy for small hash functions.

2. No single determinate value of α is able to regenerate $\hat{g}_{\vec{x}}(\alpha) = \hat{p}$ if we want the rotating vector function to coincide with the non-rotating vector for each point $\vec{x}$. The noise patterns for the non-rotating vectors and the rotating vectors will be statistically equivalent, but different.

   The method in the next section addresses these flaws by requiring that $\hat{p}$ is an equal-area distribution of directions, and assuring that $\hat{g}_{\vec{x}}(0) = \hat{p}$.

4 Well-behaved method

The effect of the orthogonalization step in the two-vector algorithm is to project the second direction vector onto a point on the great circle normal to $\hat{r}$, that is, $\hat{q}$ becomes a point on the equator of a sphere whose north pole is $\hat{r}$. If we rotate the entire sphere so that $\hat{r}$ is rotated onto the $\vec{n} = (0, 0, 1)$ pole, then $\hat{q}$ would be on the equator. Running this backwards gives us and algorithm that does not have the disadvantages of the optimized method above, at the cost of some additional computations to perform the extra rotation:

1. Generate a unit direction vector and a point on the equator:

$$\hat{p} = \hat{\mathfrak{g}}(\sigma_1(\vec{x})) \qquad \hat{q}' = \hat{\mathfrak{h}}(\sigma_1(\vec{x}))$$

   Where $\hat{\mathfrak{h}}$ now selects a pseudo-random direction on the equator. Note that, once again, we re-use the same hash value for both selections. This requires some care in the mapping, so that the directions of $\hat{g}$ do not become correlated in a manner that shines through in the final animated noise pattern.

2. The rotation angle ξ from $\hat{n}$ to $\hat{p}$ is defined by the normal to both vectors and their scalar product.

$$\hat{r} = \widehat{\hat{p} \times \hat{n}} \qquad \xi = \arccos(\hat{p} \bullet \hat{n})$$

3. Now, rotate $\hat{q}'$ with the rotation that takes $\hat{n}$ to $\hat{p}$:

$$\hat{q} = \hat{q}'\cos\xi - (\hat{r} \times \hat{q}')\sin\xi + \hat{r}(\hat{r} \bullet \hat{q}')(1 - \cos\xi)$$

4. Finally, compute $\hat{g}$ as before:

$$\hat{g}_{\vec{x}}(\alpha) = \hat{p}\cos\alpha + \hat{q}\sin\alpha$$

CC BY NC ND

F11A3F8110946E1816F767C5DBA2919F
November 18, 2021

5 Implementation

The implementations of the optimized algorithm and the well-behaved algorithm both start with the no-rotation vector generation. Many of the operations are similar between the two.

For brevity, we introduce the notation $s_* = \sin(*)$, and $c_* = \cos(*)$, e.g. $c_\theta = \cos\theta$.

The 3-D hash k is computed by a permutation polynomial $k = P(i)$, operating modulo-N on the integer lattice coordinates (u, v, w) for the simplex corner. We hash each of the three dimensions in sequence with the same hash function, chained by an addition, in the tradition of previous procedural noise implementations:

$$\vec{x} = (x, y, z)\sigma_1(\vec{x}) = P(P(P(u) + v) + w)$$

We choose $\hat{\mathfrak{g}}(k)$ as an equal-area mapping on a unit sphere through a Fibonacci spiral, and $\mathfrak{h}(k)$ as a linear mapping from k to compute an angle uniformly distibuted over $[0, 2\pi]$. Working out the math, this breaks down as:

$$\hat{\mathfrak{g}}(k) = \begin{bmatrix} s_\phi c_\theta \\ s_\phi s_\theta \\ c_\phi \end{bmatrix} = \begin{bmatrix} c_\theta\sqrt{1-c_\phi^2} \\ s_\theta\sqrt{1-c_\phi^2} \\ c_\phi \end{bmatrix}$$

$$\mathfrak{h}(k) = \psi$$

Where:

$$\begin{aligned} k &\in \mathbb{Z}_N \\ \theta &= a_1 k \mod 2\pi \\ \psi &= (a_2 k + c_2) \mod 2\pi \\ c_\phi &= (a_3 k + c_3) \mod \pm 1 \end{aligned}$$

Here, a_i and c_i are mapping constants that need to be chosen appropriately to avoid undesirable visual structure in the noise pattern due to inter-correlation between vectors created from the same hash k. For a proper Fibonacci spiral mapping of rotation axes, which gives a nice equal-area distribution of directions on the sphere, we want to set $a_1 = \frac{2\pi}{\varphi}$ where φ is the golden ratio $(1+\sqrt{5})/2$, $a_3 = -\frac{2}{289}$ and $c_3 = 1 - \frac{1}{289}$. Our particular implementation uses $a_2 = \frac{10\pi}{289}$ and $c_2 = 0$. We do not claim that these values are optimal in any formal manner with respect to the visual quality of the noise output, but they give a nice-looking animated noise field without visible structure such as semi-regular patterns or inadvertent spatial and/or temporal periodicity.

5.1 No rotation

If rotating gradients are not desired, $\hat{\mathfrak{g}}$ is simply passed through, with α ignored:

$$\hat{g}_{\vec{x}} = \hat{\mathfrak{g}}(\sigma_1(\vec{x}))$$

5.2 Fast algorithm

In summary, the computations required are:

$$\begin{aligned} \hat{r} &= \hat{\mathfrak{g}}(\sigma_1(\vec{x})) \\ \psi &= \mathfrak{h}(\sigma_1(\vec{x})) \\ \hat{q}' &= \widehat{\hat{r} \times \hat{n}} \\ \hat{p}' &= \hat{q}' \times \hat{r} \\ \hat{g}_{\vec{x}}(\alpha) &= \hat{p}'\cos(\alpha - \psi) + \hat{q}'\sin(\alpha - \psi) \end{aligned}$$

Expanding out the cross products, we get:

$$\hat{q}' = \widehat{\hat{r} \times \hat{n}} = \left\| \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ r_x & r_y & r_z \\ 0 & 0 & 1 \end{vmatrix} \right\| = \frac{1}{\sqrt{r_y^2 + r_x^2}} \begin{bmatrix} r_y \\ -r_x \\ 0 \end{bmatrix}$$

$$\hat{p}' = \hat{q}' \times \hat{r} = \begin{vmatrix} \hat{i} & \hat{j} & \hat{k} \\ q'_x & q'_y & 0 \\ r_x & r_y & r_z \end{vmatrix} = \frac{1}{\sqrt{r_y^2 + r_x^2}} \begin{bmatrix} -r_x r_z \\ -r_y r_z \\ r_y^2 + r_x^2 \end{bmatrix}$$

Looking at the normalization square roots, we have:

$$\sqrt{r_y^2 + r_x^2} = \sqrt{\left(1 - c_\phi^2\right) c_\theta^2 + \left(1 - c_\phi^2\right) s_\theta^2} = \sqrt{1 - c_\phi^2}$$

So, for $\hat{p}$ and $\hat{q}$, this yields:

$$\hat{q}' = \frac{\cancel{\sqrt{1-c_\phi^2}}}{\cancel{\sqrt{1-c_\phi^2}}} \begin{bmatrix} s_\theta \\ -c_\theta \\ 0 \end{bmatrix} \quad \hat{p}' = \frac{\cancel{\sqrt{1-c_\phi^2}}}{\cancel{\sqrt{1-c_\phi^2}}} \begin{bmatrix} -c_\phi c_\theta \\ -c_\phi s_\theta \\ \sqrt{1-c_\phi^2} \end{bmatrix}$$

And, finally:

$$\hat{g}_{\vec{x}}(\alpha) = \hat{p}'\cos(\alpha - \psi) + \hat{q}'\sin(\alpha - \psi)$$

5.3 Well-behaved algorithm

In summary, we need to compute:

$$\begin{aligned} \hat{p} &= \hat{\mathfrak{g}}(\sigma_1(\vec{x})) \\ \hat{q}' &= \hat{\mathfrak{h}}(\sigma_1(\vec{x})) \\ \hat{r} &= \widehat{\hat{p} \times \hat{n}} \\ c_\xi &= \hat{p} \bullet \hat{n} \\ \hat{q} &= \hat{q}' c_\xi - (\hat{r} \times \hat{q}') s_\xi + \hat{r}(\hat{r} \bullet \hat{q}')(1 - c_\xi) \\ \hat{g}_{\vec{x}}(\alpha) &= \hat{p} c_\alpha + \hat{q} s_\alpha \end{aligned}$$

CC BY NC ND

F11A3F8110946E1816F767C5DBA2919F
November 18, 2021

$\hat{p}$ and $\hat{q}'$ expand as:

$$\hat{p} = \begin{bmatrix} c_\theta\sqrt{1-c_\phi^2} \\ s_\theta\sqrt{1-c_\phi^2} \\ c_\phi \end{bmatrix} = \begin{bmatrix} c_\theta s_\phi \\ s_\theta s_\phi \\ c_\phi \end{bmatrix} \quad \hat{q}' = \begin{bmatrix} s_\psi \\ c_\psi \\ 0 \end{bmatrix}$$

Then $\vec{r} = s_\xi \hat{r}$, which means $\hat{r} = \frac{1}{s_\xi}(\hat{p} \times \hat{n})$, and $\hat{r}$ and c_ξ expand as:

$$\vec{r} = \hat{p} \times \hat{n} = \begin{vmatrix} i & j & k \\ p_x & p_y & p_z \\ 0 & 0 & 1 \end{vmatrix} = \begin{bmatrix} p_y \\ -p_x \\ 0 \end{bmatrix}$$
$$c_\xi = \hat{p} \bullet \hat{n} = p_z = c_\phi$$

Focusing on the rotation, we have:

$$\begin{aligned} \hat{q} &= \hat{q}' c_\phi + \hat{r}\,(\hat{r} \bullet \hat{q}')\,(1 - c_\phi) - (\hat{r} \times \hat{q}')\,s_\phi \\ &= \hat{q}' c_\phi + \frac{\vec{r}(\vec{r} \bullet \hat{q}')}{s_\phi^2}(1 - c_\phi) - (\vec{r} \times \hat{q}') \end{aligned}$$

Expanding the dot and cross products yields:

$$\begin{aligned} \vec{r} \times \hat{q}' &= \begin{vmatrix} i & j & k \\ r_x & r_y & r_z \\ q'_x & q'_y & q'_z \end{vmatrix} = \begin{vmatrix} i & j & k \\ p_y & -p_x & 0 \\ s_\psi & c_\psi & 0 \end{vmatrix} \\ &= \begin{bmatrix} 0 \\ 0 \\ p_y c_\psi + p_x s_\psi \end{bmatrix} \\ \vec{r} \bullet \hat{q}' &= p_y s_\psi - p_x c_\psi \\ &= (s_\psi s_\theta - c_\psi c_\theta)\,s_\phi \end{aligned}$$

Substitute $\vec{r} \bullet \hat{q}'$ into the middle term of the rotation:

$$\begin{aligned} &= \frac{\vec{r}(\vec{r} \bullet \hat{q}')}{s_\phi^2}(1 - c_\phi) \\ &= \begin{bmatrix} s_\theta \\ -c_\theta \\ 0 \end{bmatrix}(s_\psi s_\theta - c_\psi c_\theta)(1 - c_\phi) \end{aligned}$$

The components of $\hat{q}$ are then:

$$\begin{aligned} q_x &= s_\psi c_\phi + s_\theta\,(s_\psi s_\theta - c_\psi c_\theta)\,(1 - c_\phi) \\ q_y &= c_\psi c_\phi - c_\theta\,(s_\psi s_\theta - c_\psi c_\theta)\,(1 - c_\phi) \\ q_z &= -\,(p_y c_\psi + p_x s_\psi) \end{aligned}$$

The x and y components can be implemented as a `mix()` in shader code:

$$\hat{q}_{xy} = \mathrm{mix}\left(\begin{bmatrix} s_\theta\,(s_\psi s_\theta - c_\psi c_\theta) \\ -c_\theta\,(s_\psi s_\theta - c_\psi c_\theta) \end{bmatrix}, \begin{bmatrix} s_\psi \\ c_\psi \end{bmatrix}, c_\phi\right)$$

And, finally:

$$\hat{g}_{\vec{x}}(\alpha) = \hat{p}\cos(\alpha) + \hat{q}\sin(\alpha)$$

Note that for $\alpha = 0$, $\hat{g}_{\vec{x}} = \hat{p}$, which falls back to the simple no-rotation algorithm.

6 Discussion

By computing $\vec{q}$ via some geometric transforms, and performing algebraic optimization, we have created two fast algorithms that compute a pseudo-random rotating vector using only one hash function. A simplex in 3-D has four corners, and the 4-vector parallelism in GPU shading languages maps nicely to computing vectors for all four corners in parallel.

The no-rotation algorithm, using just the Fibonacci spiral on a sphere, uses 5 quad-FMAs and 4 scalar *sincos* operations to compute four pseudo-random vectors. The fast algorithm uses a total of 8 quad-FMAs and 8 scalar *sincos* ops to compute four rotating vectors, and the well-behaved algorithm uses a total 11 quad-FMAs and 9 scalar *sincos* operations. On a modern GPU, the *sincos* operation is heavily accelerated, and these algorithms show good performance.

The fast algorithm has a slight but useful speed advantage over the more well-behaved algorithm, but we chose the well-behaved one as the default in our implementation, because of its fast dynamic shortcut when $\alpha = 0$. The code contains provisions to switch to the fast algorithm by changing a simple `#define` statement.

CC BY NC ND