Spherical Fibonacci Mapping

Benjamin Keinert[1]* Matthias Innmann[1] Michael Sänger[2] Marc Stamminger[1]
[1]University of Erlangen-Nuremberg [2]hypnolords GbR

Abstract

Spherical Fibonacci point sets yield nearly uniform point distributions on the unit sphere $\mathcal{S}^2 \subset \mathbb{R}^3$. The forward generation of these point sets has been widely researched and is easy to implement, such that they have been used in various applications.

Unfortunately, the lack of an efficient mapping from points on the unit sphere to their closest spherical Fibonacci point set neighbors rendered them impractical for a wide range of applications, especially in computer graphics. Therefore, we introduce an inverse mapping from points on the unit sphere which yields the nearest neighbor in an arbitrarily sized spherical Fibonacci point set in constant time, without requiring any precomputations or table lookups.

We show how to implement this inverse mapping on GPUs while addressing arising floating point precision problems. Further, we demonstrate the use of this mapping and its variants, and show how to apply it to fast unit vector quantization. Finally, we illustrate the means by which to modify this inverse mapping for texture mapping with smooth filter kernels and showcase its use in the field of procedural modeling.

CR Categories: I.3.m [Computer Graphics]: Miscellaneous

Keywords: spherical Fibonacci, inverse mapping, constant time

1 Introduction

Spherical Fibonacci (SF) point sets are a well-known approach to generate a very uniform sampling of the sphere. A large number of applications use this type of sample distributions on spheres, e.g. numerical simulations on the sphere as presented by [Swinbank and Purser 2006], or for uniformly positioned display pixels on spherical surfaces [Brockmeyer et al. 2013]. Additionally, the benefits of spherical Fibonacci point sets have been recognized in computer graphics to be well suited for quasi-Monte Carlo ray tracing techniques [Marques et al. 2013].

However, these point sets possess a major drawback: In contrast to other spherical sample distributions (e.g. cube maps [Greene 1986], Octahedron Environment Maps [Engelhardt and Dachsbacher 2008]), the means by which to find the closest sample point for a given point on the unit sphere remain cumbersome. Literature suggests that an exhaustive search is necessary [Larkins et al. 2012], rendering these point sets ineligible for many applications. In this paper, we present an analytic and efficient solution for this problem. We describe how the nearest and surrounding samples can be determined efficiently for an SF point set of arbitrary size, given an arbitrary point on the unit sphere. We address precision problems and show how to overcome these by exploiting the unique properties of SF point sets and leveraging fused multiply-add operations.

*e-mail: benjamin.keinert@fau.de

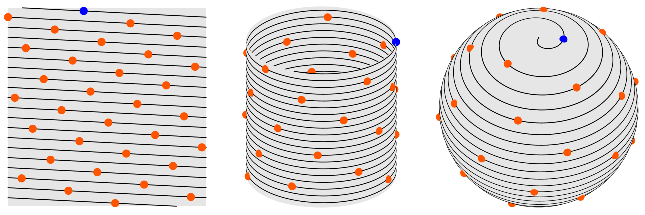


Figure 1: *Illustration of the construction of the point set* $\mathbf{SF}_i^{32}$*, showing the mapping from the grid through a cylinder to the sphere. The first point of the point set is marked blue.*

Our inverse mapping opens the door to a number of spherical Fibonacci point set applications that have been deemed impractical up to now. We show, how unit vectors can be quantized very efficiently, how spherical functions can be stored with highly improved uniform cell sizes, and how we can apply the inverse mapping to the field of procedural modeling. Although we focus on the illustration of our mapping for spheres for the remainder of this paper, our inverse mapping can readily be adapted to hemispheres as well. As an example, we show the application of an hemispherical variant for normal quantization.

2 Previous Work

Phyllotactic patterns associated with Fibonacci numbers appear in countless places in nature [Newell and Shipman 2005]. They have been the scope of scientific investigations for several decades. For instance, [Vogel 1979] discussed how to construct – probably the most popular example of these patterns – the sunflower head.

[Larkins et al. 2012] surveyed various spherical data structures for normal binning, where SF point sets were the most accurate method. Although they improved the performance of the inverse mapping by examining only a subset of all candidates, their approach was still resource-intensive and did not run in constant time.

In the context of numerical integration on spheres, the preeminence of spherical Fibonacci point sets has been discussed in several publications [Hannay and Nye 2004], [González 2010]. [Marques et al. 2013] used spherical Fibonacci point sets for quasi-Monte Carlo (QMC) ray tracing and showed that this sampling pattern is superior to other QMC sampling strategies on the hemisphere.

Various applications for procedural content generation based on phyllotaxis were shown in [Fowler et al. 1992]. [Brockmeyer et al. 2013] approached designing and developing hemispherical displays with printed optics by using an SF grid pixel distribution. Other fields such as meteorology [Swinbank and Purser 2006] have also taken an interest in SF's admirably uniform and nearly isotropic sample distribution. They demonstrated the feasibility of using this grid for a Eulerian finite-difference model for shallow-water simulations. The authors also provide a comprehensive derivation of many relations, which we based our work upon.

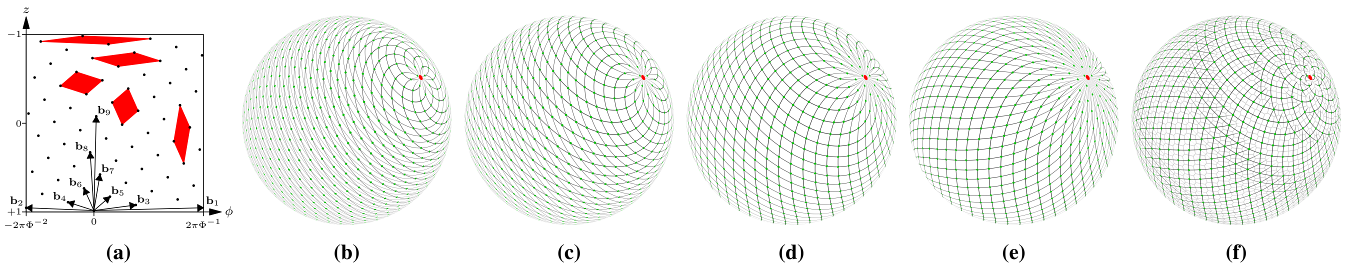


Figure 2: *(a) Basis vectors $\mathbf{b}_k$ on the grid $(\phi, z)^{\mathrm{T}}$ for $n = 64$. The red quadrilaterals correspond to cells of grids formed by different consecutive basis vectors. Note that due to the horizontal wraparound $\mathbf{b}_1 \equiv \mathbf{b}_2$. (b)-(e) Basis vector pairs $\mathbf{b}_k, \mathbf{b}_{k+1}$ for $n = 1024$ and from left to right $k = 6, 7, 8, 9$. Each basis vector pair forms nearly equally sized grid cells in different areas of the sphere. (f) Dominant SF basis vector pair visualization for $n = 1024$ with the pole marked in red, SF samples green, and the Voronoi tessellation in light gray.*

3 Spherical Fibonacci Point Sets

In the following, we limit our work to spherical Fibonacci point sets of the form as described by [Marques et al. 2013]. The construction rules for these sets are straightforward when using spherical coordinates $(\phi, \theta)^{\mathrm{T}}$, for which we use the common parameterization:

$$\mathbf{P}(\phi, \theta) = (\cos(\phi)\sin(\theta),\ \sin(\phi)\sin(\theta),\ \cos(\theta))^{\mathrm{T}}.$$

A visualization of the underlying geometric construction of SF point sets is shown in Figure 1. We denominate points on the sphere using parameters $(\phi, z = \cos\theta)^T$. In this space, the samples follow a line (with horizontal wraparound), and a generative spiral on the sphere. The step size along this spiral is defined in terms of the *golden ratio* $\Phi = (\sqrt{5}+1)/2$, the positive solution of the equation $\Phi^{-1} = \Phi - 1$. A point with index i of an SF point set with n samples is given as:

$$\mathbf{SF}_i^n = \mathbf{P}\left(\phi_i, \cos^{-1}(z_i)\right), \tag{1}$$

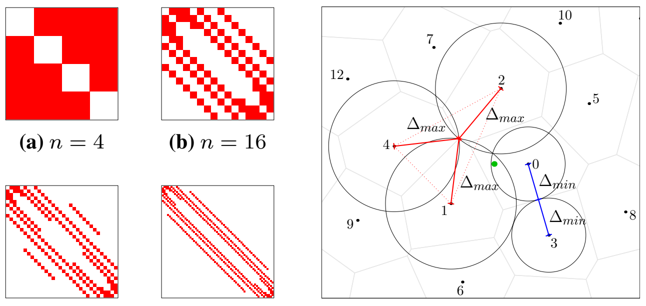
$$\phi_i = 2\pi\left[\frac{i}{\Phi}\right],\quad z_i = 1 - \frac{2i+1}{n},\quad i \in \{0, \ldots, n-1\}, \tag{2}$$

where $[x]$ is the fractional part of x: $[x] = x - \lfloor x \rfloor$. Following the work of [Marques et al. 2013], [González 2010] and [Swinbank and Purser 2006], z_i is chosen such that the first and last sample of the point set do not coincide with the poles, since this yields an overall more uniform configuration.

In parameter space, this grid is similar to a Rank-1-Lattice [Dammertz and Keller 2008] in 2D. Each of these lattices has a generative vector, but due to the wraparound the points form a 2D lattice. As shown by [Swinbank and Purser 2006] for the case of Fibonacci grids, useful basis vectors $\mathbf{b}_k$ are the vectors between two points with a Fibonacci number F_k as index difference – where $\mathbf{b}_2$ acts as a generative vector under horizontal wraparound. Figure 2a shows the first few basis vectors in the parameter domain. These basis vectors can be expressed as:

$$\mathbf{b}_k = \left(-(-1)^k 2\pi\Phi^{-k},\ \frac{-2F_k}{n}\right)^{\mathrm{T}}. \tag{3}$$

An important observation is that two successive basis vectors form grid cells with varying anisotropy. The mapping to the sphere changes anisotropy again, depending on z. The effect on the spherical grid can be seen in Figure 2b - 2e, where distinct pairs of consecutive basis vectors generate nearly square cells in different areas of the sphere depending on k and z. For each z, the pair of basis vectors $\mathbf{b}_k, \mathbf{b}_{k+1}$ with the most quadratic cells is called the *dominant* pair, and k the *zone number* of z. Figure 2f illustrates the resulting dominant basis vector pairs varying over the sphere, which yield



(a) $n = 4$ **(b)** $n = 16$ **(c)** $n = 32$ **(d)** $n = 64$ **(e)** *Configuration at the North Pole*

Figure 3: *(a)-(d) Adjacency matrices of the Delaunay triangulations for different n. The tessellation at the poles becomes constant for $n \geq 16$. (e) Close-up of the topology at the North Pole (green) showing the geometry generating the bounds Δ_{max} and Δ_{min}. Straight lines in this image represent great circular arcs.*

nearly isotropic *local grids*. These local grids are the foundation of our inverse mapping (Section 4). In line with [Swinbank and Purser 2006], we derive the zone number for our parameterization by expressing the basis vectors in terms of local Cartesian coordinates:

$$\hat{\mathbf{b}}_k = \left(-(-1)^k 2\pi\Phi^{-k}\sqrt{1-z^2},\ \frac{-2F_k}{n}\frac{1}{\sqrt{1-z^2}}\right)^{\mathrm{T}}. \tag{4}$$

To be able to derive the dominant zone number $\hat{k}$, we use the approximation $F_k \approx \Phi^k/\sqrt{5}$, as shown by [Swinbank and Purser 2006]. As carried out in detail in the appendix, we compute the derivative of $\|\hat{\mathbf{b}}_k\|_2^2$ and set it to zero, yielding the real-valued zone number $\hat{k}$ as:

$$\hat{k} = \log_{\Phi^2}\left(\sqrt{5}n\pi\sin^2\theta\right) = \log_{\Phi^2}\left(\sqrt{5}n\pi(1-z^2)\right) \tag{5}$$

The zone number changes smoothly, resulting in a smooth transition of the local grid and thus Voronoi cells. Only near the poles the zone number changes quickly, and the regular grid structure gets lost. See Figure 3e for a close-up illustration of the geometric configuration at the north pole. This configuration can also be found at the south pole in rotated form. In Figure 3a - 3d, the adjacency matrices of the Delaunay triangulation are shown for various n. As can be seen, the topology near the poles becomes constant for $n \geq 16$. Since the Voronoi tessellation is dual to the Delaunay triangulation, this property applies to the Voronoi topology equally. The most irregular Voronoi cells are near the poles, where the zone number changes most rapidly, but since the topology converges to a constant configuration in the polar areas for increasing n, we observe the following bounds on the extents of the Voronoi cells: The

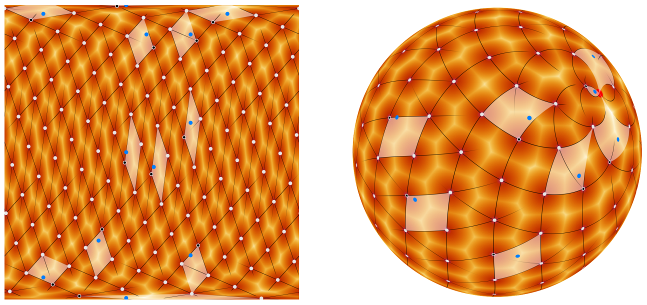


Figure 4: *Visualization of the cells used for the inverse mapping with $n = 128$ SF samples (white dots). The highlighted cells are used for finding nearest neighbors for samples on the sphere (blue dots) in the SF point set. White dots with a black incircle show the found nearest neighbors. The degenerate cells near the North Pole (red) on the sphere still yield the correct nearest neighbor. The color gradient represents the distance to the closest SF point. We refer to the supplemental material for an animation of this figure.*

minimum incircle angle Δ_{min} and the maximum circumcircle angle Δ_{max} – more precisely these bounds are half apex angles of spherical cones – of all spherical Voronoi cells of an SF point set with $n \geq 8$ are given by:

$$\Delta_{min} = \frac{1}{2}\cos^{-1}\langle \mathbf{SF}_3^n, \mathbf{SF}_0^n \rangle, \tag{6}$$

$$\Delta_{max} = \cos^{-1}\left\langle \frac{\mathbf{v}}{\|\mathbf{v}\|_2}, \mathbf{SF}_1^n \right\rangle, \tag{7}$$
$$\text{with } \mathbf{v} = (\mathbf{SF}_2^n - \mathbf{SF}_1^n) \times (\mathbf{SF}_4^n - \mathbf{SF}_1^n).$$

The bound Δ_{max} is generated by the spherical Delaunay triangle with the vertices $\{\mathbf{SF}_1^n, \mathbf{SF}_2^n, \mathbf{SF}_4^n\}$, whereas Δ_{min} is half the geodesic distance between $\mathbf{SF}_0^n$ and $\mathbf{SF}_3^n$. When SF is used for vector quantization, Δ_{max} is the bound for the maximum error (see Section 5.1), whereas Δ_{min} can be used to constrain objects to their respective Voronoi cells when SF is used in the context of procedural modeling (see Section 5.3). Figure 3e depicts the geometric construction of these bounds.

4 Inverse Mapping

In the following, we depict how to map a point $\mathbf{p} = (p_x, p_y, p_z)^{\mathrm{T}} \in \mathcal{S}^2$ to its nearest neighbor in a spherical Fibonacci point set by using the properties of SF grids elucidated in the previous section. Our inverse mapping is based on the observation that we can use the dominant basis vectors $\mathbf{b}_{k'}$ and $\mathbf{b}_{k'+1}$ at the point of interest to retrieve a grid cell that reliably contains the nearest neighbor. Furthermore, the direct mapping between an SF sample's $z_i = \cos(\theta_i)$ and its corresponding index i enables retrieving the actual indices in the SF point set. See Figure 4 for an illustration of the mapping.

Given a point $\mathbf{p} \in \mathcal{S}^2$, we first compute the corresponding integral zone number k' from p_z using Equation 5 as $k' = \max(2, \lfloor \hat{k} \rfloor)$ ($\mathbf{b}_2$ is the first valid dominant basis vector). We construct a matrix $\mathbf{B}_{k'}$ from two consecutive basis vectors $\mathbf{b}_{k'}$ and $\mathbf{b}_{k'+1}$ forming the basis $\mathbf{B}_{k'}$ for the local grid:

$$\mathbf{B}_{k'} = \begin{pmatrix} \mathbf{b}_{k'} & \mathbf{b}_{k'+1} \end{pmatrix}. \tag{8}$$

With $\mathbf{B}_{k'}$, we can transform a point $\mathbf{c} = (c_u, c_v)^{\mathrm{T}}$ on the local grid corresponding to a zone number k' to its associated ϕ and z values as depicted in Equation 9. We add the offset $z_0 = 1 - {}^1\!/_n$ such that the coordinate $\mathbf{c} = (c_u, c_v)^{\mathrm{T}} = (0,0)^{\mathrm{T}}$ on the grid maps to the first point of the spherical Fibonacci point set.

$$\mathbf{C}_{k'}(c_u, c_v) = (\phi, z)^{\mathrm{T}} = \mathbf{B}_{k'}(c_u, c_v)^{\mathrm{T}} + (0, z_0)^{\mathrm{T}} \tag{9}$$

Solving for $\mathbf{c}$ leads to the inverse transformation:

$$\mathbf{C}_{k'}^{-1}(\phi, z) = (c_u, c_v)^{\mathrm{T}} = \mathbf{B}_{k'}^{-1}(\phi, z - z_0)^{\mathrm{T}}. \tag{10}$$

For a given point $\mathbf{p} \in \mathcal{S}^2$, we now compute the lowest coordinate of a corresponding cell on the local grid by rounding down component-wise:

$$\mathbf{c_p} = \lfloor \mathbf{C}_{k'}^{-1}(\operatorname{atan2}(p_y, p_x), p_z) \rfloor. \tag{11}$$

This cell is given by its four corners on the local grid for the zone number k' as: $\{\mathbf{c_p}, \mathbf{c_p} + (0,1)^{\mathrm{T}}, \mathbf{c_p} + (1,0)^{\mathrm{T}}, \mathbf{c_p} + (1,1)^{\mathrm{T}}\}$. To find the closest point in $\mathcal{S}^2$, we compute the point in $\mathcal{S}^2$ for each of the four corners from its spherical coordinates given by Equation 9. In polar regions, corners that do not correspond to points in the SF point set are ignored in further computations. The cell corner yielding the smallest geodesic distance to $\mathbf{p}$ is the nearest neighbor in the SF point set. We retrieve its index i from its z value by solving Equation 2 for i and rounding to the nearest integer:

$$i = \left\lfloor \frac{(1-z)n - 1}{2} + \frac{1}{2} \right\rfloor = \left\lfloor \frac{n - zn}{2} \right\rfloor \tag{12}$$

This computation of the nearest neighbor remains valid, even if the zone number changes within the cell. Figure 4 illustrates that this is true even near at the poles.

Implementation An HLSL implementation of the inverse mapping can be found in Listing 1. The aforementioned algorithm is extremely prone to numerical inaccuracies if implemented with standard floating point arithmetic. Therefore, we will now show how to implement this inverse mapping robustly. Computations involving the golden ratio Φ and powers of Φ in particular are tremendously fragile.

```
#define madfrac(A,B) mad((A),(B),-floor((A)*(B)))

float inverseSF(float3 p, float n) {
   float phi = min(atan2(p.y, p.x), PI), cosTheta = p.z;

   float k = max(2, floor(
      log(n * PI * sqrt(5) * (1 - cosTheta*cosTheta))
         / log(PHI*PHI)));

   float Fk = pow(PHI, k)/sqrt(5);
   float F0 = round(Fk), F1 = round(Fk * PHI);

   float2x2 B = float2x2(
      2*PI*madfrac(F0+1, PHI-1) - 2*PI*(PHI-1),
      2*PI*madfrac(F1+1, PHI-1) - 2*PI*(PHI-1),
      -2*F0/n,
      -2*F1/n);
   float2x2 invB = inverse(B);
   float2 c = floor(mul(invB, float2(phi, cosTheta - (1-1/n))));

   float d = INFINITY, j = 0;
   for (uint s = 0; s < 4; ++s) {
      float cosTheta = dot(B[1], float2(s%2, s/2) + c) + (1-1/n);
      cosTheta = clamp(cosTheta, -1, +1)*2 - cosTheta;

      float i = floor(n*0.5 - cosTheta*n*0.5);
      float phi = 2*PI*madfrac(i, PHI-1);
      cosTheta = 1 - (2*i + 1)*rcp(n);
      float sinTheta = sqrt(1 - cosTheta*cosTheta);

      float3 q = float3(
         cos(phi)*sinTheta,
         sin(phi)*sinTheta,
         cosTheta);

      float squaredDistance = dot(q-p, q-p);
      if (squaredDistance < d) {
         d = squaredDistance;
         j = i;
      }
   }
   return j;
}
```

Listing 1: *An HLSL implementation of our method for finding the index of the nearest point on an SF point set of size n.*

The vector $\mathbf{b}_k$ can be reformulated (see the appendix) such that we can avoid powers of Φ in its direct computation, where we only need a power of Φ to compute the k-th Fibonacci number as $F_k = \lfloor \Phi^k/\sqrt{5} + 1/2 \rfloor$:

$$\mathbf{b}_k = \left(2\pi \left[\frac{F_k + 1}{\Phi} \right] - \frac{2\pi}{\Phi}, \frac{-2F_k}{n} \right)^{\mathrm{T}}. \qquad (13)$$

Since the computation of the angle ϕ is also prone to precision problems for larger n, we only use Equation 9 to recover the z value for each cell corner and then recompute its index i in the SF point set with Equation 12. This rounding step yields the maximum possible precision since we now simply recompute $(\phi_i, z_i)^{\mathrm{T}}$.

Additionally, we leverage the higher precision of fused multiply-adds (FMA/mad), which implement the operation $\mathrm{mad}(a, b, c) = a \cdot b + c$ at a level of precision higher than that of multiplication followed by addition; this would involve two rounding steps instead of one. FMA operations are present in many current architectures, particularly in GPUs. Therefore, we rewrite terms in the form $[a \cdot b]$ as $\mathrm{mad}(a, b, -\lfloor a \cdot b \rfloor)$. For example, the term $\left[i\Phi^{-1} \right]$ is computed as $\mathrm{mad}(i, \Phi - 1, -\lfloor i(\Phi - 1) \rfloor)$ using the property $\Phi - 1 = \Phi^{-1}$.

Furthermore, we avoid the rejection of invalid neighbor candidates (i.e. $|z| > 1$) lying outside the SF point set in the polar regions by mirroring the cell corners' z values at the polar axis to valid samples. This step is not required to retrieve the correct nearest neighbor, but simplifies the implementation. Instead of measuring geodesic distances on the sphere, we use squared euclidean distances, yielding the same nearest neighbors while avoiding precision problems.

Our implementation relies on the $\min(a, b)$ and $\max(a, b)$ implementations to return the non-NaN-valued parameter, if one of both parameters is NaN (not a number). Additionally, we assume $\mathrm{atan2}(0, 0) = \mathrm{NaN}$. These requirements hold true for current DirectX/HLSL implementations. Otherwise, points lying exactly on the poles must be addressed separately. In our experiments, the GPU implementation yields correct results for arbitrary $n \leq 2^{21}$. From that point on, it works for power of two values up to $n = 2^{23}$. The underlying cause for this is the limited precision of 32-bit floating point values.

5 Applications

5.1 Unit Vector Quantization

Since memory bandwidth is a bottleneck in rendering applications, e.g. in the context of g-buffer compression, unit vector quantization can be used for normal compression. [Meyer et al. 2010] showed that the common representation of a three-dimensional unit vector with $3 \cdot 32 = 96$ bits is highly redundant, because only 51 bits are required to achieve the maximum possible floating point precision. For a survey on unit vector quantization see [Cigolle et al. 2014].

In general, there are two objectives for any quantization scheme: precision and performance. To minimize error and thus maximize precision, a uniform distribution of the representatives is necessary. Although there are methods that aim to minimize the global quantization error through lookup tables, e.g. [Smith et al. 2012], our approach enables *constant time* encoding and decoding without possibly huge lookup tables or heavy precomputations. Encoding with SF works as follows: Given a vector $\mathbf{p} \in \mathcal{S}^2$ and the number of representatives n used for encoding, we apply the inverse mapping (Section 4) and obtain an index $i \in \{0, \ldots, n-1\}$. To decode a given index i to a unit vector $\mathbf{p} \in \mathcal{S}^2$, the forward SF mapping has to be applied (Section 3).

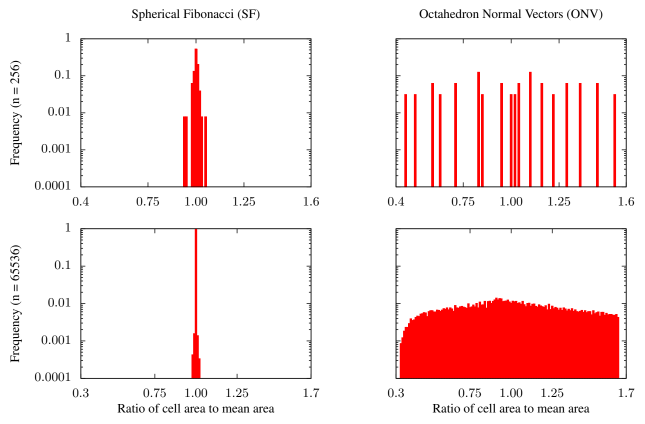


Figure 5: *Histogram of cell areas of SF and ONV for $n = 256$ (top) and $n = 65536$ (bottom) samples. Note the logarithmic scale of the frequency axis.*

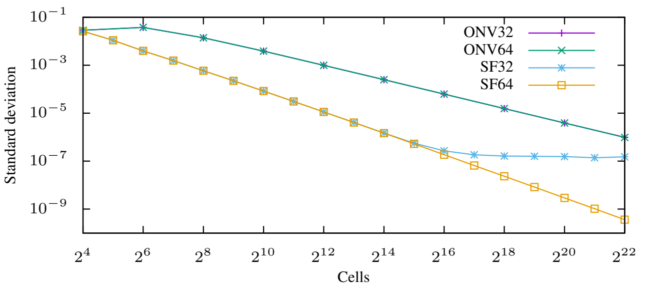


Figure 6: *Standard deviation of the cell areas for SF and ONVs for an increasing number of cells. Both variants are implemented for single (32-bit) and double (64-bit, CPU) precision. Note that both axes use logarithmic scale.*

We evaluate our quantization in two different ways and compare the results against the state-of-the-art Octahedron Normal Vectors (ONV) as presented by [Meyer et al. 2010]. For comparison, we use the more precise ONV implementations, which work in the range. HSF and HONV denote the algorithms for hemispheres. The first part of our evaluation measures the uniformity of the point distribution. We calculated the (spherical) Voronoi cells V_i for a given set of samples $\mathbf{S}_i$, where $\mathbf{S}_i$ is the Voronoi site of the cell $V_i, i \in \{0, \ldots, n-1\}$ and $d(\cdot, \cdot)$ is the geodesic distance:

$$V_i = \{\mathbf{p} \in \mathcal{S}^2 \colon d(\mathbf{p}, \mathbf{S}_i) \leq d(\mathbf{p}, \mathbf{S}_j), \forall i \neq j\}.$$

Ideally, the spherical area A_i of all Voronoi cells V_i should be equal: $A_i = {}^{4\pi}/_{n}$. In Figure 5, the ratios of the cell area to the mean area for $n = 256$ and $n = 65536$ are shown and compared to the ONV quantization. As can be seen, the SF point sets yield a significantly higher frequency of cells that have the average cell area. Also, the minimum and maximum ratio are considerably closer to 1 (minimum: 0.94, maximum: 1.05), whereas the ONVs generate a distribution in the range from 0.4 to 1.6 and 0.3 to 1.7, respectively.

To compare the uniformity against ONV for an increasing number of representatives, we evaluate the standard deviation σ of the cell area (see Figure 6): $\sigma = {}^{1}/_{n} \sum_i (A_i - {}^{4\pi}/_{n})^2$. The SF quantization outperforms ONV for any number of bits in terms of uniformness. Another interesting finding is the fact that, starting from $n \approx 32768$ samples, the maximum possible floating point precision is apparently attained, as the 64-bit variant still yields a decreasing standard deviation. The algorithm with double precision (64-bit) was implemented on the CPU with analogous instructions,

Quantization	Root-mean-square error (RMSE) (°)										Runtime (ns)	
	4	6	8	10	12	14	16	18	20	22	Encode	Decode
ONV	7.58	3.86	1.95	0.978	0.489	0.244	0.122	0.0611	0.0305	0.0153	**0.0998**	0.0171
SF (Ours)	**7.46**	**3.62**	**1.80**	**0.889**	**0.448**	**0.222**	**0.112**	**0.0555**	**0.0280**	**0.0143**	0.1718	**0.0143**
HONV	6.25	3.28	1.67	0.837	0.419	0.209	0.1050	0.0524	0.0262	0.0131	**0.0901**	0.0160
HSF (Ours)	**5.80**	**2.73**	**1.31**	**0.642**	**0.318**	**0.158**	**0.0788**	**0.0396**	**0.0194**	**0.0105**	0.1732	**0.0149**

Table 1: *A comparison of our SF quantization against Octahedron Normal Vectors in terms of root-mean-square error (RMSE) and runtime per unit vector. We achieve a lower RMSE in every case and an even better runtime for decoding. The encoding performance is slightly worse as more instructions are necessary. The runtime was measured on an NVIDIA GTX 760.*

e.g. fused multiply-add. The second part of the evaluation measures the quantization error and the runtime for SF point sets as well as ONV. To this end, we generated 2^{26} points with a uniform random distribution on a sphere and calculated error measurements, i.e. the maximum and the root-mean-square error. Our worst case error is almost equal to the other algorithms for every number of bits and matches the theoretical bound Δ_{max}, which only occurs near the poles (see Section 3). Furthermore, the root-mean-square error (RMSE) is lower in all cases (see Table 1). Our measurements of every ONV variant fit very well to those of [Cigolle et al. 2014]. Following the approach of these authors, the performance of our algorithm is also evaluated in nanoseconds per unit vector (measured on an NVIDIA GTX 760) and the stated times include the parallelism of the GPU.

Due to the increased instruction count for encoding (see Listing 1), the runtime is slightly higher than ONV (by a factor of $\approx 1.7-1.9$). However, our decoding performance is faster than ONV (by a factor of $\approx 1.1 - 1.2$). This is of particular interest in the case of normal maps, where encoding is typically done in an offline preprocessing step and only decoding occurs during rendering.

5.2 Textures and Filtering

The properties shown in the previous sections make SF points sets well suited for storing spherical signals. Our inverse mapping and its variants can easily be used to conduct texture lookups similar to those implemented with cube mapping. For storage, we simply store a color value, for instance, in a buffer for each index i.

As shown in the previous section, SF point sets easily outperform octahedral parameterizations on the sphere at the same sample rate. Therefore, SF point sets readily outperform Octahedron Environment Maps (OEM) [Engelhardt and Dachsbacher 2008] in sampling quality. Since these yield a slightly better distribution of samples on the sphere than cube maps [Greene 1986], SF texture representations also outperform cube maps in terms of sampling quality.

It is also possible to implement mipmapping using the lookup methods of the next sections on two buffers with different SF point set resolutions (or one, which contains multiple resolutions). The results of the lookups get then linearly interpolated.

5.2.1 Nearest Neighbor Filtering

Nearest neighbor filtering is straightforward to implement when using our inverse mapping. For a given direction vector, the inverse mapping is used to retrieve the index of the corresponding sample on the sphere, after which we then perform a buffer lookup with this index to fetch the assigned values.

5.2.2 Smooth Filtering

Since nearest neighbor filtering is not suited for many applications, we extend our mapping to fetch and filter samples from a larger neighborhood. We apply the same mapping strategy as described

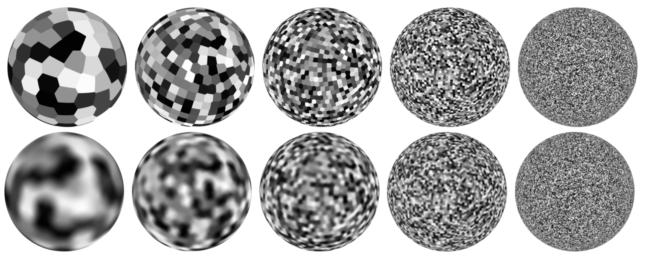


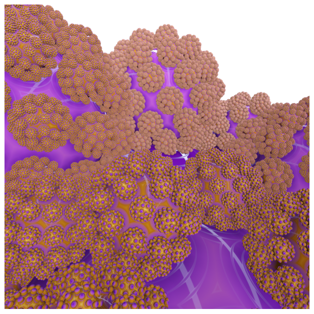
Figure 7: *SF texture mapping examples: Nearest neighbor filtering (top row) and smooth filtering (bottom row) filtering for 128, 512, 2048, 8192, and 65536 samples.*

in Section 4, but instead of rounding down, we round to the closest local grid coordinate and sample over a 3×3 neighborhood $\{\mathbf{c_p} + (i,j)^{\mathrm{T}}, \quad i,j \in \{-1,0,1\}\}$. Unfortunately, for regions around the poles ($k' \leq 4$) the sampled neighborhoods become too distorted to recover all samples for filtering properly. In case a lookup coordinate $\mathbf{p}$ falls into these polar caps, we sample over 16 samples that affect these regions directly, enabling smooth texture lookups on the entire spherical surface. Note that for larger n the polar regions become relatively small and thus only affect the overall performance of the technique to a minor extent.

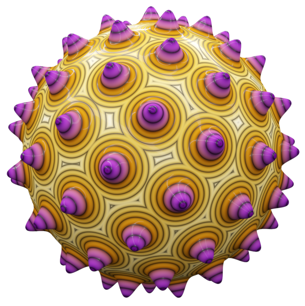
For filtering, we use the weighted mean of all samples by employing a *smoothstep*-based weighting scheme. As an approximation of the geodesic distances, we use the euclidean distance δ_s between the lookup position $\mathbf{p}$ and each sample's position. For the radius h of the smoothing kernel we choose $h^2 = {}^{4\pi}/_{n}$, which is the average spherical Voronoi cell area for n samples. This yields the sample's weights $\omega_s = 3t^2 - 2t^3$ with $t = \max(0, 1 - {}^{\delta_s}/_{h})$. Figure 7 shows a comparison of our filtering techniques at varying sample resolutions. We measured their performance by conducting 2^{27} lookups in random directions on a $512 \times 512 \times 6$ cube map and an SF points set with the same number of samples. While the cube map performed lookups with 0.022 ns/sample, our SF techniques were significantly slower, with 0.1863 ns/sample and 0.6817 ns/sample for the nearest and smooth filtering approaches, respectively. This huge performance difference can simply be explained by the GPU's hardware acceleration for the cube map lookups. Additionally, SF based sampling is not very cache efficient by construction, since spatially close samples partially reside in distant memory locations.

5.3 Procedural Modeling

In addition to the apparent applications for vector quantization and texture lookups for spherical signals, our inverse mapping can also be employed in procedural modeling. When applied to implicit function representations, our inverse mapping can, for instance, be used to place objects along spherical surfaces. It also can be used to generate implicitly defined displacements along SF grids as shown in Figure 8.



(a)



(b)

Figure 8: *(a) Sphereflake-like fractal with four recursion levels, resulting in a total of* $2^{24}+1$ *spheres using 256 SF samples. (b) A sphere surface with implicit displacement mapping using the inverse mapping (*$n = 128$ *SF samples).*

The bound Δ_{min} (see Section 3) is a great convenience in this context. The maximum size of objects distributed along an SF point set can be derived from it, such that they fit into all SF Voronoi cells. Both images are rendered using sphere tracing (see [Hart 1996] and [Keinert et al. 2014]).

The SF sphereflake object (Figure 8a) can easily be implemented as an implicit function $f : \mathbb{R}^3 \to \mathbb{R}$ with a single loop over all recursive levels. At each level, we compute the distance to a single sphere. In case the maximum depth is not reached, we build an orthonormal basis around the vector corresponding to the closest SF sample for the sphere of the respective level. After that, we transform the space – using the orthonormal basis – such that the sphere of the next level lies on the sphere of the current recursive level. Computing the minimum of the distances to the spheres of all levels yields an implicit function with the surface $f^{-1}(\{0\})$, which can be rendered using sphere tracing.

The sphere with SF displacements (Figure 8b) is also represented as an implicit function, where we simply add procedural displacements based on the geodesic distance to the nearest SF sample. Given that our inverse mapping runs in constant time, *arbitrary* numbers of object placements and surface modifications can be applied without increasing the cost for one single evaluation of the implicit function.

6 Limitations

While our proposed methods perform well, they exhibit a few limitations – on the one hand caused by the properties of SF point sets and on the other by implementation issues.

The number of addressable points of our implementation is limited by the available floating point precision. Therefore, our 32-bit GPU implementation can handle a maximum of 2^{23} samples, where the number of samples n can only be chosen arbitrarily (i.e. non power of two) for $n \leq 2^{21}$. While a double precision variant improves upon these issues, the performance impact for a GPU implementation is significant. Alternatively, a fixed-point arithmetic implementation of our method should be capable of addressing a larger number of SF points, while still being able to use 32-bit arithmetic.

The proposed implementation for the inverse mapping is rather computationally intense. Nevertheless, the performance can be significantly improved by omitting the recomputations (Section 4, Implementation) in the innermost loop, if n is known to be smaller ($n \leq 2^{16}$). Graphics hardware tends to provide more and more compute performance, whereas memory throughput/latency remains the major bottleneck. Additionally, latency hiding techniques can readily be applied since the inverse mapping does not require any memory operations at all.

The application of our inverse mapping for texture filtering (Section 5.2) is currently problematic performance-wise, since no direct hardware support exists and incoherent memory accesses have to be conducted. A mapping based on space-filling curves (e.g. Morton order) could be applied to the local grids to improve upon this issue. We consider this an interesting direction for future research.

7 Conclusion

In this paper, we showed an analytic constant time inverse mapping for SF point sets. Further, we addressed typical implementation issues and presented an efficient GPU implementation. We demonstrated the use of this inverse mapping and its variants in various example applications.

Even though spherical Fibonacci point sets are not the globally best distribution of samples on a sphere, they yield excellent sampling properties and are extremely simple to construct in contrast to other more sophisticated spherical sampling schemes. The presented fast inverse mapping from points on the unit sphere to SF point sets enables a very broad range of applications.

Spherical Fibonacci point sets should be easily applicable to spatial hashing approaches for spherical data, where the lack of memory coherence – in contrast to texture storage – would be less relevant. Furthermore, SF point sets can be used to place icons in virtual reality environments on a sphere around the user. Both, efficient icon placement and picking are easy to implement with SF point sets and our inverse mapping, respectively.

Acknowledgements

We thank Urs Ganse for the inspiring correspondence and his helpful comments in preparing this manuscript. We would further like to thank the authors of [Cigolle et al. 2014] for providing the source code of their measurement framework. This research was supported by the German Research Foundation (DFG), grant GRK-1773 Heterogeneous Image Systems.

References

BROCKMEYER, E., POUPYREV, I., AND HUDSON, S. 2013. Papillon: Designing curved display surfaces with printed optics. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, ACM, 457–462.

CIGOLLE, Z. H., DONOW, S., EVANGELAKOS, D., MARA, M., MCGUIRE, M., AND MEYER, Q. 2014. A survey of efficient representations for independent unit vectors. *Journal of Computer Graphics Techniques (JCGT) 3*, 2 (April), 1–30.

DAMMERTZ, S., AND KELLER, A. 2008. Image synthesis by rank-1 lattices. In *Monte Carlo and Quasi-Monte Carlo Methods 2006*. Springer, 217–236.

ENGELHARDT, T., AND DACHSBACHER, C. 2008. Octahedron environment maps. In *Proceedings of the Vision, Modeling, and Visualization Conference 2008, VMV 2008, Konstanz, Germany, October 8-10, 2008*, 383–388.

FOWLER, D. R., PRUSINKIEWICZ, P., AND BATTJES, J. 1992. A collision-based model of spiral phyllotaxis. In *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New York, NY, USA, SIGGRAPH '92.

GONZÁLEZ, Á. 2010. Measurement of Areas on a Sphere Using Fibonacci and Latitude-Longitude Lattices. *Mathematical Geosciences 42*, 1, 49–64.

GREENE, N. 1986. Environment mapping and other applications of world projections. *Computer Graphics and Applications, IEEE 6*, 11 (Nov), 21–29.

HANNAY, J. H., AND NYE, J. F. 2004. Fibonacci numerical integration on a sphere. *Journal of Physics A: Mathematical and General 37*, 48.

HART, J. C. 1996. Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer 12*, 10, 527–545.

KEINERT, B., SCHÄFER, H., KORNDÖRFER, J., GANSE, U., AND STAMMINGER, M. 2014. Enhanced sphere tracing. In *Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference*, Eurographics Association, Cagliari, Italy, 1–8.

LARKINS, R. L., CREE, M. J., AND DORRINGTON, A. A. 2012. Analysis of binning of normals for spherical harmonic cross-correlation. In *IS&T/SPIE Electronic Imaging*, International Society for Optics and Photonics.

MARQUES, R., BOUVILLE, C., RIBARDIÈRE, M., SANTOS, L. P., AND BOUATOUCH, K. 2013. Spherical Fibonacci Point Sets for Illumination Integrals. In *Computer Graphics Forum*, vol. 32, Wiley, 134–143.

MEYER, Q., SÜSSMUTH, J., SUSSNER, G., STAMMINGER, M., AND GREINER, G. 2010. On floating-point normal vectors. *Computer Graphics Forum 29*, 4, 1405–1409.

NEWELL, A., AND SHIPMAN, P. 2005. Plants and fibonacci. *Journal of Statistical Physics 121*, 5-6, 937–968.

SMITH, J., PETROVA, G., AND SCHAEFER, S. 2012. Encoding Normal Vectors using Optimized Spherical Coordinates. *Computers & Graphics 36*, 5.

SWINBANK, R., AND PURSER, R. J. 2006. Fibonacci grids: A novel approach to global modelling. *Quarterly Journal of the Royal Meteorological Society 132*, 619, 1769–1793.

VOGEL, H. 1979. A better way to construct the sunflower head. *Mathematical Biosciences 44*, 179–189.

Appendix

Derivation of Equation 5

Given Equation 4, with $\sin\theta = \sqrt{1-z^2}$, the objective function to minimize is:

$$f(k) = \left\| \left(-(-1)^k 2\pi\Phi^{-k}\sin\theta,\ \frac{-2F_k}{n\sin\theta} \right)^{\mathrm{T}} \right\|_2^2 = \\ = (-1)^{2k}4\pi^2\Phi^{-2k}\sin^2\theta + \frac{4F_k^2}{n^2\sin^2\theta}.$$

We then apply the approximation $F_k \approx \frac{\Phi^k}{\sqrt{5}}$:

$$f(k) \approx \hat{f}(k) = 4\pi^2\Phi^{-2k}\sin^2\theta + \frac{4\Phi^{2k}}{5n^2\sin^2\theta},$$

The derivative of $\hat{f}$ with respect to k is given by:

$$\hat{f}'(k) = -8\pi^2\sin^2(\theta)\ln(\Phi)\cdot\Phi^{-2k} + \frac{8\ln\Phi}{5n^2\sin^2\theta}\cdot\Phi^{2k}.$$

Solving for the zero-crossing $\hat{f}'(k) \overset{!}{=} 0$ yields:

$$\begin{aligned} \frac{8\ln\Phi}{5n^2\sin^2\theta}\cdot\Phi^{2k} &= 8\pi^2\sin^2(\theta)\ln(\Phi)\cdot\Phi^{-2k} \\ \Leftrightarrow \Phi^{4k} &= 5n^2\pi^2\sin^4(\theta) \\ \Rightarrow (\Phi^2)^k &= \sqrt{5}n\pi\sin^2(\theta) \\ \Leftrightarrow k = \log_{\Phi^2}(\sqrt{5}n\pi\sin^2(\theta)) &= \log_{\Phi^2}(\sqrt{5}n\pi(1-z^2)). \end{aligned}$$

Derivation of Equation 13

In the following, we will show how to deduce Equation 13 from Equation 3. In the following we assume $k \in \mathbb{N}, k \geq 2$. Since the second component is identical we only consider the first component of both equations:

$$\underbrace{-(-1)^k 2\pi\Phi^{-k}}_{\text{Equation 3}} = \underbrace{2\pi\left[\frac{F_k+1}{\Phi}\right] - \frac{2\pi}{\Phi}}_{\text{Equation 13}}.$$

Simplification and stripping the factor of 2π on both sides yields:

$$-\left(-\frac{1}{\Phi}\right)^k = \left[\frac{F_k+1}{\Phi}\right] - \frac{1}{\Phi}.$$

The absolute value of the term $-(\,{-}\,{}^1/_\Phi)^k$ decreases strict monotonically for increasing k, as $\frac{1}{\Phi} \approx 0.618 \in (0,1)$. Given that $k \geq 2$, the lower and the upper bound of its value are given as:

$$\underbrace{-\frac{1}{\Phi^2}}_{\approx -0.382} \leq -\left(-\frac{1}{\Phi}\right)^k \leq \underbrace{\frac{1}{\Phi^3}}_{\approx 0.236}.$$

Applying a shift of ${}^1/_\Phi$, yields:

$$\underbrace{\frac{1}{\Phi} - \frac{1}{\Phi^2}}_{\approx 0.236} \leq \frac{1}{\Phi} - \left(-\frac{1}{\Phi}\right)^k \leq \underbrace{\frac{1}{\Phi} + \frac{1}{\Phi^3}}_{\approx 0.854}.$$

We then use the invariance property for the interval $[0,1)$ of the fractional part operator, i.e. $[x] = x - \lfloor x \rfloor = x$ for $x \in [0,1)$:

$$\begin{aligned} \frac{1}{\Phi} - \left(-\frac{1}{\Phi}\right)^k &= \left[\frac{1}{\Phi} - \left(-\frac{1}{\Phi}\right)^k\right] \\ \Leftrightarrow -\left(-\frac{1}{\Phi}\right)^k &= \left[\frac{1}{\Phi} - \left(-\frac{1}{\Phi}\right)^k\right] - \frac{1}{\Phi}. \end{aligned}$$

Applying the identity $\left(-\frac{1}{\Phi}\right)^k = -\frac{1}{\Phi}F_k + F_{k-1}$ (decomposition of powers of the golden ratio), yields:

$$\begin{aligned} -\left(-\frac{1}{\Phi}\right)^k &= \left[\frac{1}{\Phi} - \left(-\frac{1}{\Phi}\right)^k\right] - \frac{1}{\Phi} = \\ &= \left[\frac{1}{\Phi} - \left(-\frac{1}{\Phi}F_k + F_{k-1}\right)\right] - \frac{1}{\Phi} = \\ &= \left[\frac{1}{\Phi} + \frac{1}{\Phi}F_k\right] - \frac{1}{\Phi} = \left[\frac{F_k+1}{\Phi}\right] - \frac{1}{\Phi}. \end{aligned}$$

The reintroduction of the factor 2π and slight reformulation leads to the equality from the beginning of our derivation:

$$-(-1)^k 2\pi\Phi^{-k} = 2\pi\left[\frac{F_k+1}{\Phi}\right] - \frac{2\pi}{\Phi}.$$