Journal of Computer Graphics Techniques Vol. 7, No. 2, 2018 http://jcgt.org

Cube-to-sphere Projections for Procedural Texturing and Beyond

Matt Zucker and Yosuke Higashi
Swarthmore College

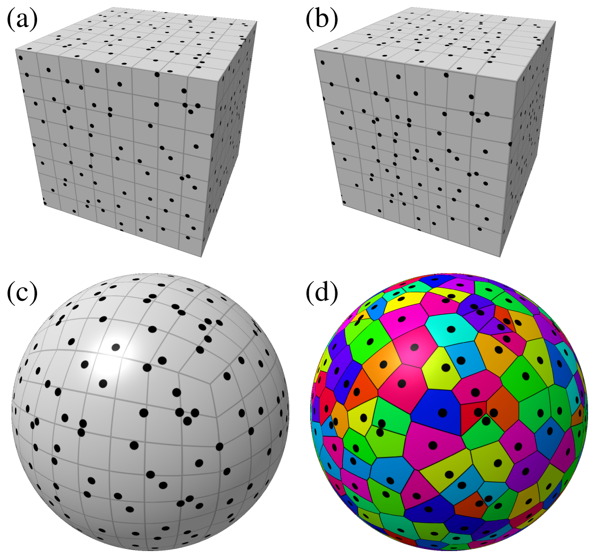

Figure 1. Real-time shader illustrating stratified sampling of the sphere. (a) jittered uniform grid on cube; (b) nonlinear warp of each face; (c) projection to sphere via normalization; (d) resulting Voronoi diagram, generated in constant time per fragment.

Abstract

Motivated by efficient GPU procedural texturing of the sphere, we describe several approximately equal-area cube-to-sphere projections. The projections not only provide low-distortion UV mapping, but also enable efficient generation of jittered point sets with $O(1)$ nearest-neighbor lookup. We provide GLSL implementations of the projections with several real-time procedural texturing examples. Our numerical results summarize the various methods' ability to preserve projected areas as well as their performance on both integrated and discrete GPUs. More broadly, the overall cube-to-sphere approach provides an underexplored avenue for adopting existing 2D grid-based methods to the sphere. As an example, we showcase fast Poisson disk sampling.

1. Introduction

Spheres are ubiquitous primitives in computer graphics, used to render everything from molecules to planets. Sampling and tessellating the sphere is fundamental to texturing and modeling these common objects. Most users of modern graphics APIs are familiar with cube mapping [Greene 1986], a technique that is commonly used to map between 3D direction vectors and 2D textures tiling the surface of a cube. In this paper, we present a family of invertible mappings that can be considered as a generalization of cube mapping. When constructed to preserve relative area, these mappings are useful for sampling the sphere and have applications to procedural texturing and modeling.

1.1. Related Work

Like the spherical Fibonacci (SF) mapping of Keinert et al. [2015], the methods presented here can be used to generate well-dispersed sets of points on the sphere with efficient nearest-neighbor lookup, suitable for many Voronoi-based texturing techniques. However, compared with SF mapping, graphics programmers and artists may prefer to use the *sphered-cube* approach we describe, because it allows many existing techniques developed for 2D Cartesian grids to be adapted to the sphere.

Although our primary motivation is GPU-based procedural texturing of spheres, the methods we study are closely related to sampling methods used for rendering, especially determining illumination. Arvo [1995] demonstrated an approach for stratified sampling of spherical triangles and general 2-manifolds [2001]. Others have used cylindrical projection to create equal-area maps from plane to sphere [Shao and Badler 1996; Wong et al. 1997]. However, the resulting singularities prevent efficient nearest-neighbor search (Section 4.2) on jittered grids. We show that the sphered-cube approach can be used to generate blue-noise samples on the sphere, which can lead to higher-quality Monte Carlo estimates of spherical integrals [Singh 2015].

In contrast to other 2D representations of the sphere, such as octahedron normal vectors [Meyer et al. 2010] or HEALPix [Gorski et al. 2005], the chief advantage of the approach we present is that it allows straightforward application of techniques designed for 2D Cartesian grids. In general, our analysis comes at a time when the computer graphics community is showing a renewed interest in the mathematics of map projections [Lambers 2016] and novel applications of re-parameterizing the sphere [Heitz et al. 2016].

The area-preserving cube-to-sphere projections we describe have been of particular interest to other researchers in recent years [Everitt 2016; Brown 2017]. Although this approach dates back over four decades [Chan and O'Neill 1975], it nonetheless remains an underexplored method in computer graphics for adapting a broader range of grid-based techniques to spherical domains. Beyond the in-depth analysis and quantitative comparison we provide, our main contributions are to optimize free parameters for previously published projections that had been chosen *ad hoc* and to increase the visibility of the overall cube-to-sphere approach. We also introduce a new method: the 1D odd-polynomial projection.

1.2. Organization of this Article

The remainder of this paper is structured as follows: we begin in Section 2 by reviewing several familiar procedural texturing primitives along with requirements for their implementation on spherical domains. Sections 3 and 4 derive the projections that enable the sphered-cube approach and demonstrate how it satisfies these requirements. Section 5 provides a thorough comparison of the mappings we present, as well as our recommendations on their suggested use. Finally, Section 6 showcases fast generation of spherical Poisson disk (blue noise) point sets as a concrete example of adapting other existing 2D grid-based methods using the sphered-cube approach.

2. GPU-based Procedural Texturing of the Sphere

In this section, we summarize several familiar 2D procedural texturing primitives and discuss both speed and quality requirements for implementing these primitives on spherical domains using the GPU. Subsequent sections will construct our sphered-cube approach to meet these requirements.

2.1. Procedural Texturing Primitives and Their Rquirements

Perhaps the most well-known procedural texturing primitive is *value noise*, where continuous intensities are randomly assigned to points on a regular lattice. Nearby intensities are computed by applying a reconstruction kernel to surrounding grid points, typically a cubic Hermite spline or similar function. Whereas value noise utilizes a regular lattice of points, *cellular texturing* (a.k.a. Worley [1996] noise) uses points sampled from an irregular distribution. Cellular texturing may simply assign every texel the random color of its nearest sample point, or smoothly modulate colors based upon the distances to the closest k points. Recently, Quílez [2015] proposed a generalization of both primitives dubbed *Voronoise*. Regardless of the underlying domain—2D plane, sphere, or other smooth manifold—all three procedural texturing primitives we just described collectively share some fundamental requirements:

(i) Equidistributed sample points – the implementation must be able to rapidly generate discrete sets of evenly spaced sample points of a desired density.

(ii) Jittered point sets – the implementation must be able to generate visually non-uniform samples, with neither dense clumps nor large empty regions. This can be accomplished via *stratified sampling* (a.k.a. *jittered grids*) in which the domain is partitioned into regular subdomains of equal area which are then sampled uniformly.

(iii) Efficient nearest sample search – the implementation must be able to quickly determine the closest k sample points to any arbitrary point in the domain, irrespective of the total number of sample points.

Of these, value noise requires (i) and (iii), and the other primitives require (ii) and (iii).

2.2. Spherical Domains & Comparisons to Previous Approaches

For 2D planar domains, the requirements listed above are typically met by using uniform or jittered grids. The methods we describe in Sections 3 and 4 facilitate their use on the sphere while avoiding artifacts and other shortcomings of previous approaches.

Keinert et al. [2015] show how to implement value noise with their *spherical Fibonacci* mapping. Their approach requires special-case code to identify the set of bracketing points at the poles, and the resulting texture is anisotropic in these regions.

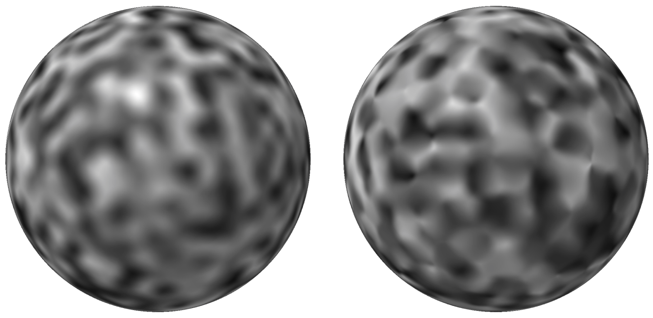

Figure 2. Real-time fragment shader generating value noise on both a uniform (left) and jittered (right) grid with $m = 16$ points per cube face edge.

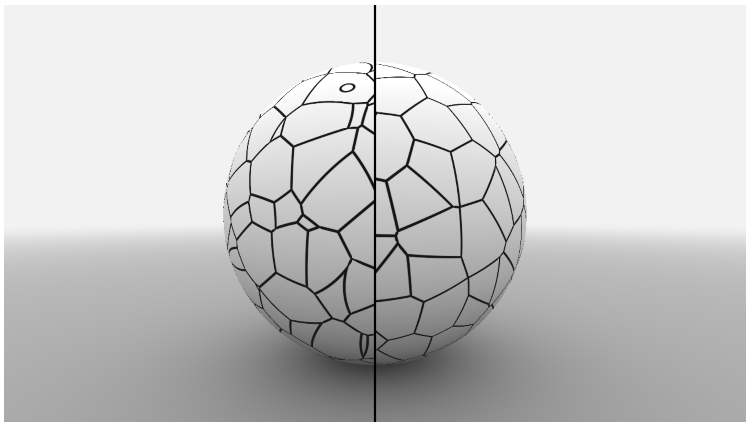

Figure 3. Real-time fragment shader comparing two types of Voronoi cells. Left side: Points taken from a jittered 3D grid. Note apparent curvature of cell borders as well as high variation in region sizes on the sphere. Right side: Points from jittered 2D grid on the sphered cube. 3D grid based on MIT-licensed code © Íñigo Quílez [2013].

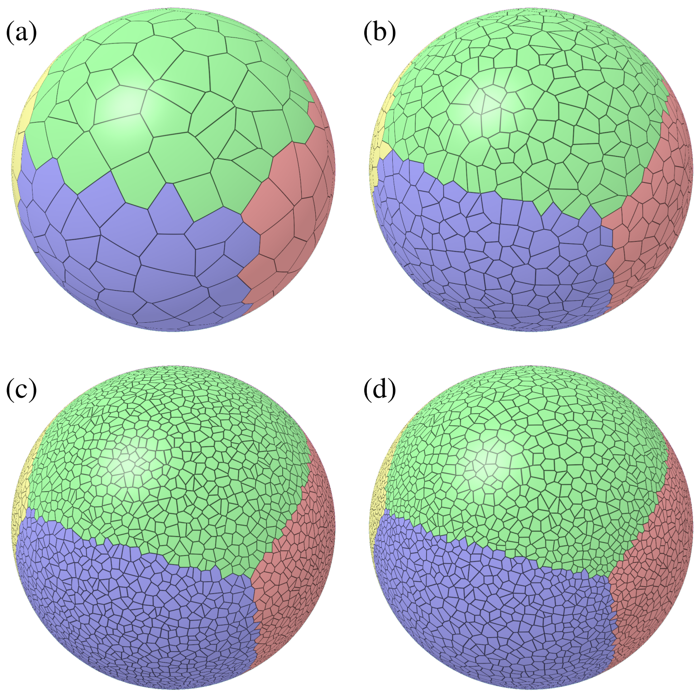

Figure 4. Voronoi diagrams of randomized point sets, colored by originating cube face. From top left: (a)–(c) Varying $m = 8$, 16, and 32 points per cube edge, mapped through tangent warp (Section 3.2); (d) Same as (c), but using the identity warp (Section 3.1). Using this non-equal-area projection, Voronoi regions shrink near cube edges and grow near face centers.

Our sphered-cube approach uses the same grid-traversal code (Listing 3) everywhere and the resulting value noise appears visually homogeneous, as shown in Figure 2. In further contrast to spherical Fibonacci mapping, we can straightforwardly create jittered grids, allowing us to implement cellular texturing and Voronoise as well.

One popular approach for texturing the surface of an arbitrary solid object with cellular noise is to generate a jittered grid on a 3D lattice and compute distances from each point on the surface to the k-nearest 3D sample points. However, as shown in Figure 3, this approach has two main drawbacks, especially when used to texture spheres. Although the Voronoi boundaries of a 3D jittered grid are piecewise planar, their intersections with the sphere do not follow geodesics along its surface, inducing undesirable extraneous curvature with respect to the local tangent frame. Furthermore, the intersections of the 3D Voronoi cells with the sphere exhibit substantial size variation as the corresponding sample points vary in distance to the sphere's surface. Using the sphered cube mitigates both of these effects, and the underlying cube structure is invisible as long as the cube-to-sphere projection is approximately area-preserving (see Figure 4 for an example of the converse).

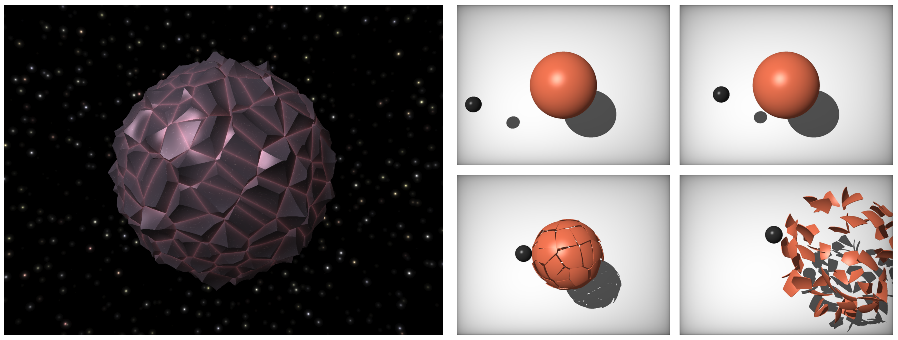

Figure 5. Example scenes. Left: Planetoid with height-mapped cellular noise among a jittered grid of stars on a skydome. Right: Simulating brittle fracture along Voronoi boundaries.

In summary, spherical Fibonacci mapping satisfies requirements (i) and (iii) but not (ii). The popular 3D-to-2D approach for cellular texturing satisfies (ii) to some extent, but suffers visually. In contrast, the sphered-cube approach we describe in the following sections satisfies all three requirements. Furthermore, the underlying techniques provide a low-distortion UV mapping which is also useful for traditional (i.e., non-procedural) texture mapping.

We conclude this section with two demonstrations of the utility of the sphered-cube approach in procedural texturing and modeling. The left-hand scene in Figure 5 is rendered using real-time ray marching (a.k.a. sphere tracing), and the right-hand scene is rendered using real-time ray tracing. Both scenes are rendered at interactive rates by pure fragment shaders drawing a single full-screen quad. The GLSL source for both shaders can be found in the supplemental materials.

3. Projections from Cube to Sphere

In this section, we present a taxonomy of several cube-to-sphere projections. All of the approaches described here map a point $\vec{p}$ on the surface of a cube with vertices at ± 1 to a vector $\hat{\omega}$ on the surface of a unit sphere. Our goal is to construct mappings that are *area-preserving*; that is, equal-sized subregions of cube faces are mapped to equal areas on the sphere. This is desirable in many graphics applications, for example stratified sampling [Arvo 2001] or panoramic VR video [Brown 2017]. In our procedural texturing application, this property provides an even distribution of sample points over the sphere, as illustrated in Figure 4.

There are several ways to parameterize the 2D surface of the cube. In this work, we choose to separate the discrete and continuous parts of the representation. We represent the former—the cube face and its local frame—as a signed 3×3 permutation matrix $\mathbf{P}$ whose elements are zero or ± 1. We can then represent the continuous

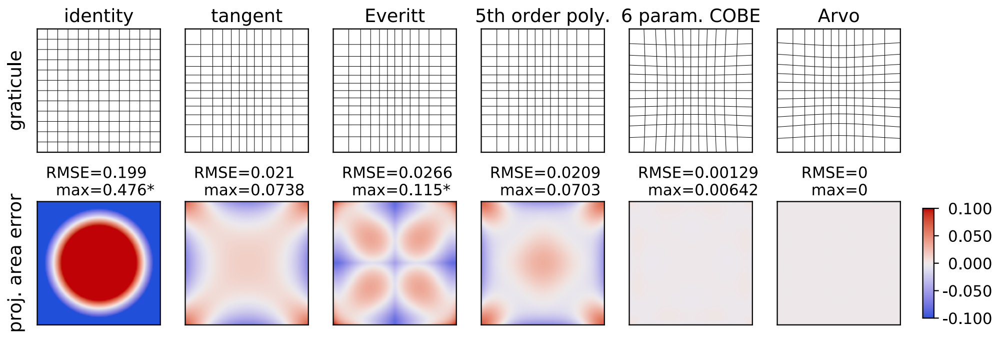

Figure 6. Illustration of the approaches described in Sections 3.1 to 3.6. The top row shows the warped grid for each method; the bottom shows projected area error (E_A in Equation (8)). Areas shaded in red are enlarged by projection onto the sphere, blue areas are reduced. *Asterisks above indicate that the maximum area error exceeds the color-map limits.

coordinates within the cube face as a vector $(a, b) \in [-1, 1] \times [-1, 1]$ such that

$$\vec{p} = \mathbf{P} \begin{bmatrix} a \\ b \\ 1 \end{bmatrix}. \tag{1}$$

For example, the point $\vec{p} = (-0.3, -1, 0.2)^T$ could be represented as

$$\mathbf{P} = \begin{bmatrix} 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \end{bmatrix}, \quad \text{with} \quad \begin{aligned} a &= 0.2, \\ b &= 0.3. \end{aligned}$$

Given an original pair of cube-face coordinates (a, b), each approach we describe below computes a new set of coordinates (u, v) with $u = f_u(a, b)$ and $v = f_v(a, b)$. The *warp functions* $f_u, f_v : \mathbb{R}^2 \mapsto \mathbb{R}$ distort 2D grid points on a cube face prior to normalization onto the sphere, as shown in the top row of Figure 6. Although each cube face undergoes distortion, no point ever moves to a new cube face. Hence, we can reuse **P** to represent the warped location $\vec{q}$ of point $\vec{p}$:

$$\vec{q} = \mathbf{P} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \mathbf{P} \begin{bmatrix} f_u(a, b) \\ f_v(a, b) \\ 1 \end{bmatrix}. \tag{2}$$

Finally, regardless of which pair of warp functions is used, the unit vector $\hat{\omega}$ on the sphere is obtained by simply normalizing $\vec{q}$:

$$\hat{\omega} = \frac{\vec{q}}{\|\vec{q}\|}. \tag{3}$$

The projections all have some properties in common. Each method exhibits horizontal and vertical symmetry with $f_u(-a, b) = -f_u(a, b)$ and $f_v(a, -b) = -f_v(a, b)$;

therefore, $f_u(0,b) = f_v(a,0) = 0$ and so the center of the cube face is not displaced. Furthermore, the edges of the cube face are not displaced, so $f_u(1,b) = f_v(a,1) = 1$. However, the approaches differ in several key qualitative aspects. Each one may be:

- *bivariate* (the general case), or *univariate* if there exists a function $f : \mathbb{R} \mapsto \mathbb{R}$ with $f_u(a,b) = f(a)$ and $f_v(a,b) = f(b)$;
- *diagonally symmetric* with $f_u(a,b) = f_v(b,a)$; i.e., the warp treats the parameters a and b interchangeably. All univariate warps are diagonally symmetric;
- *analytically invertible* or only approximately invertible via numerical root finding (we discuss this in detail in Section 4).

Aside from these key categorical distinctions, the most important quantitative difference between the approaches described here is the degree to which they are area-preserving; that is, a region covering a given percentage of the unit cube covers the same percentage of the unit sphere after transformation.

Here we follow Arvo's [2001] recipe for deriving the area element of the transformation defined by the composition of Equations (1) to (3). According to the method he provides, the area element is given by

$$dA(a,b) = \sqrt{\left(\frac{\partial\hat{\omega}}{\partial a}\cdot\frac{\partial\hat{\omega}}{\partial a}\right)\left(\frac{\partial\hat{\omega}}{\partial b}\cdot\frac{\partial\hat{\omega}}{\partial b}\right) - \left(\frac{\partial\hat{\omega}}{\partial a}\cdot\frac{\partial\hat{\omega}}{\partial b}\right)^2}, \tag{4}$$

where $\frac{\partial\hat{\omega}}{\partial a}$ and $\frac{\partial\hat{\omega}}{\partial b}$ are the partial derivatives of the unit vector $\hat{\omega}$ with respect to the input cube face coordinates a and b. They are defined as

$$\frac{\partial\hat{\omega}}{\partial a} = \frac{1}{(u^2+v^2+1)^{\frac{3}{2}}}\begin{bmatrix} -uv\frac{\partial f_v}{\partial a} + v^2\frac{\partial f_u}{\partial a} + \frac{\partial f_u}{\partial a} \\ u^2\frac{\partial f_v}{\partial a} - uv\frac{\partial f_u}{\partial a} + \frac{\partial f_v}{\partial a} \\ -u\frac{\partial f_u}{\partial a} - v\frac{\partial f_v}{\partial a} \end{bmatrix}, \tag{5}$$

$$\text{and} \quad \frac{\partial\hat{\omega}}{\partial b} = \frac{1}{(u^2+v^2+1)^{\frac{3}{2}}}\begin{bmatrix} -uv\frac{\partial f_v}{\partial b} + v^2\frac{\partial f_u}{\partial b} + \frac{\partial f_u}{\partial b} \\ u^2\frac{\partial f_v}{\partial b} - uv\frac{\partial f_u}{\partial b} + \frac{\partial f_v}{\partial b} \\ -u\frac{\partial f_u}{\partial b} - v\frac{\partial f_v}{\partial b} \end{bmatrix}. \tag{6}$$

Substituting these definitions into Equation (4) and simplifying, we obtain

$$dA(a,b) = \left[\frac{\partial f_u}{\partial a}\frac{\partial f_v}{\partial b} - \frac{\partial f_u}{\partial b}\frac{\partial f_v}{\partial a}\right]\left(u^2+v^2+1\right)^{-\frac{3}{2}}. \tag{7}$$

For an exact equal-area cube-to-sphere mapping, $dA(a,b)$ is equal to $\frac{\pi}{6}$ everywhere. However, if the mapping is not equal-area, we may choose its free parameters to minimize the root-mean-squared error of the area element on a grid of (a,b) samples:

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n} E_A(a_i,b_i)^2}\,, \tag{8}$$

where $E_A(a,b) = dA(a,b) - \frac{\pi}{6}$. This optimization scheme was used previously by Chan and O'Neill [1975]; one contribution of our work is to apply it to a number of approaches whose parameters were not previously chosen optimally. For each of the warping approaches described below with free parameters to optimize, we minimize Equation (8) over a uniform grid of 256×256 samples covering the [0, 1] range of (a, b). See Table 1 for a summary of the numerical optimization results and Table 3 for optimal values of the free parameters. GLSL implementations of all mappings and their inverses can be found among the supplemental materials.

3.1. Identity Function (Standard Cube Maps)

The most straightforward projection from the cube to the sphere sets $u = a$ and $v = b$. Although this mapping exhibits the largest area error of all of the approaches described here, many graphics programmers are familiar with this approach as it forms the basis of standard cube maps [Greene 1986].

3.2. Tangent Warp Function

The tangent warp function is a univariate, invertible approach with a single free parameter θ. It is inspired by an analogy to 2D, illustrated in Figure 7: if we wish to divide each quadrant of the unit square into $2n$ equal-angle wedges, the y-coordinates along the $x = 1$ line will be given by $y = \tan(\frac{\pi i}{4n})$ for any integer-valued $i \in [-n, n]$. Then, when the square points are projected onto the unit circle, they are evenly spaced in arclength. When replacing the 2D square and circle with a 3D cube and sphere, the tangent warp loses its exact area-preserving property but nevertheless substantially outperforms the identity function. A similar mapping was previously proposed by Bitterli et al. [2015] in the context of importance sampling environment maps through rectangular portals. Researchers at Google VR subsequently used it explicitly as a cube-to-sphere projection [Brown 2017]. In contrast to these previous uses of

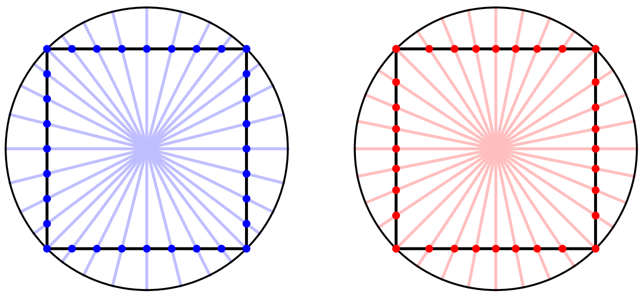
Figure 7. Motivation for tangent-based warp. Left: Equal-length subdivisions of the square result in unequal subdivisions of the circumscribed circle; Right: warping square coordinates through the tangent function guarantees uniform subdivisions of the circle.

the tangent mapping, our implementation introduces and optimizes a free parameter to better preserve area.

The free parameter is implicitly fixed at $\frac{\pi}{4}$ by both of the works we just cited, and this value is indeed optimal for equalizing angles in the 2D motivating example. However, in 3D it is possible to preserve area more effectively by using the warp

$$u = f(a) = \frac{\tan(a\,\theta)}{\tan(\theta)} \tag{9}$$

for some angle $\theta > 0$. The inverse of the tangent warp is given by

$$a = f^{-1}(u) = \frac{1}{\theta}\,\tan^{-1}\left(u\,\tan(\theta)\right). \tag{10}$$

3.3. Everitt's Univariate Invertible Warp Function

Everitt [2014] proposed a univariate, invertible warp function and a subsequent refinement [2016] whose respective formulas are

$$f(a) = \operatorname{sgn}(a)\left(\tfrac{3}{2} - \tfrac{1}{2}\sqrt{9 - 8\,|a|}\right), \tag{11}$$

$$\text{and}\quad f(a) = \operatorname{sgn}(a)\left(\tfrac{11}{6} - \tfrac{1}{6}\sqrt{121 - 96\,|a|}\right). \tag{12}$$

The general warp is best understood by considering its piecewise-quadratic inverse

$$a = f^{-1}(u) = u\,(\epsilon + (1-\epsilon)\,|u|) \tag{13}$$

for some $\epsilon > 1$. Upon inspection, $f^{-1}(1) = 1$, and $f^{-1}(-u) = -f^{-1}(u)$. To obtain the forward warp function we solve for u and find

$$u = f(a) = \operatorname{sgn}(a)\left(\frac{\epsilon - \sqrt{\epsilon^2 - 4\,(\epsilon - 1)\,|a|}}{2(\epsilon - 1)}\right). \tag{14}$$

Everitt's Equation (11) is a special case of Equation (14) with $\epsilon = 1.5$, and Equation (12) sets $\epsilon = 1.375$.

3.4. Odd 1D polynomial

To finish off the univariate methods, we introduce the odd (antisymmetric) polynomial

$$f(a) = k_1 a + k_2 a^3 + k_3 a^5 + \dots \tag{15}$$

By construction, $f(-a) = -f(a)$. An odd polynomial with n coefficients has degree $2n-1$, and to enforce the constraint that $f(1) = 1$, we require that $\sum_{i=1}^{n} k_i = 1$. The inspiration for our design of this method is that trigonometric functions, particularly inverse functions, can be expensive to evaluate and are hence frequently replaced by polynomial approximations. In this case, instead of optimizing polynomial coefficients to approximate $\tan(x)$ as closely as possible, we directly optimize the coefficients to minimize Equation (8). Indeed, the polynomials obtained through optimization visually resemble the tangent function. However, unlike the tangent and Everitt approaches, they are not analytically invertible for $n > 2$.

3.5. COBE Quadrilateralized Spherical Cube

Chan and O'Neill [1975] proposed a bivariate, diagonally symmetric polynomial

$$f_u(a,b) = \lambda a + (1-\lambda)\, a^3 + (1-a^2)\, a \sum_{(i+j)\geq 1}^{\infty} \gamma_{ij}\, a^{2i}\, b^{2j}. \tag{16}$$

By construction, Equation (16) satisfies the requirements that $f_u(-a,b) = -f_u(a,b)$, and $f_u(1,b) = 1$. Owing to diagonal symmetry, the second cube coordinate is defined by $f_v(a,b) = f_u(b,a)$. Their model is often called the COBE quadrilateralized spherical cube model, as it was developed in support of the NASA Cosmic Background Explorer (COBE) project.

3.6. Arvo's Exact Equal-area Method

Arvo [2001] provides a recipe for analytically constructing an area-preserving parameterization between smooth 2D surfaces. We apply Arvo's method to construct an equal-area mapping from cube face to sphere, arriving at the warp functions:

$$u = f_u(a,b) = \frac{\sqrt{2}\tan\left(\frac{\pi a}{6}\right)}{\sqrt{1-\tan\left(\frac{\pi a}{6}\right)^2}} \quad\text{and}\quad v = f_v(a,b) = \frac{b}{\sqrt{1+(1-b^2)\cos\left(\frac{\pi a}{3}\right)}}. \tag{17}$$

Arvo's recipe also produces an inverse mapping given by

$$a = \frac{6}{\pi}\tan^{-1}\left(\frac{u}{\sqrt{u^2+2}}\right) \quad\text{and}\quad b = \frac{v\sqrt{u^2+2}}{u^2+v^2+1}. \tag{18}$$

Like the COBE approach, this pair of warp functions is bivariate. However, unlike COBE, the resulting warp is not symmetric about the diagonal of a cube face. Although each face is warped continuously to the sphere, the overall cube-to-sphere projection is discontinuous along several cube edges, as evidenced by the T-vertices in Figure 8. Due to this discontinuity, Arvo's method is not suitable for the methods we describe (especially as illustrated by Listing 3 and Figure 9), and we therefore omit it from the comparison in Section 5.

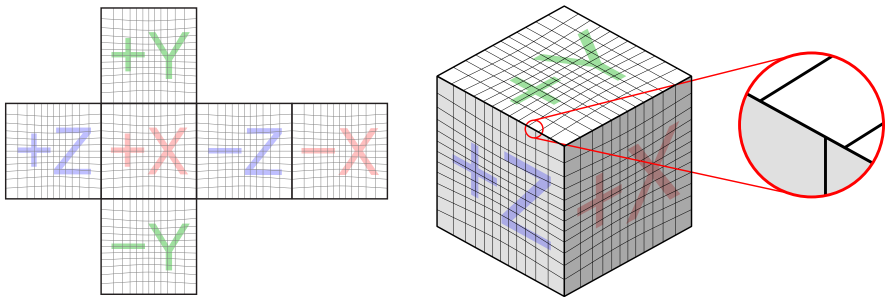

Figure 8. Using Arvo's method to map the entire cube to a sphere results in discontinuities along cube edges, as evidenced by the T-vertices shown here.

4. Inverse Mappings and Nearest-neighbor Lookup

The cube-to-sphere mappings defined in Sections 3.1 to 3.5 can be used to distribute samples pseudorandomly on the surface of a sphere, as shown in Figure 1. The first step is to generate a jittered grid of points on the cube by discretizing each face into $m \times m$ square grid cells and randomly choosing a sample within each cell. These samples can then be warped and projected to the sphere via Equations (1) to (3). Given a unit vector $\hat{\omega}$, it is possible to identify the closest sample point by brute-force iteration over all $6m^2$ warped and projected samples. However, a far more efficient method begins by first identifying the unwarped grid cell on the cube corresponding to $\hat{\omega}$.

To accomplish this, it must be possible to invert the mappings described in Section 3; that is, find the (a, b)-coordinates on a cube face that map to a given $\hat{\omega}$. The first step is to project $\hat{\omega}$ onto the cube, as when performing a cube-map lookup. We identify the coordinate with the greatest absolute value and construct an appropriate signed permutation matrix $\mathbf{P}$ establishing the cube face and local coordinate system. We can then obtain the warped-cube coordinates (u, v) by computing

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \mathbf{P}^T \hat{\omega} \tag{19}$$

and setting $u = x/z$ and $v = y/z$ (i.e., dividing by the value of the maximal coordinate). What remains is to invert the mapping function: given $u = f_u(a, b)$ and $v = f_v(a, b)$, solve for a and b. The inverses of the tangent warp and Everitt's warp are given in Sections 3.2 and 3.3. However, the odd 1D polynomial method (Section 3.4) and the COBE method (Section 3.5) must be inverted numerically.

4.1. Numerical Inverses

For both the 1D polynomial and COBE warps, we begin by fitting an approximate inverse polynomial $\tilde{f}^{-1}$ of the same parametric form as Equation (15) or Equation (16), respectively. Each polynomial fit minimizes a sum of squared residual errors e_i. In the case of the univariate polynomial, we choose the coefficients of $\tilde{f}^{-1}$ to minimize

$$e_i = \left| \tilde{f}^{-1}(f(a_i)) - a_i \right| \tag{20}$$

over a linearly spaced set of samples $a_i \in [0, 1]$. For the bivariate polynomial, we choose the coefficients of $\tilde{f}^{-1} : \mathbb{R}^2 \mapsto \mathbb{R}$ to minimize

$$e_i = \left\| \begin{bmatrix} \tilde{f}^{-1}(u_i, v_i) \\ \tilde{f}^{-1}(v_i, u_i) \end{bmatrix} - \begin{bmatrix} a_i \\ b_i \end{bmatrix} \right\|, \quad \text{with} \quad \begin{aligned} u_i &= f_u(a_i, b_i) \\ v_i &= f_v(a_i, b_i) = f_u(b_i, a_i) \end{aligned} \tag{21}$$

over a uniform grid of $(a_i, b_i) \in [0, 1] \times [0, 1]$ (note diagonal symmetry).

Although the initial inverse estimate may exhibit significant approximation error, numerical root-finding techniques can virtually eliminate the approximation error in just a few iterations. We use the secant method and its multidimensional generalization, Broyden's method [Press et al. 1992], to refine the univariate and bivariate inverses, respectively. Their GLSL implementations may be found in Listing 1 and Listing 2. See Table 1 for a summary of the accuracy of these numerical inverses.

```
vec2 unwarp_secant(vec2 uv) { // estimate ab such that warp(ab) = uv
    vec2 ab = unwarp_approx_univariate(uv); // initial estimate
    vec2 err0 = warp(ab) - uv, delta_ab = -0.1 * err0;
    ab += delta_ab; // second estimate for secant
    for (int i=0; i<3; ++i) {
        vec2 err1 = warp(ab) - uv, delta_err = err1 - err0;
        if (length(err1) < ERR_TOL) { break; }
        vec2 inv_slope = mix(vec2(1.0), delta_ab / delta_err,
                             lessThan(vec2(1e-7), abs(delta_err)));
        delta_ab = -err1 * inv_slope; // mix prevents divide-by-zero
        ab += delta_ab; err0 = err1;
    }
    return ab;
}
```

Listing 1. Vectorized secant method for inverting univariate mapping, implemented in GLSL.

```
vec2 unwarp_broyden(vec2 uv) { // estimate ab such that warp(ab) = uv
    mat2 H = mat2(1.0); // approximate inverse Jacobian
    vec2 ab = unwarp_approx_bivariate(uv); // initial estimate
    vec2 err0 = warp(ab) - uv, delta_ab = -0.1 * err0;
    ab += delta_ab; // second estimate for Broyden
    for (int i=0; i<3; ++i) {
        vec2 err1 = warp(ab) - uv;
        if (length(err1) < ERR_TOL) { break; }
        vec2 Hdf = H * (err1 - err0);
        H += outerProduct((delta_ab - Hdf) / dot(delta_ab, Hdf),
                          delta_ab * H);
        delta_ab = -H * err1; ab += delta_ab; err0 = err1;
    }
    return ab;
}
```

Listing 2. Broyden's method for inverting bivariate mapping, implemented in GLSL.

4.2. Nearest neighbors in Jittered Sphered-Cube Grids

Having established a reliable inverse mapping, we can now identify the k nearest sample points to any point $\hat{\omega}$ on the sphere in constant time, following these steps:

1. Unproject $\hat{\omega}$ to get the permutation matrix $\mathbf{P}$, and use the inverse mapping to get the unwarped face coordinates (a, b).
2. Search an $s \times s$ neighborhood of grid cells (e.g., $2{\times}2$ for $k = 1$, or larger for $k > 1$, see below) to find the closest sample point(s).
3. When ranking sample points, use distance on the sphere to $\hat{\omega}$ (as opposed to distance on the flat cube faces). It is sufficient to choose the sample point that has the maximum dot product with the 3D vector $\hat{\omega}$.

The single closest sample point always lies within a $2{\times}2$ neighborhood centered on the cell corner closest to $\hat{\omega}$. However, to correctly identify the second-closest sample, it is necessary to search a $4{\times}4$ neighborhood instead. In general, a larger k implies larger neighborhoods. Since it does not depend on the grid size m, but only on the neighborhood size s, the algorithm outlined above runs in constant time with respect to the total number of sample points.

Aside from using the correct metric to rank points—distance on the sphere, as opposed to the cube—when adopting grid-based methods using the cube-to-sphere approach, it is vital to respect edge-wrapping effects while iterating over neighborhoods. For example, a $3{\times}3$ neighborhood is the set of grid cells on the cube that can be accessed by moving up one step in the horizontal and vertical directions from the center cell. Although in the 2D planar domain this always results in a neighborhood size of nine cells, there may be fewer on the cube, as shown in Figure 9.

We address this issue with an algorithm (Listing 3) that handles edge wrapping and rejects steps (e.g., offsets to neighboring cells) that would cross more than one cube edge. With respect to the central point at $(0.75, 0.75, 1)$ in Figure 9, the algorithm permits a step of $(+0.5, 0, 0)$, causing a wrap across to the positive-x face of the cube, or a step of $(0, +0.5, 0)$ that wraps across to the positive-y face. However, the

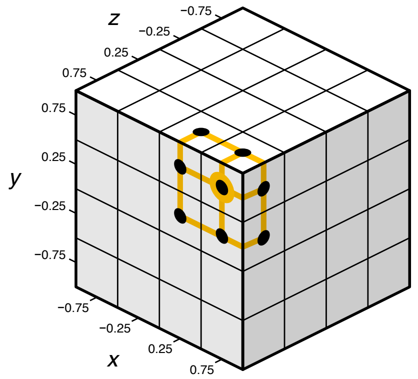

Figure 9. On the cube, considering all cells within ±1 horizontal or vertical steps of a central cell may result in fewer neighbors than would be found on a 2D planar grid.

```
bool wrapCube(in mat3 PT,      // transpose of permutation matrix
              inout vec2 uvn, // displaced 2D coords on cube face
              out mat3 PTn) { // new permutation matrix
    vec2 uvn_clamp = clamp(uvn, -1.0, 1.0);
    vec2 extra = abs(uvn_clamp - uvn);
    if (min(extra.x, extra.y) > 0.0) {
        return false; // more than one coord.  left [-1, 1], so reject!
    } else {
        float esum = extra.x + extra.y; // distance past edge
        if (esum == 0.0) {
            PTn = PT; // didn't cross edge, same as starting face
        } else { // need to compute starting face
            vec3 p = PT * vec3(uvn_clamp, 1.0 - esum); // pos in 3D
            PTn = getPT(p); // permutation matrix for Equation (1)
            uvn = (p * PTn).xy; // update cube face location
        }
        return true; // displacement was valid
    }
}
```

Listing 3. Validating displacements that may potentially cross cube edges.

algorithm rejects a step of $(+0.5, +0.5, 0)$, as it would cross more than one cube edge. The total number of permitted steps therefore corresponds to the actual neighborhood size on the cube.

5. Comparisons and Recommendations

As Figure 6 and Table 1 show, the warp methods we have analyzed vary substantially in terms of their ability to preserve area. Of the univariate methods, the fifth-order polynomial performs best both in terms of RMSE and worst-case error. In turn, it is substantially outperformed by the bivariate six-parameter COBE method.

To benchmark these methods' runtimes, we used a GLSL fragment shader to render the Voronoi diagram for a single warped cube face into a 1024×1024 quad for each warp method. The shader evaluates a single inverse warp (including iterative numerical inverses for the 1D polynomial and COBE methods) and 16 forward warps to traverse a 4×4 neighborhood around the input point.

Table 2 summarizes the mean draw time and standard deviation for 100 draw calls using each warp method, on two different GPUs: an integrated Intel IRIS and a discrete NVIDIA GTX 1080. Although the discrete GPU is a factor of 20–40× faster than the integrated one, they share some broad trends in common. For both GPUs, the fastest approximately area-preserving warp is the fifth-order odd 1D polynomial (despite requiring an iterative numerical inverse solver), and the slowest is the COBE method, though not by more than a factor of 2× over the 1D polynomial. These relative performance statistics are illustrated in Figure 10.

Warp method	**Parameter source**	**Area RMSE**	**Area max error**	**Initial inv. error**	**Numerical inv. error**
identity	N/A	0.199	0.476	–	–
tangent	$\theta = \frac{\pi}{4}$ [Brown 2017]	0.047	0.0933	–	–
	$\theta \approx 0.8687$ (optimal)	0.021	0.0738	–	–
Everitt	$\epsilon = 1.5$ [Everitt 2014]	0.0353	0.246	–	–
	$\epsilon = 1.375$ [Everitt 2016]	0.0436	0.112	–	–
	$\epsilon \approx 1.4511$ (optimal)	0.0266	0.115	–	–
5th order poly.	optimal	0.0209	0.0703	0.00109	5.55×10^{-16}
6 param. COBE	optimal	0.0013	0.00642	0.00187	3.96×10^{-8}

Table 1. Summary of numerical results for projected area error and inverse approximations. Area RMSE is the optimization objective (Equation (8)) and area max error is the worst-case observed value for E_A in that equation. Inverse errors refer to the worst-case observed value for the initial inverse guess and subsequent refinement by an iterative solver. Dashes indicate the method has an analytic inverse.

	Intel IRIS		**NVIDIA GTX 1080**	
Warp method	**mean**	**std.**	**mean**	**std.**
identity	2.262 ms	0.056 ms	0.071 ms	0.011 ms
tangent	3.277 ms	0.057 ms	0.163 ms	0.006 ms
Everitt	2.926 ms	0.051 ms	0.178 ms	0.004 ms
5th order poly.	2.846 ms	0.033 ms	0.143 ms	0.009 ms
6 param. COBE	3.581 ms	0.038 ms	0.218 ms	0.006 ms

Table 2. Frame timing data collected on two different GPUs.

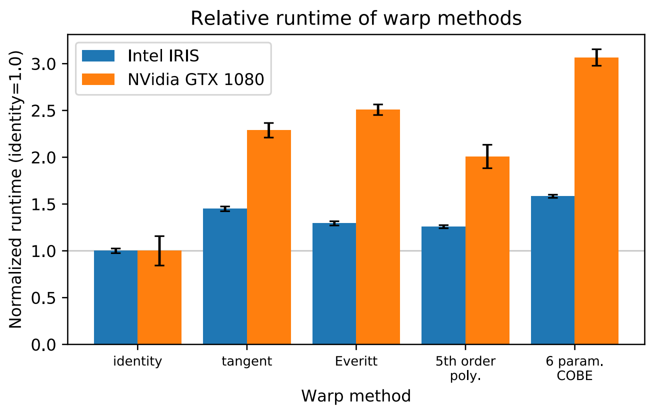

Figure 10. Timing data from Table 2 normalized to show trends across two different GPUs. Error bars indicate ±1 standard deviation.

Warp method/param.		**Forward**	**Inverse**
tangent	θ	0.8687	–
Everitt	ϵ	1.4511	–
5th order poly.	k_1	0.1239	0.1433
	k_2	0.1305	−0.4865
	k_3	0.7456	1.3432
6 param. COBE	λ	0.7240	1.3774
	γ_{10}	−0.0941	−0.2129
	γ_{01}	0.0276	−0.1178
	γ_{20}	−0.0623	0.0694
	γ_{11}	0.0409	0.0941
	γ_{02}	0.0342	0.0108

Table 3. Optimal free parameters for the mappings defined in Sections 3.2 to 3.5 along with their approximate inverses (Section 4.1). Dashes indicate the method has an analytic inverse.

Summarizing our results: if speed is paramount, we recommend the fifth-order odd 1D polynomial. It is also the best univariate method in terms of projected area error. If area-preserving capability is the chief criterion, we recommend the COBE method (although for our procedural texturing application, all of the methods visually appear nearly identical). Finally, to maximize ease of implementation, we recommend the tangent method, as it outperforms Everitt's method in preserving area.

6. Generalizing 2D Grid-based Methods to the Sphere

Beyond texturing, the cube-to-sphere approach enables a broader range of 2D grid-based methods to be extended to the sphere with minimal distortion. This is a particularly useful feature because methods on the plane that make use of an underlying grid structure are often simple, fast, and easily parallelizable on the GPU. As an example, we extend a state-of-the-art 2D Poisson disk sampling algorithm to the spherical domain using cube-to-sphere mapping and show that the quality of the resulting point set is not compromised.

Poisson disk sampling generates a uniformly random set of points such that no two are closer than a specified minimum distance. Naïve dart throwing [Cook 1986] becomes intractably slow as the number of points increases, so considerable effort has gone into improving runtime, especially in the planar domain [Wei 2008; Gamito and Maddock 2009] as well as on arbitrary surfaces [Bowers et al. 2010; Cline et al. 2009]. Comparatively little work has been done on explicitly generating Poisson disk distributions on a sphere. Generating the point set on a unit square and then lifting it up to the sphere using an equal-area projection preserves the point set's asymptotic distribution uniformity, but not distances between samples [Marques et al. 2013]. Other algorithms that directly sample on the sphere [Cline et al. 2009; Gamito and Maddock 2009] are not straightforward to parallelize on the GPU.

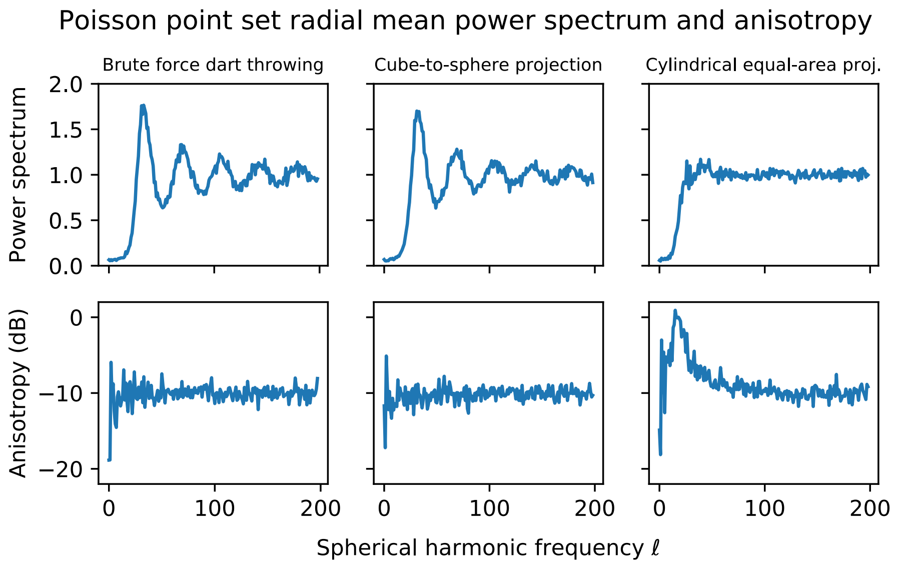

Figure 11. Comparing radially averaged power spectrum and anisotropy for ~300 points using brute force dart throwing, Wei's [2008] multiresolution algorithm adapted to the sphere with our 5th-order polynomial cube-to-sphere mapping, and 2D dart throwing lifted to the sphere by Lambert cylindrical equal-area mapping.

Instead, we propose to use the structure of the sphered cube to aid with direct Poisson disk sampling on the sphere. As an example, we adapt Wei's [2008] multiresolution algorithm to the sphere using the 5th-order polynomial mapping. Wei uses a grid structure on the unit square to make Poisson disk sampling GPU-friendly, and demonstrates that the resulting point sets still exhibit excellent blue noise qualities. By changing the underlying spatial structure from a 2D grid to a sphered cube and using the methods of Section 4 for conflict checking, we are able to achieve high quality Poisson disk sampling on the sphere that is parallelizable on the GPU.

We examine the quality of various Poisson disk point sets on the sphere by computing the relative radius, radial mean power spectrum, and anisotropy [Lagae and Dutré 2007]. Figure 11 compares Wei's algorithm combined with cube-to-sphere mapping with brute force dart throwing directly on the sphere and brute force dart throwing on the unit torus (i.e., unit square with periodic boundaries) lifted to the sphere by Lambert cylindrical equal-area mapping. Due to singularities at the poles, the latter point set exhibits poor blue noise characteristics and high anisotropy at low frequencies. The relative radius is around 0.2 to 0.4, well below the range $[0.65, 0.85]$ suggested by Lagae and Dutré. In contrast, our adaptation of Wei's algorithm generates point sets with very similar spectral properties to brute force dart throwing. The relative radius is consistently around 0.75, within the suggested range of $[0.65, 0.85]$. Far less distortion is introduced because sampling and conflict checking are performed

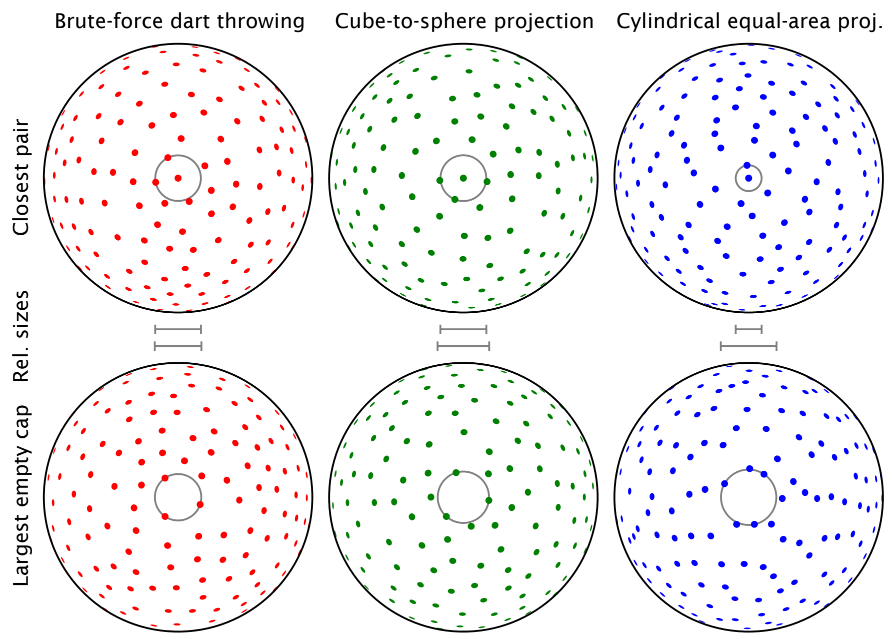

Figure 12. Comparing one example spherical point set per sampling method. Each is rotated to highlight both the closest point pair as well as the largest empty spherical cap.

directly on the sphere, yielding nearly identical quality to Wei's original results on the plane. Figure 12 shows example point sets from all three methods, illustrating that the cylindrical equal-area mapping yields far less regular distributions than the other two methods.

7. Conclusion

We have shown how cube-to-sphere mappings can aid in GPU procedural modeling and texturing of the sphere. In particular, we have detailed several pre-existing projections and one novel one. We have also shown how to efficiently implement fast nearest-neighbor query for jittered grids on the sphere using this approach. Finally, we demonstrated how to use these mappings to adapt general 2D grid-based methods to the sphere, showcasing Poisson disk sampling as an example. This is a promising application area for the sphered-cube approach based on the popularity of 2D grid-based methods, and because algorithms for sampling the plane are typically much simpler and faster than algorithms for sampling arbitrary meshes or manifolds.

References

Arvo, J. 1995. Stratified sampling of spherical triangles. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, ACM, New

York, ACM SIGGRAPH, 437–438. URL: https://dl.acm.org/citation.cfm?id=218380.218500. 2

Arvo, J. 2001. Stratified sampling of 2-manifolds. *SIGGRAPH 2001 Course Notes*. URL: https://pdfs.semanticscholar.org/4b29/674656bbf4067f23f0c24fe1b2e7ae198d7f.pdf. 2, 6, 8, 11

Bitterli, B., Novák, J., and Jarosz, W. 2015. Portalmasked environment map sampling. *Computer Graphics Forum 34*, 4, 13–19. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.12674. 9

Bowers, J., Wang, R., Wei, L.-Y., and Maletz, D. 2010. Parallel Poisson disk sampling with spectrum analysis on surfaces. *ACM Transactions on Graphics (TOG) 29*, 6, 166:1–166:10. URL: https://dl.acm.org/citation.cfm?id=1866188. 17

Brown, C., 2017. Bringing pixels front and center in VR video. URL: https://www.blog.google/products/google-vr/bringing-pixels-front-and-center-vr-video/. 2, 6, 9, 16

Chan, F., and O'Neill, E. 1975. Feasibility study of a quadrilateralized spherical cube Earth data base. Tech. Rep. 2-75, Environmental Prediction Research Facility, Naval Postgraduate School. URL: https://ntrl.ntis.gov/NTRL/dashboard/searchResults/titleDetail/ADA010232.xhtml. 2, 9, 11

Cline, D., Jeschke, S., White, K., Razdan, A., and Wonka, P. 2009. Dart throwing on surfaces. *Computer Graphics Forum 28*, 4, 1217–1226. URL: https://onlinelibrary.wiley.com/doi/full/10.1111/j.1467-8659.2009.01499.x. 17

Cook, R. L. 1986. Stochastic sampling in computer graphics. *ACM Trans. Graph. 5*, 1, 51–72. URL: https://dl.acm.org/citation.cfm?id=8927. 17

Everitt, C., 2014. Twitter update. URL: https://twitter.com/casseveritt/status/550483976243412993. 10, 16

Everitt, C., 2016. "Projection" repository. URL: https://github.com/casseveritt/projection/blob/ee0c792a748d6786ce6010d839f4f1f43e71184b/envmap.h#L197. 2, 10, 16

Gamito, M. N., and Maddock, S. C. 2009. Accurate multidimensional Poisson-disk sampling. *ACM Transactions on Graphics (TOG) 29*, 1, 8:1–8:19. URL: https://dl.acm.org/citation.cfm?id=1640451. 17

Gorski, K. M., Hivon, E., Banday, A., Wandelt, B. D., Hansen, F. K., Reinecke, M., and Bartelmann, M. 2005. HEALPix: a framework for high-resolution discretization and fast analysis of data distributed on the sphere. *The Astrophysical Journal 622*, 2, 759–771. URL: http://iopscience.iop.org/article/10.1086/427976/meta. 2

Greene, N. 1986. Environment mapping and other applications of world projections. *IEEE Computer Graphics and Applications 6*, 11, 21–29. URL: http://ieeexplore.ieee.org/abstract/document/4056759/. 2, 9

Heitz, E., Dupuy, J., Hill, S., and Neubelt, D. 2016. Real-time polygonal-light shading with linearly transformed cosines. *ACM Transactions on Graphics (TOG) 35*, 4, 41:1–41:8. URL: https://dl.acm.org/citation.cfm?id=2925895. 2

Keinert, B., Innmann, M., Sänger, M., and Stamminger, M. 2015. Spherical Fibonacci mapping. *ACM Transactions on Graphics (TOG) 34*, 6, 193:1–193:7. URL: https://dl.acm.org/citation.cfm?id=2818131. 2, 4

Lagae, A., and Dutré, P. 2007. A comparison of methods for generating Poisson disk distributions. *Computer Graphics Forum 27*, 1, 114–129. URL: https://onlinelibrary.wiley.com/doi/full/10.1111/j.1467-8659.2007.01100.x. 18

Lambers, M. 2016. Mappings between sphere, disc, and square. *Journal of Computer Graphics Techniques Vol 5*, 2, 1–21. URL: http://jcgt.org/published/0005/02/01/. 2

Marques, R., Bouville, C., Ribardière, M., Santos, L. P., and Bouatouch, K. 2013. Spherical Fibonacci point sets for illumination integrals. *Computer Graphics Forum 32*, 8, 134–143. URL: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12190. 17

Meyer, Q., Smuth, J., Suner, G., Stamminger, M., and Greiner, G. 2010. On floating-point normal vectors. *Computer Graphics Forum 29*, 4, 1405–1409. URL: https://onlinelibrary.wiley.com/doi/full/10.1111/j.1467-8659.2010.01737.x. 2

Press, W. H., Teukolsky, S. A., Vetterling, W. T., and Flannery, B. P. 1992. *Numerical recipes in C*. Cambridge University Press. URL: http://apps.nrbook.com/c/index.html. 13

Quílez, Í., 2013. Shadertoy: Voronoi - 3D. URL: https://www.shadertoy.com/view/ldl3Dl. 4

Quílez, Í., 2015. Voronoise. URL: http://www.iquilezles.org/www/articles/voronoise/voronoise.htm. 3

Shao, M.-Z., and Badler, N. 1996. Spherical sampling by Archimedes' theorem. Tech. Rep. MS-CIS-96-02, University of Pennsylvania Department of Computer and Information Science. URL: https://repository.upenn.edu/cgi/viewcontent.cgi?article=1188&context=cis_reports. 2

Singh, G. 2015. *Sampling and variance analysis for Monte Carlo integration in spherical domain*. PhD thesis, Université Claude Bernard-Lyon I. URL: https://hal.archives-ouvertes.fr/tel-01217082/document. 2

Wei, L.-Y. 2008. Parallel Poisson disk sampling. *ACM Transactions on Graphics (TOG) 27*, 3, 20:1–20:9. URL: https://dl.acm.org/citation.cfm?id=1360619. 17, 18

Wong, T.-T., Luk, W.-S., and Heng, P.-A. 1997. Sampling with Hammersley and Halton points. *Journal of Graphics Tools 2*, 2, 9–24. URL: http://www.tandfonline.com/doi/abs/10.1080/10867651.1997.10487471. 2

Worley, S. 1996. A cellular texture basis function. In *Proceedings of the 23rd annual conference on computer graphics and interactive techniques*, ACM, New York, SIGGRAPH '96, ACM SIGGRAPH, 291–294. URL: https://dl.acm.org/citation.cfm?id=237267. 3

Index of Supplemental Materials

GLSL shader source code corresponding to many of the figures can be found online:

- Figures 1 and 4: Basic approach, jittered grid Voronoi diagrams
  `https://www.shadertoy.com/view/MtBGRD`
- Figure 5, left: Space scene
  `https://www.shadertoy.com/view/Mtj3DV`
- Figure 5, right: Simulating brittle fracture
  `https://www.shadertoy.com/view/llXXz4`
- Figure 2: Value noise and Voronoise
  `https://www.shadertoy.com/view/Ml2czm`
- Figure 3: Comparison with Voronoi regions from 3D grid
  `https://www.shadertoy.com/view/Xd3SRj`
- Figure 6: All mappings, iterative inverse solvers, basis for benchmarking
  `https://www.shadertoy.com/view/XdlfDl`

Thanks to Íñigo Quílez and BeautyPi for maintaining Shadertoy.com and furnishing it with many valuable examples.

Author Contact Information

Matt Zucker
Swarthmore College
500 College Avenue
Swarthmore PA 19081, USA
mzucker1@swarthmore.edu
http://swarthmore.edu/NatSci/mzucker1

Yosuke Higashi
Swarthmore College
500 College Avenue
Swarthmore PA 19081, USA
yhigash1@swarthmore.edu

M. Zucker and Y. Higashi, Cube-to-sphere Projections for Procedural Texturing and Beyond, *Journal of Computer Graphics Techniques (JCGT)*, vol. 7, no. 2, 1–22, 2018
http://jcgt.org/published/0007/02/01/

Received:	2018-04-04		
Recommended:	2018-05-15	Corresponding Editor:	Stephen Hill
Published:	2018-06-26	Editor-in-Chief:	Marc Olano