The Tennis-ball Map: A Low-Distortion Sampling-Efficient Parameterization of the Sphere

Dr. Gary Herron
Department of Computer Science
Digipen Institute of Technology
gherron@digipen.edu

Abstract

We introduce a parameterization of the sphere, called the tennis-ball parameterization, which exhibits properties highly desirable for computer graphics applications which need to represent values over a sphere, for instance spherical images for view-independent reflections maps and omni-directional shadow maps. We are particularly interested in using such maps in applications requiring high frame-rate real-time rendering of dynamic scenes.

We use some carefully selected metrics to compare the tennis-ball parameterization to a number of other parameterizations found in traditional computer graphics, as well as several found in applied mathematics and sciences. Under the restrictions implied by view-independence and the high frame rate of real-time applications, we will show that the tennis-ball map exhibits some greatly improved characteristics compared to the other parameterizations.

1 Introduction

There are a number of graphical techniques which require *spherical images*, that is the sampling of values over a spherical or hemispherical domain and the storage of the results into one or more rectangular texture maps. These include reflection and environment maps [OpenGL, Haeberli93, Heidrich98] which record a radiance field over the sphere, and (omni-directional) shadow maps [Williams78, Segal92, Brabec02] which record light penetration distance from a point light source into a scene. Other techniques include illumination maps and sky maps for global illuminations calculations [Kim07] and the recording of form-factor calculations in radiosity applications [Cohen85].

In this paper, we are particularly interested in the use of spherical images in situations requiring high-frame-rate, real-time rendering of dynamic scenes. The requirement of **dynamic** scenes is meant to include multi-pass rendering pipelines where, for each screen refresh, a spherical image (e.g., reflection maps or omni-directional shadow maps) must be created in one rendering passes and indexed in a later pass. Many examples of such applications can be found in the realm of modern computer games where the quest for realism is unbounded and the trade-off between realism and real-time refresh rates can be severe.

We present a parameterization of the sphere inspired by the "unwrapping" of a baseball [Thompson98] or tennis ball along its seam to create two flat and nearly rectangular sections. The resulting parameterization of the sphere, which we call the tennis-ball map, is compared to the parameterizations found in traditional computer graphics, as well as some found in cartography, some newly defined maps from [Snyder01], and several found in applied math situations (cosmic, atmospheric and geologic sciences). These are:

- *Latitude-longitude map* [Blinn76]: Often referred to as spherical or polar coordinates, where constant parameter lines form the longitude and latitude lines. This projection is called the *plane chart* in cartography.
- *Sphere map* [OpenGL]: Also called the "gazing sphere", has been available for many years now on graphics cards. This uses the *Lambert equal area* projection from cartography.
- *Cube map* [Voorhies94, OpenGL]: Also available on graphics cards for years. This is formed from six instances of cartography's *gnomonic* projection of a sphere onto the six planes of a cube.
- *Dual paraboloid map* [Heidrich98]: A more recent map, this uses the analogy of projections via two parabolic mirrors onto the center circular portion of two rectangular domains. This projection is referred to as *stereographic* in cartography.
- *Polar-Capped etc.* [Snyder01]: With the insight allowed by their careful analysis of spherical maps, this paper proposes five new maps with an intriguing range of properties.
- *HealPix* [Gorski05] and *Rhombic Dodecahedron* [Fu09] Maps: HealPix is used in cosmology and the Rhombic Dodecahedron is used in panoramic video processing. Both are mapping of the sphere via projections into twelve rectangular components. They have some nice properties for scientific analysis, but the need for twelve components make then impractical for real-time dynamic graphics applications.
- Yin-Yang Grid [Kageyama2004], and the Bi-Mercator grid [Purser04]. These are two parameterizations (or family of parameterizations) developed for use in atmospheric sciences for the analysis of data over a spherical domain. They are very similar to the tennis-ball map, but include features not needed in computer graphics (such as overlap between the components), and to our knowledge, have never been used or analyzed for use in computer graphics.

The comparison of the tennis-ball map to these maps will be made via the following criteria including two very useful metrics defined in [Snyder01]:

- **Sampling requirement:** The pixel sampling induced by a map should support the conservation of the

highest frequency content across the sphere with the fewest number of samples. We use a quantity derived from the *stretch metric* of [Snyder01] which identifies the worse-case pixel stretching, and then ensures that there are enough pixels to represent the desired frequency content there. This metric also accounts for wasted pixels caused by modern graphics hardware's need to allocate only rectangular domains even when a map requires some non-rectangular portion. For example, see [OpenGL] which uses only the central circular portion of a square domain.

- **Anisotropy:** Any mapping from a rectangle to a sphere must necessarily introduce some distortion. We quantify this distortion with a metric that measures *anisotropy*, also from [Snyder01]. We favor small anisotropy since large anisotropy can cause blurring artifacts when filtering for MIPMAPS, and even modern hardware with anisotropic filtering capabilities can handle only limited amounts of anisotropy.
- **Number of components** and **Efficiency of creation:** Many of the parameterizations of the sphere considered in this paper consist of several components. That is, they are piecewise maps, using several rectangular domains and associated mapping functions to cover the sphere. Because of our interest in real-time applications with **dynamic** scenes, we are interested in maps with as few components as possible. For instance, when creating a reflection map, each component requires a pass of the scene's geometry through the full rendering pipeline, including the transformations, rasterization, and lighting/texturing phases. Even with hardware assisted rendering, excessive components will decrease frame-rates dramatically.
- **Efficiency of indexing:** The use of any spherical parameterization, for instance as a reflection map, at scene rendering time must index the corresponding texture map(s) many times. This requires the (inverse of the) mapping functions to be simple to calculate.

Table 1 summarizes the above metrics applied to many of the spherical parameterizations. The results are listed in an order sorted first by sampling requirements, and then by the other two criteria in either order.

In section 2, we will derive the tennis-ball parameterization, and in section 3 we will discuss the hardware-accelerated creation of spherical images using the parameterization. In section 4 we will present an analysis of its properties and compare it to other known parameterizations.

Map	Sampling Requirement	Maximum Anisotropy	Number of Components
Tennis-ball	14.8	1.41	2
Polar-Capped (stretch invariant)	14.8	1.41	3
Polar-Capped (conformal)	16.5	1	3
Dual Equidistant	19.7	1.57	2
Low Distortion Equal Area	19.7	3.45	1
Latitude-longitude	19.7	∞	1
Cube	24	1.73	6
Dual Parabolic	32	1	2
OpenGL	∞	∞	1
Theoretically Optimal Maps			
Isometric	12.57	1	∞
Non-Isometric	10.9	1.73	∞

Table 1: Comparison of sphere parameterizations, sorted first by sampling requirements, and then by the other two criteria in either order. Sampling requirement is proportional to the number of pixels needed to reproduce a given worst-case frequency content. Smaller values imply a more efficient use of pixels resources. Small values of maximal anisotropy are preferred, as large values result in potential MIPMAP filtering artifacts. A minimal number of components is highly desirable in real-time dynamic situations, where generating an spherical image requires a full pass through the rendering pipeline for each component. The last two maps, being theoretical limits as the number of components approaches infinity, are for comparison purposes, and cannot actually be realized.

2 The tennis-ball parameterization

The tennis-ball parameterization will be developed in several steps. First, in section 2.1, we define the "seam" curve that separates the sphere into two congruent components, then in section 2.2, we define a parameterization for each component. Finally, in section 2.3, we consider the inverse of the parameterization function, necessary for both the creation and indexing of a spherical image using the tennis-ball parameterization.

2.1 The seam curve

The inspiration for the tennis-ball parameterization comes from observing how tennis balls and baseballs are composed of two congruent nearly rectangular pieces attached along a single seam. See figure 1a. For a mathematical description of the seam curve, we consider the derivation from [Thompson98] where a one-parameter family of seam curves is defined. This family contains both the traditional baseball seam as well as the tennis-ball seam (as shown in figure 1a). We find the tennis-ball seam more suitable for our purpose.

Following [Thompson98] we start the derivation with a unit circle in the XY plane and a short straight curve $\bar{s}(t) = \left(t, \sqrt{2}/2\right)$ for $0 \le t \le \sqrt{2}/2$. This curve, shown in Figure 1b, is then projected (Figure 1c) onto the +Z hemisphere of the unit sphere, resulting in the quarter circle $s(t) = \left(t, \sqrt{2}/2, \sqrt{1/2 - t^2}\right)$. The full seam curve is then

composed of eight transformed copies of this segment. The four on the +Z hemisphere are produced by reflections of $\bar{s}(t)$ across the X and Y axes before projection onto the +Z hemisphere, and the four on the -Z hemisphere are produced by a 90° rotation of $\bar{s}(t)$, followed by the same reflections and then projection. Continuity of the eight segments can be seen by noting that the endpoints of all eight segments occur at eight points listed in table 2.

$(0,\ \pm\sqrt{2}/2,\ \sqrt{2}/2)$	2 points on the +Z hemisphere
$(\pm\sqrt{2}/2,\ \pm\sqrt{2}/2,\ 0)$	4 points on the XY equator
$(\pm\sqrt{2}/2,\ 0,\ -\sqrt{2}/2)$	2 points on the -Z hemisphere

Table 2: The endpoints of the seam curve segments.

We observe (see figure 1d) for use later in section 2.3, that the eight segments, taken in pairs, lie on four half planes, two at $y=\pm\sqrt{2}/2$ restricted to $z\geq 0$, and two at $x=\pm\sqrt{2}/2$ restricted to $z\leq 0$.

2.2 The parameterization

The seam curve divides the sphere into two congruent components which we will label C_1 and C_2. Out of the many orientations possible with respect to the coordinate axes, we choose component C_1 to be centered on the +Z-axis, wrapping around to encompass both +X and -X axes, and component C_2 centered on the -Z-axis, wrapping around to encompass both +Y and -Y axes. See Figure 2a. We further choose to parameterize each component with its own spherical coordinate system (θ,φ), carefully oriented so that the equator runs lengthwise through the component, and the poles and the meridian-seam is centered in the portion of the sphere unused by the component. See Figure 2b. For the sake of symmetry, we choose nonstandard ranges for the two angles. We parameterize the equatorial direction with $-\pi\leq\varphi\leq\pi$, placing $\varphi=0$ at the center of the component and the meridian-seam at $\varphi=\pm\pi$. Similarly, we parameterize from pole-to-pole with $-\pi/2\leq\theta\leq\pi/2$, placing $\theta=0$ at the center of the component. Thus, the parameterizations of the two components are

$$\begin{aligned} C_1:\ S_1(\theta,\phi) &= (\sin\phi\cos\theta,\ \sin\theta,\ \cos\phi\cos\theta) \\ C_2:\ S_2(\theta,\phi) &= (\sin\theta,\ \sin\phi\cos\theta,\ -\cos\phi\cos\theta) \end{aligned} \qquad (1)$$

The domain of each parameterization, approximately oval in shape, fits within a rectangle whose bounds can be calculated by considering the inverse of the points of table 1, or by the following symmetry considerations. Noting that the XY equator must be split into 4 equal segments, each occupying ¼ of the equator, we see that each component covers exactly ¾ of the circumference around its equator, and consequently exactly ¼ of a pole-to-pole great circle. Because we've chosen to parameterize each component with $\theta=\varphi=0$ at the center, we can conclude that each component occupies a range of parameters bounded by the rectangle (see figure 2c.)

$$-\tfrac{\pi}{4}\leq\theta\leq\tfrac{\pi}{4}\ \text{ and }\ -\tfrac{3\pi}{4}\leq\varphi\leq\tfrac{3\pi}{4}\,. \qquad (2)$$

2.3 The inverse parametrization

Any graphical use of a spherical parameterization needs the inverse of that parameterization, both when creating a spherical image, where objects at arbitrary directions are projected onto the sphere, and later when arbitrary directions are used to query the image. In the case of multiple components like the tennis-ball parameterizations, an initial step is necessary: Given any direction vector $R=(a,b,c)$, one must determine into which component the vector points. Then the inverse for that component's parameterization can be applied to compute θ and φ.

The inverse functions of equations (1) are simply:

$$\begin{aligned} C_1:\ (\theta,\varphi) &= (\text{asin}(b),\ \text{atan2}(a,\ c)) \\ C_2:\ (\theta,\varphi) &= (\text{asin}(a),\ \text{atan2}(b,-c)) \end{aligned} \qquad (3)$$

(We've used the two parameter version of arctangent [Atan2] which returns angles over the range $(-\pi,\pi]$, and avoids the singularities and restricted range of the usual arctangent.)

As for determining which component contains a given unit vector R, note that the planes mentioned at the end of section 2.1, and shown in figure 1d, give us some easy coordinate tests. In particular, in the +Z hemisphere the component is determined by testing whether or not R is between the upper pair of half planes $y=\pm\sqrt{2}/2$, and similarly in the -Z hemisphere, by testing whether or not R lies between the lower pair of half planes $x=\pm\sqrt{2}/2$. That is, for unit vector $R=(a,b,c)$

$$\text{Component}(R)=\begin{pmatrix} C_1: & c\geq 0 \text{ and } |b|\leq\sqrt{2}/2 \\ C_1: & c\leq 0 \text{ and } |a|\geq\sqrt{2}/2 \\ C_2: & \text{otherwise} \end{pmatrix}$$

3 Real-time creation of tennis-ball maps

The process of creating a spherical image according to the tennis-ball (or any other) parameterization is simple enough in concept: For each pixel sample (u,v) in the domain(s) of a parameterization S, the world space geometry is sampled in the direction $S(u,v)$ (with respect to some given central point), and the value(s) of interest (e.g., closest object depth for a shadow map, and closest object color for a reflection map) becomes the value of the pixel sample.

3.1 Rendering pipeline

We are interested in using all of the hardware-assisted rendering power of modern graphics cards to create a spherical image. This involves using the inverse of S as a projection from $\mathbb{R}^3$ onto the component domain(s) of S. Since the projective capabilities of graphics cards are limited to perspective and orthogonal projections, and the tennis-ball map's inverse is **not** of that form, we must transform the tennis-ball map's inverse into one that can be handled by the hardware. This involves "unwrapping" the tennis-ball component, and the space surrounding it, so that radial projections onto the sphere become orthogonal projections.

The equation for this unwrapping is nothing more than the projection map of (3) extended with a third coordinate to preserve radial distance from the sphere surface. That is, for a direction vector $R=(a,b,c)$:

$$C_1:\ (\theta,\varphi,r) = (\text{asin}(b),\ \text{atan2}(a,\ \ c),\ |R|)$$
$$C_2:\ (\theta,\varphi,r) = (\text{asin}(a),\ \text{atan2}(b,-c),\ |R|) \qquad (4)$$

This is diagrammed in figure 3a which shows a 2D representation of the unwrapping restricted to a slice through the component's equator. The figure also demonstrates the unwrapping's action on the space around the component, and several polygons embedded in that space, one of which is stretched across the seam and will be dealt with in the next section.

Performing this unwrapping transformation and a subsequent orthographic projection at a pixel level is beyond the capabilities of modern graphics cards, but an approximation can be achieved by performing the unwrapping transformation on polygon vertices followed by the usual hardware-assisted linear interpolation, rendering, and orthographic projection. As in [Brabec02], we assume the scene geometry can be tessellated fine enough for this approximation to be acceptable. Such a fine tessellation can be achieved at model creation time, or at run-time on the CPU, or perhaps best of all, as a geometry shader/program on a modern programmable GPU where it becomes part of the graphics pipeline.

A fully hardware assisted generation of an image for each component can now be achieved by implementing the tessellation in a geometry program (if desired), the unwrapping transformation in a vertex program, followed by a scaling of the area of interest (as seen in equation (2)) to the hardware clipping boundaries and an orthographic projection.

3.2 Seam crossing polygons

There remains the problem of dealing with polygons which cross the seam. Figure 3a demonstrates the problem associated with naively transforming the vertices of a seam-crossing polygon. The resulting unwrapped polygon would enter the rendering pipeline, cutting incorrectly across the whole scene. A number of solutions to handle such polygons exist, all of which can be implemented via the geometry, vertex or pixel/fragment shader programs available on modern graphics processors.

There is a large buffer area between the rectangle containing a component's domain of $\pm\frac{\pi}{4}\times\pm\frac{3\pi}{4}$ and the full range of the unwrapping transformation (4) of $\pm\pi\times\pm 2\pi$. If we add to the tessellation requirements of the previous section the added condition that any (seam-crossing) polygon has an angular extent of no more than $\pi/2$ with respect to the tennis-ball parameterization's center point, then all the vertices of a seam-crossing polygon are guaranteed to be in the buffer zone. Any of the following schemes to recognize such polygons and remove them from the rendering pipeline will accomplish our goal. See figure 3b.

1. A geometry program (or shader) can recognize such a polygon and choose not to propagate it.
2. A vertex program can recognize such a polygon and choose to transform all of its vertices to the same side of the seam. It does not matter which side is chosen since all endpoints are in the buffer zone and will be clipped away.
3. In a technique similar to the "alpha test" method from [Brabec02], a vertex program can associate a value of $\alpha=0$ or $\alpha=1$ to every vertex depending on whether (4) transforms the vertex to a point projecting, respectively, outside or inside the component's domain. Then an associated pixel/fragment program can recognize pixels from a seam-crossing polygon as those with an interpolated value of $\alpha=0$ and chose to **not** output such pixels.

4 Analysis

In order to compare the tennis-ball parameterizations to other parameterizations of the sphere, we use several metrics derived in [Snyder01]. These metrics are meant to recognize the efficient use of both pixel resources and rendering pipeline resources in real-time dynamic scene rendering on modern graphics processors.

4.1 Metrics

When choosing the number of pixels needed to produce a spherical image according to the tennis-ball parameterization, we follow the lead of [Snyder01] in assuming we want to conserve at least a given frequency of detail at all points across the image. Thus, we will choose the smallest number of pixel samples possible while guaranteeing to reproduce a given worst-case frequency content everywhere. We will do this via a *stretch metric* which calculates the amount of stretching/shrinking a pixel undergoes in the mapping from the sphere, identifying the worst-case stretching, and choosing the number of pixels needed to ensure the desired frequency content at such points. The resulting metric, called *sampling requirement* in [Snyder01] produces a value which is proportional to the number of pixel samples needed to reproduce a given worst-case frequency content, and is therefor an ideal measure with which to compare the efficiency of pixel utilization of a spherical parameterization.

The stretch metric also gives us another point of comparison. A parameterization's *anisotropy* quantifies the amount stretching in one direction versus its orthogonal direction. Modern graphics hardware can deal with limited amounts of anisotropy, but some of the parameterizations we analyze exhibit large, or even unbounded, anisotropy. Too much anisotropy can result in excessive blurring.

A further point of comparison is the number of components in a parameterization. A small number of components is important for minimizing the real-time indexing calculations, and especially important for minimizing the hardware-accelerated real-time creation of spherical images according to a parameterization, as was discussed in section 3.1.

The *stretch-metric* of [Snyder01] is defined thus: A mapping to the sphere will locally transform a infinitesimal circle to an ellipse, and the lengths of the major and minor axes of the ellipse, λ_1 and λ_2 represent the local stretching/shrinking of the map. These values can be calculated as singular values of the Jacobian of the map. That is, for map $S(u,v)$ with partial derivatives $S_u(u,v)$ and $S_v(u,v)$, the lengths of the major and minor axes, calculated from the singular values of the Jacobian are

$$\lambda_1^2(u,v) = \tfrac{1}{2}\left((a+c) + \sqrt{(a-c)^2+4b^2}\right)$$
$$\lambda_2^2(u,v) = \tfrac{1}{2}\left((a+c) - \sqrt{(a-c)^2+4b^2}\right) \quad (5)$$

where $a = S_u \cdot S_u$, $b = S_u \cdot S_v$, and $c = S_v \cdot S_v$.

The value λ_1 represents the maximal amount of stretch at any point. The worst-case stretching across a domain D of a component is then

$$\lambda_1^* = \max_{(u,v)\in D} \lambda_1(u,v) .$$

If domain D has an area A, then the value

$$Y = A\,\lambda_1^{*2} \quad (6)$$

is the constant of proportionality relating a desired feature size to the number of pixels need to preserve that feature size across all of D. This is called *stretch sampling requirement* in [Snyder01] where its full derivation can be found..

We also define the *anisotropy* of a map in terms of λ_1 and λ_2 simply as $\alpha = \lambda_1/\lambda_2$. Again, see [Snyder01]. Values of α range from one (meaning no anisotropy, i.e., conformal) upwards. Larger values of α may result in undesirable artifacts when performing filtering operation on a spherical images, such as MIPMAPs on reflection maps or various per-filtering convolutions applied to shadow maps (see variance [Donnelly06], convolution [Annen07], and exponential [Annen08] shadow maps).

For the tennis-ball parameterization, we calculate the sampling requirement thusly: Using the definition S_i of either component from (1), and normalizing the domain with $S(u,v) = S_i(2\pi u, 2\pi v)$ we calculate, via (5), and a good computer algebra system [Sage]

$$\lambda_1 = \lambda_1^* = 2\pi$$
$$\lambda_2 = 2\pi\cos u .$$

In this parameterization, the domain (2) becomes $\pm 1/8 \times \pm 3/8$ with an area of $3/16$ which yields a sampling requirement from (6) of $\frac{3}{16}(2\pi)^2 = \frac{3}{4}\pi^2$ for a single component. For the two components taken together, the sampling requirement is twice that at $Y = \frac{3}{2}\pi^2 \simeq 14.80$.

The anisotropy of the tennis-ball map $\alpha = \lambda_1/\lambda_2 = 1/\cos(2\pi u)$ achieves its maximal value $\alpha = \sqrt{2}$ when $u = \pm 1/8$, or equivalently, along the equatorial-parallel boundaries $\theta = \pm\pi/4$.

These values for the tennis-ball map and many of the maps from [Snyder01], are tabulated in table 1.

4.2 Comparison

Examining the sampling requirement results displayed in table 1, we see that no parameterization has a sampling requirement better than the tennis-ball map. The one map that has the same sampling requirement, the stretch invariant version of the polar-capped map, achieves the same values because it uses the same portion $\theta = \pm\pi/4$ of the longitude-latitude map (i.e., the plane chart projection). However it requires 3 components, and so is at a disadvantage compared to the tennis-ball map.

Examining the anisotropy values, we see the tennis-ball map has a value of 1.4, which is considered small. It is of questionable value to attempt to decrease it, but we find two conformal maps in table 1 that do so. However both maps offer significant disadvantages compared to the tennis-ball map. The conformal polar-capped map requires a third component and a sampling requirement 11% higher, and the dual-parabolic map has a sampling requirement 116% higher.

Finally, examining the number of components for each map in table 1, we can draw some conclusions. Firstly, the theoretical maps mentioned at the bottom of table 1, show that, if one is willing to increase the number of components far enough, the sampling requirement can be reduced. However the reduction will never be more than about 26%, and to achieve that would require an unwieldy number of components that is counter to our constraints of real-time generation of spherical images. Secondly, all the maps in table 1 with the same number of components as the tennis-ball map are at a disadvantage with respect to both other measures. Thirdly, it is consistent with our real-time constraints to consider maps with fewer components (i.e., one) than the tennis-ball map. Since generating spherical image requires a full pass through the rendering pipeline for each component, a map which reduces the number of components to one is potentially useful. Table 1 offers several maps with only one component. If one is highly constrained by the number of components, and not so concerned over efficient use of pixels resources, the "Low Distortion Equal Area" map of [Snyder01] offers a single component for an increase of sampling requirement of 33% and an increase of anisotropy of 145%. The other 1-component maps fair much worse in comparison to the tennis-ball map. The longitude-latitude map has a 33% increase in sampling requirement and infinite anisotropy, and the OpenGL map which has both infinite anisotropy and infinite sampling requirement.

Several maps mentioned in the introduction are not included in table 1. The Yin-Yang Grid [Kageyama04] [Kageyama05], and the Bi-Mercator grid [Purser04] use the same parameterization as the tennis-ball map and so should have the same metrics only slightly modified by their inclusion of a small overlap between the two components. The overlap represents a slight increase of the area A in (6) and a slight increase in the u parameter beyond the value $u = \pm 1/8$ (or equivalently $\theta = \pm\pi/4$) in calculating λ_2. This leads to a slight worsening of both metrics when compared to the non-overlapping tennis-ball parameterization, and the increase grows as the amount of overlap grows. Without overlap, both parameterizations are identical to the tennis-ball map and so have the same metrics.

A further intriguing idea from [Snyder01] comes from analyzing hexagonal arrays of pixels rather than the usual rectangular arrays. Because of the better packing of pixels in hexagonal arrays, they appear to allow for modest improvements in sampling-requirement. We await the inclusion of hexagonal arrays of pixels in hardware before pursuing this idea further.

5 Conclusion

We have presented a new parameterization of the sphere, called the tennis-ball map, motivated by the unrolling of the two

panels of a tennis ball, and especially designed for use in high frame-rate real-time rendering of dynamic scenes. The parameterization is compared to other parameterizations via several metrics meant to measure the efficient use of pixel resources, the amount of anisotropy of the map, and the number of components.

Because of our interest in real-time dynamic graphics situations, we dismiss from consideration any map with a large number of components. Of the remaining maps, we find that, compared to the tennis-ball map, none have a more efficient use of pixels, several decrease the anisotropy by only a small amount but worsen the other metrics, and the one map that has comparable values for the two metrics requires an extra component.

Thus we believe that the tennis-ball map offers a real advantage in any graphical operation requiring spherical images in a situation where efficient use if pixels, small anisotropy, and a small number of components are of importance.

References

[Atan2] Wikipedia, http://en.wikipedia.org/wiki/Atan2

[Annen07] T. Annen, T. Mertens, P. Bekaert, H.-P. Seidel, and J. Kautz. Convolution shadow maps. In Rendering Techniques 2007, volume 18 of Eurographics / ACM SIGGRAPH Symposium Proceedings, pages 51–60. Eurographics, June 2007.

[Annen08] Annen, T. and Mertens, T. and Seidel, H.-P. and Flerackers, E. and Kautz, J. (2008) Exponential shadow maps. In: Bartram, L. and Shaw, C., (eds.) Graphics Interface 2008, WIndsor, Ontario, Canada, May 28 - 30, 2008: Proceedings. (pp. pp. 155-161). Canadian Human-Computer Communications Society/ A.K. Peters Ltd: Mississauga, Canada.

[Blinn76] J. F. Blinn, and M. E. Newell. Texture and reflection in computer generated images. Communications of the ACM , 19:542—546, 1976.

[Brabec02] Stefan Brabec, Thomas Annen, Hans-Peter Seidel. Shadow Mapping for Hemispherical and Omnidirectional Light Sources, In Proc. of Computer Graphics International, 2002

[Cohen85] Michael F. Cohen, Donald P. Greenberg. The hemi-cube: a radiosity solution for complex environments, International Conference on Computer Graphics and Interactive Techniques archive Proceedings of the 12th annual conference on Computer graphics and interactive techniques, 1985.

[Donnelly06] W. Donnelly and A. Lauritzen. Variance shadow maps. In SI3D ’06: Proceedings of the 2006 symposium on Interactive 3D graphics and games, pages 161–165, New York, NY, USA, 2006. ACM Press.

[Fu09] Chi-Wing Fu, Liang Wan, Tien-Tsin Wong, Chi-Sing Leung: The Rhombic Dodecahedron Map: An Efficient Scheme for Encoding Panoramic Video. IEEE Transactions on Multimedia 11(4): 634-644 (2009)

[Green86] N. Greene. Environment mapping and other applications of world projections. IEEE Computer Graphics & Applications, 6(11):21–29, 1986.

[Haeberli93] Paul Haeberli and Mark Segal. Texture mapping as a fundamental drawing primitive. Fourth Eurographics Workshop on Rendering , pages 259–266, June 1993.

[Gorski05] Gorski, K.M., Hivon, E., Banday, A.J., Wandelt, B.D., Hansen, F.K., Reinecke, M., Bartelmann, M., "HEALPix: A Framework for High-Resolution Discretization and Fast Analysis of Data Distributed on the Sphere", 2005, *AP.J.* 622, p.759.

[Heidrich98] Wolfgang Heidrich and Hans-Peter Seidel. View-independent environment maps. In Proceedings of the SIGGRAPH/Eurographics Workshop on Graphics Hardware, 1998.

[Kageyama04] Kageyama, A. and T. Sato, 2004: ‘‘Yin-Yang grid’’: An overset grid in spherical geometry. Geochem. Geophys. Geosyst.,

[Kageyama05] Kageyama, A, 2005: ‘‘Yin-Yang grid’’: An overset grid in spherical geometry. Geochem. Geophys. Geosyst.,

[Kim07] Jonghyub Kim, Yongho Hwang, Hyunki Hong. Using irradiance environment map on GPU for real-time composition, Proceedings of the multimedia 8th Pacific Rim conference on Advances in multimedia information processing, 2007.

[OpenGL] OpenGL ARB. OpenGL Specification, Version 1.1, 1995.

[Osman06] Brian Osman, Mike Bukowski, Chris McEvoy. Practical implementation of dual paraboloid shadow maps, ACM Siggraph Video Game Symposium, Proceedings of the 2006 ACM SIGGRAPH symposium on Video games.

[Purser04] The Bi-Mercator Grid as a Global Framework for Numerical Weather Prediction, R. James Purser. Presentation at The 2004 Workshop on the Solution of Partial Differential Equations on the Sphere, Organizers: M. Satoh, P., Swarztrauber,, J. Drake, D. Williamson, Yokohama, Japan, July 20-23, 2004

[Sage] Sage, a free open-source mathematics software system, http://www.sagemath.org/

[Segal92] Marc Segal, Carl Korobkin, Rolf van Widenfelt, Jim Foran, and Paul Haeberli. Fast shadow and lighting effects using texture mapping. In Computer Graphics (SIGGRAPH’92 Proceedings) , pages 249–252, July 1992.

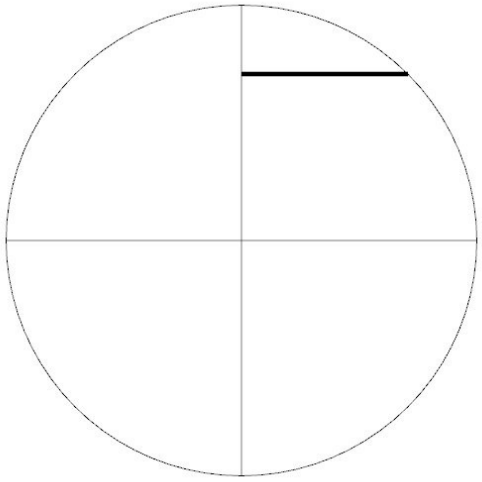
[Snyder01] John Snyder and Don Mitchell. Sampling-efficient mapping of spherical images. Technical report, Microsoft Research, 2001.

[Thompson98] Richard B. Thompson. Designing a Baseball Cover, College Mathematical Journal, Vol 29, No 1, January 1998.

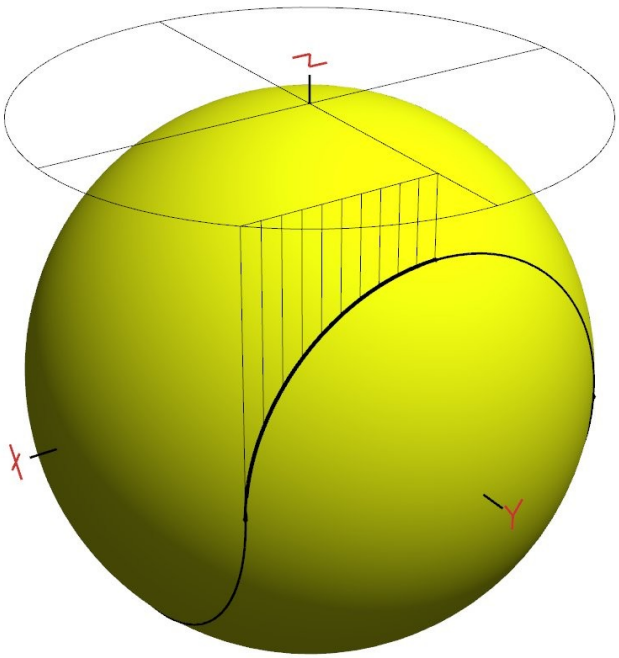
[Voorhies94] Douglas Voorhies and Jim Foran. Reflection vector shading hardware. Proceedings of SIGGRAPH 94 , pages 163–166, July 1994. ISBN 0-89791-667-0.

[Williams78] Lance Williams. Casting curved shadows on curved surfaces. In Computer Graphics (SIGGRAPH ’78 Proceedings) , pages 270–274, August 1978..

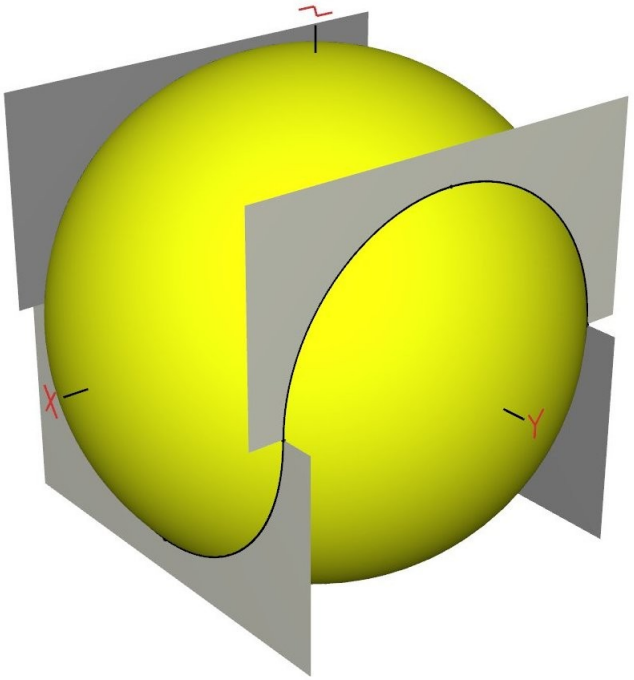
(a)



(b)

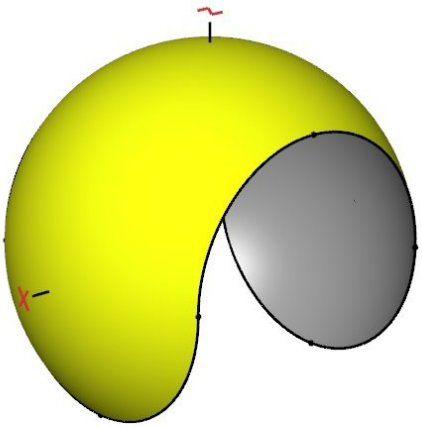


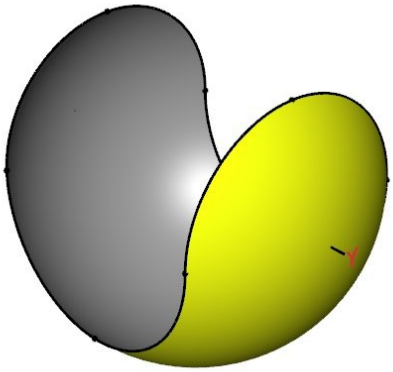
(c)



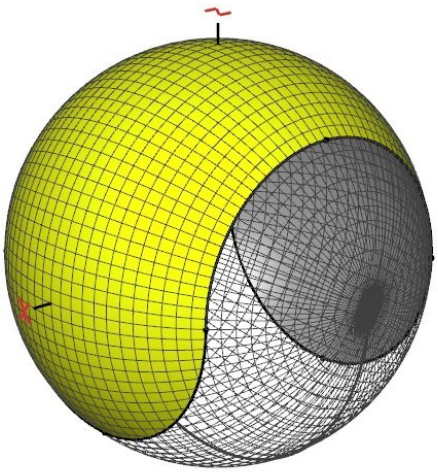
(d)

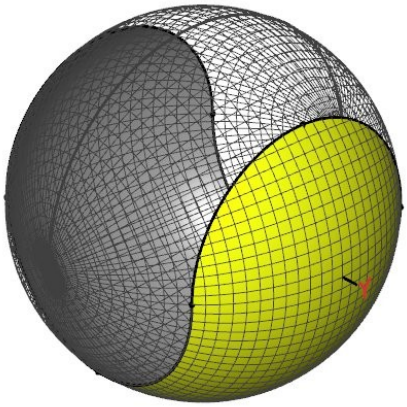
Figure 1: (a) The tennis-ball seam; (b) The 2D curve (straight bold segment), (c) its projection onto a sphere, and (d) the four half planes containing all eight seam segments.



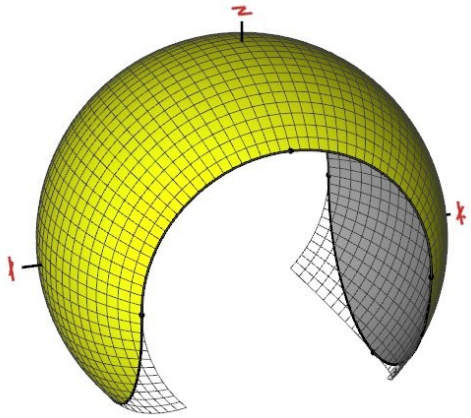


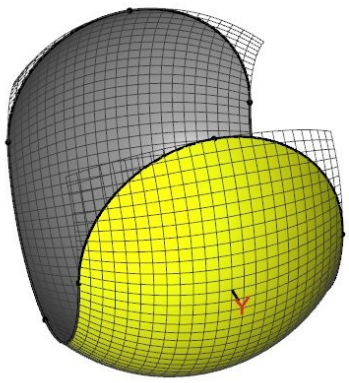
(a)



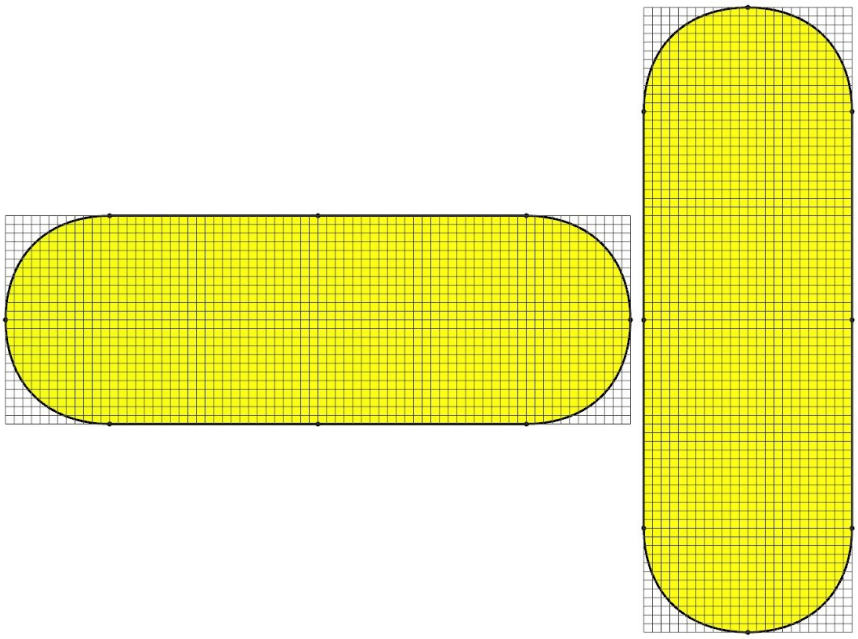


(b)





(c)



(d)

Figure 2: The two components: (a) separated; (b) shown with a full spherical parameterization indicating the position of the poles and seams for each; (c) the parameterization of each restricted to $\pm\pi/4 \times \pm 3\pi/4$ *; and (d) the two components unrolled.*

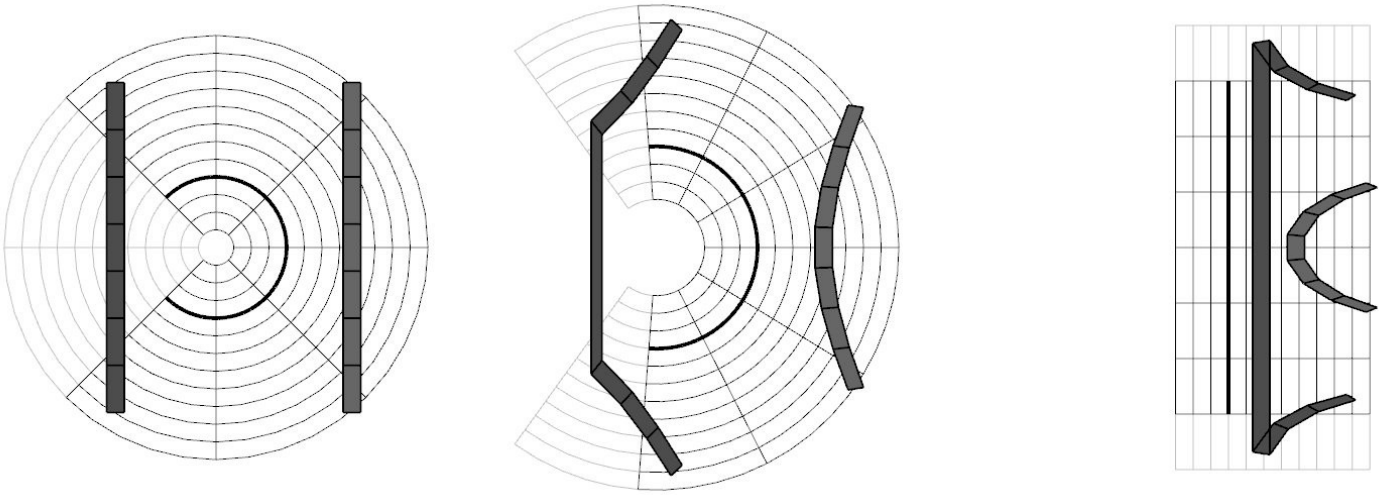


Figure 3a:A sequence of images demonstrating the unwrapping of one tennis-ball component. Left: A 2D slice through a tennis-ball component (dark three-quarter circle), the polar grid of space around it, and a number of shaded polygons embedded in that space. Middle: The beginning of the unwrapping showing the tearing of one polygon across the seam. Right: The final unwrapping to be followed by an orthographic projection onto the component, now a straight line segment. The seam-crossing polygon must be eliminated for the projection to produce a correct image. The offending polygon has endpoints outside the region projecting onto the component, a fact used to recognize and eliminate it.

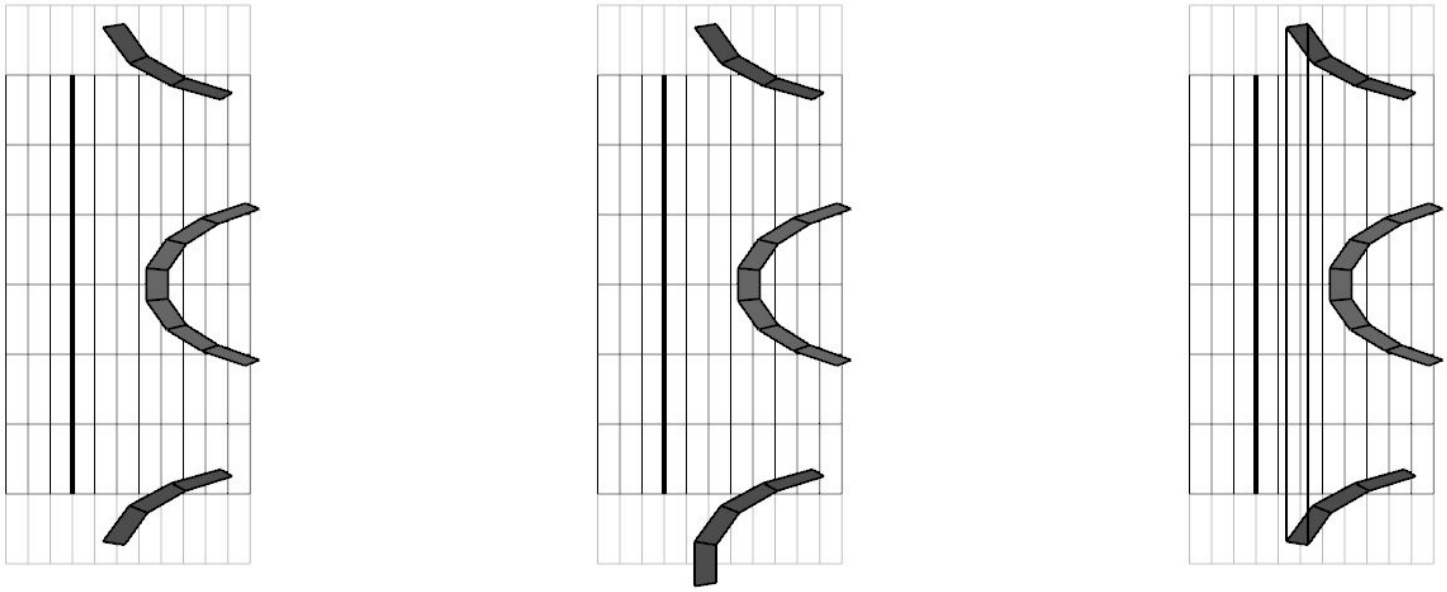


Figure 3b: Three solutions to the seam-crossing polygon of figure 3a. Left: remove the polygon from the graphics pipeline. Middle: transform polygon vertices all to one side of the seam. Right: Chose to discard each pixel of the polygon.