Octahedron Environment Maps

Thomas Engelhardt, Carsten Dachsbacher

Visualization Research Center, University of Stuttgart
Email: {thomas.engelhardt, dachsbacher}@visus.uni-stuttgart.de

Abstract

In this paper, we examine the octahedron as a parameterization scheme to represent environment maps in real-time rendering applications. We discuss two projection schemes and show two convenient unfold mappings to pack the platonic solid into quadratic or rectangular textures. We carry out an analysis and consider rendering performance for interactive applications as well as the quality of representation. We further discuss applications and scenarios which benefit from this parameterization.

1 Introduction

Environment mapping is an established and well known technique widely used in rendering. In general the entire environment seen from one point in space is projected onto a parametric surface and stored in one or more textures which can be easily accessed. It is equally important in interactive rendering as an efficient means to approximate reflections of distant objects onto geometry of arbitrary shape, as well as a means for storing directional data in general. It plays an essential role in illumination algorithms capturing environmental lighting and has found further applications, such as omnidirectional shadow mapping for example.

This paper is structured as follows: First we give a short overview on environment mapping techniques and related work. In Section 3 we discuss the new octahedron representation for environment maps and describe the necessary transformations, followed by Section 4 addressing issues which arise when implementing this technique. In Section 5 we present our results concluding with a short discussion in Section 6.

2 Previous Work

The very first environment map has been introduced by Blinn and Newell [1] who transformed a reflection vector into spherical coordinates to access a two dimensional texture. This method suffers from texture distortions in polar regions and visible texture seams. Williams [12] and Hoffmann [9] introduced spherical environment maps storing an orthogonal view onto a perfect mirror sphere in a texture. This technique was the first to receive hardware support, however, this method exhibits strong non-uniform sampling and a singularity in the viewing direction.

Greene [3] introduced the cube map which matured to the most popular technique used today. It neither suffers from severe under-sampling, nor from singularities. It combines speed and ease of use, both in creation and look up. Above all it lends itself to capturing real world as well as synthetic scenarios naturally.

Because the cube map consists of six textures (which have to be rendered in dynamic scenes) Heidrich and Seidel [5] developed the dual-paraboloid map projecting the environment onto two mirror paraboloids. This approach does not share the drawbacks inherent to spherical environment mapping and no singularities or severe undersampling occurs. The creation of such maps is rather involved and not without limitations, because the underlying projection does not allow direct creation with cameras (no such lenses exist) as cube maps do, and they are not based on a linear perspective projection. That is, either high geometry tessellation to reduce the approximation error or more involved rendering algorithms [2] are necessary.

3 Octahedron Environment Maps

The octahedron as a possible parameterization scheme for the spherical domain has already been

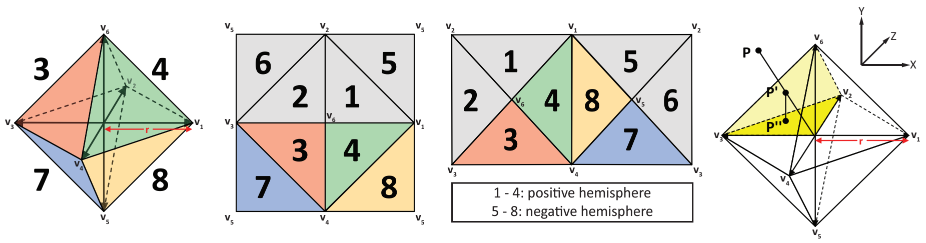

Figure 1: We project the environment on the octahedron, which can be unfolded and packed into a single quadratic or rectangular texture. We show how to directly render into these textures and how to efficiently lookup octahedron environment maps. For the projection, a point p is first projected onto the octahedron faces, and next projected orthogonally on the XZ-plane.

used by Praun et al. [10], who utilized the 1-to-1 mapping of the spherical domain to a 2D texture to store spherical geometry images. We examine the octahedron with regard to the use as environment maps for real-time rendering applications, including the generation of, and look-up to octahedron environment maps (OEMs) with graphics hardware (see Figure 7 for an example application).

Each side of the octahedron represents one octant of directions and we describe two projections to map directions to points on the sides. Further, the octahedron sides may be arranged differently in two-dimensional textures (see Figure 1).

In the following, we assume that all coordinates are given relative to the octahedron's coordinate system as illustrated in Figure 1 (right). This of course can be easily achieved setting up the appropriate transformations during rendering.

3.1 Planar Projection

The first projection scheme that we analyze uses a projection similar to a spherical projection. A point $p = (p_x, p_y, p_z)^T$ lies on the sphere with radius r, if $p_x^2 + p_y^2 + p_z^2 = r^2$. Similarly, for a octahedron with vertices $(\pm r, 0, 0)^T$, $(0, \pm r, 0)^T$, $(0, 0, \pm r)^T$, a point resides on the surface of the platonic solid, if $|p_x| + |p_y| + |p_z| = r$. Thus, we derive this projection scheme, which we denote as *planar projection*, for a point p with (see also Figure 1):

$$p' = \frac{p}{|p_x| + |p_y| + |p_z|} \tag{1}$$

To unfold the octahedron into a quadratic texture, points on the positive hemisphere, i.e. $p_y > 0$, are orthogonally projected onto the XZ-plane (Figure 1). The lower hemisphere is unfolded by splitting all edges adjacent to $(0, -r, 0)^T$. Hence the projected points p'' are obtained by:

$$p''_q = \begin{cases} (p'_x, 0, p'_z)^T & p'_y \geq 0 \\ (\sigma(p'_x)(1 - \sigma(p'_z))p'_z, 0 \\ \ \sigma(p'_z)(1 - \sigma(p'_x))p'_x)^T & p'_y < 0 \end{cases} \tag{2}$$

where $\sigma(x)$ is the sign function.

Our second unfolding approach maps the octahedron to a rectangular texture with an aspect ratio of $2 : 1$ as shown in Figure 1. First equation 1 is applied, then the points p' are projected orthogonally on the XZ-plane and the coordinates from the 2 hemispheres are mapped into the layout by following transformations.

$$p''_r = \begin{cases} (p'_x - p'_z - 1, 0, p'_x + p'_z)^T & p'_y \geq 0 \\ (p'_z - p'_x + 1, 0, p'_x + p'_z)^T & p'_y < 0 \end{cases} \tag{3}$$

According to the layout mapping $p''_r \in [-2; 2] \times [-1; 1]$ and $p''_q \in [-1; 1]^2$ have to be mapped to the respective texture coordinates for access and rendering.

3.2 Perspective Projection

In allusion to the cube map the perspective projection can be thought of as another means to obtain an octahedron environment map. The scene can be rendered once for each octahedron side, each set up with its own projection matrix. In this case, the frusta - in contrast to the cube mapping - are of tetrahedral shape, i.e., we have a triangular view port, for which clipping is not natively supported by graphics hardware and has thus to be performed manually, either in object or image space.

The look up to octahedron maps using these projections works analogously to cube maps: First the octant of a look up vector has to be determined, and next the the projection transformation and the mapping to the octahedron layout (as described above) is computed.

We examined this parameterization for the sake of completeness, but compared to a cube map, there are no advantages to be expected (8 projections instead of 6, plus additional clipping). Thus, in the remainder of this paper, we focus on the planar parameterization, which is simpler and thus beneficial for some applications.

4 Implementation

4.1 Direct Rendering of Octahedron Maps

The planar projection cannot be expressed with the matrix machinery, as the absolute value (see Eq. 1) cannot be expressed accordingly, however it can be easily used with programmable graphics hardware. Furthermore, triangles overlapping octant boundaries cannot be handled correctly: Transformations and projections on GPUs are performed per-vertex and the scan-line conversion or rasterization stage (assuming linear triangle edges) produces wrong results whenever a triangle intersects more than octant (illustrated in Figure 2). Please note, that this is somewhat similar to the projection errors with paraboloid mapping. Similarly, we can keep the error low by using finely tessellated geometry, but in order to produce correct results additional effort is necessary, which is described in the following.

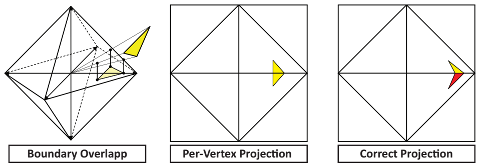

Figure 2: Ignoring octant boundaries leads to incorrect projections.

Linear interpolation is correct within each octant, i.e., we can solve the problem by splitting triangles at octant boundaries. We can leverage this by using geometry shaders: A triangle is split at the XY-, YZ-, ZX-plane into smaller triangles (up to 27), such that each output triangle resides in a single octant only. After the clipping, the projected coordinates are computed and the affine transformation according to the layout (quadratic or rectangular) is applied. Please observe that for point-based rendering techniques special octant boundary treatment may not be mandatory, if the error is neglectable, due to the small size of point sprites.

4.2 Texture Look-ups

Unfiltered Texture Lookup The texture coordinates for the lookup into an octahedron map for a given direction vector can be computed analogously to the aforementioned projection and layout transformation. The unfiltered lookup is very efficient when used together with the quadratic layout. Computing the texture coordinates for a direction vector d given in the octahedron coordinate system can be achieved with the following few instructions (shader pseudo code):

```
 // projection onto octahedron
d /= dot( 1, abs(d) );
// out-folding of the downward faces
if ( d.y < 0.0f )
 d.xy = (1-abs(d.zx)) * sign(d.xz);
// mapping to [0;1]ˆ2 texture space
d.xy = d.xy * 0.5 + 0.5;
color = tex2D( octaMap, d.xy );
```

Filtered Texture Lookup For a mip-mapped lookup, we need to consider two aspects. First, when determining the required mip-level for a texture lookup, the decision is based on the extend of the quadrilateral region spanned by the screen space pixel cell transformed into texture space. It is computed from the texture coordinate and its partial derivatives [8]. Please note, that for per-pixel computed reflections, the mip-level is typically computed manually depending on surface curvature or glossiness. We denote the partial derivatives in texture space by dX and dY. The mip map level l is then derived by:

$$\begin{aligned} e &= \max(\langle dX, dX\rangle, \langle dY, dY\rangle) \\ l &= \log_2 \sqrt{e} \end{aligned} \tag{4}$$

However, in the case of the octahedron map the quadrilateral region might cross two adjacent octants which have been separated due to the unfolding. For the quadratic layout, this case occurs when the quadrilateral resides in the negative hemisphere. A possible situation is depicted in Figure 3 (center) for the quadratic layout, but similar configurations occur for the rectangular layout as well.

Thus, we need a special treatment when a cell spans two disjoint regions in the texture space. Fortunately, the symmetry of the octahedron allows us to reflect the coordinates of pixel cells in the lower to the upper hemisphere, thus resulting in a contiguous quadrilateral which is then used to compute the correct mip-level (Figure 3, right). In practice, we compute the mip-level of both, the original and the mirrored cell, and use the minimum of both values.

For the rectangular layout both hemispheres are mapped to contiguous regions. Thus, when *texture arrays* (supported by DirectX 10) are used – one texture per hemisphere – then filtering is automatically handled correctly, also for the per-vertex texture coordinate computation. If no texture arrays are used, then similar treatment as described above for the quadratic layout is necessary.

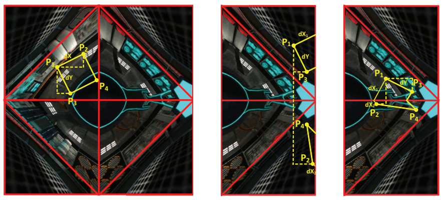

Figure 3: The quadrilateral region in texture space corresponding to a pixel cell in the negative hemisphere is mapped to disjoint regions (center). This yields to incorrect mip-levels, however, by exploiting symmetry we reflect at the XZ-plane and obtain the correct area estimate for mip map level determination (right).

The second aspect for a filtered texture look-up is that due to the unfolding a wrap-around access to the texture does not yield correctly bi-linearly interpolated color values (this applies to parabolic, and to some extend to cube maps as well). This can be best solved by introducing a ”safety-border”, that is by duplicating pixels as shown in Figure 4 such that samples for bi-linear interpolation and mipmapping are available.

Please note, that the maximum correctly represented mip-level depends on the width of the safety-border. We also used octahedron maps with summed area tables for rendering glossy reflections [6]. In this case we used a safety-border width determined by the maximum glossiness of the BRDF (Figure 7 shows an example). Please note, that for accurate BRDF modeling, multiple SAT queries are required which holds for all spherical parameterizations and (pre)filtered environment mappings.

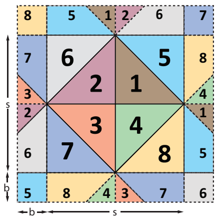

Figure 4: The OEM texture of dimensions $s \times s$ is extended with a texture border of width b

5 Results

5.1 Sampling

In order to evaluate the sampling quality of the octahedron map, we compute the solid angle covered by each pixel (relative to the solid angle of a sphere over the total number of pixels). The comparison of the octahedron parameterization to a cube map is shown in Figure 5. The highest sampling rate is close to the octahedron vertices, but stays within an acceptable range across the entire surface. As expected, the planar projection yields a more balanced sampling than the perspective projection; the overall quality is comparable to a cube map.

5.2 Direct Rendering

We further measured the rendering performance for an adequately complex scene (Figure 8) creating three environment maps of approximately equal pixel count. We directly render a 512×512 quadratic layout *(QL)* OEM, a rectangular layout (*RL*) OEM of two 362×362 textures and a cube map of six 209×209 textures. The timings are listed in Table 1. We have implemented both, the planar and perspective projection with explicit clipping in the geometry shader stage as described in Section 4.1. We denote this approach, which generates up to 81 vertices, *strict split*. As mentioned in Section 3.2 the clipping for perspective OEMs can

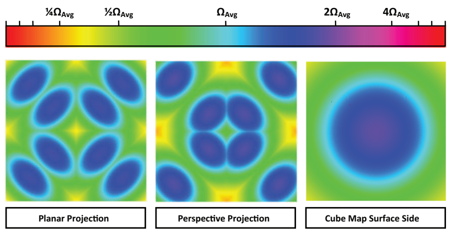

Figure 5: Dark blue and purple regions indicate areas of undersampling while regions of green to yellow indicate areas of directional oversampling.

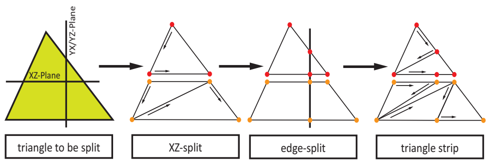

Figure 6: The triangle is split at the XZ-plane. The convex polygons obtained in the positive (red vertices) and negative (orange vertices) half-space are either directly triangulated or after the boundaries have been split by all remaining split planes.

also be done in image space by discarding pixels that do not fall into the triangular viewport. We call this method *clip mask*.

For the rectangular layout we present results for the planar projection and only examine two less accurate clipping techniques: The clipping in the geometry shader whose performance heavily depends on the amount of vertices output to the succeeding pipeline stages proved to be the bottleneck in our implementation. Exploiting the fact that both hemispheres are mapped to contiguous regions we only split triangles mapped to both textures due to intersecting the XZ-plane. We denote this approach, which outputs two triangle strips of at most 7 vertices, *XZ-split* (Figure 6, *XZ-split*).

The second approach, called *edge strip*, proceeds with the vertices created by the XZ-split in both half-spaces (positive and negative y-axis) independently. Those vertices form convex polygons whose edges are split by the remaining split planes (Figure 6, *edge-split*). Afterwards two triangle strips are directly created from the obtained points (Figure 6, *triangle strip*). Please note, that while this naïve stripping preserves the correctness of the projection on both polygon boundaries, it does not guarantee correct interpolation, e.g. of texture coordinates, within the triangle strips. Of course this is not correct, but rather tolerable than a wrong primitive shape. This approach outputs a maximum of 15 vertices in total.

5.3 Applications and Advantages

Despite the difficulties introduced with the direct rendering of OEMs, they provide two unique features. First, OEMs have the ability to generate summed area tables (SATs) over the the entire spherical domain. When creating SATs on GPUs, e.g. with the method by Scheuermann et al. [6], the input and output parameterization can be chosen independently. This is because multiple render passes are required for the generation and the first one can include a reparameterization, i.e. also easily renderable cube or paraboloid maps can be used as input. Furthermore, a lookup to an OEMs is an efficient means to make directional data available to all shader stages. For vertex or geometry shaders there is no support for cube maps and the OEM lookup presented in Section 4.2 is more efficient than any lookup to cube or paraboloid maps computed by hand.

The octahedron parameterization also provides compact storage when multiple (hemi-)spherical domains are to be stored, e.g. (hemi-)spherical shadow maps for instant radiosity methods [7] or for sparse-sampling of scene radiance [4]. Instead of parabolic parameterizations which waste up to 20 percent of texture space, OEMs can be packed without wasting texture space into texture atlases.

As OEMs, latitude-longitude environment maps allow to parameterize the spherical domain to a rectangular texture, and consequently, they would also allow to compute SATs, provide compact storage, and lookup in vertex and geometry shaders. However, the sampling is worse and exhibits two singularities in polar regions, the lookup requires trigonometric, and thus expensive, operations and rendering to latitude-longitude maps requires clipping and splitting as well.

6 Conclusions

We examined octahedron environment maps as an alternative for environment map rendering which represent the entire spherical domain of directions in a single quadratic or rectangular texture while providing efficient texture look-ups. Although the

EnvMap	Clipping	FPS[s]
Planar QL	strict split	16
Planar RL	XZ-split	34
Planar RL	edge strip	3
Perspective QL	strict split	16
Perspective QL	clip mask	95
Cube Map		250

Table 1: Rendering performance of our test scene for a screen resolution of 1920×1200. All timings were taken on an Intel Q6600 Core2Duo processor with 2.4GHz, a GeForce 8800GTX running Vista x64 with 4GB of main memory.

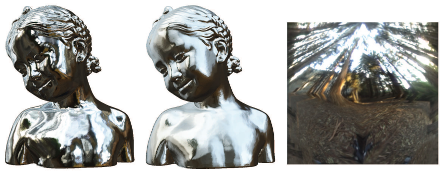
Figure 7: Screenshot of a rendering with SATs created from octahedron environment maps to adjust surface glossiness at real-time. The creation of the 512×512 pixel SAT takes about 3.3ms on a NVIDIA 8800GTX.

planar projection and unfolding transformations are rather simple and provide good sampling (comparable to the cube map), the direct rendering is not yet suitable for interactive applications due to the geometry shader performance.

Nonetheless OEMs offer benefits not known from other techniques, namely the ability to create spherical summed area tables, efficient look-ups in all shader stages on GPUs, and tight-packing into texture atlases.

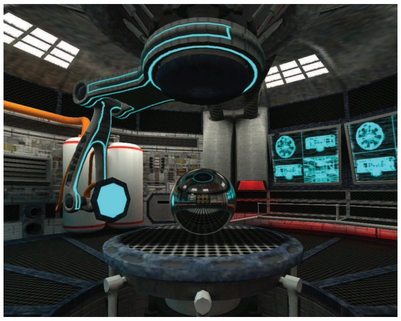
Figure 8: Our test scene of approximately $105k$ triangles for dynamic environment map generation.

References

[1] James F. Blinn and Martin E. Newell. Texture and reflection in computer generated images. *Communications of the ACM,* 19(10):542–547, 1976.

[2] Jean-Dominique Gascuel, Nicolas Holzschuch, Gabriel Fournier and Bernard Peroche. Fast Non-Linear Projections using Graphics Hardware. *ACM Symposium on Interactive 3D Graphics and Games,* 2008

[3] Ned Greene. Environment Mapping and Other Applications of World Projections. *IEEE Computer Graphics and Applications,* 6(11):21–29, 1986.

[4] Gene Greger, Peter Shirley, Philip M. Hubbard and Donald P. Greenberg. The Irradiance Volume. *IEEE Comput. Graph. Appl.,*18(2):32–43, 1998.

[5] Wolfgang Heidrich. View-Independent Environment Maps. *Proceedings of Eurographics/SIGGRAPH Workshop on Graphics Hardware '98,* 1998.

[6] Justin Hensley, Thorsten Scheuermann, G. Coombe, M. Singh, and Anselmo Lastra. Fast Summed-Area Table Generation and its Applications. *Eurographics '05,* 2005

[7] Samuli Laine, Hannu Saransaari, Janne Kontkanen, Jaakko Lehtinen and Timo Aila. Incremental Instant Radiosity for Real-Time Indirect Illumination. *Proceedings of Eurographics Symposium on Rendering 2007,* 2007

[8] Microsoft Corporation. DirectX Software Development Kit 10.

[9] Gene S. Miller and C. Robert Hoffman. Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments. *SIGGRAPH '84 Advanced Computer Graphics Animation course notes,* 1984.

[10] Emil Praun and Hugues Hoppe. Spherical parametrization and remeshing. *ACM Trans. Graph.,*22(3):340–349, 2003.

[11] Nicolai Steiner. Pyramidal Environment Maps. *Diploma Thesis, University of Stuttgart,* 2008.

[12] Lance Williams. Pyramidal Parametrics. *SIGGRAPH Comput. Graph.,* 17(3):1–11, 1983.