Interactive Refraction on Complex Static Geometry using Spherical Harmonics

Olivier Génevaux* Frédéric Larue* Jean-Michel Dischler*

LSIIT UMR CNRS-ULP 7005
Université Louis Pasteur Strasbourg I, France

Abstract

Accurate refraction, thanks to raytracing, has always been a popular effect in computer graphics imagery. However, its use has been severely hindered in interactive rendering due to the lack of efficient and realistic techniques geared toward polygon oriented rendering.

In this paper, a method to achieve realistic and interactive refractive effects through complex static geometry is proposed. It relies on an offline step where many light paths through the object are pre-evaluated. During rendering, these precomputed paths are used to provide approximations of actual refracted paths through the geometry, enabling further sampling of an environment map. Light paths valuable information, namely final output direction when leaving refractive object, is compressed using frequency domain based spherical harmonics. The matching decompression procedure, entirely offloaded onto graphics hardware, is handled at interactive speed.

CR Categories: I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Color, shading, shadowing, and texture

Keywords: refraction, real-time rendering, spherical harmonics

1 Introduction

Refraction, bending light rays traveling through an interface between two transparent media of different densities, is one of the most prominent optical effects to be seen. However, its simulation in computer graphics remains challenging. Indeed, resulting effect is complex since light rays are usually deflected more than once. Such an effect can easily be simulated on complex objects using raytracing. Nevertheless, interactive framerate cannot be reached on commodity hardware.

The method proposed in this paper is designed to interactively deal with multiple bounces refraction on a large class of complex geometries. It is devised to be tractable on a hardware accelerated feed-forward pipeline. This requires to simplify the refraction model. The first simplification takes care of the environment surrounding the object. As commonly done in hardware rendering, environment is assumed to be far enough to be handled as a cube-map. As a consequence, parallax effects are neglected. The second approximation enables to drastically decrease computations as well as the amount of data. Fresnel laws induce both a reflected and a refracted ray at each encountered interface. Retaining only the most significant path of the whole ray tree reduces the number of involved rays.

Figure 1: Refractive shell and its bead. Light deflection is visible on all the mesh, even on self-occluded parts.

Similar to surface lightfields, the proposed technique stores numerous directional data on the surface of the object. What is stored is a mapping between the viewing direction and the final exiting ray direction all over the surface. This mapping, evaluated using raytracing during a precomputation step, is stored in a compact way using spherical harmonics. It is then used during interactive rendering to index the environment map.

Previous work is summarized in section 2. Then, principles of the method are given in section 3. Spherical harmonics, used to handle the data view dependence, are reviewed in section 4. The decompression step, using current programmable graphics hardware, is described in section 5. Finally, results are discussed in section 6.

2 Previous Work

Simulating refraction is highly dependent on both the hidden surface removal procedure used and the balance between the desired level of accuracy and the rendering speed.

The only method where refraction is truly easy to simulate is raytracing [Whitted 1980]. Indeed, due to its very nature, where rays are traced back from the camera into the scene, correct refraction can be handled without any difficulty: refractive rays are simply spawned as necessary, enriching the ray tree. However, this simplicity is only achieved when objects are described using smooth formulations. In the case of discretely sampled volumetric objects, more complex procedures are required to obtain high quality results [Li and Mueller 2005]. The more elaborate beam tracing technique exhibits the problem of describing refraction on finite area

*e-mail:{genevaux, larue, dischler}@dpt-info.u-strasbg.fr

I3D 2006, Redwood City, California, 14–17 March 2006.
© 2006 ACM 1-59593-295-X/06/0003 $5.00

surfaces. Addressing this problem might require approximating refraction [Heckbert and Hanrahan 1984].

If interactive rendering is desired, raytracing proves impractical on commodity hardware. Polygon rendering processes are then preferred, simplifying the refraction model, thus trading accuracy for rendering speed. Refraction on thin objects has been approximated using a distortion field applied to the scene background [Kay and Greenberg 1979]. The same distortion field approach has also been used to approximate complex virtual lenses behavior [Heidrich et al. 1997]. Lifting the thin object restriction, a hardware accelerated technique has been developed [Wyman 2005], where two-sided refraction on arbitrary geometry is approximated. This technique provides realistic effects, even with objects of strongly varying depth. However, only able to deal with double refraction, accuracy strongly depends on the convexity of the object.

Matting algorithms have been developed [Zongker et al. 1999] to capture refraction occurring inside real objects. The goal is to allow further relighting using the exact same viewpoint.

Instead of considering arbitrary scenes, methods have been devoted to specific scenes or particular objects. Refraction through a single surface, such as an ocean surface viewed from above sea level, can be devised analytically at mesh vertices, to further index some texture map figuring the sea bottom [Ts'o and Barsky 1987]. Multiple effects, including refraction, occurring in gemstones, can be faithfully rendered using hardware assisted beam tracing [Guy and Soler 2004]. However, striving for accuracy, the method is tailored to handle small models composed of a few hundredth planar faces. Among polygon based rendering techniques, environment mapping can be avoided. Yet, multiple rendering passes have to be performed [Diefenbach and Badler 1997]. Each pass then handles refraction through a single planar surface. Due to complex handling of refraction with polygonal techniques alone, hybrid approaches combining polygon rendering and raytracing have been proposed. They mainly operate full raytraced refraction computations at mesh vertices, combined with an adequate in-between polygon filling procedure. To keep framerate at interactive level, dedicated hardware may be used [Ohbuchi 2003] or an adaptive mesh tesselation procedure can be used [Hakura and Snyder 2001].

If parallax effects are desired, elaborate techniques have been proposed, such as representing the neighboring environment using lightfields [Heidrich et al. 1999; Yu et al. 2005].

Outside of the previous techniques, completely simulating refraction during rendering, a method heavily relying on precomputation has been proposed [Heidrich et al. 1999]. Trading memory consumption for interactivity, refraction induced patterns are accurately precomputed and stored, allowing further interactive rendering. The view-dependence of this information is handled using a lightfield framework. However, even using vector quantization, refraction can only be stored coarsely due to memory constraints, lowering quality. Moreover, given the two planes parameterization used for the lightfield, viewpoint freedom is restrained.

Using the same rationale, surface lightfields [Chen et al. 2002] uses an optimized scheme to store exiting radiance directly on the object surface. Complex objects are then captured in a fixed lighting environment and rendered for any viewpoint afterwards. Going further, precomputed radiance transfer [Sloan et al. 2002] represents a transfer function on the surface to capture self-shadowing and interreflections. Using spherical harmonics representation for both the transfer functions and the light stage, environment can be interactively and efficiently varied.

Building upon these ideas, the method presented in this paper stores directional geometric informations directly on surface, instead of photometric quantities. Similar to Sloan [2002], the directional information stored on the surface is represented using spherical harmonics. Yet, actual handling of spherical harmonics differs significantly: basis functions are explicitly evaluated instead of capitalizing upon basis orthogonality.

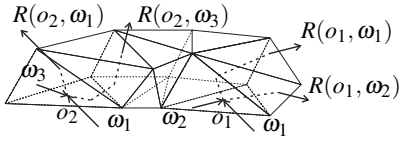

Figure 2: Refracted path mapping.

3 Method Principles

The presented method falls in line with most of the polygon based rendering methods designed to handle refraction. It makes the assumption that the refractive object can be completely uncoupled from its environment, considered as a distant environment map.

Under this assumption, all light paths interacting with the refractive object share the same common structure. A sight ray, leaving the virtual camera lens, gets deflected several times by the interface between the object and the environmental media, finally hitting the distant environment. According to Fresnel equations, both reflected and transmitted rays should be spawned at each encountered interface [Guenther 1990]. Instead of handling the whole resulting ray tree, which would prove impractical interactively, only the most significant branch is retained. The representative path is chosen greedily at each interface, between reflected and transmitted candidates, depending on their Fresnel attenuation factors [Hakura and Snyder 2001]. Note that retained path might interact with the object interface many times before finally leaving the object.

This single path approximation enables to establish a refracted path mapping. As stated in equation 1, it is defined over the whole object surface O and the whole viewing direction sphere S^2. For any point $p \in O$, and any viewing direction $\omega \in S^2$, it stores the exiting direction of the path entering in the object at p along ω, as shown by figure 2.

$$R(p,\omega) : O \times S^2 \rightarrow S^2 \tag{1}$$

Capturing this mapping requires to sample it. The set of samples $O_d = \{o_i\} \subset O$ spanning the object surface depends on the nature of the handled object. The choice of this set is addressed in section 5. The set of samples $S_d^2 = \{\omega_j\} \subset S^2$ spanning the viewing sphere is described in section 4.2. An interpolation $\hat{R}(p,\omega)$ of these sampled data is then required. Similar to lightfield mapping, the four-dimensional mapping R is expressed as a sum of products of lower dimensional functions as stated by equation 2.

$$R(p,\omega) \approx \hat{R}(p,\omega) = \sum_{o_i \in O_d} S_{o_i}(p) \cdot R_{o_i}(\omega) \tag{2}$$

At each sample o_i, the mapping $R_{o_i} = R(o_i,\cdot)$ is represented using spherical harmonics. Such a frequency scheme has been chosen because of its spherical nature and its compactness. Moreover, note that each compressed function R_{o_i} is intrinsically continuous and does not explicitly interpolate between directional samples S_d^2. Such a representation acts as a an efficient compression scheme too. The S_{o_i} are defined as linear or bilinear interpolation functions between surface sampling locations $o_i \in O_d$.

Given this mapping R, simulating refractive effects for any viewpoint q is immediate. It reduces to query $\hat{R}$ with adequate viewing

direction ω_q on the whole surface and to sample the environment accordingly.

The proposed method then proceeds along these lines. The mapping R is sampled during a precomputation step using raytracing. At each spatial sample $o_i \in O_d$, it is further compressed using spherical harmonics, leading to the R_{o_i} functions. Then, interactive rendering is performed using a two-steps procedure. First, a refraction map $\mathcal{R}_{\rho,q}$ matching the current viewpoint q is extracted from the compressed data. Its layout is defined according to a parameterization ρ of the spatial samples set O_d into texture space. In essence, this map stores how light is refracted through the object viewed from q, for all the object surface. The parameterization ρ is chosen so that hardware texture filtering embodies the spatial interpolation functions S_{o_i}. Formally, the refraction map $\mathcal{R}_{\rho,q}$ is defined as $\mathcal{R}_{\rho,q}(\rho(o_i)) = R_{o_i}(\omega_q)$. Final rendering is then performed by sampling the environment with respect to the directions stored in $\mathcal{R}_{\rho,q}$.

To adequately capture the refractive object behavior, dense sampling of the four dimensional mapping R is required. Yet, the spherical harmonics representation is efficient enough to keep memory usage tractable. Both this compression scheme and the matching hardware implementation allow to provide interactive and plausible rendering of refractive effects on complex geometries. However, reliance on a precomputation step induces two major limitations: object geometry cannot be modified at runtime and material densities are held fixed. Moreover, the use of frequency based compression tends to smooth data. This prevents discontinuities in the refraction pattern to be accurately captured, reverting to smoother variations. Furthermore, relying on fixed texture filtering restrains S_{o_i} to piecewise linear interpolation functions.

4 Spherical Harmonics

As previously stated, at each spatial sample $o_i \in O_d$, the refracted path mapping R_{o_i} is independently compressed using spherical harmonics (SH) [MacRobert 1967]. It should be noted that the SH use differs significantly from its common use in rendering. Frequently, the orthogonality property of the basis is capitalized upon to reduce the convolution operation to a single dot product [Sloan et al. 2002]. In this paper, no convolution is dealt with, and direct basis functions evaluation is required.

4.1 Definition

Spherical harmonics, denoted Y_l^m, constitute a family of increasing frequency functions, defined over the sphere S^2. As spherical harmonics form a complete family of orthogonal polynomials, any square-integrable function φ can be expressed as a linear combination of the Y_l^m basis functions . The involved α_l^m weights are defined as the convolution of the compressed function with the desired Y_l^m basis function over the sphere, as stated in equation 3.

$$\varphi = \sum_{l=0}^{\infty} \sum_{m=-l}^{l} \alpha_l^m \cdot Y_l^m \qquad \alpha_l^m = \int_{S^2} \varphi \cdot Y_l^m ds \tag{3}$$

The Y_l^m functions are complex valued functions defined for each direction (θ, ϕ), using the associated Legendre polynomials P_l^m and a normalization constant K_l^m, shown in equation 4.

$$Y_l^m = Y_l^m(\theta,\phi) \quad = \quad K_l^m P_l^m(\cos\theta) e^{im\phi}$$

$$K_l^m \quad = \quad \sqrt{\frac{2l+1}{4\pi}\frac{(l-m)!}{(l+m)!}} \tag{4}$$

Despite being compact and recursively computable, this formulation of spherical harmonics is not suitable to be used in a hardware supported decompression algorithm since it makes use of angular quantities and trigonometric functions. However, recalling the sphere parameterization in use, described in equation 5, it is possible to rewrite basis functions using Cartesian coordinates, leading to equation 6.

$$[x,y,z] \leftrightarrow [\sin\theta\cdot\cos\phi, \sin\theta\cdot\sin\phi, \cos\theta] \qquad \begin{cases} \theta \in [0,\pi] \\ \phi \in [0,2\pi[\end{cases} \tag{5}$$

$$Y_l^m = Y_l^m(x,y,z) = K_l^m \sum_{p,q,s} \frac{1}{p!q!s!} \left(-\frac{x+iy}{2}\right)^p \left(\frac{x-iy}{2}\right)^q z^s \qquad \begin{cases} p+q+s \geq 0 \\ p+q+s = l \\ p-q = m \end{cases} \tag{6}$$

Using this new formulation, all basis functions are seen to be polynomials of the x, y, and z variables.

Since real-valued rather than complex-valued functions are going to be represented, it is possible to define simpler y_l^m real-valued spherical harmonics, using equation 7.

$$y_l^m = \begin{cases} \sqrt{2}\, Re(Y_l^m) & m > 0 \\ \sqrt{2}\, Im(Y_l^m) & m < 0 \\ Y_l^0 & m = 0 \end{cases} \tag{7}$$

4.2 Practical Compression

The projection on the spherical harmonics basis must be done on the many sampling points distributed over the object surface. An efficient way to perform the compression is therefore needed. At each point o_i, the mapping R is only known on sampled directions $\omega_j \in S_d^2$. It is assumed that the mapping is continuously interpolated in-between thanks to interpolation functions B_j as shown in equation 8. In this paper, regular subdivision of the two angular parameters of S^2 was used to define S_d^2. The B_j functions then naturally arise from bilinear interpolation in each quadrangular region of the sphere.

$$R(o_i,\omega) = R_{o_i}(\omega) = R_{o_i}(\theta,\phi) = \sum_j r_j \cdot B_j(\theta,\phi) \tag{8}$$

Considering this definition, weights w_l^m can be related to samples values $r_j = R_{o_i}(\omega_j)$, leading to equation 9.

$$w_l^m = \sum_j r_j \cdot \int_{S^2} y_l^m \cdot B_j \, ds \tag{9}$$

As expressed in equation 10, the weights vector $\mathscr{W}$ can be directly related to the samples vector $\mathscr{R}$. Since the matrix $\mathscr{M}$ is only dependent upon the samples distribution S_d^2 spanning the sphere, it can be accurately precomputed and reused across all hemispheres as long as distribution is left unchanged.

$$\mathscr{W} = \mathscr{M} \cdot \mathscr{R} \qquad \mathscr{M}_{ij} = \int_{S^2} y_i \cdot B_j \, ds \tag{10}$$

Compressing a sampled function is hence reduced to a single matrix-vector product. In the case at hand, the same compression

```
if first pass
   s = 0
else
   s = fetch current partial sum value

[x,y,z] = get local viewing direction

for all polynomials y_i assigned to the pass
   p_i = evaluate y_i(x,y,z)
   w_i = fetch weight matching y_i
   s = s + w_i · p_i
                                   Shader generation control
output s                           Actual shader instructions
```

Figure 3: Decompression shader structure.

matrix is reused over all samples $o_i \in O_d$, performing sampling in local tangent frames. Instead of sampling both sides of the object surface, only outer viewpoints are considered. This imply that only the hemisphere above the surface is sampled. The remaining of the sphere is completed by mirroring existing data [Sloan et al. 2003]. The output direction is represented using Cartesian coordinates. Each coordinate is compressed on its own, leading to a SH coefficients vector for each component.

4.3 Deficiencies Handling

Since the infinite series described in equation 3 is necessarily truncated in practical applications, it is not possible to exactly encode the refracted path mapping and the highest frequencies are lost. However, abruptly discarding coefficients above some given frequency, hence applying a box filter in the frequency domain, results in spurious oscillations of the reconstructed function, called the Gibbs phenomenon. To attenuate this problem, it is possible to apply a semi-Gaussian low-pass filter instead of a box filter in the frequency space. Directly updating spherical harmonics weights, this smoothing is of no cost. The width of the filter is empirically determined to balance ringing suppression against excessive blurring of the reconstructed function [Westin et al. 1992].

Lossy compression using SH might result in noise in the rendered refraction, especially if surface sampling is dense. To alleviate such a problem, it is possible to further smooth reconstructed values. While some pointwise accuracy may be lost during smoothing, visual results are much more pleasing while retaining their overall appearance. The effects of both smoothing filters, the frequency one and the spatial one, are demonstrated in figure 4.

5 Implementation

Data gathered in the refraction map $\mathcal{R}_{\rho,q}$ arise from the $R_{o_i}(\omega_q)$ evaluation for the current viewpoint q, based on SH decompression. Yet, decompressed values are not directly used, but processed beforehand. The first processing prevents defects related to texture filtering and the second one applies smoothing described in section 4.3.

The raw SH decompression is first performed into a directional map denoted $\mathcal{D}_{\mu,q}$. The refraction map $\mathcal{R}_{\rho,q}$ is then the result of the processing chain applied to $\mathcal{D}_{\mu,q}$. Since these two maps may not share the same layout, they use two ρ and μ parameterizations.

Figure 4: Effects of filtering: raw decompression, semi-Gaussian smoothing filter on SH weights and SH weights filter combined with spatial smoothing.

Two different rendering algorithms are further detailed. The first one deals with surfaces requiring no, or weak, refraction map continuity at their borders, as opposed to the second one, designed to cope with more stringent continuity constraints. Both start with the directional map $\mathcal{D}_{\mu,q}$, whose construction is detailed first, and end by providing the refraction map $\mathcal{R}_{\rho,q}$ used during final rendering.

5.1 Directional Map Building

In order to build the directional map $\mathcal{D}_{\mu,q}$ from the weights computed during compression, one has to expand the frequency series described in equation 3. This must be done for each surface sample $o_i \in O_d$, using the proper local viewing direction $\omega_{o_i}(q)$ defined by the current viewpoint q.

Recalling spherical harmonics Cartesian definition given in equation 6, decompression is reduced to evaluate the weighted sum of the SH basis functions. Due to the limited workload that can be carried through a programmable hardware rendering pass, multipass rendering is necessary to achieve reconstruction as soon as the number of basis functions increases beyond some modest figure. Indeed, decompression requires a large number of primitive instructions and a consequent number of different textures. Each pass, handling a subset of the basis functions, is run on all the map $\mathcal{D}_{\mu,q}$, updating all samples simultaneously. Using the closed form of the basis functions, an automatic shaders generation procedure symbolically computes all involved polynomials, and distributes them among passes. Distribution is governed by a greedy algorithm based on polynomials computation cost. Such a procedure helps to avoid the generation of unnecessary passes. The typical structure of these passes is described in figure 3.

In order to accurately track partial series values between passes, intermediate decompression buffers built out of floating point textures are employed. Floating point precision enables to use as many passes as needed, providing that memory is not exhausted and shader limitations are not violated. Spherical harmonics coefficients, as well as partial sum values, are fetched from texture maps. All of these buffers are parameterized using μ. Each weights texture holds the three coefficients pertaining to a given basis function, the three vector components being mapped to the texture color channels. As a consequence, $(l+1)^2$ texture maps are required to reach SH of order l. Their size is directly related to the cardinal of O_d. Integer quantization of the weights using eight bits per channel has not been found to adversely affect the final result, as shown in figure 5.

5.2 Isolated Surface Algorithm

Isolated surface is defined as a surface where no, or weak, continuity is desired at border, such as one of the six different sharply bounded faces of a cube, some light bulb inserted into an electrical

Figure 5: Comparison between SH weights storage texture: 4×32 bits floating point and 4×8 bits quantized.

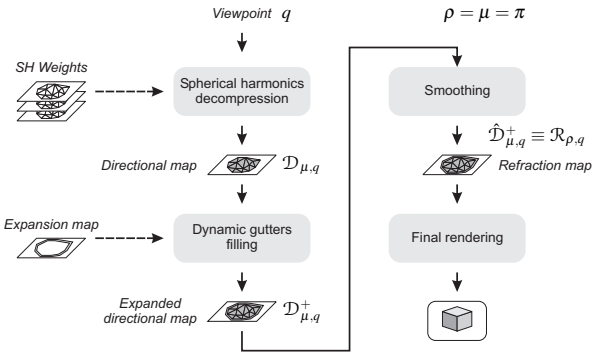

Figure 6: Isolated surface rendering pipeline.

fixture, or any object whose texture border cannot be viewed by the user. A more precise definition is a surface whose texture space parameterization π is using a single chart, no vertex being duplicated, preserving mesh topology once unfolded.

The set of surface samples O_d is induced by the center of texels and can be non uniformly distributed over the object surface. Anyway, neighboring relations between o_i samples are preserved inside the map and $\mu = \rho = \pi$ can be chosen for simplicity.

Retaining mesh topology in texture space has other multiple implications. Firstly, the directional map building efficiency can be increased. Indeed, the mesh footprint induced by μ rarely fills up all the available texture space. Yet, valid samples are easily accessed by rendering the unfolded mesh in texture space, saving computations. Secondly, samples local viewing directions $\omega_{o_i}(q)$ can be computed at vertex level and interpolated across triangles. Thirdly, smoothing described in section 4.3 can be performed directly in texture space, using simple local averaging filters.

Perfect continuity at border being not required, dynamic gutters around mesh footprint are efficient enough to avoid processing of unassigned texels during hardware texture filtering. Once computed, the directional map $\mathcal{D}_{\mu,q}$ is expanded inside gutters, leading to the map $\mathcal{D}^+_{\mu,q}$. Care should be taken so these gutters are made large enough to provide adequate safety margins, especially if strong spatial smoothing is applied. The resulting smoothed map is denoted $\hat{\mathcal{D}}^+_{\mu,q}$.

In the case of isolated surfaces, the refraction map $\mathcal{R}_{\rho,q}$ used for final rendering corresponds to the $\hat{\mathcal{D}}^+_{\mu,q}$ map. The whole rendering pipeline is summarized in figure 6.

5.3 Joining Surfaces Algorithm

Unfortunately, most of the surfaces do not satisfy with the previous stringent requirements and all possible parameterizations include at least one duplicated edge. Most of borderless surfaces exhibit such a property.

Figure 7: Continuity handling comparison. *Left* and *Center*: isolated surface algorithm (section 5.2) without and with dynamic gutter. In the center picture, small discontinuities might still be spotted. *Right*: algorithm designed for seamless continuity (section 5.3).

To behave satisfactorily during final rendering, this partition must not induce any discontinuity at charts borders. Adjacent parts being stored non contiguously in texture space, regular texture filtering performs defectively. At first glance, filling dynamic gutter areas with values matching the opposite chart border may seem an effective way to ensure continuity. However, despite providing improvements, such a procedure cannot completely solve the problem due to eventual dissimilar sampling of the duplicated edge. Figure 7 depicts this problem.

Due to the immutable nature of the interpolation algorithm provided by graphics hardware, the only freedom lies in preprocessing data to further circumvent undesirable effects.

Even if no π compatible parameterization of the object surface can be found, an automatic procedure enabling to handle seamless continuity is proposed. This procedure devises the ρ parameterization suitably and applies a simple algorithm executed at runtime by graphics hardware to prevent potential filtering defects. The devised ρ is based on regular sampling of all mesh triangles in texture space, according to some user-supplied rate.

Since ρ involves significant duplication of samples in texture space, this parameterization is not used all along the rendering process to prevent computational waste. In fact, ρ is only used to layout data for final rendering and thus used to format the refraction map $\mathcal{R}_{\rho,q}$. The bulk of computation, especially building of the directional map $\mathcal{D}_{\mu,q}$, is performed using the μ parameterization, chosen to avoid samples duplication. This μ parameterization is computed by packing all samples $o_i \in O_d$ inside a minimally sized texture map.

5.3.1 Interpolation correction

Bilinear filtering I_{bil} interpolates four samples over a normalized square cell, in a way described by equation 11, using notations referring to figure 8 left.

$$\begin{aligned} I_{bil}(\alpha,\beta) \;=\; & (1-\alpha)\cdot(1-\beta)\cdot v_{00} \;+\; \alpha\cdot(1-\beta)\cdot v_{10} \\ + \quad & \alpha\cdot\beta\cdot v_{11} \;+\; (1-\alpha)\cdot\beta\cdot v_{01} \end{aligned} \tag{11}$$

Following this equation, one can state that the resulting value along any edge of the cell is linearly interpolated from the two ends of the edge. On the contrary, all values located strictly inside the cell get non null contributions from all the four samples.

Yet, focusing on three samples and the subsequently enclosed area, barycentric interpolation I_{bar}, described in equation 12, may be achieved. Indeed, it stems from bilinear interpolation providing the value carried by the fourth external sample complies with the

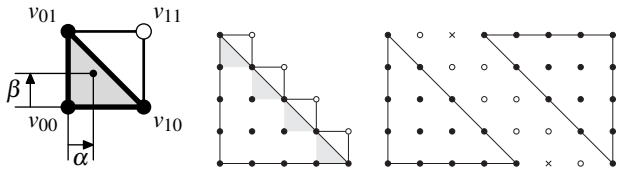

Figure 8: Interpolation correction. *Left*: bilinear interpolation cell. *Center*: triangle mixed interpolation setup. *Right*: triangles interleaved storage.

requirement expressed in equation 13. In this case, along the diagonal edge, values arising from the interpolation also become linearly interpolated from the two ends of the edge. Considering this scheme, any quantity can be correctly interpolated over the induced triangular cell, on behalf of the immutable bilinear filtering.

$$I_{bar}(\alpha,\beta) = \alpha \cdot v_{10} + \beta \cdot v_{01} + (1-\alpha-\beta) \cdot v_{00} \tag{12}$$

$$v_{11} = v_{01} + v_{10} - v_{00} \tag{13}$$

Such a scheme provides a way to interpolate three values across a triangle. Nonetheless, it can be extended to increased sampling rate, providing sampling pattern is regular, as can be seen in figure 8 center. Triangle yet requires to be aligned on axis, values to be interpolated across the triangle defined at regularly distributed locations. In this situation, interpolation cells can be divided into two groups: those completely enclosed inside the triangle, where regular bilinear filtering is suitable, and those featuring one of their corner outside the triangle. The latter kind can be treated along the previously described lines, reverting to barycentric interpolation. As a result, more densely sampled triangles can be handled, still providing satisfactory interpolation.

The mesh is then split and unconnected triangles are stored using this scheme. To make better use of storage space, an interleaved upside down layout is employed, as depicted in figure 8 right. All triangles share the same sampling rate, hence their edges too, whatever their orientation in texture space. As a consequence, any two edges can further be made matching by interpolation correction.

Starting from this layout, the set of surface samples O_d is built from the map texels centers, not taking account of copies generated by duplicated edges. The parameterization ρ is in turn defined as the transformation that maps O_d to this layout.

5.3.2 Algorithm

In the first place, SH decompression is performed to compute the directional map $\mathcal{D}_{\mu,q}$ using the more efficient parameterization μ. Mesh topology not being preserved in this parameterization, one has to explicitly store local frames associated to samples o_i to compute the local viewing direction $\omega_{o_i}(q)$. These data are stored in texture maps.

Then, spatial smoothing described in section 4.3 is applied, still in the μ optimized layout to avoid unnecessary computations. Unfortunately, samples neighboring relation not being preserved in μ texture space, smoothing cannot be performed directly using a simple local averaging filter. As a consequence, smoothing requires to precompute, for each sample, all involved neighboring samples. This list of closest samples and their respective weighting coefficients are built during offline preprocessing and stored in textures. Despite being more costly than smoothing in texture space, such an approach enables to compute the smoothed map $\hat{\mathcal{D}}_{\mu,q}$ avoiding complications at mesh borders.

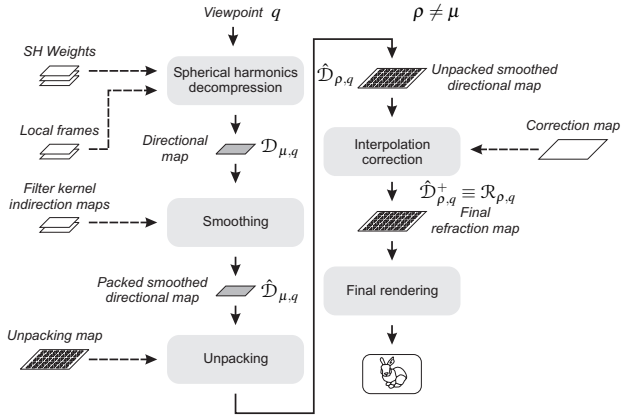

Figure 9: Connected surface rendering pipeline.

Once smoothing has been performed, data are unpacked into the map $\hat{\mathcal{D}}_{\rho,q}$ matching the ρ induced layout designed for final rendering. Due to proper layout, interpolation defects can then be fixed as described in section 5.3.1. The resulting map $\hat{\mathcal{D}}^{+}_{\rho,q}$ corresponds to the refraction map $\mathcal{R}_{\rho,q}$ since refracted directions it contains can be interpolated over the whole mesh surface in a continuous way.

The complete rendering pipeline is summarized in figure 9. While more complex operations are required than in the case of an isolated surface, the whole pipeline can still be completely offloaded onto graphics hardware. Except for directional map building, each part of the whole algorithm is run using a single hardware rendering pass. Yet, circumventing interpolation artifacts is paid by requiring an identical sampling rate on all triangles belonging to the mesh. As a consequence, mesh geometry has to be as homogeneous as possible so as to avoid large surface sampling discrepancies.

5.4 Improvements

Reliance on preprocessing implies that sampled geometry has to remain static as well as material properties. Yet, while sampling is tied to object space, the mesh can be rigidly transformed providing appropriate space remapping is performed. Concerning materials, variations can be achieved using some fallback approximation. When materials carry identical densities, no perturbation of sight ray occurs. One can then interpolate between this straight path and the precomputed one to simulate variable indices of refraction. Still, it should be clearly stated that this procedure does not rely on any physical assumption.

More significantly, considering that only three of the four texture maps channels are filled in weights textures, one can store another information for free. This available slot is dedicated to store the distance d traveled by the light inside the object. It is then possible to modulate the refracted environment during final rendering. The transmittance $\tau(d) = e^{-\kappa \cdot d}$ can be computed on the fly, given the material optical depth κ. Due to the vectorized nature of graphics hardware and the fact that basis polynomials are already computed, no additional computations are induced.

6 Results

Results computed using the technique described in section 5.3 are now presented. Main statistics of all examples are gathered in ta-

Model	Samples / Rays count	Tri. count	SH Tex. size	Mem. usage	Fps
Shell	34 k / 70 M	4288	186^2	17 MB	52
Bunny	35 k / 72 M	69451	187^2	35 MB	40
Mug	42 k / 86 M	9184	205^2	24 MB	43
Teapot	41 k / 84 M	2256	203^2	23 MB	45
Lucy	119 k / 244 M	238734	346^2	121 MB	7

Table 1: Rendering statistics. All examples use SH of order 8.

ble 1. All measurements are made on an AMD Athlon XP1800+, equipped with a 128 MB ATI Radeon 9700 Pro. Using this graphics hardware, SH can be handled up to order eight. In all situations, viewport comprises approximately 1350×1100 pixels. Yet, image size remains mostly of no significant influence in performance since refraction map building occurs in texture space. In all presented examples, primary reflection is added to refraction, both being weighted according to their respective Fresnel coefficients.

On this hardware, interactive timings are achieved by a significant margin, even on objects such as the Stanford bunny. This example is depicted in figure 10, along a glass mug half filled with water. This latter example demonstrates the ability of the method to handle complex objects involving multiple media. Indeed, even if such sampled optical device is more complex than a single object, the same technique can still be applied unchanged.

The presented technique can be visually compared to existing techniques in figure 11. It compares the presented method with the classical single-sided refraction, the double-sided approximation technique [Wyman 2005] and a raytraced reference. Except for the method presented in this paper, the bead inside the shell is visually discarded, due to the layered structure of the object. Moreover, on this highly non-convex object, double-sided refraction approximations may prove wrong enough to generate visible artifacts.

Finally, effects of compression are presented in figure 12, varying the order of SH. Even if low orders significantly miss important features of refractive patterns, behavior still appears realistic. This latter fact hints that decompression can be performed progressively. Trading visual quality to interactivity, the order of basis can be varied according to some target framerate. Moreover, to further improve interactivity, one might as well easily split the SH decompression process over multiple frames, albeit at the price of a slight visual latency.

One might note sparkling or aliasing on some parts of the objects. Indeed, spatial sampling pattern on object surface is chosen at precomputation time. If the chosen fixed sampling rate is higher than actual display rate, aliasing artifacts may appear. Unfortunately, due to continuity constraints on the refraction map, one cannot filter the latter using mipmapping. This problem is even more acute since refraction map behaves inherently as an indirection map. Strong variations in the environment may then translate in visible sparkling. This hints that care should be exercised concerning the sampling rate choice. Nevertheless, it may prove difficult to handle objects presenting strong distance discrepancies, such as sea surface. Another source of sparkling lies in the pointwise sampling of the environment. Indeed, on areas of high curvature, large environment regions tend to be projected on small groups of pixels. Consequently, strong variations in the environment may result in noisy refraction.

Concerning preprocessing, presented examples were computed using the regular sampling of a reference hemisphere using 2048 directions. As an example, the bunny, comprising about 35,000 spatial samples, was sampled in about 26 minutes, using about 71.5 million rays. It was further compressed in 320 seconds.

Figure 10: Significant model and multiple media composite object made of glass and water.

7 Conclusions and Future Work

A method allowing to display refractive behavior has been presented. Building on both raytracing and polygon rendering schemes, the method provides realistic effects at interactive framerate. Raytracing is used to precompute light travel inside the object for any viewpoint. Then, sampled data, stored using a compact representation, are used to deduce rays exiting the object and to sample the environment map accordingly. This rendering step can be performed at interactive framerate thanks to programmable graphics hardware. A suitable SH formulation is used to devise an adequate hardware basis functions evaluation procedure.

While not cheap, neither in memory consumption nor in computations, the method is able to deal with many complex static geometries in a satisfactory way. Nonetheless, visual artifacts may appear if the surface sampling rate exceeds the display sampling rate. This prevents the method to correctly deal with objects that are stretched away from the viewer. Moreover, all effects cannot be captured because of their high directional frequencies. Yet, in this case, method always provides believable effects, gracefully reverting to smoother variations due to the progressive nature of the SH based compression.

Future work should investigate ways to further reduce memory consumption and to improve data representation. To achieve this goal, presenting similarities with surface lightfields, the presented approach might take advantage of related advanced compression techniques. Moreover, the presented method might benefit from increasing programmability of graphics hardware to group all steps into a single rendering pass. A single pass may alleviate aliasing problems, rendering being made directly in screen space instead of using intermediate textures. Finally, an alternate algorithm should be proposed to handle joining surfaces with more flexibility with respect to the sampling choice.

Acknowledgements

Environment maps courtesy of Paul Debevec. Stanford bunny provided by the Stanford University Computer Graphics Laboratory. Decimated lucy model courtesy of Project Gamma of Inria.

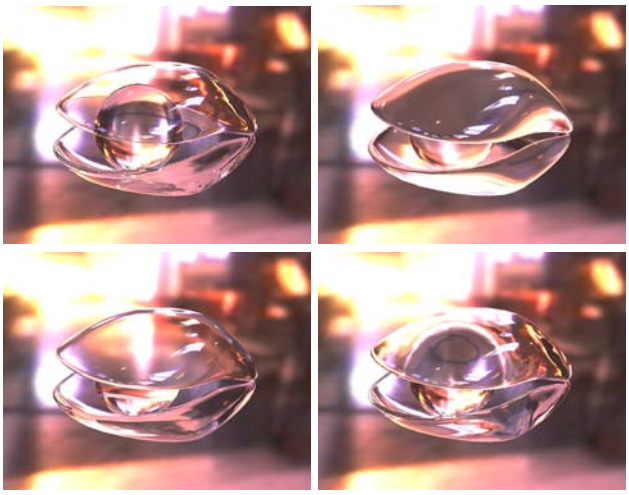

Figure 11: Comparison of methods. *Top*: raytracing reference and single-sided refraction. *Bottom*: double sided approximation and proposed method.

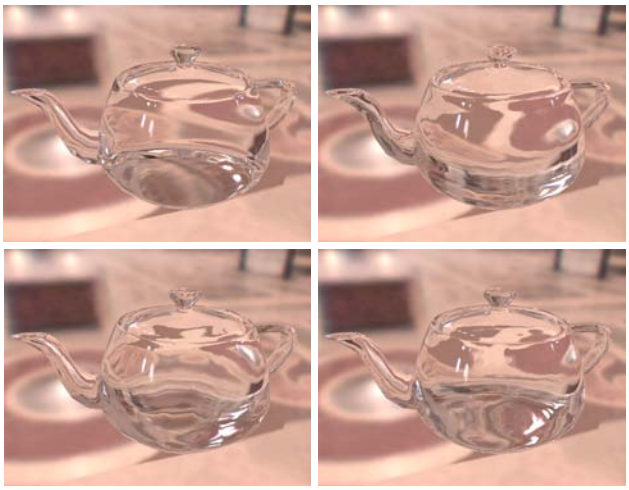

Figure 12: Different levels of decompression. Raytracing reference and SH up to order $l = 1, 4, 8$. Memory usage and framerate respectively 9, 13, 23 MB and 122, 100, 45 fps.

References

CHEN, W.-C., BOUGUET, J.-Y., CHU, M. H., AND GRZESZCZUK, R. 2002. Light field mapping: efficient representation and hardware rendering of surface light fields. *ACM Transactions on Graphics 21*, 3, 447–456.

DIEFENBACH, P. J., AND BADLER, N. I. 1997. Multi-pass pipeline rendering: Realism for dynamic environments. In *Proceedings of the 1997 symposium on Interactive 3D graphics*, ACM, 59–70.

GUENTHER, R. D. 1990. *Modern Optics*. John Wiley & Sons.

GUY, S., AND SOLER, C. 2004. Graphics gems revisited. *ACM Transactions on Graphics 23*, 3, 231–238.

HAKURA, Z. S., AND SNYDER, J. M. 2001. Realistic reflections and refractions on graphics hardware with hybrid rendering and layered environment maps. In *12th Eurographics Workshop on Rendering*, Eurographics, 289–300.

HECKBERT, P. S., AND HANRAHAN, P. 1984. Beam tracing polygonal objects. *Computer Graphics 18*, 3, 119–127.

HEIDRICH, W., SLUSALLEK, P., AND SEIDEL, H.-P. 1997. An image-based model for realistic lens systems in interactive computer graphics. In *Graphics Interface*, 68–75.

HEIDRICH, W., LENSCH, H., COHEN, M. F., AND SEIDEL, H.-P. 1999. Light field techniques for reflexions and refractions. In *Rendering Techniques '99*, Eurographics, 187–196.

KAY, D. S., AND GREENBERG, D. P. 1979. Transparency for computer synthesized images. *Computer Graphics 13*, 2, 158–164.

LI, S., AND MUELLER, K. 2005. Accelerated, high-quality refraction computations for volume graphics. In *International Workshop on Volume Graphics 2005*, 73–81.

MACROBERT, T. M. 1967. *Spherical harmonics: an elementary treatise on harmonic functions with applications*. Pergamon Press.

OHBUCHI, E. 2003. A real-time refraction renderer for volume objects using a polygon-rendering scheme. In *Computer Graphics International*, 190–195.

SLOAN, P.-P., KAUTZ, J., AND SNYDER, J. 2002. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. *ACM Transactions on Graphics 21*, 3, 527–536.

SLOAN, P.-P., HALL, J., HART, J., AND SNYDER, J. 2003. Clustered principal components for precomputed radiance transfer. *ACM Transactions on Graphics 22*, 3, 382–391.

TS'O, P. Y., AND BARSKY, B. A. 1987. Modeling and rendering waves: Wave-tracing using beta-splines and reflective and refractive texture mapping. *ACM Transactions on Graphics 6*, 3, 191–214.

WESTIN, S. H., ARVO, J. R., AND TORRANCE, K. E. 1992. Predicting reflectance functions from complex surfaces. *Computer Graphics 26*, 2, 255–264.

WHITTED, T. 1980. An improved illumination model for shaded display. *Communications of the ACM 23*, 6 (June), 343–349.

WYMAN, C. 2005. An approximate image-space approach for interactive refraction. *ACM Transactions on Graphics 24*, 3, 1050–1053.

YU, J., YANG, J., AND MCMILLAN, L. 2005. Real-time reflection mapping with parallax. In *SI3D '05: Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM, 133–138.

ZONGKER, D. E., WERNER, D. M., CURLESS, B., AND SALESIN, D. H. 1999. Environment matting and compositing. In *Proceedings of ACM SIGGRAPH 99*, ACM, 205–214.