Tech report WECS-2002-10: View-dependent Texture Maps*

Cindy M. Grimm William D. Smart John F. Hughes

ABSTRACT

We present a technique for adding view-dependent information to geometry, allowing us to write down, for each point on the surface, what the surface looks like from every direction. This data structure allows the incorporation of "image-based" objects into ordinary rendering, allows broader viewing conditions for image-based objects (one can look at a torus from "inside the hole," for example), and provides opportunities for compression of image-based objects and for separate interpolation of intensity and color data for image-based objects.

We based our work on a surface representation that supports models of arbitrary topology, and discuss various methods for capturing, representing, and compressing the view-dependent information.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling, Curve, Surface, Solid, and Object Representations, Splines, Image-based Rendering.

1 Introduction

This paper describes work designed to bridge the gap between image-based rendering and the traditional rendered geometry approach. It may be viewed in two ways. The first is as adding geometry to an image-based rendering. In this view, knowledge of the geometry helps us to make choices about which samples to use when "rendering" new images. The second is that is adds view-dependence to traditional geometric objects, *i.e.*, creates objects that have interesting surface behavior.

In conventional rendering, the light arriving at an object is reflected from it; the reflectance process is modeled by some relatively simple model like the Phong lighting model. The dependence of the reflected light on the view direction is relatively simple. This fails to capture various important phenomena, but improved models, in which the BRDF is recorded at each point and used to transform incoming light to outgoing light, have been developed. The success of such models depends not only on the accuracy of the surface model, but on the accuracy of the model of the incoming light. The light arriving at a point of the surface is usually modeled by some approximation to the true incident light, consisting perhaps of direct light, or direct light plus some important indirect lighting. In some stochastic rendering schemes, samples are taken from the entire incident light field, an approximation that's (for some sampling strategies) guaranteed to converge to a good approximation of the incoming light as the number of samples increases.

In image-based rendering[1], the effects of the surface on the incoming light are recorded by viewing the image from multiple directions. In this case, *all* of the incident light is considered, but only a few samples of the outgoing light-field are recorded; it's essentially dual to the conventional rendering model, in which the outgoing field can be approximated for any view, but the incoming field and the way it's transformed have been approximated by some relatively simple sampling model. To construct the outgoing field in directions not previously observed, one interpolates between "nearby" observations.

Slightly more formally, one may consider each point of an object X as something that transforms an incoming distribution of light (described as a function on S^2, the sphere of directions from which light may arrive) into an outgoing distribution of light, again parameterized as a function on S^2. Thus the entire behavior of an object can be characterized by a function

$$T : X \times A^{S^2} \to A^{S^2},$$

where A denotes the space in which "amount of light" is measured (for intensity computations, one may just use $\Re$), and A^{S^2} denotes the set of all functions from S^2 to A. If P is a point of X, then $T(P, \cdot) : A^{S^2} \to A^{S^2}$ is a version of the BRDF: when applied to a delta-function along a single incoming direction, the resulting function is the reflected light distribution. Image-based rendering takes a fixed lighting condition f and *indirectly* records the value of $T(\cdot, f)$ on a finite number of sample directions using the constancy of radiance along rays: by observing the radi-

*This work was supported by a grant from the NSF.

[1]For the moment, consider the special case of image-based rendering where the images are gathered from real-world data using a digital camera.

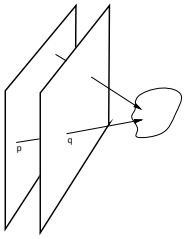

Figure 1. The Lightfield and Lumigraph represent the outgoing lightfield (i.e., the transformed incoming lightfield) by sampling the radiance along rays passing through two parallel planes.

ance along some ray that intersects the object, one has indirectly collected data about the function $T(\cdot, f)$. Thus the Lightfield [LH96] and Lumigraph [GGSC96] representations both generate functions on (subsets of) $\Re^2 \times \Re^2$ (called light-slabs) where the value at a point (p, q) is the radiance along a ray passing through the points p and q of two parallel planes in $\Re^3$ (see Figure 1). Conventional rendering takes an explicitly known value of $T(P, \cdot)$ (or an approximation of T) and applies it to an approximation of the incoming field.

In this paper, we study a different representation of the function $T(\cdot, f)$ – one in which the geometry of the object is more explicitly taken into account. This has two advantages. The first is that the light-slab representation fails to capture certain occlusion properties (see Figure 2), but our representation can handle these better, making it possible to render such image-based objects within a conventional renderer. The second advantage is that for fixed f, the function $T(P, f)$ may vary slowly as a function of the surface point P (or may be written as a combination of a slowly-varying term and a rapidly varying one). If all the data for a point P and its neighbors are nearby one another in the representation, there's an opportunity for enormous compression of the data. More simply: the pattern of light emitted from a point P on the surface is likely to be very similar to that emitted from a nearby point P'. If the surface is textured with a rapidly-changing texture, then at least the patterns are likely to be similar, although a darkly-colored spot will emit proportionally less light than an brightly-colored one. Conventional rendering takes this to a limit, and treats the BRDF and texture map as independent functions on the surface, multiplying them to get the true reflectance.

Keeping in mind this model of what we're trying to represent - the light leaving an object, written as a function on the set of rays in empty space - we can now describe our representation. We take the object of interest and construct a smooth surface S that closely approximates it. In the case

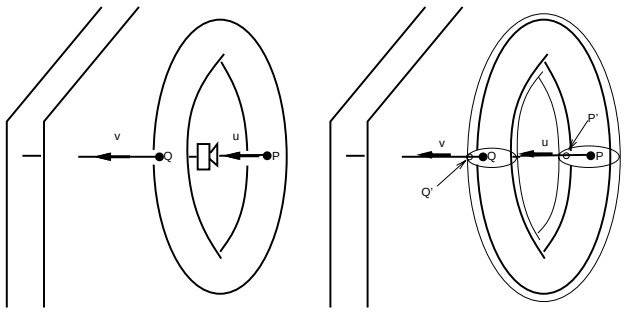

Figure 2. (a) To compute a view of the object from the camera position shown, one determines the appearance of P by extending a ray through the two planes defining the light-slab; unfortunately, the data recorded there is the appearance of the point Q along that ray. The problem is that the lightfield *should* be a function on the set of rays in empty space, but a light-slab approximates it by a function on the set of rays in space; there should be different values on the rays labeled u and v, but in a light-slab, there's only one. (b) In a surface-based representation, there's room for different values at u and v (the value for u is stored in the sphere of directions at P'; the value for v is stored in the sphere of directions at Q'.)

of synthetic objects, the approximating surface may be exact; in the case of objects captured by digital photography, it's more approximate. We represent this surface as a *manifold* [GH95], i.e., as a collection of overlapping patches, each of which is homeomorphic to a planar disk. We consider the set of outward-pointing rays from each point of this object; for each point, this set can be parameterized by a hemisphere. We'll represent the emitted light as a function on the collection of all of these outgoing rays[2] by recording *samples* - the values of the function on certain outgoing rays - and then interpolate to determine values elsewhere. The details of the surface representation and the interpolation schemes form the bulk of the remainder of the paper.

A first application of this representation is as a capture mechanism for both real and synthetic objects, where we capture both the geometry and emitted-light behavior of the object. We can then place this object into a new scene, replacing the traditional lighting calculations with the emitted-light that we've recorded. Because the captured object has at least a crude approximation of the real geometry (it need not be exact) we get around problems of occlusion that would plague a light-slab representation[3].

A second, related use is to capture surface emitted-light characteristics so they may be applied to a new surface. We do not explore this application, but we do discuss methods for compressing captured data and filling in missing sample points.

The third use is as a view-dependent texture map. Using inverse painting techniques the user can paint what the surface looks like from one or more directions. Because the user is painting on the actual geometry, self-occlusion is not a problem.

The use of geometry also provides us with an intermediate step between the capture and compression stages. We can employ knowledge of the scene to interpolate between actual samples before passing them on to the compression techniques. Since many compression techniques work best on regularly spaced data, this can be very helpful.

2 Related work

There have been a variety of image-based rendering techniques reported over the last few years. Approaches vary from "pure" image-based approaches, where no knowledge of geometry is used [GGSC96] [LH96] to approaches which assume some basic geometry [DTM96] or depth cues [SGHS98] [MB95]. At the heart of all of these papers is the question, "Given this set of samples (several images), how do I construct a new set of samples (i.e, a new image)?" Some papers control the location of the samples to simplify reconstruction [Che95] [SS97] [SH99]. Others assume knowledge of the camera position and make some assumptions about the geometry [MB95] [SGHS98] [YM98]. Like Lumigraphs and Lightfields, we capture a set of rays passing through space, only we capture at a surface instead of a plane. This means we lose the tremendous advantage of regularly spaced samples, but gain the benefit of geometry we can place in a scene without worrying about occlusion.

Several papers recently have focussed on capturing the lighting of a scene in order to apply it to new geometry placed in the scene [Deb98] [CON99]. Curless, *et. al.*, [CL96] describe a method for capturing how an object affects light passing through it.

The two papers most closely related to this work are by Debevec, Taylor and Malik [DTM96] and Miller and Mondesir [MRP98]. The former uses simple geometry to guide reconstruction of samples, essentially projecting the images onto the simplified geometry of the building. We take this approach one step further and re-write the image information on the object itself, as Miller et al. showed was feasible.

3 The surface representation

In this section we provide an intuitive description of manifold surfaces their uses. A detailed description is in [GH95].

The key to manifolds is that they represent complicated surfaces by gluing together *overlapping* simple ones. We use spline surfaces as the simple surface type, although any other class of embeddings of surface pieces into $\Re^3$ could be substituted. Traditional methods of stitching together simple surfaces into complicated ones involve gluing the surfaces together edge-to-edge by constraining their boundaries. The problem with this approach is illustrated in 2D in Figure 3. Three curves are stitched together end-to-end by matching the end-point derivatives. Notice that if we change the geometry of the middle curve even slightly, the resulting "curve" is no longer even continuous. The manifold approach, using the same three curves, is shown in Figure 4. Here the final curve is defined as a *blend* of the three input curves. Now, even if one (or more) of the curves change, the blended curve is still smooth. In general, the multiple surface pieces should all "agree", and the blending functions are used as "indicators" – the blend function has the largest value at a point tells you which patch the point is "most inside." But the use of blend functions to smooth out differences between corresponding surface patches is valuable as well.

As an additional benefit, the derivatives are also smooth (provided the individual embeddings and the blend functions have smooth derivatives). Therefore, we can define

[2] We've deliberately ignored transmission; transparent surfaces will be addressed in future work.

[3] This application is dual to work of Debevec [Deb98] in which synthetic objects were added to scenes in which an *incoming* lightfield (a kind of enhance environment map) had been recorded.

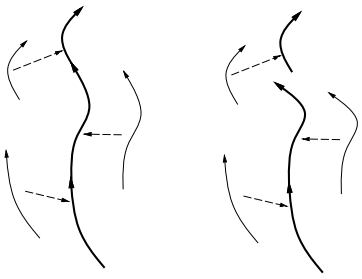

Figure 3. The traditional approach to stitching together curves. Each curve has its own parameterization and continuity of the glued-together curve is achieved by matching endpoints (left). If one of the curves changes, continuity must be re-established (right).

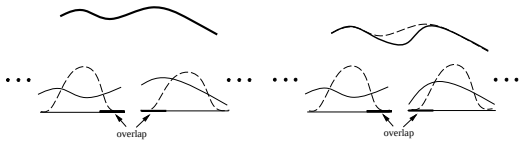

Figure 4. The manifold approach to curve stitching. Each curve has its own parameterization. We specify how curves overlap in parameter space and also define blend functions (dashed) for each of the curves (left lower). The blend functions sum to one and are used to blend between one curve and the next where they overlap, resulting in the thick line (left upper). Changing one of the curves (right lower) does not affect the continuity of the resulting curve (right upper) (original blend shown dashed).

the normal at every point on the surface, using the parameterization of any of the charts that contain that point.

Manifold surfaces are constructed similarly, but out of overlapping 2D surface patches (about three patches overlap at a typical point of our surfaces). We use a mesh to describe the network of patches and to give a first approximation to the geometry (see Figure 5). The control points of the individual patches are initially assigned locations which are linear combinations of these mesh vertices. We use the Catmull-Clark subdivision [HKD93] rules to determine the linear combinations.

A brief note on parameterization: Most embeddings "behave" best when their parameterization is uniform, *i.e.*, taking a δ-sized step anywhere in parameter space produces a Δ-sized step in 3D. With any reasonably complicated geometry, it becomes difficult to "stretch" a single embedding to fit without greatly distorting the parameterization. In fact, some surfaces (the sphere) admit no smooth, uniform, global parameterization. Manifold structures on a surface address this problem by dividing up the surface into small pieces, each of which is easily modeled with a "well-behaved" simple embedding.

Figure 5. Left: The mesh used to specify the connectivity and geometry of the surface patches. Right: The resulting surface.

We can extend the idea of blending between the embedding functions defined on overlapping charts to other types of functions. For example, suppose that for each chart we have a function that defines the color at every point of that chart. We can blend together these functions to get a global coloring function. If the function pieces agree at corresponding points, the glued-together function will look like each of them; if they disagree, the gluing-together blends them. The functions we use to blend between the chart functions can be altered to give different effects. There is a trade-off between using a smooth, slowly changing blend function (which reduces discontinuities, but introduces blur) and an abrupt one.

When we use manifolds as the geometric foundation for representing the emitted light from an object, there are two different approaches to rendering. The first is to directly query the surface color by intersecting it with a ray from the camera and evaluating the "outgoing light" function at that point in that direction. This has two costs; the first is intersecting the ray with the surface, the second is the cost of evaluating the function. For making an image, this intersection and evaluation must happen for every pixel the object occupies.

The second approach, suitable for parallelization, is to render the object with a texture map, updating the texture map when the camera changes. In this case, we cast a ray from the center of each pixel in the texture map (mapping the location of the pixel to 3D using the embedding equation) back to the camera. The cost for this is filling in the pixel value plus the function evaluation. There are three advantages and two related disadvantages to using a texture map:

+ The texture map can be updated independently of drawing the object, and at a slower rate if needed.

+ Parts of computations can be cached since we are always evaluating the function at the same points.

+ The number of texture map pixels can be varied to roughly match the number of pixels the object occupies on the screen.

− We may do more function evaluations than strictly needed if the texture map is too dense.

− The texture map may not have enough pixels to accurately capture the variation in the surface.

We used ray intersections for the pictures in this paper, but use texture mapping in our interactive prototype. This prototype was used to test results but all the figures in this paper were produced using the ray intersection method.

A brief note on the texture mapping: Each chart is assigned a portion of the texture map, with pixel boundaries lying along the tessellation boundaries. Usually we "pad" these texture map pieces with an extra pixel on all sides to make OpenGL behave correctly. We change the resolution of the texture map by increasing or decreasing the size of the texture map piece belonging to each chart, with a minimum size of 1 pixel plus padding for each chart. This gives us seamless, fairly uniform texture mapping (i.e., the non-uniformity of the texture map is no worse than the non-uniformity in the individual embeddings).

4 The Emitted-Light Function

In this section we describe the emitted-light function on the surface and discuss two different coordinate systems for expressing it. The emitted-light function tells, for each point on the surface and each ray emanating from that point, the color[4] of that point when viewed from a point along that ray (see Figure 6). If, for each point on the surface, the color of that point were the same from all directions then this function would be little more than a standard texture map. Another way to think about the emitted-light function is as a view-dependent texture map, i.e., a texture map that changes with the viewing direction (albeit one that is examined directly to get the emitted light, rather than being multiplied by some incoming light weighted by some lighting function).

The position on a parameterized surface corresponds to a parameter point in $\Re^2$. A direction at a point on a parameterized surface can be defined by a unit vector, i.e., a point on the sphere, S^2. Therefore the emitted-light function for a small patch of surface is defined by a color value (which we will represent as a point of $\Re^3$) at a parameter point in $\Re^2$ and a direction vector in $S^2(\Re^3)$: the function-piece from which we'll build the global emitted-light function is a function

$$R : \Re^2 \times S^2 \rightarrow R^3 : ((s,t), \mathbf{v}) \mapsto \mathbf{R}((\mathbf{s},\mathbf{t}), \mathbf{v}) \in \Re^{\mathbf{3}}.$$

For an arbitrary surface there usually is no single parameterization of the entire surface, and so as described previously, we divide the surface into overlapping regions. For a single point on a surface, the outgoing direction vectors lie in a hemisphere located at that point (see Figure 6). As we move across the surface the vectors always fall into a hemisphere but that hemisphere is continually changing as we move, so that the aggregate of all the direction vectors (ignoring base points) will cover the entire sphere.

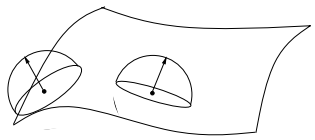

Figure 6. At each point of a surface-piece, there's a hemisphere of outward pointing vectors; at different points, the hemispheres will generally have different orientations.

Just as there's generally no way to parameterize an entire surface, there's generally no way to parameterize the set of all direction-vectors emanating from all points of a surface. There are two choices: (1) parameterize them locally, just as we did for the surface, and (2) use a redundant absolute coordinate system – simply write down the (x, y, z) coordinates of each unit vector. Each has its advantages. Before describing them, though, let us recall how they'll be used: we will be asking, for a point P on the surface and a direction vector $\mathbf{v}$ at that point, what is the emitted-light function's value at $(P, \mathbf{v})$. To determine this, we'll look for samples of the function at locations near $(P, \mathbf{v})$. How is "nearness" determined? By some combination of proximity on the surface and similarity of direction (see Section 5). We'll use absolute coordinates to help us quickly locate samples that have similar directions; we'll then use a local coordinate frame to compare more precisely.

4.1 Absolute coordinate system

The absolute coordinates of a point-direction pair are used to rapidly find samples whose directions are close to the given direction, by computing dot products. Because dot

[4] More precisely, the radiance at that point in the given direction; we record this at each of three wavelengths, and refer to the resulting triple as a "color".

products are expensive, we use a binning procedure to narrow the search. We start by defining a set of points (the *bin points*) that are approximately evenly spaced on the sphere [5]. We then assign each sample to the bin point it is closest to, as measured by the dot product. To find the closest samples (close in *direction*) to a given point-direction pair, we take the dot product of the direction with all of the bin points, then chose points in the n bins which are closest, where n is typically 3-5. To ensure that the closest points are found, n must be at least 3.

Typically, the desired number of bins is the smallest number such that the number of sample points in each bin is relatively small compared to the entire number of samples. For the examples in this paper, the number of bins was between 800 and 1000.

4.2 Local coordinate system

The local coordinates of a point-direction pair are defined surface-piece by surface-piece (see Figure 8). For a given surface-piece, parameterized by a function

$$X : \Re^2 \rightarrow \Re^3 : (s,t) \mapsto X(s,t),$$

the coordinates of the "point" part of the point-direction pair are just the (s,t) parameter coordinates; to assign coordinates to the direction portion of the pair, we build a coordinate frame at the basepoint: the first vector is the normalized tangent to the s-parameter curve (i.e., dX/ds); the second is the outward normal at P, and the third is the cross-product of these. The resulting triple of coordinates is then stereographically projected to $\Re^2$ by sending

$$(x,y,z) \mapsto \frac{(y_p - y_c)}{(y_p - y)}(x,z),$$

where y_p and y_c are parameters of the stereographic projection; in general, we use $y_c = 0$ and $y_p = -1$, although in the special case of reconstruction via neural-net backpropagation (see Section 5) other choices are needed. As a happy coincidence, for any point P, we can take point-direction pairs whose points are near P (i.e., in the same chart, provided that the chart does not curve too much) and assign $\Re^2$ coordinates to the "direction" part using the coordinate frame at P and the stereographic projection above. While point-direction pairs at P, at least ones where the direction is outward-pointing, will all have (x,y,z) coordinates in which y is non-negative, directions from points nearby P may have negative z-coordinates; if we choose y_c smaller than the smallest such z-coordinate, the stereographic projection will remain injective on the portion of the sphere where we apply it.

[5] We subdivide each face of an icosahedron into a equilateral triangle grid and project the resulting vertices onto the sphere.

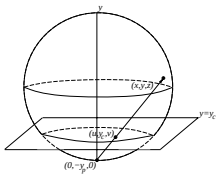

Figure 7. Projecting part of the sphere onto the plane at $y = y_c$. The part above the cutting plane is projected into a disk (shaded) in the plane; the projection is injective on this part.

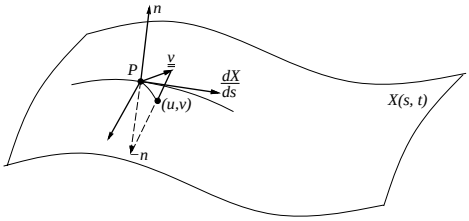

Figure 8. For typical point-direction pair, $(P, \mathbf{v})$, the surface parameterization gives (s,t) coordinates to P, and gives a coordinate frame at P, defined by the tangent to the s parameter-curve at P, the normal at P, and their cross product. The coordinates of $\mathbf{v}$ in this frame are stereographically projected from the endpoint of the inward normal to get a pair (u,v) of coordinates with $u^2 + v^2 \leq 1$.

5 Construction of new points

The previous section gave us a method for writing down samples. Section 6 will discuss possible methods for collecting samples; for now we will assume that we have a collection of N samples $\{P_i : 1 \leq i \leq N\}$, each an element of $\mathbf{Z} \times \Re^2 \times \mathbf{S^2} \times \Re^3$ where the first factor tells which surface piece the sample is associated with, the second gives the local coordinates of the basepoint, the third the direction, and the fourth is the color of that sample.
Constructing a new data point can then be phrased as, "given a new point $p \in R^2 \times S^2$, what color should we assign to it, based on the known samples P_i?" This question is complicated by the existence of two unrelated coordinate systems, one for the position and one for the direction. Simply taking the closest sample in Euclidean distance space is clearly not the answer.

We first discuss assumptions we can make about the function the samples came from. Based on these assumptions we can define what the "best" interpolation method is; different assumptions will lead to slightly different interpolation methods. Finally, we detail an implementation which makes the interpolation method feasible.

There are two factors influencing the reflected color; the first is the BRDF of the surface, the second is the location and direction of the light sources. If the surface is homogeneous, the BRDF is uniform, and the light sources are an infinite distance away then the samples which point in the same direction will have the same values. Of course, BRDFs are not uniform, surfaces are not homogeneous, and light sources are rarely at infinity. Still, if the surface and the BRDF are not changing rapidly then the best samples to reconstruct from are ones which are pointing in the correct direction, even if they are a slight distance away. This will capture the motion of highlights. If, however, the surface color is changing rapidly but the BRDF has no large specular component (as in the case of a highly varied texture map on a diffuse surface) then the best match for color is a sample located at the same point, even if it points in another direction.

Under these assumptions, the best samples to interpolate are those which point "roughly" in the correct direction and are as close as possible to the point on the surface. If we are interested in tracking highlights then "roughly" will be much closer than if we are mostly interested in the correct color. Another way to phrase this is, "look for samples pointing in the correct direction at the desired surface point". If no samples are to be found, look at all the samples pointing in the correct direction and take the ones that are located closest to the surface point (if there are any nearby).

Because color and intensity differ in their best method for interpolation, one solution is to convert the output colors of the samples into an intensity plus color representation (*e.g.* YIQ). Intensity can then be reconstructed using samples which point in the same direction, while color is reconstructed using samples closest on the surface. This is an approximation of the classical geometry-based graphics approach, in which lighting is computed using a (usually constant) approximation of the material BRDF, and then modulated by the texture map.

5.1 Implementation

To make this approach feasible, we need to quickly find samples pointing in the correct direction near a point on the surface. We solve this problem by binning the samples beforehand. In this initial step we also propagate samples around as needed to fill out bins which are missing samples.

The surface domain is subdivided evenly in s and t, while the directions are binned using the method described in Section 4.1. There are two variables; the number of divisions in the domain and the number of direction bins. The number of domain divisions should be big enough so that for any given domain bin we only get a small number (10-300) of samples falling in each direction bin. If the domain divisions are too big we will spend too much time sorting the samples in the bin at reconstruction time. Empty bins will be filled in with neighboring samples so the only problem with divisions that are too small is that it is expensive to fill and keep unnecessary bins.

The number of direction bins to use depends on whether or not we are tracking highlights. In general for highlights, we chose a level where the dot product between a bin and its closest neighbors is greater than 0.86, the distance at which we consider two samples to point in the same direction. Colors typically get one level less.

After we have assigned our samples to the appropriate bins, we propagate as needed to fill in empty bins. To prevent inappropriate filling in, we limit the propagation to about one fifth of the size of the domain. For any missing direction bin in a domain bin, we examine all the samples which fall in that direction bin and take the one or two which are closest as measured on the surface domain.

At reconstruction time we take the union of the nine chart domain bins surrounding the bin the sample point falls into. We then examine the spherical bins; whenever one (or more) of the samples falls into the same spherical bin, we take the one (or more) which is (are) closest in the surface domain to the input point. The more points we allow per spherical bin, the more blending we get.

This gives us the samples to use for reconstruction. Since these samples point mostly in the same direction, we can project them down onto the plane using the technique described in Section 4.2 (in which all point-direction pairs near a particular point are projected to a plane via a single stereographic projection). We now interpolate just in

the plane using a locally weighted regression technique.

Locally weighted regression (LWR) is a variation of standard linear regression techniques. Training the algorithm simply involves storing the sample points. When LWR is called on to make a prediction of at a query point, it performs a linear regression on the sample points, with each point being weighted according to its distance from the query point. Sample points that are closer to the query have more influence over the regression, while those sufficiently far away have little or no influence. It should be noted that a new regression is required for every query point, unlike a standard global regression, where one model is used for all queries.

Using LWR allows us to have a globally non-linear model using simple and well-understood regression techniques. However, if implemented naively, predictions can be extremely expensive to perform (since each one involves a regression over all of the sample points). It is possible, however, to reduce the cost of predictions significantly using appropriate algorithms, such as those surveyed by Atkeson, Moore and Schaal [AMS97].

We can make a good guess for the correct kernel size by setting it to be twice the average distance from the input points to the origin.

6 Capture process

We have experimented with two different capture processes. The first is the usual "snapshot" approach, where several pictures are taken of the object. The second approach is only possible with a ray-tracer; we take a "picture" of the emitted-light function at evenly spaced points on the surface.

The snapshot approach produces a camera matrix and an image. To convert this into emitted-light-function samples, we intersect the ray from the camera through the center of each pixel with the object. The point of intersection gives us a point in the domain of each chart which overlaps that point (typically three). We also write down the ray and the color of the pixel, giving us a complete sample. The samples produced by this method tend to be clustered by direction but fairly evenly spaced in the surface domain. The left side of Figure 9 shows this clustering in effect; notice how the locations of the clusters move from one side of the chart domain to the other.

The uniform sampling approach, applicable only for synthetic scenes, is to take multiple samples of a single point on the surface from uniformly spaced directions. We do this for evenly spaced points in the domain of each chart. The right side of Figure 9 illustrates the kind of data produced by this method.

To produce the synthetic examples we used the Radiance ray tracer [War94], which produces radiance values which are not clipped to $(0, 1)$. The Radiance ray tracer also lets us take samples at arbitrary points and directions.

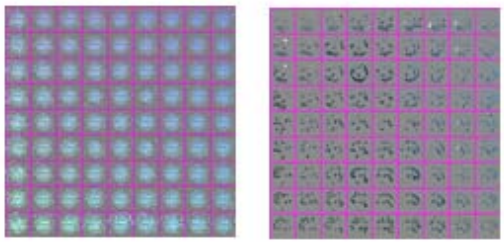

Figure 9. An illustration of the samples produced for a single chart. Each grid square corresponds to all of the samples for an area of the chart. Within the grid square the samples are mapped into the square using the local spherical projection. Points without samples are colored grey. This chart corresponds to roughly one sixth of the sphere. Left: Uniform samples. Right: Snapshot samples.

The real-world capture was accomplished using the techniques in [Gor]. The capture camera was positioned on a Faro arm, which provides accurate position information. The geometry was constructed by fitting a manifold surface (shown in Figure 17) to geometry found by volume carving [Gor]. (The emitted-light values we used were simply color-values from the digital camera; no conversion to radiance values was done. Despite this gross lapse, the results look reasonable, probably because the range of radiance values was relatively small.)

For the painted fish, we took nine snapshots then colored them by scanning in hand-drawn images. The input images were from the top, side, front, back, and side tilted up and down.

One thing to note is that where the charts overlap they get the same samples. In our surface implementation most parts of the surface have three overlapping charts. This has the disadvantage that we are keeping (and reconstructing) three copies of most of the samples. In our opinion this is well worth the cost for two reasons. First, the charts will agree for a large overlap region, preventing noticeable overlap boundaries. Second, most sampling and compression routines have problems with boundaries. Since the charts overlap so much, we can ignore the boundary artifacts because we will never see them (the blend functions will have gone to zero by the time we reach them).

7 Alternative representations and compression

We explored several alternative methods for representing the emitted-light function: wavelets, images, neural networks, and Gaussian mixture models. For four of these cases we need to use a local coordinate system for the spherical direction. There is a trade-off between using the same local coordinate system for the entire chart and using a different coordinate system for each surface point. If the local coordinate system changes too rapidly lose the coherency of the highlight directions. However, we can only use the same coordinate system as long as the vectors all fall within the same area. The local parameterization described in Section 4.2 can be extended to include more than just the upper hemisphere. In this way we can use the same coordinate system for more of the chart, or all of it if the surface is relatively flat.

Note that for both the wavelet and the image approach we rotate the edge chart (which is shaped like a diamond) using an affine projection to maximize the amount of space the chart takes up in the rectangular domain.

7.1 Wavelet

We experimented with a four dimensional Haar wavelet [SDS95]. The first two dimensions are the (s, t) position in the chart, the second two are coordinates (u, v) in the local frame. If the wavelet is divided into n divisions, we also use n local frames, one for each grid square. We use the method described in Section 5 to produce the evenly spaced samples.

To reconstruct, we project the input sample point using the local frame closest (on the surface) to the input point. We pass the surface position plus the projected point to the wavelet.

7.2 Image

We can also store the function as an image, where each grid square is a set of samples for a region of the surface (see Figure 9). The images can be compressed using JPEG or MPEG techniques. The two variables are the number of divisions of the chart, and the number of direction samples. In this case, we use one local frame for each of the chart grid squares.

To fill in the image, we first project all the samples into it, averaging when more than one sample falls into a pixel. To fill in the remaining pixels we use the technique in Section 5.

Image reconstruction is similar to the wavelet reconstruction except we interpolate between the sixteen (four charts by four directions) pixels surrounding the input sample. We use the local coordinate system of each of the four chart samples to determine which of their four direction samples to use.

Figure 10. Real and reconstructed images of a glass sculpture with non-spherical topology. From left to right: The real image of one of the original snapshots, the original snapshot reconstructed using LWR, then two novel images reconstructed using LWR.

7.3 Neural networks

Using a backpropagation network [MR86] with one hidden layer, we experimented with two different input methods. In the first case we used as inputs the surface domain point plus the actual direction vector (a five dimensional input) with between five and twenty hidden units. We also tried projecting the direction vector into a single local coordinate system (valid if the chart does not curve too much) resulting in a four dimensional input. The advantage of using such neural networks is that, once trained, evaluation is relatively fast and is a constant time operation. However, training the networks is often time-consuming, several hours to train on our sample sizes, and the quality of the learned model is very dependent on the parameters (number of hidden units, learning rate, *etc.*) selected.

7.4 Gaussian mixtures

In this approach, we model the function as the sum of a set of Gaussian models, each of which has the form:

$$G(p) = C_j \frac{e^{-(\sum_{0<i<4} \frac{(\mu_i - p_i)^2}{\sigma_i^2})/2}}{4\pi^2 \prod_{0<i<4} s_i}$$

We used a naive, greedy gradient descent procedure to fit the parameters, $\vec{\mu}$ and $\vec{s}$, to the data. We chose to try this method because it seems well-suited to modeling the sorts of variations that we expect to see in our functions.

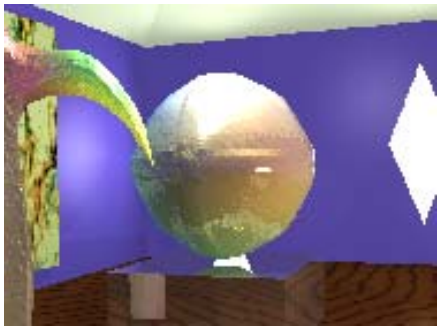

Figure 11. The ray-tracing world used for the synthetic experiments.

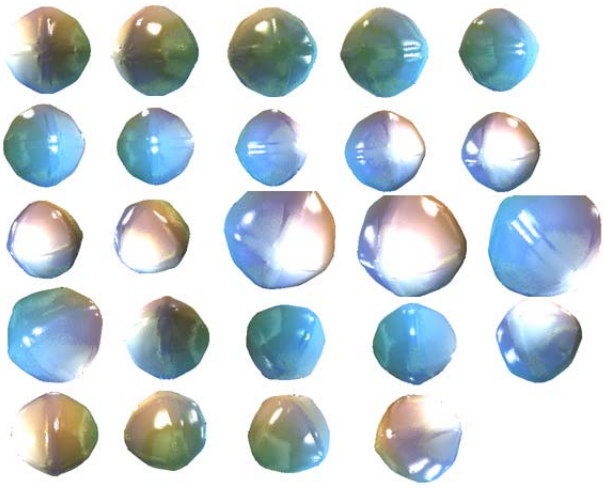

Figure 12. Twenty five of the twenty eight camera angles used for initialization (one of the remaining images can be found in the results image). Each image is 260 $\times$ 190 pixels.

8 Results

We first discuss the synthetic experiments, then the real-world data experiment, and finally the hand-drawn result. Storage sizes required for the various approaches are then summarized.

8.1 Synthetic

Figure 11 shows the world used for this experiment. There are two very bright windows at one end, an overhead light, and two flood floor lights. The floor and one wall are textured, there is a mirrored surface under the object, and a palm tree overhanging the object. The color on the sphere is constantly changing, while the glass sculpture is mostly uniform, with sudden changes. The sphere's geometry has both low and high frequency components while the glass sculpture is smooth.

We performed the same experiment on four different types of surfaces: metal, plastic, textured, and glass. Figure 12 shows the snapshots used to initialize the data; they cover the entire sphere and were placed by hand to simulate holding a real camera. The glass example used forty two images which varied in size but were mostly 355 $\times$ 356 pixels.

We used the YIQ conversion for every case except the textured sphere. Figures 13 and 10 shows the results of reconstructing using the original samples, as described in Section 5.

Approximate reconstruction times are summarized in Table 1. Although the glass sculpture's geometry is more complicated than the sphere's, there are fewer samples per chart, which accounts for the quicker rendering times.

Figure 14 shows some results of the Haar wavelet reconstruction. They suffer noticeably from discontinuity artifacts. On average, after removing coefficients whose contributions are negligible, there were between 60,000 and

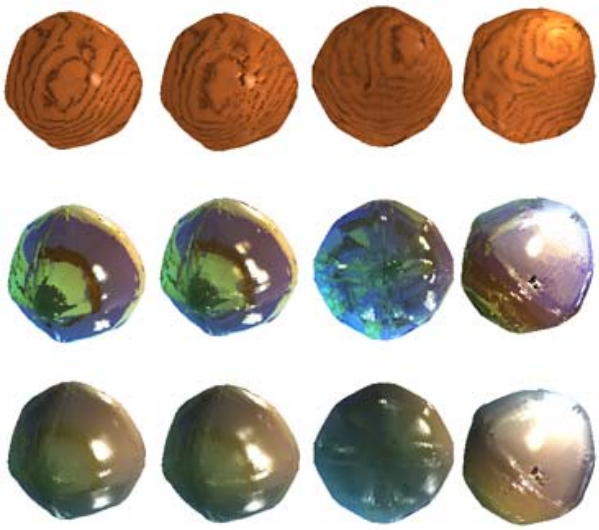

Figure 13. Real and reconstructed images of a "sphere" with both smooth and abruptly changing geometry. From left to right: The real image of one of the original snapshots, the original snapshot reconstructed using LWR, then two novel images reconstructed using LWR. From top to bottom: The plastic, metal, and textured spheres.

Model	LWR	Image	Wavelet
Plastic	870	1	13
Metal	870	1	15
Texture	870	1	12
Glass	95	1	5
Stuffy	120	1	6
Fish	83	1	

Table 1. Reconstruction results. Times are relative.

Figure 14. The types of artifacts introduced by discontinuous wavelets.

90,000 non-zero numbers kept per chart. Even if the rate of compression were higher, there are still two fixes to be made. The first is to use a smooth wavelet to remove the artifacts. Second, we need control over the size of the chart dimension versus the direction dimension. Experiments with the image reconstruction format indicate that the direction dimension needs to be almost twice the chart dimension for highly reflective objects, while the reverse is true for highly textured objects.

Figures 15 and 16 show the results for image reconstruction. Typical dimensions run between 8 and 25 divisions for the chart, and 10 to 35 divisions for the direction, which results in an image size from 200 $\times$ 200 to 875 $\times$ 875, with an average size of 250 $\times$ 250, since we rarely need both dimension to be high. There is a fair amount of latitude in choosing these dimensions, as indicated by the low to high image comparison. The number of divisions was specified by giving independent lower and upper bounds for both the chart dimension and the direction dimension. Within those bounds, the actual size (chart dimension $\times$ direction dimension squared) is chosen to be roughly the number of input samples.

We had marginal success with neural networks and Gaussian mixture models when using uniform samples, but were unsuccessful with snapshot samples. Adding regularly spaced samples (using the LWR reconstruction method) helps, but also increases the data size.

8.2 Real

Nineteen pictures of a teddy bear (called Stuffy) were taken using a Faro arm, which gives accurate camera positions. The pictures were used both as input images and to create approximate geometry of the object [Gor]. We constructed a manifold and fit it to the approximate geometry, as shown in Figure 5. Figure 17 shows several reconstructed images, four of which are the original images.

While reconstructing, we discovered that the camera matrices were not quite correct. This results in ghosting in the inbetween images, which can best be seen in the video. For this reason, the best reconstruction technique is LWR, since this only produces ghosting on inbetween images. Also, since there were only nineteen 640 $\times$ 480 images, most of which were only one quarter full, the actual data size and number of samples per chart is fairly small.

An example of the image reconstruction and the ghosting artifacts are shown in Figure 18.

8.3 Hand-drawn images

Our final example is a fish colored with nine hand-drawn images. This example illustrates the effects of varying the number of sample points interpolated for each direction. At one extreme we take only the single sample pointing in the correct direction (see Figure 19). This reproduces the hand-drawn images correctly, but shows a sharp line between the individual images. At the other extreme, we blend four to six sample points, resulting in a blending of the texture maps.

8.4 Storage requirements

Numbers are summarized in Table 2

For the LWR method we store the original samples (eight numbers per sample, the five input numbers and the color) and the bin information. The bin information is an $n \times n$ array of bit vectors each the size of the number of samples for that chart.

The images are currently stored as arrays of floats. The total size is the size of each image ($n \times n \times 3$) for each chart. Obviously, the data set would be much smaller stored as JPEGS. Initial tests indicate that the images can be compressed by as much as 80 $\%$ before noticeable is seen.

The wavelets are stored as arrays of floats. The total size is $3n^4$ where n is the number of divisions. About 20 $\%$ of the coefficients are negligible.

Object	N Charts	LWR Samples	Min	Avg	Max	Total size (approx)
Sphere	26	559,386	30,538	56,331	123,311	22,675,088
Glass	1200	1,324,783	0	2911	22,499	54,263,264
Stuffy	914	1,187,895	0	3614	19653	17,761,159
Fish	578	621,478	94	2835	12283	5,739,931
	N Charts	Avg per side	Image Total Size			
Sphere (hires)	26	625	15,234,375			
Sphere (lowres)	26	120	561, 600			
Glass	1200	160	46,080,000			
Stuffy	914	120	19,742,400			
Fish	578	80	8,670,000			
	N Charts	Avg per side	Wavelet Total Size			
Sphere	26	16	2,555,904			
Glass	1200	8	7,372,800			
Stuffy	914	4	350,976			

Table 2. Approximate data sizes in doubles. Images and wavelets are kept as arrays of floats

9 Future work

The compression techniques presented here are by no means a comprehensive list. As stated earlier, a smooth wavelet approach or a more elaborate energy minimization scheme might work. Also, hybrid approaches might be feasible, both between individual charts and within a single chart.

There is also the question of what to do when there are only a few, possibly mis-matched, images to interpolate. One possibility is to provide an explicit in-between function to create more samples. If we consider the camera locations as points on the sphere then we can create new images in places where there are no images by interpolating between just the three camera locations containing the new camera location. This would introduce some global knowledge about which samples to interpolate and how much to interpolate by.

In a different direction, there is no need for the geometry of the surface to exactly match the geometry of the object being captured. For example, we could use a simple blob to model a bushy plant. This would require an alpha mask in addition to a color in order to model the silhouettes properly. This could also be viewed as a form of model compression where we replace complicated geometry with simple geometry plus view-dependent information.

References

[AMS97] Christopher G. Atkeson, Andrew W. Moore, and Stefan Schaal. Locally Weighted Learning. *AI Review*, 11:11–73, 1997.

[Che95] Shenchang Eric Chen. Quicktime VR - An Image-Based Approach to Virtual Environment Navigation. In *SIGGRAPH 95 Conference Proceedings*, pages 29–38. Addison Wesley, August 1995.

[CL96] Brian Curless and Marc Levoy. A Volumetric Method for Building Complex Models from Range Images. In *SIGGRAPH 96 Conference Proceedings*, pages 303–312. Addison Wesley, August 1996.

[CON99] Brian Cabral, Marc Olano, and Philip Nemec. Reflection Space Image Based Rendering. *Proceedings of SIGGRAPH 99*, pages 165–170, August 1999.

[Deb98] Paul Debevec. Rendering Synthetic Objects Into Real Scenes: Bridging Traditional and Image-Based Graphics With Global Illumination and High Dynamic Range Photography. In *SIGGRAPH 98 Conference Proceedings*, pages 189–198. Addison Wesley, July 1998.

[DTM96] Paul E. Debevec, Camillo J. Taylor, and Jitendra Malik. Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach. In *SIGGRAPH 96 Conference Proceedings*, pages 11–20. Addison Wesley, August 1996.

[GGSC96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski, and Michael F. Cohen. The Lumigraph. In *SIGGRAPH 96 Conference Proceedings*, pages 43–54. Addison Wesley, August 1996.

[GH95] Cindy M. Grimm and F. John Hughes. Modeling Surfaces of Arbitrary Topology using Man-

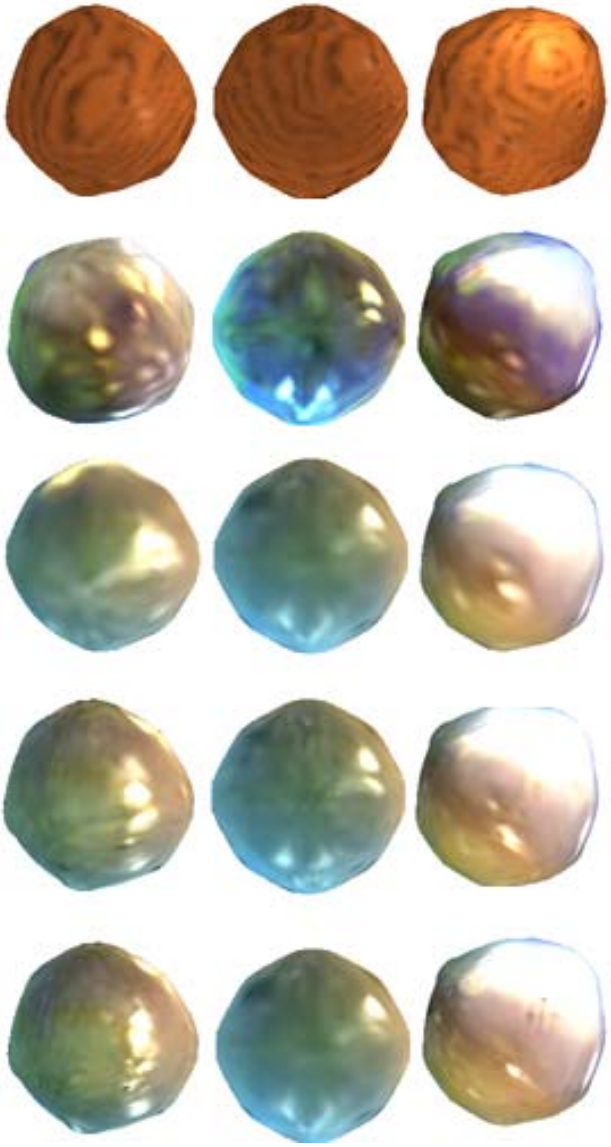

Figure 15. Examples reconstructed using image reconstruction. From left to right: an original snapshot followed by two novel snapshots. From top to bottom: plastic, metal, and textured.

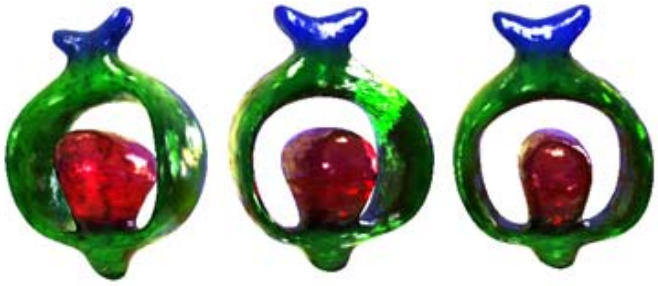

Figure 16. The glass sculpture reconstructed using image reconstruction. From left to right: an original snapshot followed by two novel snapshots.

Figure 17. A real-world example of a stuffed animal ("Stuffy") reconstructed from nineteen 640 $\times$ 480 images. From top left to bottom right: four novel views, four original views.

Figure 18. "Stuffy" reconstructed using the image and wavelet techniques. Blowups show the ghosting artifacts.

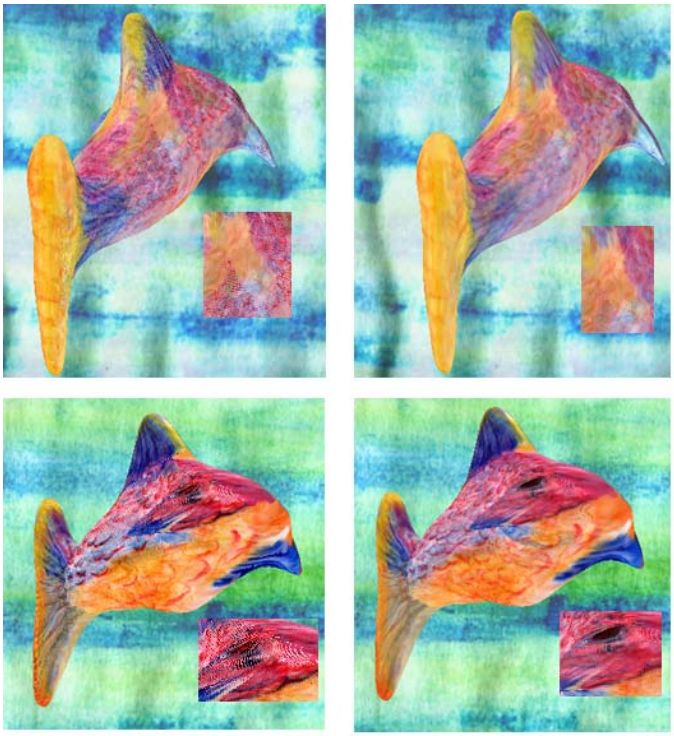

Figure 19. Left: Taking one sample. Right: Taking four samples.

ifolds. In *SIGGRAPH 95 Conference Proceedings*, pages 359–368. Addison Wesley, August 1995.

[Gor] Steven J. Gortler. Unpublished work submitted to SIGGRAPH 2000. We appreciate Gortler's having told us about this work.

[HKD93] Mark Halstead, Michael Kass, and Tony DeRose. Efficient, Fair Interpolation Using Catmull-Clark Surfaces. *Proceedings of SIGGRAPH 93*, pages 35–44, August 1993.

[LH96] Marc Levoy and Pat Hanrahan. Light Field Rendering. In *SIGGRAPH 96 Conference Proceedings*, pages 31–42. Addison Wesley, August 1996.

[MB95] Leonard McMillan and Gary Bishop. Plenoptic Modeling: An Image-Based Rendering System. In *SIGGRAPH 95 Conference Proceedings*, pages 39–46. Addison Wesley, August 1995.

[MR86] James McClelland and David Rumelhart, editors. *Parallel Distributed Processing*. MIT Press, Cambridge, MA, 1986. Volumes 1 and 2.

[MRP98] Gavin S. P. Miller, Steven Rubin, and Dulce Ponceleon. Lazy Decompression of Surface Light Fields for Precomputed Global Illumination. *Eurographics Rendering Workshop 1998*, pages 281–292, June 1998.

[SDS95] Eric J. Stollnitz, Tony D. DeRose, and David H. Salesin. Wavelets for Computer Graphics: Part 1. *IEEE Computer Graphics & Applications*, 15(3):76–84, May 1995.

[SGHS98] Jonathan W. Shade, Steven J. Gortler, Li-wei He, and Richard Szeliski. Layered Depth Images. In *SIGGRAPH 98 Conference Proceedings*, pages 231–242. Addison Wesley, July 1998.

[SH99] Heung-Yeung Shum and Li-Wei He. Rendering with Concentric Mosaics. *Proceedings of SIGGRAPH 99*, pages 299–306, August 1999.

[SS97] Richard Szeliski and Heung-Yeung Shum. Creating Full View Panoramic Mosaics and Environment Maps. In *SIGGRAPH 97 Conference Proceedings*, pages 251–258. Addison Wesley, August 1997.

[War94] Gregory J. Ward. The RADIANCE Lighting Simulation and Rendering System. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, pages 459–472. ACM Press, July 1994.

[YM98] Yizhou Yu and Jitendra Malik. Recovering Photometric Properties of Architectural Scenes from Photographs. In *SIGGRAPH 98 Conference Proceedings*, pages 207–218. Addison Wesley, July 1998.