Spiral Texture Mapping

Minglun Gong and Yee-Hong Yang

Department of Computer Science, University of Alberta, Edmonton, Alberta, Canada

Abstract

Spiral texture mapping extends and combines two main categories of texture mapping approaches, namely, displacement texture mapping and view-dependent texture mapping. A spiral texture contains two components: a view-independent one and a view-dependent one. The first component contains a layered depth image, which stores the depth information and the extracted diffuse color information of the underlining object. The second component can be considered as a set of images, which uniformly sample the highlights and reflections of the object from different directions. The rendering algorithm uses the first component to solve visibility problems, and then uses the second component to produce highlights and reflections. Since spiral textures can produce both parallax effects and non-diffuse effects, they can be used to represent shiny complex real objects or environments. In addition, a novel sampling technique, which is based on the shape of a spiral, is proposed in this paper and used to sample the view-dependent component. The new sampling scheme can evenly sample all rays that pass through a planar rectangular texture. The experimental results show that our approach can generate very good results with a reasonable number of samples.

Keywords: Image-based rendering, Texture mapping.

1 Introduction

Texture mapping [4] has long been used to enhance the realism of rendering results by adding two-

dimensional details to objects. It is considered as the earliest image-based rendering approach. The conventional texture mapping technique has two limitations: (1) the texture map cannot represent geometric details of the object, and hence, cannot produce parallax and self-occlusion effects; (2) it records only the appearance of the surface from a single direction, and hence, cannot generate non-diffuse effects, such as highlights and reflections.

Approaches have been proposed to address both limitations. The first limitation is alleviated by the introduction of the displacement texture mapping technique [6]. A displacement texture map specifies the amounts by which a desired surface deviates from the approximate models. Hence, it alters the shapes of the approximate models and adds geometric details. Displacement textures have been rendered using micro-polygons in Cook's original approach. Later on, other rendering approaches are proposed using ray-tracing [14], forward image warping [17], and backward image warping [19].

Displacement maps can only represent a restricted family of surfaces, which are continuous and non-folded along the mapping direction[1]. Hence, they cannot be used for some complex objects, where rays defined by the texture image have more than one intersection with the desired surface. This problem is addressed in the image-based object approach [16] by using layered depth images rather than depth images to define displacements. A layered depth image [22] is a view of the scene from a single camera view, but it stores more than one sample along each line of sight. An image-based object consists of six layered depth images that share a single center of projection and are mapped onto the six sides of the bounding cube of the object. Since visibility problems are solved, the image-based object can be used to represent more complex geometric objects. However, this approach makes no attempt to produce non-diffuse effects.

On the other hand, view-dependent texture mapping is proposed to handle the second limitation [8]. It

[1] In displacement maps of these surfaces, depth differences between adjacent pixels are always meant to represent a surface slope, and therefore, can be treated as being connected.

can generate non-diffuse effects according to the observer's viewpoint by compositing together textures obtained from multiple views. It can also provide limited level of parallax effects since the appearances of unmodeled geometry details are simulated. In their later work, Debevec, et al. [9] use visibility preprocessing, polygon view maps, and projective texture mapping to reduce the computational cost and produces smoother blending results.

Nonetheless, the sparse set of perspective images used in view dependent texture mapping cannot guarantee all view-dependent illumination and geometric details are captured. Therefore, it cannot fully solve the visibility problem and faithfully reconstruct the non-diffuse effects. The light field [11] and lumigraph [10] approaches fully sample the rays that pass through two planes: the camera plane and the focus plane. Therefore, they can be considered as densely sampled view-dependent textures defined on the focus plane. These approaches can produce both self-occlusion effects and non-diffuse effects correctly. However, they require too many samples, which are hard to acquire and to deal with.

In this paper, the first motivation of our proposed technique is to combine together the above two research directions, i.e., using depth information to produce correct projections and using multiple samples to generate non-diffuse effects. Our second motivation is to propose a novel sampling technique, which is based on the shape of a spiral, to evenly sample all rays that pass through a planar rectangular texture.

The spiral texture we defined contains two components: a view-independent one and a view-dependent one. The first component is an extension of displacement texture. In this part, we adopt the idea of using layered depth image to depict the geometric details of the object. However, instead of keeping the illumination of the object that is observed from a single direction, we use layered depth image to store the extracted diffuse colors of the object. The second component is an extension of view-dependent texture. In this part, we try to sample the texture from different directions evenly and densely enough. The color we keep here is the non-diffuse portion only, i.e., the difference between the color that is observed from the given direction and the diffuse color portion. Since we do not rely on the second component to solve visibility problems, a much smaller sampling density can be used.

2 Related Works

Besides research mentioned earlier, there are other works also related to our approach. For example, some approaches try to use depth information to produce sharp rendering results for sparsely sampled scenes [20,21]. However, these approaches use per-image depth maps. In our opinion, there are a lot of redundancies in per-image depth maps, and they are hard to compress since only lossless compression techniques can be applied. Our approach treats geometry of the object as view-independent information and only samples it from one view direction.

The idea of separating the view-independent part and view-dependent part comes from Lischinski and Rappoport's approach [12]. Their approach represents simultaneously and separately both view-independent scene information and view-dependent appearance information of a synthetic scene. Our approach extends the idea to real scenes, and therefore, we have to handle the problem of separating view-independent and view-dependent colors for real scenes. In addition, using our new sampling scheme, we demonstrate how to sample the view-dependent information uniformly.

At least two approaches, the two-sphere parameterization and the sphere-plane parameterization, have been proposed to address the uniformly sampling problem [3]. Nevertheless, these two parameterizations are designed to sample the rays that pass through a sphere, and hence, cannot uniformly sample rays that pass through a planar texture. Many real world objects have large length-to-width ratio. Therefore, using a sphere to enclose such an object and then sample the sphere will generate many useless samples, i.e., rays that do not intersect with the object. Using six or even more rectangles to enclose these objects can give a much tighter bound than using a sphere, and hence, can reduce the number of samples needed. In addition, using planar texture as primitive also gives users more flexibility since they can choose to represent only the selected side of the object or environment. Another difference between the two sampling schemes is the way to sample the directions. The polyhedron subdivision approach used in [3] has the drawback that it cannot use an arbitrarily specified sampling rate. The number of samples increases four times for each subdivision process.

Our approach also has a strong relationship with the surface light field approach [23] since we all try to handle shiny real objects. Both approaches start from a geometric model of the object and a set of fully calibrated photos. Both approaches resample these photos to generate an intermediate representation for the appearances of the object. The most distinct difference is that in surface light field the intermediate representation is defined in the object space, while in our approach it is defined in the image space. Therefore, the rendering time for surface light field depends on the complexity of the geometric model, while ours does not. Furthermore, in the surface light field approach, even though the directions of the rays are uniformly sampled using subdivided octahedrons, the positions of the rays are not uniformly sampled. In their case, the sampling rate for ray positions actually varies with the view direction and the tessellation of the model. In contrast, our approach can uniformly sample both in the directions and in the positions of the rays that pass through the object.

As a texture mapping technique, our approach is also related with approaches that try to synthesize bidirectional texture functions [7,13]. However, we are working on different levels of scales in geometry. The bidirectional texture functions are mainly used for the mesostructure level, such as bumps and dents. The spiral texture is mainly used for the macrostructure level. That is, we can use the spiral texture defined on a simple rectangle to replace the complex model of an object. However, our current approach also does not try to sample the light directions, and therefore, we do not support relighting.

The organization of this paper is as follows. In the next section, the sampling scheme used for spiral texture is discussed. The generation and rendering processes for spiral texture is described in Section 4 and Section 5, respectively. Section 6 presents the experimental results. The paper concludes in Section 7 with discussions on future work.

3 The Spiral Sampling Scheme

To sample the rays that pass through the planar texture surface, we first try to evenly sample all possible

view directions, which is the hemisphere that covers the planar texture. Then, we try to sample the projection positions adaptively according to the view direction to provide constant sampling rate.

3.1 Uniformly Sample the View Directions

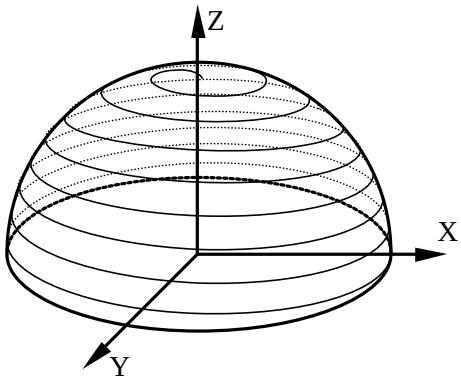


Figure 1: Using spiral to sample the projection direction uniformly.

As shown in Figure 1, to sample the directions, we use a one-dimensional spiral curve to cover the two-dimensional hemisphere first. Points that are spaced at equal distance along the curve are then used to sample the spiral itself. The parametric function of the spiral can be defined as:

$$\begin{cases} x(a) = \sin(ka)\cos(a) \\ y(a) = \sin(ka)\sin(a) \\ z(a) = \cos(ka) \end{cases} \tag{1}$$

where a is the spiral parameter, k the parameter that controls the density of the spiral curve.

The above equation defines a spiral that starts from the top of the hemisphere. Any given point in this spiral defines a direction, whose yaw angle is a and pitch angle is ka. Clearly, the pitch between adjacent lines of the spiral is $2kp$. The length of the spiral can be computed using the following integration:

$$L(a) = \int_0^a \sqrt{(x'(q))^2 + (y'(q))^2 + (z'(q))^2}\, dq = E(ka, -k^{-2}) \tag{2}$$

where $E(f,m)$ is the elliptic integral of the second kind, which is defined as:

$$E(f, m) = \int_0^f \sqrt{1 - m\sin^2(q)}\, dq = \int_0^{\sin(q)} \frac{\sqrt{1 - mt^2}}{\sqrt{1 - t^2}}\, dt \tag{3}$$

To cover the whole hemisphere, the range of the parameter a should be [0, p/(2k)]. To keep the sampling rate along the spiral the same as that across the spiral, the spiral curve is sampled with a set of points that are spaced at distance 2kp. Therefore, the total number of samples needed can be calculated by:

$$n(k) = \left\lfloor \frac{L(p/2k)}{2kp} \right\rfloor + 1 = \left\lfloor \frac{E(p/2, -k^{-2})}{2kp} \right\rfloor + 1 \qquad (4)$$

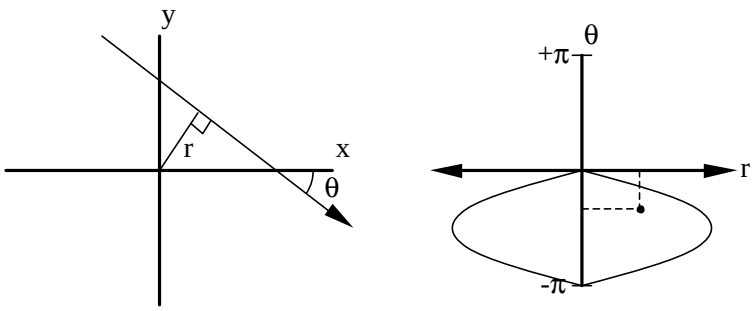
The spiral parameter, a, for any given sample f can be then calculated by solving the following equation:

$$L(a) = f \times 2kp \qquad 0 \le f < n(k) \qquad (5)$$

In practice, after the sampling rate, k, is decided, we pre-compute and tabulate the mapping relation between sample f and spiral parameter a. Normally, the value of k we use varies from 1/16 to 1/32, and the corresponding numbers of samples are within the range of [42, 164]. Hence, very little memory is required to store the mapping table.

Clearly, when k is small, i.e. the sampling is dense enough, the above sampling scheme will give an equal number of samples per unit solid angle.

3.2 Uniformly Sample the Projection Position



(a) (b)

Figure 2: A ray under (a) the Cartesian space and (b) the line space.

Now, we need to consider how to evenly sample the positions of the rays. First, we illustrate the problem in the two-dimensional case. Figure 2 shows a two-dimensional ray in both the Cartesian space and the corresponding line space. In the line space, each ray is represented by a point and each set of rays by a

region [11]. Assume that we want to sample the set of rays that pass through a line segment on the X-axis. It is easy to understand that the corresponding region in the line space for this set of rays is the region enclosed by two sine curves, as shown in Figure 2(b). Therefore, under the ideal sampling scheme, the corresponding points of the sampling rays should populate this region uniformly. Obviously, the two-plane parameterization, in which a fixed number of samples is used for different directions, will over-sample the areas close to q=0 and q=p (as shown in Figure 2). To avoid this, we should determine the number of samples needed adaptively according to the length of the segment's projection along different directions.

The extension to three dimensions is straightforward. To uniformly sample the set of rays that pass through a rectangle on the X-Y plane, we should adjust the number of samples needed according to the area of the rectangle's projection on the plane that is perpendicular to the view direction. To do so, assuming that the texture is sampled using $M \times N$ pixels in the perpendicular direction, for any given view direction (x,y,z), the number of pixels we used is $R_x M \times R_y N$, where R_x and R_y are the scale factors and are defined as:

$$\begin{cases} R_x = \sqrt{\dfrac{z}{\sqrt{z^2 + x^2 y^2}}}\sqrt{1 - x^2} \\ R_y = \sqrt{\dfrac{z}{\sqrt{z^2 + x^2 y^2}}}\sqrt{1 - y^2} \end{cases} \tag{6}$$

The scale factors are defined to provide the following two properties:

- The number of samples used for a given direction is proportion to the projection size of the rectangle along the direction, i.e.:

$$R_x \times R_y = \frac{z}{\sqrt{z^2 + x^2 y^2}}\sqrt{1 - x^2}\sqrt{1 - y^2} = z$$

- The number of samples along x(y) direction will not change if the projection is perpendicular to the x(y) axis, i.e.:

$$R_x = 1 \Leftrightarrow x = 0 \quad R_y = 1 \Leftrightarrow y = 0$$

3.3 Comparison with Two-Plane Parameterization

Now let us try to compare the numbers of samples needed for the spiral texture parameterization and for the two-plane parameterization under the same sampling density requirement. Again, we assume that the texture is sampled using $M \times N$ pixels in the perpendicular direction. For the spiral texture parameterization, the total number of samples required can be computed by:

$$m = \frac{\int_0^{p/2k} M \times N \times z(q) \times \sqrt{(x'(q))^2 + (y'(q))^2 + (z'(q))^2}\, dq}{2kp}$$
$$= \frac{M \times N \sqrt{1+k^2}}{4k^2 p} + \frac{M \times N}{4p} \operatorname{arctanh}\left(\frac{1}{\sqrt{1+k^2}}\right)$$

For the two-plane parameterization, in order to use a single light slab to cover the whole hemisphere[2], one of the defining planes has to be put at infinity. Under the same sampling density requirement, i.e. the directional space is sampled at $2kp$ interval and the maximum number of samples needed per direction is P, the total number of samples required can be computed by:

$$m' = \frac{p}{2kp} \times \frac{p}{2kp} \times M \times N = \frac{M \times N}{4k^2}$$

The ratio between the numbers of samples needed by these two parameterization schemes can be calculated by:

$$h = \frac{m'}{m} = \frac{p}{\sqrt{1+k^2} + k^2 \operatorname{arctanh}\left(\frac{1}{\sqrt{1+k^2}}\right)}$$

The above equation shows that when $k \to 0$, the ratio $h \to p$. That is, when the hemisphere is sampled dense enough, the number of samples needed under the new parameterization scheme is $1/p$ of the

[2] Use more than one light slab to cover the hemisphere need even more samples.

number of samples needed under the two-plane parameterization scheme. In practice, when k is within the range of [1/16, 1/32], the corresponding value of h is within the range of [3.09, 3.13].

4 Generate Spiral Textures

To generate spiral textures, we need to sample the underlining object. Here, we will mainly discuss how to create spiral textures for real objects. Generating spiral textures for synthetic scenes is similar and more straightforward.

Similar to the surface light field approach, we need to have the geometric model and some photos of the object. The geometric model can be obtained from a range scanner. The photos should sample the object from different angles and the camera should be calibrated at each location. Depending on the application, the user can decide on how many spiral textures to approximate the object and where to place them. A support rectangle is then specified for each texture using the coordinates of the four corners of the rectangle. The support rectangle of the texture serves as the image plane for both the view-independent component and the view-dependent component. The user can also specify the resolution, a.k.a. sample rate, for each texture.

4.1 Generate View-Independent Component

The view-independent part of a spiral texture can be considered as a layered depth image. Depending on the application, the user can choose either a parallel projection or a perspective projection. If the latter one is picked, the center of projection should be supplied by the user.

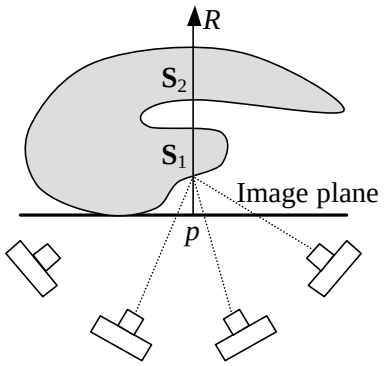


Figure 3: Sample the view-independent component.

Here what we need to do is to find out both the depth information and the diffuse color information for each pixel of the layered depth image. As shown in Figure 3, to obtain the depth information for a given pixel *p*, we can simply send out the corresponding ray *R* from the image plane and find the intersections between *R* and the geometric model of the object. During the intersection calculation, only surfaces that are facing towards the image plane are considered. The distances between these intersections and the image plane are computed and quantized according to the maximum depth available. The quantized values are then stored in the layered depth image in ascending order.

To find the diffuse color information of each intersection requires more computations. The idea here is to find out what the colors of the intersection are in different photos, and then to extract the diffuse color portion from these colors. To do so, we first send out rays from the intersection **S** to the viewpoints of all photos available to test whether **S** is visible in these photos. The color in the photo where **S** is visible is then used as a color sample of intersection **S**. A histogram is used to maintain the distribution of these color samples. After all the photos are processed, the histogram is used to extract the diffuse color information for the intersection.

Different strategies can be used to extract the diffuse color portion from the distribution. For example, we can use the mean, median, or the mode of all color samples. We found that the color distribution of a given intersection has a heavy tail feature, i.e., a small set of samples has much higher intensities than the rest of the samples. Therefore, the median or mode should be used since they are insensitive to the presence of these high value samples. In the experiment shown in this paper, the median is used.

4.2 Generate View-Dependent Component

The view-dependent part of the spiral texture can be considered as a set of images with different resolutions. No matter which kind of projection is used in the view-independent part, we always use parallel projection here.

According to the sampling rate requirement, the number of images needed is determined using equation 3. To generate any one of these images, we first find the corresponding spiral parameter, a, using the lookup table, which is pre-generated by solving equation 5. The direction of the projection, **d**, is then calculated using equation 1. Now, we can compute the scale factor according to **d** using equation 6. Finally, by multiplying the scale factor and the user-specified resolution together, we know at which resolution we should sample the image plane for this view direction.

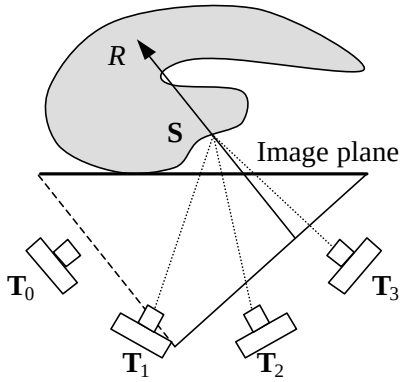


Figure 4: Sample view-dependent component.

Now we need to calculate the non-diffuse color information for each pixel in the image. The idea here is to find out what the color should be for the pixel first, and then subtract the diffuse portion from it. To do so, for any given pixel in the image, we shoot a ray R from the image plane along the direction of projection **d** to intersect with the object. Different from the view-independent component, here we only need to find the closest intersection, **S**, between R and the geometric model of the object.

If such an intersection, **S**, is found, we need to find what color it should have from the photos we have. The k-nearest neighbor interpolation approach [2] is adopted here. Assume the intersection is visible in photo i, whose position is $\mathbf{T}_i$. We calculate the dot product between **d** and ($\mathbf{T}_i$-**S**). Then, we try to find

four photos that give the highest dot product values and use three of them for interpolation. Assuming the fourth photo has the smallest dot product value, which is w, the weights for each of the first three photos are calculated by $\mathbf{d}\cdot(\mathbf{T}_i-\mathbf{S})-w$. The weighting technique guaranties a smooth blending since the weight of a photo will drop to zero before it is no longer used for interpolation.

The color we get from interpolation tells us what the intersection **S** looks like from direction **d**. The diffuse color portion of intersection **S** can be found by simply projecting **S** to the layered depth image we computed above. If multiple textures are used to sample the object from different directions, which layered depth image is used depends on the normal of the surface at location **S**.

Now we can subtract the diffuse color portion from the color of the intersection so that only the non-diffuse portion is left. Since the color of the intersection viewed from the current direction may have lower intensity than the diffuse color, the difference can be a negative value. To sample the difference using 8 bits, we define a color for zero difference globally. This color is added to the color difference and the result is clamped to within the range of [0, 255]. Which color should be used for zero difference value depends on the average brightness of the object's diffuse colors. In the experiment shown in this paper, the color (32, 32, 32) is used.

5 Render Spiral Textures

The above generation process creates an intermediate representation of the object, and therefore, the geometric model and photos of the object are no longer required. The rendering process for spiral textures can also be separated into two steps: rendering the view-independent component and rendering the view-dependent component. This is a nice feature since we can improve the rendering result progressively. That is, the rendering system can choose to perform only the first step when high interaction rate is required, and complete both steps when high quality rendering result is needed.

5.1 Render View-independent Component

Rendering the view-independent component is similar to the problem of rendering the layered depth

image. Since we allow multiple textures defined for the same object, the rendering algorithm should be able to find the closest point from multiple layered depth images. This brings an extra problem to forward image warping approaches since the occlusion compatible order [15] does not exist among images taken at different centers of projection. In [12], this problem is solved using Z-buffer and depth comparison. As a result, more memory is needed and an additional pass is required to scan the warping result and to fill holes. In image-based object [16], the problem is solved by requiring the six layered depth images to share the same center of projection. This is a good idea, except it may not work well when more than six planes are used to approximate the underlining object more closely. Using a common center of projection for all these surfaces may introduce large distortions.

To avoid the above problems, here we use a backward searching approach, which is similar to the image-based ray tracing approach [5,12]. For completeness, the algorithm is described below.

Based on the layered depth image, a three-dimensional volume is created using the boundaries of the image, the image plane, and the maximum available depth. For a given testing ray, we clip it with this volume first. We then try to visit the pixels in the layered depth image that lie in the projection of the testing ray, in either row dominant order or column dominant order according to the orientation of the testing ray. At each pixel location, we calculate the depth interval spanned by the testing ray. The depths of samples that are stored at this pixel are compared against this interval. If only one sample is found whose depth lies inside the interval, it will be the ray-surface intersection. If multiple samples fall inside the interval, the one closest to the ray origin will be used.

To accelerate the search process, the quadtree structure is used. A leaf in the quadtree links to the corresponding pixel of the original layered depth image. An internal tree-node stores the minimum and the maximum depth of all samples that are within the corresponding region. For a given testing ray we can find out whether an intersection exists by a depth-first traversal of the quadtree. Whenever we encounter an internal tree-node whose depth range is disjoint with the depth interval of the ray, the entire sub-tree can be safely skipped. If the two depth intervals overlap each other, we traverse the children

recursively. In order to encounter the closest intersection first, the four children are visited in the same order as the ray does.

5.2 Render View-dependent Component

As shown in Figure 9(b), the rendering results of the view-independent component gives us the correct projection of the object viewed from a given viewpoint. For Lambertian surfaces, this is already good enough. For shiny surfaces, we need to render the view-dependent component to add highlights and reflections.

Rendering view-dependent component is basically an interpolation process. For a given ray, we need to find the closest rays sampled and interpolate them. How to find the closest samples depends on the sampling scheme used. In two-plane parameterization, since rays are not uniformly sampled, the closest samples can be found easily by intersecting the ray with the two planes. When the directions are sampled using polyhedron subdivision approach [3], searching for the closest samples involves calculating the intersection between the ray and the polyhedron, followed by a recursive subdivision process to locate the patch. When an unstructured sampling scheme is used [8], even more computation is needed. Basically, we need to compare the given ray with the corresponding rays in all the photos available.

In our approach, using the property of the spiral parameterization scheme, we can find the closest available samples using a direct and efficient procedure. The procedure can be best explained as a two-step process. Considering the view-dependent component as a set of parallel projected images, we first try to find which images sample the texture from the closest directions. Then we try to determine which pixel should be used within each of these images.

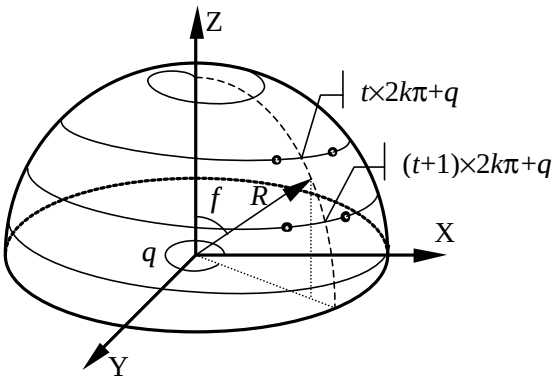


Figure 5: Find the closest directions sampled.

As shown in Figure 5, the first step starts from converting the given ray *R*, defined in the texture's bcal coordinate using a vector, into two angles, i.e., the yaw angle q and pitch angle f. Then, we can estimate how many rounds the spiral has rotated on the hemisphere before it reaches this direction using the following equation:

$$t = \left\lfloor \frac{f - q \cdot k}{2kp} \right\rfloor$$

Our approach tries to find four closest samples and interpolate them. As shown in Figure 5, two of the samples lie on the t^{th} round of the spiral curve, and the other two samples lie on the t+1th round of the curve. Obviously, the spiral parameter a for the first two samples should be close to and on the two sides of value $t\times 2k\pi+q$. The other two samples should have a values close to and on the two sides of the value $(t+1)\times 2k\pi+q$. Therefore, we can search within the lookup table to find the images that sample these directions.

In the second step, we need to find out which pixel we should use within each of these images. Instead of determining the pixel directly using the parameter of the given ray [3,11], here we use depth correction to accommodate for the amount that the real object deviates from the approximate rectangle.

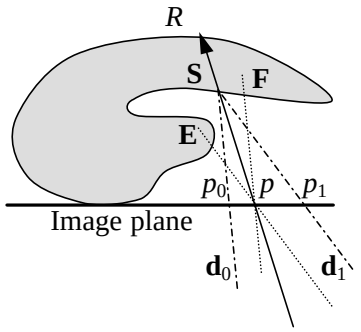


Figure 6: Find the right pixels for interpolation.

Figure 6 shows a cross section of the spiral texture. R is the ray we want to compute and it intersects the object at location **S** and the image plane at p. Assume I_0 and I_1 are two of the four images we found, which sample directions $\mathbf{d}_0$ and $\mathbf{d}_1$, respectively. I_0 and I_1 have the same image plane, which is shown in Figure 6, but have different resolutions due to different project directions. If we try to interpolate the pixels corresponding to location p in images I_0 and I_1, we are actually interpolating points **E** and **F** on the object. Therefore, incorrect highlight or reflection may be generated. To solve this problem, we should try to interpolate the pixels that correspond to locations p_0 and p_1 in images I_0 and I_1, respectively. Clearly, how far p_0 and p_1 deviate from p depends on both the depth of the intersection and the sampling rate. The location of p_0 and p_1 can be found simply by sending rays from **S** along $\mathbf{d}_0$ and $\mathbf{d}_1$, and calculate their intersections with the image plane.

6 Experimental Results

The "fish" data used in surface light field is used here for experiments [23]. This dataset contains a geometric model and 638 useable photos of a small ceramic fish. The geometric model of the fish, obtained by a range scanner, contains 129664 triangles.

In our experiment, five spiral textures are generated for the five sides of the fish, excluding the bottom. According to the aspect ratio of the fish, 270×310 pixels are used to sample the front and backsides, 183×310 for the left and right sides, and 270×183 for the topside. These resolutions are actually higher than the size of the fish shown in the original photos in order to avoid aliasing problem.

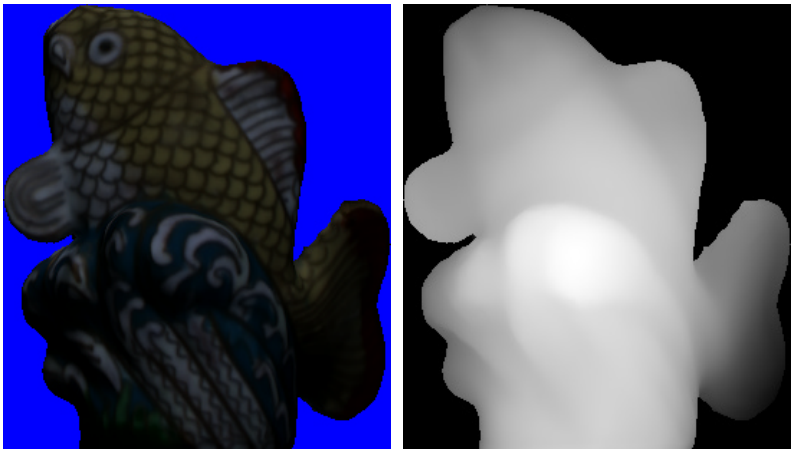


Figure 7: The view-independent component of the spiral texture.

Parallel projected layered depth image is used for the view-independent component of each texture. Two layers are used for the left and right sides, and one layer is used for the front, back, and top sides. The depth information is quantized using 8 bits. The total size for the view-independent components of the five textures is 1.3MB before compression, and 460KB after LZW compression. Figure 7 shows the diffuse color information and depth information for the front side texture.

Two different sampling rates are experimented for the view-dependent component. For a low sampling rate, the value of parameter k is equal to 1/16, which yields 42 sample directions for the hemisphere. For a higher sampling rate with k equals to 1/24 yields 93 samples. This means that we sample the unit sphere using 84 and 186 directions, respectively. This is much lower than that is used in uniformly sampled light fields [3], where 64K and 245K samples are used to sample the unit sphere. It is also comparable with that is used in ref [12], where 66, 258, and 1026 directions are used.

Before compression, the total size for the view-dependent components of the five textures is 20MB with the low sampling rate and 44MB with the high sampling rate. Since only non-diffuse portions exist, these images are not as colorful as the original photos. Hence, we can apply color compression, which will bring the size down to 6.7MB and 14MB, without noticeable quality degradation (see Figure 8). We can also apply vector quantization with 12-dimensional vectors, which can compress the data further to 1.7MB and 3.5MB.

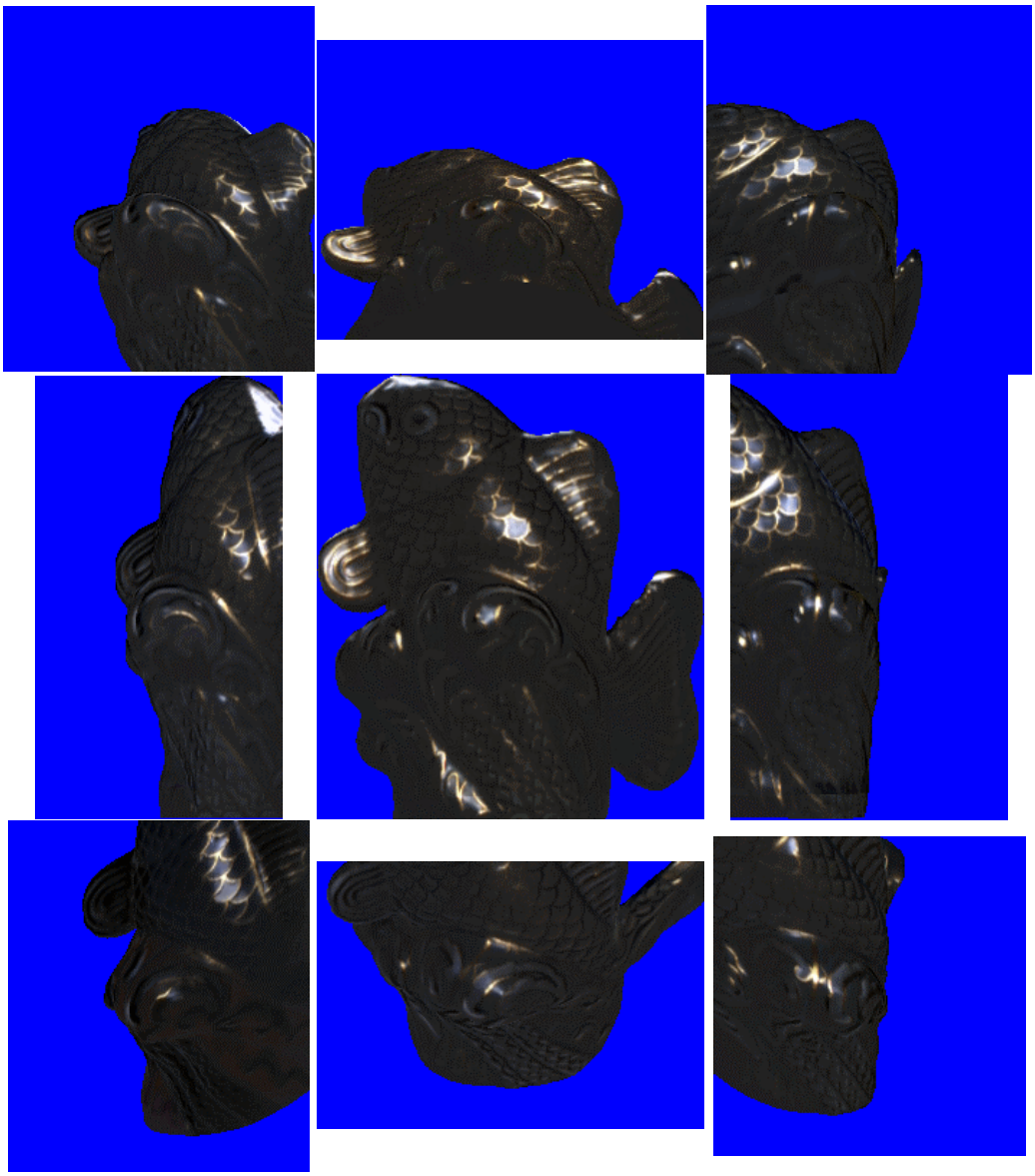
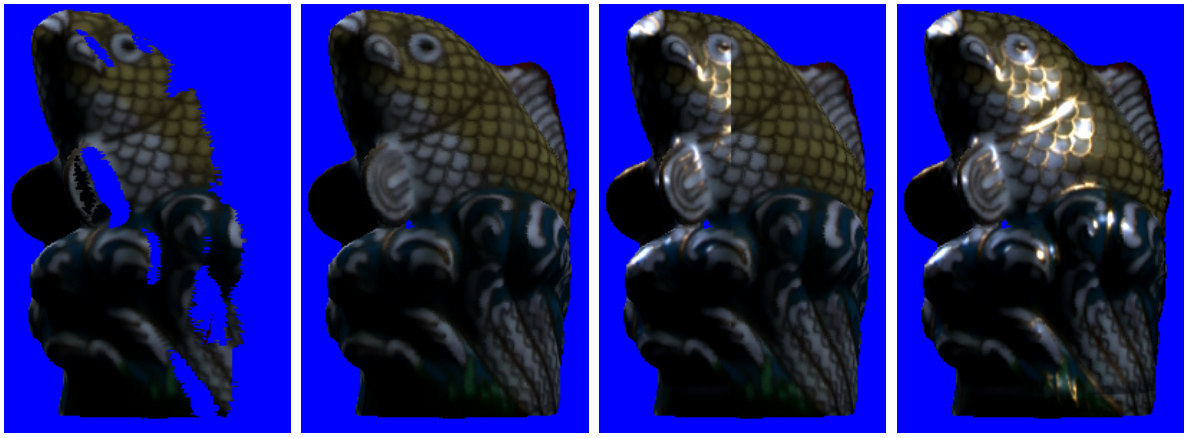


Figure 8: Several samples of the view-dependent component (after color compression).

Figure 8 shows several samples of the view-dependent component[3]. It shows that different samples have different resolutions. It is interesting to observe that due to adaptive sampling, the fish shown in these photos are not as distorted as the Buddha shown in ref [11], which is sampled using the two-plane parameterization scheme.

---

[3] For illustration purpose only, blue color is used here to indicate no intersections, instead of the color for zero difference we defined.



(a) (b) (c) (d)

Figure 9: Rendering process (a)(b) use view-independent parts only (c)(d) add view-dependent parts.

Figure 9 shows the decomposed rendering process. Figure 9(a) is generated using the view-independent component of the left texture only. Figure 9(b) uses both left and front textures, which gives the correct projection of the fish from this view direction. The view-dependent component of the left texture is added in Figure 9(c), in which only half of the fish has highlights. The view-dependent part of the front side is added in Figure 9(d). We can notice that the highlights obtained from the two different textures are seamlessly connected.

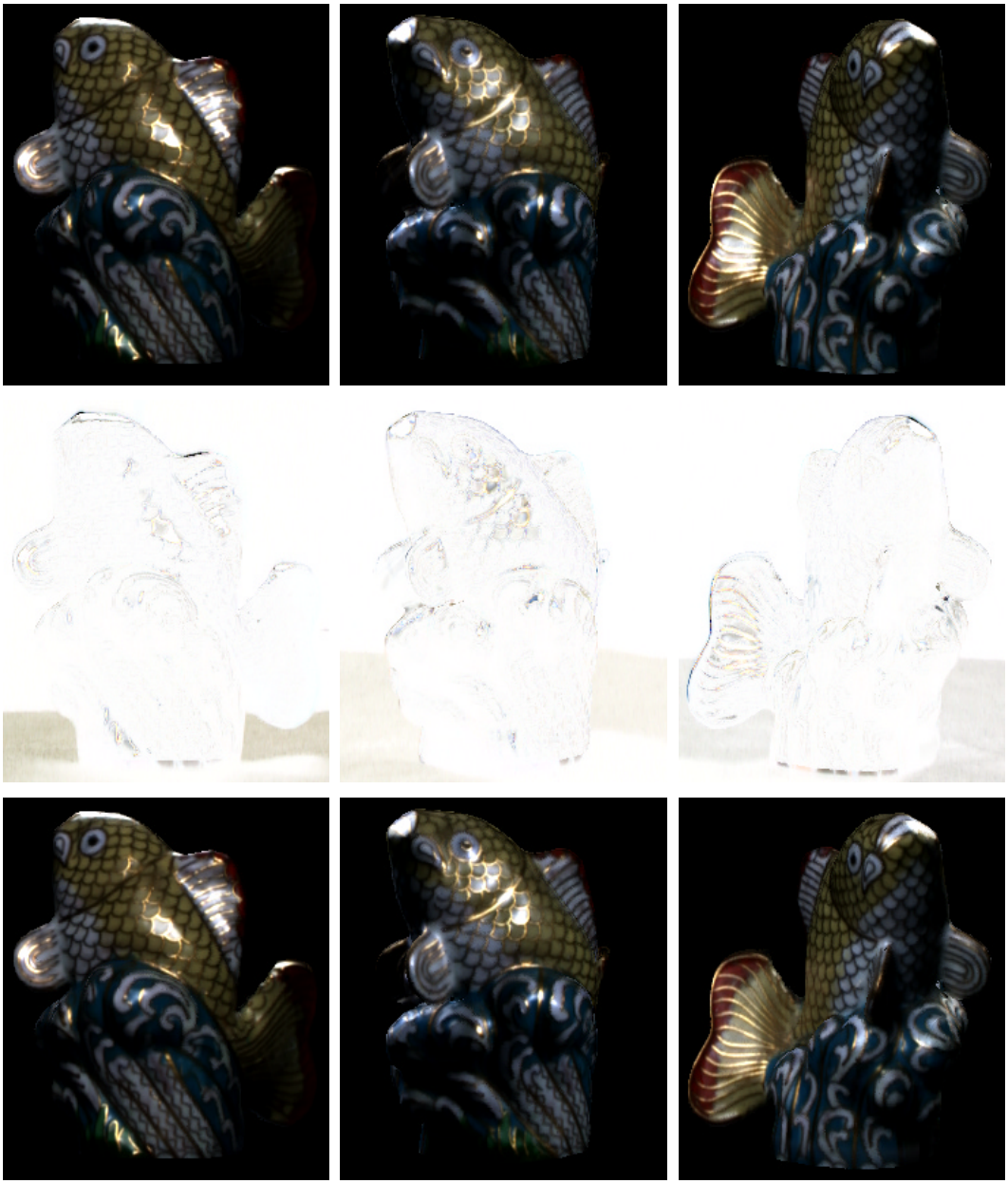


Figure 10: Rendering results comparison for different views.

Figure 10 shows the rendering results from different viewpoints. The camera settings we used here are the same as three of the photos in the original database. As a result, we can use the original photos as ground truth to evaluate our algorithm. Images in the first row of Figure 10 are the original photos. Those in the second row are the rendering results using the high sampling rate version. The third row shows the differences between images in the first two rows. The darker the color, the larger the error is. It is noteworthy that our algorithm can faithfully reproduce the appearances of the shiny fish in all

directions. The last row shows the rendering results using the low sampling rate version. It shows that dropping half of the view-dependent samples does not significantly degrade the quality of rendering results. If we exam the right image in the last row, we can notice that some highlights on the tail of the fish are lost from this particular view direction. However, it is interesting to see that the highlights for the golden stripes on the tail are preserved. We believe that the reason for this phenomenon is: the highlight for the ceramic has a higher frequency than that for the golden stripes. Therefore, the highlights for the ceramic are not captured in our low sampling rate version, while the highlights for the golden stripes are.

(a) (b) (c) (d)

Figure 11: Rendering results under different compression approaches.

When compressions are applied to the view-dependent component, only the highlights and the reflections of the object are affected. Figure 11 compares the results with different compressions schemes using an area with both highlights and inter-reflections (on the body of the fish). Figure 11(a) is the original photo. Figure 11(b) is the result generated using uncompressed view-dependent component, which faithfully reproduces the details of the original photo, even though not as sharp. Color compression is used in Figure 11(c), where the result looks identical to the uncompressed version. In Figure 11(d), the view-dependent component is compressed using 12-dimensional vector quantization. For vector quantization, we generate a codebook with 256 codewords for each of the five spiral textures. The result shows that the quality of the image does not degrade too much.

To render the fish with output resolution of 240×320 pixels, our current implementation takes about 0.3

second on a Pentium 4 PC running Windows 2000. About 0.1 second is used to render the view-independent component and 0.2 second for the view-dependent component.

7 Conclusions

In this paper we propose a new image-based rendering technique, the spiral texture mapping. The new approach combines the ideas of using depth information to solve visibility problems (displacement texture mapping) and using multiple samples to produce non-diffuse effects (view-dependent texture mapping). It also integrates the idea of separating the view-independent information and the view-dependent information within the combination. As a result, the visibility problems are solved solely based on the depth information in the view-independent component, while the non-diffuse effects are produced through interpolating the view-dependent component.

Comparing to existing image warping based approaches [16,17,19], our main improvement is adding another view-independent component. Therefore, we can handle shiny objects or environments. When using multiple spiral textures to fully represent an object, the view-dependent component is similar to the image-based object since both approach use multiple layered depth images. However, these two approaches are not exactly the same since: (1) we allow the use of parallel projection, and therefore, the user can use a more complex shape to closely approximate the object; and (2) we store the extracted diffuse colors rather than the colors that are observed from a single direction.

Although our approach does sample the appearances of the scene or object from different directions, it requires much less samples than light field or lumigraph related approaches [3,10,11]. This is because (1) under the same sampling rate requirement, the sampling scheme we propose needs less than one third of the samples needed by the two-plane parameterization scheme; (2) the use of depth information and the separation of view-independent and view-independent information make it possible to generate acceptable results under much lower sampling rate.

A novel sampling scheme is also proposed, which is another contribution of this paper. The new

sampling scheme, which is defined based a spiral, can uniformly sample both the directions and the positions of rays that pass through a planar rectangle texture. Therefore, we can always use multiple rectangles to approximate complex objects and samples these rectangles. Since for some objects, using six or even more rectangles to enclose them can gives much tighter bound than using spheres, less useless samples exist in our new sampling scheme than those in the two-sphere parameterization or sphere-plane parameterization.

It is noteworthy that using multiple textures to sample an object will not introduce any redundancy, as long as the corresponding support rectangles form a convex solid. This is because even though the two adjacent textures may sample the same direction, the projected positions they sample do not overlap each other. As a result, using multiple textures may actually decrease the number of samples needed if such a setting can provide a tighter bound for the object.

Spiral textures have many applications. For instance, we can use spiral textures to depict shiny real objects and put them in the virtual scene. We can also use spiral textures to represent environments and use them as image portals in indoor walkthrough applications. In the later case, our approach can produce more realistic rendering results than existing approaches [1,18] since the highlights of the scene are preserved.

The shiny fish, which is used in surface light field, is used in our paper to demonstrate the first application. The rendering results using uncompressed high sampling rate spiral textures are comparable with the results of uncompressed pointwise faired surface light field. However, the former one requires 46MB (included view-independent part) in size and the latter one 180MB. This difference is mainly caused by the uniform sampling scheme we used. The spiral textures compressed using vector quantization (3.5MB + view-independent part) have comparable size with the surface light field compressed using function quantization (2.7MB + size of the geometry model). But the rendering result of our approach looks much better than that of the latter one, since a less aggressive compression scheme is used and only the view-dependent component is compressed. In addition, unlike surface light field, the

rendering cost for spiral texture is independent of scene complexity since it is defined in the image space.

Acknowledgement:

The authors acknowledge Wood et al. for sharing the photos and the geometric model of the fish on the web. Also, the authors would like to thank appropriate funding agencies.

References

[1] D. G. Aliaga & A. A. Lastra, "Architectural Walkthroughs Using Portal Textures," *IEEE Visualization*, IEEE, pp. 355-362, Phoenix, AZ, USA, October 19-24, 1997.

[2] C. Buehler, M. Bosse, L. McMillan, S. Gortler, & M. Cohen, "Unstructured Lumigraph Rendering," *Siggraph Annual Conference*, ACM, pp. 425-432, Los Angeles, August 12-17, 2001.

[3] E. Camahort, A. Lerios, & D. Fussell, "Uniformly Sampled Light Fields," *Eurographics Workshop on Rendering*, EG, pp. 117-130, Vienna, Austria, June 29-July 1, 1998.

[4] E. Catmull, "A Subdivision Algorithm for Computer Display of Curved Surface," Ph.D. Dissertation, University of Utah, 1974.

[5] D. Cohen & A. Shaked, "Photo-Realistic Imaging of Digital Terrains," *Eurographics Annual Conference*, EG, pp. 363-373, Barcelona, Spain, 1993.

[6] R. L. Cook, "Shade Trees," *Siggraph Annual Conference*, ACM, pp. 223-231, Minneapolis, MN, USA, July 23-27, 1984.

[7] K. J. Dana & S. K. Nayar, "Correlation Model for 3D Textures," *International Conference on Computer Vision*, Kerkyra, Corfu, Greece, September 20-25, 1999.

[8] P. E. Debevec, C. J. Taylor, & J. Malik, "Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-Based Approach," *Siggraph Annual Conference*, ACM, pp. 11-20, New Orleans, LA, USA, August 4-9, 1996.

[9] P. E. Debevec, Y. Yu, & G. Borshukov, "Efficient View-Dependent Image-Based Rendering with Projective Texture-Mapping," *Eurographics Workshop on Rendering*, EG, pp. 105-116, Vienna, Austria, June 29-July 1, 1998.

[10] S. J. Gortler, R. Grzeszczuk, R. Szeliski, & M. F. Cohen, "The Lumigraph," *Siggraph Annual Conference*, ACM, pp. 43-54, New Orleans, LA, USA, August 4-9, 1996.

[11] M. Levoy & P. Hanrahan, "Light Field Rendering," *Siggraph Annual Conference*, ACM, pp. 31-42, New Orleans, LA, USA, August 4-9, 1996.

[12] D. Lischinski & A. Rappoport, "Image-Based Rendering for Non-Diffuse Synthetic Scenes," *Eurographics Workshop on Rendering*, EG, pp. 301-314, Vienna, Austria, June 29-July 1, 1998.

[13] X. Liu, Y. Yu, & H.-Y. Shum, "Synthesizing Bidirectional Texture Functions for Real-World Surfaces," *Siggraph Annual Conference*, pp. 97-106, Los Angeles, CA, USA, August 12-17, 2001.

[14] J. R. Logie & J. W. Patterson, "Inverse Displacement Mapping in the General Case," *Computer Graphics Forum*, Vol. 14, No. 5, pp. 261-273, December, 1995.

[15] L. McMillan & G. Bishop, "Head-Tracked Stereoscopic Display Using Image Warping," *International Symposium on Electronic Imaging: Stereoscopic Displays and Virtual Reality Systems II Vol. 2409*, SPIE, pp. 21-30, San Jose, CA, USA, February 7-8, 1995.

[16] M. M. Oliveira & G. Bishop, "Image-Based Objects," *Symposium on Interactive 3D Graphics*, ACM, pp. 191-198, Atlanta, GA, USA, April 26-28, 1999.

[17] M. M. Oliveira, G. Bishop, & D. McAllister, "Relief Texture Mapping," *Siggraph Annual Conference*, ACM, pp. 359-368, New Orleans, LA, USA, July 23-28, 2000.

[18] M. M. Rafferty, D. G. Aliaga, V. Popescu, & A. A. Lastra, "Images for Accelerating Architectural Walkthroughs," *IEEE Computer Graphics and Applications*, Vol. 18, No. 6, pp. 38-45, November-December, 1998.

[19] G. Schaufler & M. Priglinger, "Efficient Displacement Mapping by Image Warping," *Eurographics Workshop on Rendering*, EG, pp. 175-186, Granada, Spain, June 21-23, 1999.

[20] H. Schirmacher, W. Heidrich, & H.-P. Seidel, "High-Quality Interactive Lumigraph Rendering Through Warping," *Graphics Interface*, pp. 87-94, Montréal, Québec, Canada, May 15-17, 2000.

[21] H. Schirmacher, M Li, & H.-P. Seidel, "On-the-Fly Processing of Generalized Lumigraphs,"

Eurographics Annual Conference, EG, Manchester, United Kingdom, September 4-7, 2001.

[22] J. Shade, S. J. Gortler, L.-W. He, & R. Szeliski, "Layered Depth Images," *Siggraph Annual Conference*, ACM, pp. 231-242, Orlando, FL, USA, July 19-24, 1998.

[23] D. Wood, D. Azuma, W. Aldinger, B. Curless, T. Duchamp, D. Salesin, & W. Stuetzle, "Surface Light Fields for 3D Photography," *Siggraph Annual Conference*, ACM, New Orleans, LA, USA, 2000.