DYNAMIC LIGHT TEXTURES

By

STEPHEN BROOKS

B. Sc., Brock University, 1997

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

Master of Science

in

THE FACULTY OF GRADUATE STUDIES

(Department of Computer Science)

We accept this thesis as conforming

to the required standard

The University of British Columbia

November 1999

Nov. 22, 1999
Date

Computer Science
The University of British Columbia
2366 Main mall
Vancovuer, BC
Canada V6T 1Z4

Abstract

A central motivation of computer graphics research is the generation of images, which approximate views of objects with sufficient detail as to render the distinction between the real and the artificial difficult or impossible. Much of the required visual complexity can be found on the surfaces of objects. Rich textures, such as animal fur or heavy cloths, introduce a level of detail which is immediately recognized as natural by the human visual system. It is, however, the detail of complex macro-structured surfaces which makes them difficult to model and expensive to render.

Presented in this thesis is a new method for the rendering of complex natural surfaces, which combines traditional 2D texturing and light field rendering. Here, real surfaces are approximated utilizing light emitted from complex surfaces, captured and tiled as surface light textures. As a form of texturing this method inherits issues associated with other types of texturing. But, with this new form of texturing we also see the introduction of research problems that are unique to the method. For example, low-incidence object silhouettes are given special consideration as the real texture edges are grafted at surface silhouettes to avoid unrealistic smooth polygonal or spline boundaries. Moreover, we partially overcome the static nature of light fields with a complementary technique which animates the parameter space of the light field rather than the original captured object.

Contents

List of Tables

List of Figures

Acknowledgements

The following people and organizations are acknowledged for their guidance and support during the development of the research presented here. Their contribution is greatly appreciated.

- Funding support was provided by the National Science and Engineering Research Council (NSERC) in the form of a PGS A scholarship, the BC Advanced Systems Institute (ASI), the University of British Columbia Department of Computer Science, and the Imager Computer Graphics Laboratory.
- Hardware and software provided by Dinesh Pai and the ACME group through an NSERC equipment grant, Larry Gritz for BMRT and the Stanford graphics lab for light field code.
- Technical support and feedback was provided by my supervisor, Alain Fournier, my second faculty reader, David Lowe, and assorted Imager lab members.
- A variety of assistance and counter-assistance was provided by friends, family, Jenny, Hunter and Sheba.

Stephen Brooks

The University of British Columbia

August 1999

1 Introduction

Computer graphics has become incorporated into widely disseminated applications for productivity and entertainment, many requiring not merely the flexibility which computer generated images offer, but also images which are accurate in their approximation of natural phenomena, with the additional provision of efficient rendering speeds. Arguably, the most compelling measure of success in computer graphics is found in the degree to which the computer production of images reduces the disparity between artificial and real imagery. For this goal to be approached, computer generated images require the same degree of convincing details which are characteristic of natural photography. Therefore, each object embedded within a scene must conform to human visual expectations. An object's surface features contribute significantly to the expected projection of appearance on the human eye and without such detail, objects can appear manufactured and unnatural. A real rabbit owes its appearance to the presence of fur on its body, and to the recognizable details on the fur and at the silhouette (Figure 1.1).

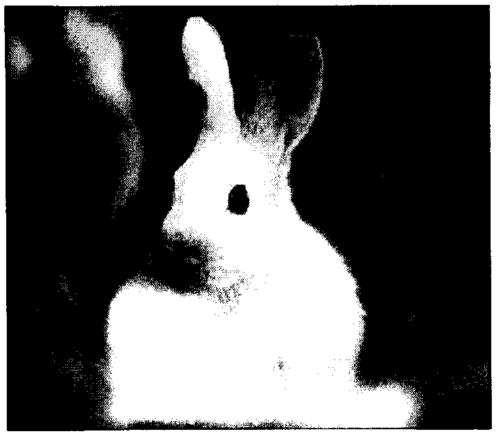
Figure 1.1: Real rabbit.

For another example, the geometry of a duckling without the expected downy surface details would appear as a plastic facsimile, as can be seen in Figure 1.2.

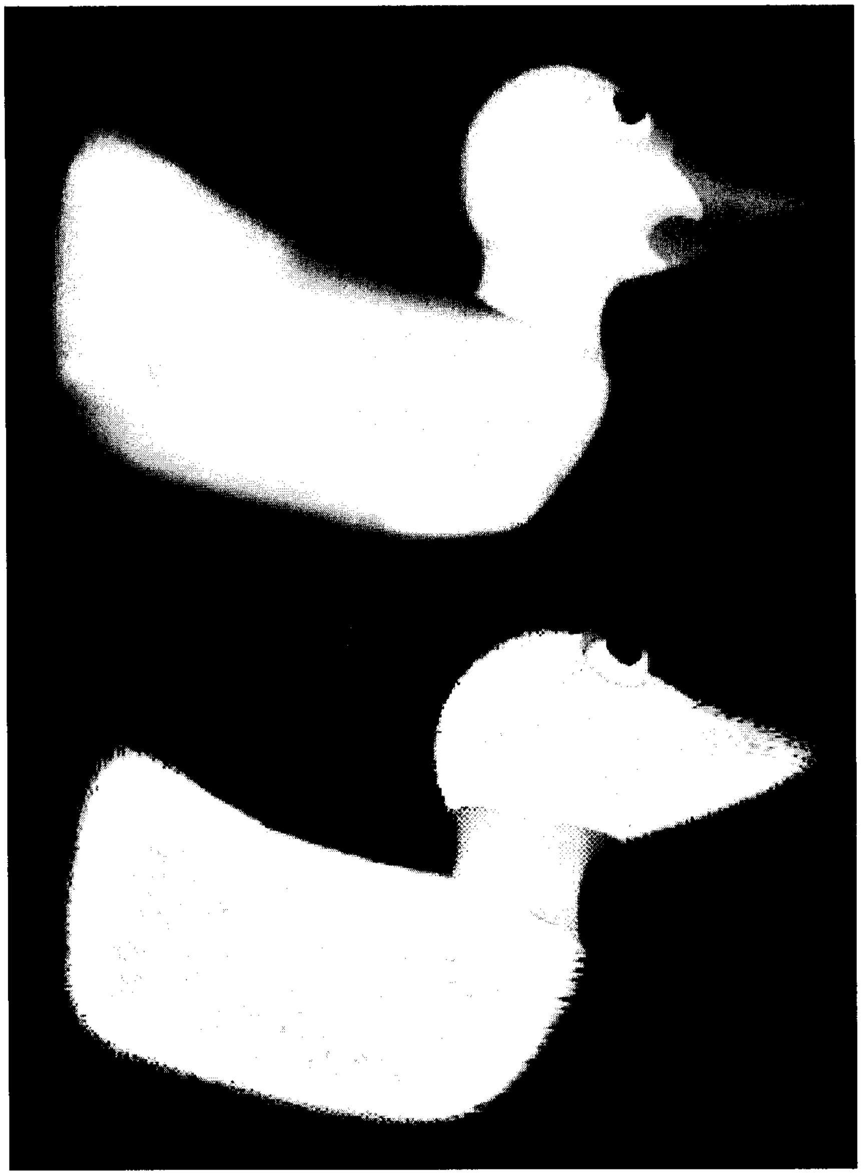

Figure 1.2: Untextured duck (Top). Furry duck (Bottom).

For correctly approximating natural phenomena one cannot strictly rely on hardware improvements to generate more objects at a faster rate. With brute force methods, it is not necessarily true that N+1 surface features is an improvement upon N surface features if those features each do not themselves capture the visual realism of the surface. A dominant theme of previous work that has sought to model natural surfaces has been to tile geometries or artificially produced volumetric data across surfaces. These methods tend to yield sparsely tiled surfaces that are convincing only at a distance or at low incidence angles, or else, require prohibitively high computation time. However, the main shortcoming of these solutions is found in the assumption that the geometric or volumetric tiles can be modeled artificially in the first place. Instead, we introduce the potential use of real physical data from complex macro-structured surfaces, animated and integrated with arbitrary geometric surfaces. In doing so, we may more closely match the appearance of natural textures in which higher sampling rates do, in fact, translate into greater accuracy.

Additionally, as the complexity of the scene grows it often implies a concurrent increase in the work of the animator or modeler, forcing application designers to customize "special effects" in the absence of a unified method for rendering surfaces. We wish to avoid the repetitive effort and expense, which is applied in tailoring solutions to generate each type of natural surface. In the research presented we describe a single technique that can be employed for simulating many disparate surfaces and simulate textures that have yet to be modeled with previous natural phenomena methods.

To this end, we utilize a recently developed technique that has enjoyed an increasing attention as a form of realistic rendering, namely, *Image Based rendering* (IBR). This approach substitutes the geometric datum as the atomic data item with that of the color sample. With sufficient quantities of such samples captured from a sufficient breadth of viewpoints, the object can be subsequently reconstructed from arbitrary viewpoints. However, this color data must remain static in time and gathered under a fixed lighting environment. This is due to the unfeasibility of constructing light fields under all possible lighting conditions, in all possible geometry. The loss of the dimension of time places sharp constraints on the applicability of IBR techniques in general, and *light field* rendering in particular. Typically, light field software permits the viewing of objects disjoint in time, situated in essentially a void.

We circumvent the time constraint, at least partially, by animating not the object itself, but the parameter space of the light field within which the object is captured. In doing so, we allow the re-introduction of time without the prohibitive cost. Furthermore, we tile the animated light field across the surface of a geometrically defined object, thereby creating a kind of *dynamic light texture*.

Chapter 2 provides background pertaining to traditional forms of texturing surfaces. This is followed in with an overview of IBR and light field rendering in particular as it stands prior to the publication of this thesis. All previous work is discussed and shown in relation to the present work. Chapter 3 begins the discussion of the current research, detailing our methods of animating light fields. Then the process of tiling a light across a geometric surface is described in Chapter 4, with results and future work presented in Chapters 5 and 6 respectively.

2 Previous Work

We begin with an overview of computer graphics techniques, which directly relate to the current work presented. We do this in order to provide the reader with an understanding of the fields as well as to delineate the boundary between Dynamic Light Textures and work that has preceded it. As this research can be viewed as a bringing together of surface texturing, Image Based Rendering (IBR) and the realistic simulation of natural phenomena, the number of related topics is necessarily quite broad. And so, here we will concentrate strictly on providing the necessary background for understanding texturing and Light Field Rendering.

2.1 Texturing

Natural surfaces are complex. Too illustrate this, Figure 2.1 shows a number of natural surfaces such as fur, heavy cloths, reptilian skin and corroded rocks. These types of surfaces are often too complex to model directly as geometric surface detail, both for reasons of rendering time and of modeling effort. This is partly because the final color registered by the human eye can be a result of multiple interactions of light with the surface geometry and local illumination characteristics. Some of the aspects, which effect appearance, include self-shadowing, a varying Bi-Directional Reflection Function (BRDF) and transparency, to name a few [Glas95].

Figure 2.1: Complex Natural Surfaces

The use of texture mapping, in its various forms, has the capacity to enrich the visual appearance of objects without requiring the considerable computations that would be necessary for a truly physically based simulation of the surface [Heck89]. Work on Texture mapping began with the early work of Blinn and Newel, altering surface color [Blin76]. Texturing has also been applied to a wider range of surface attributes such as: surface normal [Blin78], transparency [Gard85] and surface displacement [Cook84], among others. Figure 2.2 illustrates a number of common 2D texturing methods, implemented as Renderman shaders. Generally, texture mapping entails the mapping of a function f onto a 3D surface. We can group various flavors of texturing by the domain of the function, that is, whether the domain is two or three-dimensional.

2.1.1 Generalized Texturing of Surfaces

A 2D Texture Map is a method of controlling the color of a surface on a texel-by-texel basis, where a texel, which stands for TEXture Element, is the smallest addressable unit of a tiled image.

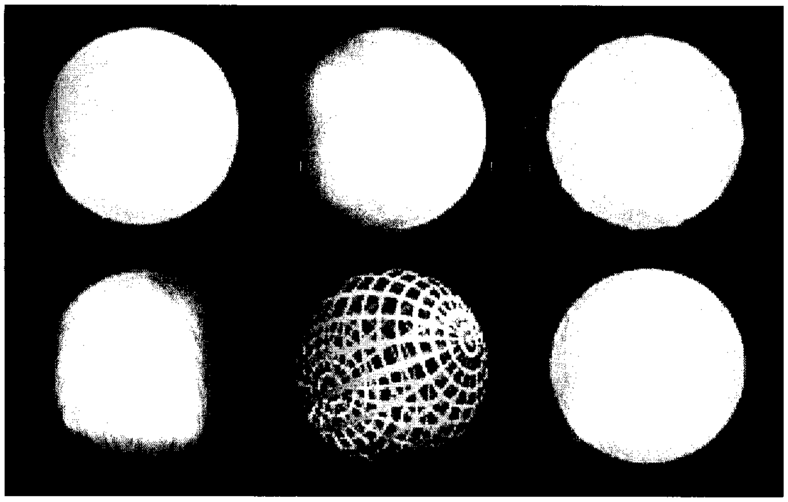

Figure 2.2: 2D Textures (clockwise from top-left): un-textured, procedural turbulence, displacement, bump, transparency and image map.

However, 2D surface patterns can also be generated by other means such as clonal mosaic models [Walt98], reaction diffusion [Mein95] or with the use of genetic algorithms [Sims90] for which the reader is referred to the given references. We will return our focus to yet another method, procedural texturing, when we look at the construction of artificial light fields [Eber94, Peac85, Perl89].

Despite the fact that 2D texturing can increase the overall visual interest, it is limited in a number of ways. First and foremost, the color at any point on the surface is not dependent on the viewing angle of the observer. As can be seen in the wood grain texture in Figure 2.2, there is a noticeable "billboard" aspect that cannot be overcome with higher resolutions or with more sophisticated filtering. This effect is heightened as the camera changes direction and draws near to the object. Moreover, the more complex and structured the texture is expected to be, the less convincing 2D texturing is. It is sufficient to consider tiling a single image of fur across a surface, displayed from a changing point of view, to understand this limitation.

The limitations of 2D texturing are often masked with the complementary use of bump mapping functions, which perturb the surface normal **N** at point P, based on the grayscale value of its input image or internal computation [Blin78]. Here, brightness is used as a gradient such that 0 corresponds to a surface depression, and 1 is an elevation. A good example of bump map use is for simulating the surface of an orange. At some low resolution an orange is a roughly a ball. The small surface pits, which emerge at higher resolutions, can be well captured with bump mapping, but at low incidence viewing, the silhouette exposes the limitations of this method: that the boundary between an object and the background is not effected by adjusting the surface normals.

Generally speaking, as the complexity of the surfaces increase, so do the techniques used to model them. That is true both in terms of implementation requirements and run-time. Fur has been modeled with various brute force methods, representing hair with large numbers of polygons or particles [Anjy92, Disc99, Flei95, Wata92]. Although these methods handle close-ups with well articulated strands, results can contain severe aliasing and/or computational costs which, in some algorithms, actually increase as the subject decreases in screen size. For example, in the work presented in Anjyo *et al.* in 1992, the scenes contained 50,000 or more individual hair strands with only moderately convincing appearance. Alternatively, volumetric techniques such as those found in [Kaji89, Meye98 and Neyr98] produce more visually convincing results under certain conditions. These techniques proceed to address the illumination and rendering of hairs by pre-computing a volume *texel* which is tiled across a surface. Hair geometry is rasterized into this texel and final rendering is accomplished using volume rendering. It is beyond the scope of this thesis to describe these methods in any detail, and so the interested reader is directed to the original papers. However, we will return to some of these methods in a comparison of results.

2.2 Image Based Rendering

In the past, most if not all of rendering could be described as a process of rasterizing a rectangular window of geometric models in a virtual space. With a strong analogy to the photographer's studio the geometric objects are placed relative to some arbitrary reference point, the lighting is set, camera is configured and placed and the "picture is taken". With real and artificial rays of light extending to and from the camera and the scene's surfaces, the color and intensities of the reflected and emitted light are then gathered within a rectangular film or image. Animation can be seen as a sequence of such renderings each with the objects, lighting or camera slightly altered. The extended analogy to the movie studio augments this parallel, but generally the structure of the process remains. Although it cannot be said that this computer-rendering paradigm has been replaced, there has been, of late, a competing rendering model that at the same time strengthens the cinematic analogy in some respects, yet distorts and limits it in others, namely image-based rendering.

Image-based rendering might be subdivided further into View Interpolation methods [Lipp80, Seit96] and Light Field methods [Levo96, Gort96]. Generally speaking, view interpolation involves epipolar analysis, and the establishing of correspondences between images or range images [McMi95, Chen95]. Inwardly, multiple views on surface of sphere surrounding object are taken; outwardly multiple panoramic views at spaced intervals are gathered. Objects are then re-rendered from new viewing locations often using depths to resolve overlaps.

Recently, the use of real images has been employed in the construction of panoramic environments with the introduction of the QuickTime VR software [Chen95, Chen93]. And, similar work in IBR introduced the concept of view-dependent morphed textures with the Facade Software used in the reconstruction of Architecture from photographs [Debe96]. Advantages of using view interpolation include: modest computation that is independent of scene complexity, no expenditure of modeling effort and the capacity to capture hard to model objects and surfaces. However, these methods are limited in that they require depth ranges or correspondences between images which would be quite difficult to obtain from surfaces such as fur.

2.2.1 Light field Rendering

As the IBR technique presented in this paper can be seen as a bridge between traditional 2D texturing and light field rendering, we now proceed with an introduction to light fields. But before delving into the particulars of the constructed light field let us quickly review the light field's description. Light fields have been described by one of their originators as:

> "A new image-based rendering technique that does not require depth information or image correspondences, yet allows full freedom in the range of possible views. The idea behind the technique is a representation of the light field - radiance as a function of position and direction in regions of space free of occluders (free space)...Creating a light field from a set of images corresponds to inserting 2D slices into the 4D light field representation...Once a light field has been created, new views may be constructed in real time by extracting 2D slices in appropriate directions." [Levo96]

Light fields can be understood as a discrete 4D slice of a the 7D plenoptic function [McMi95]:

$$P(t, \lambda, x, y, z, \theta, \phi)$$

where t is time, λ is wavelength, x, y and z are coordinate positions in 3D, and where θ and ϕ denote the viewing direction. This idealized function will produce the correct quantity for each wavelength viewed from a given position and angle at a particular point in time. The reduction of dimensions is possible if we compute a RGBA color value for any position and angle at $t = t_n$, yielding a 5D light field:

$$L(x, y, z, \theta, \phi)$$

This can be further reduced provided the object of interest is located in free space wherein the value of any given light ray is constant over its length. This produces the much reduced 4D light field:

$$L(x, y, \theta, \phi)$$

where (x, y) denotes the planar position, and (θ, ϕ) defines the direction. But, we notice that the flexibility of the light field decreases as dimensions are removed, capturing only a discrete sampling of light of static objects in unconcluded space.

The main issues that need to be considered are to a large extent related to the considerable size of light field objects, as light fields often require hundreds to thousands of images. We can succinctly approach these issues by briefly answering the following questions:

Parameterization	How are the light samples structured?
Construction	How are the samples acquired?
Display	How are new viewpoints constructed?
Compression	How can we exploit coherence in 4D?

Table 1: Light Field Issues

Because light fields can be as large as 100MB uncompressed, it is essential that the parameterization of the 4D data permits computationally efficient accessing. And, because we wish to cover views of an object completely and evenly, uniform sampling of position and viewing angle is also desirable. A simple parameterization that allows rapid access and good coverage is the parallel positioning of two grid planes which we will call

UV and *ST*. This two plane addressing of light vectors is accessed by providing two coordinate pairs (*u*, *v*) and (*s*, *t*) discrete on [0..1]. Together, the *UV* and *ST* planes comprise a single *light slab*. Figure 2.3 shows a simple diagram of a possible light slab.

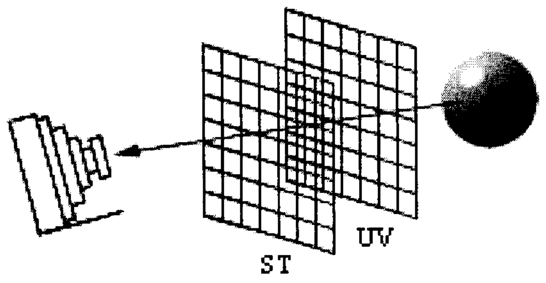

Figure 2.3: A Light Slab

For our purposes we will consistently name the UV plane as that closest to the captured object and ST as that furthest away. A light field is a simply a set of such slabs. Typically, for full coverage of an arbitrary object, the light field will consist of 6 slabs, each parallel to the axial planes, forming a cube around the object.

As a function over 4 variables which produces radiance values in an double grid structure, the light field can be constructed as a set of input images. Each input image is essentially a 2D slice of the larger 4D structure. One can imagine a camera which is placed at each point on the ST plane, with the produced image as a captured image of the UV plane at one point in time, as shown in Figure 2.4.

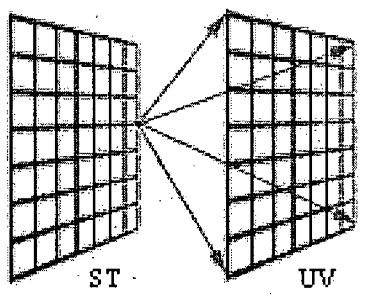

Figure 2.4: 2D Light Field Slice

These images can be generated artificially with a renderer, or with a real camera. Although the light field requires many images the process is a good candidate for automation, as in [Levo96].

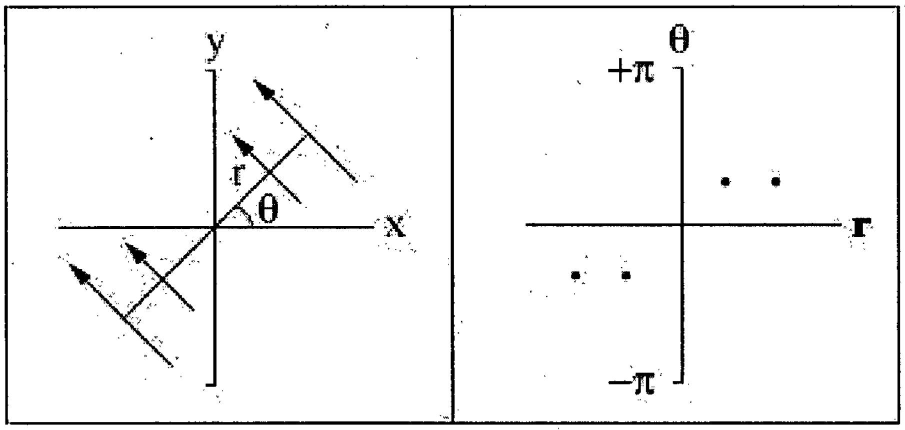

Figure 2.5: Plenoptic coverage in point space (left) and line space (right).

Generating new views of an object corresponds to extracting and re-sampling a 2D slice of the light field. For a light field *L* and an image *I*, each ray passing through the center of projection of the virtual camera and a pixel $I(x, y)$ is intersected with UV at (u, v) and with ST at (s, t), as in Figure 2.6.

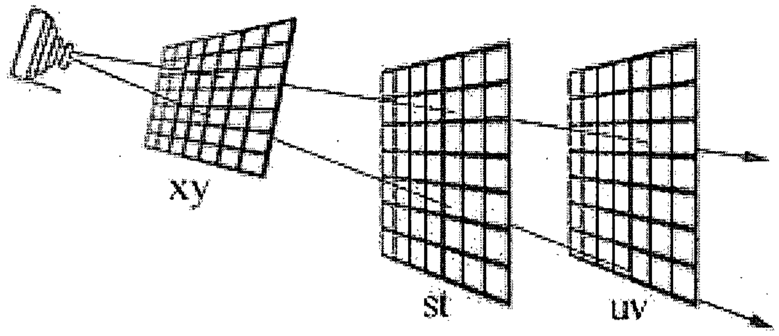

Figure 2.6: Re-sampled view of light slab.

Once the u, v, s, t positions are known the RGB color value $L(u, v, s, t)$ is directly accessed:

$$Image(x, y) = L(u, v, s, t)$$

Only those rays with (u, v) and (s, t) coordinates inside both quadrilaterals are represented in the light slab. It is unlikely that the intersection point of the UV and ST planes are exactly at grid points, which means that the closest grid points must be sampled and interpolated. Common interpolation methods are bilinear over ST or UV, or quadrilinear over both ST and UV. This has been improved upon in the work done by [Gort96] which uses an approximate geometry of the captured object to improve the quality of the reconstruction at lower sampling densities. Additionally, we will merely note that due to the size of light fields a high compression rate is required. Compression rates can be achieved at ratios of (100:1) due to coherence in 4 dimensions, with fast decoding and offline encoding.

Recent advances in Light Field Rendering have augmented the original light field concept in some fashion. Multiple light fields have been successfully merged [Chiu98], layered [Kang99], augmented with geometric scenes [Mene99], and also used as canned light sources [Heid98]. Related to the present work, [Mill98] has used light fields as a storage construction for surfaces in pre-computed global illumination. Current research is also being applied to the re-lighting of light fields [Mene99]. However, fully animated light fields and the integration of light fields with geometric objects remains a currently studied problem.

3 Dynamic Light Fields

3.1 Constructing Light Fields

In order to show the potential of dynamic light fields, and conversely, to delimit the boundaries of its usefulness, it is necessary to construct light fields that convey the full extent to which a light field can be convincingly animated. We will focus on a particularly difficult and computationally expensive macro-structured surface: animal fur.

3.1.1 Constructing Artificial Light Fields

In its construction of light fields, the Stanford light field authoring package makes use of the RenderMan interface for sampling 2D image slices. In particular Larry Gritz's Blue Moon Rendering Tool was used as the primary rendering engine, "which are a set of rendering programs and libraries which adhere to the RenderMan™ standard as set forth by Pixar" [Grit96]. At the core of the RenderMan interface is the procedural texture shader language. The shading language allows the programmer to procedurally direct all aspects of surface shading calculations executed by the RenderMan renderer. These include surface shading, light source description, atmospheric effects, and surface displacement. Fortunately, the phenomena which procedural textures readily generate tend to correlate with those that are best animated with dynamic light fields, such as fur.

Like traditional 2D texturing, procedural textures use the *UV* surface coordinates as input but do not simply index color values in a stored image array. It is common for procedural textures, such as turbulence, to be based on fields of isotropic psuedo-random noise with an appropriate value range and distribution [Eber94]. The random noise fields are fixed grids of random numbers of N dimensions, interpolated linearly or, more frequently, with a polynomial fitting, such as with Catmull-Rom splines, see Figure 3.1 below.

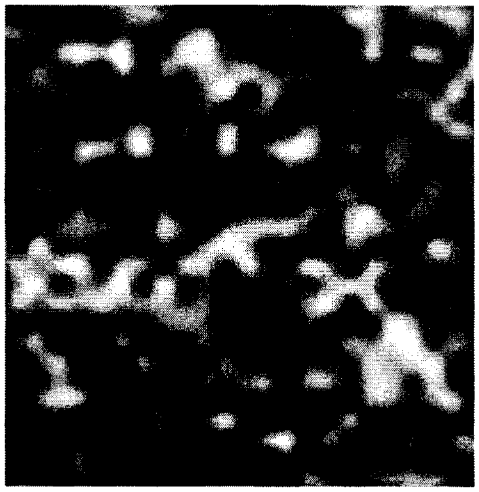

Figure 3.1: Catmull-Rom Spline interpolated noise field.

To move from noise to turbulence, self-similarity is imposed by summing the noise at the current point with scaled down noise values at other points. To compute a sample *turb*(x) from the turbulence function, noise is combined with amplitude varying as a function of frequency, building up a stochastic spectral function:

$$turb(x) = \sum_{f=1}^{k} abs(noise(2^f x) / 2^f)$$

Where k is the smallest integer for which $\frac{1}{2^{k+1}} > width(pixel)$, and the *noise*() function is an interpolating access function to values stored in the random noise array. A 2D slice of a 3D turbulence function is shown in Figure 3.2.

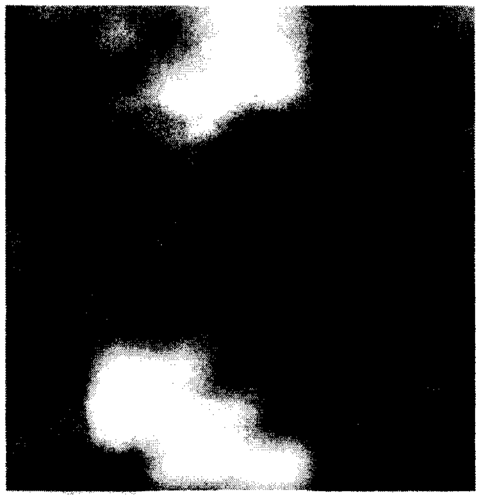

Figure 3.2: Self-similar turbulence function.

Although procedural textures tend to require greater computation, each value is computed as needed at an arbitrary level of detail. Additionally, the procedural textures are designed to be band-limited by low-pass filtering the random noise, thereby avoiding the necessity for filtering out aliasing due to high frequencies. And while this type of texture has a very compact representation when compared with image, volumetric or light textures, they can be difficult to construct in a predictable way.

In the present work, various pseudo-stochastic procedural noise-based turbulence functions have been used both in the construction of the light fields as well as to drive the movement of the light field's *ST* plane, as will be described in Section 3.2.1. To illustrate, shown in Figure 3.3 is a fur patch constructed as a randomized array of cones. The cones are arranged according a regular grid which has been subjected to a bivariate Gaussian jitter displacement.

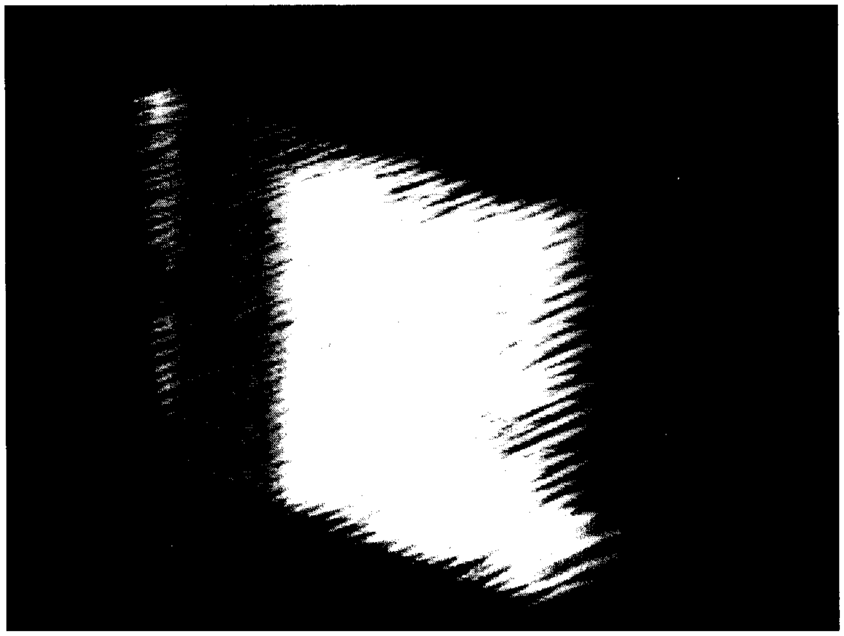

Figure 3.3: Bird's eye view of procedural fur texture.

To compute the coloration and opacity of the cones, we begin with a white noise function with a low pass filter to safely remain within the Nyquist Frequency when rendered under normal conditions. This is a lattice noise, with each point in the lattice being indexed as (x, y, z). The noise function calculates an interpolated value between each random value in the lattice where the interpolation is computed with a Catmull-Rom cubic spline function. Each cone exhibits a surface texture local to itself. The texture is a flame-like turbulence, as described in [Eber94] on pages 279-280, which indexes a dark-brown (at the base) to light-brown (at the tip) color array, which is itself interpolated with a spline. Additionally, the cones in the fur patch are subject to a similar, procedural opacity function. We conclude our discussion of generating artificial fur with an array of images that comprise a subset of the fur light field rendered with the described cone array. At the top-left is *ST* image (0.6875,0), increasing to (0.6875, 0.25) moving right and (0.8125,0) moving down. At bottom-right is *ST* image (0.8125, 0.25).

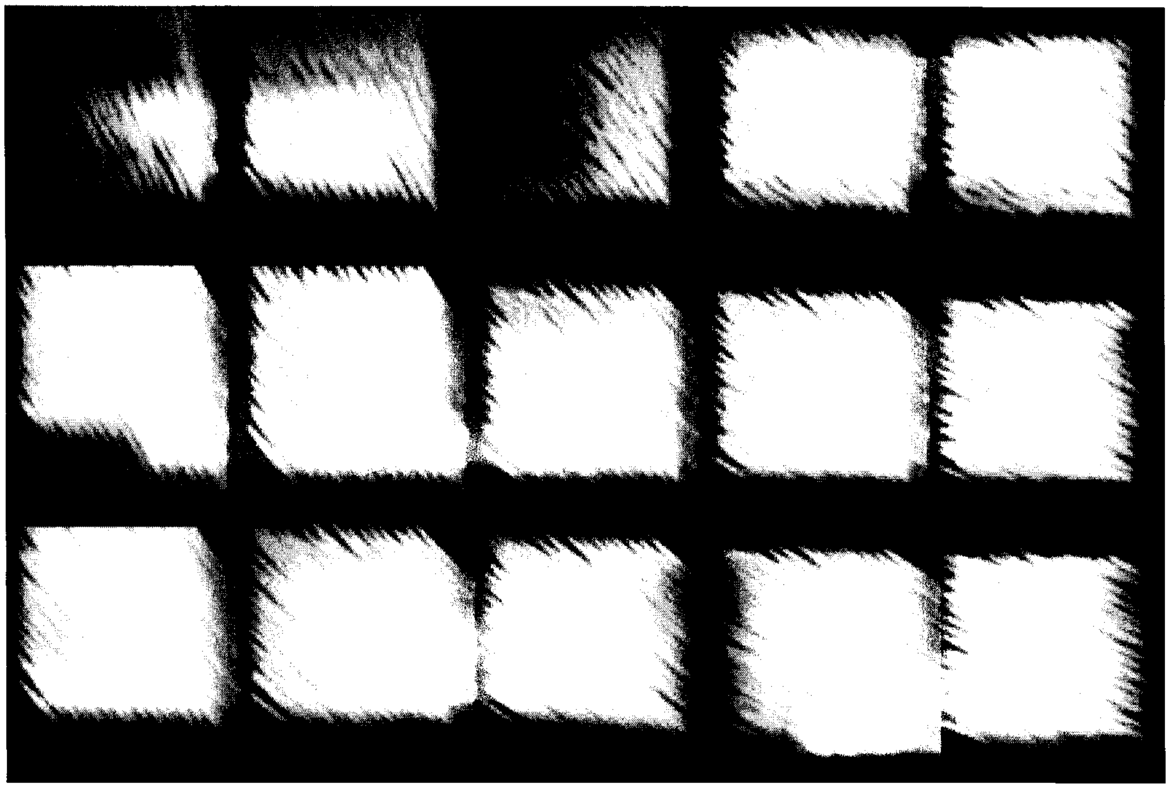

Figure 3.4: Each *ST* image is a 2D slice of the 4D light field.

3.2 Dynamic Light Fields

Producing animated light fields with a brute force technique would be expensive to the point of unfeasibility. It would require capturing a light field at many points in time for each articulation of an object. Apart from the fact that this may entail a multi-generational project for significant animations, for animating fur it would be extremely difficult to capture an entire light slab as it is moving. To circumvent this seeming limitation of light fields, we do not animate the original object; we instead animate the parameter space, thereby animating the light.

3.2.1 Animating the Parameter Space

Work presented in "Light Field Rendering" [Levo96] and "The Lumigraph" [Gort96] each defines a finite double-grid data-structure for the storage and re-construction of sample color data surrounding a given object. Because we wish to animate the captured object by dynamically changing the parameter space of the light field, it is therefore the grid, either *UV* or *ST*, which is subjected to animated displacement. Now, suppose for a moment that we take a flat patch of fur and capture the light data emanating from that fur in such a way as to place the *UV* plane on or about its surface. Then, by shifting the grid points on the *ST* plane in waves approximating a wind motion[1], the resulting animation yields a movement that is approximately equivalent to an animation in which the original object is altered. Figure 3.5 depicts a 2D orthogonal projection of this scenario with the arrows denoting the displacement of grid points in the *ST* plane. Once the (s, t) position of a particular pixel (x, y) in image I is known, the computed S and T displacements

[1] An animation technique described in [Eber94] on page 187 which is claimed to be a wind-like motion.

dispS(s, t, *t'*), *dispT*(s, t, *t'*) at time *t'* is directly added and the light field, L, is sampled at the new position:

$$I(x, v, t') = L(u, v, s + dispS(s, t, t'), t + dispT(s, t, t'))$$

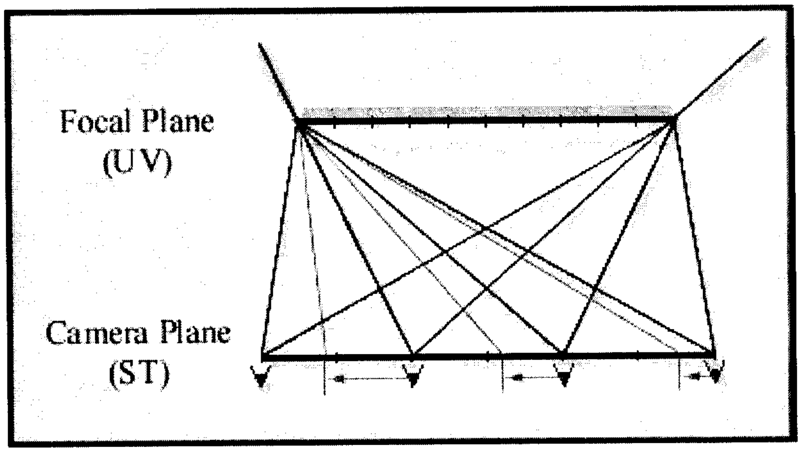

Figure 3.5: Displacing *ST* coordinates.

For the purpose of illustration a turbulent warping function and a procedural turbulence which approximates the animation of gases are used for generating displacements. The warping function is an application of the 3D lattice noise discussed earlier, with values ranging from -0.5 to 0.5. In this case the 3rd dimension is used as time. As the time frame moves through the lattice, the scaled and interpolated value in the lattice is indexed by the s and t coordinates and subsequently used as *dispS*(s, t, *t'*) and *dispT*(s, t, *t'*). Only a subset of the function's range is used at any time, with separate samplings of the functions used for each of the *S* and *T* directions.

The second function simulates gas movement in a solid space. This procedure incorporates a horizontal helical path used which produces an upward swirling effect. The interested reader is referred to [Eber94] for further details on solid texturing of gases. Time slices of the functions are shown in Figure 3.6 with the values of x $\in$ [-0.5..0.5] scaled to the range of [-255..255] and with negative values shown as (255-x). Visually, in the figure, dark areas are negative displacements with light areas being positive.

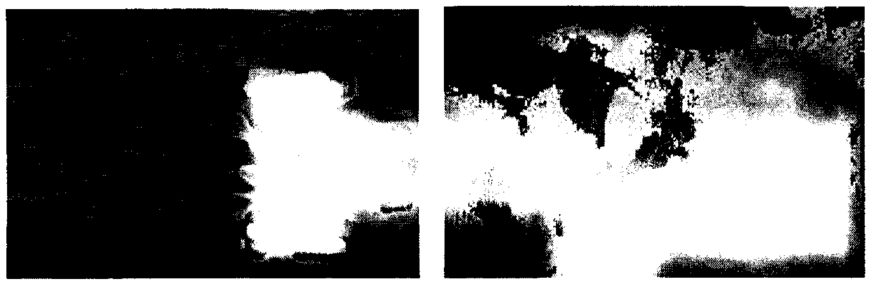

Figure 3.6: Time slice of the Animated Warp (left) and Wind (right).

Figure 3.7 depicts the effects of the animated warp on the previously discussed fur patch in successive frames, at a high amplitude:

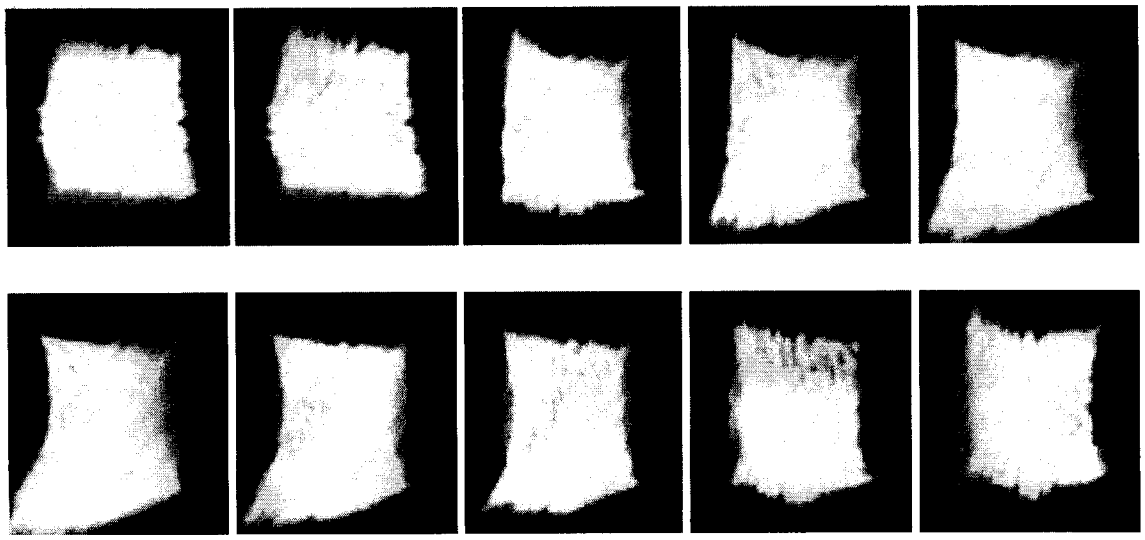

Figure 3.7: Animated stills of Artificial Fur.

3.2.2 Animating Articulated Objects

We might ask to what extent the animation of the *ST* grid is applicable to articulated objects and if the movement of the grid can be interactively directed. These questions have been explored with the following animation techniques which alter the *ST* plane indexing, acting as translation functions for each *ST* point. Control of the each translation function is affected in various ways by user manipulation of Bezier splines and with the movement of displacement points. In discussing each of the techniques, we will assume a *UV* space between 0.0 and 1.0 in each dimension. Figure 3.8 shows a dragon light field alongside a grid ranging in both *S* and *T* directions from 0.0 to 1.0 [Levo96]. The dotted square in the center of the grid denotes the current extent of the *ST* space that is being utilized for display.

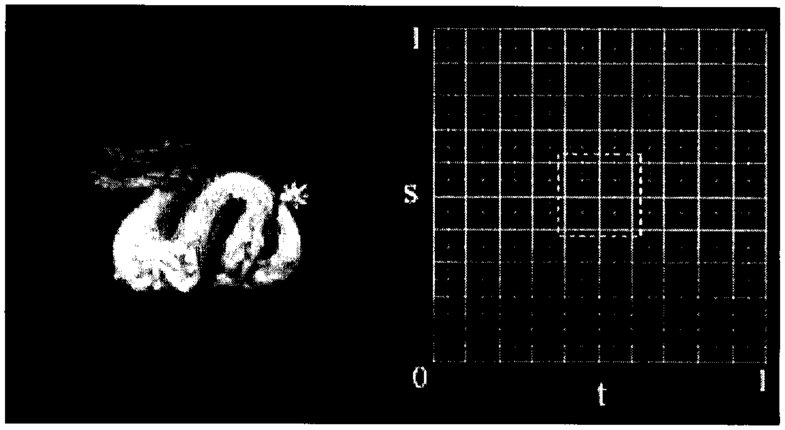

Figure 3.8: Dragon shown with *ST* plane coordinates.

The first *ST* displacement method which we will discuss, spline distortion, is to a certain extent analogous to the fish eye transform in 2D imaging. The further the current *s* and *t* point is away from the center *C* positioned at C_s=0.5, C_t=0.5 in *ST*, the further it is translated in the same direction. This effect is mitigated by the user-controlled Bezier curve as seen in Figure 3.9 which allow the user to control the amplitude of the displacement at positive and negative distances from *C*. The Bézier curve is a vector-based function of two variables in the form:

$$\boldsymbol{M}(d,t') = [X(d), Y(d)]$$

with $\boldsymbol{M}$ evaluated at time t', where d is the current point under evaluation, so that for each d, $\boldsymbol{M}(d, t')$ calculates a point on the curve. To define $\boldsymbol{M}(d, t')$ we begin with a Bernstein polynomial of degree n:

$$B_i^n(d) = \binom{d}{i}\ d^i(1-d)^{n-i}$$

Now if $P(t')$ represents a set of control points whose state is set at time t' then:

$$M(d,t') = \sum_{i=0}^{n} B_i^n(d)\, P_i(t')$$

represents a Bezier curve as d varies from 0 to 1. The spline curve level at the s and t positions is multiplied with the signed distances from C:

$$dispS(s, t') = M(s,t') * (s - C_s),$$

$$dispT(t, t') = M(t,t') * (t - C_t)$$

This allows the user to set the broadness of the displacement, or, for example, make the displacement one-sided.

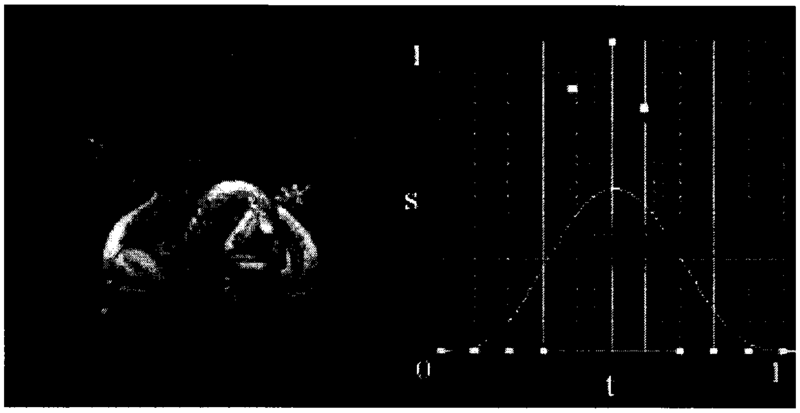

Figure 3.9: Spline Distortion of the *ST* plane

The related "Slider displacement" is more direct than the previously mentioned user control. In this interface, shown in Figure 3.10, there are two separate user-controlled curves: one affecting the *S* direction, M_s, and the other in the *T* direction, M_t.

The scaled curve's value is directly added as an offset to each pixels s and t positions, so that $dispS(s, t, t') = M_s(s, t')$ and $dispT(s, t, t') = M_t(t, t')$.

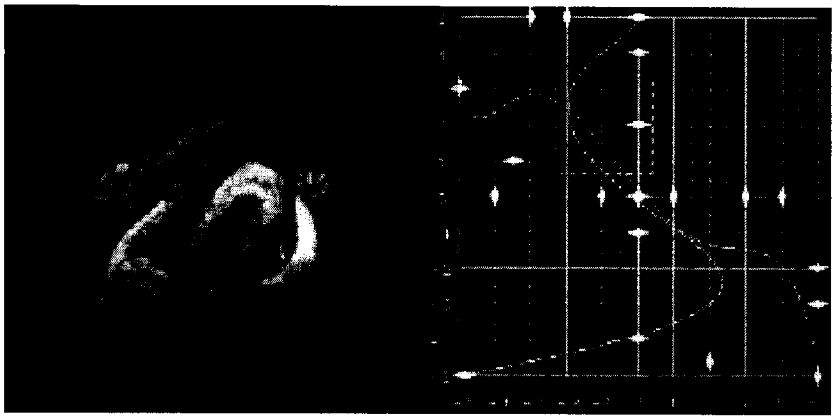

Figure 3.10: Slider Displacement of the *ST* plane

Another approach which has been developed, Repulse Point distortion, allows the user to directly alter the *ST* space. The name, Repulse Point, is given to imply that the *ST* space is pushed away from the each control point in the set R in a particular direction. A Repulse Point $r \in R$ with position (r_s, r_t) acts with a circular radius; the closer the (s, t) point is to r, the stronger is the effect. Each r displaces the s or the t position in the positive or negative direction, as indicated to the user from its color. The sum total of all displacement amounts from each r provides the displacement for s and t :

$$dispS(s, t, t') = \sum_{r \in R_s} \left(\frac{k}{\left(\sqrt{(s - r_s)^2 + (t - r_t)^2} \right)} * \frac{(s - r_s)}{abs(s - r_s)} \right)$$

$$dispT(s, t, t') = \sum_{r \in R_t} \left(\frac{k}{\left(\sqrt{(s - r_s)^2 + (t - r_t)^2} \right)} * \frac{(t - r_t)}{abs(t - r_t)} \right)$$

scaled by some constant *k*. Shown in Figure 3.11, are a number of control points acting on the Dragon. The color of the point indicates two things to the user: its brightness conveys the power of the warp and its color the direction. In combination the object can be rotated differentially. In the figure, we see that only the head of the Dragon is twisted.

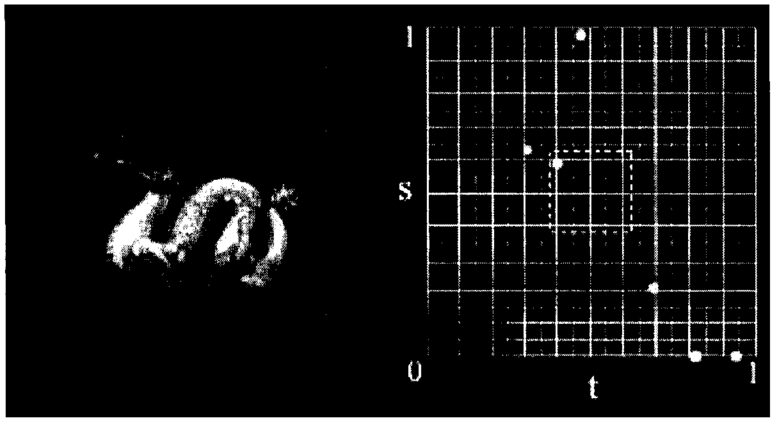

Figure 3.11: Repulse Point Distortion of the *ST* plane.

Further examples are given in Figure 3.12, in which the top row shows successive frames of the Dragon's plume becoming larger. The bottom row shows the twisting of the Dragon's head in sequence.

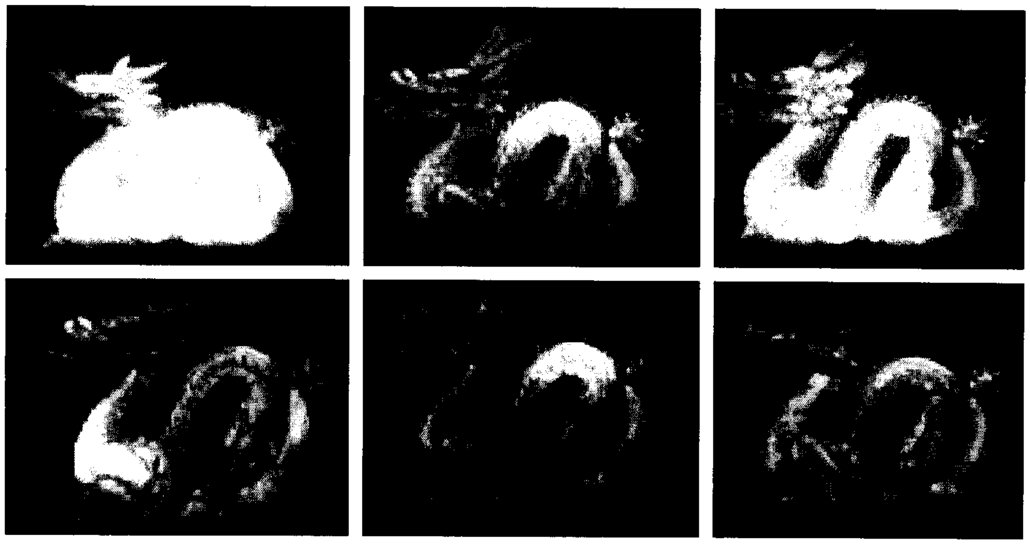

Figure 3.12: Dragon's plume becoming larger (top), twisting of the Dragon's head (bottom) in sequence.

The interactive animation of *ST* has been explored with partial success for articulated objects. Objects can be stretched and warped in a cartoon fashion, but for animating selective portions of an object it must be true that the selected portion is visually separate from the rest of the object body. Put another way, the portion of the object that is to be animated must not occlude other parts of the object. However, these techniques can also be applied, without caveats, to directly manipulate the light field textures which follow.

4 Light Textures

Upon further consideration of the animated light field of fur, it might be claimed that it is an interesting artifact in itself to have a dynamic light field mimicking such a complex and difficult to simulate surface. However, in an effort to increase the utility of dynamic light fields we have developed a method for texturing the light field itself across a surface, such as a NURB. This light texture, once tiled, could then be subjected to the same methods of animation but this time globally over the entire surface.

4.1 Tiling Light Fields

A textured surface is parameterized with the 2D coordinates *u* and *v*. For parametrically defined objects, the parameterization is produced as a natural byproduct of the surface construction. For other surfaces, such as polygons or implicitly defined objects such as quadratics, the *UV* parameterization must be explicitly determined. For example, for planar polygons, each triangle is parameterized with an affine transformation between each of its vertices *i* with coordinates (*x, y, z*) and (*u, v*):

$$[x, y, z] = [u, v, 1]\begin{bmatrix} A \\ B \\ C \end{bmatrix}$$

The surface coordinates (*u, v*) are used to index each color when tiling a raster image across a surface.

If we define 2D texturing more broadly, as a set of functions F with each texture function $f^i \in F$, altering surface attribute i, we can allow for greater flexibility of surface appearance. By way of introduction to tiling light textures we now describe the familiar bump mapping technique, which has some similarities to light texturing. A bump mapping function, $Bmap \in F$, perturbs the surface normal *N* at point *P*, based on the grayscale value of its input image or internal computation [Blin78]. In the case of image input, brightness is used as a gradient such that 0 corresponds to a surface depression, and 1 corresponds to an elevation. The perturbation must be independent of the surface orientation and position to avoid an unwanted animation of the surface as the object moves. Therefore, the normal is perturbed with a function which relies on surface derivatives $\boldsymbol{u}' = \frac{dP}{du}$ and $\boldsymbol{v}' = \frac{dP}{dv}$ at point *P* on the surface, such that $N(P) = \boldsymbol{u}' \cdot \boldsymbol{v}'$. To gain independence from the surface's orientation and position we define two new vectors ***q*** and ***r*** which are coplanar and together with *N* form a new coordinate system such that:

$$\boldsymbol{q} = \boldsymbol{N} \cdot \boldsymbol{v}',$$
$$\boldsymbol{r} = \boldsymbol{N} \cdot \boldsymbol{u}',$$
$$\boldsymbol{N}^d = \boldsymbol{B}^u \boldsymbol{q} - \boldsymbol{B}^v \boldsymbol{r}$$

where $\boldsymbol{B}^u$ and $\boldsymbol{B}^v$ are partial derivatives of the Bump Map *B* at (*u, v*). And so, the new normal $\boldsymbol{N}'$ is the sum of **N** and the new displacement $\boldsymbol{N}^d$:

$$\boldsymbol{N}' = \boldsymbol{N} + \boldsymbol{N}^d$$

Light texturing tiles the light vectors of a light field across a surface. Here we also make use of surface derivatives $\boldsymbol{u}' = \frac{dP}{du}$, $\boldsymbol{v}' = \frac{dP}{dv}$ and normal *N* of the underlying surface that we are texturing across. It is also required that both the *UV* ant *ST* positions for each viewing ray are known. But, in order to be clear, it is important to reiterate at this point the naming convention we are using for the light field on the one hand, and of the underlying surface on which we are texturing on the other. Recall that we have adopted the convention that the *ST* grid is the camera plane that is furthest from the fur texture that we have captured as a light field. The *UV* grid is then the focal plane, located at the texture itself. The surface on which we are to tile the light field also is defined with *UV* parameters. It is an artifact of the standard naming of light fields and surfaces that *UV* is employed for both. However, this is in a sense convenient as the *UV* space of the underlying NURB, or polygonal, object is to be integrated with the *UV* and *ST* parameters of the tiled light field. The situation is shown in Figure 4.1. The tiling method is as follows:

1. Equate the scaled *UV* space of surface with *UV* space of light field.
2. Transform the viewing direction from surface coordinates to light field coordinates.
3. Intersect *ST* with transformed viewing vector from the point on the *UV* plane.

Formally, the *UV* space of the object, *O*, with points at (O_u, O_v), and the scaled *UV* space of the light field, *L*, with points at (L_u, L_v), are to be equated. What remains is the determination of the *ST*, at (L_s, L_t), the coordinates that specify the particular light ray we are looking for in the light field. This *ST* position is gained by taking the viewing vector from the eye, $w = (w_x, w_y, w_z)$, to the object's surface at point $P = (P_x, P_y, P_z)$, and transforming it from the coordinate space of the NURB to the light field's coordinate space. The new vector, $\boldsymbol{w}' = (w'_x, w'_y, w'_z)$, is calculated as:

$$[w'_x, w'_y, w'_z] = [w_x, w_y, w_z]\,[M]$$

where,

$$M^{-1} = \begin{bmatrix} u'_x & u'_y & u'_z \\ v'_x & v'_y & v'_z \\ N_x & N_y & N_z \end{bmatrix}$$

and where $\boldsymbol{u}' = (u'_x, u'_y, u'_z) = \frac{dP}{du}$ and $\boldsymbol{v}' = (v'_x, v'_y, v'_z) = \frac{dP}{dv}$ are the partial derivative of the surface at point *P* and ***N*** is the normal to the surface at *P*. Then, by intersecting the transformed viewing vector, $\boldsymbol{w}' = (w'_x, w'_y, w'_z)$, with the light field's *ST* plane, at (L_s, L_t), plane from (L_u, L_v), on the *UV* plane, we acquire both the necessary *UV* and *ST* coordinates. To illustrate, Figure 4.2 shows a NURB with the *UV* space tiled 3 x 3; the left image shows the resulting 3 x 3 light texture of a fur patch.

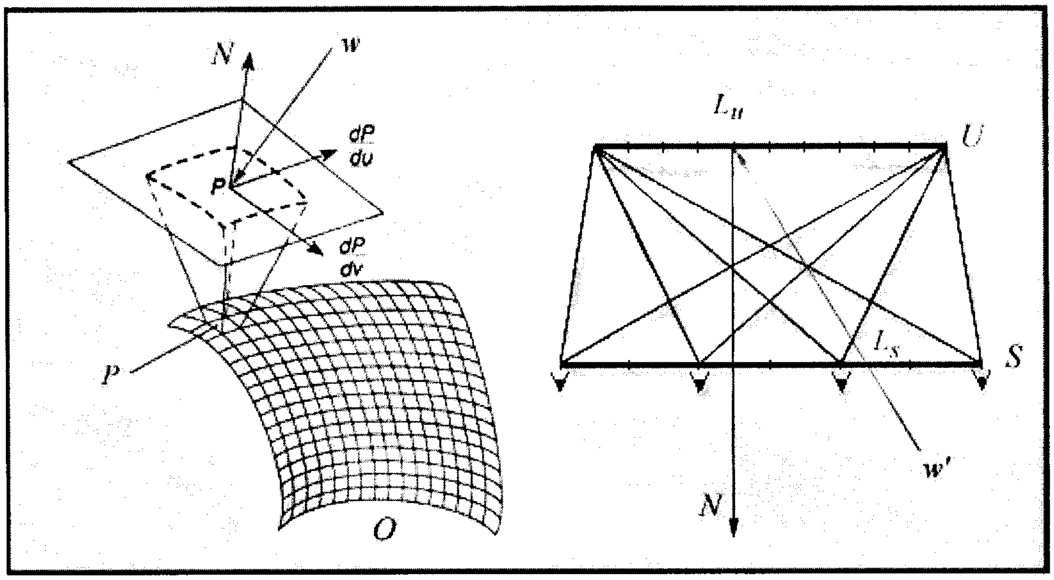

Figure 4.1: Transforming from surface coordinates (left) to Light Field coordinates (right).

The calculations for this aspect of the technique are simple and, therefore, fast. Once the light texture is tiled, we can then animate the global *ST* space of the tiles surface with the same animated displacement techniques discussed previously. With this new tiling method there are, however, a number of issues that must be addressed.

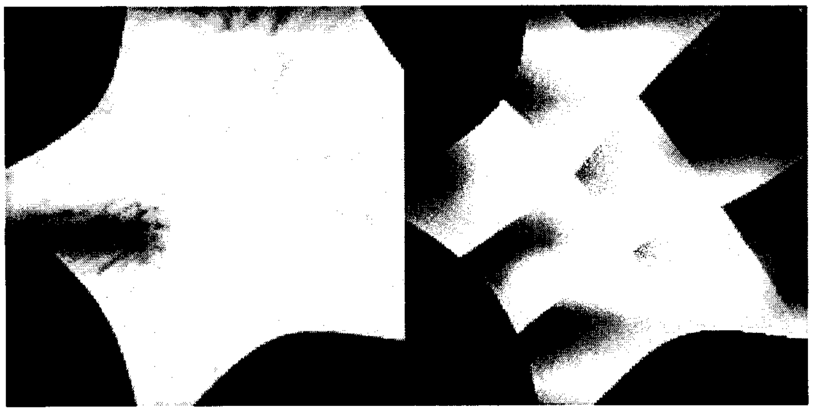

Figure 4.2: Artificial Fur Light Texture (3x3).

4.2 Filtering Light Textures

Before filtering the Light Texture, we need to crop *UV* within the real texture area, thereby avoiding the real edges of the original texture. This cropping can be as simple as setting the bounds within the texture in the first instance when capturing the light data. Using the cropped light texture the near-edge samples are filtered with complementary samples on other side. The key to edge filtering is to strike a balance between the removal of discontinuities and excessive blurring. In the case of light texturing, filtering can be in all *U, V, S* and *T* directions, or some subset thereof. For the constructed examples presented here, the quadrilinear filtering in the re-sampling of the light field itself was sufficient to mitigate discontinuities, however, further research could be applied to this issue.

4.3 4D Multi-Resolution Analysis/Synthesis

An alternate solution, which has been partially explored, extends the work of [DeBo97] From 2D to 4D. The work of [DeBo97] describes a two phase process for generating new textures from input images. In synthesizing the 2D texture, the input image is first analyzed, identifying the joint occurrence of texture features at multiple resolutions. This is accomplished by constructing Laplacian Pyramids, one for each of the Gaussian, and it's first and second derivatives. The filters can be seen in Figure 4.3, with sample Gaussian pyramids in Figure 4.4. To generate the original Gaussian filter the following formula has been applied:

$$G(x_1,x_2,\mu_1,\mu_2,\sigma_1,\sigma_2)=\frac{1}{(2\pi\sigma_1\sigma_2)}e^{-\left(\left(\frac{x_1-\mu_1}{\sigma_1}\right)-{}^{2}\left(\frac{x_2-\mu_1}{\sigma_2}\right)^{2}\right)^{\frac{1}{2}}}$$

where (x_1,x_2) is the current pixel position, (μ_1,μ_2) denotes the filter center and (σ_1,σ_2) denotes the standard deviation in each of the two directions.

In the synthesis phase, a new texture is generated by sampling successive spatial frequency bands from the input texture based on similar features at lower spatial frequencies. To re-arrange the texture, each level in the pyramid structure is re-sampled from the original at that same level. Furthermore, constraints are maintained between "parent" and "child" levels to avoid high frequencies at "parent" edges by constraining sampling with the features gathered from the Gaussian derivative pyramids. For a complete discussion of technique the author is directed to [DeBo97], however, a brief description follows for convenience: for each pixel, (x, y), in each level, i, in the synthesis pyramid, S[i](x, y), assess a potential swap pixel, (x',y'), in the analysis pyramid, A[i](x',y'), by computing the distance between all parent pixels in S and A at each level j in A(x',y')'s parent feature structure. If the sum distance is below the acceptable distance metric, then the pixel is added to the set of swappable pixels from which the synthesized pixel is selected, in a uniform psuedo-random fashion.

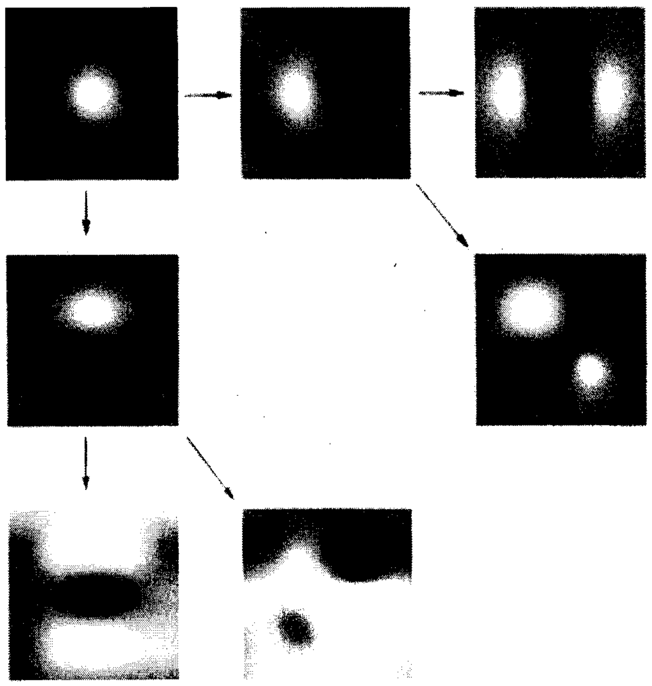

Figure 4.3: Gaussian Filters in 2D with arrows pointing to derivatives.

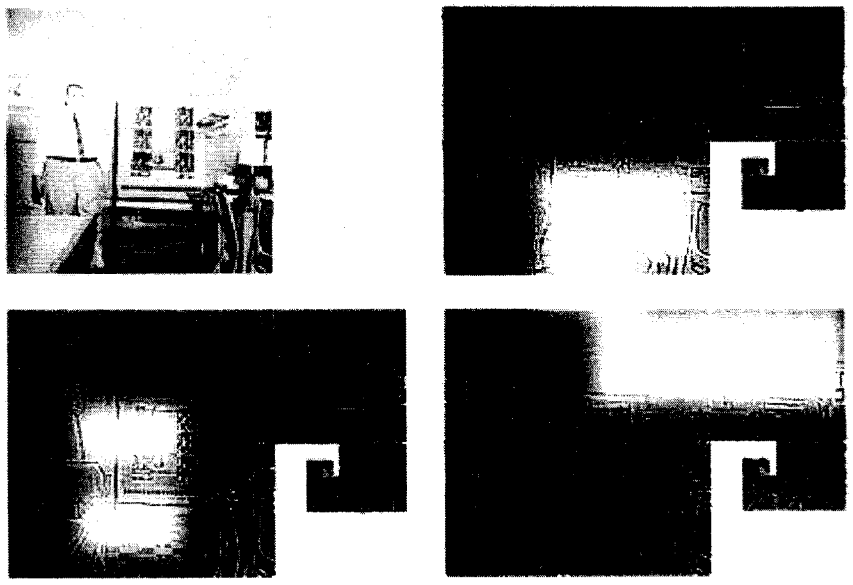

Figure 4.4: Laplacian Pyramids: (clockwise from top left) original image, pyramids on R, G and B channels.

The method of analysis and synthesis of 2D textures has been re-implemented in order to determine its effectiveness in synthesizing textures such as fur, with the intent of extending the method to the generation of 4D light textures. In order to demonstrate the correctness of the re-implementation, Figure 4.5 contains synthesized textures from [DeBo97], followed by Figure 4.6 which contains similar textures generated from the current re-implementation.

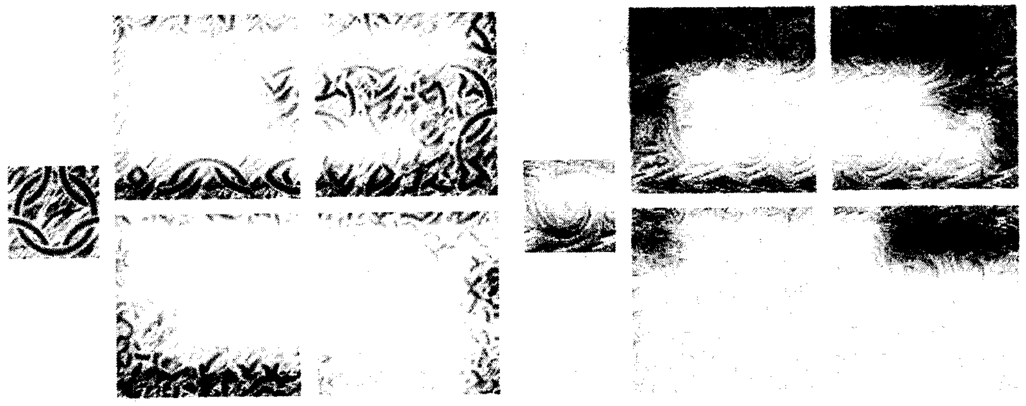

Figure 4.5: Original tiles are to the left, with synthesized textures from [DeBo97] to the right. Top left texture has lowest threshold, increasing clockwise.

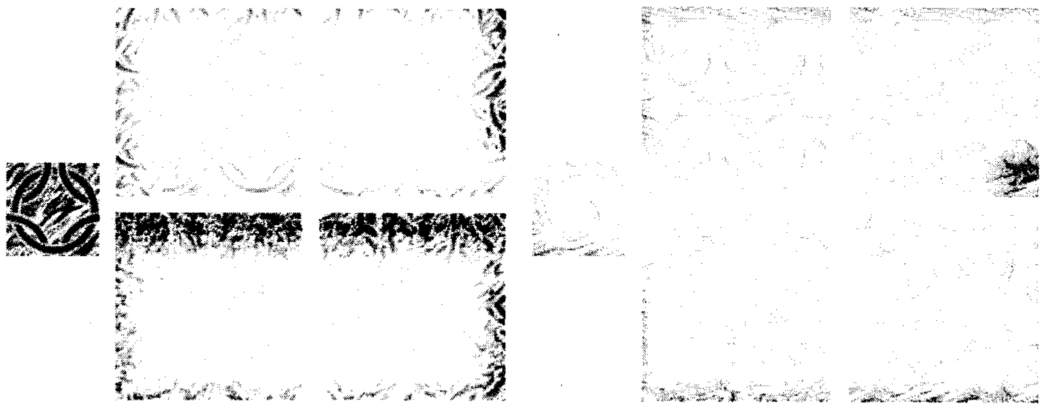

Figure 4.6: Original tiles are to the left, with synthesized textures from current re-implementation to the right. Top left texture has lowest threshold, increasing clockwise.

Considering that the synthesis procedure contains a random component, the textures above are similar enough to indicate correctness.

One of the key motivations for using light textures is the potential for using real light data acquired from real surfaces. In order to test the viability of using the multi-resolution method on real surfaces, such as fur, images were acquired of an actual fur sample using the ACME measurement facility. The UBC ACME facility is an integrated robotic system which can be used to gather a large number of registered measurements. Amongst the varied components, a 3 CCD color video camera exists which was used to gather multiple images of fur textures. But, as the synthesis is applied to an image sample retrieved from the ACME measurement facility, it becomes clear that the multi-resolution method does not work equally well for all types of textures. The synthesis procedure works well for input images which are tillable in the first instance and exhibit distinct textural lines. But, for the textures we are primarily interested in, such as fur, the synthesis process tends to introduce more discontinuities than it prevents, as can be seen from Figure 4.7.

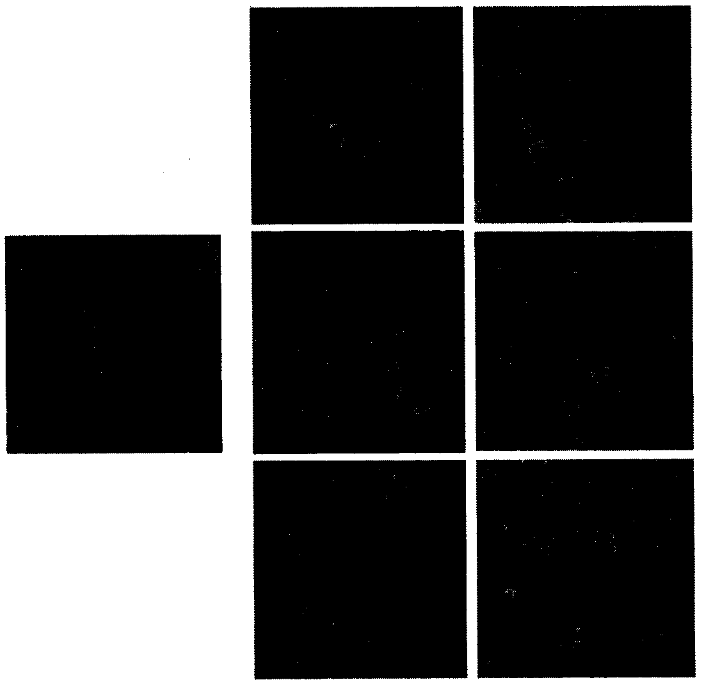

Figure 4.7: 2D Multi-Resolution Synthesis of Fur Image from ACME Facility. Top left texture has lowest threshold, increasing clockwise.

We initially proposed the use of a 4D Gaussian filter along with its first and second partials to extract edge responses at multiple scales from light textures. In 4D, the light field is to be decomposed into multiple Laplacian pyramids [Burt83] of down-sampled light fields. These responses are used to condition a probabilistic re-sampling of the Gaussian pyramid to synthesize a set of unique light texture tiles, with the expectation that each newly constructed tile will possess edges inherently compatible with its neighbor's. But, in addition to the unimpressive results of synthesizing fur textures in

2D, moving from 2D to 4D proved to be too computationally and memory intensive for development, debugging and execution on light fields of significant size. Light fields which can currently be synthesized are blurred and suffer from ghosting to such an extent as to render it difficult to determine if a synthesis algorithm is, in fact, working as planned.

To illustrate this, Table 2 details memory requirements of light textures of varying quality. The qualitative descriptions indicate that a light texture at a "good", or typical, resolution is beyond available computing power. The "minimum" level of resolution which is need to obtain distinctive textural features is also somewhat beyond available means. And, the "current" level that is now attainable is not sufficient for development.

Quality of Light Field	**Size of Light Field (UxVxSxT)**	**Memory Requirements**
Good	256x256x128x128	$\approx$68.719 Gigabytes
Minimum	256x256x32x32	$\approx$4.295 Gigabytes
Current	128x128x16x16	$\approx$268.435 Megabytes

Table 2: Light Field memory requirements.

The memory requirements are calculated with following formulas:

$$p = n^2 + n + 1 - C(n,2)$$

where n is the number of dimensions (4), and p is the number of pyramids. The formula for p can be understood as the initial Gaussian, 1, plus its first derivatives, n, plus its second derivatives, n^2, minus the redundant second derivatives, n choose 2. For an example of what is meant by redundant second derivatives, refer to the identical second derivative filters in the lower right corner of Figure 4.3. The height of each pyramid is simply computed as:

$$h = \log_2(\min(q, r))$$

and the total memory requirements as:

$$\sum_{k=0}^{h} \left(\frac{1}{16}\right)^k (4 * q^2 * r^2 * p)$$

where q and r are the UV and ST dimensions, each level i is 16th the size of level (i-1), and assuming 4 bytes per pixel. This calculation also rests on the assumption that each of the R, G and B channels can be computed separately, otherwise the memory size increases by a factor of three.

Therefore, upon consideration of results obtained in 2D for the textures in which we are interested, and the expense of computation that would be needed for anything but a toy light field, the extension to of the analysis/synthesis procedure to 4D has been, at least, postponed until further computational resources permit its continuation. Due to these constraints only an unstructured collection of 2D images were gathered with the ACME facility for test purposes. Though, it should be noted that ACME is fully capable of acquiring light fields once programmed to do so.

4.4 Silhouette Grafting

When viewing an object which globally exhibits a macro-structured texture with a given depth, silhouette effects are produced between the boundary of the object and its immediate background. An example of this is given in Figure 4.8. Without such silhouettes, light texturing would share the same visual discrepancy as bump mapping. As described so far, the light textures at low incidence viewing vectors do not as yet produce the "fuzzy" edges of the fur surface that would be expected. As described, the boundary of the object behaves like the underlying surface and not like the edges of the texture. This limitation can be overcome by grafting an actual edge from the original texture where the object boundary should be.

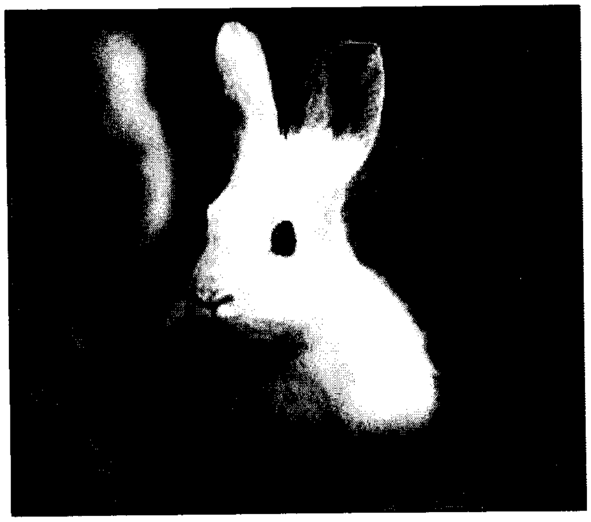
Figure 4.8: Silhouette Example

To implement this we first make use of the related technique of displacement mapping to provide a depth, *d*, to the texture. Displacement mapping takes bump mapping a step further by modifying point *P*, on object *O*, itself rather than normal, *N* at *P*. For polygons, this usually involves an increased division of the surface into a higher resolution mesh of coplanar polygons. Each *P* is then translated with a function ***Disp*** such that the new point:

$$P' = P + \boldsymbol{Disp}(u,v)\,.$$

Although displacement mapping is a marked improvement on bump mapping, it still falls short of being able to model the complexity of surfaces that we are primarily interested in, such as fur. But, we make use of a constant displacement in order to define a double boundary for the surface, *O*, yielding the trivial formula:

$$\boldsymbol{Disp}(u,v) = d * \boldsymbol{N}$$

where *N* is the normalized outward-pointing normal at *P*. The effect of this displacement is shown in Figure 4.9.

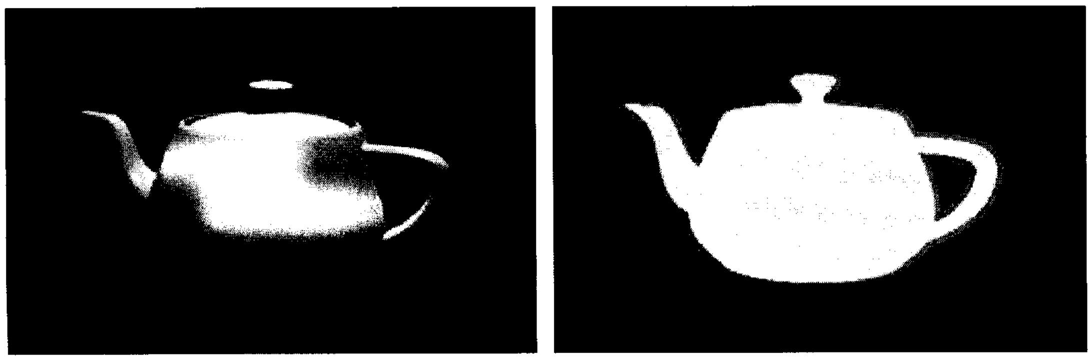

Figure 4.9: Tea Pot (left) and silhouette of inner/outer surfaces (right).

The inner surface, O_I, and the exterior surface, O_E, are then used together in a boolean test to determine if the viewing vector $w = (w_x, w_y, w_z)$ passes strictly through a silhouette. With an intersection function, *inter*(O, w), returning TRUE if view vector w intersects surface O, the table below shows the three categories of vectors:

		inter(O_I, w)	
		TRUE	*FALSE*
inter(O_E, w)	*TRUE*	Regular	Silhouette
	FALSE	N/A	Background

Table 3: Determining Silhouettes.

It is clear from the table that we are interested in $inter(O_E, w) \cap \neg\, inter(O_I, w)$.

Graphing a silhouette onto the rendered object requires a secondary light field, *M*, to be created. This light field is comprised of a single slab in which the *UV* and *ST* planes are defined perpendicular to the texture being captured. The *UV* plane rests along the textures edge; the *ST* plane is set parallel to *UV* at some distance away. An example of a edge light field can be seen in Figure 4.10 below:

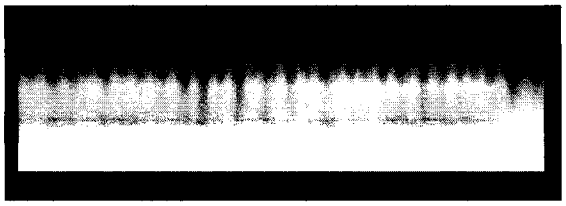

Figure 4.10: Light texture edge.

Once the light field edge has been created and the silhouette has been identified on object O, the UV position in the light field edge, (M_u, M_v), must be determined for each silhouette pixel. We first crop and scale the UV plane of the light field edge so that M_v values of 0.0 and 1.0 correspond to positions at the fur's base and top respectively. The value in the M_u direction then defines the horizontal position from left to right, 0.0 to 1.0. For determining the M_u position in the edge light field, the user has the option of setting the value to a scaled combination of the underlying object's O_u and O_v surface coordinates. Examples of which are: $M_{\mathrm{u}} = ((c * O_u)\bmod 1)$, $M_{\mathrm{u}} = ((c * O_v)\bmod 1)$ or $M_{\mathrm{u}} = \frac{(((c * O_u)\bmod 1) + ((k * O_v)\bmod 1))}{2}$, for some scaling constants c and k. To compute the vertical position, M_v, the distance, d, that the viewing vector, $\boldsymbol{w} = (\boldsymbol{w}_x, \boldsymbol{w}_y, \boldsymbol{w}_z)$, travels within the object's silhouette to surface exit point

Q $(Q_x \quad {}_y, Q_z)$ P $(P_x \quad {}_y, P_z)$

$$M_v = \frac{\sqrt{(P_x - Q_x)^2 + (P_y - Q_y)^2 + (P_z - Q_z)^2}}{r},$$

for some scaling constant r.

The ST position, (M_s, M_t), is then calculated in the same fashion as in the standard light texturing described earlier. With both the UV and ST coordinates known, the intensity value from the edge light field is directly accessed and applied as a transparency channel over the silhouette region. Examples are shown in Figure 4.11 below.

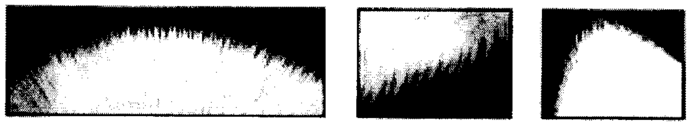

Figure 4.11: Grafted silhouettes.

In addition, to provide greater control, varying lengths and widths of the silhouette can be applied over the surface of an object to correspond to a proportionate (L_u, L_v) scaling of the light texture and object, O_E, thickness. The length, perpendicular to the object, and horizontal scaling of the silhouette can be set by simply assigning varying scale values over the object's surface, as in Figure 4.12. Alternatively, silhouette size can be assigned with the use of a smoothly changing texture as a kind of light texture height-map.

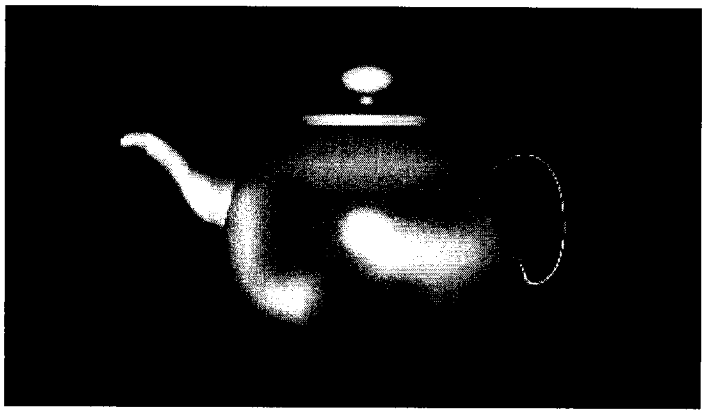

Figure 4.12: Varying silhouette size. Different colors indicate different scale factors.

Also, when taking lighting into account, the same edge can be acquired twice and made use of from two opposing directions. That is, a silhouette edge may be captured with both front and back lighting, each used according to the circumstance.

5 Results and Discussion

5.1 Evaluating Light Textures

In assessing the quality of results, judgment is left to the reader. A number of sample images and image sequences have been created so that accuracy and degree of visual interest can be gauged. Figure 5.1 shows a teapot tiled with a light field generated from a wicker-like geometric model. Figures 5.2-5.4 show various applications of the fur light texture.

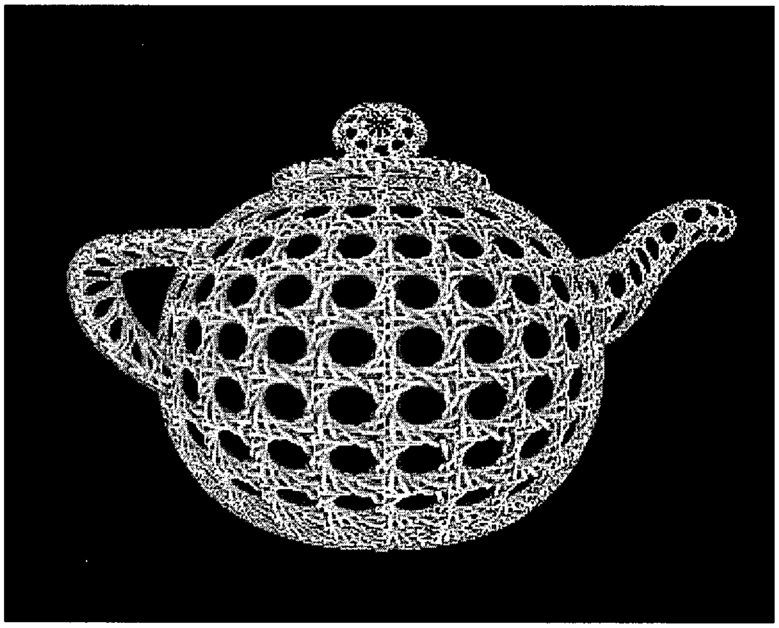

Figure 5.1: Wicker Teapot.

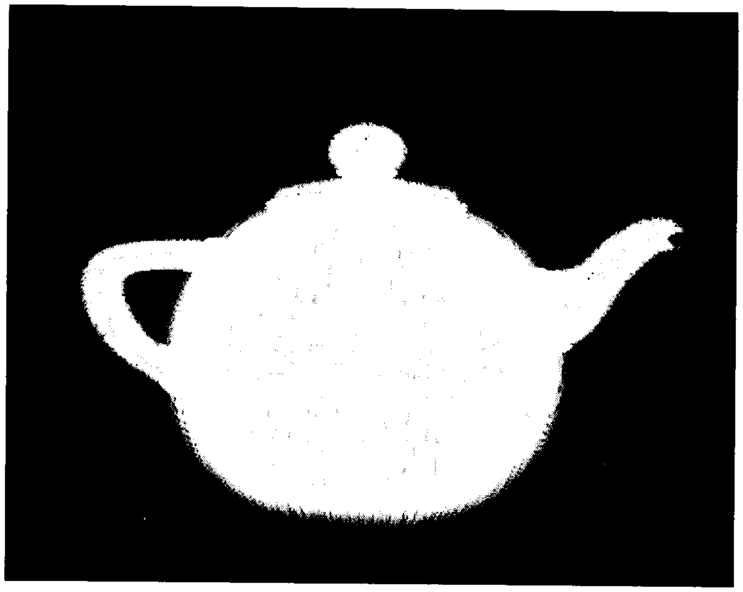

Figure 5.2: Furry Teapot.

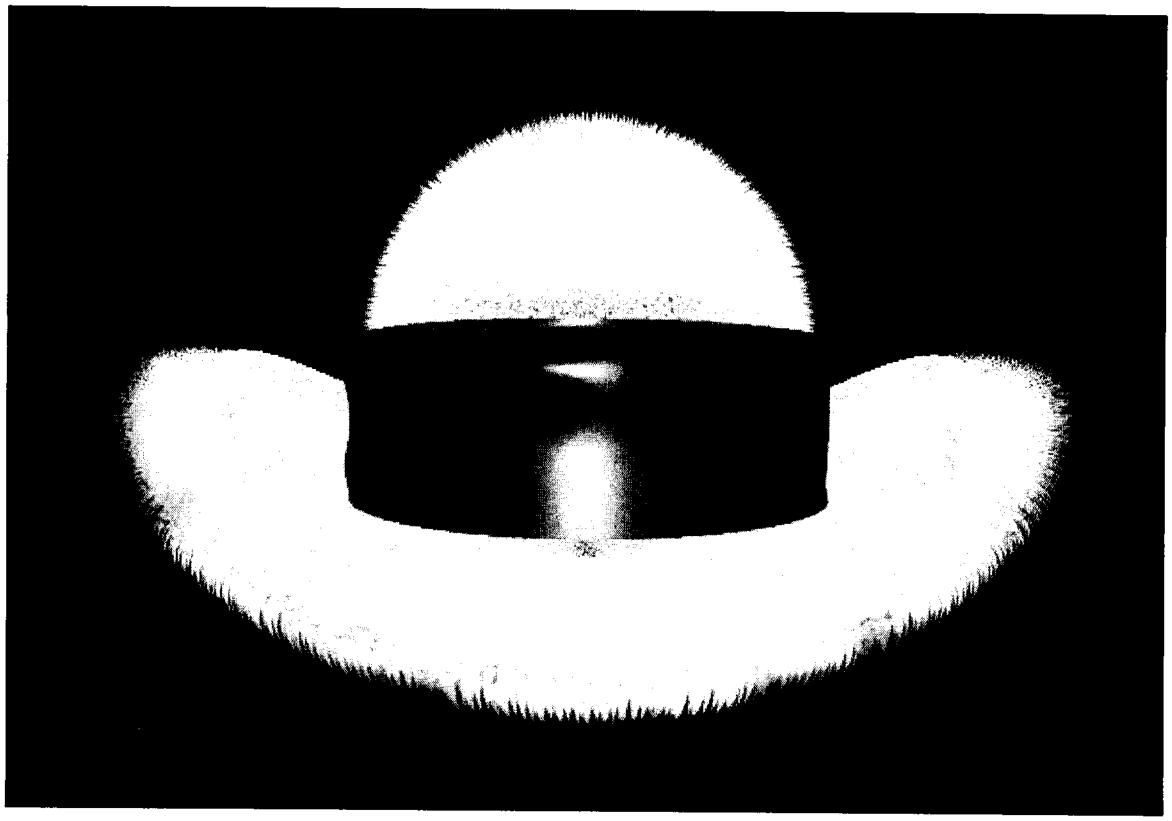

Figure 5.3: Fur Hat.

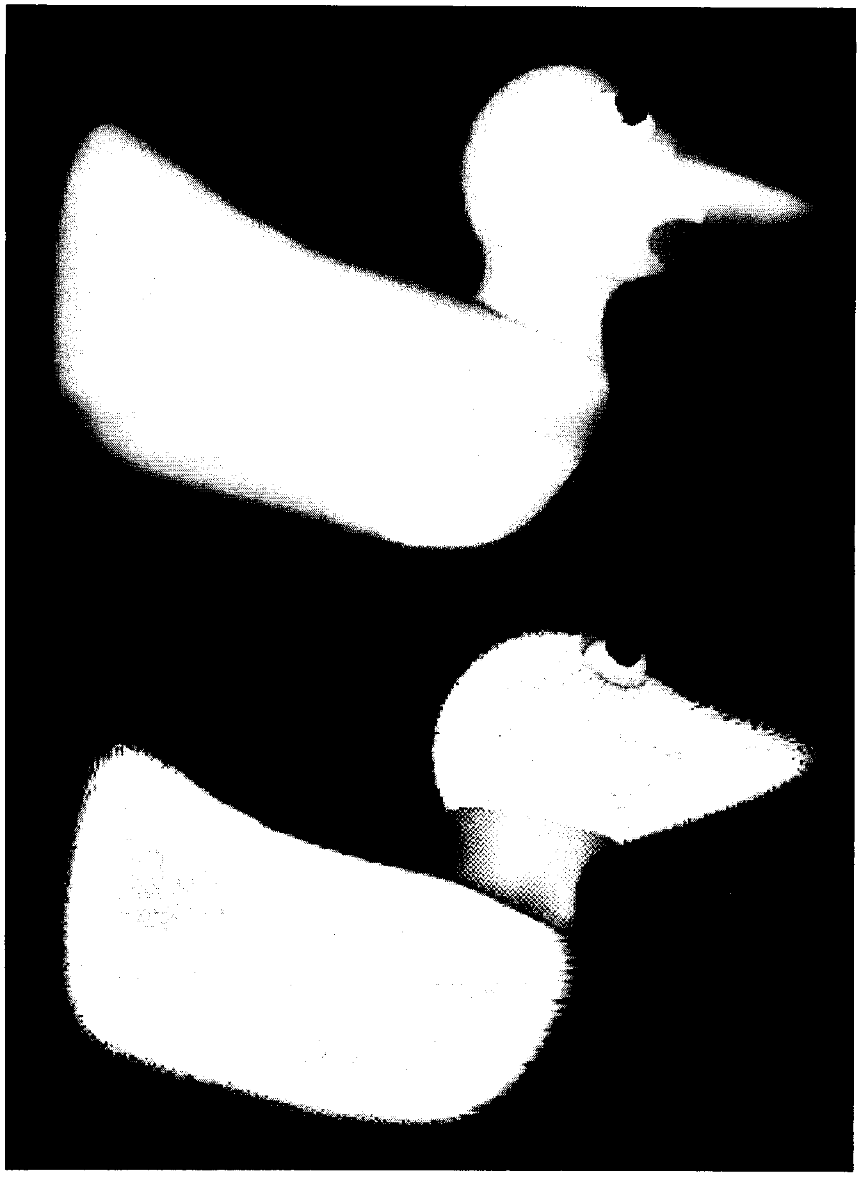

Figure 5.4: Untextured duck (Top). Furry duck (Bottom).

As noted previously, the animation techniques of section 3.2.1 can be applied to light textures tiled across the object's surface. Figure 5.5 depicts a sequence of still images of the procedural wind function displacing the *UV* coordinates of the underlying teapot.

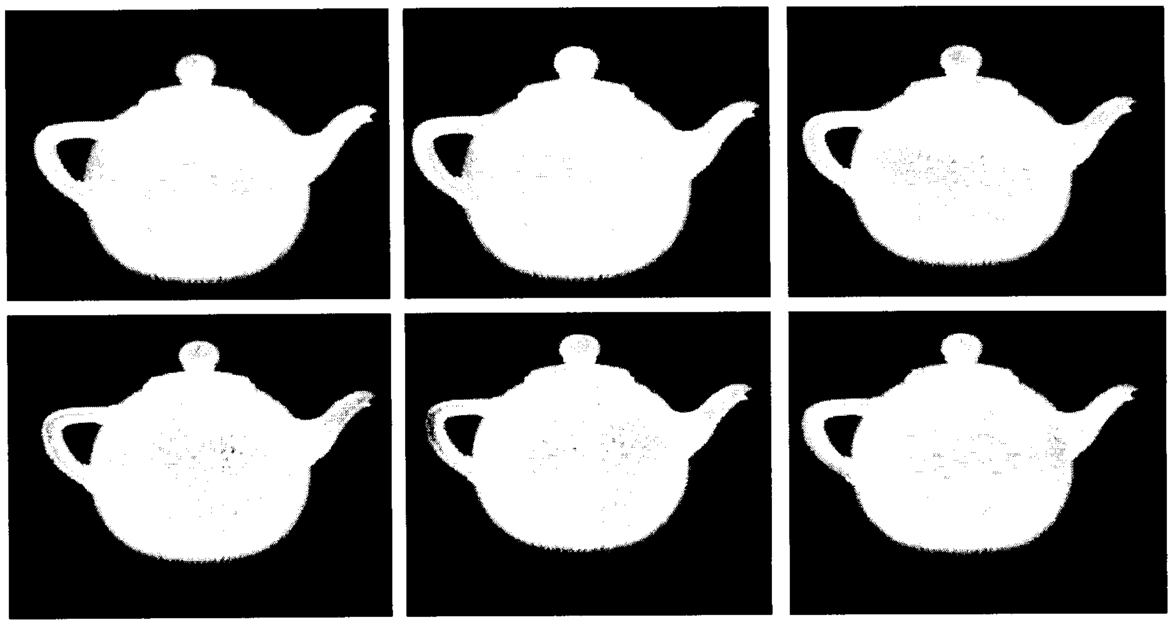

Figure 5.5: Furry Teapot with procedural wind displacement sequence.

5.2 Cost Analysis

Critical components for evaluating any technique that makes use of light field rendering are the associated space and time requirements. The computational requirements of light field rendering as described in [Levo96] is roughly two times the cost of texture mapping. Light texturing, as implemented here, incurs the cost of light field rendering plus the cost of raytracing the untextured object. In terms of space requirements, a light field with 1 slab, 32x32 images/slab, 256x256 pixels/image, for example, would take something on the order of 3.4 megabytes of storage for a compressed data. Each of the quantities related to efficiency are presented in the table below:

Stage	**Significant Quantities**
Acquisition	5 hours
Compression	4 hours
Rendering Rate	1.37 minutes/frame

Table 4: Computational Requirements

Both the acquisition and compression stages are entirely pre-computed, and so can be considered as a one-time expense, separate from the rendering rate. Once the light textures have been pre-computed, they may be re-used indefinitely. In fact, it is only the pre-computation of these aspects which make light field rendering effective. The most important quantity is rendering speed and it is instructive to further divide the time requirements of the rendering rate into:

1. The cost of ray tracing an un-textured surface
2. + The cost of transforming vectors from surface to light field space
3. + The cost of light field rendering (roughly 2x cost of texture mapping).

Significant quantities computed at a screen size of 500x500 of the "fur hat" on a Pentium II 450 mhz with 512 megs of RAM from each of the rendering stages are as seen below:

Stage	**Significant Quantities**
Ray Tracing	1.27 min/frame
Vector Transforms	5.3 sec/frame
Light Field Rendering	0.71 sec/frame

Table 5: Display Rate Requirements

Although the display rate is not real-time, one must consider the visual density of the fur surface and the modest difference in rendering speed between raytracing an untextured surface and raytracing a light textured surface. The same image rendered with only the "constant" Renderman™ shading model required 1.255 min/frame, and with the "plastic" Renderman™ shading model it required 1.3 min/frame. The rendering rate achieved is well within the capacity of current computing power, and is a significant improvement over brute force, geometrically based approaches such as [Kaji89] and [Flei95]. An attempt has been made to compare the computational requirements of various methods of rendering complex surfaces in Table 6. However, it should be noted that the hardware on which the rendering was benchmarked varies significantly, as does the quality of results, and, possibly, image size.

Stage	**Significant Quantities**	**System**
3D Texels [Kaji89]	2 h/frame	16 cpu IBM 3090 mainframe
Cellular Textures [Flei95]	“many hours”	Unknown
Volumetric Textures [Neyr98]	(10 min to 1h)/frame	SGI Indy
Interactive Volumetric Textures [Neyr98]	2.5 frames/sec	SGI O2 with hardware acceleration

Table 6: Comparison of Complex Surface Methods

6 Conclusions and Future Work

6.1 Conclusions

Light textures inherit most of the advantages and disadvantages of light fields themselves. Complex surfaces are rendered at speeds independent of surface complexity with improvements in quality requiring only increased resolutions. It adds a fixed cost per pixel to the rendering time of an un-textured surface. Moreover, multiple types of surfaces can be simulated without the large expenditures of research effort that would be needed if each type were to be generated using a uniquely tailored solution.

The required attributes of a texture in order to be a candidate for light texturing are fairly minimal. Almost anything that can be placed on a flat table and which can be subsequently filtered is a candidate for the method. Complementing this property is the fact that light textures can be tiled on most surfaces, only requiring: a surface normal, surface derivatives and viewing vector for each rendered pixel. And as the measurement of perceived accuracy of artificial images of complex natural scenes is a difficult research problem in itself, it might be argued that since light texturing is potentially reality-based, the necessity for such metrics is diminished as the approach would implicitly approach real surface appearance. With this in mind, we suggest that the use of actual light samples from natural surface constitutes a certain step in the right direction, for which the current work has laid a foundation.

However, the main limitation is currently the inability to re-light the surface. This is indeed a major obstacle if the problem cannot be addressed, but we feel that it is in the category of an open problem rather than an unmanageable impediment.

6.2 Future Work

The work presented here leaves a number of avenues open to further investigation. We might speculate on the development of further techniques such as implementing a method of light texture re-lighting. As mentioned previously, concurrent research by [Mene99] is directed towards this very problem. However, this re-lighting method requires an octree reconstruction of the original object's geometry. The geometry is approximated by carving away voxels that are considered to be outside of the light field. A voxel is considered outside if there is some light ray in the light field which passes through it and does not intersect the object. But, for a light texture, the entire geometry cannot be reconstructed from the silhouette. This is partly due to the complexity of the surface, but also because closer strands of hair will occlude strands which are more distant. Since the geometry cannot be approximated, the method cannot be applied to the application we are interested in.

Alternatively, the brute force method of capturing a light field for every direction of incoming light would require the dataset to move from 4D to 6D and would not be feasible. Yet, if rotational symmetry is assumed for lighting a surface such as fur, it might be possible to approximate the lighting effect with, say, half a dozen light fields with equally spaced angles between lights. Or, it may be that any attempt to re-light a light texture would require assumptions to be made about the particular texture, or that textural features need to be extracted and made use of in some fashion. The use of real light fields as textures may also introduce additional challenges such as minimizing initial lighting imbalance.

Alternate, physically based methods [Shin92] of animating the *ST* grid could be employed by generating the *ST* plane movement from a mixture of wind models and the kinematical movement of the underlying object. For example, in an attempt to model fur attached to loose skin, the *ST* texturing of the light field might be forced to play "catch up" with a delay in matching the *ST* movement of the underlying object, relative to the objects velocity.

The use of procedural vortex functions can be used to approximate the curling of fur by rotating view vectors around surface normals. A simple example of this can be seen in Figure 6.1. But, there may be other methods of rotating the light vectors, such as by making use of vector flow fields. Furthermore, as an extension of the translation and rotational methods described thus far, a procedural light field animation language interface might be created, providing access to *UV*, *ST*, normals, color, illumination, multiple light fields, opacity, and convolutions.

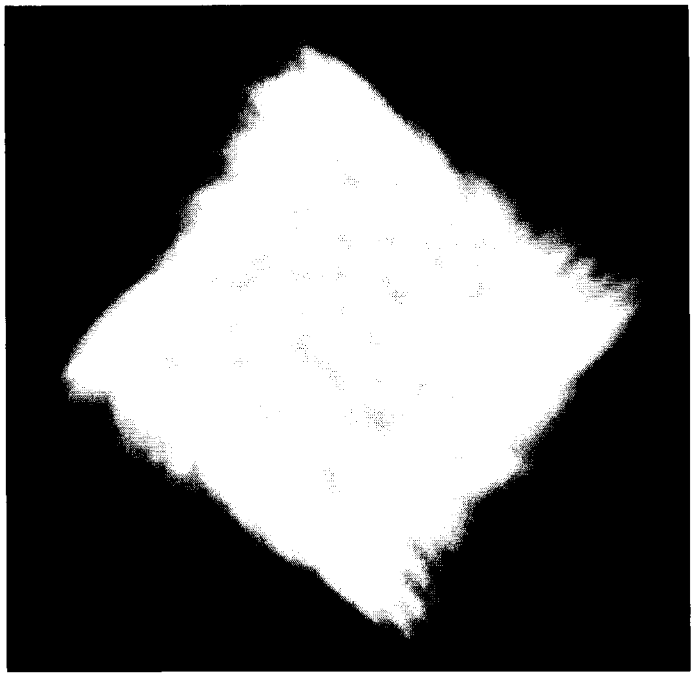

Figure 6.1: Rotating the normals.

Also, 3D warping and repulse point displacement techniques might be developed for light textures, analogous to the 2D versions previously described. These controls would operate in the space of the underlying surface to give direct user control of the global light texture movement.

Additional complexity could be introduced by blending multiple light textures over a single surface. In light texture Mip-Maps we see another potential research direction, specifically with regard to optimization. Further work could also be applied to the filtering of light textures with respect to the scaling caused by the changing distance of the object to the virtual camera. And, finally, more effort could be expended on fully exploring the range of surfaces that can be modeled with light textures, such as: grasses, downy feathers, heavy cloths and weaves, corroded metals, structured stones, bricks and reptilian skins.

Bibliography

[Anjy92] Ken-ichi Anjyo, Yoshiaki Usami and Tsuneya Kurihara. "A Simple Method for Extracting the Natural Beauty of Hair." In *Computer Graphics* (*SIGGRAPH '92 Proceedings*), volume 26(2), pages 111-120. ACM Press, July 1992.

[Blin78] J. F. Blinn. "Simulation of Wrinkled Surfaces." In *Computer Graphics* (*SIGGRAPH '78 Proceedings*), pages 286-292. ACM SIGGRAPH, Addison Wesley, August 1978.

[Blin76] J. F. Blinn and M. E. Newel. "Texture and Reflection in Computer Generated Images." In *Communications of the ACM*, pages 542-547, 1976.

[Burt83] Peter J. Burt and Edward H. Adelson. "The Laplacian Pyramid as a Compact Image Code." In *IEEE Transactions on Communications*, volume 31(4), pages 532-540. IEEE Computer Society, April 1983.

[Chen95] Shenchang Eric Chen. "Quicktime VR - An Image-Based Approach to Virtual Environment Navigation." In *Computer Graphics* (*SIGGRAPH '95 Proceedings*), pages 29-38. ACM SIGGRAPH, Addison Wesley, August 1995.

[Chen93] Shenchang Eric Chen and Lance Williams. "View Interpolation for Image Synthesis." In *Computer Graphics* (*SIGGRAPH '93 Proceedings*), pages 279-288. ACM SIGGRAPH, Addison Wesley, August 1993.

[Chiu98] C. Chiu. *Merging Multiple Light Fields*. M.Sc. Thesis, Department of Computer Science, University of British Columbia, 1998.

[Cook84] Robert L. Cook. "Shade Trees." In *Computer Graphics* (*SIGGRAPH '84 Proceedings*), pages 223-231. ACM SIGGRAPH, Addison Wesley, August 1997.

[Dald97] Agnes Daldegan, Nadia Thatmann, Tsuneya Kurihara and Daniel Thalmann. "An Integrated System for Modeling, Animating and Rendering Hair." In *Eurographics '93 (Proc. Eurographics*), volume 12(3), pages 211-221. Eurographics, Blackwell Publishers, 1997.

[Dana97] Kristin J. Dana, Bram van Ginneken, Shree K. Nayar and Jan J. Koenderink. "Reflectance and Texture of Real-World Surfaces." in *Proceedings of the CVPR 1997*, pp. 151-157.

[Debe96] Paul E. Debevec, Camillo J. Taylor and Jitendra Malik. "Modeling and Rendering Architechure from Photographs: A hybrid geometry-and image-based approach." In *Computer Graphics* (*SIGGRAPH '96 Proceedings*), pages 11-20. ACM SIGGRAPH, Addison Wesley, August 1996.

[DeBo97] J. S. De Bonet. "Multiresolution Sampling Procedure for Analysis and Synthesis of Texture Images." In *Computer Graphics* (*SIGGRAPH '97 Proceedings*), pages 361-368. ACM SIGGRAPH, Addison Wesley, Aug 1997.

[Disc98] Jean-Michel Dischler. "Efficiently rendering macrogeometric surface Structures using bi-directional texture functions." 9th EG Workshop on rendering, Published in *Rendering Techniques 98*, pages 169-180. Springer Verlag, 1998.

[Disc99] Jean-Michel Dischler and Djanchid Ghazanfarpour. "Interactive Image-based Modeling of Macrostructured Textures." In *IEEE Computer Graphics and Applications,* volume 19(1), pages 66-74. IEEE Computer Society, January 1999.

[Eber94] David S. Ebert, F. Kenton Musgrave, Darwyn Peachey, Ken Perlin and Steven Worley. *Texturing and Modeling: A Procedural Approach.* AP Professional, Cambridge, MA, 1994.

[Flei95] Kurt W. Fleischer, David H. Laidlaw, Bena L. Currin and Alan H. Barr. "Cellular Texture Generation." In *Computer Graphics* (*SIGGRAPH '95 Proceedings*), pages 239-248. ACM SIGGRAPH, Addison Wesley, Aug 1995.

[Gard97] Geoffrey Y. Gardner. "Visual Simulation of Clouds." In *Computer Graphics* (*SIGGRAPH '85 Proceedings*), pages 297-303. ACM SIGGRAPH, Addison Wesley, August 1985.

[Glas95] Andrew S. Glassner. *Principles of Digital Image Synthesis*. Morgan Kaufmann, San Francisco, 1995.

[Gold97] Dan B. Goldman. "Fake Fur Rendering." In *Computer Graphics* (*SIGGRAPH '97 Proceedings*), pages 66-74. ACM SIGGRAPH, Addison Wesley, August 1997.

[Gort96] Steven J. Gortler, Radek Grzeszczuk, Richard Szeliski and Micheal F. Cohen. "The Lumigraph." In *Computer Graphics* (*SIGGRAPH '96*

Proceedings), pages 43-54. ACM SIGGRAPH, Addison Wesley, August 1996.

[Grit97] Larry I. Gritz. *Blue Moon Rendering Tools: User Guide.* Blue Moon Systems. Silver Spring, MD, revised 1 December 1997.

[Heck89] Paul S. Heckbert. *Fundamentals of Texture Mapping and Image Warping*. Technical Report No. UCB/CSD 89/516, California Institute of Technology, 1989.

[Heid98] W. Heidrich, J. Kautz, Ph. Slusallek and H.-P. Seidel. "Canned Lightsources." *Proceedings of the EG Rendering Workshop '98*, June 1998.

[Heid99] W. Heidrich, H. Lensch, M. Cohen and H.-P. Seidel. "A Warping-based Refinement of Lumigraphs." *Winter School in Computer Graphics (WSCG) '99*, Feb. 1999.

[Hill90] F.S. Hill Jr. *Computer Graphics*. Macmillan Publishing Company, New York, 1990.

[Kaji89] James T. Kajiya and Timothy L. Kay. "Rendering Fur with Three Dimensional Textures." In *Computer Graphics* (*SIGGRAPH '89 Proceedings*), volume 23(3), pages 271-280. ACM SIGGRAPH, Addison Wesley, July 1989.

[Kang99] Sing Bang Kang and Houng Quynh Dinh. "Multi-Layered Image-Based Rendering." In *Graphics Interface (Proc. Graphics Interface)*, pages 98-106. Canadian Human-Computer Communications Society, June 1999.

[Levo96] Marc Levoy and Pat Hanrahan. "Light Field Rendering." In *Computer Graphics* (*SIGGRAPH '96 Proceedings*), pages 31-42. ACM SIGGRAPH, Addison Wesley, July 1996.

[Lipp80] Andrew Lippman. "Movie-Maps: An Application of the Optical Videodisc to Computer Graphics." In *Computer Graphics* (*SIGGRAPH '80 Proceedings*), volume 14(3), pages 32-42. ACM SIGGRAPH, Addison Wesley, July 1980.

[McMi95] Leonard McMillan and Gary Bishop. "Plenoptic Modeling: An Image-Based Rendering System." In *Computer Graphics* (*SIGGRAPH '95 Proceedings*), pages 39-46. ACM SIGGRAPH, Addison Wesley, August 1995.

[Mein95] H. Meinhardt. *The Algorithmic Beauty of Sea Shells*, release 1, Springer-Verlag, 1995.

[Mene99] Daniel Meneveaux and Alain Fournier. "Reshading Light Fields." In *Proceedings of the Western Computer Graphics Symposium.*, March 1999.

[Meye98] Alexandre Meyer and Fabrice Neyret. "Interactive Volumetric Textures." In *Eurographics Rendering Workshop 1998*, pages 157-168. Springer Wien, July 1998.

[Shin92] Mikio Shinya and Alain Fournier. "Stochastic motion — Motion under the influence of wind." In *Computer Graphics Forum (EUROGRAPHICS '92 Proceedings),* volume 11(3), pages 119-128. Sept 1992.

[Mill98] Gavin Miller, Steven Rubin and Dulce Ponceleon. "Lazy Decompression of Surface Light Fields for Precomputed Global Illumination." 9th EG Workshop on rendering, Published in *Rendering Techniques 98*, pages 281-292. Springer Verlag, 1998.

[Neid93] Jackie Neider. *OpenGL programming guide: the official guide to learning OpenGL*, release 1, Addison-Wesley, Reading, Mass., 1993.

[Neyr98] Fabrice Neyret. "Modeling, Animating, and Rendering Complex Scenes Using Volumetric Textures." *In IEEE Transactions on Visualization and Computer Graphics*, volume 4(1), pages 55-70. IEEE Computer Society, January 1998.

[Open92] OpenGL Architecture Review Board. *OpenGL reference manual: the official reference document for OpenGL*, release 1, Addison-Wesley, Reading, Mass., 1992.

[Pack89] Nancy Huddleston Packer and John Timpane. *Writing Worth Reading*. 2nd ed. St. Martin's Press, New York, 1989.

[Peac85] D. Peachey. "Solid Texturing on Complex Surfaces." In *Computer Graphics* (*SIGGRAPH '85 Proceedings*), volume 19(3), pages 279-286. ACM SIGGRAPH, Addison Wesley, July 1985.

[Perl89] K. Perlin. "Hypertextures." *Computer Graphics,* volume 23(3), pages 253-262. July 1989.

[Schi99] H. Schirmacher, W. Heidrich and H.-P. Seidel. "Adaptive Acquisition of Lumigraphs from Synthetic Scenes." In *Eurographics '99*, volume 18(3), 1999.

[Seit98] Steven M. Seitz and Kiriakos N. Kutulakos. "Plenoptic Image Editing." In *Proc. 6th Int. Conf. Computer Vision*, pages 17-24. 1998.

[Seit96] Steven M. Seitz and Charles R. Dyer. "View Morphing." In *Computer Graphics* (*SIGGRAPH '96 Proceedings*), pages 21-30. ACM SIGGRAPH, Addison Wesley, August 1996.

[Sims90] Karl Sims. "Particle Animation and Rendering Using Data Parallel Computation." In *Computer Graphics* (*SIGGRAPH '90 Proceedings*), pages 405-413. ACM SIGGRAPH, Addison Wesley, August 1990.

[Turk91] G. Turk. "Generating Textures on Arbitrary Surfaces Using Reaction-Diffusion." *Computer Graphics*, volume 25(4), pages 298-298, 1991.

[Upst90] Steve Upstill. *The RenderMan Companion: a programmer's guide to realistic computer graphics*, Addison-Wesley, Reading, Mass., 1990.

[VanG97] Allen Van Gelder and Jane Wilhelms. "An Interactive Fur Modeling Technique." In *Graphics Interface (Proc. Graphics Interface)*, pages 181-188. Canadian Human-Computer Communications Society, May 1997.

[Walt98] Marcelo Walter, Alain Fournier, and Mark Reimers. "Clonal Mosaic Model for the Synthesis of Mammalian Coat Patterns." In *Graphics Interface (Proc. Graphics Interface)*, pages 82-91. Canadian Human-Computer Communications Society, Oct 1998.

[Wata92] Yasuhiko Watanabe and Yasuhito Suenaga. "A Trigonal Prism-Based Method for Hair Image Generation." In *IEEE Computer Graphics and Applications,* volume 12(1), pages 47-53. IEEE Computer Society, January 1992.

Glossary

2D Texture - A 2D Texture Map is a method of controlling the color of a surface on a texel-by-texel basis. This is commonly achieved by tiling a raster image across a surface. But, 2D surface patterns can also be generated by other means such as "procedural textures" which use the *UV* surface coordinates as input.

3D Texture - 3D textures require the use of the full 3D coordinates of a point in space as input. Although there can be additional input, such as time, the three positional axis are the minimum. This method has been called 'solid texturing', as the 'interior' of the object exhibits the texture throughout, provided the object remains within the defined domain of the texture function. The limitations of this approach are the space (i.e. volumetric data) or time requirements (i.e. procedural function).

Ambient Light - It is the light that has been reflected off secondary surfaces in the environment. Ambient light can be viewed as the remaining light that illuminates object in a world when all light attributable to the direct rays from a specific light sources is removed. Typically, ambient light is simulated in one of two ways: through "radiosity" estimations, or, more simply, by providing an ambient light coefficient, or coefficients, which are applied to each object's surfaces. In the absence of ambient light, an object that resides in shadow would not be visible.

Articulation - Objects are considered articulated if they are composed of multiple parts, some or all of which are separably moveable.

Band-Limited – A function is said to be band-limited if the coefficients of its Fourier transform are non-zero for a finite range of frequencies.

Bilinear Interpolation - Bilinear Interpolation computes a weighted average of the four surrounding elements for some attribute. This convolution is repeated for each element. It is termed bilinear in that the interpolation is conducted linearly in both the horizontal and vertical directions.

Bump Map - A bump map is a method for creating surface complexity without altering the geometry of the original object. Normals are adjusted in accordance with the grayscale values of a tiled bitmap image or with procedurally generated values. Although the bump can provide convincing surface appearance, the object's edges can look anomalous as they remain unchanged.

Cellular Texture - Cellular Textures are tiled patterns of geometrical elements. Each element is constructed through a biologically patterned process of cellular growth constrained to the "host" object's surface. Cellular Texture elements can be defined with volume datasets and with polygonal meshes. [Fleischer 95]

Displacement Map - Displacement mapping takes bump mapping a step further by modifying point P itself rather than normal, N(P). This involves an increased division of the surface into a higher resolution mesh of coplanar polygons.

Filter – An image filter is commonly employed in signal reconstruction. It is a function that interpolates discrete samples to form an approximation of the original function. The interpolation filter is convolved with the sampled data to remove or mitigate spurious discontinuities and fill missing data when samples are spaced too far apart. Other uses for filters include edge detection and image compression.

Hypertexture - An Hypertexture is a solid texture with no well-defined boundary [Perl89]. It constitutes a density function that details the transition at the area within the object and the area without. An 'Hypertextured' object can be divided into three regions: a hard region within which the object is entirely solid, a soft region where the objects density is non-zero and an outside region. It is the hypertexture's behavior in its soft region that characterizes it.

Image-Based Rendering - A reality-based method for generating photo realistic computer graphics. This method differs from geometrically based rendering in that the basic data element consists of photometric observations. [McMillan 95]

Laplacian Pyramid - A decomposition of N-dimensional data into a hierarchy of bands. Each band, or level, is a constructed recursively from the level below through the application of local operators at multiple scales and identical shape. At each recursive step the data is low passed filtered with the result subtracted from the original data at that level, yielding an array containing high frequencies and a complementary array containing low-passed data at a reduced sample density.

Light Field - Light Field rendering reconstructs a subset of the plenoptic function in free-space. A Light Field is an image-based method for capturing the complete appearance of both synthetic and real-world objects. It is a structured representation of light captured under fixed lighting conditions. As a "pure" image-based method, Light Fields do not capture geometric representations of the original objects, though geometry may be approximated from the gathered light.

Light Slab- A Light Field is parameterized as a set of Light Slabs. These slabs are defined as a pair of parallel planes, *ST* and *UV*. Each plane is discretely subdivided in each of the *S*, *T* and *U*, *V* dimensions. For each (*u, v, s, t*) point in the defined space, a color sample is maintained, which are subsequently used as input for reconstructing views of the original object with the appropriate reconstructions kernel (typically bilinear or quadrilinear).

Light Texture - A light texture is a light field that has been tiled over the surface of a geometrically defined surface.

Macrostructure - The macrostructure of a surface texture refers to those characteristics that are manifest global to the surface. Macro features of textures are those qualities that emerge at lower resolutions.

Mip-map – Mip-maps are a hierarchical structure used to store pre-filtered textures at multiple resolutions. The mip-map is arranged in a square divided into areas three of which contain the R, G and B components of the texture at the original resolution. The remains area recursively stores a MIP-MAP of the original texture filtered with 2x super-sampling.

Morphing - Short for metamorphosing, morphing refers to an animation technique in which one image, or geometry, is gradually transformed into another.

Normal - A normal to a plane is defined as the cross product of two vectors lying in that plane. A flat polygon situated in 3-D Coordinate Space has an orientation, pointing out from the surface of the polygon and perpendicular to that surface. As there are two normals, one on each side of the surface, pointing in opposing directions, the choice of the side from which the normal projects defines the front or "face" of the polygon. The concept of the normal extends to curve surfaces, where the normal is defined as the cross product of the surface's partial derivatives at a given point.

Pixel - Pixels, short for "pixel elements" might be defined as the smallest element of a display or image whose intensity can be adjusted.

Plenoptic Function - The plenoptic function, first defined by Adelson and Bergen, describes all that is visible from a given point in space. It is a seven dimensional function, defining an intensity for each point in the parameter space where the dimensional axis are: time (t), wavelength(λ), position (x, y, z), and viewing direction (θ, ϕ).

Point Sampling - Also known as nearest neighbor, point sampling is the simplest form of sampling. The attribute's (color, alpha, etc.) value at the position in the defined space is directly selected. The process in repeated for each sample forming the image being reconstructed.

Procedural Texture - Procedural texture mapping is the use of a function or set of functions applied to a set of points in order to generate a texture. Procedural textures are defined algorithmically, typically returning an intensity value or values given the current point in space. The advantages of procedural texturing are that minimal space is required to store the texture's data, as the values are generated as needed. This yields the additional benefit that the textures resolution is not fixed and so can be "zoomed" to an arbitrary degree. This is also a disadvantage as the values need to be computed for each pixel, at each time step, thereby exhibiting the classic trade-off between computational time and space.

Reaction Diffusion - Reaction-diffusion simulates a chemical process over a surface. This process produces stable patterns by diffusing and reacting two or more psuedo-chemicals, which leave a color trace over the surface.

Real-time - In computer graphics, it refers to the generation and display of animation, whether it be pre-computed or interactively generated, without noticeable lag or flicker at some resolution which is sufficient for the application.

Rendering - The range of computation that this term refers to continues to broaden as new techniques of generating views of graphic objects into raster or vector based images emerge. Broadly, the rendering process produces a structured area of intensity levels yielding a n-dimensional slice of a plenoptic function in a given real or artificial area.

Sampling – Sampling is a reduction from the continuous to the discrete by selecting a finite set of values of a function or signal in some structured or random fashion. This is the converse of signal reconstruction.

Texel - Related to pixels, the texel, which stands for TEXture Element, is the smallest addressable unit of a texture map.

View Dependant Texture - View dependent textures are macro-structured surface textures where the intensity of the diffuse surface color (or some subset of the area) depends not only on its location in the objects *UV* space, but also on the viewing angle.

View Interpolation - View interpolation utilizes a partial plenoptic sampling combined with image flow fields in order to reconstruct new viewpoints. [McMillan 95]