MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

A Survey and Classification of Real Time Rendering Methods

Matthias Zwicker, Markus Gross, Hanspeter Pfister

TR2000-09 March 2000

Abstract

The following report is the first of a series of technical documents of the joint research project on the mathematical properties of the representation, compression and dynamics of Surfels, which is carried out by ETH (Zurich, Switzerland) and MERL (Boston, USA). It documents the work performed at ETH between December 1998 and March 1999. The objective of the initial phase of the project was to give a survey and classification of existing techniques, each dealing with a particular issue involved in the project or pursuing a similar purpose as the project. Consequently, this report is mainly presenting a structured literature overview of relevant contributions.

Revision History:–

1. First version, *May 16, 1999*
2. Published as MERL TR, *March 29, 2000*

Table of Contents

1 Introduction

1.1 Purpose of this Report

The following report is the first of a series of technical documents of the joint research project on the *mathematical properties of the representation, compression and dynamics of Surfels*, which is carried out by ETH (Zurich, Switzerland) and MERL (Boston, USA). It documents the work performed at ETH between December 1998 and March 1999. The objective of the initial phase of the project was to give a survey and classification of existing techniques, each dealing with a particular issue involved in the project or pursuing a similar purpose as the project. Consequently, this report is mainly presenting a structured literature overview of relevant contributions.

1.2 Introduction

The research being performed in this project is motivated by the desire to render complex three dimensional objects as efficiently as possible. It is observed that a considerable overhead is introduced with the conventional polygon based rendering paradigm as soon as the polygonal rendering primitives get smaller than a pixel on the screen. This has led to the use of images, rather than polygons, as primitives. Images naturally match the resolution of the frame buffer and as a consequence, the rendering cost is proportional to the number of pixels in the image rather than to scene complexity. Recently, a wide variety of image based rendering methods has been presented demonstrating the speed and viability of the image based rendering paradigm. Despite their success, those methods come with serious drawbacks such as large memory requirements, noticeable artifacts from many viewing directions, the inability to handle dynamic lighting, restricted position of the viewpoint and others.

The fundamental approach of this research is to use points as a display primitive. Objects are represented as a dense set of surface point samples. In contrast to conventional image based methods, these point samples are different in that they contain additional geometric information and that they are view independent. The key to the success of a method working with such samples is the ability to efficiently project the samples onto the screen and reconstruct continuous surfaces without holes on the screen. To this aim, a suitable representation of point samples, an algorithm for projecting them efficiently and a method to reconstruct continuous surfaces have to be developed. The system envisioned should feature the speed of image based methods with the quality, flexibility and memory requirements approaching those of the polygonal approach.

1.3 Overview

In chapter 2, the problem statement underlying this project is presented, along with an introduction to the concept of *Surfels*, which is fundamental to the research being done. The main research issues are listed, motivating the direction of the literature survey given in chapter 3.

2

Problem Statement and the Concept of Surfels

2.1 Problem Statement and Goal of the Project

The core objective of this project is to design novel *methods for real time image synthesis of three dimensional scenes*. A scene is supposed to consist of geometrically complex objects with different surface properties, such as texture, reflectance and transparency. It should be possible to perform rigid motion of objects in the scene, as well as to perform shape deformation. The objects should be rendered under various lighting conditions applying a local illumination model, which includes multiple, dynamically changing light sources and produces correct shadows. The rendering algorithm should be capable of delivering images at interactive frame rates, even for very complex objects.

The most common rendering paradigm that is able to perform such a task is the *geometry based rendering* approach. Objects are basically represented as triangular surface meshes, whereas each triangle is rendered onto the screen by transforming its vertex coordinates to screen space and then scan converting its projected area.

This project will examine the concept of *Surfels* (see section 2.2) as an alternative rendering primitive. Our goal is to analyze under which conditions the Surfel representation is superior to conventional techniques, come up with solutions for the problems involved and demonstrate the rendering performance with a prototype implementation.

2.2 The Concept of Surfels

The concept of Surfels (surface elements) [42,57] has been proposed with the aim of developing a representation of three dimensional objects tailored for fast rendering. The Surfel paradigm basically consists of a novel approach to surface discretization. Unlike classical surface discretizations, i.e. triangle or quadrilateral meshes, Surfels match the resolution of the frame buffer. A single Surfel can be viewed as a point sample of the object surface with attributes comprising spatial position, texture, normal and others. Combining those two fundamental ideas, the Surfel representation of a three dimensional object essentially consists of a set of surface point samples with a certain spatial density, such that its surface can be reconstructed on a screen with a given resolution. The rendering process is accomplished by shading each Surfel and splatting

it onto corresponding image pixels. The Surfel representation of geometric objects is computed as a pre-process with appropriate sampling procedures.

2.3 Research Issues

To develop a Surfel system, a wide variety of issues has to be considered. The most important ones can be summarized as follows, motivating the focus of the literature survey (chapter 3):

2.3.1 Surfelization, Sampling

Surfel generation (*Surfelization*) is a discretization, or more precisely, a sampling process of the underlying geometry, where the sampling density has to be adapted to the resolution of the frame buffer. Additionally, the computation of correct sampling rates has to be governed by given surface and texture error bounds. Particular surface representations such as NURBS, subdivision surfaces, progressive meshes or implicit surfaces have to be examined.

2.3.2 Representation and Compression

To be most useful, Surfels need to be sampled at multiple resolutions. Many problems arise in efficiently storing and using these samples. Hierarchical representations, progressive methods and level-of-detail (LOD) approaches should be considered for this purpose. Those concepts could be applied to develop an advanced data structure, allowing efficient storage and elegant level-of-detail rendering.

2.3.3 Rendering

A rendering procedure that provides interactive frame rates and high image quality is envisioned. Therefore, the procedure has to be fast and simple, opening the way for hardware acceleration. Consequently, any surfel rendering algorithm has to conform to the underlying representation. The aliasing problem has to be investigated.

3

Literature Survey

3.1 Real Time Rendering Techniques

3.1.1 Overview

There are two fundamentally different paradigms providing solutions to the problem of synthesizing images from three dimensional scenes in real time, namely *geometry based* and *image based* rendering (*GBR* and *IBR*). In the context of this overview, those two names are not used as labels to classify different techniques, but as terms for the set of certain essential characteristics of different rendering concepts. As will be shown, there are lots of individual techniques making use of a mix of properties associated both with the geometry based as well as the image based paradigm.

The characteristics of the two paradigms are described by distinguishing four conceptual components of a real time rendering system (table 3.1). The first part describes the form of the initial *scene description* that can be handled by the paradigm. For geometry based rendering, the scene consists of geometrically defined surfaces with various properties, such as color, reflectance or transparency, and a formal specification of the lighting conditions. The image based rendering paradigm relies on a scene description in terms of the *plenoptic function* [1] (see also section 3.1.5). The second stage is the *discretization step*, which transforms the initial scene description to the internal representation that will be fed into the rendering procedure. Generally spoken, this is achieved by sampling the scene description. In the case of GBR, the term *tesselation* is most commonly used, which means that the surfaces are discretized into triangular meshes. The triangulation is optimized regarding the geometric properties, e.g. curvature, of the surfaces. In contrast to that, the sampling strategy of the IBR approach is independent of the scene and solely optimized regarding the screen resolution. The result of the discretization procedure is stored in an *internal representation*. In GBR, this representation consists of a set of primitives, such as lines, points, triangles, triangle strips, triangle fans etc. and the description of the light sources. Conversely, IBR stores a set of *n*-dimensional samples. Finally, the *image synthesis* stage is producing a particular view on the scene. In the GBR paradigm, this is essentially a simulation problem, which is tackled by a conventional rendering pipeline [16]. For IBR, image synthesis is achieved by reconstructing a particular two dimensional slice of the plenoptic function.

Note that for either paradigm, conventional systems only perform the image synthesis stage in real time, which is indicated by the grey shading in table 3.1. The discretization is usually done in a separate preprocessing step.

Table 3.1: Properties of real time rendering paradigms

	Geometry based paradigm	**Image based paradigm**
Scene	Description in terms of geometry, surface properties, lighting conditions	Description in terms of the plenoptic function
Discretization	Sampling (tesselation) optimized regarding geometric properties	Sampling optimized regarding screen resolution
Representation	Set of primitives (polygons, polygon strips, light sources)	Set of *n*-dimensional samples
Image synthesis	Conventional rendering	Reconstruction

The properties of the GBR paradigm induce a number of problems. First of all, the discretization procedure requires a three dimensional surface representation, which can be difficult to provide. Additionally, surface properties such as curvature and texture have to be known to perform reasonable sampling. In the rendering step, the processing of primitives that are smaller than the resolution of the frame buffer is an overhead. Hence, the rendering cost is highly dependent on the scene complexity, meaning the number of primitives in a scene.

On the other hand, the IBR approach has certain advantages over GBR. There is no need for the construction of a geometric model. Models from real scenes can be acquired from sampled images, e.g. from a CCD-camera. Unlike GBR, the rendering cost in IBR is independent of the scene complexity. But IBR is basically restricted to a single operation, namely the reconstruction of a sampled function. Thus it is not clear how to elegantly provide other operations like changing the shape of objects, changing the lighting conditions or the reflectance properties.

There has been a lot of research activity to develop efficient rendering techniques and most of them make use of ideas from the geometry based as well as from the image based rendering paradigm. An overview arranging some of those techniques is given in fig. 3.1 The techniques are ordered from left to right, ranging from purely geometry based to image based techniques. In between, there is a bunch of approaches making use of properties from both sides. In each of the four stages mentioned above, there is a certain contribution from the GBR as well as the IBR paradigm. In the figure, the weight of the contribution from the paradigms is indicated by the distance of a small circle to the corresponding property, whereas the properties of the GBR paradigm are placed on the upper and the corresponding IBR properties on the lower side. A circle lying exactly in the middle of the line between the GBR and IBR characteristic of a certain stage therefore means equally weighted contributions from both paradigms.

As annotated below the figure, the techniques are further categorized into four groups each described in more detail in one of the following subsections.

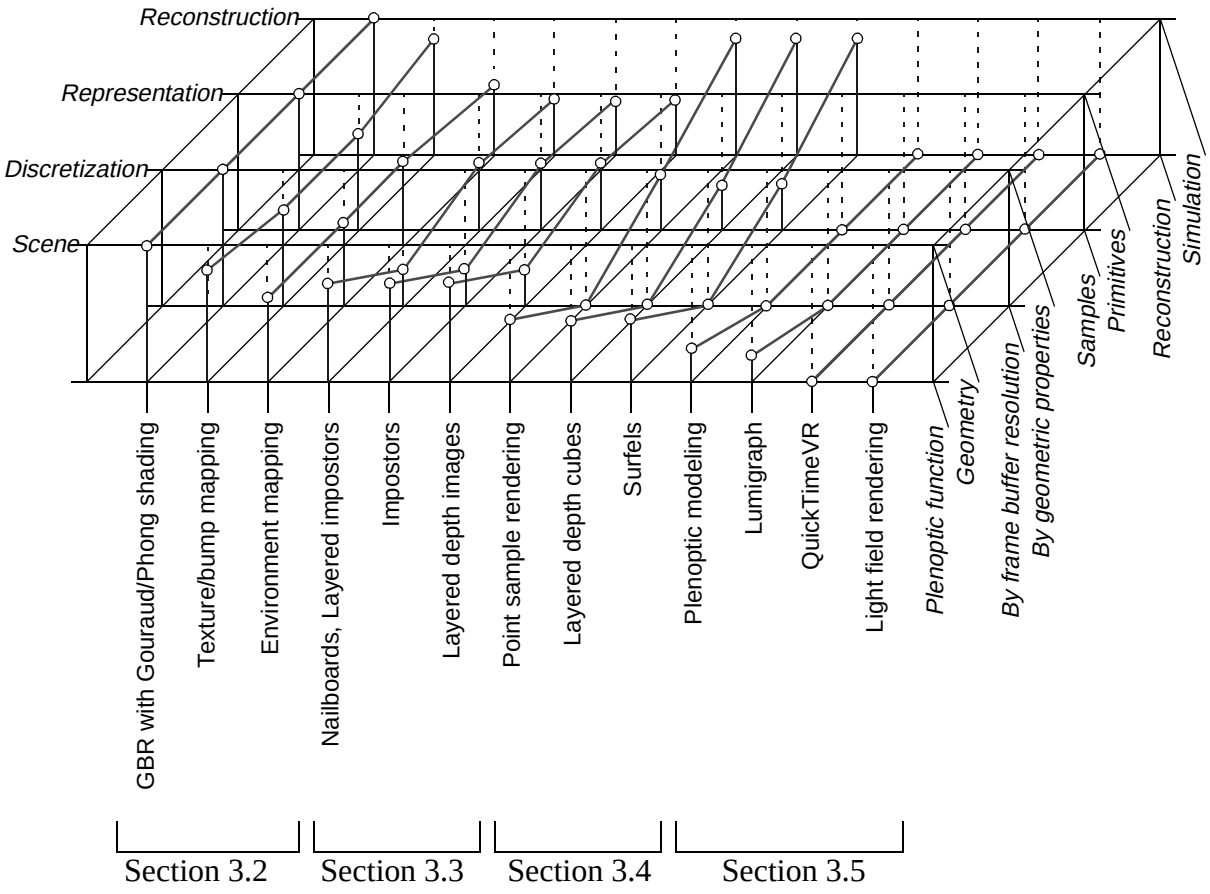


Figure 3.1: Overview of rendering techniques

3.1.2 Geometry Based Rendering and Texture Mapping

3.1.2.1 Shading Techniques

In geometry based rendering the illumination of a scene has to be simulated by applying a shading model. As hardware systems provided more and more computing power, those models became more sophisticated. Gouraud shading [19] is a very simple technique that linearly interpolates color intensities calculated at the vertices of a rendered polygon across the interior of the polygon. Gouraud shaded polygonal models look smooth, since the intensities change continuously across polygon edges. This approach is still used in today's hardware accelerated rendering pipelines. Phong introduced a more accurate model [43] that is able to simulate specular highlights. He also proposed to interpolate normals instead of intensities on rendered polygons, thus enabling more accurate evaluations of the actual shading model. Blinn adapted the Torrance-Sparrow model [55] that describes the reflection of light from real surfaces for use in computer graphics [3]. This model has predicted experimental results very closely, whereas some differences arise with the Phong shading function. All of those models are *local* in the sense that they fail to model global illumination effects such as reflection.

There is a second class of illumination models that can be applied to polygonal scenes, the so called *global illumination* models. Unlike the methods mentioned above, these methods are able to simulate the inter-reflections between surfaces. On the one hand diffuse inter-reflections can be simulated by the *radiosity* method, on the other hand specular reflections are handled by

recursive ray-tracing techniques. But both of them are computationally too complex to be used for real time image synthesis on available hardware.

3.1.2.2 Texture Mapping

In order to improve the visual realism of images synthesized by geometry based rendering systems, a number of techniques have been developed. The basic mechanism to enhance the geometry based approach is to add image based information to the rendering primitives.

The most common of those techniques is *texture mapping*, which enhances the visual richness of raster scan images immensely while entailing only a relatively small increase in computation. A concise survey of texture mapping is given in [24]. Texture mapping is simply defined as the mapping of a function onto a surface in 3-D. Therefore, a texture is a one, two or three-dimensional function, which can be represented by discrete values in an array or a mathematical expression. The study of texture mapping is split into two topics: the geometric mapping that warps a texture onto a surface, and the filtering that is necessary in order to avoid aliasing. In general, texture mapping can be applied to any parametric surface patch. In real time systems however, surfaces are represented as triangular meshes, meaning that texture mapping is applied to each triangle. Clearly, a texture has to be sampled when the surface upon which it is mapped is scan converted. This sampling is adapted to the resolution of the frame buffer, i.e. it makes use of the image based sampling strategy. Texture mapped polygons thus can be regarded as image based rendering primitives.

There are many different uses for texture mapping, differing mainly in the parameters that are mapped onto surfaces:

The idea to map an *image* (i.e. color intensities) onto a surface patch was first introduced by Catmull in [8]. He was the first to observe that one could make a correspondence between any point on a patch and an intensity on a picture by establishing a relation between the bivariate parameterization of the patch and the image. He also realized that in practice this method was prone to aliasing problems and suggested alleviating it by mapping areas to areas instead of points to points.

In [5], Blinn introduced the idea to use a texturing function to perform a small perturbation of the direction of the surface normal before using it in the shading calculations. This process subsequently became known as *bump mapping* and yields images with realistically looking surface wrinkles without the need to model each wrinkle as a separate surface element.

Other researchers proposed to map various appearance parameters to surface patches. Blinn proposes to map the specularity (i.e. the glossiness coefficient) [4]. In his work on the visual simulation of clouds [17], Gardner uses texture mapping to map transparency (i.e. the alpha channel) onto surfaces. Another idea is to map the diffuse reflection property [39]. Cook [11] describes a general method called shade trees to provide flexible shading models. As one of the more exotic uses of shade trees he presents an extension to bump maps called *displacement maps*. Since the location is an appearance parameter, it can actually be moved as well as the surface normal can be perturbed. Cook considers these maps almost a type of modeling.

There is an other kind of texture mapping called *environment or reflection mapping*, first discussed by Blinn [6]. A reflection map is not associated with a particular object in the scene but with an imaginary infinite radius sphere, cylinder or cube surrounding the scene [20]. Whereas in the above approaches the texture function is evaluated using the surface parameters u and v, in reflection mapping techniques, the texture is evaluated using the surface normal or the reflected ray direction, leading to diffuse reflection maps [39] and specular reflection maps [6] respectively. The technique can be generalized for transparency as well, evaluating by the refracted ray direction [26]. Environment mapping facilitates the simulation of complex lighting

environments, since the time required to shade a point is independent of the number of light sources. Moreover, it is an inexpensive approximation to ray tracing for mirror reflection and to radiosity methods for diffuse reflection of objects in the environment.

Independent of the optical and semantic implications of a particular technique, there are two tasks common to all texture mapping approaches: the mapping of the texture function from texture space to screen space and the filtering that is necessary in order to avoid aliasing. The focus of both Heckbert's master thesis [25] as well as his survey [24] is on those computational aspects of texturing. The mapping from texture space to screen space is split into two phases. The first step is the surface parametrization that maps texture space to object space, followed by the standard modeling and viewing transformations that map object space to screen space, typically by perspective projection. These two mappings are composed to find the overall 2-D texture space to 2-D screen space mapping, and the intermediate 3-D space isn't always mentioned explicitly (fig. 3.2).

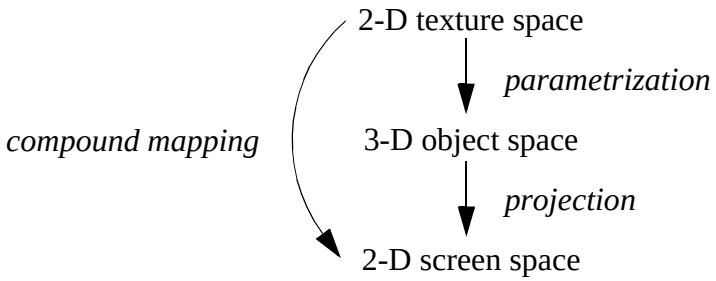


Figure 3.2: The compound mapping is the composition of the surface parametrization and the viewing projection

This simplification suggests texture mapping's close ties with image warping and geometric distortion. A comprehensive treatise approaching the problem from this side is given in Wolberg's book [61]. Heckbert presents the most basic 2-D mappings such as affine mappings, bilinear mappings and projective mappings and discusses them in detail. Once the compound mapping is known, the texture mapped surface can be rendered using one of the following general approaches: scanning in screen space, scanning in texture space and scanning in multiple passes. Screen order, sometimes called *inverse mapping*, is the most common method. In [25], an efficient incremental technique to perform correct texture mapping on planar polygons for projective mappings with screen order scanning is explained. It requires two divisions per pixel, additionally to the linear interpolation of the texture parameters along a scanline.

After the mapping is computed and the texture is warped, the image must be resampled on the screen grid. This process is called filtering. The cheapest texture filtering method is point sampling, wherein the texture value nearest the desired sample point is used. But in general, this results in strong aliasing artifacts. To get rid of those artifacts, more sophisticated filtering methods are needed. For a more general introduction to the aliasing problem in computer graphics, see e.g. [16]. Nonlinear mappings such as projective mappings require space variant filters, whose shape varies as they move across the image. Space variant filters are more complex and less well understood than space invariant filters. For texture filtering, those methods rely on the approximation of a certain pre-image in texture space being mapped to one screen pixel. Pixels are either regarded as rectangles or circles, resulting in pre-images approximated by quadrilaterals or ellipses. The most straightforward filtering technique is direct convolution, which directly computes a weighted average of texture samples on the pre-image.

Mentioning the idea of texture mapping for the first time in [8], Catmull computes an unweighted average of the texture pixels being mapped to each screen pixel. Though he provides few details, his filter appears to be a quadrilateral with a box cross section. An improvement to this was presented by Blinn and Newell [6]. They used a better filter taking the form of a square pyramid with a base width of 2x2 pixels in screen space. At each pixel the 2x2 region is inverse mapped to the corresponding quadrilateral in texture space. The values in the texture pattern within the quadrilateral are then weighted by a pyramid distorted to fit the quadrilateral and summed. A more sophisticated filter was proposed by Feibush, Levoy and Cook [15]. An arbitrary filter function is centered on each pixel and a bounding rectangle is found. Instead of mapping the filter to texture space, only the bounding box is transformed to texture space. Then all corresponding texture values are mapped to screen space and a weighted average is formed using a two dimensional lookup table of the filter indexed by each sample's location. Since the filter is in a lookup table, any high quality filter can be used. Heckbert proposed the elliptical weighted average (EWA) in [21], which is discussed in [25] in more detail as well. This technique assumes overlapping circular pixels in screen space and maps them to arbitrarily oriented ellipses in texture space. Similar to Feibush's approach, the filter is stored in a lookup table, but instead of mapping texture pixels to screen space, the filter is mapped to texture space. In contrast to Feibush's method, the filter table indices are computed incrementally such that it is not necessary to map each pixel from screen space to texture space or vice versa, as in Feibush's method.

Direct convolution methods often are extremely slow, since a pixel pre-image can be arbitrarily large along silhouettes or at the horizon of a textured plane. To speed up the process, the texture can be prefiltered in order to reduce the number of texture samples that have to be accessed for each screen pixel during rendering. Two data structures have been used for prefiltering: image pyramids [14][59] and integrated arrays [12]. Several methods for prefiltering textures are summarized below, each making its own trade-off between speed and filter quality.

An early method using a pyramidal data structure was proposed by Dungan [14]. The pyramid is built by filtering the texture to resolutions of powers of two. To filter an elliptical texture area one of the pyramid levels is selected on the basis of the average diameter of the ellipse and that level is point sampled. This idea was improved by Williams in [59] by proposing a trilinear interpolation scheme for pyramidal images. Bilinear interpolation is performed over two levels of the pyramid followed by linear interpolation between them. This filter has a constant cost of 8 pixel accesses and 7 multiplies per screen pixel. Williams also introduced a particular layout for color image pyramids called the *mipmap* (mip stands for 'multum in parvo'). Greene suggested to combine the EWA filter with an image pyramid. Unlike other prefiltering techniques such as trilinear filtering, this approach allows arbitrarily oriented ellipses to be filtered, yielding higher quality. As an alternative to the pyramidal filtering techniques, Crow proposed the so called summed area table [12], which allows orthogonally oriented rectangular areas to be filtered in constant time. The original texture is pre-integrated in the u and v directions and stored in a high-precision summed area table. To filter a rectangular area the table is sampled in four places.

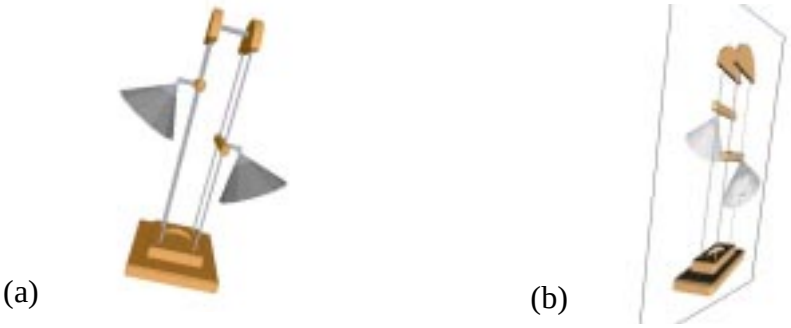
3.1.3 Image Warping

3.1.3.1 General Image Warping Techniques

Available real time rendering systems are designed to render every frame from scratch even though smooth frame sequences contain a great amount of coherence. It has been observed that this process can be accelerated significantly by reusing image data instead of considering the complete geometric description of the scene to generate new frames, thus exploiting spatial and

temporal coherence. By measuring the geometric and photometric error induced by this approximation, it can be determined whether image data must be refreshed or not. In this subsection, a list of techniques is summarized each using different kinds of image based information and associating it with geometric primitives in a different way. But common to all of them is the underlying mathematics used to reproject the image data to synthesize new views, which is basically the same as used for texture mapping. In fact, some of the techniques simply use hardware texture mapping facilities to achieve this task. The techniques described are commonly summarized as *image warping* techniques. A comprehensive coverage of all aspects of image warping is provided in [61].

In [31], Maciel and Shirley describe a visual navigation system which uses texture mapped primitives to represent clusters of objects to maintain high and approximately constant frame rates. Together with traditional LOD representations, the textured clusters are called impostors. An impostor is defined as an entity that is faster to draw than the true object, but retains the important visual characteristics of the true object. The key issue is how to decide which impostors to render to maximize the quality of the displayed image without exceeding a user-specified frame time. In contrast to this approach, where all impostors are generated in a pre-process, Schaufler introduced the dynamically generated impostors in [46]. In his work, an impostor is always represented by an opaque image of an object mapped onto a transparent polygon (fig. 3.3). He points out that as the number of objects increases the approach with pre-generated impostors becomes impractical because of texture storage requirements. Moreover, texture resolution cannot be chosen properly in advance and the discretizations of viewing directions limits the points of view for which impostors can be used. Instead he proposes to generate impostors per object and during render time, arguing that the generation of an impostor is hardly more costly than rendering the object into the final image. To determine the validity of an impostor, he approximates the maximum angle under which the user would see any point on the object and the image of the point on the impostor. As long as this angle remains below a certain threshold the impostor is considered valid. He observes that if the viewpoint is not translated but only the viewing direction is changed impostors will always be valid. In that case the projection of the textured rectangle onto the screen accounts for the changed intersection of the projection rays with the viewing plane.



(a) (b)

Figure 3.3: An object (a) and its impostor (b). The impostor is seen from a different viewpoint than the original geometry.

The image warping approaches described so far all treat the images as planar polygons. Depth priorities are assigned to individual image layers in order to composite multiple layers. But this does not correctly resolve visibility in general, since only one object can be in front of the other. Replacing complex objects by partially transparent textured polygons also produces visibility errors because polygons mutually intersect or intersect other geometry. Another short-

coming of warping planar images is that parallax as well as disocclusion effects can't be simulated.

Schaufler proposed a rendering primitive called *nailboards* that solves the visibility problem [47], extending his concept of dynamically generated impostors [46]. Nailboards are polygons onto which an RGBΔ texture is mapped to mimic the appearance of a complex object. For every pixel in the texture (texel) the additional component Δ measures how far the point of the object depicted on this texel deviates from the polygon. When the texture of the nailboard is generated from object geometry, the Δ values are taken from the depth buffer and stored in the texture together with the RGB values of the image. When the nailboard is rendered, the Δ values are used to change the depth values of the polygon into the depth values of the object represented by the nailboard (fig. 3.4). Schaufler describes the transformations needed in order to determine the depth buffer values when a nailboard is rendered, thus allowing correct determination of visibility using a depth buffer. He points out that nailboards can be mixed arbitrarily with polygonal rendering because they fully integrate into depth buffered rendering.

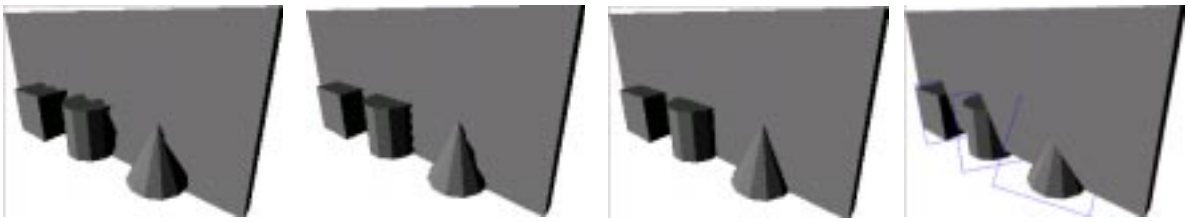


Figure 3.4: From left to right: objects represented by nailboards with 2,4 and 8 bits to represent deviation from nailboard polygon. Far right: visibility artifacts with impostors (figure from [47]).

Another extension of [46] is the concept of *layered impostors* [48]. A dynamically generated impostor replaces an object by one transparent polygon onto which the opaque image of the object is mapped. A layered impostor consists of many such transparent polygons. On every polygon all drawn texels show those parts of the surface of the object which are at a similar distance to the viewer of the polygon, i.e. the pixels of one single image are distributed to multiple texture layers according to their depth values (fig. 3.5). Since layered impostors provide approximate depth values it is possible to resolve visibility correctly. A similar error metric as with single layer impostors is used to estimate the accuracy of layered impostors, but they can be reused by an order of magnitude longer.

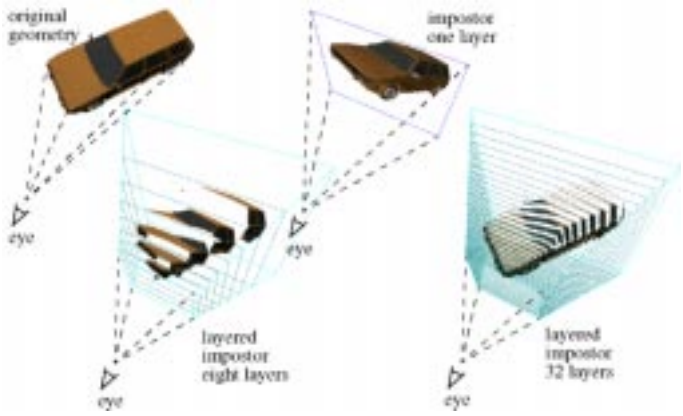


Figure 3.5: Principles of layered impostors (figure from [48])

In [36], McMillan observes the computational advantage of perspective mappings considered as image warping functions [25,61] over pure geometric representations in computer graphics. The warping equations are uniquely invertible and well suited for incremental evaluation. But since single perspective transforms can only convey planar shapes, textures are generally considered as merely an augmentation of the shading process rather than a shape description. He consequently develops a modified perspective mapping function that can represent shape as well as changes of viewing positions. The resulting warping equation takes the form of a perturbation of the planar perspective mapping. The perturbation is given through a generalized disparity term containing the depth information. McMillan explains the relationship of this generalized disparity to the well known stereo disparity. The second important contribution by McMillan is an algorithm that computes visibility without depth [35,34] for his new warping technique. When introducing a perturbation to the perspective mapping, it is not invertible any more, meaning that the one-to-one correspondence between pixels in the reference and the reprojected image is lost (fig. 3.6). Consequently, holes appear and a visibility problem arises. Interestingly enough, the latter can be solved independent of any information relating to the geometry of the scene, including depth information. McMillan describes how to compute a particular drawing order in which the last surface written at each pixel corresponds to the visible surface at that point, similar to the classic painter's visibility algorithm [45]. The drawing order can be determined by finding the projection of the desired viewpoint on the surface to be reprojected. A prove of the algorithm's correctness is provided in [34]. Approaches to the problem of reconstructing the holes are discussed later in this section. A number of applications that use McMillan's warping technique were developed, some of the work is mentioned in the next section. In [37] a variation of the algorithm is applied to reproject a cylindrical image to a planar view, this paper is described in more detail in section 3.1.5.

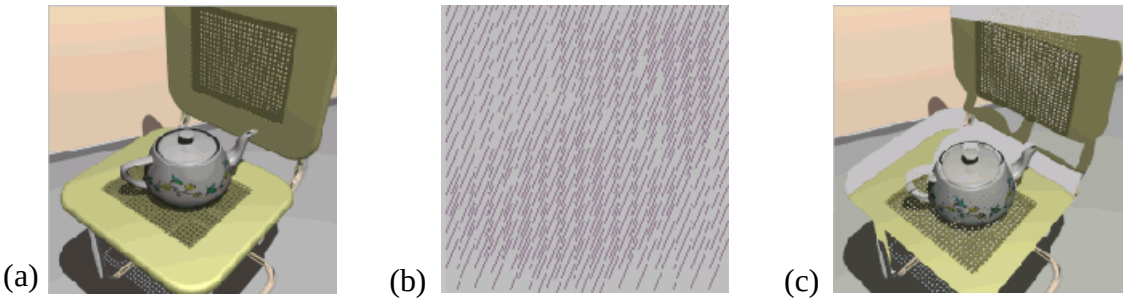


Figure 3.6: McMillan's warping process: (a) reference image (b) lines describing translation of points when warped (c) destination image with holes, but correct parallax (figures from [36])

Two new image based primitives, namely *sprites with depth* and *layered depth images*, were introduced by Shade in [51]. Their work relies heavily on McMillan's ordering algorithm described above. They start with explaining the concept of sprites, which is basically the same as Schaufler's concept of impostors: a textured polygon containing some portion of the scene projected to the image plane. This concept is enhanced by adding an out-of-plane displacement component at each pixel in the sprite, resulting in a primitive very similar to Schaufler's nailboards. But instead of just using the depth information for correct determination of visibility, they describe a warping algorithm that uses this information to represent the shape of the warped objects i.e. simulates parallax effects, too. As mentioned above, in this case the mapping function is not invertible any more. When forward mapping the source image to the destination image, visibility ambiguities as well as holes appear in the latter. McMillan provides an elegant solution to the visibility problem, but does not mention the holes problem in either [36,35] or

[34], though some solutions are proposed in [33] and [37]. In Shade's work, this problem is addressed by using a two step algorithm that first forward maps the original displacements and then backward maps the sprite using the new view based displacements. He points out that the use of sprites with depth provides a fast means to warp planar or smoothly varying surfaces, but more general scenes require the ability to handle more general disocclusions and large amounts of parallax as the viewpoint moves. Such problems can be solved by the layered depth image (LDI) representation. Like a sprite with depth, they store depth values for each pixel along with its color (i.e. a depth pixel), but in addition contain potentially multiple depth pixels per pixel location. The farther depth pixels, which are occluded from the LDI center, will act to fill in the disocclusions that occur as the viewpoint moves away from the center. LDIs are rendered using a fast incremental warping algorithm. It forward maps pixels and splats them into the output image. The warping algorithm as well as a method for calculating the splat size, which is based on an estimated size of the reprojected pixel, are described in detail. They use McMillan's ordering algorithm to resolve visibility. Moreover, a real-time rendering system is described performing two tasks in parallel: a low-priority task constructs layered depth images from multiple prerendered images containing a single depth value and a high-priority task generates images at interactive frame rates using the LDIs.

An overview of the image warping techniques described so far including texture mapping as described in section 3.1.2.2 is given in table 3.2.

Table 3.2: Overview of image warping techniques

Method	Mapping Function	Mapping Direction	Depth Information	Visibility Calculation	Hole Filling
Hardware supported texture mapping	perspective	backward	no	backward mapping	backward mapping
Dynamic Impostors	perspective	backward	no	backward mapping	backward mapping
Nailboards	perspective	backward	one value per pixel, for visibility calculation	z-buffer	backward mapping
Layered Impostors	perspective	backward	depth per layer, for visibility calculation, parallax and inter-object disocclusion	z-buffer	backward mapping
Images with generalized disparity values	perspective with disparity perturbation	forward	one disparity value per pixel, for parallax	list ordering	multiple reference images
Sprites with Depth	perspective with depth	two-pass forward-backward	one per pixel, for parallax	list ordering	two-pass mapping
Layered Depth Images	perspective with depth	forward	multiple values per pixel, for parallax and disocclusion	list ordering	splatting

The so called *view interpolation* method [10] is a slightly different approach to the problem of synthesizing new views using image data. Instead of warping one or multiple reference images to generate a new view, new views are interpolated between several source images using an image morphing technique. Image morphing essentially relies on pixel-by-pixel correspondences between preacquired images called morph maps, which have to be precomputed as well. Using these maps, corresponding pixels are linearly interpolated to create in-between images at interactive frame rates. The interpolation is an approximation to the transformation of the pixel coordinates by a perspective mapping. They state that when the viewpoint moves parallel to the viewing plane, the linear interpolation produces an exact solution. Since the interpolation is a forward mapping process, overlaps and holes may occur in the interpolated image, much as with

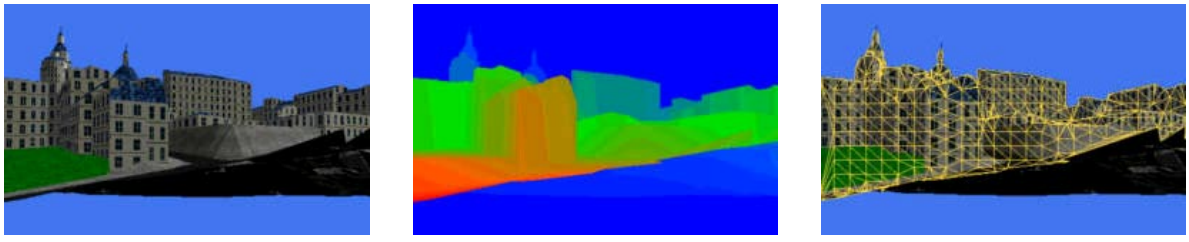
image warping techniques. They propose to use multiple source images to avoid holes. Otherwise holes can be filled by interpolating from neighboring pixels. Another possibility would be to treat a pixel as a little square, interpolate its vertices and scan convert its area in the new image, but they consider this method too costly. The visibility problem caused by overlaps is solved by establishing a list-priority for pixels, guaranteeing correct visibility for a certain range of viewpoints through rendering order. Moreover, they observe that since adjacent pixels tend to move together in the mapping, a block compression scheme such as a quadtree can be applied to compress the morph map. The priority for correct visibility can then be assigned per block instead of per pixel.

3.1.3.2 Applications of Image Warping

A myriad of applications for the image warping techniques summarized in the previous section has been developed. Some examples are collected in this subsection. Rendered images of parts of a scene can be used elegantly as dynamically created view-dependent level-of-detail (LOD) models. This is the basic approach to accelerate rendering of the two papers described next.

Shade et al. presented a method that utilizes path coherence to accelerate walkthroughs of geometrically complex static scenes [50]. A BSP-tree that hierarchically partitions the geometric primitives is constructed as a preprocessing step. In the course of a walkthrough, images of nodes at various levels of the hierarchy are cached for reuse in subsequent frames. A cached image is reused by texture-mapping it onto a single quadrilateral that is drawn instead of the geometry contained in the corresponding node. Clearly, a cached image is a dynamically generated impostor in the sense of [46]. But unlike those impostors, images are cached using a spatial rather than an object hierarchy. Thus they represent regions of the scene rather than individual objects. Spatial partitioning allows depth-sorting the cached images and therefore correct visibility calculation avoiding occlusion artifacts. In fact, Schaufler presented a method [49] very similar to Shade's. The two approaches differ mostly in the formulation of the error metric in the cost-benefit analysis performed in order to decide whether or not to cache an image.

Sillion et al. describe a system for real-time visualization of urban scenery [52]. They introduce a *meshed impostor* representation for the distant geometry in a complex scene. Meshed impostors combine three dimensional geometry to correctly model depth discontinuities and parallax, and textures to rapidly display visual detail. They consist of the image of the distant geometry, which is used as a texture, as well as a triangular mesh, which is used to approximate the geometry (depth) of the distant scene. They describe an algorithm to manipulate those impostors in the context of a city walkthrough considering the special properties of such scenes.



(a) (b) (c)

Figure 3.7: Meshed impostors (a) texture, (b) depth image, (c) triangle mesh. (Figures from [52])

In [33], a more general method to re-render a scene from nearby viewpoints is proposed. This work applies McMillan's algorithm to perform warping of images containing prerendered scenes, so called reference images. The problems of McMillan's algorithm mentioned in section 3.1.3.1 are attacked by extending it in two ways. On the one hand they avoid occlusion-related artifacts by warping two different reference images and compositing the results (fig. 3.8). If a region of the scene is visible in any of the reference frames, it can be correctly placed in the derived frame. On the other hand, they develop heuristics to reconstruct portions of the scene which happen to be occluded in both reference images. They propose to apply a reconstruction kernel with varying size depending on the disparity and normal vector orientation of the corresponding reference pixel. This approach is a form of splatting. The second technique treats the reference frame as a mesh. Reconstruction occurs by rendering the mesh triangles in the derived frame, linearly interpolating the original pixel colors across the reconstructed mesh element. If the reference-frame mesh is treated as completely continuous, surfaces are implied at silhouette boundaries between foreground and background that almost never actually exist. In the derived image, these surfaces appear as so called "rubber sheets" stretching from the edge of the foreground object to the background object and falsely occluding objects behind them. They developed a sophisticated compositing algorithm that takes a binary decision for each pixel in the derived frame, the contribution of which reference image to use in order to avoid occlusion and rubber sheet artifacts.

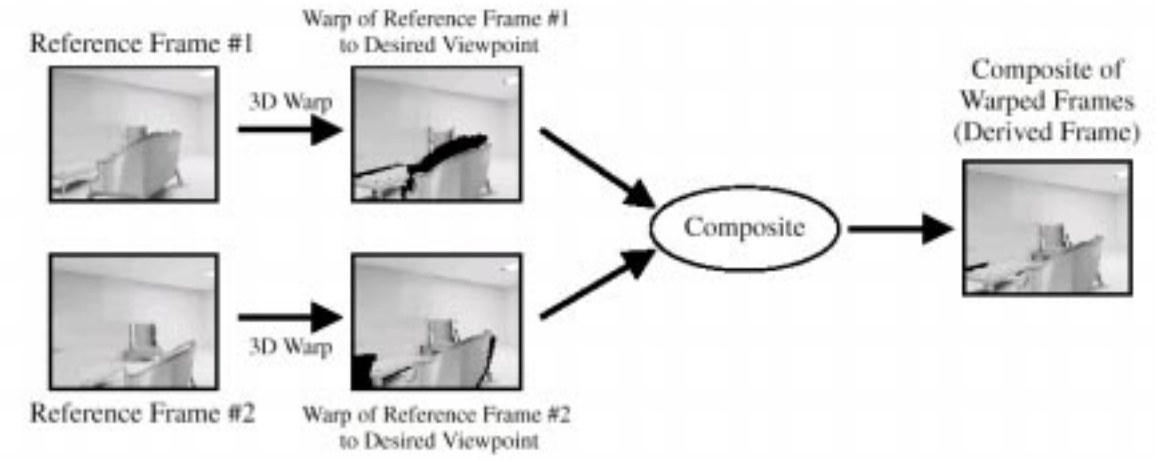


Figure 3.8: Warping with multiple reference images and compositing. (Figure from [33])

3.1.3.3 Hardware Architectures for Image Warping

If one doesn't consider hardware support for texture mapping, not many hardware systems for image warping have been implemented to date. Just two of them are described in this subsection.

It was observed (e.g. [46]) that as long as the viewpoint is not translated but only the direction of view is changed, new views of a three dimensional scene can correctly be generated by a planar perspective reprojection of a projected image of the scene. This observation is the basic idea of Regan and Pose's *virtual reality address recalculation pipeline* [44] as well. They proposed a hardware architecture for virtual reality systems that guarantees very high frame rates of 60+ Hz during user head rotation. Special purpose hardware called address recalculation pipeline is used to perform orientation viewport mapping post rendering, meaning that the orientation mapping occurs after the scene has been rendered rather than as the scene is being rendered. For this purpose, the rendered view must completely encapsulate the user's head, so

when the user's head orientation changes the view from the new orientation can be computed by the address recalculation pipeline from the prerendered image, the pipeline running at video display rates. They chose the surface of a cube as the encapsulating rendering surface. Consequently, the pipeline implements an image warping algorithm similar to the approaches described in section 3.1.2.2. Additionally, they combine the address recalculation pipeline with an image composition technique [41], which makes it possible to render different parts of a scene at different rates. The determination of the rate at which objects have to be redrawn is demand driven, meaning that an object is not redrawn until its image in the display memory has changed by a predetermined threshold. This concept is particularly powerful in conjunction with the post rendering image warping approach, since the rendered image of an object is almost independent of user head rotations and does not need to be re-rendered in that case.

The *Talisman* hardware architecture described by Torborg and Kajiya [54] employs a similar combination of post rendering image warping and image composition. This system was tailored to provide real-time 3-D graphics for the PC at a low cost in order to promote 3-D graphics as an ubiquitous medium. Individually animated objects are rendered into independent image layers which are composited together at video refresh rates to create the final display. During the compositing process, a full affine transformation is applied to the layers to simulate 3-D motion of objects. Layers are treated as flat polygons and presented to the compositor in back to front order, allowing proper transparency calculations using an alpha buffer. The rendering pipeline therefore contains a conventional polygon pipeline followed by an image warping and composition unit. Whereas the polygon pipeline is designed to provide a pixel rendering rate of 40 Mpixels per second with anisotropic texturing and antialiasing, the image layer compositor has a compositing rate of 320 Mpixels per second, in the best case multiplying the overall 3-D rendering performance. It is due to the host software to assign objects to image layers and maintain a priority list of the layers to be updated based on perceptible error and object priorities. This error is computed based on a least square error of selected points within the object, but little detail on this is provided in the paper.

3.1.4 Object Representation with Point Samples

Warping algorithms rely on images as models for scenes or objects. Images therefore can be regarded as point sampled representations of the underlying scene, whereas the point samples are acquired by projecting the geometry to a projection manifold (e.g. plane, cylinder). Some warping algorithms use more information than a conventional two dimensional image can provide, for example multiple depth values per pixel [51]. Still, those techniques have in common that the representation of the scene is view-dependent and doesn't provide a complete model that can be used for rendering from arbitrary points of view. In this section, methods are described that attempt to represent objects with point samples in a view-independent, object rather than image centered fashion.

Levoy and Whitted considered the use of *points as a display primitive* and as a meta-primitive to represent objects in [32]. They proposed an approach requiring two separate stages. The first stage converts objects to points, usually in a preprocessing step, and the second renders those points. They explain their ideas with a simple example, namely a flat rectangular polygon represented as an array of points. The shading of each point is calculated using a color texture, a reflectance map and a perturbation of its height coordinate. A rendering algorithm is described that is able to display such data with the texture properly filtered, edges antialiased and the array completely obscuring its background. The focus of this work is on the filtering technique applied to achieve this. A radially symmetric Gaussian filter is centered at each image pixel and the contribution of each source point to each pixel is determined by computing the distance from

the point to the center of the pixel, then weighting that distance according to the Gaussian function. This can be regarded as a form of splatting [58]. Obviously, the sum of the contribution varies, since the density of source points in close proximity to the center of a pixel varies with the viewing transformation. This problem can be readily solved by summing the weights for each pixel and then dividing the accumulated color by this sum. When a spatially perturbed texture is rendered, silhouette edges may occur at any place in the interior of the texture. Consequently, variations in the sum of the contributions in a display pixel may result not only from variations in the density of the source points but from partial coverage along silhouette edges as well. In the latter case, partial coverage should be computed in order to perform proper blending between texture and background color. They propose to discern between the two cases by predicting the density of source points in a neighborhood surrounding the current transformed source point. The prediction is used to pre-normalize the contributions forcing them to sum to unity. If they do not sum to unity despite the pre-normalization, it is because the texture only partially covers the pixel. The sum of the contributions is then equal to the coverage and may be used to control blending. They propose an algorithm for determining visibility that uses so called bins, each basically storing the contributions of one surface to a display pixel. Once all surfaces are rendered, the bins are merged to compute the final color of the pixel. They shortly discuss the conversion of geometries into points. Their rendering algorithm requires that surfaces have to be contiguous and differentiable in a small neighborhood around each point but nothing is required about the spacing or distribution of the points.

The *delta tree* representation was introduced by Dally, McMillan et al. [13] as an object-centered approach to image based rendering. It basically represents an object using a set of reference images with depth. They observe that typically several hundred images from different viewpoints are required to represent an object and that rendering an image of an object at a new viewpoint may involve reprojecting pixels from several of those images. Unlike this, an ideal space-efficient representation would store exactly one sample for every potentially visible component of an object down to some minimal solid angle of resolvability. Moreover, a representation that requires reading only one sample for each pixel rendered to the screen would provide for optimal rendering performance. The delta tree stores images with depth of an object in a multirooted tree. The images, so called views, are taken from certain viewpoints regularly distributed over a sampling sphere centered on the object. The sampling sphere is parametrized by two angles (θ,ϕ) and a region on the sphere is defined as the area covered by a given range for each angle. Each node in the tree corresponds to a region of the sampling sphere and stores one particular image from a certain viewpoint lying on a corner of the region associated with the node. Each child of a node inherits a quarter of the region of its parent and stores an image of the object taken from that corner of its region that is also a corner of the region of its parent (fig. 3.12, (a) and (b)). In order to provide a space-efficient representation, the image stored at each node is compressed by discarding pixels that can be reconstructed by warping the images of its ancestors to the viewpoint of the node. This results in so-called partial views. A partial view is a sparse matrix of pixels, which is stored as a list of arrays or patches of 8x8 pixels. These patches are transformed into the frequency domain using a DCT, thus elegantly enabling image filtering and compression similar to JPEG. The structure of the delta tree implies that certain nodes contain views taken from identical positions on the sampling sphere. Consequently, these views are shared among nodes by reference to ensure space-efficient representation. The image synthesis process traverses a portion of the delta tree and reprojects the views associated with the nodes visited to a new viewpoint. The traversal path is determined such that the object as seen from the new viewpoint can be reconstructed with a minimal set of views (fig. 3.12, (c)). This process thus performs a kind of implicit visibility culling by leaving out all the views that are not relevant to the new viewpoint. The projected partial views are composited using a z-

Buffer. Reprojection is achieved by applying McMillan's warping algorithm [36] that uses fast incremental calculations. In order to avoid holes, four neighboring points are connected to a quadrilateral which is then scan converted. A limitation of representing objects with images from a sampling sphere is that in general, views from inside the convex hull of an object cannot be generated, because surfaces may be visible inside the object that are not visible from the sampling sphere. In practice, they work around this problem by subdividing objects to render views from inside the convex hull.

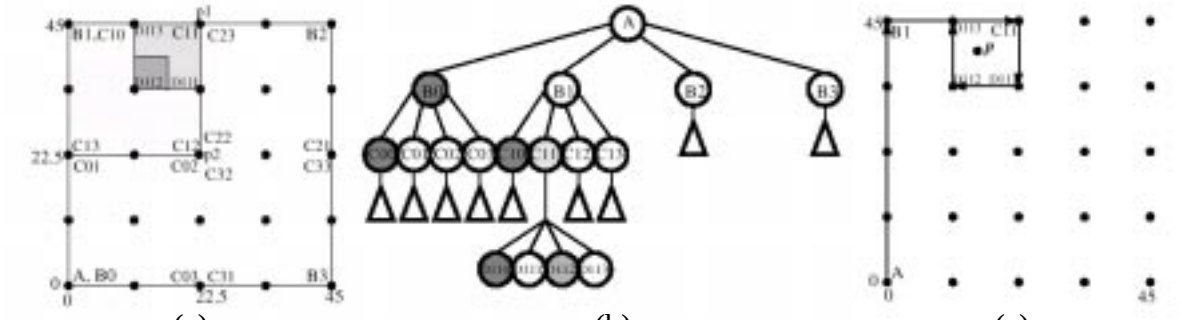


Figure 3.9: The delta tree: (a) Each node corresponds to a region of the viewing space, the whole tree covers $(\theta,\phi)=[0..45,0..45]$. (b) The associated tree structure. (c) Traversal path to display a new view from point *p*. (Figure from [13])

Grossman and Dally proposed an object representation consisting of a dense set of surface point samples (fig. 3.10), which contain color, depth and normal information [23][22], thus enabling Z-buffer composition, Phong shading and other effects such as shadows. They called their system *point sample rendering*. The point samples are obtained by sampling orthographic views on an equilateral triangle lattice. They present a method for determining the side length of the triangles that guarantees so called adequate sampling of the object and thus controls sampling density. The samples from each projection are grouped into blocks of 8x8 samples and a greedy algorithm is used to collect a suitable subset of all blocks needed to represent the whole object but avoiding redundancy.

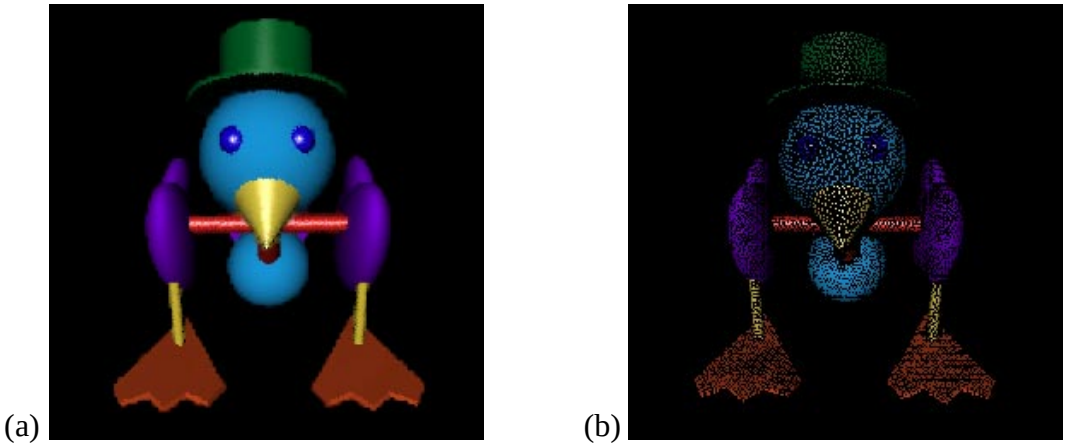


Figure 3.10: An object in (a) and it representation as a dense set of surface point samples in (b). (Figure from [22])

The blocks can be projected to screen using a fast incremental warping algorithm similar to the approaches summarized in section 3.1.3. The fundamental challenge of this technique is the

on-screen reconstruction of continuous surfaces. They propose a method for detecting gaps in projected surfaces and propose to use a push-pull algorithm similar to the one described in [18] in order to reconstruct the surfaces. This problem is not specific to point sample rendering; it is the general problem of image reconstruction given incomplete information, or more general the problem of scattered data interpolation. The described rendering process includes sophisticated visibility testing based on the blocks of samples. For each block, they calculate a visibility cone for fast backface culling and a visibility mask, which is a concept based on normal masks as introduced by Zhang [63].

It is very instructive to compare this technique to the delta tree representation summarized in the last paragraph. Instead of taking perspective projected views, point sample rendering acquires samples from orthographic projections and consequently uses a different warping algorithm. Samples are grouped in 8x8 blocks as well, but there is no structure such as the delta tree that organizes those blocks in a hierarchical fashion. Instead of the implicit visibility culling achieved by traversing the tree, other techniques for visibility computation are applied. Furthermore, the push-pull algorithm used in point sample rendering implements a much more efficient surface reconstruction technique than the simple quadrilateral rasterization performed in the delta tree approach.

The concept of *surfels (surface elements)* was developed at MERL [42,57] and is very similar to the point sample rendering approach by Grossman and Dally. The idea was inspired by the Caviar rendering tool, which was developed by Animatec [2]. In fig. 3.11, a sample image of an object rendered with the Caviar player is shown. The player provides interactive frame rates on a Pentium class PC without using dedicated graphics hardware.

Figure 3.11: Object rendered with Caviar Player by Animatec. The blocky artifacts are due to the box reconstruction filter used for splatting the point samples.

Objects are represented by surface point samples containing color, depth and normal information, similar to the point sample rendering approach [22]. The techniques differ mainly in the sampling strategy and the data structure used to store the samples. Surfels are acquired by rasterizing the object on a regular three dimensional rectangular grid. The grid spacing is identical to the pixel spacing of the frame buffer and thus controlling the sampling density. They store the samples in sequential lists. The rendering process can then be accelerated by using incremental calculations. Several solutions to the surface reconstruction problem are proposed. They considered a shear-warp factorization of the viewing transformation [27]. The object is first pro-

jected to a base plane by a shearing operation and then warped to the final view. This method has the disadvantage that the object has to be resampled twice. Holes could be avoided by supersampling the object. This is a brute force method that significantly increases rendering cost. A third alternative is to use a splatting method, where the contribution of one sample point is spread to multiple pixels in the frame buffer. This is similar to the method proposed by Levoy and Whitted [32], it anti-antialiases by blurring the colors of the object. The system developed by Animatec also implements a splatting technique that uses a box reconstruction kernel. This amounts to simply treating each point sample as a little square with varying size according to the magnification of the object. In [42], some work on physics based motion and deformation of surfel objects is presented as well.

The *layered depth cube* representation was introduced by Lischinski and Rappoport in [30] and can be regarded as an extension to Shade's layered depth images [51]. This paper provides some methodological contribution: it places image-based rendering, light field rendering and volume graphics in a common framework of discrete raster-based scene representations. The layered depth cube representation combines view-independent scene information and view-dependent appearance information. All view-independent scene information (i.e. geometry and diffuse shading) is represented using three orthogonal high-resolution LDIs. The view dependent scene information is stored as a separate, larger collection of low-resolution LDIs (fig. 3.12). They describe rendering algorithms that recombine these two components. The rendering process consists of two stages: first, 3-D warping using the fast algorithm described in [51] is applied to the view-independent LDIs. But since samples are not guaranteed to arrive in back to front order, the simple splatting approach of [51] cannot be used. Instead, they apply a simple heuristic to detect and fill holes using a ray tracing technique. This stage results in a primary image, which is a correct representation of the geometry of the scene as seen from the new viewpoint. This image can be shaded using a local shading model, e.g. Phong's model [43]. In the second stage, reflections are calculated using one of the following two techniques. The first alternative is called light-field gather and integrates the view-dependent low-resolution LDIs by using them similar to environment maps. The second technique applies ray tracing through the view-independent LDIs. The overall process doesn't reach interactive frame rates, but provides ray tracing-like image quality. The paper [30] hardly introduces any new techniques, but presents an appealing novel combination of different techniques resulting in a system with interesting properties.

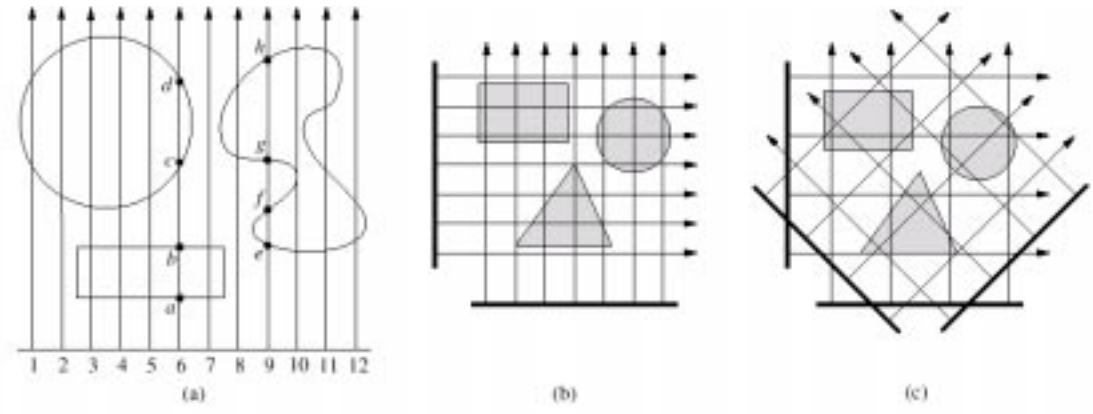


Figure 3.12: (a) A parallel LDI of a 2D scene. Pixel 6 stores scene samples *a, b, c, d*, pixel 9 stores samples *e, f, g, h*. (b) the view-independent high-resolution and (c) the view-dependent low-resolution LDI in 2-D. (Figure from [30])

An overview of the methods summarized in this section is given in table 3.3. For the layered depth cube representation, only the first stage of the reconstruction algorithm is considered, since the other methods don't provide techniques similar to the light field gather or image based ray tracing step.

Table 3.3: Overview of object representations with point samples

Method	Sampling	Data Structure for Storing Samples	Warping	Surface Reconstruction
Levoy	example: regular 2-D grid; not described for the general case	example: 2-D array; not described for the general case	per point transformation	splatting, uses estimation of density of projected samples
Delta Tree	multiple perspectively projected images	delta tree storing compressed partial views at its nodes; each consists of DCT transformed 8x8 blocks of samples	McMillan's algorithm per partial view	rasterization of quadrilaterals
Point Sample Rendering	multiple orthographically projected images	8x8 blocks of samples; visibility mask per block	incremental warping per block	hierarchical z-buffer, push-pull filtering algorithm
MERL Surfels	rasterization on 3-D grid	linear list	incremental offset warping	shear warp factorization, supersampling or splatting
Layered Depth Cube	three orthographically projected LDIs	not described	McMillan's algorithm	ray tracing

3.1.5 Image Based Rendering Methods

This section summarizes techniques that show all the properties making up the *image based rendering paradigm* as described in section 3.1.1. Unlike the image warping methods presented in section 3.1.3, which use images as intermediate representations of the scene restricted to certain objects or certain viewpoints, scenes are completely modeled by images in these techniques. This resembles the methods described in section 3.1.4, but in contrast to those *object centered* approaches, the rendering systems described below are *image centered*. Moreover, they don't rely at all or not as much on geometrical information as the methods from section 3.1.4. As a consequence, they provide all the advantages of the image based rendering paradigm. Most important, the rendering cost is truly independent of scene complexity and the acquisition of real scenes is achieved by sampling images, e.g. from a CCD-camera. On the other hand, they are basically restricted to static scenes with static illumination, though some papers make suggestions on how to work around this problem. A second problem appearing in certain techniques is that the movement of the viewpoint is confined to particular locations.

The *plenoptic function* of Adelson and Bergen [1] can be used to formulate a concise problem statement for these image based rendering systems. The plenoptic function is a parametrized function for describing everything that is visible from a certain point in space. It can be parametrized using seven parameters. Imagine an idealized eye which can be placed at any point in space (V_x,V_y,V_z). From there any of the viewable rays can be selected by choosing an azimuth and elevation angle (θ,ϕ) as well as a band of wavelengths λ, which should be considered. In the case of a dynamic scene, the time t can additionally be chosen, at which the function should be evaluated. This results in the following form for the plenoptic function:

$$p = P(\theta, \phi, \lambda, V_x, V_y, V_z, t) \tag{3.1}$$

Image based rendering systems can be completely described by specifying particular processes for sampling, reconstruction and resampling of the plenoptic function. Most methods

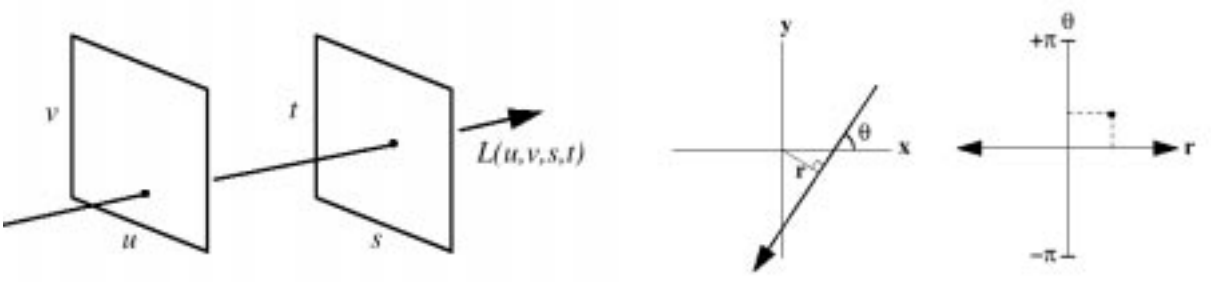
acquire samples only at a single point in time, which directly leads to the limitation to static scenes and illumination as mentioned above.

The *movie-map* system by Lippman [29] is an early approach to navigate in virtual environments. The streets of a city are filmed from a set of points with a distance of only a few feet between each other. Four cameras are used to shoot the views at every point, thereby giving the user the ability to pan to the left and the right. At playback time, two videodisc players are used to retrieve corresponding views to simulate the effect of walking. This system was restricted to redisplaying stored images and therefore provided only limited navigability and interaction. Lippman's plenoptic function reconstruction technique can thus be characterized as a nearest neighbor interpolation. When given a set of input parameters $(\theta, \phi, \lambda, V_x, V_y, V_z, t)$, the movie-map system can select the nearest partial sample (i.e. image). This can also be interpreted as a table based evaluation of the plenoptic function. Several other systems were developed (e.g. the "Virtual Museum" [38]), which required that every displayable view was created and stored in the authoring stage as well, thus having similar disadvantages.

Apple's *QuickTimeVR* system [9] uses 360-degree panoramic images to compose a virtual environment. The images are created with computer rendering, specialized panoramic cameras or by stitching together overlapping photographs taken with a regular camera. The stitcher uses a correlation based image registration algorithm to match and blend adjacent pictures, but no details are provided on that. Panoramic images can be regarded as cylindrical environment maps (c.f. [Greene:1986:AWP]). They describe a player for panoramic images that allows the user to perform continuous panning in the vertical and horizontal directions. Obviously, the panoramic images provide less than 180 degrees vertical field-of-view. The player performs continuous zooming as well. It uses a warping algorithm that is able to display perspectively correct planar projections from the cylindrical images at interactive frame rates, but this algorithm is not described in the paper. Walking in 3-D space can be accomplished only by hopping to different panoramic points. Notably, the authors focus on the overall design of the system rather than on the technical details.

The *plenoptic modeling* system [37] was presented by McMillan and Bishop at the same time as [9]. Moreover, it's a very similar concept that uses cylindrical projections to represent the samples of the plenoptic function as well. They describe the registration process in much more detail than [9]. The process uses several planar projected images taken from a common point of view and applies a sophisticated camera model and correlation methods to establish a mapping function between any pair of those images. The images can thus be reprojected onto arbitrary surfaces. Unlike [9], they make use of multiple cylindrical projections from different viewing positions to eventually compute stereo disparities between cylinder pairs. This is achieved by establishing a cylindrical epipolar geometry. Applying methods from computer vision, they get an additional disparity image for each cylinder, which implicitly stores depth information for each pixel and can be used in the rendering process to simulate parallax effects. The reconstruction algorithm is an extension to McMillan's warping algorithm [36] that handles cylindrical to planar projections. It integrates the disparity values into the rendering process to correctly simulate parallax effects. McMillan's list ordering approach [35] is used to resolve visibility. Not much information is given on the hole reconstruction problem. It seems that they use a splatting approach as mentioned in their related work [33] and introduced by Westover [58]. This involves computing the size of the projected kernel based on the current disparity value and the derivative along the epipolar curves. In order to avoid occlusion artifacts, they warp multiple reference images as described in [33] and combine them on a pixel-by-pixel basis according to the smallest reconstruction kernel.

Levoy and Hanrahan introduced the *light field rendering* method in [28]. This is a simple and robust technique for generating new views of a scene from arbitrary camera positions without depth information or feature matching, but simply by combining and resampling the available images of the scene. The major idea behind the technique is a representation of the so-called light field, which describes the radiance as a function of position and direction. This definition is equivalent to the plenoptic function introduced by Adelson and Bergen [1]. Note that in regions of space free of occluders (free space), this is a 4-D function since the radiance doesn't change along a ray in free space. 4-D light fields may be interpreted as functions on the space of oriented lines. The crucial issue in choosing a representation of the 4-D light field is how to parametrize the space of oriented lines such that it provides favorable properties. They propose to parametrize lines by their intersection with two arbitrary planes and call this configuration a *light slab* (fig. 3.13 (a)). By convention, the coordinate system on the first plane is *(u,v)* and on the second plane is *(s,t)*. They argue that this representation provides efficient geometric calculations. Most important, the inverse mapping of an image pixel to the parameters of the corresponding line in the light slab is a projective map. Another issue related to the parametrization is the sampling pattern. Assuming that all views are equally likely to be generated, then any line is equally likely to be needed. In the so called *line space*, each line is represented as a point (fig. 3.13 (b)). Thus all regions of line space should have an equal density of samples.



(a) (b)

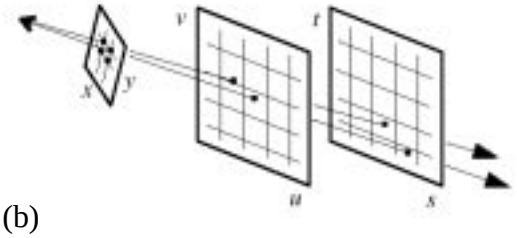
Figure 3.13: (a) The light slab representation. (b) Definition of the line space. Each oriented line in Cartesian space is represented by a point in line space. (Figures from [28])

Light fields can easily be generated by assembling a collection of images. For synthetic scenes, a 2-D array of images is rendered, each representing a slice of the 4-D light slab. The center of projection of each image is placed at a sample location on the *uv*-plane and the perspective projection is sheared such that the *xy* samples of each image exactly lie on the sampling locations on the *st*-plane. They shortly discuss the aliasing problem coming with this sampling process and suggest a prefiltering method that has to be applied in 4-D line space. A method for acquiring light fields from real scenes is presented as well. They built a special computer-controlled camera gantry that positions the center of projection of the camera at the sampling positions on the *uv*-plane. The camera is panned and tilted to point to the center of the object. The acquired image is reprojected to the *st*-plane using standard texture mapping. Light field arrays are very large (fig. 3.14 (a)) and they have to be compressed in order to make creation, transmission and display practical. Fortunately, they contain a high degree of redundancy in all four dimensions and thus can be highly compressed. The compression process can be asymmetrical, since compression is done in a preprocess and decompression has to be achieved in real time. Moreover, decompression has to be possible with random access, since the set of samples that have to be accessed to extract an image is dispersed in memory. They developed a compression pipeline consisting of two stages: the first step is a vector quantization followed by an entropy coding step (they simply use gzip) resulting in an overall compression ratio of about 120:1.

Decompression is performed by applying these two steps in reversed order. First, gzip decoding is performed as the file is loaded into memory. Vector dequantization is then applied on the fly as samples addressed by *(u,v,s,t)* coordinate tuples are requested by the display engine. The display engine extracts an image from the light slab given the image geometry by resampling a 2-D slice of lines from the 4-D light field. Each line represents a ray through the eye point and a pixel center. Once the *(u,v,s,t)* parameters of a line are calculated, the radiance has to be resampled at those line parameters (fig. 3.14 (b)). A big advantage of the light slab representation is that this inverse transformation from *(x,y)* to *(u,v,s,t)* reduces essentially to two texture coordinate calculations per ray, which is an operation accelerated by most of today's graphics hardware. They tried out different interpolation techniques for resampling the radiance such as nearest neighbor interpolation, linear interpolation in *uv* only or quadrilinear interpolation in *uvst*. Quadrilinear interpolation coupled with the proper prefiltering generates images with fewer artifacts than the other techniques. However, it is obviously more expensive. A major limitation of the light field rendering approach is that the observer is restricted to regions of space free from occluders. It is proposed to address this by stitching together multiple light fields based on a partition of the scene geometry.

	buddha	kidney	hallway	lion
Number of slabs	1	1	4	4
Images per slab	16x16	64x64	64x32	32x16
Total images	256	4096	8192	2048
Pixels per image	256^2	128^2	256^2	256^2
Raw size (MB)	50	201	1608	402
Prefiltering	uvst	st only	uvst	st only

(a)



(b)

Figure 3.14: (a) Typical light slab configurations and storage requirements. (b) The resampling process. (Figures from [28])

At the same time as [28], the *Lumigraph* system [18] was published by Gortler et al. This approach is very similar to light field rendering. They propose the same parametrization of the space of oriented lines in 3-D space free of occluders using the intersection points on two parallel planes. They use the ray coordinates for an illustrative analysis of the parametrization, which is equivalent to the so-called ray space of [28]. The theoretical aspects of the discretization step is discussed in more detail than in [28]. They describe the general problem of projecting a continuous multidimensional function to a finite dimensional function space. It is mentioned that this projection can be interpreted as point sampling the continuous function after it has been low pass filtered with the dual of the reconstruction kernel. Unlike Levoy's work, which doesn't rely on any kind of geometric information about the scene, they developed a method for integrating depth information into the discrete Lumigraph representation. Given a depth value at which a ray intersects a surface of the object, the corresponding basis functions can be reshaped to match the continuous Lumigraph function more closely. This idea of shaping the support of the basis functions to match the structure of the function being approximated is well known from finite element methods, e.g. from the radiosity method. In the Lumigraph system, depth corrected quadrilinear basis functions are used. They describe the acquisition of a Lumigraph representation from synthetic and real scenes. For synthetic scenes, they apply the same technique as [28]. Whereas Levoy et al. developed a computer-controlled camera gantry to acquire real scenes, the Lumigraph system uses a regular hand-held camera. To achieve this goal, the camera has first to be calibrated to determine the mapping between directions and image coordinates. Next, the pose of the camera has to be computed. Both problems have been

subject to extensive research in the area of computer vision. Gortler et al. use an algorithm originally developed by Tsai [56] and extended by Willson [60]. The approximate shape of a real object needed to perform the depth correction mentioned above is computed using the octree construction algorithm [53]. When acquiring a Lumigraph, each pixel in an input image represents a single sample of the Lumigraph function. Since a hand-held camera is used, the sample points in the domain cannot be pre-specified or controlled. In addition, there is no guarantee that incoming samples are evenly spaced. Constructing a Lumigraph from these samples is similar to the problem of multidimensional scattered data approximation. But the distribution of the data samples have two qualities complicating the process: first, sampling density can be quite sparse with large gaps, and second, sampling density is typically very non-uniform. They developed an algorithm combining concepts of a hierarchical polynomial fit filter by Burt [7] and a method proposed by Mitchell [40] that is able to handle non-uniformly distributed samples. Their novel algorithm proceeds in three phases. The first is called *splat*, whereas the sample data is used as a first approximation of the discrete Lumigraph representation. Second, the *pull* phase computes coefficients for basis functions at a hierarchical set of lower resolution grids by combining coefficients from higher resolution grids. The third phase is called *push*. It combines information from each lower resolution grid with the next higher resolution grid filling the gaps and avoiding excessive blurring. The reconstruction process described by Gortler et al. is again very similar to the one proposed by Levoy and Hanrahan. Texture mapping hardware is used to accelerate the process. Unlike Levoy and Hanrahan, the compression problem is discussed only shortly and no implementation is described. With a resolution of 32x32 on the *(s,t)* and 256x256 on the *(u,v)* plane and six pairs of parallel planes building up a complete viewing cube, an uncompressed Lumigraph typically requires 1.125 GB of storage. They state that preliminary experiments suggest that a 200:1 compression ratio reducing the storage requirements to under 6MB could be achieved with almost no degradation in image quality, which seems to be consistent with the results from [28].

None of the two techniques summarized above is able to display dynamic scenes or varying lighting conditions. The second problem is addressed by [62]. Wong et al. describe an image-based rendering system with controllable illumination, which is based on the light field and Lumigraph systems. Following the terminology of Levoy and Hanrahan, they extend the 4-D light field by adding two more dimensions that represent the bidirectional reflectance distribution function (BRDF) for each 4-D ray stored in the light field. They call this an *apparent* BRDF since it can be interpreted as the BRDF of a pixel on the *st*-plane that captures the aggregate reflectance of objects visible through that pixel window. Whereas the light vector from a light source defines one direction of the BRDF, the viewing direction is implicitly defined by the coordinates of the corresponding ray in the light field. They argue that the rendering and storage cost of this approach doesn't depend on the scene complexity, the technique avoids aliasing problems and can be easily integrated in the light slab data structure. It is pointed out that the apparent BRDF represents the response in the presence of the rest of the scene, not merely a surface reflectivity. Consequently, if one works with views that include shadows, they appear in the reconstruction as well. The BRDF is measured by capturing images of synthetic or real scenes under different illumination conditions. A directional light source is cast on the scene from as many as 400 different directions and the fraction of the reflected light on each sampling ray is measured. Only a partial BRDF has to be stored for each ray in the light field, which is parametrized by two angles for the direction of the incoming light. Thus, for each ray the partial BRDF can be approximated using spherical harmonics. With this representation, 16 to 25 coefficients were found to be sufficient to approximate the BRDF. It thus requires far less storage than a table with entries for each BRDF sample. The reconstruction process computes the intensity for each ray in the light field by summing up the weighted contribution from multiple light

sources. Directional light sources can be handled easily. For point light sources, additional depth information is needed. It is possible to apply light sources from arbitrary directions and with different intensities. The main disadvantage of this approach is the huge storage requirement, which is a multiple of the basic light field or Lumigraph representation.

An overview of the image based rendering systems summarized in this subsection is shown in table 3.4.

Table 3.4: Overview of image based rendering systems

Method	Parametrization of Plenoptic Function by Ray Intersection of ...	Attributes Available per Sample	Reconstruction
Perspective Warping	point and plane	color, optionally depth	plane to plane warping (optionally with depth)
Movie Maps	point and plane, multiple images	color	image indexing
QuickTimeVR	point and cylinder, multiple images	color	cylinder to plane warping
Plenoptic Modeling	point and cylinder, multiple images	color, disparity value	cylinder to plane warping with disparity
Light Field Rendering	plane and plane (light slab), 1 to 6 slabs	color	inverse mapping, resampling with quadrilinear filtering
Lumigraph	plane and plane, 1 to 6 pairs of planes	color, depth value	inverse mapping with depth correction, resampling with quadrilinear filtering
Light Field Rendering with Controllable Illumination	plane and plane (light slab), 1 to 6 slabs	color, partial BRDF	inverse mapping, resampling with quadrilinear filtering, BRDF reconstruction

4

Conclusion

In this report, a survey and classification of real time rendering methods is presented. To this aim, the rendering process is divided into four conceptual components, namely scene description, discretization, representation and rendering. The geometry based and the image based rendering paradigm are defined by specifying a characteristic property corresponding to each of the four steps. It is shown, that almost all rendering methods make use of properties associated with either of the two paradigms. In fact, the methods are classified by determining for each of the four conceptual concepts, whether the geometry based or the image based characteristic is used. The rendering methods are summarized in four groups, starting with geometry based rendering, then image warping techniques, followed by methods relying on a point sample representation and finally image based rendering methods. For each group, the most important contributions are described, compared and the results collected in a table for overview.

The focus of this report is to give a survey of existing, state-of-the art rendering techniques. The goal of the ongoing research will be to develop algorithms for efficient rendering of 3-D scenes and suitable object representations. The method envisioned should provide a better image quality to computation cost ratio than previous techniques. Moreover, it should facilitate multi-resolution representation of objects as well as object deformation.

A

Literature

[1] E. H. Adelson and J. R. Bergen. “The Plenoptic Function and Elements of Early Vision.” *Computational Models of Visual Processing*, pages 3–20, 1991.

[2] Animatec. “Caviar Technology.” http://www.animatec.com.

[3] J. F. Blinn. “Models of Light Reflection For Computer Synthesized Pictures.” volume 11, pages 192–198, July 1977.

[4] J. F. Blinn. *Computer display of curved surfaces*. Ph.d. thesis, University of Utah, 1978.

[5] J. F. Blinn. “Simulation of Wrinkled Surfaces.” In *Computer Graphics (SIGGRAPH ’78 Proceedings)*, volume 12, pages 286–292, August 1978.

[6] J. F. Blinn and M. E. Newell. “Texture and Reflection in Computer Generated Images.” *Communications of the ACM*, 19:542–546, 1976.

[7] P. J. Burt. “Moment images, polynomial fit filters, and the problem of surface interpolation.” In *Proceedings of Computer Vision and Pattern Recognition*, pages 29–38. IEEE Computer Society Press, june 1988.

[8] E. E. Catmull. “Computer Display of Curved Surfaces.” In *Proceedings of the IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structure*, pages 11–17, May 1975.

[9] S. E. Chen. “Quicktime VR - An Image-Based Approach to Virtual Environment Navigation.” In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 29–38. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[10] S. E. Chen and L. Williams. “View Interpolation for Image Synthesis.” In J. T. Kajiya, editor, *Computer Graphics (SIGGRAPH ’93 Proceedings)*, volume 27, pages 279–288, August 1993.

[11] R. L. Cook. “Shade trees.” In H. Christiansen, editor, *Computer Graphics (SIGGRAPH ’84 Proceedings)*, volume 18, pages 223–231, July 1984.

[12] F. C. Crow. “Summed-area Tables for Texture Mapping.” In H. Christiansen, editor, *Computer Graphics (SIGGRAPH ’84 Proceedings)*, volume 18, pages 207–212, July 1984.

[13] W. Dally, L. McMillan, G. Bishop, and H. Fuchs. “The Delta Tree: An Object-Centered Approach to Image-Based Rendering.” AI Memo 1604, AI Lab, Massachusetts Institute of Technology, 1996.

[14] W. Dungan, Jr., A. Stenger, and G. Sutty. “Texture Tile Considerations for Raster Graphics.” In *Computer Graphics (SIGGRAPH ’78 Proceedings)*, volume 12, pages 130–134, August 1978.

[15] E. A. Feibush, M. Levoy, and R. L. Cook. “Synthetic Texturing Using Digital Filters.” volume 14, pages 294–301, July 1980.

[16] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes. *Computer Graphics, Principles and Practice, Second Edition*. Addison-Wesley, Reading, Massachusetts, 1990. Overview of research to date.

[17] G. Y. Gardner. “Visual Simulation of Clouds.” In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH ’85 Proceedings)*, volume 19, pages 297–303, July 1985.

[18] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. “The Lumigraph.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 43–54. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[19] H. Gouraud. “Continuous Shading of Curved Surfaces.” *IEEE Transactions on Computers*, C-20(6):623–629, June 1971.

[20] N. Greene. “Applications of world projections.” In M. Green, editor, *Proceedings of Graphics Interface ’86*, pages 108–114, May 1986.

[21] N. Greene and P. S. Heckbert. “Creating Raster Omnimax Images From Multiple Perspective Views Using the Elliptical Weighted Average Filter.” *IEEE Computer Graphics and Applications*, 6(6):21–27, June 1986.

[22] J. Grossman. “Point Sample Rendering.” Master’s thesis, Massachusetts Institute of Technology, 1998.

[23] J. P. Grossman and W. J. Dally. “Point Sample Rendering.” In N. M. G. Drettakis, editor, *Rendering Techniques ’98, Proceedings of the Eurographics Workshop in Vienna, Austria, June 29-July 1, 1998*, pages 181–192. Eurographics, Springer, July 1998.

[24] P. S. Heckbert. “Survey of Texture Mapping.” *IEEE Computer Graphics and Applications*, 6(11):56–67, November 1986.

[25] P. S. Heckbert. “Fundamentals of Texture Mapping and Image Warping.” M.sc. thesis, Department of Electrical Engineering and Computer Science, University of California, Berkeley, June 1989.

[26] D. S. Kay and D. P. Greenberg. “Transparency for Computer Synthesized Images.” In *Computer Graphics (SIGGRAPH ’79 Proceedings)*, volume 13, pages 158–164, August 1979.

[27] P. Lacroute and M. Levoy. “Fast Volume Rendering Using a Shear–Warp Factorization of the Viewing Transformation.” In A. Glassner, editor, *Proceedings of SIGGRAPH ’94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 451–458. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[28] M. Levoy and P. Hanrahan. “Light Field Rendering.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 31–42. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[29] A. Lippman. “Movie-Maps: an Application of the Optical Videodisc to Computer Graphics.” *Computer Graphics*, 14(3):32–42, July 1980.

[30] D. Lischinski and A. Rappoport. “Image-Based Rendering for Non-Diffuse Synthetic Scenes.” In G. Drettakis and N. Max, editors, *Eurographics Workshop on Rendering 1998*, pages 301–314. Eurographics, Springer Wien, 1998. held in New Orleans, Louisiana, 04-09 August 1996.

[31] P. W. C. Maciel and P. Shirley. “Visual Navigation of Large Environments Using Textured Clusters.” In P. Hanrahan and J. Winget, editors, *1995 Symposium on Interactive 3D Graphics*, pages 95–102. ACM SIGGRAPH, April 1995. ISBN 0-89791-736-7.

[32] T. W. Marc Levoy. “The Use of Points as a Display Primitive.” Technical report, University of North Carolina at Chapel Hill, 1985.

[33] W. R. Mark, L. McMillan, and G. Bishop. “Post-Rendering 3D Warping.” In M. Cohen and D. Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 7–16. ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3.

[34] L. McMillan. “Computing Visibility Without Depth.” Technical report, University of North Carolina, 1995.

[35] L. McMillan. “A List-Priority Rendering Algorithm for Redisplaying Projected Surfaces.” Technical report, University of North Carolina, 1995.

[36] L. McMillan. “Shape as Perturbation to Projective Mapping.” Technical Report TR95-046, University of North Carolina, 1995.

[37] L. McMillan and G. Bishop. “Plenoptic Modeling: An Image-Based Rendering System.” In R. Cook, editor, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 39–46. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[38] G. Miller. “The Virtual Museum: Interactive 3D Navigation of a Multimedia Database.” *The Journal of Visualization and Computer*, (3):183–197, 1992.

[39] G. S. Miller and C. R. Hoffman. “Illumination and Reflection Maps: Simulated Objects in Simulated and Real Environments.” In *SIGGRAPH ’84 Advanced Computer Graphics Animation seminar notes*. July 1984.

[40] D. P. Mitchell. “Generating Antialiased Images at Low Sampling Densities.” In M. C. Stone, editor, *Computer Graphics (SIGGRAPH ’87 Proceedings)*, volume 21, pages 65–72, July 1987.

[41] S. Molnar. *Image Composition Architectures for Real-Time Image Generation*. PhD thesis, University of North Carolina, 1991.

[42] C. Oosterbaan. “Motion and Deformation of Surfel Objects.” Master’s thesis, Delft University of Technology and MERL, 1998.

[43] B.-T. Phong. “Illumination for Computer Generated Pictures.” *Communications of the ACM*, 18(6):311–317, June 1975.

[44] M. Regan and R. Pose. “Priority Rendering with a Virtual Reality Address Recalculation Pipeline.” In A. Glassner, editor, *Proceedings of SIGGRAPH ’94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 155–162. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[45] D. F. Rogers. *Procedural Elements for Computer Graphics*. McGraw-Hill, 1985.

[46] G. Schaufler. “Dynamically Generated Impostors.” In D. W. Fellner, editor, *Modeling - Virtual Worlds - Distributed Graphics*, MVD’95 Workshop, pages 129–136, November 1995.

[47] G. Schaufler. “Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes.” In J. Dorsey and P. Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 151–162, New York City, NY, June 1997. Eurographics, Springer Wien. ISBN 3-211-83001-4.

[48] G. Schaufler. “Per-Object Image Warping with Layered Impostors.” In N. M. G. Drettakis, editor, *Rendering Techniques ’98, Proceedings of the Eurographics Workshop in Vienna, Austria, June 29-July 1, 1998*, pages 145–156. Eurographics, Springer, July 1998.

[49] G. Schaufler and W. Stürzlinger. “A Three Dimensional Image Cache for Virtual Reality.” *Computer Graphics Forum*, 15(3):227–236, August 1996. Proceedings of Eurographics ’96. ISSN 1067-7055.

[50] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. “Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 75–82. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[51] J. W. Shade, S. J. Gortler, L. He, and R. Szeliski. “Layered Depth Images.” In M. Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, Annual Conference Series, pages 231–242. ACM SIGGRAPH, Addison Wesley, July 1998. ISBN 0-89791-999-8.

[52] F. Sillion, G. Drettakis, and B. Bodelet. “Efficient Impostor Manipulation for Real-Time Visualization of Urban Scenery.” *Computer Graphics Forum*, 16(3):207–218, August 1997. Proceedings of Eurographics ’97. ISSN 1067-7055.

[53] R. Szeliski. “Rapid Octree Construction from Image Sequences.” *CVGIP: Image Understanding*, 58(1):23–32, July 1993.

[54] J. Torborg and J. Kajiya. “Talisman: Commodity Real-time 3D Graphics for the PC.” In H. Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 353–364. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[55] K. E. Torrance and E. M. Sparrow. “Polarization Directional Distribution And Off-specular Peak Phenomena in Light Reflected from Roughened Surfaces.” *J. Opt. Soc. Am.*, 7(56):916–925, 1966.

[56] R. Y. Tsai. “A Versatile Camera Calibration Technique for High-Accuracy 3D Machine Vision Metrology Using Off-the-shelf TV Cameras and Lenses.” *IEEE Journal on Robotics and Automation*, 4:323–344, August 1987.

[57] J. van Baar. “Generation and Rendering of Surface Elements.” Master’s thesis, Delft University of Technology and MERL, 1998.

[58] L. Westover. “Footprint Evaluation for Volume Rendering.” In F. Baskett, editor, *Computer Graphics (SIGGRAPH ’90 Proceedings)*, volume 24, pages 367–376, August 1990.

[59] L. Williams. “Pyramidal Parametrics.” In *Computer Graphics (SIGGRAPH ’83 Proceedings)*, volume 17, pages 1–11, July 1983.

[60] R. G. Willson. *Modeling and Calibration of Automated Zoom Lenses*. PhD thesis, Carnegie Mellon University, 1994.

[61] G. Wolberg. *Digital Image Warping*. IEEE Computer Society Press, Los Alamitos, California, 1990.

[62] T. Wong, P. Heng, S. Or, and W. Ng. “Image-based Rendering with Controllable Illumination.” In J. Dorsey and P. Slusallek, editors, *Eurographics Rendering Workshop 1997*, pages 13–22, New York City, NY, June 1997. Eurographics, Springer Wien. ISBN 3-211-83001-4.

[63] H. Zhang and K. E. Hoff III. “Fast Backface Culling Using Normal Masks.” In M. Cohen and D. Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 103–106. ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3.