Image-based Representations for Accelerated Rendering of Complex Scenes

Stefan Jeschke[†] and Michael Wimmer and Werner Purgathofer

Institute of Computer Graphics and Algorithms, Vienna University of Technology, Austria

Abstract

This paper gives an overview of image-based representations commonly used for reducing the geometric complexity of a scene description in order to accelerate the rendering process. Several different types of representations and ways for using them have been presented, which are classified and discussed here. Furthermore, the overview includes techniques for accelerating the rendering of static scenes or scenes with animations and/or dynamic lighting effects. The advantages and drawbacks of the different approaches are illuminated, and unsolved problems and roads for further research are shown.

Categories and Subject Descriptors (according to ACM CCS): [I.3.3] [Computer Graphics]: Picture/Image Generation - *Display Algorithms* [I.3.7] [Computer Graphics]: Three-Dimensional Graphics and Realism - *Color, shading, shadowing, and texture*

1. Introduction

The interactive visualization of three-dimensional models is a research area of big interest. Applications include computer games, flight and driving simulations, virtual reality scenarios, architectural visualization and computer aided design. During the exploration of a 3D model, a fluent animation is desired in order to allow a convincing immersion. One challenge for interactive rendering systems is to provide appropriate frame rates also for very complex 3D models. The performance of consumer-level hardware has increased dramatically in recent years, allowing the rendering of several Million primitives per second. Furthermore, graphics hardware has become more programmable, which allows better scene realism. On the other hand, due to the continuing desire for more detail and realism, the model complexity of common scenes has not reached its peak by far. Because of this fact, scenes are often too complex to be displayed at real-time or even interactive frame rates. The rendering acceleration of such scenes has been a hot topic in computer graphics in recent years and it seems like this is not going to change in the near future. Many algorithms for accelerating the rendering process have been proposed, according to one of the following strategies:

- The complexity of today's graphics drivers and hardware behavior often leaves space for *rendering pipeline optimization*. Removing so-called *rendering bottlenecks* might dramatically increase the performance.
- *Visibility calculations* remove invisible portions of a scene before they are sent to graphics hardware. While this provides dramatic rendering acceleration for some cases (for instance, indoor or urban scenes), the actually visible geometry may already overwhelm the hardware.
- *Geometric simplification techniques* take advantage of the fact that complex distant scene parts with small size on screen contribute only little to the output image. Consequently, representations with reduced geometric detail are used with increasing distance. This may dramatically reduce the complexity of the visible scene (terrains are good examples). Unfortunately, geometric simplification techniques are not applicable for arbitrary scene parts. For instance, if multiple objects should be merged during the simplification process, preserving all individual appearances (for instance, textural information) is not possible in an efficient way.

[†] {jeschke|wimmer|wp}@cg.tuwien.ac.at

For those cases where geometric simplification techniques cannot be applied, representations based on *appearance samples* (typically color values, acquired by rendering the scene part) have been successfully used. During rendering, such representations are displayed instead of the original (geometry-based) scene part. While this provides a similar visual output, the alternative representation can be rendered much faster, because its rendering time depends on its size on screen rather than on the geometric complexity of the original scene part. This allows a very fast display of objects with arbitrary geometric complexity. The advantage compared to geometry-based simplification techniques is that the generation process does not rely on any knowledge about the geometric structure of the original scene part. Because of this, sample-based representations can be applied for a much broader range of scenes.

There exist many types of sample-based representations, mainly characterized by the type and amount of geometric information. They are called either *image-based* or *point-based* representations, depending on the format the data is stored and processed. In literature, image-based representations for rendering acceleration are often referred as *impostors* [MS95]. Using images applied to simple geometric models as texture maps (called *billboards* or *sprites* [MH02]) for fast display of complex objects is an old trick in computer graphics, as can be observed for instance in the well known computer game 'Wolfenstein' by ID software from 1992. In many computer games, billboards have been used to model trees, grass, fire, monsters, items or static background images that show hills or city skylines. However, in this paper we concentrate on *temporary replacements* for geometric scene parts rather than on entirely sample-based models. Furthermore, we concentrate on image-based representations, because point-based rendering is a separate research field which would go beyond the scope of this paper.

The remainder of this paper is organized as follows: first, Section 2 describes basic demands and characteristics of image-based representations. Section 3 gives an overview and characterization of existing representation types. The last part (Section 4) focusses on the question when to render an impostor vs. an original (geometric) object in order to accelerate the rendering process.

2. Basic Characteristics and Challenges

In order to generate an image-based representation from an object, its appearance as well as a rough geometric structure has to be acquired. In many cases, this is done by rendering the object from a single viewpoint (called *reference viewpoint*). The acquired information is combined with geometry in order to place the image into the scene. In the simplest case, the scene part is rendered into a texture and this texture is combined with a quadrilateral. The generation process can either happen in a preprocess (a so-called *static* representation) or dynamically at runtime (called a *dynamic* representation).

2.1. Image Quality Issues

For a convincing representation, the original scene part should be represented as correctly as possible. Figure 1 shows examples for image artifacts that can occur during display. The following particular aspects have to be considered

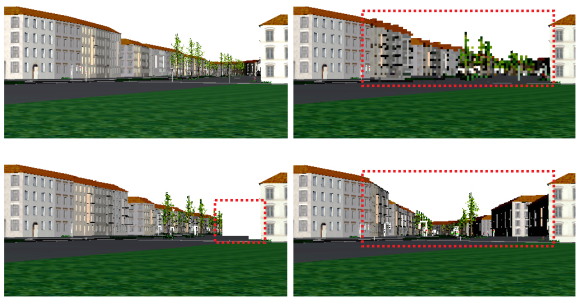
Figure 1: *Examples for image artifacts caused by image-based representations. Top-left: original scene part. Top-right: undersampling visible as "blocky" texels. Bottom-left: image gap (the representation was generated to the right of the actual viewpoint). Bottom-right: parallax errors result in wrong perspective and scene integration (the houses on the right should be invisible).*

in order to avoid such artifacts:

- The sampling resolution should not fall below the output image resolution. Otherwise, "blocky" pixels or holes become visible and the illusion of displaying a geometric object is destroyed. This means that a minimum distance to the representation must always be maintained. Furthermore, the acquisition of the appearance should provide a similar result compared to the originally rendered object.
- If the viewpoint is changed, parallax effects (kinetic depth effects) occur as nearby scene parts seem to move compared to distant scene parts. This effect can be supported by applying appropriate geometry, depending on a particular representation technique (see Section 3). Note that visible parallax effects grow with increasing distance to the reference viewpoint, which may limit the *viewing region* an impostor can be displayed for with sufficient accuracy.
- Parallax movements cause new scene parts to appear. This effect is called *disocclusion*. Of course, it is highly desirable that *all* scene parts that may become visible are included in the impostor. Otherwise, artifacts visible as so-called *image gaps* or *rubber-sheet effects* occur. The avoidance of these artifacts is not simple and most existing algorithms addressing this problem are computationally very costly.

- For a seamless integration into a scene, the border between image-based representations and adjacent geometry should not show any artifacts. This is especially important if objects are partially represented by images and partially by the original geometry in the same output image. Furthermore, the visibility between the original scene and the image-based representations should be resolved.
- Scene dynamics have to be faithfully reproduced by the image-based representation. This includes dynamic models like humans([ABT00]) and dynamic lighting effects([DDSD03]).

2.2. Limited Impostor Validity

The points stated in Section 2.1 have as a consequence that the viewing region (called *view cell*) a particular impostor can be displayed for is bounded: as long as the visual difference to the (hypothetically rendered) original object does not exceed some threshold, the representation is called *valid*. This threshold is defined depending on the impostor technique, as will be described in Section 3. The size and shape of a view cell depends on how a particular representation type addresses the image quality issues described in Section 2.1. *Box-shaped* view cells have been widely used in previous work. View cells are often defined implicitly as the level sets of specific error metrics. If the error for a viewpoint exceeds a certain threshold, the impostor is considered invalid. Frequently, such implicit view cells are *shaft shaped*([Jak00, ABT99, SS96, SLS*96]). Figure 2 shows an example for a shaft in 2D. The shaft apex lies in the center of the represented scene part. It is defined by a direction, an apex angle and a minimum distance to the object (see Figure 2). Shafts are consistent with the fact that the introduced errors are much less apparent when increasing the view distance. Consequently, the impostor can be displayed for a much larger view space region compared to box-shaped view cells. Furthermore, there are *view independent* representations [DDSD03], where the validity is only restricted by a minimum distance to the impostor. This can also be seen as a 360° shaft.

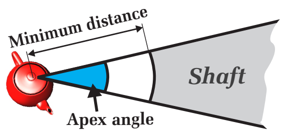

Figure 2: *A shaft-shaped view cell.*

If an impostor is no longer valid, another representation (impostor or original object) has to be displayed. This may result in more or less noticeable *popping artifacts*, depending on how well the representation resembles the original scene parts at the moment of switching. Another consequence of the limited validity is that for typical applications, numerous representations are required to provide sufficient rendering acceleration for every view in a scene. Depending on whether static or dynamic approaches are used, this may result either in high memory requirements or frequent updates during runtime. These are two fundamental issues for an efficient usage of image-based representations.

2.2.1. High Memory Requirements for Static Impostors

For the static approach, each possible impostor needs to be precalculated and stored. An impostor requires a relatively large amount of memory. This is because a large amount of appearance samples must be typically stored. In many cases, all impostors together do not fit into graphics memory or even main memory. Thus the images must be successively loaded from harddisk into main memory and into graphics memory. This operation must be finished for every impostor before it is needed for display. So-called *prefetching strategies* try to load the most probably needed samples in advance, often using predictions about future user movement. However, in case of sudden viewpoint changes, limited memory bandwidth might lead to missing samples at display time. Restricting the user velocity or temporally decreasing the image quality are solutions to this problem, but they are not always tolerable. Furthermore, the widely varying memory bandwidths of different hardware limit the portability of such a rendering system.

Another related problem is the preprocessing time needed to generate the images. In recent work [AL99, WM03], several hours of preprocessing are reported even for moderately sized scenes. This makes such approaches not useful for applications the impostors should be generated quickly, for instance, when loading a level of a computer game.

2.2.2. Frequent Dynamic Impostor Updates

For dynamically generated impostors [Sch95], the rendering resources are shared for model visualization and representation generation. An important point is that the rendering system should not be overwhelmed by this generation process. Therefore, impostor techniques with short generation times are desired. However, because such impostors are typically not valid very long, this in turn causes a frequent need for updates, which decreases the efficiency of this approach. In case of sudden viewpoint changes, too many images might have to be updated, resulting in a frame rate drop. Again, restricting the user velocity or decreasing the image quality in order to avoid such cases is not always an option. This fact makes the efficiency of dynamic techniques highly sensitive to the target application and to the performance of the target rendering system.

The issues above lead to similar demands for static and dynamic impostors: a large viewing region for which every impostor is valid, fast generation, and (especially for static images) low memory requirements. Furthermore, a high image quality without artifacts is desirable. Avoiding gaps due to disocclusions is a special challenge in this connection. Today, there is still no technique that provides all these features at the same time. Instead, a tradeoff must be found between the contrary demands. Many approaches that use image-based representations in order to accelerate the rendering of typical scenes often accept a loss of image quality to keep the amount of memory or the impostor update rate to a tolerable level.

3. Impostor Types

Several types of impostors have been presented in literature, each with the emphasis on different demands, for example fast generation and display, low memory requirements and/or good image quality for a large view cell (see Section 2.1). Especially the problem of maintaining a high image quality has been addressed by combining the images with different amounts of geometry. The following subsections describe the individual techniques in more detail.

3.1. Planar Impostors

In 1995, Maciel and Shirley [MS95] described so-called *planar impostors*. They basically consist of simple planar geometry (for instance, a quad) with an alpha texture applied to it, thus forming a billboard (see Figure 3). The generation

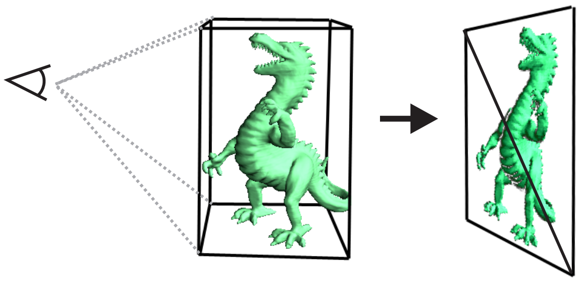

Figure 3: *A planar impostor is an alpha texture acquired by rendering the original object (left). The texture is applied to a simple planar geometry (right). In this example the complexity decreased from 108,526 to only 2 polygons.*

of a planar impostor is not computationally expensive. Displaying it is also very fast because of its very low geometric complexity. This makes this technique ideal for dynamic impostors.

On the other hand, parallax movements are not accounted for, so that the impostor needs to be updated frequently. In order to quantify the error introduced if the viewpoint is moved, Schaufler [Sch95] presents a fundamental criterion for the validity of planar impostors: as long as the so-called *error angle* between a point of the original object and its impostor representation falls below the angle of one pixel in the output image, the impostor is valid. Aliaga [Ali96] (and also Shade et al. [SLS*96]) mention that the error angle can be computed for every texel. However, this operation is usually much too costly. Schaufler [Sch95] instead estimates the error angle for the cases that the viewer moves *sidewards* and *towards* the object by using a specific oriented bounding box of the object. Shade et al. [SLS*96] basically use the same error metric, but instead of a special oriented bounding cube, they simply use the bounding box of the object. This is not conservative, but was reported to be sufficiently correct in practice. In further related work [ABT99, Jak00, Ali96, SLS*96], a simpler error estimation was presented, based on the angle the observer views an impostor in relation to the view position the impostor was generated. Furthermore, Aubel [ABT00] presented a special-purpose metric designed for impostors for animated humans, which is based on distance changes between different body parts.

In order to reduce discontinuities on the border between geometry and impostor, Aliaga [AL98] proposes to distort the geometry the same way the impostor does. Furthermore, for reducing popping artifacts, Aliaga proposes smoothly warping the represented scene part again. Although the output image is by no means correct, the impostors are reported to fit better into the scene. Ebbesmeyer [Ebb98] and Jakulin [Jak00] achieve this by alpha-blending between different representations.

Another point is the *visibility* between the scene and a flat impostor, which must be solved satisfactorily. Therefore, Schaufler [Sch97] introduced the so-called *nailboards*, which store per-pixel depth information in order to calculate the visibility with a two-pass rendering algorithm. Note that nailboards use depth information only for resolving visibility, while the impostor itself remains flat. Aubel et al. [ABT99] instead split the object into multiple small non-overlapping impostors. However, this leads to disturbing discontinuities at the impostor borders [ABT00]. Therefore, they propose storing all overlapping scene parts into one single impostor (they call it *factorized impostor*), at the cost of having to update the impostor frequently. As a second solution, they propose to solve visibility when the impostor is generated and only store visible pixels (they call it *depth corrected impostors*). However, this means that if any occluding scene part or the camera moves, the impostor has to be updated. Harris and Lastra [HL01] render clouds as impostors generated using particle systems. For resolving visibility between a cloud and object, they split it into a foreground and a background part and render an impostor for each of them. Figure 4 shows an image of their rendering system. In contrast to Aubel et al. [ABT00] (who rendered solid objects), this method works sufficiently correct, due to the fuzzy appearance of clouds.

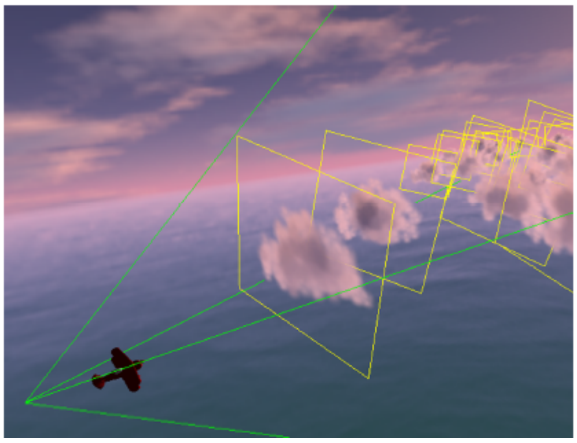

Figure 4: *Planar impostors used to represent clouds[HL01].*

3.2. Layered Impostors

The idea of using multiple texture layers (also called a *texture stack* or *shell textures*) as a general representation for complex geometry was presented by Meyer and Neyret [MN98]. Closely related to that technique, Schaufler [Sch98a] presented a *layered impostor technique*, which allows simultaneously reproducing parallax movements, and solving the visibility satisfactorily, therewith integrating seamlessly into the scene. The basic idea is to use multiple image layers at different distances from the viewer, each representing a certain depth range. Schaufler shows how the parallax errors decrease with an increasing number of layers. In order to capture as many potentially visible scene parts as possible, Schaufler [Sch98b] uses a view frustum centered *behind* the object for the impostor generation process. In order to avoid image gaps between the layers, slightly overlapping depth regions are rendered into each layer. However, no real guarantee is given for avoiding image gaps. Compared to planar impostors consisting of only one image layer, layered impostors provide a larger valid viewing region. Jakulin [Jak00] shows that this technique can be used for trees with high image quality by alpha blending between different impostor representations.

Jeschke et al. [JWS02] extended the work of Schaufler in several ways: a special spacing of the layers together with a new layer recording method ensures that all parallax movements are properly reproduced and the impostor is guaranteed to contain all scene parts that might become visible. Furthermore, an efficient visibility algorithm removes invisible scene parts from the impostor, and the visible portions in every layer are encoded separately in order to avoid storing most transparent texels. Figure 5 shows an example for that impostor technique with 13 layers and 240 impostor polygons.

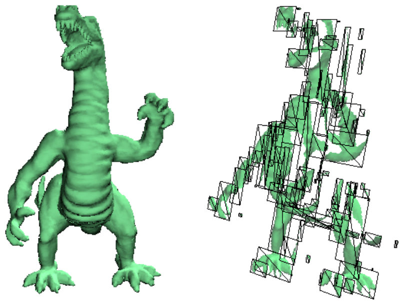

Figure 5: *Left: layered impostor from the viewer position. Right: side view with impostor polygons made visible.*

3.3. Billboard Clouds (BBC)

Decoret et al. [DDSD03] presented an impostor technique called *billboard clouds*. The basic principle is to represent an object by a set of alpha-textured planes (see Figure 6). In contrast to the techniques described above, the position and orientation of the planes is optimized for approximating the object geometry so that a given error tolerance in object space (or in image space [DDSD02]) can be met for arbitrary viewpoints. The advantage of this view independent impostor technique is that the viewer is not restricted to a bounded view cell, but only to a minimum distance to the impostor. Another advantage is that it can be used directly for standard shadow mapping algorithms. Relighting can also be applied to billboard clouds, as was shown by Decoret et al. [DDSD03]. The technique can be used for arbitrary geometry and shares concepts of image-based as well as geometry-based simplification. Dynamic lighting effects were obtained by computing online illumination with precomputed normal maps.

Current implementations of the technique show very long preprocessing times and image artifacts, especially on curved surfaces. Furthermore, high memory requirements as

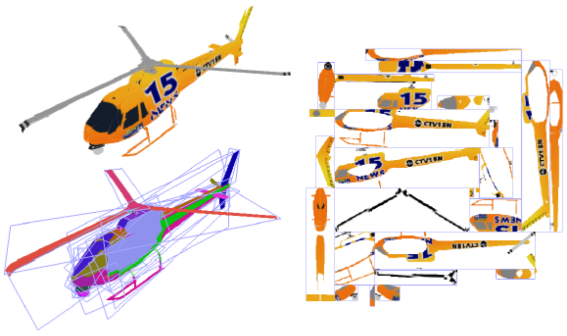

Figure 6: *A billboard cloud (top left) as a collection of 32 texture mapped quadrilaterals [DDSD03].*

well as relatively slow rendering are caused by large planes containing many transparent texels. Andujar et al. [ABC*04] presented an algorithm that greedily selects planes in watertight models, each covering as large a portion of the model as possible. However, the set of all planes is not guaranteed to be optimal. Recently, Meseth and Klein [MK04] introduced BTF textured billboard clouds, which increase the visual quality by preserving view and light dependent effects while still achieving fast rendering performance. In addition, they propose two new methods for billboard cloud generation, aimed at optimal texture memory usage.

3.4. Video-based Representations

Wilson et al. [WLY*00] store planar impostors of adjacent view cells in an MPEG2 video stream. The MPEG video compression standard provides relatively high compression rates for the representation. Although video encoding provides fairly good compression rates, resulting in low memory costs, high-frequency scene content causes aliasing artifacts and results in lower compression rates. Unfortunately, aliasing artifacts occur often when rendering distant scene parts. Another problem is that since video-based impostors actually constitute flat images, a correct visibility calculation is not possible for dynamic scene content.

One problem with the standard MPEG2 codec is that only one continuous stream of defined size can be processed and stored. If the viewer moves on a ground plane, a two dimensional access in the video stream is needed. Therefore, Wilson et al. [WMPM01] present a modified codec supporting this type of access.

3.5. Textured Depth Meshes (TDM)

The idea of *textured depth meshes* [DSV97, SDB97] is to triangulate a rendered image with respect to the z-buffer. Planar image regions are represented using individual textured polygons, whereas a tradeoff between approximation accuracy and mesh complexity must be found for non-planar regions. TDMs provide good parallax movement reconstruction, and visibility is also solved satisfactorily for many cases. They need only little additional geometry and storage compared to planar impostors. Figure 7 shows an example for a TDM.

Several methods have been proposed for TDM generation. Darsa et al. [DSV97] use a Delauney triangulation based on the Voronoi diagram of irregular sparse samples generated by a ray tracer. In that work, every triangle of the TDM is assumed to have only one color, so that color and depth must be approximated simultaneously. In later work, Darsa et al. [DC96, DCV98] obtain the triangulation only from the depth component of the samples. Color is applied to the depth mesh using textures. Sillion et al. [SDB97] extract silhouettes and depth discrepancy lines from the z-buffer, thus

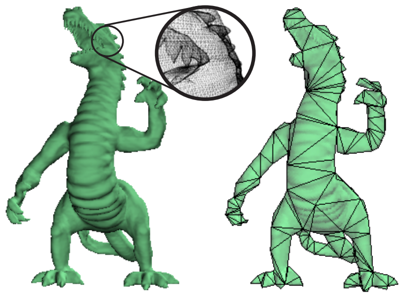

Figure 7: *Left: original model (108,526 polys). Right: textured depth mesh (177 polys).*

creating several image regions. For triangulating these regions, equally spaced points are inserted, followed by a constrained Delauney triangulation. Figure 8 shows a result of this triangulation.

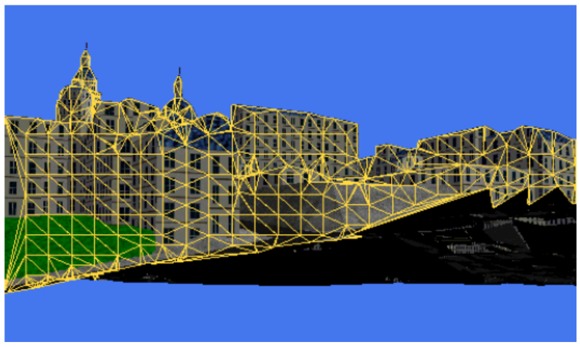

Figure 8: *A TDM for a city district [SDB97].*

Aliaga et al. [ACW*99] instead build a very dense regular mesh with image resolution, which is then simplified using a fast greedy algorithm that merges planar regions. Afterwards, an accurate but also computationally expensive mesh simplification algorithm [GH97] is applied, as was also used by Jeschke et al. [JW02]. Wilson and Manocha [WM03] first use the approach of Garland and Heckbert and afterwards the view dependent simplification algorithm of Luebke and Erikson [LE97].

In general, the simplification of a TDM is computationally costly, so that a generation at runtime seems hardly possible. In order to reduce the required storage space on harddisk, Decoret et al. [DSSD99] suggest generating the mesh geometry in a preprocess (because this is the most time-consuming operation) but the textural information on the fly. As a further interesting alternative, Chen et al. [CIKK99] rendered a high-quality image of a terrain scene at runtime, texture mapped it onto a simpler terrain geometry and reused this representation over several successive frames.

Many TDM methods suffer from disocclusion artifacts, typically visible as *rubber-sheet effects*. They are caused by distorted meshes that cover geometry which is not represented in the TDM. In order to avoid such artifacts, all potentially visible scene parts should be captured with sufficient sampling density. Therefore, Darsa et al. [DC96, DCV98] use a measurement for every triangle that indicates the quality of the sampling. If the quality is not sufficiently high, triangles from multiple TDMs generated near the actual viewer position are displayed. They proposed different mesh blending functions for optimizing the image quality with reasonable computational effort. Similarly, Wilson and Manocha [WM03] sample the geometry incrementally. This means that the visual error is estimated for a TDM, and the sampling process is repeated for a new viewpoint which is assumed to provide the missing information. This process stops if the sampling density is sufficiently high. Afterwards, redundant information is removed, and textured depth meshes are constructed from the sampled data. During runtime, multiple TDMs are rendered in order to minimize visual artifacts. Although the algorithm may provide satisfying image quality in many situations, no real guarantee is given that the error in image space is smaller than a certain value, and many TDMs have to be rendered for complex views.

Another method for avoiding disocclusion artifacts was presented by Decoret et al. [DSSD99], who generate multiple meshes for scene parts at different distances from the viewer. *Visibility events* are introduced in order to quantify depth discrepancy and thereby disocclusion artifacts. During the mesh generation process, objects are treated successively. Every object is inserted into an existing mesh if the depth discrepancy is low enough. If an object fits into multiple meshes, the one with the least wasted texture space is used. If the object fits into no mesh, a new mesh is generated. If rendering resources are available at runtime, the meshes are dynamically updated, which further increases the accuracy of the representation.

Jeschke et al. [JW02] generate the TDM based on their scene layering technique already mentioned in Section 3.2. Since all visible scene parts are acquired with proper sampling density, no image gaps or rubber sheet effects occur.

3.6. Per-Pixel Depth Information

Depth images (Shade et al. [SGHS98] called them *sprites with depth*) contain per-pixel depth information for displaying images for new viewpoints. The transformation for projecting samples to a new image is called *3D image warp* [McM97]. The warp can be performed in *forward* direction by projecting every sample into the output image. McMillan and Bishop [MB95] provide an efficient method for back-to-front ordered warping of a depth image, so that visibilitiy is solved correctly without the need of a z-buffer. In contrast, searching the correct representant for an output pixel in the input image is called *backward warping*. The latter can be seen as a kind of ray tracing and is useful if only few pixels have to be displayed because of the often costly search operations.

For avoiding holes in the output image in case of magnified viewing, depth images use techniques like point splatting [SGHS98] or the triangulation to fine meshes. In fact, a depth image can also be seen as a point cloud with a regular grid structure. While both representations are based on a set of appearance samples, the main difference is the format in which the samples are stored and processed. Recent research on point-based representations has focussed on high image quality [ZPvBG01], fast and hardware-assisted point data traversal [DVS03] and display [CH02], as well as memory-efficient data structures [BWK02]. This last work discusses important characteristics of points: they perform well compared to polygons in terms of memory consumption if many small geometric details have to be represented, as is the case for statues or plants [DCSD02, MO95]. However, in the case of lower geometric complexity but rich color detail, textured polygons need by far less memory. Although displaying a point cloud can be done by graphics hardware today, warping large depth images is a relatively costly operation compared to polygon-based representations because many per-sample transformations have to be performed. Popescu et al. [PLAdON98] show how image warping can be parallelized, which is unfortunately not supported by common hardware.

The per-pixel depth information allows an accurate reproduction of parallax effects and makes it possible to correctly resolve visibility with the surrounding scene. However, disocclusion artifacts in the output image may arise during display. In order to reduce them, multiple images can be warped to fill image gaps [RAL98, MMB97]. Another possibility is to use *layered depth images* (LDI) [SGHS98], which contain multiple color and depth values for every pixel position [PLAdON98, AL99, Max96]. Figure 9 shows a comparison of the image quality obtain with a single depth image and an LDI. Because it is desirable to adapt an LDI's level

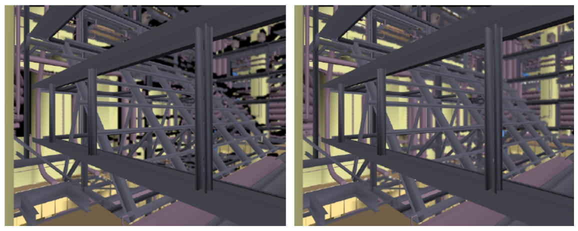
Figure 9: *Left: a single depth image, note the disocclusions visible as black gaps. Right: an LDI constructed using 9 images [AL99].*

of detail (i.e., its resolution) to different output image resolutions, Chang presented the *LDI tree* [CBL99], where LDI samples are stored and processed using an octree-like data structure.

More recently, a very high-quality impostor technique was presented by Wimmer et al. [WWS01], so-called *point-based impostors*. First, a set of points is obtained by ray casting from three viewpoints and removing redundant samples. Wimmer provides a sampling algorithm that tries to make sure that no holes become visible. They report on about two points required per screen pixel for the impostor. For every point, view dependent appearance is applied using Monte-Carlo ray tracing, thus generating a small point light field (see Section 3.7). This light field is encoded into a texture so that the rendering can be done using graphics hardware. The result is a high-quality anti-aliased impostor. However, the reported preprocessing times are relatively high. Furthermore, it might be argued that the high-quality impostors may stand out if the rest of the scene is rendered using OpenGL rendering. Therefore, the impostor technique is perfectly suited for applications that need very high-quality rendering without aliasing artifacts.

3.7. View Dependent Appearance and Dynamic Lighting: Lightfields and BTFs

Describing a complete scene with images can be intuitively understood using the *plenoptic function* [AB91]. This function describes the radiant energy received at a viewpoint x in view direction Θ and for wavelength Λ at time t:

$$\mu = Plenoptic(x, \Theta, \Lambda, t)$$

The output μ of this function can be any photometric unit useful for describing the color intensity values of images.

An image-based primitive that describes a relatively large subset of the plenoptic function [MB95] was introduced by Levoy and Hanrahan [LH96], called *light field*. Simultaneously, Gortler introduced the same concept, calling it *lumigraph* [GGSC96]. A light field describes view dependent appearance. If wavelength and time are not considered, and the domain of the light field is a convex region in space (either “inside-out” or “outside-in looking”) without obstacles, four dimensions are sufficient for a complete description of the plenoptic function. Two planes with a regular grid of sampling positions capture a light field: a camera is positioned at the sampling positions on the *entry plane* and records samples on the *exit plane*. The storage costs of a whole light field are high. Therefore, compression methods such as vector quantization and entropy encoding are applied [LH96]. Chai et al. [CTCS00] propose “plenoptic sampling,” which gives a minimum sampling density of images needed for the anti-aliased reconstruction of a light field if the minimum and maximum scene depth is known. However, since occlusions are not taken into account, image gaps are not avoided, so that the practical usability of the approach is restricted. In general, the huge amount of data still limits the use of light fields for scenes with larger extents.

So-called *surface light fields* [WAA*00] encode the directionally dependent radiance for points on an object surface. Consequently, only the radiance per fixed sample position is encoded, whereas the geometry is given separately. Because parallax effects and disocclusions are not encoded in the light field but represented by the geometry, a much better image quality compared to a standard light field can be achieved.

The image-based scene representations mentioned so far do not support *relighting*, i.e., dynamic light source modifications. Changing lighting conditions (such as moving shadows, specular highlights or anisotropic effects like metallic reflections) require the whole representation to be rebuilt. To overcome this problem, Wong [WHON97] extends light fields using the *bidirectional reflectance distribution function* (BRDF) [NRH*77]. This function encodes the relation between the incoming and outgoing light for every sample point, so that arbitrary lighting conditions can be reproduced at the cost of even higher memory requirements compared to light fields. The *bidirectional texture function* (BTF) [Dis98] is a discretized BRDF on an object surface. It is used for complex object surface descriptions because it provides view and illumination dependent appearance changes and can be encoded so that it can be rendered with standard graphics hardware. For small surfaces, it requires a tolerable amount of memory and can also be tiled or instantiated just like common textures [SvBLD03, MNP01, MMSK03].

3.8. Summary of Impostor Types

In this section, impostor techniques with different characteristics have been described. Table 1 summarizes the characterization with respect to important criteria including memory requirements, generation time, image quality issues and size of the validity region. Note that the memory requirements are listed here only for a single impostor. However, the actual memory requirements must always be seen in combination with the validity region. As an example, a single planar impostor needs by far less memory than a billboard cloud, many planar impostors are needed to represent an object from all sides, so the billboard cloud is more memory efficient in the end.

In summary it can be said that planar impostors are fast to generate, but since they are only valid within a small region, numerous updates are required. This makes them useful especially for runtime generation if storing numerous impostors generated in a preprocess is not possible. The validity region of layered impostors is larger due to the support of parallax effects and disocclusions at the cost of slightly longer generation time. Billboard clouds and textured depth meshes are very slow to generate, but offer large valid viewing regions. Especially the view independent characteristic of billboard clouds allows very large viewing regions and simple integration of the impostors into existing LOD systems. Depth images are fast to generate (LDIs are slightly slower

Impostor type	Memory requirements	Generation time	Support for parallaxes	Disocclusion artifacts	Validity region
Planar	low	fast	no	none	small
Layered	low/medium	medium/fast	yes	image gaps	medium
Billboard clouds	low/medium	very slow	yes	none	large
Video	low	medium/slow	yes	none	small/medium
TDM	low	slow	yes	rubber sheets	medium
Depth images	medium	fast	yes	image gaps	small
LDI	medium/high	medium/fast	yes	image gaps	medium

Table 1: *Characteristics of different impostor techniques. Note that disocclusion artifacts listed here concern viewpoints outside the validity region.*

to generate because more images have to be acquired), but if the impostors are generated in a preprocess, compression techniques have to be used in order to reduce the high memory consumption. Finally one can say that there is no best technique for all demands and applications. This will also become clear in the following section.

4. Using Image-based Representations for Rendering Acceleration

After describing different types of image-based representations, this section gives an overview how these techniques have been used in different systems for accelerating the rendering process. Every rendering system has to decide for every output view which scene parts to display with images, and where to rely on the original geometric representation. The goal is to maintain at least a reasonable image quality while at the same time achieving a high rendering acceleration for every output view.

4.1. Full screen methods

In a conventional rendering pipeline, every output image is rendered from scratch. In contrast, the idea of the rendering acceleration techniques described in this section is to redisplay previously rendered output images in order to save rendering time. In this way, frame-to-frame coherence is explored so that the rendering speed can be decoupled from scene complexity to a certain degree. The main decisions to be made are which scene parts to update and which kind of image-based representation to use for them.

As a simple but effective approach, Bishop et al. [BFMS94] presented *frameless rendering*. Here, pixels are displayed instantly and in a random order using ray casting. This allows displaying fluent animations if the output images cannot be rendered sufficiently fast, at the cost of reduced image quality due to varying times for the update of every pixel.

Chen and Williams [CW93] presented an approach that interpolates between images of different reference viewpoints in order to generate an output image. This technique provides reasonable image quality for viewpoints along the line between two reference images. McMillan and Bishop [MB95] describe a system called *plenoptic modeling* for acquiring cylindrical images and synthesizing new views from them. By taking into account the relative camera positions, it allows exact view reconstruction for all interpolated output images.

Reagan and Pose [RP94] presented a hardware architecture called *virtual address recalculation pipeline* that shows many aspects of image-based rendering acceleration. In that approach, the scene is partitioned into depth layers with different distances to the viewer. Every layer has its own video memory. Because changes occur more often for near scene parts, the layers are updated at different frame rates (so-called *priority rendering*). The acceleration was more than one order of magnitude compared to the traditional rendering pipeline at that time. Lengyel and Snyder [LS97] enhanced this concept and partitioned the output image into *coherent image layers*. The image layers are generated with respect to object perception in the fore- and background, different object movements, and efficient usage of texture memory. Rendering resources are then adaptively distributed so that fast-moving foreground objects get more rendering resources than a hardly changing background. The rendering system was implemented on Talisman [TK96], a rendering hardware prototype that directly supports rendering with such coherent image layers.

Mark et al. [MMB97] present a method called *post-rendering 3D image warping*. Only every n-th output image is rendered from geometry. Images in between are generated by composing two to three reference images rendered from previous and predicted future viewpoints. This ensures that most visible scene parts are present in the reference images so as to avoid image gaps.

In the work of Larson and Simmons [LS99] and Simmons and Sequin [SS00], graphics hardware is used to reproject already cached radiance values by using a triangulated unit sphere centered at the viewpoint. Figure 10 shows how image quality is improved over time if many samples can be reused. The render cache of Walter et al. [WDP99, WDG02] is in the same spirit.

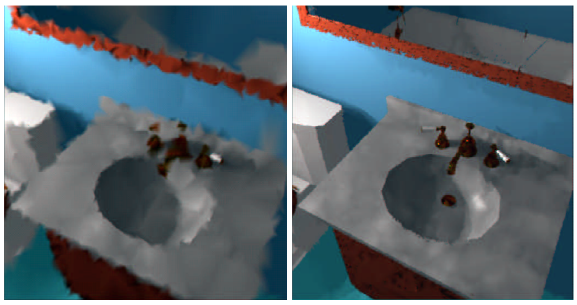
Figure 10: *Improved image quality over time by reusing previously rendered samples [SS00].*

Since images are generated at runtime, all methods described here share the problem that too many scene parts might need to be updated between two consecutive output images. For that case, the rendering speed and/or the image quality may drop significantly. This can be avoided by using precalculated images. As an early related technique, the Quicktime VR System developed by Chen [Che95] stores cylindrical panoramic images that are obtained from a fixed viewpoint. These images are stored on harddisk and reprojected for display. User movement is restricted to rotation around the viewpoint, zooming, and discrete changes of the viewpoint. In contrast, the technique of Darsa et al. [DC96, DSV97, DCV98] allows arbitrary continuous movements while the scene is represented by multiple TDMs recorded from multiple view points in the scene. Image gaps and rubber sheet effects are avoided by simultaneously displaying triangles of several TDMs in the output image.

4.2. Far field impostors

This strategy partitions the view space into a set of view cells, and for every cell, the scene is split into a *near field* and a *far field*. While the near field is rendered using geometry, the far field is displayed using image-based representations. Distant scene parts often show complex geometry that covers only few pixels on the screen. Such scene parts allow high rendering acceleration and at the same time low memory requirements for static impostors. Furthermore, the images are valid for a large viewing region because of only few apparent parallax movements due to the large distance to the viewer.

Aliaga et al. [ACZ*97] describe a method called *textured box culling*. In that approach, the far field is represented by a cube consisting of 6 precalculated textured quads. This results in visible popping artifacts when the view cell is changed at runtime. In subsequent work, Aliaga et al. [ACW*99] use textured depth meshes (see Figure 11) for the far field representation. Wilson et al. [WLY*00] instead

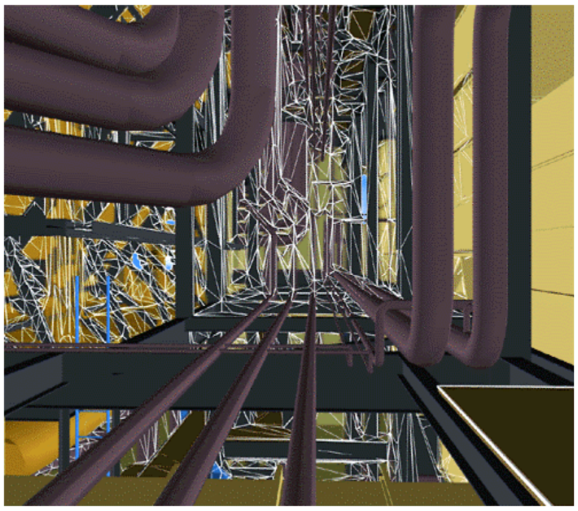
Figure 11: *Distant geometry is represented using a texture depth mesh [ACW*99].*

use video-based impostors (see Section 3.4) and Jeschke et al. [JWS02] use layered impostors (see Section 3.2).

Sillion et al. [SDB97] and later Decoret et al. [DSSD99] apply a similar strategy to an urban environment using TDMs (see Section 3.5). However, instead of using a regular grid of view cells, the special scene characteristic is exploited: view cells coincide with street segments and TDMs are placed at the end of street segments. The is done because they assume that facades close to the viewer completely occlude the scene behind. Exploiting visibility information greatly reduces the required impostor memory. Figure 8 shows an output image generated with that system.

Wimmer et al. [WGS99] divide the scene into a near and a far field during runtime. While the near field is rendered using conventional graphics hardware rendering, the far field is rendered using ray casting. They argue that with increasing distance, more geometric primitives cover an output pixel, so that ray casting performs better than conventional rendering. A radiance cache in the form of a panoramic image was used to keep the number of cast rays low for every frame.

4.3. Object-centered approaches

In this impostor usage strategy, whether a scene part is rendered from geometry or using an impostor is decided separately for each object. This decision is typically based on the projected area of the object on screen and/or the distance between object and viewer. The image-based representations efficiently display complex objects from a certain distance, because in this case they cover only few pixels on the screen and parallax effects are rather small so that the update frequency is only low. For instance, in computer games, impostors are typically used in conjunction with geometric levels of detail: at very far distances, planar billboards are displayed instead of geometric models. The following subsections describe some special scenes where impostors have been successfully applied.

4.3.1. Forests

Weber and Penn's classic paper [WP95] on modeling and rendering realistic trees also includes point-based rendering. Although that publication focuses mainly on the modeling aspect, they make use of point and line primitives for leaves and branches, respectively. The representation created from their model is not explicitly converted to geometry, but interpreted at runtime. At close distances, full-resolution polygonal geometry is created, whereas at progressively increasing distances, stems will be rendered as lines and leaves as points. Heuristic methods are used for the transition between these representations.

An algorithm proposed by Max and Ohsaki [MO95] uses precomputed z-buffer views to approximate arbitrary viewpoints. These views are acquired through parallel projection from a number of viewpoints generated through a simple longitude/latitude sphere partitioning scheme. Since there is little coherence between leaves in a tree, the reconstruction for an arbitrary view point is performed on a per-pixel basis. This typically leaves some pixels undefined where no information can be extracted from the available views. The authors have chosen to implement multiple z-buffer layers to reduce these artifacts. Dynamic shading and shadowing is supported by storing compressed normal vector and material information for each pixel of the precomputed views. During the shading post-process, these values are used to compute diffuse and phong shading. Shadows can be found by reconstructing a z-buffer view for the light source and testing output pixels against this buffer. Since normal vector and material information is available, "deferred shading" can be applied in a post-processing step once for each output pixel. Later Max extended this work to hierarchical rendering [Max96] as well as utilizing texture mapping graphics hardware [MDK99].

Jakulin [Jak00] records trees from six viewpoints using a static layered impostor technique [Sch98b]. This results in four megabytes per tree, so that impostors for numerous trees can be stored in graphics hardware. By instantiating the trees, a complete forest scene can be modelled using impostors without the need of enormous amounts of impostor memory.

More recently, Decaudin and Neyret [DN04] presented a similar forest modelling and rendering method that uses volumetric textures. It is designed for forest flyovers. The volumetric texture consist of slices parallel to the ground plane. In order to avoid artifacts at grazing angles near the horizon, the volumetric texture is sliced differently for such viewing angles. With that method they were able to render forests consisting of about 37,000 trees at interactive frame rates. Figure 12 shows a result of their system.

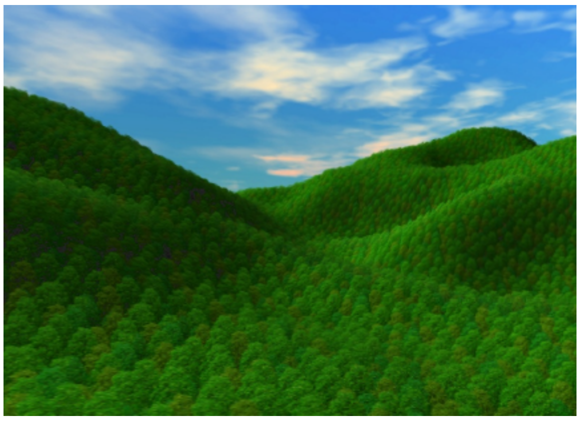

Figure 12: *A forest scene rendered from volumetric textures [DN04].*

Another interesting aspect when dealing with forest rendering was described by Cohen-Or et al. [COSL*04]: when moving on the ground through a forest, close trees hide most of the foliage behind them. Their method uses *aggressive visibility* to tag large and mostly occluded portions of the scene as background, which is rendered using planar impostors. The trick is to ensure that the foreground is always dense enough so that errors in the impostors do not become obvious.

Another aspect are dynamic lighting effects for allowing for instance rendering a forest over a whole day with convincing lighting changes. The image-based rendering system proposed by Meyer et al. [MNP01] provides a framework for rendering trees with effects such as shading, self shadowing, and dynamic illumination. They combine a hierarchy of bidirectional textures (HBT) to provide planar images for each given viewpoint and light direction with a hierarchical visibility structure for self-shadowing and casting shadows. This representation is efficient for trees, as it is hierarchical and instancing is heavily used. BTFs (see Section 3.7) are computed by associating a representation with each pair of view and light directions. Between 6 and 258 different view directions and light directions were used. During rendering, an arbitrary configuration can be approximated by interpolating 9 BTFs. These BTFs are associated to each level in the hierarchy either by creating a new, unique BTF or through instancing. During rendering, either the BTF mapped impostor or the actual geometry is rendered depending on the distance. To support dynamic lighting effects, approximate visibility cube maps are computed for each level of the hierarchy. Since occlusion depends on the position within the hierarchy, separate cube-maps need to be generated for all instances. Shadowing can then be computed during rendering by traversing the hierarchy of visibility cube maps. Casting shadows is supported through “traditional” shadow maps rendered from the light source.

4.3.2. Hair and fur

Lengyel [Len00] used the idea of multiple texture layers presented by Meyer and Neyret [MN98] (see Section3.2) for rendering shaded hair. While at very close distance the actual hair geometry should be rendered, image layers represent it from a certain distance. The number of layers scales with the viewer distance in order to provide a sufficiently detailed image. In later work, Lengyel et al. [LPFH01] extended the method for rendering very large furry models without the need for extensive texture memory, and they provided higher image quality at object silhouettes.

4.3.3. Clouds

The efficiency of dynamic impostors [Sch95] depends on how the speed gain obtained from the impostor display makes up for the additional effort for impostor updates. Harris and Lastra [HL01] represent clouds with dynamic impostors. Because clouds are internally represented as a particle system, impostors are also efficient for nearby clouds as they are much faster to render than the particles. Furthermore, because clouds do not contain high-frequency details like sharp edges, popping artifacts and blocky pixels are not a big problem. Here impostors are even used in order to avoid artifacts that would occur when directly rendering the particles. Note that their particle system also includes a cloud shading algorithm that approximates multiple forward scattering in a preprocess, and first order anisotropic scattering at runtime. Figure 4 shows an output image of that rendering system.

More recently, Wang [Wan04] presented a cloud rendering system that allows easily creating and rendering many different types of clouds. Similar to the approach of Harris and Lastra, planar impostors are used to render the clouds. Ambient light and a directional light is supported for cloud shading. The system is mainly designed for artistic modeling work rather than as being a physically valid model. However, rendering the clouds is faster compared to the particle system of Harris and Lastra and the visual quality is comparable.

4.3.4. Crowds

Aubel et al. [ABT98, ABT99] presented a rendering system for human crowds which uses simple planar impostors that were updated *dynamically* in order to accelerate the rendering process.

Tecchia and Chrysanthou [TC00] used *static* planar impostors for displaying humans. A snapshot of a human is recorded at 16×8 positions, 16 in horizontal and 8 in vertical direction. This is done for 10 human poses of a walking animation. At runtime, the right impostor is chosen depending on the pose and view direction, thus reducing the polygonal complexity of the human to only a single quad. With this method, it was possible to render up to 10,000 humans interactively. In subsequent work [TLC02a], they made better use of texture memory, applied texture compression and varied the color of the shirts and trousers in order to obtain more variability without significant additional rendering and storage cost. Furthermore, they added dynamic lighting [TLC02b] to the characters by using precomputed normal maps. The humans were also able to cast shadows [LCT01] on the ground and got shadowed by buildings, which further improved realism. They were able to render about 2,000 shaded and shadowed people based on multipass rendering. Recently, Dobbyn et al. [DHOO05] extended that work by switching to the original geometric representation if a human comes close to the camera (thus forming a 2-level representation). Furthermore, shadows were applied using the stencil buffer and programming capabilities of modern graphics hardware was used to speed up the display process. Finally, they were able to render about 10,000 shaded and shadowed humans interactively. Figure 13 shows an output image of their system.

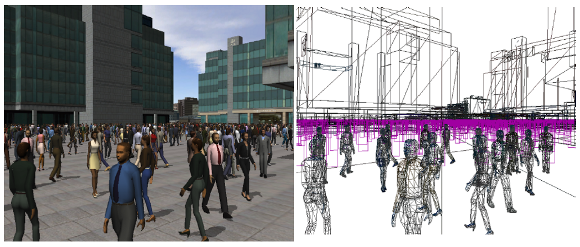
Figure 13: *Human crowd in the “Virtual Dublin” project [DHOO05]. Note the planar impostors for humans from a certain distance.*

4.4. Architectural Models

For architectural models, approaches that partition the model into cells (coinciding with rooms) and portals (coinciding with open connections between cells like doors and windows) [ARB90] provide efficient visibility culling. When image-based representations are placed in portals so that they represent the geometry of neighboring cells, only the current cell has to be rendered from geometry. If the viewer comes near a portal, the adjacent cell is also rendered from geometry in order to avoid disturbing image artifacts. Portals are a convenient place for using impostors because the complex geometry behind it covers only few pixels on the screen. Furthermore, because the overall number of portals in a model is quite limited, precalculated impostors may fit into main memory or even in graphics hardware memory.

Aliaga and Lastra [AL97] use planar impostors placed in the portals. In order to obtain a reasonable image quality, multiple impostors are generated from viewpoints on a semicircle in front of the portal. In contrast, Rafferty et al. [RAL98] warp depth images in order to reduce the number of impostors needed for a convincing representation. For avoiding disocclusion artifacts, two depth images are warped simultaneously to the output image. Popescu et al. [PLAdON98] instead use a layered depth image (LDI) for every portal. For fast LDI display, they use a parallel warping algorithm and clip invisible samples using a hierarchical data structure for the LDI. Simmons and Sequin [SS01] use textured depth meshes with three TDMs per portal, each recorded from another viewpoint. Rubber sheet artifacts are avoided by composing multiple TDMs during rendering. Rafferty et al. [RAPL98] give a comparison of the approaches from Aliaga, Rafferty and Popescu. Furthermore, while online generation of portal impostors would be possible, Rafferty et al. [RAL98] reported "animation hick-ups" due to the too time-consuming impostor generation process parallel to the model display.

4.5. Hierarchical Image Cache

Schaufler and Stürzlinger [SS96] as well as Shade et al. [SLS*96] concurrently presented rendering systems using dynamic impostors, called *hierarchical image cache*. The algorithm relies on a hierarchy of the model, with nearby objects clustered first. During runtime, impostors are dynamically generated for the leaf nodes of the hierarchy. Impostors for intermediates nodes are generated from the impostors of their children. The impostor update is controlled by the error metrics mentioned in Section 3.1. In order to render an output image, the tree is traversed, and the first node containing an impostor that is valid for the current view point is displayed, and subtree traversal is discarded. Consequently, with increasing distance to the viewer, impostors are displayed for ever higher hierarchy nodes, thus greatly accelerating the rendering process.

The hierarchical image cache relies on coherent output images. A problem may arise if the viewer moves very fast so that many nodes have to be frequently updated. In general the update rate of nodes near the viewer is so high that they are rendered from geometry. Shade et al. calculate the lifetime of each potential impostor based on the distance to the viewer. It is only generated if its lifetime justifies the generation cost. Furthermore, they mention the possibility of generating impostors predictively in order to avoid sudden drops of the output frame rate if many nodes need to be instantly updated. Using visibility calculations for reducing the number of impostor updates might be an interesting extension to that approach. However, in some situations the user movement needs to be restricted to avoid too many impostor updates or the image quality must be temporarily decreased. This is the main drawback introduced by the use of dynamic impostors.

4.6. Guaranteed Frame Rates

An interesting application is the *guarantee* of a frame rate through the use of image-based representations. Note that the object-centered approaches described in Section 4.3 are not useful if too many objects should be visualized, so that the sheer number of objects (and therewith rendering calls) already overwhelms hardware capabilities.

4.6.1. Dynamic Representation Selection: Enhancements of Funkhouser's Adaptive Display Algorithm

Funkhouser and Sequin [FS93] presented a predictive LOD selection algorithm for maximizing the image quality and meeting a desired frame rate. This is done using a cost-benefit heuristic: the cost of an object basically describes the time it needs to be rendered, while the benefit describes the importance of an object for an output image. A greedy algorithm is applied that preferably selects objects with high benefit/cost ratio until a user-defined frame time budget has been reached.

Maciel and Shirley [MS95] enhanced the work of Funkhouser and Sequin by introducing *static* impostors into the framework. They replace nodes of an input model hierarchy with impostors, taking advantage of the fact that impostors can be applied to arbitrary scene parts. This allows displaying many objects faster and with higher image quality than graphics hardware capabilities would allow for a geometric representation. However, huge texture memory requirements were also experienced, and they also mentioned prefetching impostors dynamically from harddisk. In subsequent work, Mason and Blake [MB97] provide a LOD selection algorithm for hierarchically organized scenes with a result that is at least half as good as the optimal solution.

Schaufler [Sch96] shows how *dynamic* impostors can be incorporated into the framework of Funkhouser and Sequin. Instead of just displaying an object with an appropriate level of detail, an impostor is generated and displayed using a certain level of detail if the object is suitable for an impostor representation. If impostors are displayed instead of original objects, more frame time is left for generating higher quality impostors from better LOD selections. This means that the image quality is improved progressively over time if the user stands still. In contrast, in the original framework of Funkhouser and Sequin, the LOD selection is always the same.

4.6.2. Static Representation Selection

A possible way to maintain a desired frame rate is to use far field representations (see Section 4.2) and shrink the near field until the remaining geometry in the near field together with the far field representation can be rendered fast enough [WMPM01, WM03]. However, a problem arises if the memory needed for the image-based representation grows beyond what is tolerable. For such situations, a lower number of samples can be used, thus decreasing the image quality. For instance, Aliaga et al. [ACW*99] balance the image quality error caused by a TDM representation with the error caused by a geometric LOD simplification in the near field by adapting the distance of the border between near and far field.

However, static image-based representations should not be placed indiscriminately, but only where needed for a frame rate guarantee in order to make best use of available memory. Aliaga et al. [AL99] developed an algorithm for using LDIs only where they are necessary so that a prescribed primitive budget is not exceeded, depending on the view position and direction. They hierarchically subdivide the view space using a grid of points adapted to the local model complexity. For every grid point and view direction, an optimization algorithm selects a model subset which is represented using an LDI so that the primitive budget is met. The model subset is chosen using a cost-benefit heuristic aimed at low memory requirements. This means that small, distant and complex model subsets are preferred to large, nearby and less complex ones. Figure 9 shows an output image from that system.

A constraint of that approach is that only one single LDI can be displayed per output view, which might lead to situations where scene parts are present in multiple LDIs at a grid point. Furthermore, many similar impostors for distant scene parts might be generated for adjacent grid points. To overcome this drawback, Jeschke et al. [JW05] allow layered impostors (see Section 3.2) to be generated for *arbitrary* combinations of view cells and nodes of a model hierarchy. First, they identify those views within a scene that have to be accelerated (calling them *problem views*) even after applying visibility culling and geometric levels of detail. Afterwards, for every node of the model hierarchy, a huge number of view cells are generated, each called an *impostor candidate*. Every candidate is rated with respect to its combined rendering acceleration for all problem views and the amount of memory the final impostor would need. This means, that candidates that accelerate many problem views and need only little memory have a better ratio than memory costly impostors that accelerate only few problem views. Finally, a greedy optimization algorithm successively selects the candidates with the best ratio and generates the respective impostors until all problem views can be rendered sufficiently fast. The sensitive impostor selection algorithm allows storing the impostors of a whole city model completely into graphics hardware, thus avoiding dynamic texture prefetching. Another interesting aspect in that work is the use of a rendering time estimation heuristic [WW03] instead of the simple polygon budget used by most previous approaches.

4.7. Summary of Impostor Applications

In Section 4 we have described known strategies on how to use impostors for different scenes so that the rendering accelerations are high and the impostor updates/memory requirements are low. Table 2 summarizes these methods. The wide variety shows that the best use of impostors heavily depends on the type of scene and interaction method. For instance, layered impostors for forests as presented by Decaudin [DN04] work fine for flyovers, whereas forest walkthroughs need a different technique.

5. Summary and Future Challenges

In this STAR, an overview of impostors used for accelerated rendering was given. First, general characteristics, demands and challenges of image-based rendering acceleration techniques have been stated. Afterwards, image-based primitives presented in literature have been presented, ranging from simple planar billboards to more complex techniques like LDIs. In the last part, different ways for using impostors for rendering acceleration for different scenes and demands have been shown, mostly utilizing a priori scene information like visibility or geometric scene structure (e.g. instantiated objects), or addressing frame-to-frame coherence. There exists no impostor technique that should be favored for all demands and types of scenes. A review on many impostor-based rendering systems can be found in the PhDs of Jeschke [Jes05] and Wilson [Wil02].

An increasingly important issue that has to be addressed by impostors is scene dynamics: impostors should appear like the original scene parts also if scene conditions change. Especially the desire for more scene realism causes the demand for a more flexible use of impostors. For instance, in current games, every scene part may cast shadows, so it is desirable that an impostor can cast shadows as well. Until now, shadows were only presented for the billboard cloud technique [DDSD03], which naturally supports the use of shadow mapping algorithms due to its view independent characteristic. However, solutions must be found to support this feature also for view dependent impostors.

Another important aspect is the increasing use of programmable graphics hardware for realistic scene appearances. The support of complex shading effects like metal shading and environment map reflections is highly desirable for impostors. Many papers propose the use of normal maps for online lighting calculations (an example was presented for the billboard cloud technique [DDSD03]). While this allows the reproduction of dynamic lighting, a problem

	Planar	Layered	BBC	Video	TDM	Depth images/LDI
Full screen	static/dynamic	dynamic	-	-	static/dynamic	dynamic
Far field	static	static	-	static	static	static/dynamic
General objects	static/dynamic	static/dynamic	static	-	static	-
Forest	static	static	-	-	-	static
Hair/Fur	-	static	-	-	-	-
Clouds	static/dynamic	-	-	-	-	-
Humans	static/dynamic	-	-	-	-	-
Architecture	static	-	-	-	static	static/dynamic
Hierarchical image caching	dynamic	-	-	-	-	-
Guaranteed frame rates	static/dynamic	static	-	-	-	static

Table 2: *Overview of impostor rendering systems using static and/or dynamic impostor generation.*

occurs if various objects with different shading effects are represented by a single impostor. In this case, *all* shading programs would have to be present in the shading program for the impostor. The situation becomes even more difficult if the appearance of an object is defined by a combination of the vertex *and* pixel shader, as is the case for environment maps. Wimmer et al. [WWS01] presented a general approach that is based on a light field for representing view dependent appearance changes, mainly at the cost of long impostor generation times and high memory requirements. However, the inclusion of dynamic effects like for example a skyscraper with moving clouds in the windows has not been solved until now. Dynamic lighting makes the problem even more difficult, because the appearance depends on the configuration of the lights in a scene. No general solution for this problem has been found until now and doing so seems to be challenging, particularly with regard to memory requirements.

In conclusion, impostors as image-based representations offer a convenient way to have the time needed to render a scene part depend primarily on the number of pixels it covers on screen rather than on the complexity of its geometric representation. However, care has to be taken about the rapidly rising amount of memory needed for storing the impostors. Past work has shown different ways for doing so. The next challenge is the development of techniques that increase the flexibility of impostors with respect to dynamic shading effects which programmable graphics hardware offers.

References

[AB91] Adelson E. H., Bergen J. R.: The plenoptic function and the elements of early vision. In *Computational Models of Visual Processing*, Landy M., Movshon J. A., (Eds.). MIT Press, 1991, pp. 3–20. 8

[ABC*04] Andujar C., Brunet P., Chica A., Navazo I., Rossignac J., Vinacua A.: Computing maximal tiles and application to impostor-based simplification. *Computer Graphics Forum 23*, 3 (2004), 401–410. 6

[ABT98] Aubel A., Boulic R., Thalmann D.: Animated impostors for real-time display of numerous virtual humans. In *Proceedings of the 1st International Conference on Virtual Worlds (VW-98)* (Berlin, July 1–3 1998), Heudin J.-C., (Ed.), vol. 1434 of *LNAI*, Springer, pp. 14–28. 12

[ABT99] Aubel A., Boulic R., Thalmann D.: Lowering the cost of virtual human rendering with structured animated impostors. In *WSCG'99 Conference Proceedings* (1999), Skala V., (Ed.), Univ. of West Bohemia Press, pp. 345–352. 3, 4, 12

[ABT00] Aubel A., Boulic R., Thalmann D.: Real-time display of virtual humans: Levels of detail and impostors. In *IEEE Transactions on Circuits and Systems for Video Technology* (2000), pp. 207–217. 3, 4

[ACW*99] Aliaga D., Cohen J., Wilson A., Baker E., Zhang H., Erikson C., Hoff K., Hudson T., Stürzlinger W., Bastos R., Whitton M., Brooks F., Manoclia D.: MMR: An interactive massive model rendering system using geometric and image-based acceleration. In *1999 Symposium on interactive 3D Graphics* (Apr. 1999), Spencer S. N., (Ed.), ACM SIGGRAPH, ACM Press, pp. 199–206. ISBN 1-58113-082-1. 6, 10, 14

[ACZ*97] ALIAGA D., COHEN J., ZHANG H., BASTOS R., HUDSON T., ERIKSON C.: *Power Plant Walkthrough: An Integrated System for Massive Model Rendering*. Tech. Rep. TR97-018, Department of Computer Science, University of North Carolina - Chapel Hill, 1997. 10

[AL97] ALIAGA D. G., LASTRA A. A.: Architectural walkthroughs using portal textures. In *Proceedings of the conference on Visualization '97* (Oct. 1997), Yagel R., Hagen H., (Eds.), IEEE, pp. 355–362. 13

[AL98] ALIAGA D. G., LASTRA A. A.: Smooth transitions in texture-based simplification. *Computers and Graphics 22*, 1 (Feb. 1998), 71–81. ISSN 0097-8493. 4

[AL99] ALIAGA D. G., LASTRA A. A.: Automatic image placement to provide a guaranteed frame rate. In *SIGGRAPH 99 Conference Proceedings* (Aug. 1999), Rockwood A., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 307–316. 3, 7, 14

[Ali96] ALIAGA D. G.: *Portal Textures: Texture Flipbooks for Architectural Models*. Tech. Rep. TR96-049, Department of Computer Science, University of North Carolina - Chapel Hill, 1996. 4

[ARB90] AIREY J. M., ROHLF J. H., BROOKS, JR. F. P.: Towards image realism with interactive update rates in complex virtual building environments. In *Symposium on Interactive 3D Graphics* (Mar. 1990), Riesenfeld R., Sequin C., (Eds.), pp. 41–50. 12

[BFMS94] BISHOP G., FUCHS H., MCMILLAN L., SCHER ZAGIER E. J.: Frameless rendering: Double buffering considered harmful. In *SIGGRAPH 94 Conference Proceedings* (July 1994), Glassner A., (Ed.), Annual Conference Series, ACM SIGGRAPH, ACM Press, pp. 175–176. ISBN 0-89791-667-0. 9

[BWK02] BOTSCH M., WIRATANAYA A., KOBBELT L.: Efficient high quality rendering of point sampled geometry. In *Proceedings of the 13th Eurographics workshop on Rendering* (2002), Eurographics Association, pp. 53–64. 7

[CBL99] CHANG C.-F., BISHOP G., LASTRA A.: LDI tree: A hierarchical representation for image-based rendering. In *SIGGRAPH 99 Conference Proceedings* (Aug. 1999), Rockwood A., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 291–298. 7

[CH02] COCONU L., HEGE H.-C.: Hardware-accelerated point-based rendering of complex scenes. In *Proceedings of the 13th Eurographics workshop on Rendering* (2002), Eurographics Association, pp. 43–52. 7

[Che95] CHEN S. E.: Quicktime VR - an image-based approach to virtual environment navigation. In *SIGGRAPH 95 Conference Proceedings* (Aug. 1995), Cook R., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 29–38. 10

[CIKK99] CHEN B., II J. E. S., KUO E., KAUFMAN A.: Lod-sprite technique for accelerated terrain rendering. In *Proceedings of the 10th IEEE Visualization 1999 Conference (VIS '99)* (1999), Ebert D., Gross M.,, Hamann B., (Eds.), IEEE Computer Society, pp. 291–298. 6

[COSL*04] COHEN-OR D., SAYER E., LERNER A., CHRYSANTHOU Y., DEUSSEN O.: Aggressive visibility for rendering extremely complex foliage scenes, 2004. http://www.cs.tau.ac.il/∼alan/aggressive.htm. 11

[CTCS00] CHAI J.-X., TONG X., CHAN S.-C., SHUM H.-Y.: Plenoptic sampling. In *SIGGRAPH 2000 Conference Proceedings* (2000), Akeley K., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 307–318. 8

[CW93] CHEN S. E., WILLIAMS L.: View interpolation for image synthesis. In *SIGGRAPH 93 Conference Proceedings* (Aug. 1993), Kajiya J. T., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 279–288. ISBN 0-201-51585-7. 9

[DC96] DARSA L., COSTA B.: Multi-resolution representation and reconstruction of adaptively sampled images. In *SIBGRAPI'96 Proceedings* (1996), pp. 321–328. 6, 7, 10

[DCSD02] DEUSSEN O., COLDITZ C., STAMMINGER M., DRETTAKIS G.: Interactive visualization of complex plant ecosystems. In *Proceedings of the 13th IEEE Visualization 2002 Conference* (2002), Moorhead R., Gross M.,, Joy K. I., (Eds.), IEEE Computer Society, pp. 219–226. 7

[DCV98] DARSA L., COSTA B., VARSHNEY A.: Walkthroughs of complex environments using image-based simplification. *Computers and Graphics 22*, 1 (1998), 55–69. 6, 7, 10

[DDSD02] DECORET X., DURAND F., SILLION F. X., DORSEY J.: *Billboard Clouds*. Tech. Rep. 4485, INRIA, Rhône-Alpes, 2002. 5

[DDSD03] DECORET X., DURAND F., SILLION F. X., DORSEY J.: Billboard clouds for extreme model simplification. *ACM Trans. Graph. 22*, 3 (2003), 689–696. 3, 5, 14

[DHOO05] DOBBYN S., HAMILL J., O'CONOR K., O'SULLIVAN C.: Geopostors: A real-time geometry/impostor crowd rendering system. In *Proceedings of Symposium on Interactive 3D Graphics and Games* (2005), pp. 95–102. 12

[Dis98] DISCHLER J.-M.: Efficient rendering macro geometric surface structures with bi-directional texture functions. In *Rendering Techniques '98*

(1998), Drettakis G., Max N., (Eds.), Eurographics, Springer-Verlag Wien New York, pp. 169–180. 8

[DN04] DECAUDIN P., NEYRET F.: Rendering forest scenes in real-time. In *Eurographics Symposium on Rendering '04* (2004), H. W. Jensen A. K., (Ed.), pp. 93–102. 11, 14

[DSSD99] DECORET X., SILLION F., SCHAUFLER G., DORSEY J.: Multi-layered impostors for accelerated rendering. *Computer Graphics Forum (Proc. Eurographics '99) 18*, 3 (Sept. 1999), 61–73. ISSN 1067-7055. 6, 7, 10

[DSV97] DARSA L., SILVA B. C., VARSHNEY A.: Navigating static environments using image-space simplification and morphing. In *1997 Symposium on Interactive 3D Graphics* (Apr. 1997), Cohen M., Zeltzer D., (Eds.), ACM SIGGRAPH, ACM Press, pp. 25–34. ISBN 0-89791-884-3. 6, 10

[DVS03] DACHSBACHER C., VOGELGSANG C., STAMMINGER M.: Sequential point trees. *ACM Transactions on Graphics 22*, 3 (2003), 657–662. 7

[Ebb98] EBBESMEYER P.: Textured virtual walls - achieving interactive frame rates during walkthroughs of complex indoor environments. In *Proceedings of the Virtual Reality Annual International Symposium* (1998), IEEE Computer Society, p. 220. 4

[FS93] FUNKHOUSER T. A., SÉQUIN C. H.: Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH 93 Conference Proceedings* (Aug. 1993), Kajiya J. T., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 247–254. ISBN 0-201-51585-7. 13

[GGSC96] GORTLER S. J., GRZESZCZUK R., SZELISKI R., COHEN M. F.: The lumigraph. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), Rushmeier H., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 43–54. held in New Orleans, Louisiana, 04-09 August 1996. 8

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), Whitted T., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 209–216. ISBN 0-89791-896-7. 6

[HL01] HARRIS M. J., LASTRA A.: Real-time cloud rendering. In *EG 2001 Proceedings*, Chalmers A., Rhyne T.-M., (Eds.), vol. 20(3) of *Computer Graphics Forum*. Blackwell Publishing, 2001, pp. 76–84. 4, 5, 12

[Jak00] JAKULIN A.: Interactive vegetation rendering with slicing and blending. In *Proceedings of Eurographics 2000 (Short Presentations)* (Aug. 2000), de Sousa A., Torres J., (Eds.), Eurographics. 3, 4, 5, 11

[Jes05] JESCHKE S.: *Accelerating the rendering process using impostors*. PhD thesis, University of Rostock, 2005. 14

[JW02] JESCHKE S., WIMMER M.: Textured depth meshes for real-time rendering of arbitrary scenes. In *Proceedings of the 13th Eurographics workshop on Rendering* (2002), Eurographics Association, pp. 181–190. 6, 7

[JW05] JESCHKE S., WIMMER M.: Automatic impostor placement for guaranteed frame rates and low memory requirements. In *Proceedings of Symposium on Interactive 3D Graphics and Games* (2005), pp. 103–110. 14

[JWS02] JESCHKE S., WIMMER M., SCHUMANN H.: Layered environment-map impostors for arbitrary scenes. In *Proceedings of the Graphics Interface 2002* (May 27–29 2002), Canadian Information Processing Society, pp. 1–8. 5, 10

[LCT01] LOSCOS C., CHRYSANTHOU Y., TECCHIA T.: Real-time shadows for animated crowds in virtual cities. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology (VRST-01)* (New York, Nov. 15–17 2001), Shaw C., Wang W., (Eds.), ACM Press, pp. 85–92. 12

[LE97] LUEBKE D., ERIKSON C.: View-dependent simplification of arbitrary polygonal environments. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), Whitted T., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 199–208. ISBN 0-89791-896-7. 6

[Len00] LENGYEL J. E.: Real-time hair. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000* (London, UK, 2000), Springer-Verlag, pp. 243–256. 12

[LH96] LEVOY M., HANRAHAN P.: Light field rendering. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), Rushmeier H., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 31–42. 8

[LPFH01] LENGYEL J., PRAUN E., FINKELSTEIN A., HOPPE H.: Real-time fur over arbitrary surfaces. In *Proceedings of the 2001 Symposium on Interactive 3D graphics* (2001), ACM Press, pp. 227–232. 12

[LS97] LENGYEL J., SNYDER J.: Rendering with coherent layers. In *SIGGRAPH 97 Conference Proceedings* (Aug. 1997), Whitted T., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 233–242. ISBN 0-89791-896-7. 9

[LS99] LARSON G. W., SIMMONS M.: The holodeck interactive ray cache. In *ACM SIGGRAPH 99 Conference abstracts and applications* (1999), ACM Press, p. 246. 10

[Max96] MAX N.: Hierarchical rendering of trees from precomputed multi-layer Z-buffers. In *Rendering Techniques '96 (Proceedings of the Eurographics Workshop on Rendering 96)* (June 1996), Pueyo X., Schröder P., (Eds.), Eurographics, Springer-Verlag Wien New York, pp. 165–174. ISBN 3-211-82883-4. 7, 11

[MB95] McMillan L., Bishop G.: Plenoptic modeling: An image-based rendering system. In *SIGGRAPH 95 Conference Proceedings* (Aug. 1995), Cook R., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 39–46. 7, 8, 9

[MB97] Mason A. E. W., Blake E. H.: Automatic hierarchical level of detail optimization in computer animation. *Computer Graphics Forum 16*, 3 (1997), 191–199. 13

[McM97] McMillan L.: *An Image-based Approach to Three-Dimensional Computer Graphics*. Ph.d. thesis, University of North Carolina at Chapel Hill, 1997. also available as UNC Technical Report TR97-013. 7

[MDK99] Max N., Deussen O., Keating B.: Hierarchical image-based rendering using texture mapping hardware. In *Proceedings of the Eurographics Workshop on Rendering '99* (June 1999), Eurographics, Springer-Verlag, pp. 57–62. 11

[MH02] Möller T., Haines E.: *Real-Time Rendering*. A. K. Peters Limited, 2002. 2nd edition, ISBN 1568811829. 2

[MK04] Meseth J., Klein R.: Memory efficient billboard clouds for btf textured objects. In *Vision, Modeling, and Visualization 2004* (November 2004), Girod B., Magnor M.,, Seidel H.-P., (Eds.), Akademische Verlagsgesellschaft Aka GmbH, Berlin, pp. 167–174. 6

[MMB97] Mark W. R., McMillan L., Bishop G.: Post-rendering 3D warping. In *1997 Symposium on Interactive 3D Graphics* (Apr. 1997), Cohen M., Zeltzer D., (Eds.), ACM SIGGRAPH, ACM Press, pp. 7–16. ISBN 0-89791-884-3. 7, 9

[MMSK03] Meseth J., Müller G., Sattler M., Klein R.: Btf rendering for virtual environments. In *Virtual Concepts 2003* (November 2003), pp. 356–363. 8

[MN98] Meyer A., Neyret F.: Interactive volumetric textures. In *Rendering Techniques '98 (Proceedings of the Eurographics Workshop on Rendering 98)* (June 1998), Drettakis G., Max N., (Eds.), Eurographics, Springer-Verlag Wien New York, pp. 157–168. 5, 12

[MNP01] Meyer A., Neyret F., Poulin P.: Interactive rendering of trees with shading and shadows. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (2001), Springer-Verlag, pp. 183–196. 8, 11

[MO95] Max N., Ohsaki K.: Rendering trees from precomputed Z-buffer views. In *Rendering Techniques '95* (june 1995), Springer, pp. 45–54. 7, 11

[MS95] Maciel P. W. C., Shirley P.: Visual navigation of large environments using textured clusters. In *1995 Symposium on Interactive 3D Graphics* (Apr. 1995), Hanrahan P., Winget J., (Eds.), ACM SIGGRAPH, ACM Press, pp. 95–102. ISBN 0-89791-736-7. 2, 4, 13

[NRH*77] Nicodemus F. E., Richmond J. C., Hsia J. J., Ginsberg I. W., Limperis T.: *Geometric Considerations and Nomenclature for Reflectance*. Monograph 161, National Bureau of Standards (US), Oct. 1977. 8

[PLAdON98] Popescu V. S., Lastra A., Aliaga D. G., de Oliveira Neto M. M.: Efficient warping for architectural walkthroughs using layered depth images. In *Proceedings of the conference on Visualization '98* (1998), Ebert D., Hagen H.,, Rushmeier H., (Eds.), IEEE Computer Society Press, pp. 211–216. 7, 13

[RAL98] Rafferty M. M., Aliaga D. G., Lastra A. A.: 3d image warping in architectural walkthroughs. In *Proceedings of the Virtual Reality Annual International Symposium* (1998), IEEE Computer Society, p. 228. 7, 13

[RAPL98] Rafferty M. M., Aliaga D. G., Popescu V., Lastra A. A.: Images for accelerating architectural walkthroughs. *IEEE Comput. Graph. Appl. 18*, 6 (1998), 38–45. 13

[RP94] Regan M., Pose R.: Priority rendering with a virtual reality address recalculation pipeline. In *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)* (July 1994), Glassner A., (Ed.), Annual Conference Series, ACM SIGGRAPH, ACM Press, pp. 155–162. ISBN 0-89791-667-0. 9

[Sch95] Schaufler G.: Dynamically generated impostors. In *GI Workshop on Modeling, Virtual Worlds,* (Nov. 1995), Fellner D. W., (Ed.), pp. 129–135. 3, 4, 12

[Sch96] Schaufler G.: Exploiting frame to frame coherence in a virtual reality system. In *Proceedings of the 1996 Virtual Reality Annual International Symposium (VRAIS 96)* (1996), IEEE Computer Society, p. 95. 13

[Sch97] Schaufler G.: Nailboards: A rendering primitive for image caching in dynamic scenes. In *Rendering Techniques '97 (Proceedings of the Eurographics Workshop on Rendering 97)* (June 1997), Dorsey J., Slusallek P., (Eds.), Eurographics, Springer-Verlag Wien New York, pp. 151–162. ISBN 3-211-83001-4. 4

[Sch98a] Schaufler G.: Image-based object representation by layered impostors. In *Proceedings of the ACM symposium on Virtual reality software and technology* (1998), ACM Press, pp. 99–104. 5

[Sch98b] Schaufler G.: Per-object image warping with layered impostors. In *Rendering Techniques '98 (Proceedings of the Eurographics Workshop on Rendering 98)* (June 1998), Drettakis G., Max N., (Eds.), Springer-Verlag Wien New York, pp. 145–156. 5, 11

[SDB97] SILLION F., DRETTAKIS G., BODELET B.: Efficient impostor manipulation for real-time visualization of urban scenery. *Computer Graphics Forum (Proc. Eurographics '97) 16*, 3 (Aug. 1997), 207–218. ISSN 1067-7055. 6, 10

[SGHS98] SHADE J. W., GORTLER S. J., HE L., SZELISKI R.: Layered depth images. In *SIGGRAPH 98 Conference Proceedings* (July 1998), Cohen M., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 231–242. ISBN 0-89791-999-8. 7

[SLS*96] SHADE J., LISCHINSKI D., SALESIN D., DEROSE T., SNYDER J.: Hierarchical image caching for accelerated walkthroughs of complex environments. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), Rushmeier H., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 75–82. 3, 4, 13

[SS96] SCHAUFLER G., STÜRZLINGER W.: A three-dimensional image cache for virtual reality. *Computer Graphics Forum (Proc. Eurographics '96) 15*, 3 (Sept. 1996), 227–235. ISSN 0167-7055. 3, 13

[SS00] SIMMONS M., SÉQUIN C. H.: Tapestry: A dynamic mesh-based display representation for interactive rendering. In *Rendering Techniques 2000 (Proceedings of the Eurographics Workshop on Rendering 2000)* (June 2000), Péroche B., Rushmeier H., (Eds.), Eurographics, Springer-Verlag Wien New York, pp. 329–340. ISBN 3-211-83535-0. 10

[SS01] SEQUIN C. H., SIMMONS M.: Portal tapestries. *The Pennsylvania State University CiteSeer Archives* (Apr. 12 2001). http://citeseer.ist.psu.edu/466318.html. 13

[SvBLD03] SUYKENS F., VOM BERGE K., LAGAE A., DUTRÉ P.: Interactive rendering with bidirectional texture functions. In *Proceedings of the 24th Annual Conference of the European Association for Computer Graphics (EG-03)* (Oxford, UK, Sept. 1–6 2003), Brunet P., Fellner D., (Eds.), vol. 22, 3 of *Computer Graphics forum*, Blackwell Publishing Ltd., pp. 463–472. 8

[TC00] TECCHIA F., CHRYSANTHOU Y.: Real-time rendering of densely populated urban environments. In *Proceedings of the Eurographics Workshop on Rendering Techniques 2000* (London, UK, 2000), Springer-Verlag, pp. 83–88. 12

[TK96] TORBORG J., KAJIYA J.: Talisman: Commodity Real-time 3D graphics for the PC. In *SIGGRAPH 96 Conference Proceedings* (Aug. 1996), Rushmeier H., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 353–364. 9

[TLC02a] TECCHIA F., LOSCOS C., CHRYSANTHOU C.: Image-based crowd rendering. *IEEE Computer Graphics and Applications 22*, 2 (Mar./Apr. 2002), 36–43. 12

[TLC02b] TECCHIA F., LOSCOS C., CHRYSANTHOU C.: Visualizing crowds in real-time. *Computer Graphics Forum 21*, 4 (Nov. 2002), 753–765. 12

[WAA*00] WOOD D. N., AZUMA D. I., ALDINGER K., CURLESS B., DUCHAMP T., SALESIN D. H., STUETZLE W.: Surface light fields for 3D photography. In *SIGGRAPH 2000 Conference Proceedings* (2000), Akeley K., (Ed.), Annual Conference Series, ACM SIGGRAPH, Addison Wesley, pp. 287–296. 8

[Wan04] WANG N.: Realistic and fast cloud rendering. *Journal of Graphics Tools: JGT 9*, 3 (2004), 21–40. 12

[WDG02] WALTER B., DRETTAKIS G., GREENBERG D. P.: Enhancing and optimizing the render cache. In *EGRW '02: Proceedings of the 13th Eurographics workshop on Rendering* (2002), Eurographics Association, pp. 37–42. 10

[WDP99] WALTER B., DRETTAKIS G., PARKER S.: Interactive rendering using the render cache. In *Rendering techniques '99 (Proceedings of the 10th Eurographics Workshop on Rendering)* (New York, NY, Jun 1999), Lischinski D., Larson G., (Eds.), vol. 10, Springer-Verlag/Wien, pp. 235–246. 10

[WGS99] WIMMER M., GIEGL M., SCHMALSTIEG D.: Fast walkthroughs with image caches and ray casting. *Computers and Graphics 23*, 6 (Dec. 1999), 831–838. 10

[WHON97] WONG T.-T., HENG P.-A., OR S.-H., NG W.-Y.: Image-based rendering with controllable illumination. In *Proceedings of the Eurographics Workshop on Rendering Techniques '97* (1997), Springer-Verlag, pp. 13–22. 8

[Wil02] WILSON A. T.: *Spatially encoded image-space simplifications for interactive walkthrough*. PhD thesis, University of North Carolina at Chapel Hill, 2002. 14

[WLY*00] WILSON A., LIN M. C., YEO B.-L., YEUNG M., MANOCHA D.: A video-based rendering acceleration algorithm for interactive walkthroughs. In *Proceedings of the eighth ACM international conference on Multimedia* (2000), ACM Press, pp. 75–83. 6, 10

[WM03] WILSON A., MANOCHA D.: Simplifying complex environments using incremental textured depth meshes. In *Proceedings of ACM SIGGRAPH 2003* (2003), Hodgins J., Hart J. C., (Eds.), vol. 22(3) of *ACM Transactions on Graphics*, pp. 678–688. 3, 6, 7, 14

[WMPM01] WILSON A., MAYER-PATEL K., MANOCHA D.: Spatially-encoded far-field representations for interactive walkthroughs. In *Proceedings of the ninth ACM international conference on Multimedia* (2001), ACM Press, pp. 348–357. 6, 14

[WP95] WEBER J., PENN J.: Creation and rendering of realistic trees. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques* (1995), ACM Press, pp. 119–128. 11

[WW03] WIMMER M., WONKA P.: Rendering time estimation for real-time rendering. In *Proceedings of the 14th Eurographics workshop on Rendering* (2003), Eurographics Association, pp. 118–129. 14

[WWS01] WIMMER M., WONKA P., SILLION F.: Point-based impostors for real-time visualization. In *Rendering Techniques 2001 (Proceedings of the Eurographics Workshop on Rendering 2001)* (June 2001), Myszkowski K., Gortler S. J., (Eds.), Eurographics, Springer-Verlag Wien New York, pp. 163–176. ISBN 3-211-83709-4. 8, 15

[ZPvBG01] ZWICKER M., PFISTER H., VAN BAAR J., GROSS M.: Surface splatting. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* (2001), ACM Press, pp. 371–378. 7