Spatially-Encoded Far-Field Representations for Interactive Walkthroughs

Andrew Wilson Ketan Mayer-Patel Dinesh Manocha

Department of Computer Science
University of North Carolina
Chapel Hill, NC 27599-3175
{awilson,kmp,dm}@cs.unc.edu
http://www.cs.unc.edu/˜geom/Video/SE

Abstract: *We introduce the notion of spatially encoded video and use it for efficiently representing image-based impostors for interactive walkthroughs. As part of a pre-process, we automatically decompose the model and compute the far-fields. The resulting texture images are organized along multiple dimensions and can be accessed in a user-steered order at interactive rates. Our encoding algorithm can compress the impostors size by two orders of magnitude. Furthermore, the storage cost for additional impostors or samples grows sub-linearly. The resulting system has been applied to a complex CAD environment composed of* 13 *million triangles. We are able to render it at interactive rates on a PC with little loss in image quality.*

Keywords: image databases, image-based rendering, MPEG, rendering systems, spatial data structures

1 Introduction

Many applications like computer-aided design (CAD), architectural and urban visualization, flight simulation and virtual environments generate large and complex three-dimensional (3D) models composed of tens of millions of primitives. One of the important problems in computer graphics and virtual environments is to create an interactive system that enables a viewer to experience the synthetic environment by simulating a user-steered display or walkthrough of the model. However, these datasets cannot be directly rendered at interactive rates on current high-end graphics systems.

Different rendering acceleration techniques that reduce or limit the number of geometric primitives rendered in each frame have been proposed in the literature. These include visibility culling, level-of-detail modeling, and use of image-based representations. In this paper, we focus on image-based rendering techniques that have been used to replace subsets of the geometric model with image-based impostors or simplifications. Typically these impostors are used to replace geometry far from a given viewpoint. We refer to an approximation of this distant geometry as a *far-field*. A number of rendering acceleration algorithms and systems based on far-field representations have been developed [2, 3, 4, 7, 8, 18, 25, 26, 27, 28, 31].

Some key issues in using image-based impostors for large 3D datasets are memory storage, bandwidth and sampling. Each sample of reasonable resolution takes a few mega-bytes (MBs) or more. Most algorithms discretize the viewing space and generate impostors from a finite set of viewing directions. This can lead to image artifacts such as popping, cracks, aliasing or stretching. Moreover, the pre-computed far-fields for moderately sized environments (composed of a few million polygons) can easily take a few or tens of giga-bytes [2, 3, 8] and it is hard to fit them into main memory. Many of the image artifacts can be reduced by taking more impostor samples, but that adds to the storage and bandwidth complexity.

Given the storage complexity, different compression techniques have been applied to image-based representations. These include motion estimation and compensation techniques used to encode synthetic or parameterized animations [1, 6, 12, 16, 30]. These have been designed for streaming over a network. The resulting application is different from interactive walkthroughs, where the constraints imposed by faithful response to user spontaneity are different from pre-recorded videos or synthetic animations. Some impostor based rendering acceleration algorithms have used compression techniques based on DCT (JPEG) or Lempel-Ziv (PNG) to reduce the storage complexity of texture images [2, 8, 31]. However, these techniques as well as current standards like MPEG do not utilize spatial relationships and other properties unique to image-based impostors.

Main Results: In this paper, we introduce the notion of *spatially-encoded video* and use it for efficiently representing far-fields for interactive walkthroughs. As part of precomputation, we use spatial partitioning algorithms to automatically decompose the model into rectangular cells and enclose each cell with a cull-box. We approximate the geometry outside each cull-box using image-based impostors and encode them spatially. These frames are accessed in a user-steered order at runtime. They are decoded at interactive rates and rendered using projective texture mapping. The rest the model contained inside the cull-box is rendered as geometry.

The spatially-encoded video representation differs from conventional video streams and compression algorithms in many ways. The frames are organized along multiple dimensions that include position and orientation. Furthermore, our encoding algorithm utilizes spatial coherence between the impostors and model-based depth information. As compared to earlier walkthrough systems that use image-based impostors, our approach offers the following advantages:

1. Automaticity: Our approach for decomposing the model into cells, computing cull-boxes, and encoding impostors is fully automatic. Moreover, it is applicable to all geometric datasets.
2. Storage Efficiency: Our compression algorithm can lower the storage overhead by an order of magnitude as compared to earlier approaches. Moreover, as we generate more samples or impostors, our storage cost grows sub-linearly (as compared to linear growth for earlier approaches).

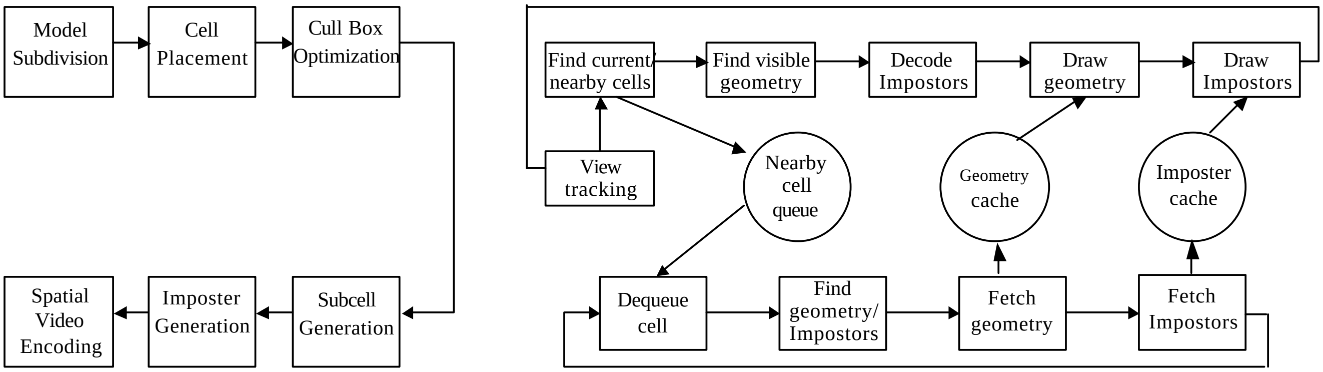

Fig. 1(a): Preprocess

Fig. 1(b): Runtime System

Figure 1: *Overview of our approach. Spatial video techniques are used for impostor representation and rendering.*

3. Reduced Pre-processing Overhead: The spatial coherence and model information reduces the encoding time.

4. Improve Runtime Performance: Our average time to access, decode and render the encoded impostors is reduced and is more predictable. It significantly reduces the load on the pre-fetching algorithm and this results in less variation in the frame rates.

5. Dynamic Impostors: We use an auxiliary index to store interframe relationships in multiple dimensions. This facilitates dynamically adding new impostors.

The resulting algorithm has been implemented on a dual-processor PC with a nVIDIA Quadro2 graphics card. We demonstrate its performance on a large and complex CAD environment composed of more than 13 million triangles. It is able to render the scene at $12 - 25$ frames a second with little loss in image quality.

Overview: The rest of the paper is organized in the following manner. We give a brief survey of related work on image-based impostors and compression techniques in Section 2. We present an overview of our approach in Section 3. We introduce spatial video in Section 4 and show that it can be used to encode different impostor representations efficiently. Section 5 presents the algorithm for decomposing the model into cells and cull-boxes as well as generating the far fields. We describe its implementation and performance on complex CAD environment in Section 6. Finally, we highlight many areas of future research in Section 7.

2 Previous Work

In this section, we give a brief survey on image-based representations, compression algorithms and spatial encoding.

2.1 Image-Based Representations for Interactive Walkthroughs

Image-based representations and impostors have been widely used for faster display of large environments. For example, many flight simulation systems have used images to represent terrains and other specialized models. In many cases these images were hand-generated. More recently, different kinds of far-field representations have been used for rendering acceleration. Different representations, ordered in terms of increasing space and time complexity, include:

1. **Point Samples:** Approximate the geometry with point primitives and render them directly [10, 23].
2. **Cached Images:** Render portions of the scene and cache the resulting images. These images are texture mapped onto planar projections [4, 18, 27, 25, 31].
3. **Texture Depth Meshes (TDMs):** Render the scene onto a planar projection, generate a depth mesh and apply a polygon simplification algorithm to the resulting mesh. At runtime, the simplified mesh is displayed using projective texture mapping [2, 7, 28].
4. **Multi-Mesh Impostors (MMIs):** Consist of multiple layers of textured meshes and limit the dis-occlusion errors [8].
5. **Range or Depth Images:** Consist of per-pixel depth along with the intensity values. They are rendered at runtime using 3D image warping [21, 20, 24]
6. **Layered Depth Images (LDIs)**: Each pixel consists of multiple depth values corresponding to all the intersections of the ray with the scene. They reduce the dis-occlusion artifacts [3, 26].

These representations are either pre-computed from a set of viewpoints or can be dynamically updated [8, 25, 27]. Other dynamic impostor representations are based on compositing individually updated image layers [15]. All of them take considerable memory resources. For example, a full-screen sized image in 24-bit color takes almost three megabytes. Additional space is needed to represent the meshes for TDMs or MMIs and the depth values for range images or LDIs. In practice, most rendering acceleration algorithms trade off storage overhead for frame rate or image fidelity. Our spatial encoding algorithm can compress all these representations, though we mainly focus on cached images and TDMs in this paper.

2.2 Compression

Data compression is a well-studied area. In this section, we briefly survey algorithms for compressing image-based representations and synthetic animations.

Some of the commonly used image compression algorithms include lossless schemes like Lempel-Ziv, transform coding schemes such as JPEG, vector quantization, etc. They have been used to compress textures in a number of software rendering and walkthrough systems [2, 3, 5, 8]. However, these algorithms do not utilize the spatial coherence between adjacent impostors.

Video compression is also a well studied area and standard video compression techniques like MPEG [14] were developed for use on natural scenes. Many algorithms have also been proposed for compressing synthetic animations, where the available model information is used to compute the optical flow

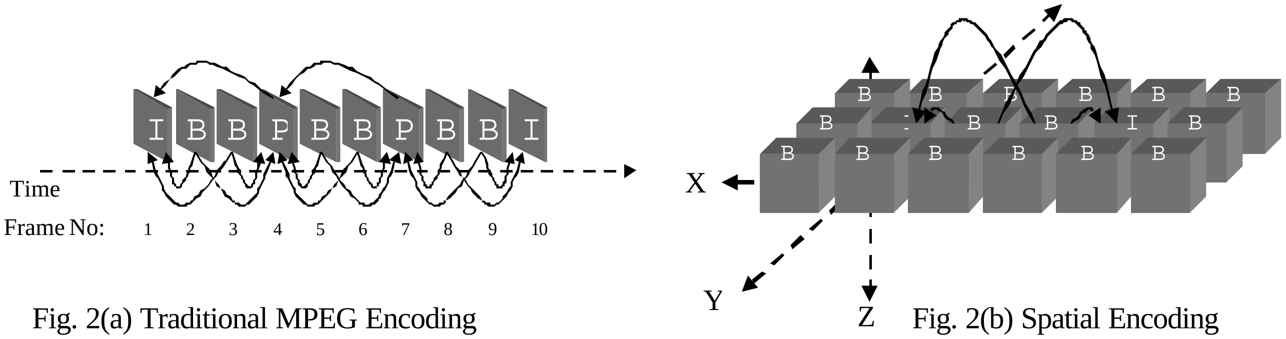

Fig. 2(a) Traditional MPEG Encoding

Fig. 2(b) Spatial Encoding

Figure 2: *Traditional MPEG Encoding vs. Spatial Encoding. The latter represents the impostors along multiple spatial dimensions. They are traversed in a user-steered order at runtime.*

field between successive frames. These include the lossless motion compensation algorithm [11], fast computation of optical flow, and per-block motion vectors using the Gouraud interpolation or texture mapping hardware [30], and accounting for non-translational block motion using a least-squares formulation [1]. Other approaches include partitioning the rendering task between the server and the client and using polygon assisted compression [16] as well as algorithms for texture intensive streaming applications that use pre-computed view-dependent textures [6]. An approach for arbitrary-dimensional parameterized animations that uses the texture mapping rasterization hardware for decoding has been presented in [12]. All of these algorithms were developed for faster transmission of synthetic animations over networks, and utilize temporal coherence between successive frames.

Many techniques have been used for compressing the dual-plane lumigraph parameterizations. These include vector quantization and entropy coding for light-fields [17], wavelet basis [13] and block-based DCT encoders [22]. Gortler et al. [9] have treated lumigraphs as arrays of 2D images and proposed using JPEG and MPEG for intra-frame and inter-frame compression, respectively. Other algorithms include model-based coders for view-dependent texture mapping [19].

3 Algorithm Overview

Our rendering algorithm uses a combination of image-based and geometry representations to render the datasets. The impostors or far-fields are used as a low-cost approximation of the far geometry. The nearby geometry is rendered using a combination of level-of-detail modeling and visibility culling. The performance and fidelity of the rendering algorithm is governed by the choice of image-based impostor. Each impostor representation (except point samples) highlighted in Section 2 is typically generated from a single viewpoint in the scene. As a result, whenever the viewer moves away from that viewpoint, image artifacts are introduced. The only way to minimize these artifacts is to pre-compute more samples or add them dynamically. However, they add considerably to the storage complexity. For example, the memory cost for a TDM that uses a few thousand triangles to represent the mesh and a 512×512 image texture in true color is a few megabytes. For a large environment, we may use tens or hundreds of thousands of cells. Furthermore, we generate multiple impostors for each cell, thereby resulting in a few million impostors for the entire model.

We use spatially encoded cached images or TDMs as impostor representations. Their main benefit arises from the fact that they can efficiently use the texture-mapped polygon rasterization hardware. Furthermore, as we generate more samples, the storage overhead for additional impostors grows sub-linearly. This allows us to pre-compute impostors from multiple viewpoints within each cell. It also reduces the load on the pre-fetching algorithm because of low storage complexity. The pre-processing and runtime phases of the algorithms are shown in Fig. 1.

Pre-process: (Fig. 1(a)) We use cells and cull-boxes to classify the model into near and far geometry [2]. Our algorithm automatically decomposes the environment into rectangular cells using spatial partitioning algorithms and encloses each cell with a rectangular cull-box. The partitioning algorithm takes into account maximum deviation error in the impostor representation from any viewpoint in the cell and based on that subdivides some of the cells into sub-cells. All the sub-cells of a given cell, share the same cull-box. It also bounds the maximum geometry contained in the cull-box for each cell. Finally, it computes six far fields for each cell and sub-cell.

Runtime System: (Fig. 1(b)) The algorithm tracks the cell that contains the viewpoint. It maintains separate geometry and impostor caches and uses pre-fetching algorithms to load the geometry and impostors. The pre-fetching algorithms run asynchronously and use spatial coherence to determine which geometry and impostors to load. The impostors corresponding to the particular cell or sub-cell are decoded and rendered using projective texture mapping. Furthermore, we use visibility culling and select an appropriate level-of-detail for each object contained in the cull box. If a second rendering pipeline is available, it is used to generate additional impostors and encodes them asynchronously. They can be dynamically added to the set of far-field representations.

4 Spatially Encoded Video

The rendering algorithm uses either cached images or TDMs as far-field representations. In either case, the texture images dominate the storage cost, which is up to $85 - 95\%$ of the entire database. The simplest compression algorithms treat each image separately, and use DCT (JPEG) encoding. In practice, we achieve $5 - 20$ times compression with JPEG. As we generate more samples, the storage overhead grows almost linearly. However, the far-field impostors corresponding to adjacent cells or sub-cells exhibit strong image or spatial coherence. In this section, we present new spatial encoding algorithms that utilize this coherence and achieve an order of magnitude improvement in the compression ratios as well as faster encoding and decoding performance.

Traditional video encoding schemes like MPEG have been designed for frame sequences that are organized along a single temporal dimension. The compression techniques attempt to exploit inter-frame coherence by using one frame to predictively encode the next one (as shown in Fig. 2(a)). Algorithms for synthetic or parameterized animations make use of model information to compute the optical flow between successive frames, that improves the performance of the compression algorithm.

We introduce the notion of *spatially-encoded video* and use it for encoding the far-fields for interactive walkthroughs. As compared to the prior work on video compression, the spatially-encoded video differs in the following manner:

1. **Multi-dimensional Representation:** The video frames are generated from a camera moving in space. However, there is no single camera path, but instead the frames correspond to samples of a multi-dimensional real world or synthetic environment. The different dimensions may correspond to the position and orientation of the camera. For dynamic environments, we can also add the time dimension.
2. **Random Access:** No assumptions are made regarding the order in which frames are accessed at runtime. They will typically be used in an interactive or user-steered application, where nothing is known *a priori* about user's motion.
3. **Interactive Decoding:** Given the interactive nature of the underlying application, it is important to decode the frames at interactive rates.
4. **Dynamic Updates:** New frames or images can be generated dynamically or on-the-fly. The underlying representation should be able to add them as encoded frames.

For interactive walkthroughs, we present a new spatial encoding scheme that extends the traditional MPEG standard.

4.1 Traditional MPEG encoding

In this section, we give a short overview of MPEG compression scheme [14, 29] as it is usually applied to video sequences. The scheme encodes the video frames as one of three types: I, P, or B. I-frames contain all information required to decode the frame. Each frame is subdivided into 16 × 16 pixel regions called macroblocks. Each macroblock is encoded using the Discrete Cosine Transform (DCT). P-frames are predictively encoded using the previous I- or P-frame. Each macroblock in a P-frame may be associated with a motion vector that defines a region in the reference frame of the same size to be used as a predictive base. The difference between the prediction region and the actual pixel values is DCT encoded. B-frames are predictively encoded in the same way using both the previous I- or P-frame as well as the subsequent I- or P-frame. Fig. 2(a) shows a typical pattern for I, P, and B frames and their referential relationships.

4.2 Multi-dimensional Impostor Representation

The far-fields used in the walkthrough system are generated by taking finite samples of the virtual environment. The camera position corresponds to the center of each cell or sub-cell. We generate a texture image for each face of the cull box and the camera's orientation and field of view is determined by each face separately (as shown in Fig. 5 in Section 5). This results in a $4D$ representation of frames, given by the $3D$ cell position and $1D$ orientation of the frames within each cell. Moreover, each impostor is "near" several other impostors in 4 different dimensions. Other sampling strategies or spatial decompositions of the environment can result in different multi-dimensional representations.

Given that the user can move in an arbitrary direction at runtime, we do not know in advance the order in which the impostors will be accessed. As a result, the compressed representation of the far fields must have support for random access. Traditional I-, P-, and B- frame sequences in MPEG-encoded streams do not support random access. Furthermore, the reference relationships between encoded frames need to reflect the underlying multi-dimensional relationship of the impostors.

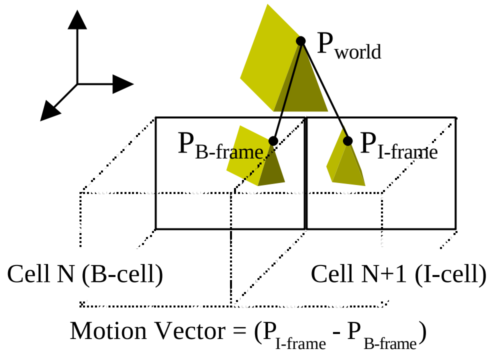

Figure 3: *The motion vector associated with $P_{B-frame}$ is calculated by back projecting the screen coordinates into world space (P_{World}) using depth information and then reprojecting P_{World} into screen coordinates for the reference frame ($P_{I-frame}$). The difference between these coordinates is a candidate motion vector for encoding the macroblock containing $P_{B-frame}$.*

To accommodate these requirements, we extend the traditional MPEG encoding scheme in three important ways.

1. We avoid using P-frames altogether.
2. We construct and store an auxiliary index which explicitly defines reference information.
3. We encode impostors by organizing them in a 4-D space and define reference relationships within that space.

The auxiliary index data structure supports random access by resolving reference relationships without having to scan through and decode the compressed representations in any particular order.

4.3 Spatial Encoding

Given our representation, we use the notion of I-cells. An I-cell is simply a cell for which all of the far-field impostors are encoded as I-frames. Surrounding each I-cell in all directions are B-cells. The far-field impostors associated with a B-cell are encoded as B-frames. Each impostor encoded as a B-frame uses the two I-frame encoded impostors in the same orientation associated with the two nearest I-cells. Thus, each I-cell is associated with 26 B-cells. Fig. 2(b) illustrates the organization of I-cells and B-cells.

4.3.1 Encoding Heuristics

A traditional video encoder is not provided with any depth information associated with the video frames. Thus, the heuristics used by available encoders for motion vector calculation are all based on some sort of search. On the other hand, our encoding algorithm uses the model and depth information. When encoding an impostor as a B-frame, the spatial encoder is provided with the depth of each pixel as well as the camera information associated with the impostor and both reference frames. To encode a macroblock, the depth associated with each pixel in the macroblock is used to calculate a candidate motion vector associated with that pixel. This is illustrated in Fig. 3, where the difference $P_{I-Frame} - P_{B-Frame}$ is a candidate motion vector for encoding the macroblock of the B-frame that contains the point.

The back projection produces 256 candidate motion vectors for each 16 × 16 macroblock for each reference frame. Because many of the pixels of a macroblock will correspond to the same object in the virtual environment, many of the 256 motion vectors will be identical since they represent the relative spatial motion of that object. Thus, the number of motion vectors to test is

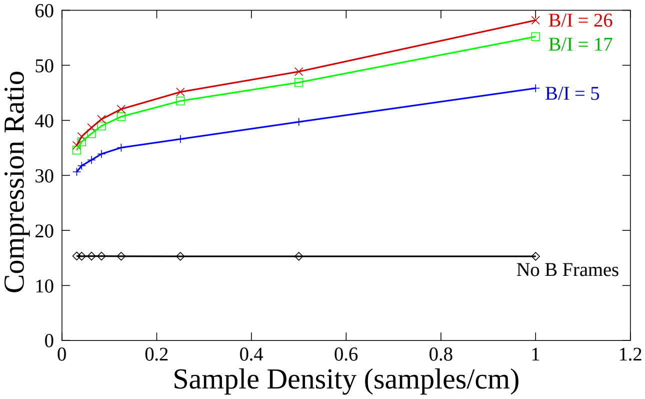

Figure 4: *Average compression ratios as a function of the sampling density and ratio of B-frames to I-frames*

kept small which speeds up encoding considerably. Traditional search heuristics do not use the depth information for finding motion vectors. Given the same limited number of candidate vectors, they will be constrained to a very small search area.

4.4 Interactive Decoding

The organization of I-cells and B-cells in our system guarantees that no more than 2 reference frames will be required to decode any particular B-frame. Since the frames are organized along spatial dimensions, decoded reference frames are cached and used to decode other nearby frames as the user interactively moves through the model. Given a user-specified direction, the access pattern for I- and B-frames is similar to a traditionally encoded MPEG stream using only I- and B-frames. By storing the multidimensional reference relationships between frames in an auxiliary index, we are able to locate each required frame in constant time. Once a frame is located and reference frame data obtained (either from cache or as a result of decoding), time required to decode a frame is the same as that for a traditional MPEG decoder.

4.5 Representing Dynamically Generated Impostors

Many rendering acceleration algorithms augment the precomputed far fields with dynamically generated impostors [8]. Typically, they use a second rendering pipeline and generate additional far fields from new viewpoints within a cell. For each additional viewpoint, our algorithm computes a sub-cell and generates the impostors corresponding to the faces of the cull-box. The center of the new sub-cell and the orientation within that sub-cell are used to add it to the multi-dimensional data structure. Since the position of the encoded representation is no longer entangled with the reference relationships, the new far-field can be added to the auxiliary index.

4.6 Compression Efficiency

The compression efficiency of our encoder is determined by two factors: the ratio of B-frames to I-frames and the degree of spatial coherence between the B-frames and their reference frames. Because each I-frame is independently encoded and does not exploit predictive encoding, for a given quality level, the size of the I-frames represents a lower bound on the size of the encoded impostors. The degree to which B-frames are compressed is highly content dependent. If the far-field impostors exhibit strong spatial coherence, the size of B-frames in the system decreases because the effectiveness of the predictive encoding increases. In general, this is related to the cell sizes or sampling density associated with the impostors. A higher sampling density (i.e. smaller cells), will result in greater coherence. Additionally, higher sampling densities also allow the ratio of B-frames to I-frames to be increased while maintaining acceptable quality since the number of B-frames that exhibit sufficient coherence with an I-frame increases.

Fig. 4 shows the results of an experiment designed to illustrate these tradeoffs. We generated a set of impostors from a large virtual environment at different sampling densities. The graph in Fig. 4 shows the average compression ratio achieved for different B/I ratios, where B is the number of B-frames coded between consecutive I-frames. We notice that for a given B/I ratio, compression efficiency increases with sampling density. The rate of increase is *super-linear*. Moreover, the second derivative of the curve relating compression ratios to the sampling density decreases. Finally, for any given sampling density, compression ratios increase with B/I.

Let s be the sampling density and $I(s)$ be the storage overhead of all the impostors generated by the system. It follows from the graph, that $I^{'}(s) > 0$ and $I^{''}(s) < 0$. This implies:

1. $I(s)$ is a *sub-linear* function. As we generate more samples, the average storage cost per impostor decreases.
2. When s is small, the rate of decrease in the average impostor size is the highest.

This fits well in the context of walkthrough applications. Typically, we don't use a very high sampling rate, otherwise prefetching will become a bottleneck. As we generate more samples to reduce the image artifacts, the storage costs increases at a slower rate.

5 Model Decomposition

Our algorithm partitions the environment into *virtual cells* and places a *cull box* around each cell [2]. At runtime, cells and cull-boxes are used to classify the scene into near and far geometry. Objects which intersect or are contained in the cull box associated with that particular cell are labeled as "near geometry". The remainder of the model outside the cull-box is classified as "far geometry". One of the most important issues in impostor-based walkthroughs are the viewpoints used for pre-computing the far-fields. In our case, the algorithm computes six impostors for each cell and sub-cell (as shown in Fig. 5). In practice, the databases have an uneven geometric distribution. Computing an optimal cell-decomposition is shown to be very hard in practice [2]. For example, if we want an absolute bound on the dis-occlusion artifacts, the algorithm may have to compute the aspect graph of the model which can take $O(m^6)$ time (m is the number of primitives). Some algorithms have been presented for special cases, when the user motion is constrained along a line and the primitives are extruded 2D objects [8].

We use a simple and adaptive approach for computing cells, sub-cells and cull-boxes. It proceeds as:

- Compute a uniform grid of rectangular cells within the model. This grid can be fairly coarse, although a denser sampling improves the results achieved by the spatial video encoder.
- Construct a cull-box for each cell. The cull-box is a cube and its center is the same as that of the cell. Choose the cull box size to be as large as possible while forcing the size of the near geometry to lie beneath some ceiling (e.g. $100,000$ polygons). The algorithm has only one degree of freedom in terms of adjusting the size of the cull-box and it performs binary search.
- If the original cell contains more geometry than our upper limit, we uniformly subdivide a cell into sub-cells and perform cull-box optimizations for each of them.

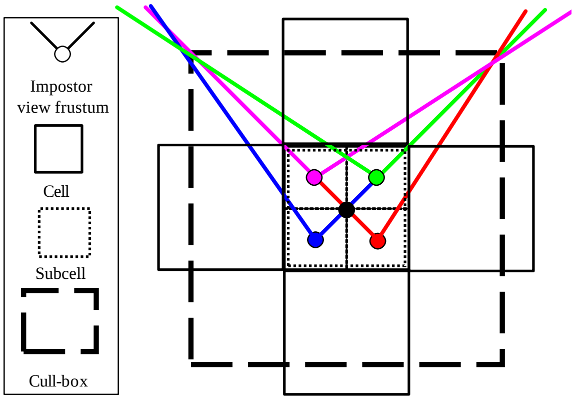

Figure 5: *Each cell is contained within a large cull box. A cell may be further partitioned into sub-cells. Impostors are sampled in six directions in 3D (north, south, east, west, up, down) from the center of each cell and sub-cell. In this figure, view frusta for the north face of each sub-cell are shown in different colors.*

- Subdivide cells into sub-cells to improve the impostor fidelity. We use different error metrics based on the relative size of the cell (or sub-cell) to its cull-box, the resolution of the texture images, and maximum pixel deviation error in the skins introduced by TDMs.

We share one copy of the model geometry among all the cells and sub-cells. The potentially visible set for each cell is computed and stored during preprocessing or computed at runtime using hierarchical view frustum culling.

5.1 Far Field Generation

The far fields in our system correspond to either cached images or TDMs. The TDM computation starts with a range image, identifies the almost-planar regions, and simplifies each height field [2, 28]. This process can result in "skins" wherever there is significant depth discontinuity in the original range image.

The use of impostors results in many image artifacts during runtime. These include stretching, aliasing and dis-occlusion artifacts as the viewer moves within the same cell (or sub-cell) and popping as the viewer jumps from one cell (or sub-cell) to an adjacent cell (or sub-cell). The three factors governing the quality of impostors are as follows:

1. The size s of each cell or sub-cell. At runtime, the viewer can be at most $\frac{1}{2}s$ away from any viewpoint used to generate the impostor.
2. The relative size of the cull-box to the cell size. Based on this ratio, the algorithm computes the maximum angular distortion for an object in the distant geometry and the maximum pixel deviation in the impostor. A smaller ratio leads to more artifacts.
3. The underlying impostor representation and the resolution of the texture image. For example, a TDM reduces the popping as compared to a cached image, when the viewer jumps between cells.

The first two parameters are taken into account by our adaptive sub-cell decomposition algorithm. More samples increase the spatial coherence between different impostor representations.

6 Implementation and Performance

In this section we describe a walkthrough system which implements the algorithms outlined in this paper. We highlight its performance on a complex CAD database and compare our work with earlier approaches.

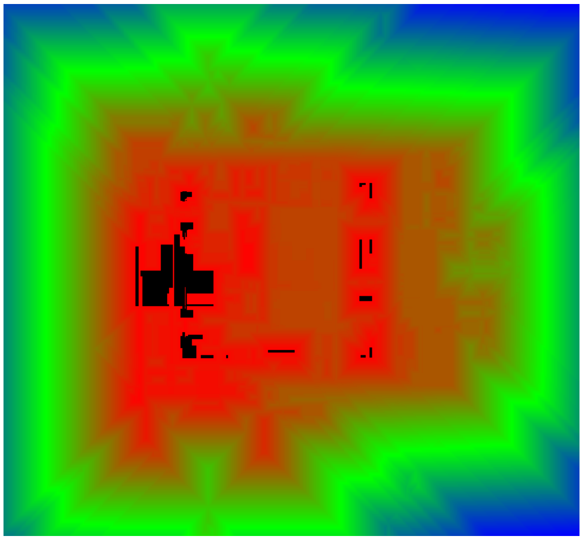

Figure 6: *Performance of cull-box optimization algorithm. The size of the cull-boxes is represented in the following color order: black < red < green < blue. The black areas represent regions with very high density of geometry with very small cull-boxes (and vice-versa for blue).*

6.1 Implementation

Preprocessing We first subdivide the model into chunks of roughly 1000 polygons apiece. We then apply the cell generation algorithm described in Section 5 to generate the cells, subcells, and cull boxes throughout the model. Given this cell grid, we render six samples of the environment and encode them spatially.

Runtime Walkthrough The interactive portion of our walkthrough system, illustrated in Fig. 1(b), is organized as two independent processes: a rendering process and a prefetching process. The rendering process is responsible for handling user interaction, locating nearby cells as the user moves through the model, and rendering both geometry and impostor data corresponding to the user's viewpoint. The prefetching process is responsible for making sure that the data needed by the rendering process is available in main memory before it is rendered. The two processes communicate through a cell queue. If the data corresponding to the current cell is not available, the rendering thread will pause and load the missing items.

6.2 Spatial Encoding and Interface

The modifications required by our spatial encoding scheme are not compatible with existing MPEG encoders and decoders. We have developed software-only implementations of both the spatial encoder and spatial decoder. The use of MPEG as a starting point was motivated by its open standard definition, proven effectiveness for video compression, and wide acceptance within the multimedia community.

We have developed a software library that defines an interface to an impostor database which contains encoded impostors and the auxiliary index used to define the reference relationships. Each impostor in the database is associated with an index number that is calculated based on its spatial position and orientation. The encoder is written as a stand-alone program that can encode a particular impostor either as an I-frame or a B-frame. The decoder is implemented in the software. It provides a simple interface to retrieve encoded the impostors from the impostor database and decodes them. The decoder also implements a LRU caching policy of the decoded impostors to avoid decoding reference frames more than once.

6.3 Performance and Results

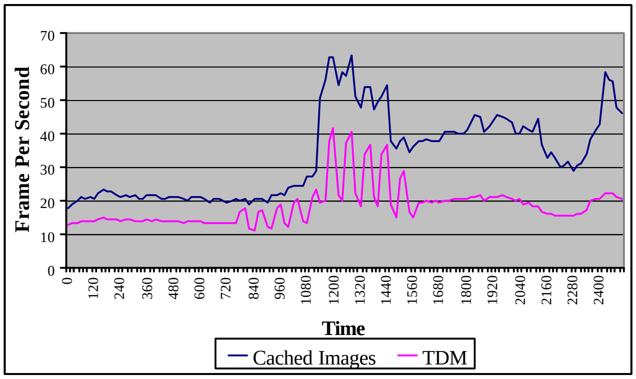

Figure 7: *This graph highlights the frame rates through a sample path (shown in the video). We use spatial encoding to represent cached images as well as TDMs.*

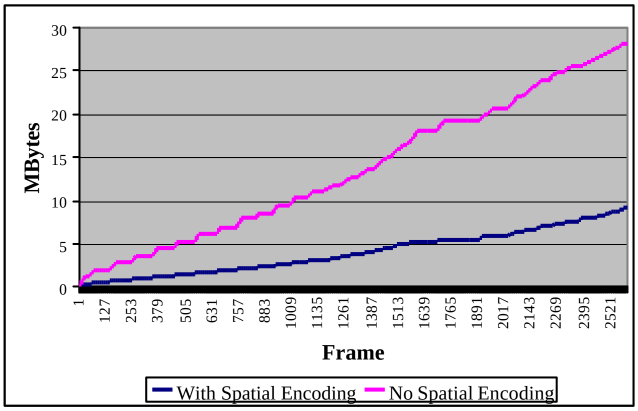

Figure 8: *This graph highlights the memory efficiency achieved by spatial encoding on the sample path shown in Fig. 7. It shows the total data fetched from the disk at run time.*

We have tested our system on a PC equipped with two 1GHz Pentium III processors and an nVIDIA Quadro 2 graphics card. The system itself was written in C++ and uses OpenGL for rendering.

We used a geometric model of a coal-fired powerplant to test the performance of our algorithms. This environment has a footprint 50 meters wide by 60 meters deep by 80 meters tall and contains roughly 13.2 million triangles (as shown in Fig. 9). It contains $1,877$ objects (before subdivision) and occupies 502 MB on disk. The power plant poses challenges to both geometric and image-based rendering acceleration algorithms. In particular, roughly half of the polygons in the model are devoted to densely packed arrays of long, thin cylindrical pipes. These pipes are difficult to simplify with level-of-detail algorithms, as they are coarsely tessellated and the polygons are very long and thin. They also cause considerable challenges in terms of using image-based samples, as a small change in the viewpoint changes the occlusion relationship between different objects (as shown in the video). Fig. 10, which was taken inside the power plant, contains a lot of high-frequency information that can pose problems for image compression schemes. Even rendering the power plant using only the original geometry shows significant aliasing in the displayed frames.

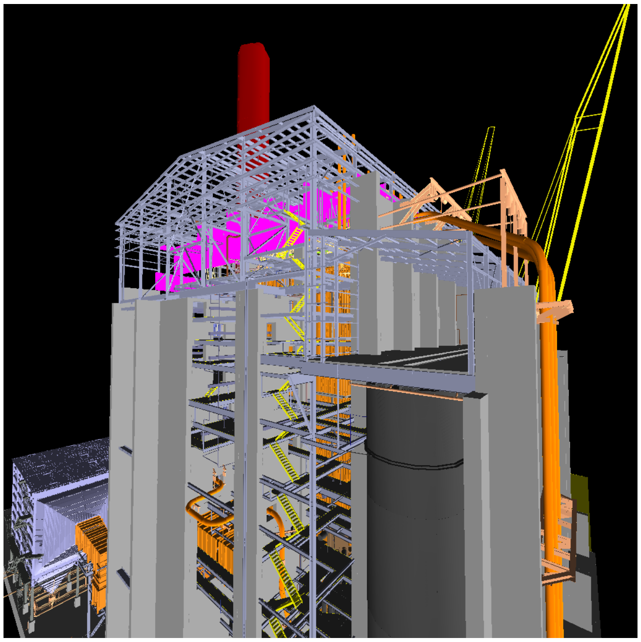

Figure 9: *The Powerplant Model:13.2 million triangles*

We generated a cell grid for a region near the top of the power plant. Cell centers were initially spaced at 1-meter intervals. We chose a budget of $100,000$ polygons for optimizing cull box sizes. Due to the highly non-uniform distribution of geometry within the power plant, optimized cull box sizes range from 1.5 meters on a side (in the middle of the arrays of pipes) to roughly 25 meters on a side (outside the building proper). Fig. 6 illustrates the distribution of cull box sizes. After sub-cell generation and cull-box optimization, we had $3,776$ cells and $6,464$ sub-cells (or $10,240$ cells). The sub-cells were generated to meet the criterion that each cull-box is at least four times larger than the size of its associated cell. The sampled images of the environment generated from the center of each cell had a resolution of 512×512 (down-sampled from 1024×1024 images).

We arrange the cell grid in memory as a graph. Each cell has links to its neighbors, its sub-cells (if any), and its parent cell (if a sub-cell). The entire cell grid occupies roughly 1.5MB in memory. We also precompute the potentially visible set for each cell at a cost of 680KB of memory. The indexing structures for our spatially encoded impostors requires 960KB of memory at runtime. We generated cached images and textured depth meshes (TDMs) for a few sample paths within the model (as shown in the video). These meshes were simplified from the original impostor depth fields ($511 \times 511 \times 2 = 522,242$ triangles) to a maximum of $10,000$ triangles apiece. TDMs can be prefetched at runtime along with model geometry and impostor images.

6.3.1 Compression and Image Fidelity

Our system takes about 16 hours to generate all the impostors and spatially encodes them. It takes 359 MB to represent $22,656$ impostors and the average size of an encoded 512×512 full color texture image is less than $16K$. The performance of the encoding algorithm varies on different texture images. During the encoding process, each 16×16 macroblock had about 7 distinct candidate vectors (among 256 vectors corresponding to each pixel). This implies that our algorithm is able to capture the

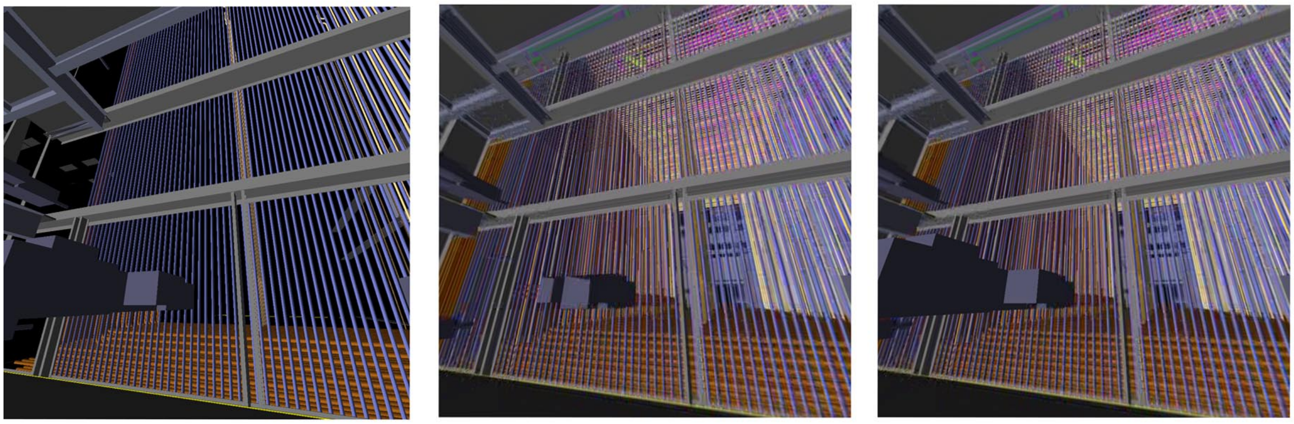

Fig. 10(a): Only Near Geometry

Fig. 10(b): Only Far Fields

Fig. 10(c): Far-Fields + Near Geometry

Figure 10: *This figure shows how far-fields are used to represent distant geometry (on the left), near geometry rendered as polygons (in the middle) and the combined geometry + impostors (on the right).*

spatial or object-to-object coherence and it eliminates the search step. The average compression ratio obtained by the encoding algorithm across the entire model is about 48. However, the variance is high and the ratio varies in the range $2:1$ to $303:1$.

Our encoding algorithm is not lossless. Like other encoders [14], it uses a constant quality factor with every macroblock. Moreover, its performance is content dependent. In general, if the original frame does not have high frequency components, the encoder produces little degradation that is not noticeable (as shown in the video). In the context of the powerplant model, we have opted for a slightly degraded compression ratio in exchange for good image fidelity. Our lowest compression ratio occurs in situations like the one in Fig. 10(a). The image contains many high-frequency components that are difficult to handle with video-based encoding techniques.

6.4 Runtime Performance

Our spatially encoded impostor representation has allowed us to achieve frame rates between 12 and 35 frames per second on a PC using the power plant model. Fig. 7 shows a graph of frame rates along a pre-recorded sample path through the model. When we render this same path using only visibility culling, we observe frame rates on the order of one frame every $10-14$ seconds. As we generate more samples, there are fewer disocclusion artifacts and it also reduces the popping problem.

The video highlights the performance of our system on different portions of the powerplant. We have also compared its performance with an implementation that does not use spatial encoding (i.e. 5 times more memory to represent the far-fields). There is little additional loss in image fidelity when we use spatial encoding.

We find that the prefetching task is not a bottleneck for our system for reasonable user velocities (up to 3 meters/second). Fig. 8 shows the cumulative number of bytes fetched along a sample path. The average impostor size in this case is less than $16K$, if we use spatial encoding, and about $49K$ if we use JPEG compression for each far-field.

6.5 Comparison with Earlier Approaches

Our spatial encoding algorithm can also be used for other impostor based rendering acceleration algorithms, as highlighted in Section 2. It will either reduce the storage overhead or let us generate more samples in the same amount of storage. In terms of interactive walkthrough of large datasets, spatial video can improve the performance of the following algorithms and systems:

- **MMIs:** The MMIs based acceleration algorithm combines pre-generated and dynamically updated impostors into a single framework [8]. It has been applied to city walkthrough, where the pre-computed impostor database is larger than $2GB$. However, the storage requirement for impostors is a major issue [8]. Our spatial encoding algorithm can utilize the spatial coherence between different MMIs, as well as among multiple layers of textured meshes within the same MMI.
- **Guaranteed Frame Rate with LDIs:** Aliaga and Lastra [3] replace portions of the model using LDIs. However, their system assumes that the entire impostor database (about $4GB$) fits into main memory. Moreover, the size of the impostor databases affects the performance of their system [3]. Our spatial encoding algorithm can be applied to the intensity values of each LDI and reduce its storage cost by a factor of two or three. However, no good techniques are known for compressing the depth information.
- **Cell Based Walkthrough System:** The pre-computed impostor database in the MMR system takes more than 10 GB and pre-fetching the data at runtime is a major bottleneck [2]. Our spatial encoding reduces the storage overhead and the compression ratios on different impostors vary in the range $2-15$ (average compression ratio is 5). It lowers the load on the prefetching process. Our algorithm for cell and cull-box decomposition algorithm is automatic. and we use an adaptive scheme to decompose the cell into sub-cells and improve the impostor fidelity. For example, the “skin” artifacts that arise in TDMs are considerably reduced.

Wilson et al. [31] have presented a variation of the MMR system, where they used cached images as impostors and arranged them as a one dimensional array. They used traditional MPEG encoding on the resulting stream of images, by treating it as a temporal sequence. Our approach utilizes the spatial coherence between different impostor representations and results in better compression ratios as compared to the algorithm presented in [31]. Furthermore, our system has fewer image artifacts.

7 Conclusions and Future Work

In this paper, we have introduced spatial video and applied it to encode the far fields for interactive walkthroughs. We extended the traditional MPEG encoding algorithm and showed that our multi-dimensional representation can be used for interactive applications. We have demonstrated its performance on a complex CAD environment composed of more than $13M$ triangles and we are able to render it at $15 - 25$ frames on a PC. Spatial video encoding can either lower the storage overhead or reduce the popping artifacts. In practice, the storage cost for additional impostors grows sub-linearly.

There are many avenues for future work. We would like to improve the encoding algorithm so that it results in fewer artifacts, especially when the images have high frequency components. This will involve more investigation of the relationship between the quality factor and compression rate. Another area of research is depth encoding. Our current scheme, based on video compression techniques, is only applicable to the image portion of a far-field impostor.

Our algorithm generates more samples because of spatial encoding. However, the rendering system has some popping artifacts when the user switches between cells. We would like to explore blending techniques to reduce the popping. We would also like to apply our rendering algorithm to other synthetic environments including terrains and urban models as well as dynamic datasets. An exciting area of research is to apply this approach to real-world models, including light-fields and lumigraphs as well as range datasets. These are captured from known spatial locations and spatial video techniques can be used to represent them efficiently.

References

[1] M. Agrawala, A. Beers, and N. Chaddha. Model-based motion estimation for synthetic animations. *Proceedings of ACM Multimedia 95*, 1995.

[2] D. Aliaga, J. Cohen, A. Wilson, H. Zhang, C. Erikson, K. Hoff, T. Hudson, W. Stuerzlinger, E. Baker, R. Bastos, M. Whitton, F. Brooks, and D. Manocha. Mmr: An integrated massive model rendering system using geometric and image-based acceleration. In *Proc. of ACM Symposium on Interactive 3D Graphics*, 1999.

[3] D. Aliaga and A. Lastra. Automatic image placement to provide a guaranteed frame rate. In *Proc. of ACM SIGGRAPH*, 1999.

[4] D. G. Aliaga. Visualization of complex models using dynamic texture-based simplification. In *Proc. of Visualization'96*, pages 101–106, 1996.

[5] A. Beers, M. Agrawala, and N. Chaddha. Rendering from compressed textures. *Proc. of ACM SIGGRAPH*, 1996.

[6] D. Cohen-Or, Y. Mann, and S. Fleishman. Deep compression for streaming texture intensive animations. *Proceedings of SIGGRAPH 99*, pages 261–268, 1999.

[7] L. Darsa, B. Costa, and A. Varshney. Walkthroughs of complex environments using image-based simplification. *Computer and Graphics*, 22(1):55–69, 1998.

[8] X. Decoret, G. Schaufler, F. Sillion, and J. Dorsey. Multi-layered impostors for accelerated rendering. *Computer Graphics Forum*, 18(3), 1999.

[9] S. Gortler, R. Grzeszczuk, R. Szeliski, and M. Cohen. The lumigraph. In *Proc. of ACM SIGGRAPH*, pages 43–54, 1996.

[10] J. Grossman and W. J. Dally. Point sample rendering. *Eurographics Workshop on Rendering*, pages 181–192, 1998.

[11] B. Guenter, H. Yun, and R. Mersereau. Motion compensated compression of computer animation frames. In *Proc. of ACM SIGGRAPH*, pages 297–304, 1993.

[12] Z. Hakura, J. Lengyel, and J. Snyder. Parameterized animation compression. *Proc. of 11th Eurographics Workshop on Rendering*, pages 101–112, 2000.

[13] P. Lalonde and A. Fournier. Interactive rendering of wavelet projected light fields. *Proc. of Graphics Interface*, pages 107–114, 1999.

[14] D. Legall. A video compression standard for multimedia applications. *Communications of the ACM*, 34(4):46–58, 1991.

[15] J. Lengyel and J. Snyder. Rendering with coherent layers. *Proc. of ACM SIGGRAPH*, pages 233–242, 1997.

[16] Marc Levoy. Polygon-assisted JPEG and MPEG compression of synthetic images. In *SIGGRAPH 95 Conference Proceedings*, pages 21–28, 1995.

[17] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH 96 Conference Proceedings*, pages 31–42, 1996.

[18] P. Maciel and P. Shirley. Visual navigation of large environments using textured clusters. In *ACM Symposium on Interactive 3D Graphics*, pages 95–102, 1995.

[19] M. Magnor and B. Girod. Model-based coding of multi-viewpoint imagery. *SPIE Conference on Visual Communications and Image Processing*, pages 14–22, 2000.

[20] W. Mark, L. Mcmillan, and G. Bishop. Post-rendering 3d warping. *Symposium on Interactive 3D Graphics*, pages 7–16, 1997.

[21] N. Max and K. Ohsaki. Rendering trees from precomputed Z-buffer views. In *Eurographics Rendering Workshop 1995*, 1995.

[22] G. Miller, S. Rubin, and D. Poncelen. Lazy decompression of surface light fields for pre-computer global illumination. *Proc. of Eurographics Workshop on Rendering*, pages 281–292, 1998.

[23] H. Pfister, M. Zwicker, J. van Baar, and M. Gross. Surfels: Surface elements as rendering primitives. *Proc. of ACM SIGGRAPH*, 2000.

[24] M. Rafferty, D. Aliaga, and A. Lastra. 3d image warping in architectural walkthroughs. *IEEE VRAIS*, pages 228–233, 1998.

[25] G. Schaufler and W. Sturzlinger. A three dimensional image cache for virtual reality. *Computer Graphics Forum*, 15(3):C227–C235, 1996.

[26] J. Shade, S. Gortler, Li wei He, and R. Szeliski. Layered depth images. *Proc. of ACM SIGGRAPH*, pages 231–242, 1998.

[27] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder. Hierarchical image caching for accelerated walkthroughs of complex environments. In *Proc. of ACM SIGGRAPH*, pages 75–82, 1996.

[28] F. Sillion, G. Drettakis, and B. Bodelet. Efficient impostor manipulation for real-time visualization of urban scenery. In *Computer Graphics Forum*, volume 16, 1997.

[29] International Telecommunications Union. Generic coding of moving pictures and associated audio information. *ITU-T Recommendation H.262*, 1995.

[30] D. Wallach, S. Kunapalli, and M. Cohen. Accelerated MPEG compression of dynamic polygonal scenes. In *Proc. of ACM SIGGRAPH*, pages 193–197, 1994.

[31] A. Wilson, M. Lin, D. Manocha, B. Yeo, and M. Yeung. Video-based rendering acceleration algorithms for interactive walkthroughs. *Proc. of ACM Multimedia*, pages 75–84, 2000.