A Video-Based Rendering Acceleration Algorithm for Interactive Walkthroughs

Andrew Wilson, Ming C. Lin, Dinesh Manocha
Department of Computer Science
CB 3175, Sitterson Hall
University of North Carolina at Chapel Hill
Chapel Hill, NC 27599
{awilson,lin,dm}@cs.unc.edu

Boon-Lock Yeo, Minerva Yeung
Intel Corporation
Microcomputer Research Labs
2200 Mission College Blvd.
Santa Clara, CA 95052
minerva.yeung@intel.com

http://www.cs.unc.edu/~geom/Video

Abstract

We present a new approach for faster rendering of large synthetic environments using video-based representations. We decompose the large environment into cells and pre-compute *video based impostors* using MPEG compression to represent sets of objects that are far from each cell. At runtime, we decode the MPEG streams and use rendering algorithms that provide nearly constant-time random access to any frame. The resulting system has been implemented and used for an interactive walkthrough of a model of a house with 260,000 polygons and realistic lighting and textures. It is able to render this model at 16 frames per second (an eightfold improvement over simpler algorithms) on average on a Pentium II PC with an off-the-shelf graphics card.

Keywords
Massive models, architectural walkthrough, MPEG video compression, virtual cells, video-based impostors

1 Introduction

One of the fundamental problems in computer graphics and virtual environments is interactive display of complex environments on current graphics systems. Large environments composed of tens of millions of primitives are frequently used in computer-aided design, scientific visualization, 3D audio-visual and other sensory exploration of remote places, tele-presence applications, visualization of medical datasets, etc. The set of primitives in such environments includes geometric primitives like polygonal models or spline surfaces, samples of real-world objects acquired using cameras or scanners, volumetric datasets, etc. It is a major challenge to render these complex environments at interactive rates, i.e. 30 frames a second, on current graphics systems. Furthermore, the sizes of these data sets appear to be increasing at a faster rate than the performance of graphics systems.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
ACM Multimedia 2000 Los Angeles CA USA
Copyright ACM 2000 1-58113-198-4/00/10...$5.00

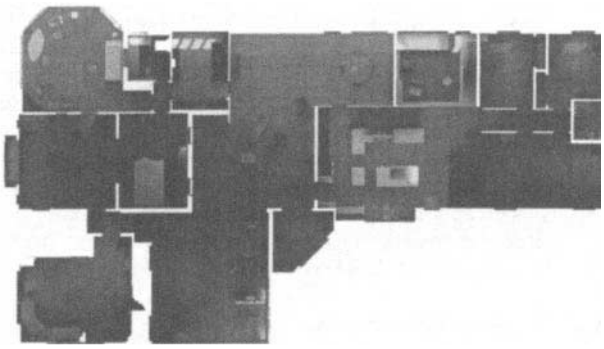

Figure 1: CAD database of a house with realistic lighting and texture. The model has over 260,000 polygons and 19 megabytes of high-resolution texture maps. This model is too large to be naively rendered at interactive rates.

One of the driving applications for interactive display of large datasets is *interactive walkthroughs*. The main goal is to create an interactive computer graphics system that enables a viewer to experience a virtual environment by simulating a walkthrough of the model. Possible applications of such a system include design evaluation of architectural models [Brooks86,Funkhouser93], simulation-based design of large CAD datasets [Aliaga99], virtual museums and places [Mannoni97], etc. The development of a complete walkthrough system involves providing different kinds of feedback to a user, including visual, haptic, proprioceptive and auditory feedback, at interactive rates [Brooks86]. Real-time feedback as the user moves is perhaps the most important component of a satisfying walkthrough system. This faithful response to user spontaneity is what distinguishes a synthetic environment from precomputed images or frames, which can take minutes or even hours per frame to calculate, and from pre-recorded video. In this paper, we focus on the problem of generating visual updates at interactive rates for complex environments.

There is considerable research on rendering acceleration algorithms to display large datasets at interactive frame rates on current graphics system. These algorithms can be classified into three major categories: visibility culling, multi-resolution modeling, and image-based representations. However, no single algorithm or approach can successfully display large datasets at interactive rates from all viewpoints. Some hybrid approaches that have been investigated use image-based representations to render "far" objects [Maciel95,Shade96,Aliaga96,Aliaga99] and geometric representations for "near" objects [Cohen97,Erikson99,Garland97,Hoppe96]. Commonly used image-based representations include texture maps, textured depth meshes, layered depth images [Aliaga99] etc. However, in terms of application to large models, these image-based representations have the following drawbacks:

- *Sampling:* Most of the algorithms take a few finite samples of a large data set. No good algorithms are known for automatic generation of samples for a large environment.
- *Reconstruction:* Different reconstruction techniques have been proposed to reconstruct an image from a new viewpoint. While some of them do not result in high fidelity images, others require special purpose hardware for interactive updates.
- *Representation and Storage:* A large set of samples takes considerable storage. No good algorithms are known for automatic management of host and texture memory devoted to these samples.

1.1 Main Contribution

In this paper, we present a method for accelerating the rendering of large synthetic environments using video-based representations. Video-based techniques have been widely used for capture, representation and display of real-world datasets. We propose the use of *video based impostors* for representing synthetic environments and rendering these scenes at interactive rates on current high-end and low-end graphics systems. We use a cell-based decomposition of a synthetic environment and associate a *far field* representation with each cell. For each cell, we generate a sequence of far-field images and MPEG compress them using offline encoding. At runtime, we decode the MPEG streams and utilize algorithms that provide nearly constant time random access to any frame, for displaying them. The frames are selected as a function of the viewpoint. We address a number of issues in cell generation and the use of encoding and decoding algorithms, then demonstrate how to combine these algorithms with multi-resolution representation and visibility culling for interactive display. The resulting system has been implemented and used for an interactive walkthrough of a model of a house with realistic lighting and textures. We have tested our system on a PC using off-the-shelf graphics hardware and achieve an average update rate of 16 frames per second in a 260,000-polygon model of a house 31 meters wide by 18 meters deep and 5 meters tall. This update rate represents a significant improvement over simpler rendering algorithms.

Organization: The rest of the paper is organized in the following manner. We survey related work in Section 2 and present our approach in Section 3. Section 4 highlights a number of implementation issues. We describe our system's performance in Section 5. Finally, we highlight areas for future work in Section 6.

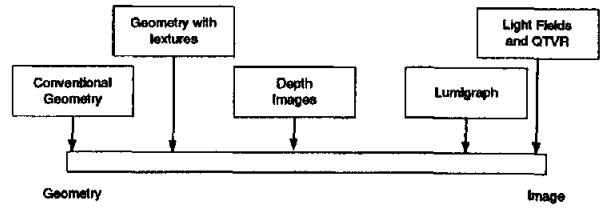

Figure 2: Continuum of representations of an environment, from purely geometric to purely image-based [Lastra99]

2 Related Work

In this section, we briefly survey related work on rendering acceleration techniques and the use of image-based and video-based representations for rendering real and synthetic environments.

2.1 Interactive Display of Large Datasets

There are two basic types of models commonly used for rendering large data sets: geometric representations (based on a description of the surfaces in the model) and image-based representations. Many hybrid combinations have also been proposed. Based on these representations, different rendering acceleration techniques have been proposed; examples include multi-resolution modeling, visibility culling, and use of image-based representations.

2.1.1 Geometric Models

By far the most common class of model representations is geometric, where *surfaces* in the model are described using polygons or curved primitives. This representation is used for CAD, visual simulation, and most scientific applications. The basic geometric representation, as it is commonly used, stores each surface only once. There is no notion of appropriate resolution, except for textures, which are commonly pre-filtered. More advanced representations keep several levels of detail for objects and select the correct level at run time [Funkhouser93]. Originally, these levels of detail were created manually. In recent years, the problem of automatic generation of levels-of-detail has received considerable attention in computer graphics, vision, and computational geometry. [Cohen97, Erikson99, Garland97, Hoppe96].

2.1.2 Image-Based Representations

There has also been work on the use of images to represent complex, but distant, portions of models at an appropriate resolution [Maciel95,Shade96,Aliaga96]. These algorithms use a surface-centric representation and use image-based impostors for distant geometry. These images have been used for rendering acceleration. However, the use of images introduces a sampling problem: how many samples are needed for high-fidelity rendering? Other image-based representations include camera-centric forms [Chen93,McMillan95,Gortler96,Levoy96]. In

Figure 2, we show these on an axis representing how much geometric information is used in these representations.

The image-based representations store a measure of the amount of light arriving at a point in space, with perhaps some information about the surface. The commercial Quicktime VR representation [Chen95] relies on panoramic images. A user cannot move freely around the space, but can only rotate. The Light Field representation [Levoy96] stores images in a four dimensional data structure representing rays between two finite planes. The Lumigraph [Gortler96] is similar to the Light Field, but can use some geometric information. Images with per-pixel depth [Chen93,McMillan95] organize information as images, but locate the samples in 3D space.

2.1.4 Visibility Culling

Besides multi-resolution models and image-based representations, other rendering acceleration algorithms are based on visibility culling. A particularly fruitful case has been in the architectural model domain because of the partitioning of space into rooms and doors (referred to in the literature as *cells and portals*) [Airey90,Teller91]. For general environments, [Greene93,Zhang97] have presented algorithms that use a combination of object space and image space hierarchies. [Aliaga99] have presented an approach that combines image-based representations, levels of detail, and occlusion culling for large geometric environments.

2.2 Combining Graphics with Video

In addition to the use of image-based representations, many researchers have proposed techniques to use video for rendering real and synthetic environments. There is considerable work on techniques using interpolations among images to create visual continuity during motion or other changes within three-dimensional virtual spaces [Chen93,Boyd98]. Other combinations of graphics and video include virtual sets, where live actors can move within computer-generated settings [Katkere97]. Carraro et al. [Carraro98] have highlighted techniques to incorporate video displays into virtual environments in the context of a multi-user simulator. The MPEG-4 proposal [MPEG4] allows specification of data displays as compositions of video and graphical objects.

2.3 Video for Multimedia Applications

There is extensive work on the use of video for multimedia applications. These include video conferencing, video-on-demand systems, internet video [Patel98], visual effects [Millerson90], etc. Most of these applications involve capturing videos of real-world scenes, then storing and organizing them efficiently using a combination of encoding and decoding algorithms [MPEG2, Kender97, Zhang93, Shen95, Yeung97, MPEG4].

3 Overview

In this section we give a brief overview of our approach. We describe an algorithm for rendering acceleration using virtual cells, which allows us to guarantee a minimum frame rate for an interactive walkthrough by rendering nearby objects as geometry and replacing distant ones with a simple video-based impostor. Finally, we explain the application of MPEG compression to these impostors.

3.1 Cell-Based Walkthrough

Architectural models of complex environments such as power plants, naval vessels, aircraft, and high-rise buildings are typically too large to render naively at interactive rates on current graphics hardware. Such models often contain anywhere from a few hundred thousand to a few hundred million polygons, divided into objects numbering in the hundreds or thousands (at least). We must reduce the number of primitives rendered at each frame in order to bring a system's performance up to an interactive frame rate of between 20 and 30 frames per second. In complex environments with large open spaces, the rendering acceleration techniques highlighted in Section 2 are not sufficient. We perform interactive walkthroughs of such environments by partitioning the model into regions that contain a bounded number of primitives, then rendering only those primitives contained in the user's current region at runtime. These regions are called *cells* within the model. Cell-based walkthrough originated with the method of cells and portals in architectural models and was generalized to *virtual cells* by [Aliaga99].

3.2 Cells and Portals

Architectural models often exhibit an intrinsic spatial subdivision: individual rooms (*cells*) connected by doorways or windows (*portals*) in otherwise opaque walls. Many architectural environments contain large, crowded spaces where the potentially visible set within a single cell is larger than the rendering budget. In these environments as well as outdoor scenes, we can apply the method of *virtual cells* to achieve rendering acceleration.

3.3 Virtual Cells

We generalize the concept of cells and portals to yield *virtual cells*. The space within an environment where the user might wish to travel is partitioned into cells using some convenient subdivision (regular grid, octree, etc.). Each cell is assigned a large *cull box* which is concentric with the cell itself. The purpose of the cull box is to divide the model into a *near field*, whose contents will not exceed the per-frame rendering budget, and a *far field* consisting of the rest of the model. The walls of the cull box correspond loosely to the walls of a room in traditional cells-and-portals. Thus, the potentially visible set for a particular cell consists of only those objects that intersect the cull box. We replace the far field with an inexpensive *video-based impostor* created as part of an offline process of cell generation. By varying the sizes of the cull boxes for different cells, we can enforce an upper bound on the number of primitives that must be rendered for any viewpoint in any cell.

Virtual cells are generated by first determining which parts of the environment will be explored by the user, choosing sample points within that region of exploration, and finally constructing a rectangular cell around each sample point. Increasing the sampling density generally yields increased fidelity at runtime at the expense of increased storage requirements and preprocessing time. The "volume of interest" is computed by subdividing regions indicated by the user. Another possible approach is to

```
for each cell:
   place viewpoint at center of cell
   set field of view so view frustum intersects cell edges
   for each direction (up, down, north, south, east, west):
      set view dir = $direction
      clip away portions of model in front of cull-box wall
      render remaining geometry
      read back frame buffer and save as image
   end
end
```

Figure 3: Algorithm for generating far-field images once cells have been generated. The resulting images are MPEG-compressed for fast runtime access.

use a Delaunay triangulation of the free space and use it to generate the virtual cells. Figure 4 shows the relationship between a cell and a cull box in our system.

3.4 Video-Based Impostors

To avoid having to render geometry for more than one cell at a time, we replace the far field with an opaque, image-based impostor at runtime. This impostor should have the following properties:

- It should be easy and fast to create.
- It should have a compact representation.
- It should closely approximate the appearance of the replaced objects.
- It should be inexpensive to render.

Several different kinds of impostors have been explored, including flat, textured quadrilaterals, textured depth meshes, layered depth images, and light fields [Aliaga99]. Each has its own tradeoffs in terms of fidelity vs. storage space and the cost of reconstruction. For simplicity, we can use flat, textured quadrilaterals. While they exhibit undesirable perspective artifacts when a piece of geometry may cross between the near and far field, they are easy to generate and impose little rendering or reconstruction load on the system at runtime.

The fidelity of the impostor to the geometry being replaced is similar to the problem of sampling and reconstruction in the context of image-based representations. The algorithm takes a finite number of samples of the environment from locations fixed during preprocessing. At runtime, these samples are used to reconstruct the appearance of the portion of the model captured by impostors. The performance of the algorithm varies considerably as a choice of these samples.

3.4.1 Creating Impostors

Image-based impostors are constructed as an offline preprocess. Once cells have been generated for a particular model, we employ the algorithm in Figure 3 to acquire six images of the far field for each cell. These images will be used at runtime as texture maps for the faces of the cull box. The OpenGL near clipping plane is used to remove portions of the model that fall inside the cull box. A typical resolution for the far-field images is 512x512 in 24-bit color, which requires 4.5 megabytes of storage for each cell during preprocessing. Since a typical environment will have hundreds or thousands of cells, these images must be compressed as part of preprocessing and decompressed on demand at runtime. We apply MPEG compression to exploit coherence between the far field images for adjacent cells.

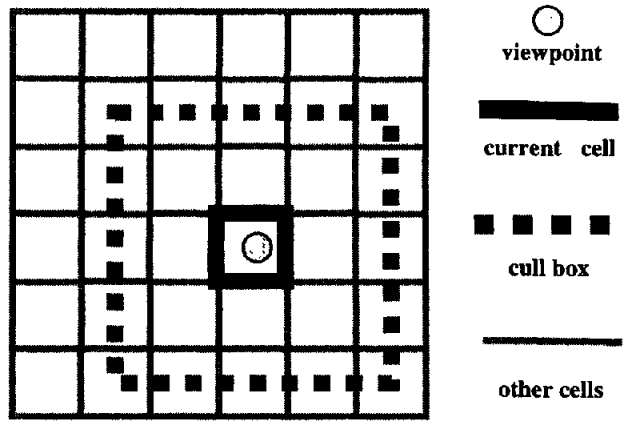

Figure 4: A sample cell grid. The cull box for the cell containing the viewpoint is shown. At runtime, all objects outside the cull box will be replaced with video impostors.

3.4.2 Video Compression of Impostors

The images used to compute impostors exhibit considerable coherence from cell to cell. By arranging the 3-dimensional cell structure into a one-dimensional list, we can impose an ordering on the cells and arrange their far-field images into a linear stream. This stream is amenable to compression using video techniques. As an example, consider a path through the model as shown in Figure 10. If the impostors from the north face of each cull box are arranged in a stream, they depict a constant-velocity pan through the environment. Furthermore, all of the objects represented in each image are some minimum distance away from the camera (typically a few meters in models between 20 and 80 meters on a side) due to the size of the cull box. This ensures that object motion due to depth parallax is small enough to be easily handled with motion prediction.

3.5 Offline encoding

Once all the images of the far field are available, we arrange them into linear streams for encoding using video techniques. The result of this encoding is the video-based representations we use to replace distant portions of the model at runtime.

3.5.1 Mapping Cells to Streams

There are many possible ways to arrange a 3-dimensional array as a 1-dimensional list. To maximize the benefits of video compression, we choose a mapping with the following properties:

- Consecutive entries in the list exhibit coherence.
- Changes from one image to the next can be accurately estimated using motion prediction.
- Adjacent cells in space often map to adjacent entries in the list.

In our initial implementation, we have chosen to treat our cell grid as an array. Rows of cells are aligned with the X axis in the model's space, and columns with the Y axis. We map between our cell structure and a 1D stream by arranging the two-dimensional array of cells in row-major order. Each of the six faces of the cull box is used to generate a separate stream of images.

3.5.2 Choice of Encoding Algorithm

In order to achieve maximum efficiency from our video impostors, we chose an encoding scheme that exploits the temporal coherence present in its input stream and is easily and cheaply accessible at runtime. We have chosen MPEG-2 compression, as it provides a satisfactory balance between these constraints. Moreover, hardware and software tools for fast access and manipulation are easily available.

3.5.3 Encoding Parameters

There are three parameters in the MPEG encoding process that govern the performance of the algorithm [MSSG]. First, the encoder allows us to request a particular bit rate for the encoded stream. Since the data is retrieved from a disk at runtime, we set this parameter to be no greater than the bandwidth available from disk to host memory. Secondly, we can constrain the search space for motion vectors in adjacent frames. Since the source images are of a static environment, the only motion is due to camera parallax. Third, we choose the structure of a group of pictures so that the discontinuities when the 1D stream "wraps around" the 2D model are encoded as intra frames. This same technique could be applied to other discontinuities, such as when the 1D stream passes through a wall inside the model.

4 Implementation

In this section, we describe an implementation of our algorithm. Our system assumes that the environment is given as a collection of (possibly texture-mapped) polygons, and that the user has specified a method for constructing cells. We divide our system into two phases: preprocessing, during which cells are created and video-based impostors are generated and compressed, and runtime, during which the user is allowed to walk through the environment. At runtime, MPEG manipulation tools are used to decompress the impostors on demand.

We use a model of a house with realistic lighting and texture for our architectural environment. The house model was constructed from the blueprints of a real house in Chapel Hill, contains some 260,000 polygons, and uses approximately 19 megabytes of textures acquired from the real house using a digital camera. Our system is implemented in C++, uses OpenGL for rendering, and runs under Windows NT.

4.1 Tools for MPEG manipulation

In this section, we give a brief overview of tools used for encoding and decoding.

4.1.1 Offline encoding

We use the freely available MPEG Software Systems Group encoder [MSSG] to generate MPEG-2 streams from the source images of each cell's far field. The encoding parameters are modified according to the cell structure we impose upon the model. In particular, we attempt to place intra frames wherever the viewpoint moves through a wall or "wraps around" to the other side of the model as a result of the 2D-to-1D cell mapping.

4.1.2 Runtime decoding

To decode the MPEG streams containing far-field images at runtime, we use MPL (MPEG Processing Library), a software library developed at Intel Microcomputer Research Labs [Yeo00]. It provides general-purpose, high performance software APIs for MPEG decoding and processing. It is targeted at applications beyond standard decoding and display. MPL offers convenient random access to different levels of an MPEG bitstream, from bits and motion vectors to full frames.

MPL supports both MPEG-1 and MPEG-2 at resolutions up to HDTV (1920x1280) and is optimized with MMX™ and SSE™ technology. Some of its advanced features include random access to any frames with near constant-time access, fast extraction of encoded frames, simultaneous decoding of multiple MPEG sequences, flexible input plug-ins, SMP support and access via callbacks to non-frame-level information in the MPEG bitstream (e.g. raw bits, blocks, macroblocks, GOP and slice, etc.). We used MPL due to its high-speed random-access capability and its ability to handle multiple streams simultaneously.

MPL's random access and backward playback capabilities are enabled by the use of index tables. After a video is created, an index table is created that maps out the frame dependencies and byte offsets of the I, P, and B frames. For instance, to access frame number N, the index table is used to identify the closest I-frame numbered N or smaller; thereafter, MPL decodes from that I-frame to retrieve frame N. The size of the index table is typically less than 1% of the entire MPEG file size. Backward playback is handled as a special case of random access.

Table 1 shows the forward, backward and random access decoding speed on a low-cost Pentium® III 400 MHz PC. As shown in the table, backward decode and random access speed of MPEG1 video at 352x240 resolutions and bit rate of 1.5 Mbps is at about 60 frames/sec, which is more than sufficient for displaying video typically captured at 30 frames/sec.

Table 1: Performance of MPL on a PIII 400MHz PC. Playback rates are given in frames per second.

Video Type	Width	Height	Forward	Backward	Random Access
MPEG1 1.5 Mbps	352	240	272.9	58.6	57.2
MPEG2 5.0 Mbps	704	480	53.7	11.8	11.7
MPEG2 10 Mbps	1280	720	22.0	4.9	4.8

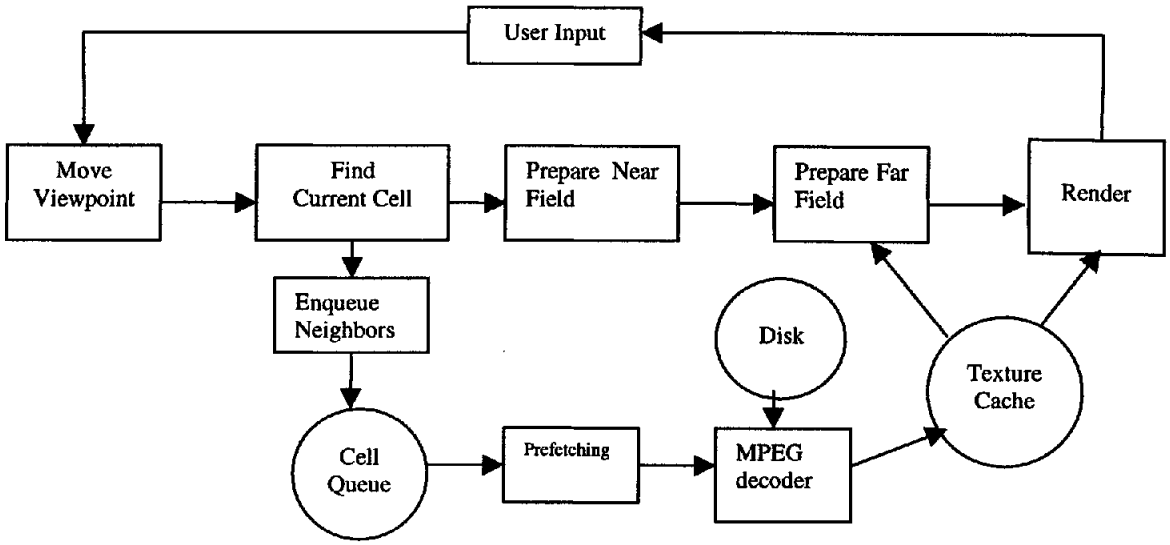

Figure 6: Runtime architecture for interactive walkthrough system. The view-management task communicates with the prefetch task through the cell queue and the texture cache.

Using MPL, we are able to support the following interactions: stop (at any frame), start or resume (from any frame), constant-speed backward playing, constant-speed forward playing and jump to any other video stream at any specified frame.

4.3 Cell structure

We construct a cell grid for the house environment by dividing its two-dimensional bounding box into squares 1 meter on a side. Each of these squares corresponds to a single cell extending from the floor of the model up through the ceiling. The cull box for these cells is 3 meters on a side. This implementation uses only a single layer of cells. However, there is nothing in our method or our system preventing us from using a truly 3-dimensional cell structure. Figure 9 shows an overhead view of the house model with cell boundaries drawn in red. The house model is 31 meters by 18 meters and is 5 meters tall. Rows of cells are aligned with the X axis in model space, and columns with the Y axis. The cell grid contains 558 cells arranged in 18 rows of 31 cells each.

4.4 Preprocessing

Our preprocessing phase, shown in Figure 5, consists of cell generation, far-field rendering, and creation of the MPEG streams that contain the compressed impostor textures. We render and store the video impostors at a resolution of 512x512 in 24-bit color. After all of the impostors have been generated, we compress them as MPEG streams, as described in sections 3.4.2 and 3.5. One stream is generated for each of the six faces of the cull box. Each video stream contains 558 frames (one for each cell in the model).

4.5 System Pipeline

The architecture of the runtime portion of our system is shown in figure 6. We have divided the system's function into two separate tasks, view management and prefetching. View management consists of the actual rendering as well as user

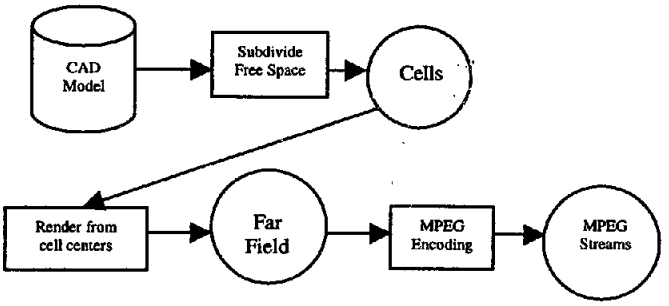

Figure 5: Preprocessing pipeline for generating cells and video streams from a CAD model.

interaction and object and texture preparation. The prefetching task is responsible for decompressing the video impostors for nearby cells. We describe each of these tasks in more detail below.

4.5.1 View Management Task

The view management task, implemented as a single thread, is responsible for generating the image the user sees each frame. It performs four functions:

- Manage user input, including rendering state, navigation mode, and motion of the viewpoint through the model.
- Request far-field textures for nearby cells from the prefetching task.
- Retrieve texture data from the prefetching task and bind it to OpenGL texture memory for rendering.
- Render the model from the user's viewpoint, including the far-field impostors.

The user can switch at will between four navigation modes: trackball (rotate the entire model), drive (move forward and

backward, turn left and right), translate (move up and down and side-to-side), and look (remain stationary and change the view direction). Navigation input is collected and applied at the beginning of each frame to minimize latency between user input and program response.

The view management task is also responsible for informing the prefetching task of nearby cells that the user might visit soon. This is accomplished by updating a nearby-cell queue whenever the user crosses a cell boundary. When this happens, the identifiers of the four cells that share a face with the (new) current cell are placed into the queue.

We have implemented the view management task within a single thread in order to avoid costly OpenGL context switches and maintain synchronization with user input. If the far-field textures for the current cell are not yet available when the view management task is ready to render a frame, it pauses and places requests for those textures at the head of the prefetching queue. The alternative, which some users may prefer, is to render a frame with an incomplete far field.

4.5.2 Prefetching Task

The prefetching task is responsible for making sure that the video impostors for both the current and nearby cells are available in memory. We implement it as a free-running process that takes as its input the cell identifiers in the nearby-cells queue. As each cell identifier is dequeued, it is checked against the texture cache. If the video impostors for that cell are already resident, no further work needs to be performed. If not, the relevant frames are decoded from each of the six MPEG streams and placed into the texture cache. The prefetching task does not actually bind these textures in OpenGL. Since this is a time-consuming operation, it is not performed as part of prediction. The view management task is left to bind textures on demand. We have implemented the prefetching task as a single thread on a uniprocessor machine, and multiple threads (to permit multiple MPEG frames to be decoded simultaneously) on multiprocessor machines.

4.6 Memory Management

When working with massive models, host memory is often a scarce resource. It is quite common for the model itself to occupy anywhere from tens of megabytes to tens of gigabytes of storage space, and for the image-based representation to be several times that size. The problem is even worse when the video-based impostor incorporates texture maps, as texture memory on current PC graphics cards is often limited in size and slow to access.

We address this problem by treating the different storage areas as caches for model and texture data. Main memory is divided into two areas: one for the model geometry and associated texture maps, and one for the decompressed far-field representations. We manage the texture cache using a least-recently-used (LRU) replacement policy. OpenGL texture memory is handled in a similar fashion, but is a much more limited resource: a typical PC graphics card may have 32MB of texture memory, more than half of which is occupied by the texture data for the model itself. It is possible to lower this memory requirement by separating textured objects from the rest of the model and binding the appropriate texture maps only when those objects are present in nearby cells.

5 Performance and Results

In this section, we describe the performance of an architectural walkthrough system implementing our algorithms. We have tested our system on a PC running Windows NT with 256MB of memory, a Pentium II™ processor running at 400 MHz, and an Intergraph Intense3D graphics card. Our geometric environment consists of a realistic model of a house containing some 45 megabytes of geometry and 19 megabytes of high-resolution texture data.

5.1 Overall Rendering Acceleration

We demonstrate the speedup achieved by our method by showing the polygon count and frame rate for a fixed path through the house model both with and without cell-based culling. Our method is able to maintain a frame rate between 10 and 20 frames/second in the house model. Naïve rendering is consistently slower than 5 fps for most views inside the house.

5.2 Breakdown of Time Per Frame

In Table 3, we show the amount of time our system spends on various tasks during each frame. These times are averaged over the duration of the same sample path through the model as in section 5.1.

5.3 Preprocessing

Table 2 shows a breakdown of time and resources spent on our preprocessing phase. Both the acquisition of far-field images and subsequent MPEG encoding to form video impostors can be easily parallelized. The encoding process can make use of as many graphics pipelines as are available. Runtime decoding is generally CPU-bound and benefits from a multiprocessor machine.

Table 2: Time and space requirements for each stage of preprocessing. These vary in direct proportion to the number of virtual cells in the model.

Preprocessing Stage	Time	Disk Space
Cell creation	<1 minute	30Kb
Impostor generation	21 min	2511MB
MPEG encoding	123 min	61 MB

Table 3: Breakdown of average frame time by task. Prefetching of textures happens in a separate thread and is not included.

Task	Avg. time per frame
Cell update	<1ms
Texture binding	32ms
Rendering	40ms
Total Frame Time	73ms

5.4 Analysis of Results

Our system is able to maintain an upper bound on the number of polygons rendered in any particular view of the model, as shown in Figure 7. This is a major step toward guaranteeing a minimum frame rate. However, we found that binding texture data in OpenGL is unexpectedly expensive on our PC graphics card. The regular downward spikes in Figure 8 are pauses between cells while the system binds the texture data for a new cell's video impostors. By comparison, the actual decoding of video impostors using MPL involves negligible computational overhead.

We also encountered problems at times matching colors between the model and the decoded video impostors, as can be seen in Figure 13. These appear to be due to the fact that different color-conversion formulae were used to convert from RGB to YUV space (during MPEG encoding) and from YUV back to RGB (during decoding).

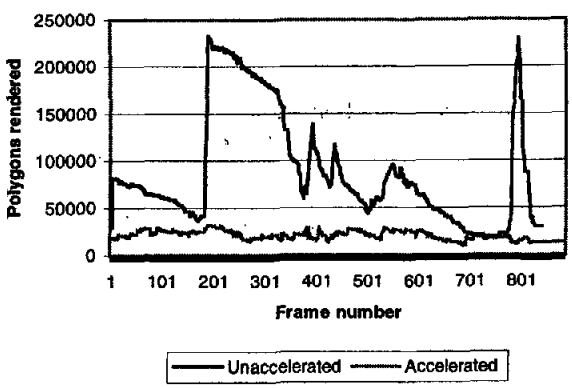

Figure 7: Polygon counts along a sample path, with and without acceleration. Our method imposes an upper bound of roughly 30000 polygons for any view of the model.

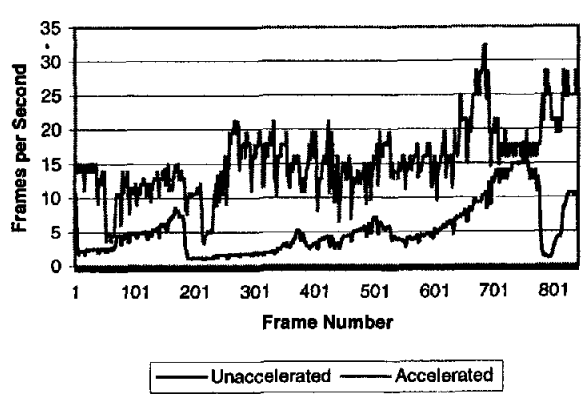

Figure 8: Frame rates along a sample path, both with and without video-based acceleration. Our system achieves an average frame rate of 16 frames per second.

6 Conclusions and Future Work

We have presented an algorithm for accelerating interactive walkthroughs of architectural environments by replacing portions of the model with video-based impostors. We have demonstrated this algorithm on a textured, radiositized model of a house. Our method achieves frame rates 8 times higher than is possible with naïve rendering on common hardware. Although this does not fully meet our goals for interactivity, it represents a considerable improvement over previous methods and suggests that future improvements can possibly yield an update rate of at least 30 frames per second. In terms of implementation and application, we dealt with several issues, including the following:

- compensating for color quantization artifacts in the video compression process
- managing limited texture memory and low host-to-graphics-pipe bandwidth
- choosing a sample density that gives acceptable results without excessive preprocessing overhead

We are exploring the following issues related to the sampling and storage of video impostors:

- Fully automatic generation of cells based upon free space within the model, including higher sample density in regions of interest. It may be possible to compute a Voronoi subdivision of free space and extract paths of maximum clearance to guide cell creation. Crowded areas of the model are natural targets for smaller cells and hence denser sampling of the far field.
- Better mappings from a 3D cell grid to a 1D stream of impostors. Video compression techniques will give better results if there are fewer discontinuities (as when the viewpoint passes through a wall) in the input stream. A mapping which stays within open regions of the model for as long as possible is more useful.
- Modifications to the encoding process to take advantage of the 3D source environment. Since we generate our video impostors from a synthetic environment using known camera parameters, it should be straightforward to estimate motion vectors during image generation rather than search for them during the encoding process. We are also investigating encoding schemes based on a 3D cell structure instead of a 1D stream of images. Such encodings could provide more efficient decoding and allow the use of impostors with fewer inherent artifacts.

7 Acknowledgements

We wish to thank some of the people whose assistance has been instrumental in this work. Matt Holliman of Intel's Media and Graphics Lab provided much-needed insight into problems with color quantization and compression. We are particularly grateful to the UNC Walkthrough team for allowing us access to the CAD model of the house and for providing infrastructure and model-translation assistance. We thank Bob Liang at Intel for inviting the first three authors to the MGL group during the summer of 1999. This research was also supported in part by

ARO contract DAAG55-98-1-0322, a DOE ASCI Grant, NSF grants NSG-9876914, DMI-9900157 and IIS-9821067, an ONR Young Investigator Award, and NIH Research Resource Award 2P41RR02170-13.

8 References

[Airey90] J. Airey, J. Rohlf, and F. Brooks, "Towards image realism with interactive update rates in complex virtual building environments", In *Proc. of ACM Symposium on Interactive 3D Graphics*, 1990, pp. 41--50.

[Aliaga96] D. G. Aliaga, "Visualization of Complex Models Using Dynamic Texture-based Simplification", *IEEE Visualization '96*,October 1996.

[Aliaga99] D. Aliaga et al., "MMR: An integrated massive model rendering system using geometric and image-based acceleration", *Proc. of ACM Symposium on Interactive 3D Graphics*, April 1999.

[Boyd98] J. Boyd, E. Hunter, P. Kelly, L. Tai, C. Phillips and R. Jain, "MPI-Video Infrastructure for Dynamic Environments", to appear in *IEEE International Conference on Multimedia Systems 98*.

[Brooks86] F. Brooks. Walkthrough: A dynamic graphics system for simulating virtual buildings. In *ACM Symposium on Interactive 3D Graphics*, Chapel Hill, NC, 1986.

[Carraro98] G. Carraro, J. Edmark and J. Ensor, " Techniques for Handling Video in Virtual Environments", *Proc. of ACM SIGGRAPH*, 1998, pp. 353-360.

[Chen93] S. Eric Chen and Lance Williams, "View Interpolation for Image Synthesis", In *Computer Graphics (SIGGRAPH '93 Proceedings)*, vol. 27, J. T. Kajiya, Ed., August 1993, pp. 279--288.

[Chen95 S. Chen, "Quicktime VR: An image based approach to virtual environment navigation", *Proc. of ACM SIGGRAPH*, 1995.

[Chim98] J. Chim, M. Green, R. Lau, H. Leong and A. Si, "On Caching and Prefetching of Virtual Objects in Distributed Virtual Environments", *Proc. of ACM Multimedia*, 1998, pp. 171-180.

[Cohen97] J. Cohen, D. Manocha, and M. Olano. "Simplifying Polygonal Models Using Successive Mappings". In *Proc. of IEEE Visualization*, Tampa, AZ, 1997.

[Erikson99] C. Erikson and D. Manocha, "GAPS: General and Arbitrary Polygon Simplification", *Proc. of ACM Symposium on Interactive 3D Graphics*, 1999.

[Funkhouser93] T. A. Funkhouser. "Database and Display Algorithms for Interactive Visualization of Architecture Model". Ph.D. thesis. CS Division, UC Berkeley, 1993.

[Garland97] M. Garland and P. Heckbert, "Surface simplification based on Quadric Error Metric", *Proc. of ACM SIGGRAPH*, 1997.

[Gortler96] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The Lumigraph" In *SIGGRAPH 96 Conference Proceedings*, August 1996, pp. 43--54.

[Greene93]N. Greene, M. Kass, G. Miller, "Hierarchical Z-buffer visibility", *Proc. of ACM SIGGRAPH* 1993.

[Hoppe96] H. Hoppe. Progressive Meshes. In *SIGGRAPH 96 Conference Proceedings*: ACM SIGGRAPH, 1996, pp. 99--108.

[Katkere97] A. Katkere, S. Moessi, D. Kuramura, P. Kelly and R. Jain, "Towards video-based immersive environments", Multimedia Systems, May 1997, pp. 69-87.

[Kender97] John Kender and B. Yeo, "Video Scene Segmentation via Continuous Video Coherence", IBM Research Report RC 21061, December 1997.

[Lastra99] A. Lastra, Private communication, University of North Carolina at Chapel Hill, 1999.

[Levoy96] M. Levoy and P. Hanrahan. "Light Field Rendering", in *SIGGRAPH 96 Conference Proceedings*, August 1996, pp. 31--42.

[Maciel95] W. C. Maciel and Peter Shirley. "Visual Navigation of Large Environments Using Textured Clusters." In *1995 Symposium on Interactive 3D Graphics*, April 1995, pp. 95-102.

[Mannoni97] B. Mannoni, "A Virtual Museum", Communications of the ACM, vol. 40, no. 9, pp. 61-62, 1997.

[McMillan95] L. McMillan and Gary Bishop, "Plenoptic Modeling: An Image-Based Rendering System", in *SIGGRAPH 95 Conference Proceedings*, August 1995, pp. 39-46.

[Millerson90] G. Millerson. *The Technique of Television Production*. Focal Press, Oxford, England, 1990.

[MPEG2] MPEG-2, ISO, ISO/IEC JTCI CD 13818, *Generic Coding of moving pictures and associated audio*, 1994.

[MPEG4] MPEG4 Home Page. In http://drogo.cselt.stet.it/mpeg.

[MSSG] MPEG Software Simulation Group home page, http://www.mpeg.org/MSSG.

[Patel98] Ketan Patel and Lawrence Rowe, "Exploiting Temporal Parallelism for Software-only Video Effects Processing", *Proc. of ACM Multimedia*, pp. 161-170, 1998.

[Rohlf94] J. Rohlf and J. Helman, "Iris Performer: A high performance multiprocessor toolkit for realtime 3D Graphics". In *Proc. of ACM Siggraph*, 1994, pp. 381--394.

[Shade96] J. Shade, D. Lischinski, D. Salesin, T. DeRose, and J. Snyder, "Hierarchical Image Caching for Accelerated Walkthroughs of Complex Environments", In *SIGGRAPH 96 Conference Proceedings*, August 1996, pp. 75--82.

[Shen95] K. Shen and E. J. Delp, "A fast algorithm for video parsing using MPEG compressed sequences", *International Conference on Image Processing*, vol. II, pp. 252-255, Oct. 1995.

[Teller91] S. Teller and C. H. Sequin. Visibility preprocessing for interactive walkthroughs. In *Proc. of ACM Siggraph*, 1991, pp. 61--69.

[Tu00] X. Tu and B. L. Yeo, "Interactive Video for E-Merchandising", Intel Internal Report, Jan. 2000.

[Wei97] Q. Wei, H. Zhang and Y. Zhong, "A robust approach to video segmentation using compressed data", *Proceedings SPIE Storage and Retrieval for Still Images and Video Databases V*, vol. SPIE 3022, pp. 448-456, Feb. 1997.

[Yeung97] M. Yeung and B. Yeo, "Video visualization for compact presentation and fast browsing of pictorial content", *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 7, pp. 771-785, Oct. 1997.

[Yeo00] B.L. Yeo, M. M. Yeung, and V. Kuriakin. "MPEG Processing Library (MPL)", Intel Internal Report, Jan. 2000.

[Zhang93] H. Zhang, A. Kankanhalli and S. Smoliar, "Automatic partitioning of full-motion video", *Multimedia Systems*, vol. 1, pp. 10-28, July 1993.

[Zhang97] H. Zhang, D. Manocha, T. Hudson, and K. Hoff, "Visibility culling using hierarchical occlusion maps", In *Proc. of ACM Siggraph*, 1997.

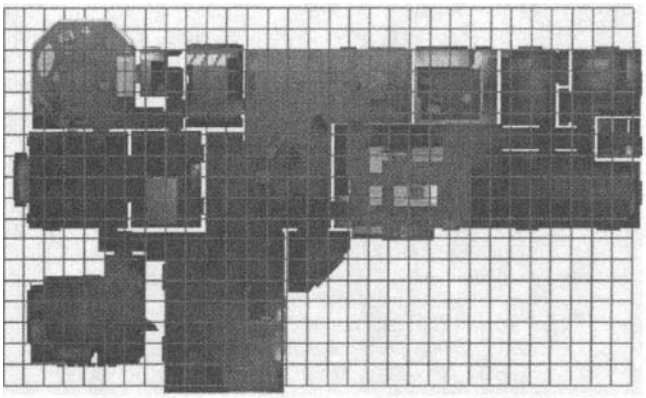

Figure 9: House model with cell grid overlaid in red. Each cell is 1 meter on a side and extends from the floor through the ceiling of the house.

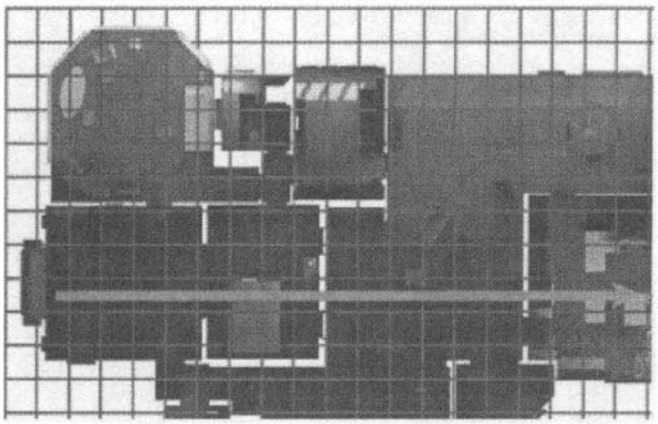

Figure 10: One row of cells through the model. These rows are used to map the cells into 1D streams for video encoding.

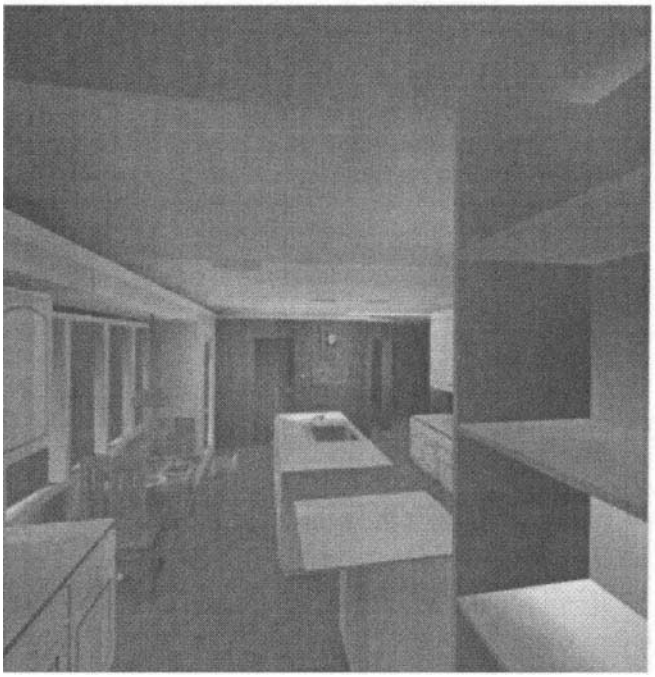

Figure 11: Video impostor showing correct perspective effects. The viewpoint is near the cell center in this image. Our system maintains a frame rate between 12 and 30 frames per second for such views.

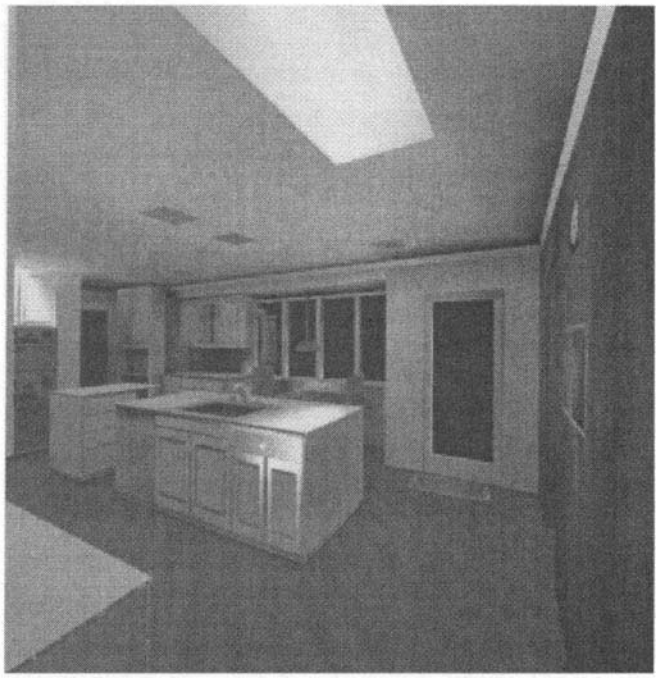

Figure 12: A view of the house with impostors disabled, for comparison with the artifacts below. This view is rendered with a frame rate of roughly 2 frames/sec.

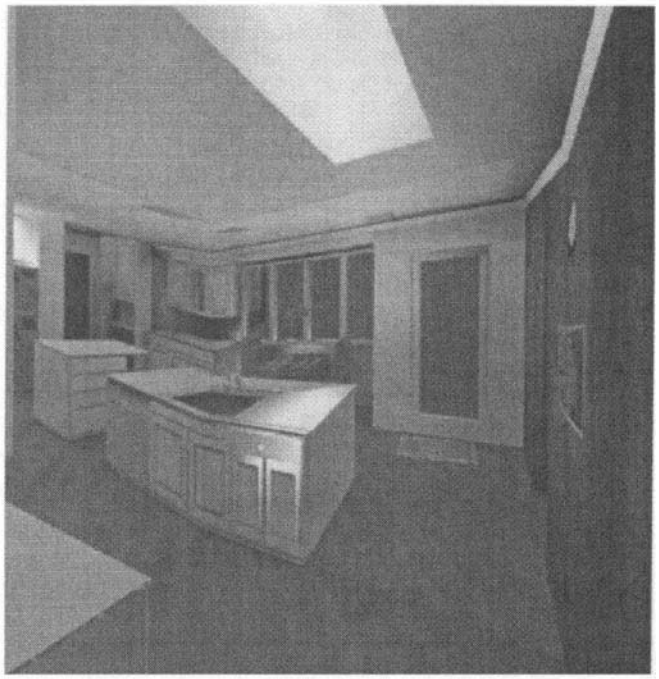

Figure 13: Example of perspective distortion caused by lack of depth parallax in impostors. The viewpoint is near the cell boundary in this image. Also note the color discontinuity between impostor and geometry.