Hierarchical Image-based and Polygon-based Rendering for Large-Scale Visualizations

Chu-Fei Chang[1], Zhiyun Li[2], Amitabh Varshney[2], and Qiaode Jeffrey Ge[3]

[1] Department of Applied Mathematics, State University of New York, Stony Brook, NY 11794, USA, chchang@cs.sunysb.edu
[2] Department of Computer Science and UMIACS, University of Maryland, College Park, MD 20742, USA, {zli, varshney}@cs.umd.edu
[3] Department of Mechanical Engineering, State University of New York, Stony Brook, NY 11794, USA, ge@design.eng.sunysb.edu

Summary. Image-based rendering takes advantage of the bounded display resolution to limit the rendering complexity for very large datasets. However, image-based rendering also suffers from several drawbacks that polygon-based rendering does not. These include the inability to change the illumination and material properties of objects, screen-based querying of object-specific properties in databases, and unrestricted viewer movement without visual artifacts such as visibility gaps. View-dependent rendering has emerged as another solution for hierarchical and interactive rendering of large polygon-based visualization datasets. In this paper we study the relative advantages and disadvantages of these approaches to learn how best to combine these competing techniques towards a hierarchical, robust, and hybrid rendering system for large data visualization.

1 Introduction

As the complexity of 3D graphics datasets has increased, different solutions have been proposed to bridge the growing gap between graphics hardware and the complexity of datasets. Most algorithms which effectively reduce the geometric complexity and overcome hardware limitations fall into the following categories: visibility determination [32,36,7,5,33,25,38], level-of-detail hierarchies [17], and image-based rendering (IBR) [10].

IBR has emerged as a viable alternative to conventional 3D geometric rendering, and has been widely used to navigate in virtual environments. It has two major advantages over the problem of increase in complexity of 3D datasets: (1) The cost of interactively displaying an image is independent of geometric complexity, (2) The display algorithms require minimal computation and deliver real-time performance on workstations and personal computers. Nevertheless, use of IBR raises the following issues:

- Economic and effective sampling of the scene to save memory without visually perceptible artifacts in virtual environments,
- Computing intermediate frames without visual artifacts such as visibility gaps,

- Allowing changes in illumination, and
- Achieving high compression of the IBR samples.

To address some of the above issues we have developed a multi-layer image-based rendering system and a hybrid image- and polygon-based rendering system. We first present a hierarchical, progressive, image-based rendering system. In this system progressive refinement is achieved by displaying a scene at varying resolutions, depending on how much detail of the scene a user can comprehend. Images are stored in a hierarchical manner in a compressed format built on top of the JPEG standard. At run-time, the appropriate level of detail of the image is constructed on-the-fly using real-time decompression, texture mapping, and accumulation buffer. Our hierarchical image compression scheme allows storage of multiple levels in the image hierarchy with minimal storage overhead (typically less than 10%) compared to storing a single set of highest-detail JPEG-encoded images. In addition, our method provides a significant speedup in rendering for interactive sessions (as much as a factor of 6) over a basic image-based rendering system.

We also present a hybrid rendering system that takes advantage of the respective powers of image- and polygon-based rendering for interactive visualization of large-scale datasets. In our approach we sample the scene using image-based rendering ideas. However, instead of storing color values, we store the visible triangles. During pre-processing we analyze per-frame visible triangles and build a compressed data-structure to rapidly access the appropriate visible triangles at run-time. We compare this system with a pure image-based, progressive image-based system (outlined above), and pure polygon-based systems. Our hybrid system provides a rendering performance between a pure polygon-based and a multi-level image-based rendering system discussed above. However, it allows several features unique to polygon-based systems, such as direct querying to the model and changes in lighting and material properties.

2 Previous Work

Visibility determination is a time- and space-consuming task. Good visibility information often takes significant time to compute and space to store. Current applications involving this problem often pre-compute the visibility information and store it for later use to improve the rendering speed. Teller and Sequin [32] divide a building into rooms and compute room-to-room visibility. Yagel and Ray [36] have proposed an algorithm to compute cell-to-cell visibility by applying a subdivision scheme. They also propose a clustering scheme to cluster cells with similar visibility. Coorg and Teller [6,7] use large occluders in the scene to perform occlusion culling for a viewpoint. Cohen-Or et al. [5] use large convex occluders to compute cell-to-object visibility. Wang et al. [33] pre-compute visible sets and simplify the regions where the visible sets are very large. Coorg and Teller [6] compute visibility information by using frame-to-frame incremental coherence. Panne and Steward [25] have presented two algorithms which effectively compress pre-computed visible sets over three different types of models.

Image-based Rendering (IBR) has recently emerged as an alternative to polygon-based rendering. The study of IBR has focused on image morphing and image interpolation for walkthroughs in virtual environments. Several ideas have been proposed to solve these problems including use of textures, environment maps, range images, depth information, movie maps, and so on. Several computer vision techniques have been being used in IBR for solving problems such as disparity maps, optical flows, and epipolar geometry. The techniques in computer graphics and computer vision are merging gradually in the newer applications to IBR.

An image morphing method usually involves two steps. The first step constructs the correspondence (mapping) between images. The second step uses the mapping to interpolate the intermediate images. Chen and Williams [4] have proposed an image morphing method. Their method uses the camera transformation and image range data to determine the pixel-to-pixel correspondence between images. They use a Z-buffer algorithm on pixels to solve the pixel overlap problem and interpolate adjacent pixels to fill holes. Chen [3] has described an image-based rendering system, which is now known as QuickTime VR. He uses 360° cylindrical panoramic images. In this system, a fixed-position camera can roll freely by simply rotating images. The pitch and yaw can be achieved by reprojecting an environment map. To achieve high quality in continuous zooming, this system interpolates the adjacent levels in a hierarchical image representation.

Adelson and Bergen [1] have proposed the plenoptic function concept. They used a plenoptic function to describe the structure of information in the light impinging on an observer. The plenoptic function is parameterized by eye position (V_x, V_y, V_z), azimuth and elevation angles θ and ϕ from any viewable ray to the eye, and a band of wavelengths λ. A view from a given eye position in a given direction is thus formulated as $P(V_x, V_y, V_z, \theta, \phi, \lambda)$. McMillan and Bishop [22] have proposed *plenoptic modeling* for image-based rendering. They have formulated the relative relationships between a pair of cylindrical projections (cylindrical epipolar geometry). They have resolved the visibility problem efficiently by a simple partitioning and enumeration method on cylinders, in a back-to-front ordering. Mark et al. [18] have proposed a post-rendering 3D warping algorithm by first reconstructing the image-space 3D mesh from reference frames and then warping the mesh into the derived frame. Rademacher and Bishop [27] have proposed "multiple-center-of-projection images"(MCOP). An MCOP image consists of a two-dimensional array of pixels and a parameterized set of cameras. A Z-buffer algorithm is used here to solve the visibility problem. To achieve a better quality, blending methods in [26] and [9] can be applied. MCOP images provide connectivity information among adjacent samples and allow different parts of scene to be sampled at different resolutions.

Shade et al. [29] have proposed layered depth images (LDI). In a LDI, each pixel contains a set of depth values along one line of sight sorted in front-to-back order. They have also proposed two rendering algorithms which heavily depend on McMillan's ordering algorithm ([20,19,22]). After pixel ordering, traditional splatting methods are applied to render the warped image. Oliveira and Bishop [24] have proposed an image-based representation for complex three-dimensional objects, the image-based object (IBO). Each IBO is represented by six LDIs sharing a single center of projection (COP). They have proposed a list-priority algorithm, which is based on epipolar geometry and an occlusion compatible order [22] for rendering.

Darsa et al. [9] have constructed image-space triangulation ([8]) from cubical environment maps with depth information. The goal of the triangulation is to establish the 3D geometric information of an environment map as accurately as possible. The triangulation represents a view-dependent simplification of the polygonal scene. Each triangle is assigned a quality which is related to the angle that the normal vector of the triangle makes with the viewing ray. In real time, the warping process involves projecting the triangles of the visible triangulations and texture mapping. A Z-buffer is used for hidden surface removal. Several blending schemes are used to render the intermediate images and improve the quality. This warping highly depends on the hardware texture mapping and transformations.

Levoy and Hanrahan [16] and Gortler et al. [11] have reduced the 5D plenoptic function (without λ) to a 4D Light Field or Lumigraph. They make use of the fact that the radiance does not change along a line unless it is blocked in free space. Light fields or Lumigraphs may be represented as functions of oriented lines. The authors have used *light slabs* as the representations. Both methods use quadralinear basis function to improve the result of interpolations. Sloan et al. [31] have extended the work of Gortler et al. [11] by using hardware texture mapping. Heidrich et al. [13] have improved the image quality of Lumigraph by adding new images to the current Lumigraph. They warp the closest images from the Lumigraph to a new viewpoint that lies on the viewpoint plane and add the new images. The warping step relies on depth information and performs the depth-correction precisely. Schirmacher et al. [28] have presented an adaptive acquisition algorithm by applying the work of Heidrich et al. They predict the error or potential benefit in image quality when adding a new image, and decide which new image should be rendered and added to the Lumigraph. Instead of using two slabs, Camahort et al. [2] have proposed a two-sphere parameterization (2SP) and a sphere-plane parameterization (SPP)for a more uniform sampling.

Nimeroff et al. [23] have effectively re-rendered the scene under various illuminations by linearly interpolating a set of pre-rendered images. Wong et al. [35,34] have proposed an algorithm which allows image-based objects to be displayed under varying illumination. They compute a *bidirectional reflectance distribution function* (BRDF) [14] for each pixel over various illuminations, and store the tabular BRDF data in spherical harmonic domain for reducing the storage.

3 Multi-Level Image-Based Rendering

In this section, we present an image-based rendering system. This system composes a scene in a hierarchical manner to achieve the progressive refinement by using different resolution images. Progressive refinement is achieved by taking advantage of the fact that the human visual system's ability to perceive details is limited when the relative speed of the object to the viewer is high. We first discuss the pre-processing and then the run-time navigation.

3.1 Image Sampling and Collection

Data sampling and collection plays a very important role in an image-based rendering system. It directly affects the storage space and the real-time performance of the system including image quality, rendering speed and user's visual perception. Different sampling strategies can be applied depending on the purpose of the system.

Environment Setting In our system the model is placed at the center of a virtual sphere. The viewer (camera) is positioned on the sphere with the viewing direction toward the origin. The viewer can move around the sphere along longitude and latitude. The camera takes one snapshot every $\Delta\theta$ degree along longitude and $\Delta\phi$ degree along latitude. Due to the symmetry of the sphere, we will have $360/\Delta\theta \times 180/\Delta\phi$ camera positions. The sampling density of camera positions may be adjusted by changing the values of $\Delta\theta$ and $\Delta\phi$. In our implementation, we use $\Delta\theta = \Delta\phi = 5^\circ$ to achieve a reasonably smooth and continuous motion with 2592 images.

3.2 Multi-Level Image Construction Algorithm

The algorithm computes n different levels of resolution of images as the basis for building the system image database. Our algorithm has the following steps:

Step 1: Decide the number n of progressive refinement levels in the system and the resolution of the display window, say $W \times W$, where $W = 2^m$, and $m \leq n$.

Step 2: Dump a Level 0 image, say I_0, at the display window resolution $(W \times W)$.

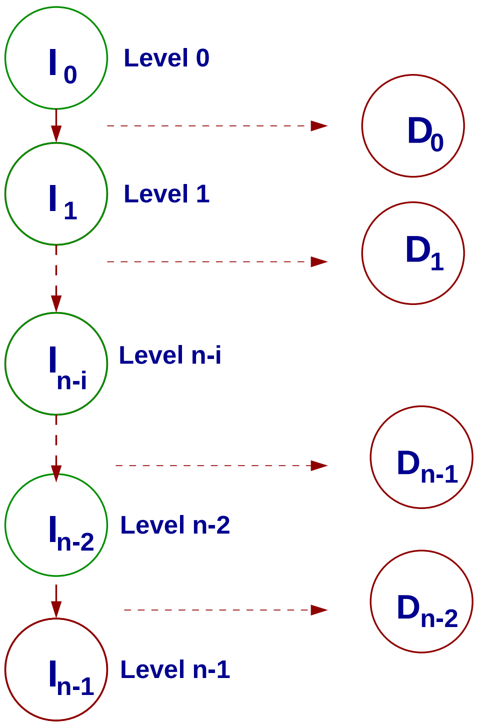


Fig. 1. Multi-Level Image Construction

Step 3: Construct Level $i+1$ image (resolution $= W/2^{i+1} \times W/2^{i+1}$), i.e., I_{i+1}. The RGB values of the level $i+1$ image are constructed from the RGB values of the level i image by the following equations:

$$R^{i+1}_{j,k} = min\{R^i_{2j,2k}, R^i_{2j+1,2k}, R^i_{2j,2k+1}, R^i_{2j+1,2k+1}\} \tag{1}$$

$$G^{i+1}_{j,k} = min\{G^i_{2j,2k}, G^i_{2j+1,2k}, G^i_{2j,2k+1}, G^i_{2j+1,2k+1}\} \tag{2}$$

$$B^{i+1}_{j,k} = min\{B^i_{2j,2k}, B^i_{2j+1,2k}, B^i_{2j,2k+1}, B^i_{2j+1,2k+1}\} \tag{3}$$

where $i = 0, 1, \ldots, n-2$. For example, we compute R^{i+1}_{00} as the minimum amongst $\{R^i_{00},\ R^i_{10},\ R^i_{01},\ R^i_{11}\}$. We repeat this step until image I_{n-1} is computed.

Step 4: Compute the $W/2^i \times W/2^i$ resolution image I_i from the $W/2^{i+1} \times W/2^{i+1}$ resolution image I_{i+1} as follows. Display I_{i+1} on a $W/2^i \times W/2^i$

resolution window using texture mapping and dump the displayed window image as T_i. Compute image difference D_i as:

$D_i = I_i - T_i, \quad i = 0, 1, \ldots, n-2$

Repeat this step until image difference D_{n-2} is computed, see Figure 1.

Step 5: Store I_{n-1}, D_{n-2}, D_{n-3}, …, D_0 in JPEG format as the database images.

This algorithm works well since texture mapping hardware provides speed and antialiasing capabilities through the OpenGL function `glDrawPixels()`. Also, image differences compress better than full images and provide an easy way to generate progressive refinement. For compression and decompression we use the public-domain JPEG software [12] in our implementation. It supports sequential and progressive compression modes, and is reliable, portable, and fast enough for our purposes. The reason we take the minimum value in equations 1–3 is so that we can store all RGB values of D_i as positive values and save a sign bit in storage.

3.3 Progressive Refinement Display Algorithm

Let us define $Tex(I_{n-1})$ as the image generated by mapping texture image I_{n-1} on a $W \times W$ resolution window, and define $Tex(D_i)$ as the image created by mapping texture image D_i on a $W \times W$ resolution window, where $i = 0, 1, \ldots, n-2$. At run time, image I_i is displayed by accumulating images $Tex(I_{n-1})$, $Tex(D_{n-2})$, $Tex(D_{n-3})$, …, $Tex(D_i)$, where $i = 0, 1, \ldots, n-1$.

If $i = n-1$, we only display image $Tex(I_{n-1})$, which has the lowest detail. We add image $Tex(D_i)$ to $Tex(I_{n-1})$, for $i = n-2, n-3, \ldots, 0$, to increase the image details. Image I_0, which is $Tex(I_{n-1}) + \sum_{i=0}^{n-2} Tex(D_i)$, has the highest detail (see Figure 2). Notice that all images are decompressed before texture mapping. The implementation is done using the OpenGL accumulation buffer and texture mapping.

In a real-time environment, progressive refinement can be achieved by displaying different levels of images, depending on how much detail of the scene the user needs to see. If the user moves with high speed, we can simply display lowest detail. As the user speed reduces, we can raise the level of detail of the displayed image in a progressive fashion.

3.4 Our Implementation and Results

In our implementation, we use three different image resolutions, 128×128, 256×256, and 512×512, for progressive refinement. We use sequential mode with quality setting 80 for JPEG compression, which gives us an unnoticeable difference from the highest quality setting of 100. We observed that the composite image quality in our system is not only affected by the lossy JPEG compression, but also by the error from image difference and the geometric

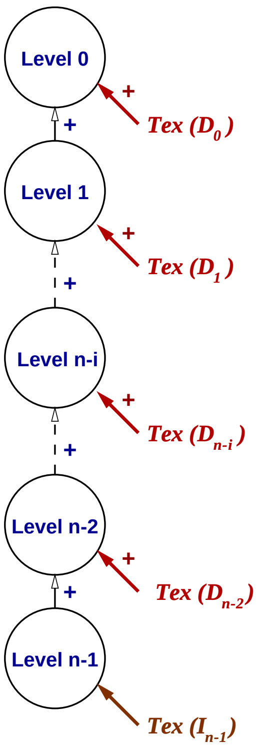


Fig. 2. Hierarchical Accumulation of JPEG Images

error from texture mapping. Table 1 shows the JPEG image reduction from full images I to image differences D. $\sum I_i$ is the sum of storage space required for all the I_i (level i) images. Similarly, $\sum D_i$ is the sum of storage space required for all the D_i (level i) images. The total storage space required is computed as $\sum(I_2 + D_1 + D_0)$. Note that the total memory consumption compares quite favorably to the original (non-progressive) storage space requirements ($\sum I_0$).

Table 2 shows the decompression time, rendering time, and frame rate on different image levels. All numbers in this table are average numbers over different models. I_i, $i = 0, 1, 2$ are full images of 512×512, 256×256, and 128×128 resolutions, respectively. D_i, $i = 0, 1$ are the image differences we discussed in Section 3.2. The *image error* is the root-mean-squared difference between the two images. The errors reported are with respect to the I_0 image.

4 Hybrid Rendering

Image-based rendering is a two-stage process. The first stage is off-line preprocessing that includes sampling of the necessary scene information and setting

Table 1. Storage for I_i, D_i and the total system

Model	Level 2	Level 1		Level 0		Total (MB)
	$\sum I_2$	$\sum I_1 \rightarrow$	$\sum D_1$	$\sum I_0 \rightarrow$	$\sum D_0$	$\sum(I_2 + D_1 + D_0)$
Bunny	10.70	21.39	10.70	53.11	31.34	52.74
Submarine	19.35	40.54	29.63	112.53	70.13	119.11
Enoyl Hydratase	21.29	37.88	21.31	107.34	46.26	88.86
Stanford Dragon	12.19	25.73	21.15	64.70	42.61	75.95
Stanford Buddha	10.70	20.15	14.22	47.22	30.86	55.78

Table 2. Multi-Level Image Rendering Comparison

Image Level	Decompression Time (msec)	Rendering Time (msec)	Speed (fps)	Image Error
$I_2 + D_1 + D_0$	98.6	23.8	7.99	0.435
$I_2 + D_0$	25.4	16.6	23.80	0.077
I_2	6.3	9.4	63.96	0.079
I_1	20.9	10.1	32.31	0.025
I_0	78.9	17.4	10.37	0.0

up data structures, possibly with hierarchy and compression, to reduce access times. The second stage deals with real-time rendering of pre-processed image data which may include image interpolation and warping. Like conventional image-based methods, our hybrid method also has two stages; and the key difference is that, instead of using three- or four-channel color values for each image, we compute the exact visibility of each triangle for each viewpoint, and only the visible (displayed) triangles are stored for each viewpoint. We outline our hybrid system below:

Preprocessing

1. **Initialization**
   1.1 Load polygonal dataset.
   1.2 Encode all primitive ids into RGB values.
   1.3 Set environment parameters such as viewing angle, distance, and resolution.
2. **Dumping process**
   2.1 Dump visible triangles for each camera position.
3. **Compression process**
   3.1 Single frame compression (run-length and Huffman compression).
   3.2 Frame-to-frame compression (intersection and union analysis).

Run-time Navigation

1. **Initialization**
   1.1 Load frame data file.
   1.2 Set frame specific world environment parameters.
   1.3 Construct Huffman trees and locate viewpoints for frames.
2. **Real time**
   2.1 New viewer position.
   2.2 Get the corresponding frame for current viewer node.
   2.3 Decompress the frame.
   2.4 Display frame.

These steps are explained in the following sections.

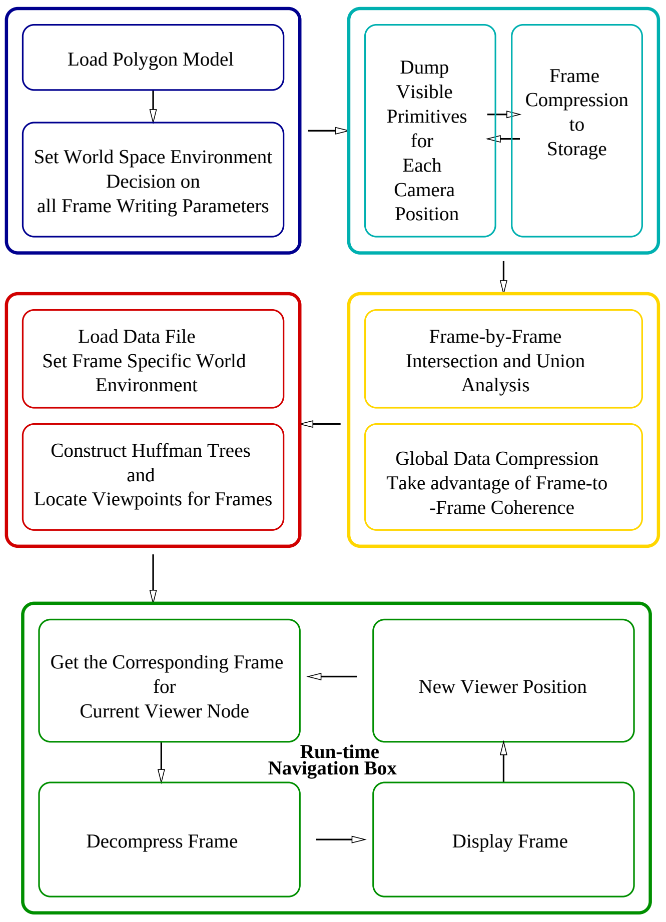

Load Polygon Model
Set World Space Environment Decision on all Frame Writing Parameters
Dump Visible Primitives for Each Camera Position
Frame Compression to Storage
Load Data File Set Frame Specific World Environment
Construct Huffman Trees and Locate Viewpoints for Frames
Frame-by-Frame Intersection and Union Analysis
Global Data Compression Take advantage of Frame-to -Frame Coherence
Get the Corresponding Frame for Current Viewer Node
New Viewer Position
Run-time Navigation Box
Decompress Frame
Display Frame

Fig. 3. Hybrid Rendering System Overview

4.1 Preprocessing

We adopt the same environment settings as we did in the JPEG image-based rendering system, see section 3.1.

Encoding Triangle IDs In order to compute the visibility for each triangle, we assign each triangle a unique id when we load the dataset. We then decompose the number, in binary format, into three consecutive bytes and assign them to R, G, and B in order. During the dumping process, we render the whole dataset with the given RGB value for each triangle as its color. Notice here that in order to render all colors correctly, the illumination and antialiasing function in OpenGL should be turned off. We then read the color buffer of this image to get the color for each pixel and compose the R, G, B back to the id. We currently use unsigned char for each single color value, which means, with a one-pass encoding process, we can encode as many as $(2^8)^3 = 16$ million triangles. For larger datasets, multiple-pass encoding processes may be needed. In our method we dump triangles for each camera position (θ, ϕ) by using the dumping process we discussed in Section 3.1 into an occupancy bit-vector, say TriMap(θ, ϕ).

Compression Process Two types of compression are relevant in image-based navigation of virtual environments: single-frame compression and frame-to-frame compression. We have only worked with single frame compression at this stage; the multiple frame compression, which needs more analysis and work, will be dealt with in the future. For representing the visible triangles in a single frame we use an occupancy bit vector (an unsigned char array) in which each bit represents the triangle id corresponding to its position in the vector. The bit is 1 if the triangle is visible in that frame, 0 otherwise.

As the size of 3D datasets increases and the resolution of image space remains fixed, the number of dumped triangles will saturate around the display resolution. In our results, the million triangle Buddha model has on an average only $5 \sim 6\%$ visible triangles for a 512×512 resolution window. It means that most bits in a bit vector would be 0, and consecutive-0-bit-segment cases would occur frequently. This result inspires us to use run-length encoding and Huffman compression.

4.2 Run-time Navigation

At run time the 3D dataset and precomputed information in compressed format is loaded first. The precomputed information not only includes the visible primitives for each frame but also the viewing parameters including viewing angle, distance, and so forth. The run-time viewing parameters should be exactly the same as those used in the dumping process. In the system, each camera position has a frame pointer pointing to the corresponding frame in compressed format. A Huffman tree, which is used for decompression, is also constructed for each frame.

At run time the viewer moves around in a virtual environment following discrete camera positions at $\Delta\theta$, $\Delta\phi$ increments which were used in the dumping process. For a given viewer position, we can locate the corresponding frame by following its frame pointer and decompress the frame by retracing the Huffman tree.

The rendering speed of the system highly depends on the number of visible triangles and the decompression time. In our implementation, the decompression function doesn't have to go through a whole data frame, it stops the decompression loop immediately whenever it detects that all dumped triangles have been found and sends them to the graphics engine. However, the decompression time still depends on the size of the frame (the size of object model) and the number of visible triangles in the frame.

5 Results

We have tested five different polygonal models on SGI Challenge and Onyx2. All models are tested at a window resolution of 512×512 pixels with 2592 images. We describe our results in this section.

Bunny, Dragon, and Buddha are scanned models from range images from the Stanford Computer Graphics Lab. The submarine model is a representation of a notional submarine from the Electric Boat Division of General Dynamics. The E. Hydratase Molecule (Enoyl-CoA Hydratase) is from the Protein Data Bank. All models have vertex coordinates (x, y, z) in floating-point format, and all triangles are represented by their three vertex indices (integers). The submarine dataset provides RGB color values (unsigned char) for each triangle. The E. Hydratase Molecule has a normal vector for each vertex.

Table 3 shows the average compression ratios for all models. The *average dumped tris %* is the average percentage of dumped triangles over all 2592 images.

In Table 4, *P* refers to conventional *Polygonal* rendering, *H* refers to the *Hybrid* rendering system discussed in Section 3, *I* refers to the *Multi-level Image-based* rendering system discussed in Section 2. The *Image Error* is the root-mean square error with respect to the images rendered using the conventional polygonal rendering method . *Dcmprs* is the time for decompression. As can be seen, the *Hybrid* method has consistently low image errors. Multi-level image-based rendering has the highest image error amongst these methods, since JPEG compression, image differences, and texture mapping all contribute to the final image error. For the *Hybrid* method, all visible triangles are stored in a bit vector and compressed by two steps: run-length encoding and Huffman compression. The decompression and rendering speeds are highly dependent on the displayed frame size and the number of dumped tri-

Table 3. Compression Ratios for the Hybrid Method

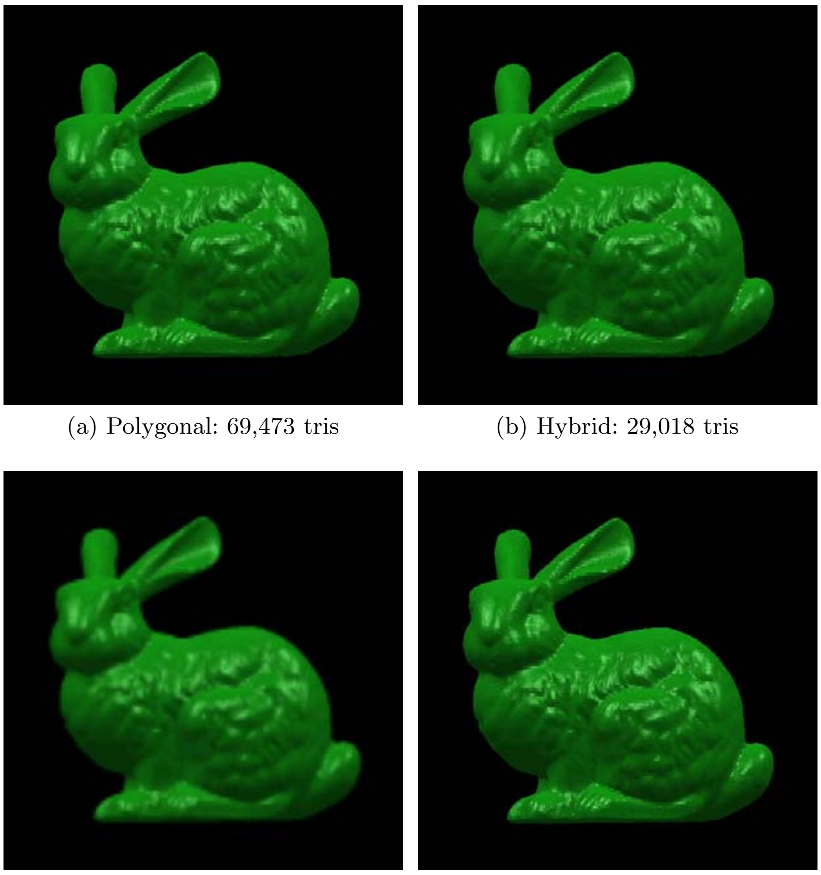
Model	Avg Dumped Tris %	Run Length Ratio	Huffman Ratio	Total Ratio
Bunny	35.68 %	1.47	1.29	1.82
Submarine	2.83 %	6.27	1.46	9.16
Enoyl Hydratase	11.21 %	3.54	1.24	4.38
Dragon	7.79 %	1.68	1.60	2.69
Buddha	4.04 %	2.32	1.75	4.07

Table 4. Comparison Results for Different Methods

Model	System	Storage	Time and Speed			Image
		Storage MB	Decompression (msec)	Rendering (msec)	Overall Speed (fps)	Image Error
Bunny 69K tris	P	1.54	0.0	61.3	16.39	0.0
	H	12.35	10.8	81.7	10.79	2.02E-4
	I	52.74	83.6	28.2	8.94	2.66E-2
Submarine 376K tris	P	12.85	0.0	3549.0	0.28	0.0
	H	13.32	11.1	118.1	7.73	7.29E-3
	I	119.11	107.8	27.1	7.40	1.42E-1
Enoyl Hy-dratase 717K tris	P	14.95	0.0	777.4	1.28	0.0
	H	37.93	32.1	177.8	4.76	4.15E-4
	I	88.86	108.6	28.9	7.26	2.69E-2
Dragon 871K tris	P	19.19	0.0	1306.9	0.76	0.0
	H	104.98	87.8	223.45	3.10	4.82E-4
	I	75.95	102.2	28.7	7.63	2.45E-3
Buddha 1087K tris	P	23.92	0.0	1638.3	0.61	0.0
	H	86.56	69.4	191.9	3.82	5.14E-4
	I	55.78	95.9	27.9	8.07	9.60E-2

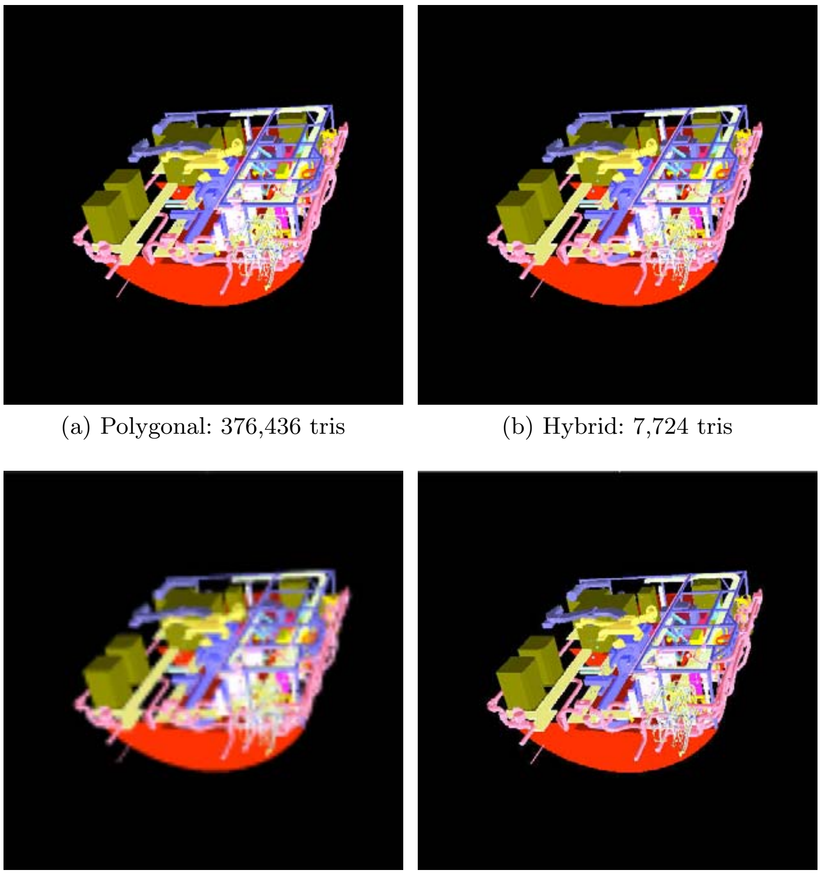
angles in that frame. The rendering speed on the Submarine is much slower than on the other models because we do the coloring for each rendered triangle. Without coloring, the *Polygonal* method has the average rendering speed about $7 - 9$ frames/sec, and the *Hybrid* method is over 10 frame/sec.

The traditional polygon rendering has the best quality amongst all methods and requires least storage space, but it has the lowest rendering speed. The hybrid method which only renders visible triangles has very good rendering speeds, but needs much more storage space than traditional polygon rendering. Multi-level JPEG provides progressive refinement and has the lowest rendering complexity, but needs a lot of storage space. Figure 4, 5 and 6 show the images displayed by these methods on various models.


(a) Polygonal: 69,473 tris (b) Hybrid: 29,018 tris

(c) JPEG Level 0 (d) JPEG Level 2

Fig. 4. Different Rendering Methods for the Bunny



(a) Polygonal: 376,436 tris

(b) Hybrid: 7,724 tris

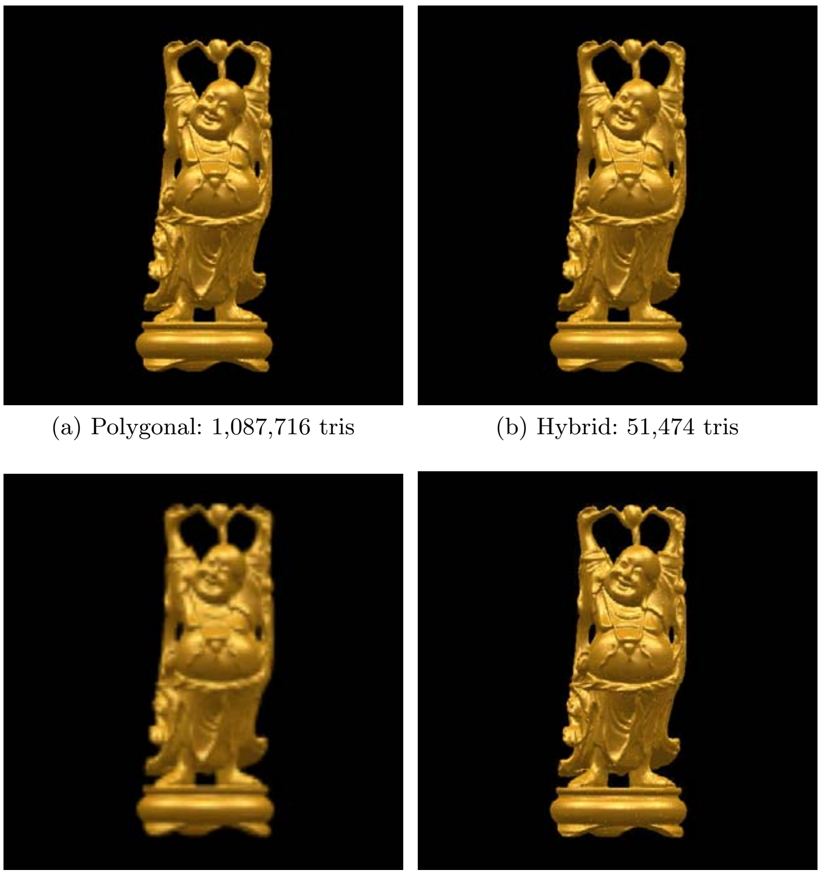
(c) JPEG Level 0

(d) JPEG Level 2

Fig. 5. Different Rendering Methods for the Auxiliary Machine Room of a notional submarine

6 Conclusions

In this paper we have presented a hybrid method as well as a progressive refinement image-difference-based rendering method for high-complexity rendering. Our hybrid method takes advantage of both conventional polygon-based rendering and image-based rendering. The hybrid rendering method can provide rendering quality comparable to the conventional polygonal rendering at a fraction of the computational cost and has storage requirements that are comparable to image-based rendering methods. The drawback is that it does not permit full navigation capability to the user as in the conventional polygonal method. However, it still retains several other useful features of the polygonal methods such as direct querying to the underlying database


(a) Polygonal: 1,087,716 tris (b) Hybrid: 51,474 tris

(c) JPEG Level 0 (d) JPEG Level 2

Fig. 6. Different Rendering Methods for the Buddha

and the ability to change illumination and material properties. In future we plan to further explore compression issues for the hybrid method by taking advantage of frame-to-frame coherence in image space and view-dependent geometric hierarchical structures.

7 Acknowledgements

We will like to acknowledge the detailed and useful comments from the reviewers that has improved the presentation of this work. This work has been supported in part by the NSF grants: DMI-9800690, ACR-9812572, and IIS-0081847. We will like to acknowledge Electric Boat Division of General Dynamics for providing us the submarine dataset and the Stanford Graphics lab for providing us the Bunny, the Dragon, and the Buddha models.

References

1. E. H. Adelson and J. R. Bergen. The plenoptic function and the elements of early vision. In M. Landy and J. A. Movshon, editors, *Computational Models of Visual Processing*, chapter 1. The MIT Press, Mass, 1991.
2. Emilio Camahort, Apostolos Lerios, and Donald Fussell. Uniformly sampled light fields. In G. Drettakis and N. Max, editors, *Rendering Techniques '98 (Proceedings of Eurographics Rendering Workshop '98)*, pages 117–130, New York, NY, 1998. Springer Wien.
3. S. E. Chen. Quicktime VR - an image-based approach to virtual environment navigation. In *Computer Graphics Proceedings (SIGGRAPH 95)*, Annual Conference Series, pages 29–38, 1995.
4. S. E. Chen and L. Williams. View interpolation for image synthesis. In *Computer Graphics Proceedings (SIGGRAPH 93)*, Annual Conference Series, pages 279–288, 1993.
5. Daniel Cohen-Or and Eyal Zadicario. Visibility streaming for network-based walkthroughs. In *Graphics Interface*, pages 1–7, June 1998.
6. S. Coorg and S. Teller. Temporally coherent conservative visibility. In *ACM Press*, Proceedings of 20th Annual Symposium on Computational Geometry, pages 78–87, May 1996.
7. S. Coorg and S. Teller. Real time occlusion culling for models with large occluders. In *Proceedings of 1997 Simposium in 3D Interactive Graphics*, pages 83–90, 1997.
8. L. Darsa and B. Costa. Multi-resolution representation and reconstruction of adaptively sampled images. In *Computer Graphics Proceedings (SIGGRAPH 96)*, Annual Conference Series, pages 321–328, 1996.
9. L. Darsa, B. Costa, and A. Varshney. Navigating static environments using image-space simplication and morphing. In *Proceedings of 1997 Symposium on Interactive 3D Graphics*, pages 25–34, April 1997.
10. P. Debevec, C. Bregler, M. Cohen, L. McMillan, F. Sillion, and R. Szeliski. *SIGGRAPH 2000 Course 35: Image-based Modeling, Rendering, and Lighting.* ACM SIGGRAPH, 2000.
11. S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen. The lumigraph. In *Computer Graphics Proceedings (SIGGRAPH 96)*, Annual Conference Series, pages 43–54, 1996.
12. Independent JPEG Group. ftp://ftp.uu.net/graphics/jpeg/.
13. Wolfgang Heidrich, Hartmut Schirmacher, Hendrik Kück, and Hans-Peter Seidel. A warping-based refinement of lumigraphs. In N. Thalmann and V. Skala, editors, *Proc. WSCG '99*, 1999.
14. J. T. Kajiya. Anisotropic reflection models. In *Computer Graphics Proceedings (SIGGRPAH 85)*, Annual Conference Series, pages 15–21, July 1985.
15. S. Kumar, D. Manocha, B. Garrett, and M. Lin. Hiarachical back-face culling. In *Eurographics Workshop on Rendering*, pages 231–240, 1996.
16. M. Levoy and P. Hanrahan. Light field rendering. In *Computer Graphics Proceedings (SIGGRAPH 96)*, Annual Conference Series, pages 31–42, 1996.

17. D. Luebke, J. Cohen, M. Reddy, A. Varshney, and B. Watson. *SIGGRAPH 2000 Course 41: Advanced Issues in Level of Detail*. ACM SIGGRAPH, 2000.
18. William R. Mark, Leonard McMillan, and Gary Bishop. Post-rendering 3D warping. In Michael Cohen and David Zeltzer, editors, *1997 Symposium on Interactive 3D Graphics*, pages 7–16. ACM SIGGRAPH, April 1997. ISBN 0-89791-884-3.
19. L. McMillan. Computing visibility without depth. Technical Report 95-047, University of North Carolina at Chapel Hill, 1995.
20. L. McMillan. A list-priority rendering algorithm for redisplaying projected surfaces. Technical Report 95-005, University of North Carolina at Chapel Hill, 1995.
21. L. McMillan and G. Bishop. Head-tracked stereoscopic display using image warping. In Stephen N. Spencer, editor, *Proceedings SPIE*, volume 2409, pages 21–30, San Jose, CA, Feb 1995.
22. L. McMillan and G. Bishop. Plenoptic modeling: An image-based rendering system. In *Computer Graphics Proceedings (SIGGRAPH 95)*, Annual Conference Series, pages 39–46, 1995.
23. J. S. Nimeroff, E. Simoncelli, and J. Dorsey. Efficient re-rendering naturally illuminated environments. In *Eurographics*, Fifth Eurographics Workshop on Rendering, pages 359–373, 1994.
24. Manuel M. Oliveira and Gary Bishop. Image-based objects. In Stephen N. Spencer, editor, *Proceedings of the Conference on the 1999 Symposium on interactive 3D Graphics*, pages 191–198, New York, April 26–28 1999. ACM Press.
25. M. Panne and A.J. Stewart. Effective compression techniques for precomputed visibility. In *Springer Computer Science*, Rendering Techniques '99, pages 305–316, 1999.
26. K. Pulli, M. Cohen, T. Duchamp, H. Hoppe, L. Shapiro, and W. Stuetzle. View-based rendering: Visualizing real objects from scanned range and color data. In *Proceedings of Eighth Eurographics Workshop on Rendering*, pages 23–34. Eurographics, June 1997.
27. P. Rademacher and G. Bishop. Multiple-center-of-projection images. In *Computer Graphics Proceedings (SIGGRAPH 98)*, Annual Conference Series, pages 199–206, 1998.
28. Hartmut Schirmacher, Wolfgang Heidrich, and Hans-Peter Seidel. Adaptive acquisition of lumigraphs from synthetic scenes. In Hans-Peter Seidel and Sabine Coquillart, editors, *Eurographics '99*, volume 18, pages C151–C159, Computer Graphics Forum, c/o Mercury Airfreight International Ltd Inc, 365 Blair Road, Avenel, MI 48106, USA, 1999. Eurographics Association, Eurographics Association and Blackwell Publishers Ltd 1999.
29. J. Shade, S. Gortler, L. He, and R. Szeliski. Layer depth images. In *Computer Graphics Proceedings (SIGGRAPH 98)*, Annual Conference Series, pages 231–242, 1998.
30. L. A. Shirman and S. S. Abi-Ezzi. The cone of normals technique for fast processing of curved patches. In *Eurographics*, pages 261–272, 1993.
31. P. Sloan, M. Cohen, and S. Gortler. Time critical lumigraph rendering. In *Symposium on Interactive 3D Graphics*, pages 17–23, 1997.
32. S. J. Teller and C. H. Sequin. Visibility preprocessing for interactive walkthroughs. In *Computer Graphics Proceedings (SIGGRAPH 91)*, Annual Conference Series, pages 61–69, 1991.

33. Y. Wang, H. Bao, and Q. Peng. Accelerated walkthroughts of virtual environments based on visibility preprocessing and simplification. In *Eurographics*, pages 17(3): 187–194, 1998.
34. T. Wong, P. Heng, S. Or, and W. Ng. Illuminating image-based objects. In *Proc. Pacific Graphics'97, Seoul, Korea*, 1997.
35. T. Wong, P. Heng, S. Or, and W. Ng. Image-based rendering with controllable illumination. In *Proc. 8th Eurographics Workshop on Rendering, St. Etienne, France*, 1997.
36. R. Yagel and W. Ray. Visibility computation for efficient walkthroughs of complex environments. In *Presence*, pages 5(1):45–60, 1995.
37. H. Zhang and K. E. Hoff III. Fast backface culling using normal masks. In *Symposium on Interactive 3D Graphics, SIGGRAPH 97*, 1997.
38. H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. In *Computer Graphics Proceedings (SIGGRAPH 97)*, Annual Conference Series, pages 77–88, 1997.