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

Chu-Fei Chang[*] Amitabh Varshney [†] Qiaode Jeffrey Ge[‡]

Abstract

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 the 3D graphics datasets has increased, different solutions have been proposed to bridge the growing gap between graphics hardware and the complexity of datasets. Most of algorithms which effectively reduce the geometric complexity and overcome hardware limitations fall into the following categories: visibility determination [7, 9, 2, 1, 8, 6, 10], level-of-detail hierarchies [5], and image-based rendering (IBR) [3]. IBR has emerged as a viable alternative to the conventional 3D geometric rendering, and has been widely used to navigate in virtual environments. It has two major advantages over the problem of increasing of 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 storage 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 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 the polygon-based systems, such as direct querying to the model and changes in lighting and material properties.

2 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.

2.1 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.bc 360/ 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 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, $\Delta\theta = \Delta\phi = 5^\circ$, to achieve a reasonably smooth and continuous motion with 2592 images.

[*]Department of Applied Mathematics, State University of New York, Stony Brook, NY 11794, chchang@cs.sunysb.edu

[†]Department of Computer Science, University of Maryland, College Park, MD 20742, varshney@cs.umd.edu

[‡]Department of Mechanical Engineering, State University of New York, Stony Brook, NY 11794, ge@design.eng.sunysb.edu

2.2 Multi-Level Image Construction Algorithm

The algorithm computes n different levels of resolutions 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 ($W \times W$).

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

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

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

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

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

Step 4: Compute $W/2^i \times W/2^i$ resolution image T_i from $W/2^{i+1} \times W/2^{i+1}$ 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 D_{n-2} image difference 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.

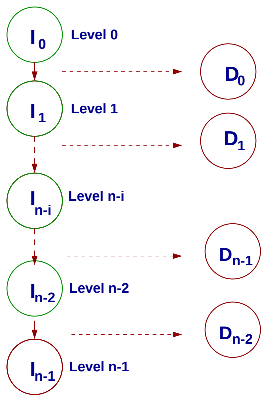


Figure 1: Multi-Level Image Construction

This algorithm works well since texture mapping hardware provides speed and antialiasing capabilities over 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 [4] 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.

2.3 Progressive Refinement Display Algorithm

Let us define $Tex(I_{n-1})$ as the image by texture mapping image I_{n-1} on $W \times W$ resolution window, and define $Tex(D_i)$ as the image by texture mapping image D_i on $W \times W$ resolution window, where $i = 0, 1, \ldots, n-2$. At run time, level i image 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 $Tex(D_i)$ image onto $Tex(I_{n-1})$, where $i = n-2, n-3, \ldots, 0$, to increase the image details. Level 0 image, which is $Tex(I_{n-1}) + \sum_{i=0}^{n-2} Tex(D_i)$, has the highest detail. Notice that all images are decompressed before texture mapping. The implementation is done by OpenGL accumulation buffer and texture mapping.

In a real-time environment, the 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.

2.4 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 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 for all the I_i (level i) images. Similarly, $\sum D_i$ is the sum of storage for all the D_i (level i) images. The total storage is computed as $\sum(I_2 + D_1 + D_0)$. Note that the total storage compares quite favorably to original (non-progressive) storage requirements ($\sum I_0$).

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

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

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

Table 2: Multi-Level Image Rendering Comparison

Table 2 shows the decompression time, rendering time, and frame rate on different image levels. All numbers in this table are the 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 2.2. The *image error* is the root-mean-squared difference between the two images. The errors reported are with respect to the I_0 image.

3 Hybrid Rendering

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

3.1 Preprocessing

We adopt the same environment settings as we did in the JPEG image-based rendering system, see section 2.1.

3.1.1 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 2.1 into a occupancy bit-vector, say TriMap(θ, ϕ).

3.1.2 Compression Process

Two types of compression are relevant in an 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 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 is 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 occur frequently. This result inspires us to use run-length encoding and Huffman compression.

3.2 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 includes 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 positon, we can locate the corresponding frame by following its frame pointer and decompress the frame by retracing the Huffman tree.

The rendering speed of 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 breaks 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.

4 Results

We have tested five different polygonal models on SGI Challenge and Onyx2. All models are tested on 512×512 resolution window 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. 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 the vertex coordinates (x, y, z) in floating-point format, and all triangles are represented by their three vertex indices (integer). Submarine has RGB color for values (unsigned char) for each triangle. The E. Hydratase Molecule has a normal vector for each vertex. All models are stored in OFF binary format.

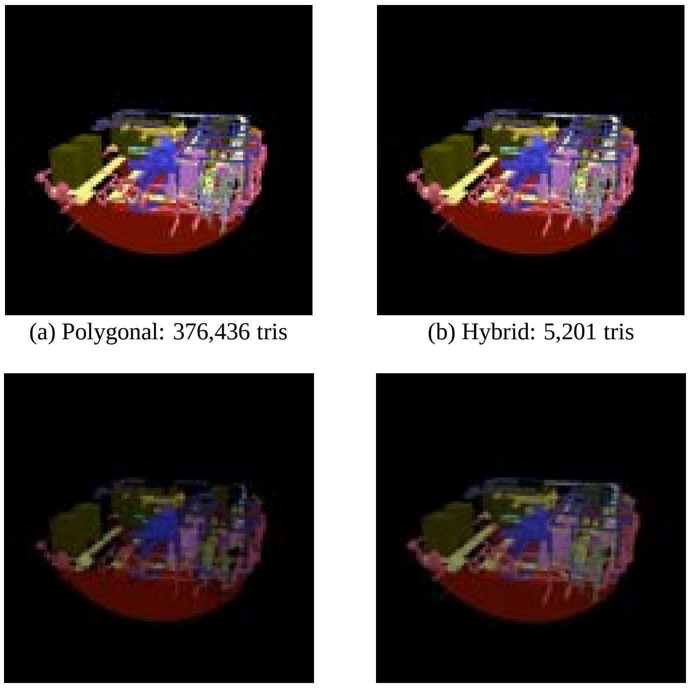
Table 3 has the average compression ratios on all models. The *average dumped tris %* is the average percentage of dumped triangles over 2592 images.

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
E. 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 3: Compression Ratios for the Hybrid Method

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 from the conventional polygonal rendering. *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 triangles in that frame. The rendering speed on the Submarine is much slower than 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 least storage, 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 than traditional polygon rendering. Multi-level JPEG provides progressive refinement and has the lowest rendering complexity, but needs a lot of storage. Figure 2 shows the images displayed by these methods on various models.



(a) Polygonal: 376,436 tris

(b) Hybrid: 5,201 tris

(c) JPEG Level 0

(d) JPEG Level 2

Figure 2: Different Rendering Methods on the Submarine

Model	System	Storage	Time and Speed			Image
		MB	Dcmprs (msec)	Render (msec)	Speed (fps)	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	136.20	107.8	27.1	7.40	1.42E-1
EnoylCOA Hydratase 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

Table 4: Comparison Results for Different Methods

5 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 that is comparable to the 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 and 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.

6 Acknowledgements

This work has been supported in part by the NSF grants: DMI-9800690, ACR-9812572, and IIS-0081847.

References

[1] Daniel Cohen-Or and Eyal Zadicario. Visibility streaming for network-based walkthroughs. In *Graphics Interface*, pages 1–7, June 1998.

[2] 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.

[3] 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.

[4] Independent JPEG Group. ftp://ftp.uu.net/graphics/jpeg/.

[5] D. Luebke, J. Cohen, M. Reddy, A. Varshney, and B. Watson. *SIGGRAPH 2000 Course 41: Advanced Issues in Level of Detail*. ACM SIGGRAPH, 2000.

[6] M. Panne and A.J. Stewart. Effective compression techniques for precomputed visibility. In *Springer Computer Science*, Rendering Techniques '99, pages 305–316, 1999.

[7] S. J. Teller and C. H. Sequin. Visibility preprocessing for interactive walkthroughs. In *Computer Graphics Proceedings(SIGGRAPH 91)*, pages 61–69, 1991.

[8] 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.

[9] R. Yagel and W. Ray. Visibility computation for efficient walkthroughs of complex environments. In *Presence*, pages 5(1):45–60, 1995.

[10] H. Zhang, D. Manocha, T. Hudson, and K. Hoff. Visibility culling using hierarchical occlusion maps. In *Proceedings of SIGGRAPH '97 (Los Angeles, CA)*, Computer Graphics Proccedings, Annual Conference Series, pages 77–88. ACM SIGGRAPH, ACM Press, August 1997.