Polygon-Assisted JPEG and MPEG Compression of Synthetic Images

Marc Levoy
Computer Science Department
Stanford University

Abstract

Recent advances in realtime image compression and decompression hardware make it possible for a high-performance graphics engine to operate as a rendering server in a networked environment. If the client is a low-end workstation or set-top box, then the rendering task can be split across the two devices. In this paper, we explore one strategy for doing this. For each frame, the server generates a high-quality rendering and a low-quality rendering, subtracts the two, and sends the difference in compressed form. The client generates a matching low quality rendering, adds the decompressed difference image, and displays the composite. Within this paradigm, there is wide latitude to choose what constitutes a high-quality versus low-quality rendering. We have experimented with textured versus untextured surfaces, fine versus coarse tessellation of curved surfaces, Phong versus Gouraud interpolated shading, and antialiased versus nonantialiased edges. In all cases, our polygon-assisted compression looks subjectively better for a fixed network bandwidth than compressing and sending the high-quality rendering. We describe a software simulation that uses JPEG and MPEG-1 compression, and we show results for a variety of scenes.

CR Categories: I.4.2 [Computer Graphics]: Compression — *Approximate methods* I.3.2 [Computer Graphics]: Graphics Systems — *Distributed/network graphics*

Additional keywords: client-server graphics, JPEG, MPEG, polygon-assisted compression

1. Introduction

In this era of open systems, it is common for multiple graphics engines that are software compatible but have greatly differing performance to reside on the same network. A research group might have a dozen low-end workstations on desktops and one high-performance workstation in a centralized laboratory. Future multi-user video games may have hundreds of set-top boxes connected by cable or phone lines to a centralized game server. Recent advances in realtime image compression and decompression hardware make it possible for the high-performance machine to operate as a rendering server for the low-end machines. This can be accomplished straightforwardly by rendering on the server, then compressing and transmitting an image stream to the client. The client decompresses and displays the image stream in a window distinct from its own frame buffer.

Unfortunately, the standards for compressing images and video - mainly JPEG [Wallace91] and MPEG [Le Gall91] - were developed for use on natural scenes, and they are not well suited for compressing synthetic images. In particular, they perform poorly at the edges of objects and in smoothly shaded areas.

In this paper, we consider an alternative solution that partitions the rendering task between client and server. We use the server to render those features that cannot be rendered in real time on the client - typically textures and complex shading. These are compressed using JPEG or MPEG and sent to the client. We use the client to render those features that compress poorly using JPEG or MPEG - typically edges and smooth shading. The two renderings are combined in the client for display on its screen. The resulting image is subjectively better for the same bandwidth than can be obtained using JPEG or MPEG alone. Alternatively, we can produce an image of comparable quality using less bandwidth.

The remainder of the paper is organized as follows. In section 2, we give an overview of our solution, and we suggest typical hardware realizations. In section 3, we describe a software simulator we have built to test our idea, and we discuss several implementation issues. In section 4, we explore ways to partition the rendering task between client and server. Some partitionings work well, and some do not, as we shall see. In sections 5 and 6, we discuss related work, limitations, and extensions.

2. Client-server relationship

Figure 1 shows the flow of data in our proposed client-server system. The hardware consists of a high-performance workstation (henceforth called the server), a low-performance workstation (henceforth called the client), and a network. To produce each frame of synthetic imagery, these two machines perform the following three steps:

(1) On the server, compute a high-quality and low-quality rendering of the scene using one of the partitioning strategies described in section 4.

(2) Subtract the two renderings, apply lossy compression to the difference image, and send it to the client.

(3) On the client, decompress the difference image, compute a low-quality rendering that matches the low-quality rendering computed on the server, add the two images, and display the resulting composite image.

Depending on the partitioning strategy, there may be two geometric models describing the scene or one model with two rendering options. The low-quality model may reside on both machines, or it may be transmitted from server to client (or client to server) for each frame. If the model resides on both machines, this can be implemented using display lists or two cooperating copies of the application program. The latter solution is commonly used in networked visual simulation applications.

To provide interactive performance, the server in such a system would normally be a graphics workstation with hardware-accelerated rendering. The client might be a lower-end hardware-accelerated workstation, or it might be a PC performing rendering in software, or it might be a set-top box utilizing a

Address: Center for Integrated Systems
Stanford University
Stanford, CA 94305-4070
Email: levoy@cs.stanford.edu
Web: http://www-graphics.stanford.edu

Figure 1: Flow of data in proposed client-server rendering system. High and low-quality renderings may differ in shading, geometric detail, or other aspects. Transmission of the low-quality geometric model is optional, so it is drawn dashed in the figure.

combination of software and hardware. Differencing and compression on the server, and decompression and addition on the client, would most likely be performed in hardware, although real-time software implementations are also beginning to appear.

One important caveat regarding the selection of client and server is that there are often slight differences in pixel values between equivalent-quality renderings computed by high-performance and low-performance machines, even if manufactured by the same vendor. If both renderings are antialiased, these differences are likely to be small. Nevertheless, they may adversely affect the reconstruction in step three.

3. Software simulation

Since no commercially available workstation yet offers both high-performance rendering and real-time compression/decompression, we have built a software simulation. Rendering is performed using the REYES rendering system [Cook87] or the SoftImage Creative Environment, and compression is performed using the Independent JPEG Group's codec [Lane] or the Berkeley MPEG-1 codec [Rowe].

Images in our simulations are represented as 24-bit RGB pixels with two exceptions. First, the codecs performs compression in YCrCb using 4:1:1 subsampling - 4 pixels of Y to 1 each of Cr and Cb. Second, pixels in the difference image D are computed from pixels in the high and low quality images H and L using the formula $D = 127 + (H - L)/2$. This formula maps zero-difference pixels to gray and maps positive and negative differences to lighter and darker colors, respectively (see figure 2c). Following this division by 2, features in the difference image are represented by pixel excursions (from the mean) half as large as corresponding features in the high-quality rendering. To prevent these features from being quantized twice as severely during compression, we adjust the quantization tables to compensate. The effect is to match the feature degradation (and code size) that would have resulted had we not divided the difference pixels by 2.

Our experiments using the JPEG codec are presented in figures 2 through 5. Selected statistics for figures 2 and 3 are summarized in table I. The table gives statistics for image-based compression, for polygon-assisted compression using resident geometric models, and for polygon-assisted compression using a transmitted low-quality model.

Whenever a model is transmitted, it should be compressed. We have not implemented compression of geometric models, and little research has been done on the subject, but we can estimate the performance of a simple lossless scheme as follows. We assume a polygon mesh, which contains on average one vertex per polygon. In our application, vertices can be transformed from object space to screen space prior to transmission, after which a two-byte fixed point representation suffices for each coordinate. Thus, we need 10 bytes per vertex (6 bytes for XYZ + 4 bytes for RGBA). One can then difference the coordinates and colors of successive vertices and encode the differences using Huffman or arithmetic coding. If successive vertices are spatially adjacent as they would be in a mesh, this technique should perform well. Danskin has used a similar method to compress sequences of mouse events in X, obtaining 3:1 compression [Danskin94]. We thus estimate that each polygon in a compressed model requires 3.3 bytes.

To help us understand the performance of polygon-assisted compression, we have computed the entropies of the high-quality renderings and difference images. As expected, the latter consistently have less entropy than the former. We have also computed the root mean square errors in the polygon-assisted compressions and image-based compressions, relative in each case to the high-quality renderings. Unfortunately, root mean square error is a poor measure of subjective image quality. In our opinion, the only meaningful way to evaluate the performance of our method is to look at the images.

We have also computed several animations using motion JPEG and MPEG-1. Our conclusions match those for still images: polygon-assisted compression yields subjectively better imagery than image-based compression for the same bandwidth. MPEG-1 looks better than motion JPEG even at higher compression rates due to its use of motion compensation, but neither looks as good as polygon-assisted compression. Unfortunately, still images or analog videotape recordings do not capture the full quality difference, which is only evident by looking at a workstation screen.

4. Partitioning strategies

Image-based compression (e.g. JPEG) of synthetic images fails most severely at the silhouette edges of objects and in smoothly shaded areas. What these two features have in common is spatial coherence. In other words, they both exhibit relatively large-scale structure. In choosing how to partition a synthetic image into a polygon rendering and a compressed difference image, we should therefore strive to incorporate into the rendering as much of the coherent structure of the synthetic image as possible. In the following paragraphs, we describe three partitioning strategies - two that work well and one that does not.

4.1. Textured versus untextured surfaces

High-end graphics workstations (such as the Silicon Graphics RealityEngine) are typically optimized for the display of textured surfaces, while low-end workstations (such as the Silicon Graphics Indy) are typically optimized for the display of untextured surfaces. Given these capabilities, the most obvious way to partition rendering between a high-end server and a low-end client is to omit surface texture on the client.

To demonstrate this, we consider a room composed of flat surfaces that exhibit smooth shading and texture (see figure 2). The model contains 1131 polygons with a fixed color at each vertex. This color was calculated using a hierarchical radiosity algorithm that approximates the diffuse interreflection among textured surfaces [Gershbein94]. The high-quality rendering (figure 2a) employs antialiasing, Gouraud-interpolated shading, and texturing. The low-quality rendering (figure 2b) employs antialiasing and Gouraud-interpolated shading but no texturing. The difference between the two renderings is shown in figure 2c.†

Figures 2d through 2g show image-based compression of the high-quality rendering using varying JPEG quality factors. Figures 2h through 2k show polygon-assisted compression using quality factors selected to match as closely as possible the code sizes in figures 2d through 2g, assuming that the geometric model resides on both machines. The quality factors, code sizes, and compression rates are given below each image. Figures 2l and 2m (on the next page of figures) give one more pair, enlarged so that details may be seen. The statistics for these two figures also appear in table I.

In every case, polygon-assisted compression is superior to image-based compression. There are two distinct reasons for this:

- The polygon-assisted rendering contains undegraded edges and smoothly shaded areas - precisely those features that fare poorly in JPEG compression.
- The difference image contains less information than the high-quality rendering, so it can be compressed using a higher JPEG quality factor without increasing code size - even higher than is required to compensate for the division by 2 in the difference image representation. Thus, texture features, which are present only in the difference image, fare better using our method.

As an alternative to comparing images at matching code sizes, we can compare the code sizes of images of equal quality. Unfortunately, such comparisons are difficult because the degradations of the two methods are different - polygon-assisted compression always produces perfect edges and smooth shading, while JPEG never does. If one allows that figure 2j generated using polygon-assisted compression is comparable in quality to figure 2d generated using image-based compression, then our method gives an additional 3x compression for this scene.

Table I also estimates the number of bytes required to generate figure 2m if the model is transmitted from server to client using the lossless compression method proposed in section 3. This size (13529 bytes) lies between the code sizes of figures 2e and 2f. Even in this case, polygon-assisted compression is superior in image quality to image-based compression, both in terms of its edges and smooth shading and in terms of the JPEG quality factor used to transmit the texture information.

† If rendered on a Silicon Graphics RealityEngine, both renderings can be performed - and the difference image computed - in a single pass through the data by remicrocoding the fragment generators.

4.2. Fine versus coarse tessellation of curved surfaces

Many algorithms for displaying curved surfaces operate by subdividing (tessellating) each surface into a mesh of small polygons. The shading applied at each polygon vertex is typically expensive - possibly including a sophisticated reflection model and texture, but the shading across a polygon is typically simple - constant or linearly interpolated. This suggests a partitioning strategy in which the client renders a surface using a coarse tessellation, and the server renders the surface twice - once using a coarse tessellation and once using a fine tessellation.

To demonstrate this, we consider a bowling pin modeled as a bicubic patch mesh (see figure 3). The geometry and surface properties are modeled in the RenderMan scene description language [Hanrahan90]. Using the REYES rendering system [Cook87], we tessellate the patch mesh twice, generating two micropolygon models with associated vertex colors.

Figure 3d shows image-based compression of the high-quality rendering using a JPEG quality factor of 15. Figure 3e shows polygon-based compression using a quality factor selected to match the code size in figure 3d, assuming that the low-quality model resides on both machines. Again, the superiority of polygon-assisted compression over image-based compression is evident, particularly along edges and in smoothly shaded areas.

4.3. Antialiased Phong-shaded versus nonantialiased Gouraud-shaded

We now demonstrate a partitioning strategy that does not work well. Our model is an extruded letter defined using a few large polygons (see figure 4) and rendered using SoftImage. The high-quality rendering uses texturing, antialiasing, Phong lighting, and Phong interpolated shading. (Although Phong shading is not supported on current high-performance workstations, it can be approximated using an environmental reflectance map.) The low-quality rendering uses Phong lighting and Gouraud interpolated shading, but no texturing or antialiasing.

Although polygon-assisted compression is still superior to image-based compression, the difference is less pronounced than in the previous demonstrations. The most obvious artifact in figure 4j is that the edges are not well antialiased. Subtracting a jagged edge (figure 4g) from an antialiased edge (figure 4f) yields an edge-like structure in the difference image (figure 4h). This structure fares poorly during JPEG compression, leaving the edge jagged in the reconstruction. This degradation will also occur to edges in textures in section 4.1, but in most applications important edges are modeled as geometry, not as texture.

The other disturbing artifact in figure 4j is a blockiness on the face of the letter. The Phong-shaded (figure 4f) and Gouraud-shaded (figure 4g) pixels on the face are similar in color, but they do not match exactly. These small differences are lost during compression, leading to the appearance of 8x8 block artifacts. These artifacts can be reduced somewhat through the application of post-processing techniques [Luo94].

For comparison, figure 5 shows a more successful partitioning of the same scene. In this case, the high and low-quality renderings differ only by the omission of texture. This partitioning is less practical, however, because it requires antialiasing on the client. An alternative partitioning would omit antialiasing on both server and client. In this case, edges would have jaggies, but these jaggies would not be compounded by JPEG artifacts.

	Room	Pin
A. High-quality rendering	fig2a	fig3a
number of polygons	1131	75,467
number of pixels	512 x 512	320 x 800
entropy (bits/pixel)	5.8	3.0
B. Low-quality rendering	fig 2b	fig 3b
number of polygons	1131	3153
C. Difference image	fig 2c	fig 3c
entropy (bits/pixel)	3.0	1.3
D. Image-based compression	fig 2l	fig 3d
JPEG quality factor	15	15
JPEG code size (bytes/frame)	9836	6030
compression ratio	80:1	127:1
error versus uncompressed (rms)	6.1	4.1
E. Polygon-assisted compression (resident model)	fig 2m	fig 3e
JPEG quality factor	41	55
JPEG code size (bytes/frame)	9759	5955
compression ratio	81:1	129:1
error versus uncompressed (rms)	7.5	2.9
F. Polygon-assisted compression (transmitted model)		
JPEG code size (from above)	9759	5955
Model size (est. bytes/frame)	3770	10510
Total size	13529	16465
compression ratio	58:1	47:1

Table I: Image-based compression versus polygon-assisted compression, compared for the scenes pictured in figures 2 and 3. In D, we select a quality factor that gives 20 frames per second while requiring 2 Mbs or less of network bandwidth. In E, we select a quality factor that matches as closely as possible the code size obtained in D, assuming that the low-quality geometric model resides on both client and server. In F, we estimate the number of bytes required to generate D assuming that a losslessly compressed low-quality model is transmitted from server to client.

5. Related work

The problem presented in this paper is a special case of two general problems: compression of geometric models and compression of synthetic images. Although these two problems have been recognized for a long time, interest in them has risen recently due to the growing synergy between digital video, computer graphics, and networking technologies.

Schemes for compressing geometric models can be categorized as lossy or lossless. Lossy schemes can be further subdivided into methods that simplify the geometry and methods that represent the full geometry using a quantized representation. Geometric simplification methods include hand-generation of hierarchical models [Clark76], automatic decimation of polygon meshes [Hoppe93], and 3D scan conversion of geometry into multiresolution voxel arrays [Wang94]. A good survey of these methods is given in [Heckbert94]. Quantized representations is largely an unexplored area; a notable exception is [Deering95] in these proceedings. Lossless compression of geometric models is also largely unexplored. The prospect of a consumer market for downloadable video games, in which the model is a significant fraction of the total game size, makes this an attractive area for future research.

Schemes for compressing synthetic images can be categorized according to the role played by the underlying geometric model. In the present paper, the model is used to partition the synthetic image into a set of polygons and a difference image. Alternatively, the model could be used to locally adapt the quantization table in a block-based compression scheme to match the characteristics of commonly occurring blocks. Adaptation could be based on block content hints from the model or importance hints from the application program. Although there is a large literature on adaptive quantization, we know of no results that incorporate information from a 3D graphics pipeline. This seems like a fruitful area for future research. Another possibility is to augment a transform coding scheme by adding basis functions that directly represent edges and bilinearly interpolated shading. The screen axis aligned rectangles of Intel's DVI PLV standard offer some of this [Luther91]. The present paper can be viewed as a generalization of this scheme to unconstrained overlapping triangles.

For animation sequences, a geometric model can be used to derive optical flow - the interframe movement of each pixel in an image. Optical flow can in turn be used to compute block motion vectors [Wallach94] or block image warps [Agrawala95]. Flow can also be used to implement non-blocked predictive coding of closely spaced views [Guenter93] or to derive an image morph that interpolates between widely spaced views [Chen93]. Among these, only [Guenter93] is lossless.

Textures and volumes provide another opportunity for combining compression and image synthesis. A lossless compression scheme for volumes based on DPCM and Huffman coding is described in [Fowler94]. Lossy schemes include vector quantization [Ning93], multidimensional trees [Wilhelms94], differences of Gaussians wavelets [Muraki94] and Haar and Daubechies wavelets [Westermann94]. Also related are algorithms for applying texture manipulation operators (such as magnifying or minifying) directly to JPEG representations of textures [Smith94].

6. Conclusions

We have described a method for using a high-performance graphics workstation as a rendering server for a low-performance workstation, and we have explored several strategies for partitioning the rendering task between client and server. Our method improves image quality over compressing and transmitting a single rendering because it removes from the transmitted image those features that compress poorly - mainly edges and smooth shading. These are instead rendered locally on the client.

Our method has several limitations. First, it will require a careful implementation to avoid excessive latency. Second, it requires either storing the low-quality geometric model on both machines or transmitting it for each frame. The former solution requires some memory in the client. The later solution depends on our ability to compress the model. Some geometric models will not compress well, and complex models will always be too large to transmit. Third, our method is most useful on scenes of moderate image complexity. For synthetic scenes whose complexity approximates that of natural scenes, our method will convey little or no advantage. Finally, our method is most useful at high compression rates (more than 25:1). If the network can support transmission of the high-quality rendering at a low compression rate, both MPEG and motion JPEG perform well enough for most interactive tasks.

Extensions to our work include investigating the compression of geometric models, employing an alternative coding technique such as wavelet-based compression, and exploring the use of polygon-assisted compression as a file format for archiving, (non-realtime) image transmission, and printing. In addition, a quantitative model of compression error that reliably captures subjective image quality is sorely needed.

Regarding the longevity of our method, while it is true that low-end machines are getting more powerful each year, there will always be a high-end machine that costs more money and delivers more performance. Thus, although the partitionings described in this paper may become obsolete, there will probably always be partitionings for which our method provides an advantage. In a similar vein, an assumption underlying our method is that compression, transmission, and decompression taken together are less expensive than rendering the original model locally on the client. Although the computational expense of compressing a pixel using transform-based coding is largely independent of image content and will probably remain constant for the forseeable future, the cost of rendering that pixel will rise as image synthesis methods become more sophisticated. This points toward a continuing niche for our method.

7. Acknowledgements

Discussions with Anoop Gupta, Pat Hanrahan, David Heeger, and Navin Chaddha were useful during this project. The suggestions of one reviewer were particularly insightful, and I have attempted to incorporate them into my revised manuscript. The possibility of remicrocoding the RealityEngine was suggested to me by Brian Cabral of Silicon Graphics. The radiosity model was provided by Reid Gershbein, and the the bowling pin model was taken from Steve Upstill's RenderMan Companion. This research was supported by the NSF under contract CCR-9157767.

8. References

[Agrawala95] Agrawala, M., Beers, A.C., Chaddha, N., ''Model-based Motion Estimation for Synthetic Animations.'' Submitted for publication.

[Chen93] Chen, S.E., Williams, L., ''View Interpolation for Image Synthesis,'' *Proc. SIGGRAPH '93* (Anaheim, California, August 1-6, 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, pp. 279-288.

[Clark76] Clark, J.H., ''Hierarchical Geometric Models for Visible Surface Algorithms,'' *CACM*, Vol. 19, No. 10, October, 1976, pp. 547-554.

[Cook87] Cook, R., Carpenter, L, Catmull, E., ''The REYES Image Rendering Architecture,'' *Computer Graphics (Proc. Siggraph)*, Vol. 21, No. 4, July, 1987, pp. 95-102.

[Danskin94] Danskin, J., ''Higher Bandwidth X,'' *Proc. Multimedia '94* (San Francisco, October 15-20, 1994), ACM, pp. 89-96.

[Deering95] Deering, M., ''Geometry Compression,'' Proc. SIGGRAPH '95 (Los Angeles, California, August 7-11, 1995), In *Computer Graphics* Proceedings, Annual Conference Series, 1995, ACM SIGGRAPH.

[Fowler94] Fowler, J.E., Yagel, R., ''Lossless Compression of Volume Data,'' *Proc. 1994 Symposium on Volume Visualization*, A. Kaufman and W. Krueger eds., ACM, pp. 43-50.

[Gershbein94] Gershbein, R., Schroeder, P., Hanrahan, P., ''Textures and Radiosity: Controlling Emission and Reflection with Texture Maps,'' Proc. SIGGRAPH '94 (Orlando, Florida, July 24-29, 1994). In *Computer Graphics* Proceedings, Annual Conference Series, 1994, ACM SIGGRAPH, pp. 51-58.

[Guenter93] Guenter, B.K., Yun, H.C., Mersereau, R.M., ''Motion Compensated Compression of Computer Animation Frames,'' *Proc. SIGGRAPH '93* (Anaheim, California, August 1-6, 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, pp. 297-304.

[Hanrahan90] Hanrahan, P., Lawson, J., ''A Language for Shading and Lighting Calculations,'' *Computer Graphics (Proc. Siggraph)*, Vol. 24, No. 4, August, 1990, pp. 289-298.

[Heckbert94] Heckbert, P., Garland, M., ''Multiresolution Modeling for Fast Rendering,'' *Proc. Graphics Interface '94* (May 18-20, 1994, Banff, Alberta), Canadian Information Processing Society, pp. 43-50.

[Hoppe93] Hoppe, H. DeRose, T., Duchamp, T., McDonald, J., Stuetzle, W., ''Mesh Optimization,'' *Proc. SIGGRAPH '93* (Anaheim, California, August 1-6, 1993). In Computer Graphics Proceedings, Annual Conference Series, 1993, ACM SIGGRAPH, pp. 19-26.

[Lane] Lane, T., *Independent JPEG Group Software Codec*, Version 4, Internet distribution, URL ftp://ftp.uu.net/graphics/jpeg.

[Le Gall91] Le Gall, D., ''MPEG: A Video Compression Standard for Multimedia Applications,'' *CACM*, Vol. 34, No. 4, April, 1991, pp. 46-58.

[Luo94] Luo, J., et al., ''A New Method for Block Effect Removal in Low Bit-Rate Image Compression,'' *Proc. ICASSP '94*, pp. V-341-344.

[Luther91] Luther, A.C., *Digital Video in the PC Environment*, 2nd edition, McGraw-Hill Book Company, New York, 1991.

[Muraki94] Muraki, S., ''Multiscale 3D Edge Representation of Volume Data by a DOG Wavelet,'' *Proc. 1994 Symposium on Volume Visualization*, A. Kaufman and W. Krueger eds., ACM, pp. 35-42.

[Ning93] Ning, P., Hesselink, L., ''Fast Volume Rendering of Compressed Data,'' *Proc. Visualization '93*, G. Nielson and D. Bergeron ed., IEEE, October, 1993, pp. 11-18.

[Rowe] Rowe, L.A., Gong, K., Patel, K., Wallach, D., *MPEG-1 Video Software Encoder*, Version 1.3, Internet distribution, URL ftp://mm-ftp.cs.berkeley.edu/pub/multimedia/mpeg.

[Smith94] Smith, B.C., ''Fast Software Processing of Motion JPEG Video,'' *Proc. Multimedia '94* (San Francisco, October 15-20, 1994), ACM, pp. 77-88.

[Wallace91] Wallace, G., ''The JPEG Still Picture Compression Standard,'' *CACM*, Vol. 34, No. 4, April, 1991, pp. 30-44.

[Wallach94] Wallach, D.S., Kunapalli, S., Cohen, M.F., ''Accelerated MPEG Compression of Dynamic Polygonal Scenes,'' Proc. SIGGRAPH '94 (Orlando, Florida, July 24-29, 1994), In *Computer Graphics* Proceedings, Annual Conference Series, 1994, ACM SIGGRAPH, pp. 193-197.

[Wang94] Wang, S.W., Kaufman, A.E., ''Volume Sampled Voxelization of Geometric Primitives,'' *IEEE Computer Graphics and Applications*, Vol. 14, No. 5, September, 1994, pp. 26-32.

[Westermann94] Westermann, R., ''A Multiresolution Framework for Volume Rendering,'' *Proc. 1994 Symposium on Volume Visualization*, A. Kaufman and W. Krueger eds., ACM, pp. 51-58.

[Wilhelms94] Wilhelms, J., Van Gelder, A., ''Multi-Dimensional Trees for Controlled Volume Rendering and Compression,'' *Proc. 1994 Symposium on Volume Visualization*, A. Kaufman and W. Krueger eds., ACM, pp. 27-34.