Compression of Computer Graphics Images with Image-Based Rendering

Ilmi Yoon and Ulrich Neumann
Computer Science Dept.
University of Southern California
Los Angeles, CA90089

ABSTRACT

We present a new compression algorithm for synthetic images that produces high compression rates by utilizing depth and color information from previously rendered images. Images predicted from prior images are combined with a residual image that may be transmitting from a remote location, to generate complete images. The image-based rendering technique provides accurate motion prediction and accelerates rendering at the same time by exploiting temporal coherence. The motion prediction is computed and evaluated in image-order, pixel by pixel, producing residual images that are sparse and do not require address or index data. The system yields a compression ratio improvement of a factor of 4-10 over MPEG, in many cases. This approach is attractive for remote rendering applications where a client system may be a relatively low-performance machine and limited network bandwidth makes transmission of large 3D data impractical. The efficiency of the server generally increases with scene complexity or data size since the rendering time is predominantly a function of image size. This technique is also applicable to archiving animation.

Keywords: computer graphics, geometry, compression, remote rendering, image-based rendering, rendering acceleration, internet application, and archiving.

1. INTRODUCTION

The intensive use of photo-realistic computer graphics and scientific visualization has spurred the development of large complex 3D models. Three-dimensional surface models routinely surpass 100 million polygons, and volume data may be even larger; for example, the male "Visible Human" model[1], contain 122 Mvoxels of MR data, 490 Mvoxels of CT data, and about 14 Gvoxels of photo-texture data. Despite advances in the hardware and algorithms for 3D model rendering, the diversity of graphics applications reveals algorithmic deficiencies and identifies new problems. For example, graphics for internet applications spurred the development of VRML for remote interaction with, and compression of, 3D[2, 3, 4]. Despite the progress in VRML revisions, the compression and transmission of large models remains a challenging problem and rendering entirely depends on local system's capability.

Networks and rendering systems capable of such large data will always be relatively rare and expensive. Therefore, it is useful to consider ways to reduce both the network bandwidth and rendering burden associated with remotely visualizing these data. The following system features are desirable for interactive remote visualization and internet applications:

- the 3D model resides at the host and is not transferred to the remote client
- minimal network bandwidth is required for each frame whether it is interactively selected or replayed from a pre-computed animation
- minimal computing resources are needed at the remote client

The minimal bandwidth requirement is also desirable for reducing the storage requirements for archived animation sequences.

In a compression strategy targeted at remote rendering, a high-performance graphics engine acts as a server, and a low-end workstation as a remote client[5]. The client renders low-quality images while the server renders both low- and high-quality images, transmitting only compressed residual images to the client. This approach assumes that the client first receives a copy (or simplified version) of the scene data before rendering begins. The server does not benefit from any acceleration; in fact, the server computes a high quality image, a low quality image, a residual image, and a compressed residual image.

Considering the popular image-compression methods such as MPEG[16, 17, 18], synthetic image animations have to be rendered to the image sequences a prior and dose not provide the interactivity. However, they typically achieve compression ratios of 10-100 and enables real time replay. MPEG is a lossy compression algorithm that uses block motion compensation to exploit the coherence between frames[6]. Techniques in JPEG and MPEG are mainly designed for video, not graphics. Video images differ from most graphics in their spectral and noise content, often masking artifacts that are clearly visible in graphics images. In addition, not many works are done to take advantage of significant information obtained in rending synthetic images.

Guenter[7] and Yun[8] used transformation information available in animation scripts and geometry data in each pixel to enhance the compression ratio of sequences of pre-rendered images. Pixel motion is predicted by the script transformations, and the difference between the predicted and target images is stored in a residual image. The residual image is sparse and has low entropy, so compression of the residual with entropy coding methods achieves a factor of 1.4 over JPEG performance. However, the techniques require a complete set of rendered images before compression can begin.

Image-based rendering (IBR) introduces motion prediction within the rendering process. IBR trades transformations and shading computations for re-projection or warping of images[9, 10]. Our work extends IBR techniques to compression. We reason that since IBR methods extract spatiotemporal coherence between images for acceleration, the same coherence data can also serve as motion prediction for compression. Image compression methods search for coherence in 2D image sequences, after the rendering process. Our method merges the rendering and compression functions to accelerate rendering of compressed images. An important motivation behind the method is to increase efficiency by directly rendering compressed images.

Our method exploits sampled geometry information (= previously rendered image) to effectively predict pixel motion and increase the compression ratio for computer generated animation. The prediction is computed by an image-based rendering method that was developed to accelerate image synthesis by a factor of 2-4 in many cases[11], and acceleration has been more improved up to 4-7 times with the code optimization.

The use of previously rendered image is conceptually similar to the use of I-frames in MPEG for the compression purpose. So, we call those images as I-frame or reference images since we also use I-frames as reference images from which other images are predicted. However, instead of projecting each block or pixel of I-frame to a frame to be compressed (forward projection), each pixel of the frame is projected back to the I-frame (backward projection). Therefore, pixel motion is computed and assessed in image-order. High confidence pixels have their color and depth interpolated directly from prior images. Pixels with low-confidence motion estimates are computed by traditional rendering and included in the residual image. Since the sequence of residual pixels is known at both the local and remote sites, there is no need to

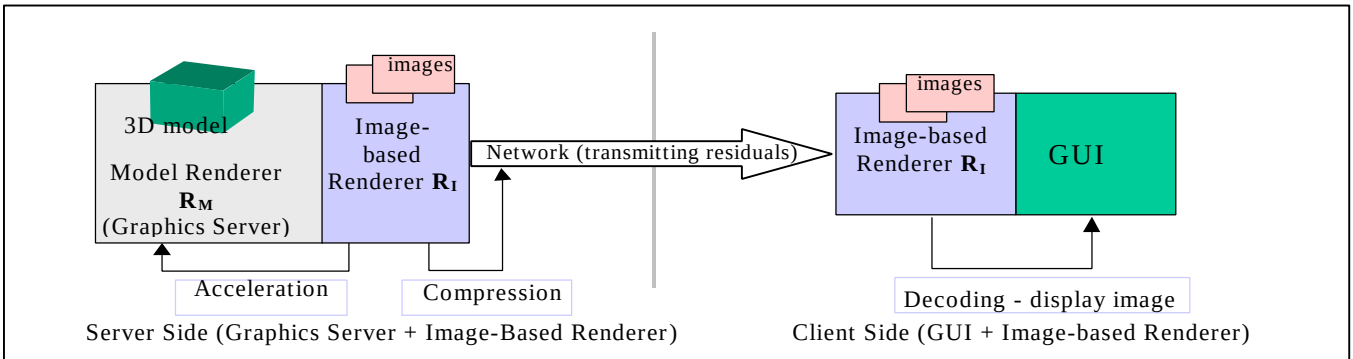


Fig 1. – Remote Rendering System. The image-based rendering system on the server side accelerates graphics rendering of compressed images. The image-based rendering system on the client side decodes the compressed data stream for display.

transmit address or index data in the residual image. This is more beneficial as the residual image gets sparser. This unique advantage shrinks our sparse residual image's size dramatically, increasing the compression ratio over the same residual image in format of lossy JPEG (~75% quality) or Gif by an order of magnitude or more, resulting overall compression ratio improvement of a factor of 4-10 over MPEG, in many cases. It makes this algorithm very effective for application to remote rendering and internet applications where a client system may be a relatively low-performance machine and limited network bandwidth makes transmission of large 3D data impractical. Since received I-frames can persist at the remote client, new images may be generated efficiently from a single or set of I-frames, whether the path of the camera movement is known a prior or not.

The remainder of this paper is organized as follows: Section two presents the overall system and some notation. Section three describes algorithm how to predict the pixel motions from I-frame and generate residual images. Section four describes the encoding and decoding sequences. Finally, we present results and discuss future research directions.

2. SYSTEM OVERVIEW

The equations below are a functional expression for the overall compression and decoding process. An index 'i' or 'j' indicates a frame from a sequence. Capital "**R"** stands for the renderers and "**I"** means the images. The functional relationships between processes are detailed below and the data flow is illustrated in figure 2.

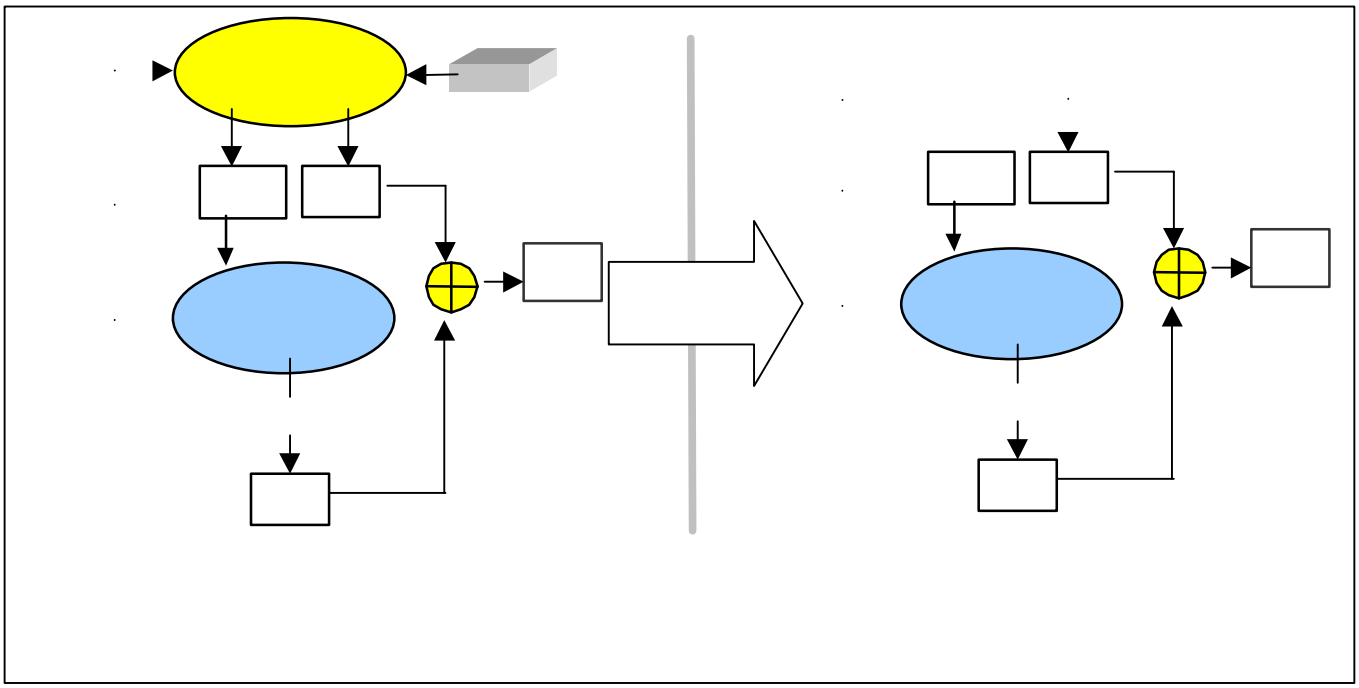
A *Model Renderer* ($\mathbf{R_M}$) is a traditional renderer that uses a 3D model to synthesize images. In our implementation, $\mathbf{R_M}$ is a ray tracer that renders volume data and polygons at the same time. *I-frames* ($\mathbf{I_I}$) are images generated from the model renderer (Eq. 1). These images contain color, depth, and object-ID at each pixel. *Extracted Images* ($\mathbf{I_E}$) are created by the *Image-Based Renderer* ($\mathbf{R_I}$) from a given I-frame (Eq. 2). The *Residual Image* ($\mathbf{I_R}$) is the set of pixels for which the image-based renderer cannot find good motion estimates for the *Extracted Images* ($\mathbf{I_E}$) from a given I-frame. Each pixel value of the *Residual Images* ($\mathbf{I_R}$) is computed by the model renderer (Eq. 3). Therefore, the *Final Image* ($\mathbf{I_F}$) is created by combining an extracted image $\mathbf{I_I}$ and a residual image $\mathbf{I_R}$ (Eq. 4).

$$\mathbf{I_I}[j] = \mathbf{R_M}(\text{Data}_{3D}) \qquad \text{(Eq. 1)}$$

$$\mathbf{I_E}[i] = \mathbf{R_I}(\mathbf{I_I}[j]) \qquad \text{(Eq. 2)}$$

$$\mathbf{I_R}[i] = \mathbf{R_M}(\text{Data}_{3D}) \cap \overline{\mathbf{I_E}}[i]) \qquad \text{(Eq. 3)}$$

$$\mathbf{I_F}[i] = \mathbf{I_E}[i] \cup \mathbf{I_R}[i] \qquad \text{(Eq. 4)}$$



The server has the entire 3D model, a model renderer ($\mathbf{R_M}$), and an image-based renderer ($\mathbf{R_I}$), and it is capable of creating all the image sequences of final images ($\mathbf{I_F}$). The remote client has only the image-based renderer ($\mathbf{R_I}$) and one or more I-frames ($\mathbf{I_I}$) received from the server. The client generates final images ($\mathbf{I_F}$) by merging the extracted images ($\mathbf{I_E}$) it computes with the residual images ($\mathbf{I_R}$) received from the server.

The final images at the server and client systems are the same ($\mathbf{I_F}$[i] at server = $\mathbf{I_F}$[i] at client). Thus, the compression is lossless in some sense. However, the images generated by the model renderer are not identical to those generated by the total system ($\mathbf{I_F}$[i] != $\mathbf{I_I}$[i]). This is due to the errors introduced by using sampled geometry and shading in the image-based renderer. Although it is more likely rendering issues, the net result is a lossy compression system. Image quality issues are discussed in more detail in section five.

3. IMAGE BASED RENDERER (R_I)

In this section, we describe how extracted frames ($\mathbf{I_E}$) are generated from I-frames ($\mathbf{I_I}$) and how the same process also accelerates the rendering speed. A high compression ratio and higher acceleration is achieved when the residual image ($\mathbf{I_R}$) contains a small number of pixels, and that depends on how much information the image-based renderer is able to extract from $\mathbf{I_I}$ during its reconstruction of $\mathbf{I_E}$. Therefore, the efficient design of image-based renerer is essential. This image-based renderer conceptually works like a ray caster [12, 13]. For each pixel of a frame, a view ray is cast from the camera into the scene. Figure 3 illustrates how view rays are projected onto an I-frame (with object ID, depth, and color for each pixel). Projected lines are searched for intersections with the sampled geometry. Information is extracted from the I-frame along a ray (line AB of Fig. 3) by looping across the pixels that lie along the projection of the ray (line A'B' of Fig. 3) until the ray's depth exceeds the pixel's depth: pixels whose areas the ray falls within are traversed, along the ray's direction, with the ray's computed depth at each pixel being tested against the pixel's depth. If the ray's depth exceeds the pixel's depth, then we assert that the ray has passed behind the pixel's surface, which may be an intersection.

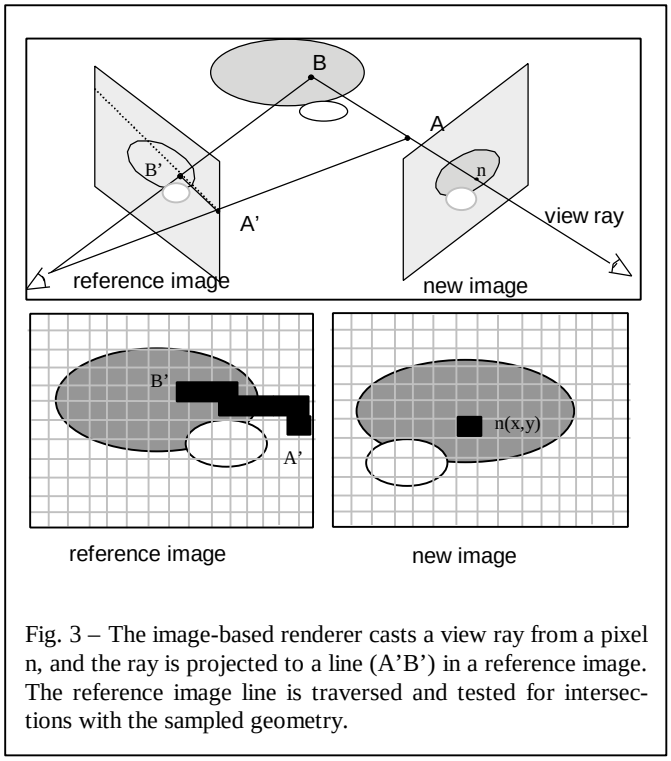
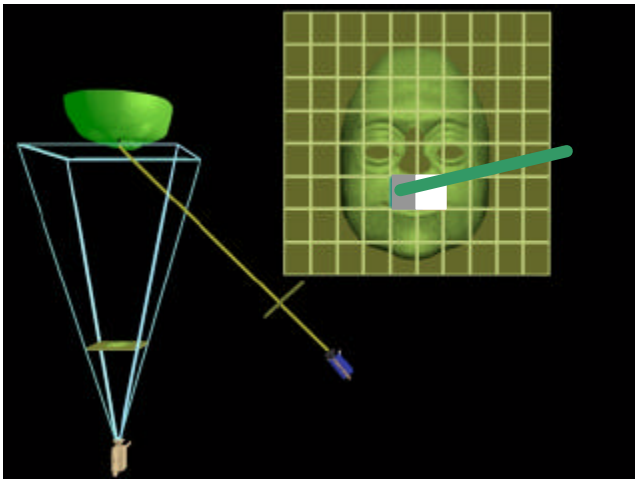
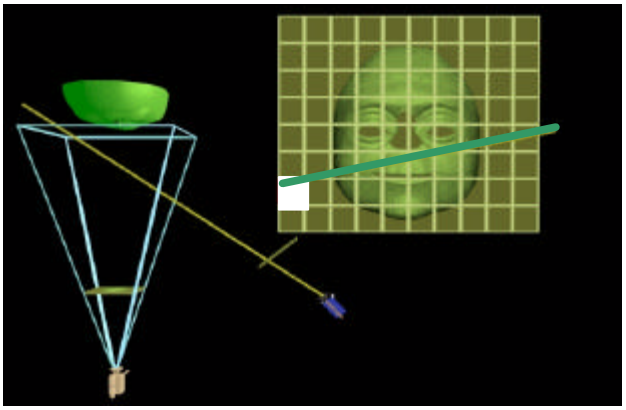


Fig. 3 – The image-based renderer casts a view ray from a pixel n, and the ray is projected to a line (A'B') in a reference image. The reference image line is traversed and tested for intersections with the sampled geometry.

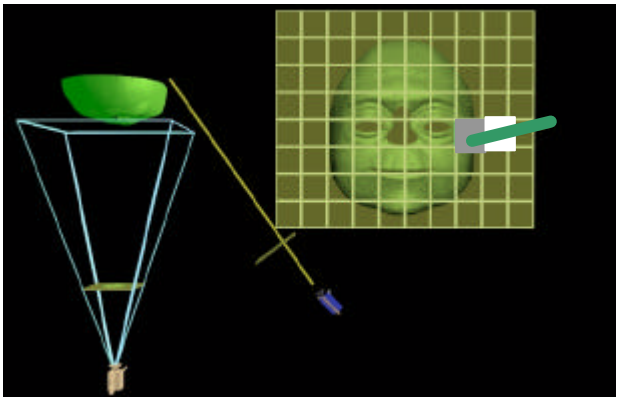
These intersections determine an inverse motion for each pixel in a new frame back to a position in an I-frame. An important aspect of this inverse motion calculation is that it facilitates an estimation of both motion and confidence. A confidence metric is essential for determining where residual-image pixels must be computed. Confidence values correspond to the intersection cases illustrated in figure 4 as definite miss, definite hit or possible hit. A definite hit is indicated (Fig. 4a) when a ray crosses from in-front to behind a surface: Right side grid image is the I-frame and the ray on top of the grid is the view ray casted from the new camera position (blue camera) and projected to I-frame. The white pixel is the last pixel that the ray is in front of the stored depth of the pixel along the ray. The gray pixel is the first pixel that the ray goes behind. Therefore it crosses from the front to the behind of the surface, resulting an intersection, definite hit. A definite miss (Fig. 4b) occurs for a ray that fails to pass behind any samples. Ray is always in front of the pixel's depth. Possible hits occur when a ray suddenly appears behind a surface it was never in front of (Fig. 4c), or when a ray exits the image before leaving the scene's bounding volume. Definite misses and definite hits are interpreted as high-confidence motion estimations. Misses cause pixels to be assigned a background color and depth. Hits cause pixel color and depth to be interpolated from neighboring samples, allowing the current pixel to be efficiently shaded and Z-buffered without explicitly doing ray-object intersection tests, that are relatively



(a) definite hit



(b) definite miss



(c) possible hit

Fig. 4 - Possible motion-prediction confidences during image-based rendering. Image with the grid is the I-frame and it is a rendered image from the view point of white camera. Rays from a new camera position (blue camera) are traversed as lines in I-frames. Rays pass in front of (pixels upto white pixel along the view ray) and behind (gray pixels) sampled geometry. Therefore, gray pixel next to white pixel implies intersection of ray and surface.

expensive operations especially when the data size is huge. Possible hits are low-confidence motion estimations. On the server (encoding) system, the point along the ray traversal where the surface may have been hit is provided to the model renderer $\mathbf{R_M}$ as an estimate of where to search for a true intersection. The model renderer is responsible for completing the rendering of the pixel and including its color and depth value in the residual image. On the client (decoder) side, the next available pixel values in the residual image are decoded and assigned to the current pixel.

Potential aliasing problems with this and other image-based rendering techniques arise from the use of sampled geometry and color. We use an object ID to alleviate some of the aliasing by relying on the assumption that two neighboring I-frame pixels with the same object ID are connected by a "smooth" surface. Our definition of "smooth" is informal and simply that interpolated color and depth approximate true color and depth. Non-smooth examples include concavities or protrusions that violate the interpolation assumption. In polygon data, those non-smooth area polygons have different object ids. However, volume data can have concavities or discontinued area within one object. Fortunately, finding these concavities is a tractable problem. If the depths of the two neighboring pixels differ by more than a certain threshold like one voxel distance, it is likely that this corresponds to a concavity, and so *possible hit* should be returned. Otherwise, the two pixels are on similar points on the surface, and points between them do not diverge greatly. Another image-based rendering problem arises with moving objects in the scene. The current implementation only deals with a moving camera, although we feel that moving objects can be handled as separate scenes that are Z-buffer composited.

4. COMPRESSION ALGORITHM

4.1 Encoding

Encoding is performed during rendering and actually accelerates the rendering process (Fig. 5). The model renderer uses 3D data (polygons or volumes) to generate one or more I-frames (color, depth, object ID). Image-based rendering extracts motion data from the I-frame(s) to generate new images. The model renderer computes the low confidence pixels in each frame and encodes them into a sparse residual image. Since the image-based motion estimation is nominally faster than model rendering, the combined process is often faster (4-7 speedup) than model rendering by itself .

The I-frames and residual images are compressed before transmission. The color data in I-frames can be compressed by many methods. Depth information is treated as geometric data and it may be compressed using motion prediction and

a direction coding technique, typically to 1 to 2 bits per pixel.[8] When volume data is rendered instead of polygons, it does not need an object ID, improving the I-frame compression ratio. For our residual images, same spatial redundancy algorithms used for I-frame can be used for effective compression because the data is very sparse. Our renderer, however, produces the output image motion-estimates in the same order during the encoding and decoding phases. This allows us to eliminate the offset or index values from the residual image and compresses it with gzip, an implementation of Lempel-Ziv encoding[14]. The result is a highly increased compression ratio when compared with spatial redundancy algorithms. As seen in figure 6, the compression ratio of our residual image is 1/10 the size of a GIF image and 1/100 of a lossless JPG image. This is indicative of the overall system performance since the transmitted data consists mainly of pure data of the residual images.

4.2 Decoding

Decoding process is very straightforward. I-frames and residual images are decompressed first. Then, the I-frames are used by the image-based renderer to generate extracted frames. When the image-based renderer recovers a low confidence motion estimate (possible hit) for a pixel, it fetches the next values from the residual image, in order to complete the pixel.

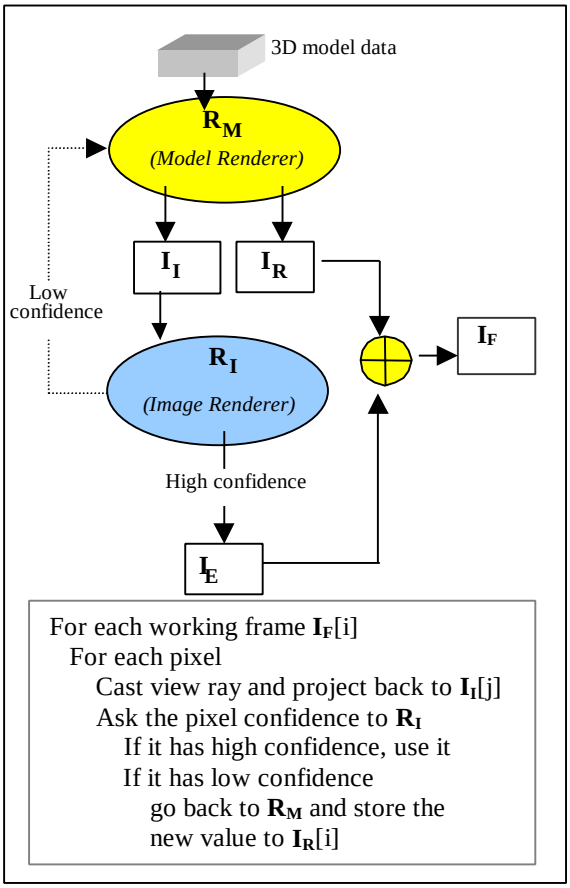


Fig. 5 – Encoding: (1) 3D models are rendered in the *model renderer* generating *I-frames*. I-frames store geometry and color information for each pixel. (2) *Image renderer* generates extracted frames from I-frames for pixels having high confidence motion, and (3) hands off to the *model renderer* low confidence pixels to be included in the residual image. I-frames and residual images are the data

5. RESULTS

Test data includes polygon data of varied complexity and volume data. We generated animations by moving the camera around the objects. As seen in Plate 1, the test sequences exhibit rapid viewpoint changes over several different data types. The compression ratios of the residual images for the four sequences in Plate 1 were compared to lossless JPEG, lossy JPEG at 75% quality, and GIF encodings of the same residual images (Fig. 6). Use of the entropy coder yields a compression ratio over ten times higher than the alternatives. We also compared the image sequence shown in figure 10 with a lossy MPEG-1 encoding (using Berkley encoder, version 1.5, 1995 at default quality). The animation was encoded using the brain data of Table 1(a) on an SGI Onyx2 R10K CPU. Our method achieved a compression ratio about 4-10 time better than the MPEG encoding (Fig. 7). As shown in figure 10, even with substantial camera movements, our residual images are small whereas the MPEG compression ratio is relatively poor in comparison. This characteristic arises because the motion prediction is robust for large motion; keeping the residual image sparse, as long as the camera remains trained in the vicinity of a reference image, regardless of zoom, rotation, or translation.

		Data		Result	
	Type	size	Image size	Bits/pixel	SNR
(a)	Volume	128x128x84	512x512	0.025574	41.78 dB
(b)	Volume	256x256x128	256x256	0.070679	31.8 dB
(c)	Polygons	897triangles	512x512	0.01593	38.17 dB
(d)	Polygons	33264 triangles	400x400	0.0454	37.04 dB

Table 1 – Size of polygon and volume data and results for images shown in Plate 1.

The image quality from the local renderer and the remote renderer are the same, as explained before. Compared to a stand-alone image-based renderer, our images are superior since our system integrates the model renderer to create the residual images that improve image quality substantially. However, overall system is a lossy compression method. The interpolated depth and color from reference images is different from what the model renderer would give. Diffuse surfaces are more forgiving than specular ones. These problems are encountered with other image-based-rendering methods and discussed in papers[9, 10, 15]. However, we find these artifacts to be subjectively minor and objectively, the images exhibits a better SNR than the MPEG-1 encodings. Figure 8 shows the SNR for each frame of the brain sequence. Images in figure 10 and plate 1 do not show visible degradation. Initial reference image is compressed without loss, so it has infinite SNR.

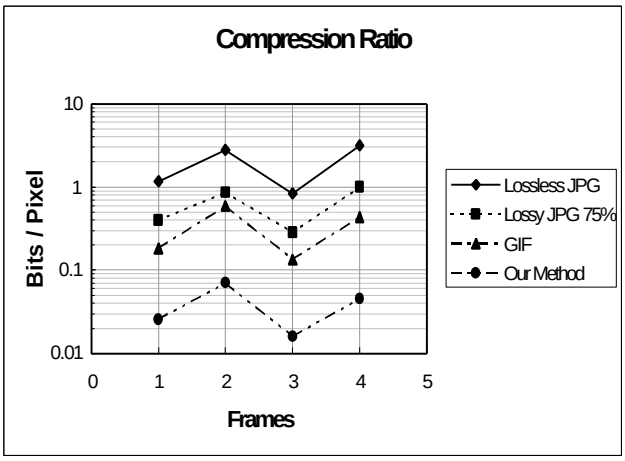


Fig. 6 - Compression ratio comparison of a residual image – Lossless JPG/ Lossy JPG of quality 75%, GIF and our method.

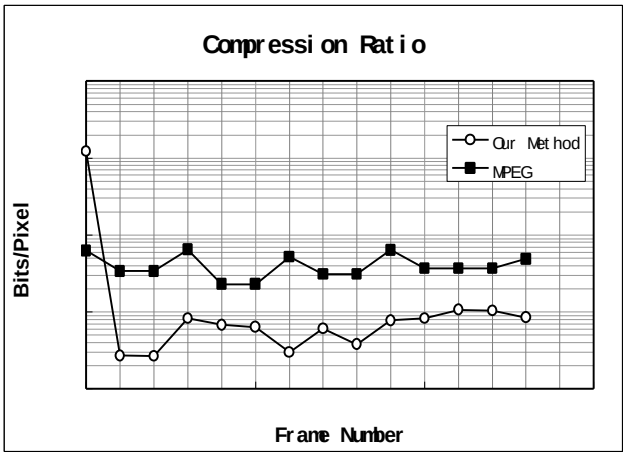


Fig. 7 - Compression ratio comparison with MPEG. (image sequences used for animation are shown in figure 10. Animations are available at http://www-scf.usc.edu/~iyoon)

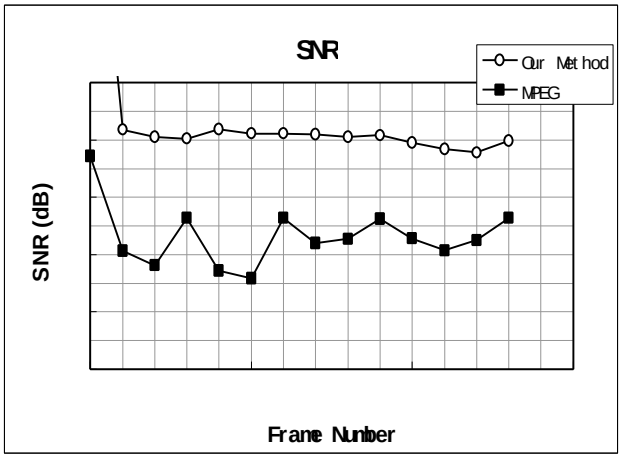


Fig. 8 - Image quality – Signal to noise ratio comparison with MPEG.

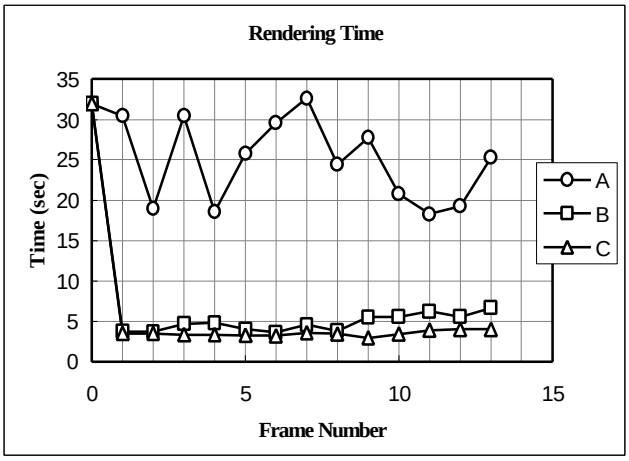
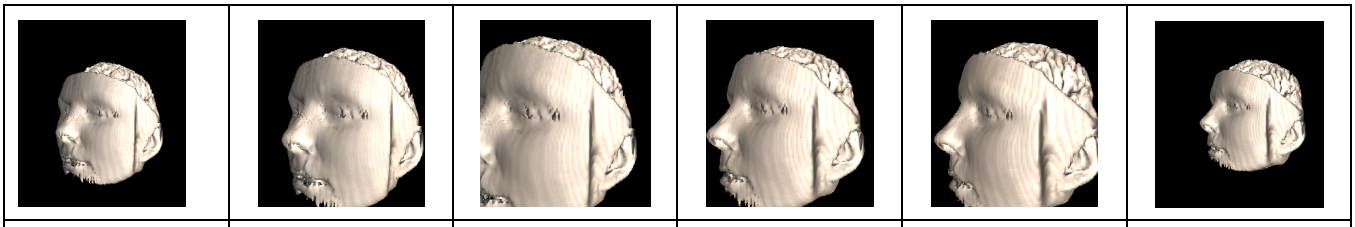


Fig. 9 - Rendering speed up of animation sequence: **A**: model renderer using 3D data, **B**: accelerated method (I_E + I_R == renderer + encoder), and **C**: accelerated method (I_E + I_R == renderer + decoder)

Figure 9 illustrates the image-based rendering acceleration of rendering speed. The IBRAC method shows a speedup of four or more in encoding and seven or more in decoding. Decoding is faster since the residual is received rather than

computed. As expected, the rendering time for the remote client side (decoding) is largely dependent on the image size, rather than the data set complexity. In the implementation, the model renderer and the image renderer use a simple ray-casting algorithm. Employing a ray-tracing acceleration technique will enhance rendering speeds (e.g., spatial subdivision by an octree).



(a) (b) (c) (d) (e) (f)

Fig. 10 - Images from the test animation sequence. Note the use of scaling and rotation.

6. DISCUSSION AND FUTURE RESEARCH

This paper presented a compression approach inspired by an image-based rendering acceleration algorithm. This combination can be effectively used for compressing synthetic image sequences. Once I-frames are generated, new views of the scene are estimated and residual images are compressed into less than 0.1 bits/pixel in many cases. This performance comes from robust motion estimation and confidence assessment. The benefits to remote rendering include low bandwidth for new images; low computation for decoding and encoding; and eliminating the need to send the 3D model data over a network.

In our future research, we plan to develop efficient use of multiple I-frames. Possible approaches include: (1) to subdivide the image into blocks and predict the most reusable I-frame for each block; (2) to develop an automatic mechanism to optimize the frequency and sequence of I-frames for a given animation script; (3) we also may consider the P-frame concept as used in MPEG

ACKNOWLEDGEMENTS

This research has been funded by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center with additional support from the Annenberg Center for Communication at the University of Southern California and the California Trade and Commerce Agency. We thank Antonio Ortega, (EE Dept.) for reviewing the text and we thank Doug Fidaleo and Clint Chua for editing assistance. Special thanks go to Joe Demers and Taeyong Kim for many of the initial ideas and guidance in the implementation.

REFERENCES

1. Visible Human Project web page http://www.nlm.nih.gov/research/visible/visible_human.
2. Djurcilov, S., Pang, A., "Visualization Products On-Demand Through the Web", VRML 98 Third Symposium on the VRML, pp 7 - 24
3. *The Internet in 3D Informarion, Images and Interaction* / edited by Earnshaw, R. and Vince, J., Academic Press, 1997
4. ISO/IEC 14772-1, The Virtual Reality Modeling Language, http://www.vrml.org/VRML97/DIS/, 4 April 1997
5. M. Levoy, "Polygon-Assisted JPEG and MPEG Compression of Synthetic Images", Computer Graphics, (Proc. SIGGRAPH 95 pp 21-28)
6. *MPEG video compression standard* / edited by Joan L. Mitchell, New York : Chapman & Hall, 1997

7. Guenter, B.K., Yun, H.C. and Mersereau, R.M., "Motion compensation compression of computer animation frames", Computer Graphics (Proc. SIGGRAPH 93, pp 297-304)

8. Yun, H.C., Guenter, B.K., and Mersereau, R.M. "Lossless Compression of Computer-generated Animation Frames", ACM Transactions on Graphics Vol 16, No. 4, October 1997, pp 359-396

9. Chen, E., and Williams, L., "View Interpolation for Image Synthesis", Computer Graphics (Proc. SIGGRAPH 93, pp 279-288)

10. McMillan, L. and Bishop, G.,"Plenoptic Modeling: An Image-Based Rendering System", Computer Graphics, (Proc. SIGGRAPH 95, pp 39-46)

11. Yoon, I., Demers, J., Kim, T.Y. and Neumann, U., "Accelerating Volume Visualization by Exploiting Temporal Coherence", (Proc. IEEE Visualization 97 LBHT, pp 21-24)

12. Kajiya, J.T. and von Herzen, B.P., "Ray tracing Volume Densities", Computer Graphics, (Proc. SIGGRAPH 84 pp 165-174)

13. Whitted, T., "An Improved illumination model for shaded display", Communications of ACM, Vol 23, No. 6, June 1980, pp. 343-349

14. Ziv, J., Lempel, A., "A universal algorithm for sequential data compression," IEEE Transactions on Informaion Theory, IT-23:337-343, 1977

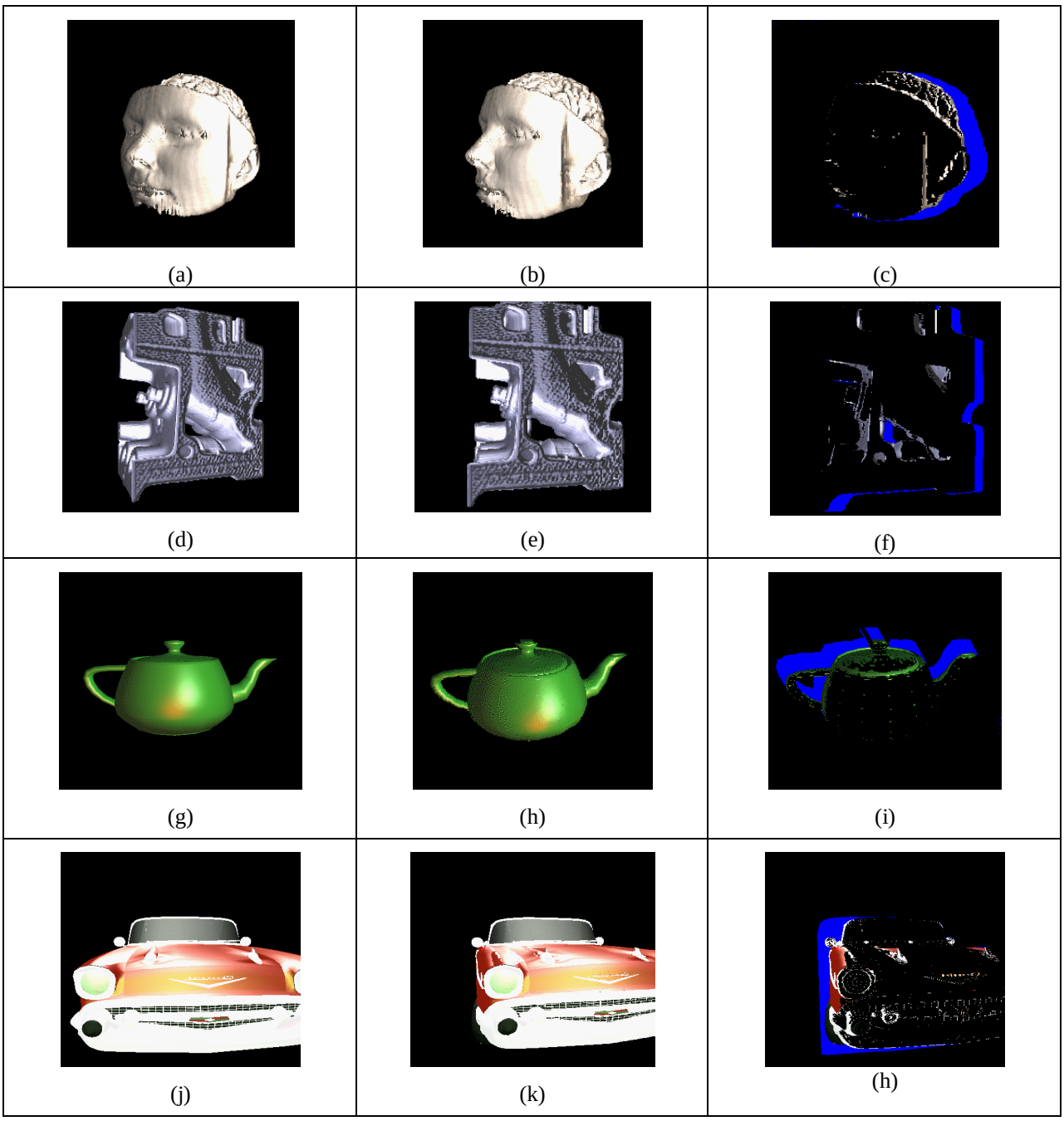
15. Demers, J., Yoon, I., Kim, T.Y. and Neumann, U., "Accelerating Ray Tracing by Exploiting Frame-to Frame Coherence," USC Technical report No. 98-668, http://www-scf.usc.edu/~iyoon/publication/usc-tecq.html

16. Hidaka, T., Ozawa, K., "Subjective assessment of redundancy-reduced moving images for interactive applications: Test methodology and report," Signal Processing: Image Commun. 2, 2 (Aug. 1990)

17. Liou, M.L., "Overview of the px64 kbps video coding standard," Commun. ACM 34, 4 (Apr. 1991)

18. MPEG proposal package description. Document ISO/WG8/MPEG/89-128 (July 1989)

Correspondence: Email: iyoon@graphics.usc.edu; WWW: http://www-scf.usc.edu/~iyoon



(a) (b) (c)

(d) (e) (f)

(g) (h) (i)

(j) (k) (h)

Plate1 - Images (a), (d), (g), (j) (left column) are used as reference imagees (I-frame), and images (b), (e), (h), (k) (center column) are images generated from $\mathbf{R_I}$ (image-based renderer) and (c), (f), (i), (l) (right column) are residual images. The blue regions show pixels with low confidence motion estimates from $\mathbf{R_I}$, that did not intersect objects or surfaces. The colored regions show low confidence pixels that were rendered by the model renderer $\mathbf{R_M}$.