EUROGRAPHICS 2000 / M. Gross and F.R.A Hopgood
(Guest Editors)
Volume 19 (2000), Number 3

Web-Based Remote Rendering with IBRAC
(Image-Based Rendering Acceleration and Compression)

Ilmi Yoon and Ulrich Neumann
Computer Science Department, Integrated Media Systems Center
University of Southern California Los Angeles CA 90089

Abstract

Recent advances in Internet and computer graphics stimulate intensive use and development of 3D graphics on the World Wide Web. To increase efficiency of systems using 3D graphics on the web, the presented method utilizes previously rendered and transmitted images to accelerate the rendering and compression of new synthetic scene images. The algorithm employs ray casting and epipolar constraints to exploit spatial and temporal coherence between the current and previously rendered images. The reprojection of color and visibility data accelerates the computation of new images. The rendering method intrinsically computes a residual image, based on a user specified error tolerance that balances image quality against computation time and bandwidth. Encoding and decoding uses the same algorithm, so the transmitted residual image consists only of significant data without addresses or offsets. We measure rendering speed-ups of four to seven without visible degradation. Compression ratios per frame are a factor of two to ten better than MPEG2 in our test cases. There is no transmission of 3D scene data to delay the first image. The efficiency of the server and client generally increases with scene complexity or data size since the rendering time is predominantly a function of image size. This approach is attractive for remote rendering applications such as web-based scientific visualization where a client system may be a relatively low-performance machine and limited network bandwidth makes transmission of large 3D data impractical.

Keywords : Image-based rendering, compression, MPEG, remote-rendering, , web-based visualization

1. Introduction

Recent advances in Internet and WWW techniques motivate desire for systems with interactive 3D graphics on the web. They draw attention and effectively assist human understanding with spatial intuitiveness and interactivity. However, interactive 3D graphics on the web reveals new problems in both traditional rendering and compression techniques especially when model size increases. For example, VRML has been developed for remote interaction with, and compression of, 3D models [Djurcilov98] [Earnshaw97] [VRML]. VRML, by itself, provides various interactions with 3D worlds such as virtual environments and architectural walk-throughs. However, large 3D models are commonplace, and despite the progress in VRML revisions and geometry compression [Li98], the transmission of large models remains a challenging problem. Model transmission introduces startup latency, and after transmission, the rendering performance is unpredictable since it depends on the client configuration. These are undesirable features for web-based applications.

Interactive 3D graphics on the web promotes easy access to high performance visualization systems for many users who are not able to otherwise have access to high computation power and large data capacity. Many scientific measurement and simulations produce very large data sets that easily reach tens of Giga Bytes [VHM][NAO]. To efficiently use scientific visualization over the web, the following system features are desirable:

- The 3D model should reside at the host and not be transferred to the remote client.
- Minimal network bandwidth should be consumed for each frame, whether interactively defined or replayed from a predefined animation.
- Modest computing resources are expected at the remote client.

The presented technique, IBRAC (Image-Based Rendering Acceleration and Compression) is a technique that has several benefits to remote or web-based rendering systems by providing such features. Its efficiency increases with scene complexity or model size because it reduces performance dependency on model data size.

In related work, many web-based visualization applications are implemented by combinations of JAVA, CGI handlers, and VRML [Trapp97]. Java or CGI handlers dynamically create VRML files from data files according to the user's input and then transmits the results to the remote user for browsing. To generate the VRML files in real time, the VRML files tend towards low complexity

for fast computation, transmission and rendering, meaning coarse models with low detail geometries and texture.

In [Levoy95], a high-performance graphics engine acts as a server, and a low-end workstation as a remote client. 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.

In [Cohen-Or97][Mann97], another strategy is presented for the partition of the rendering task in which the client is able to generate several frames autonomously without the transmission of residual images. The client receives visible portions of 3D data (only geometry, not textures) and an initial image. Images are rendered into a Z-buffer and backward-projected to the initial image to fetch the texture color. As a tradeoff between bandwidth and image quality, only selected pixels are transmitted to perform corrections in the most significant areas. To predict the area of selected pixels, relevant portions of model data are identified, transmitted and rendered in the client.

The transmission of sequence of images, rather than models, reduces startup latency. However, it reduces interactivity and traditional compression methods which 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. Furthermore, motion-based compression methods (e.g., MPEG) can not take advantage of significant information available during the rending synthetic images and require one or more sets of complete rendered images before compression can begin.

Recently introduced Image-Based Rendering (IBR) utilizes motion prediction within the rendering process. IBR reduces transformations and shading computations by re-projection or warping of images [Chen93] [McMillan95]. The efficiency of IBR generally increases with scene complexity, since the rendering time is predominantly a function of image size. As an example, in view interpolation [Chen93] the motion of pixels (i.e. the optical flow) from one camera position to another produces smooth interpolated images. Neighbor-pixel interpolation fills the holes that occur in new images from occlusion or subsampling.

IBRAC extends IBR techniques to compression. We reason that since IBR methods extract temporal coherence between images for acceleration, the same coherence data can also serve as motion prediction for compression. An important motivation behind the IBRAC method is to increase rendering and compression efficiency by directly rendering compressed images without transmitting model data sets. These combined features make IBRAC attractive to Internet applications.

IBRAC computes pixel motions by searching prior images (with depth information). But unlike MPEG, which computes motions from consecutive pre-rendered images, IBRAC computes motions (reusable pixels and their locations in new image) from any prior image and a related viewing transformation. The IBRAC method is applicable to both predetermined sequences and interactive control (e.g., remote visualization or Internet applications). In an IBRAC remote rendering application, the server has the 3D model and the remote client does not (Fig. 1). The fact that no model needs to be resident or transmitted to the client is significant since it reduces the startup bandwidth and delay in displaying the first remote image. The component called Image-based Renderer in server and client accelerate the rendering process and compress rendered images (in server) and decompress (in client). The performance of an IBRAC system with a given image size is predominantly a function of server's accelerated rendering time, not a function of client's rendering capability or model data size.

The remainder of this paper is organized as follows: Section two describes the algorithm; first system overview, second accelerated rendering and third the integration of the rendering and compression system. In section three we present results. We discuss future research directions in section four.

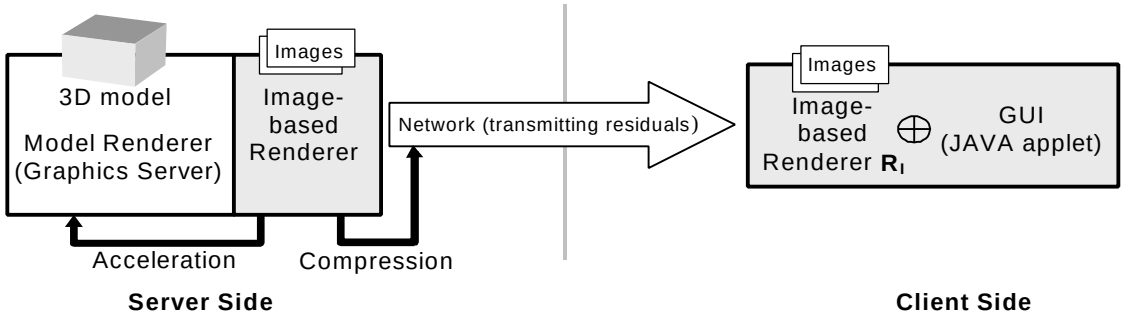


Fig 1. – IBRAC Remote Rendering System. The image-based rendering system on the server side accelerates graphics rendering of compressed images. The image-based rendering system on the client side decodes the compressed data stream for display.

2. Algorithm

2.1. System Overview

The hybrid IBR system includes two rendering subsystems (Fig. 2). One is a model renderer ($\mathbf{R_M}$) which resides only in server, and has access to the 3D geometry as shown in Fig. 2. Its current implementation is based on ray casting as described by Appel and enhanced by Whitted. Rays are cast from a camera position and pass through pixels of the image plane, intersecting 3D objects in the scene.

The second element is an image-based renderer ($\mathbf{R_I}$) that resides in both server and client, performing acceleration and compression. $\mathbf{R_I}$ tests for ray intersections in the reference images which has color and depth values at each pixel; no object data is necessary to be stored. It generates a new image by extracting reusable information from previously rendered images (through intersection test) while some pixels are not filled due to lack of reusable information. To generate a complete derive image, $\mathbf{R_M}$ is called for pixels where $\mathbf{R_I}$ fails to extract reusable information. These pixels are stored in the residual image and transmitted to the client. High spatiotemporal (i.e., spatial and temporal) coherence results in relatively few occurrences of such pixels, so the model renderer is rarely invoked.

Running time of rendering one image, image[i], by purely using the model renderer can be estimated as in Eq. 1, as running time of rendering one image using current IBRAC is estimated as in Eq. 2.

$$R(image[i]) = \sum_{0 \le j < nxn} (f(M, j)) \qquad \text{(Eq. 1)}$$

$$R(image[i]) = \sum_{0 \le j < nxn} (a \cdot \sqrt{n} + b \cdot f(M, j)) \qquad \text{(Eq. 2)}$$

In Eq. 1 and 2, M represents the size of model data and n is the size of image (assume the image is n x n). *f(M)* means the cost of model renderer per pixel, which is the function of the model size. The first term ($a \cdot \sqrt{n}$) in Eq. 2 means the cost of image-based renderer per pixel because each pixel traverses at most, the number of pixels along the diagonal, $a \cdot \sqrt{n}$. The ratio, *b* is the number of pixels in the residual image over n^2 and represents the

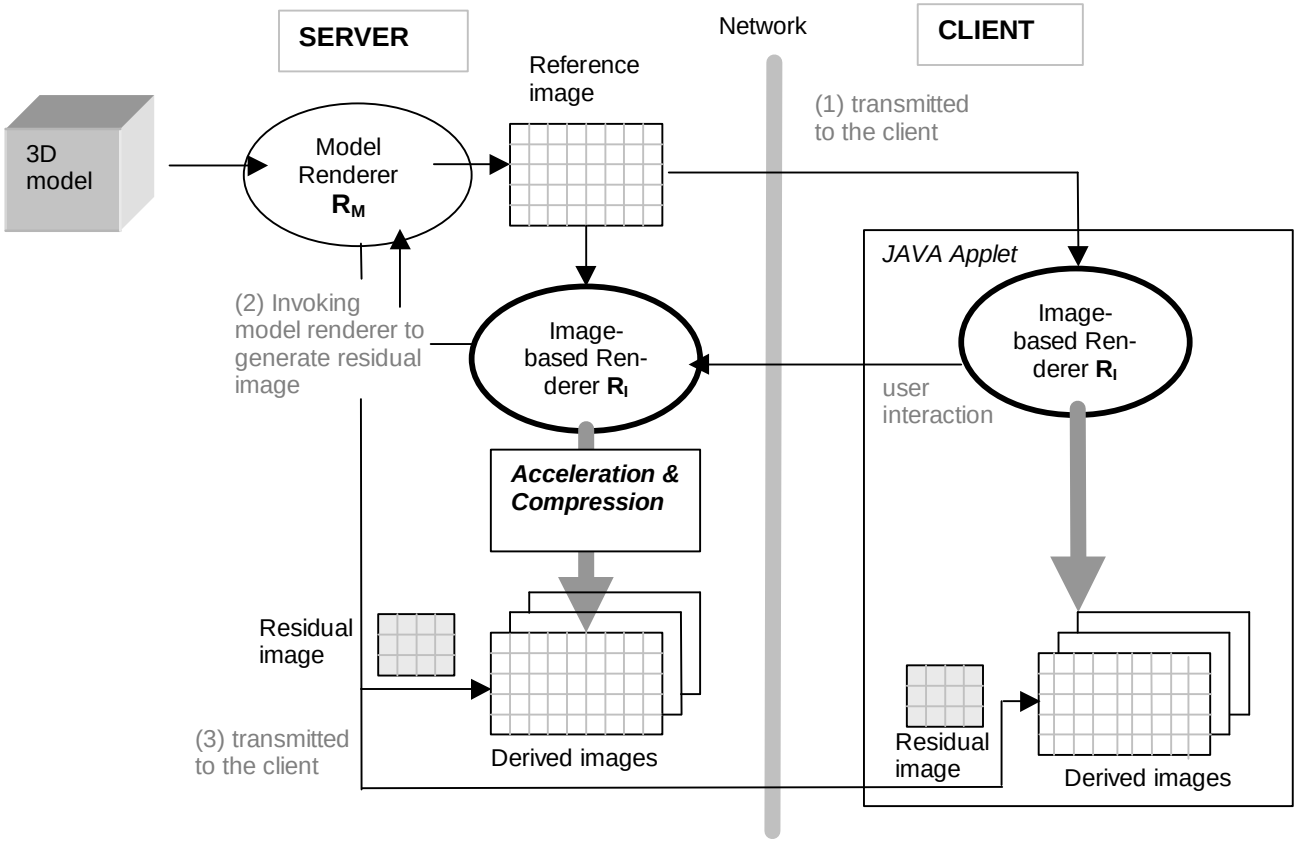


Fig. 2 – System overview of IBRAC : $\mathbf{R_I}$ (image-based renderer)and $\mathbf{R_M}$ (model renderer) are 2 sub components of IBRAC renderer. Note that $\mathbf{R_M}$ and 3D model reside only in server and $\mathbf{R_I}$ performs same task synchronously in both server and client for acceleration and compression; Model renderer generates reference image and transmits to the client. According to user interaction, Image-based renderer generates partially filled derived image using reference image and invokes model renderer to generate residual image. Residual image is transmitted to the client to complete the derived image.

frequency of invoking model renderer. It is the performance factor between two systems ($\mathbf{R_M}$ with/without $\mathbf{R_I}$). The smaller *b* is, the higher the compression and acceleration effects IBRAC achieves.

2.2. Rendering Acceleration

To extract reusable information, the image-based renderer back-projects new-image rays into previously rendered (reference) images. For each pixel p(index), a view ray AB is projected on to the reference image A'B' and then A'B' is searched for intersections (Fig.3).

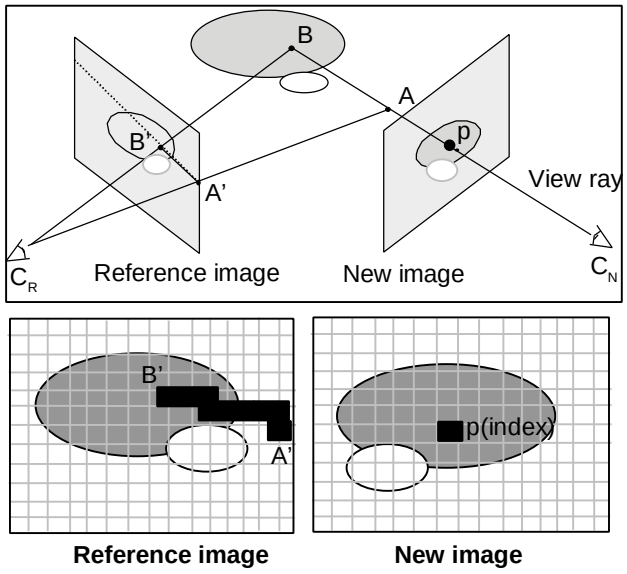


Fig. 3 – The image-based renderer casts a view ray from a pixel p, and the ray is projected to a line (A'B') in a reference image. The reference image line is traversed and tested for intersections with the sampled geometry.

Relation between AB and A'B' is as follows:

$\text{direction}(AB) = M_N^{-1} \bullet p(\text{index}) - C_N$
$p(\text{index}) = M_N \bullet A$
$p(\text{index}) = M_N \bullet B$
$B' = M_R \bullet B$
$A' = M_R \bullet A$
$\text{direction}(A'B') = \text{direction}(M_R \bullet B - M_R \bullet A)$
$= \text{direction}(M_R \bullet M_N^{-1} \bullet p(\text{index}) - M_R \bullet C_N$

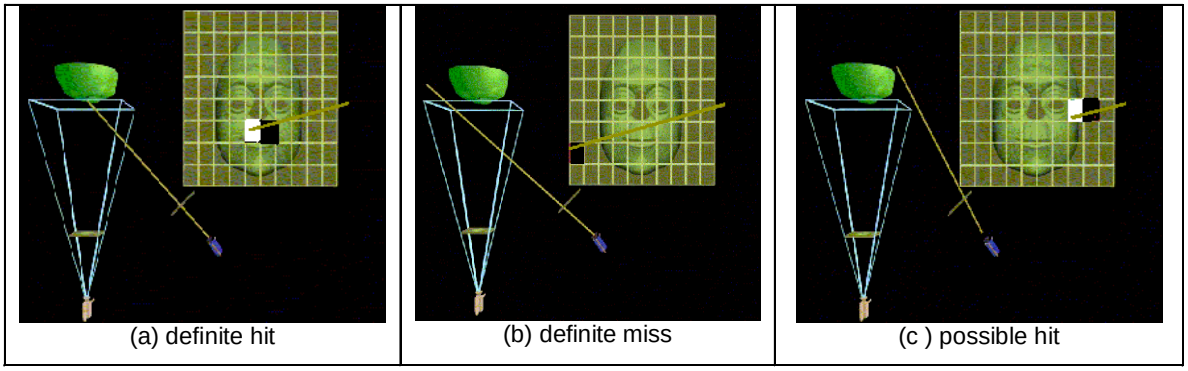
where Index (row + column x image_width) is the pixel position in new image, C_N is the new camera position, M_N is the projection matrix of the new image, and M_R is a projection matrix of the reference image.

Let's define depth values of the pixels in reference image as $\text{depth}_R(\text{index})$. Surfaces may be found by traversing the reference image starting from ($M_R \bullet C_N$) along the direction of A'B' by stepping one pixel amount (δx, δy, δz) at a time (Fig.3 Reference image). If there is no object between the new camera position and the reference image view frustum, image-based renderer starts traversing the ray within the reference image (A' in figure 3). Intersection tests compare the ray depth at every step ($z(A') + \delta z/(\delta x \text{ or } \delta y)$) with depth values at the reference image pixels ($\text{depth}_R(\text{index})$) traversed by the ray. A pixel depth value is a point sample in the center of the pixel. A ray depth is sampled at the closest point to the pixel sample position. All depth values are in the same perspective space. If the pixel depth value ($\text{depth}_R(\text{index})$) is less than the ray depth ($z(A') + \delta z/(\delta x \text{ or } \delta y)$), then we assert that the ray has passed behind a surface, causing an intersection.

```
Intersection Test (viewray AB)
    projected_viewray A'B' = projectToImage(AB);
    Pixel p = first(A'B');
    Index i = index(p);
    While (p.depth <  depthR(i) ) do
        prev_p = p
        p = next(A'B');
        // step 1 pixel in x or y direction and
        // add δz/(δx or δy)
        i = index(p);
        if(prev_p.depth - p.depth < threshold)
            return (definite hit, p);
        else if (outside_image(p) &&
                p.depth > max_depth_of_scene)
            return (definite miss, p);
        else return (possible hit, p);
```

Ray intersection tests in pixel-based scene representations are prone to uncertainty, so heuristics classify the results into three responses: *definite hit, definite miss, or possible hit* (Fig. 4). A definite hit is indicated (Fig. 4a) when a ray transitions from in-front-of to behind a surface. A definite hit returns the estimated 3D position of the intersection as well as a color estimate at that point. A definite miss (Fig. 4b) indicates that the ray missed all objects in the image and occurs for a ray that fails to pass behind any pixels. Definite misses and definite hits are high-confidence motion estimations.

A possible hit occurs 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. A possible hit also occurs when the depths of the two neighboring pixels in the reference image differ by more than a user specified error tolerance. In this case, the surface can be classified as not smooth or at a transition between objects. Possible hits are low-confidence motion estimations, therefore a possible hit returns the estimated 3D position of the intersection, allowing the model renderer to proceed from that point. High spatiotemporal (i.e., spatial and temporal) coherence results in relatively



(a) definite hit (b) definite miss (c) possible hit

Fig. 4 – Possible motion-prediction confidences during image-based rendering. The reference image is depicted with a grid and was previously rendered from the view point of the dim yellow camera shown in the lower left corner of (a), (b), and (c). Rays from a new position (shown as a blue camera) are traversed as lines in the reference frames. Rays pass in front of (pixels upto black pixels along the ray) and behind (white pixels). – Each situation where black pixel next to white pixel implies a possible intersection of ray and surface.

few occurrences of possible hits, so the model renderer is rarely invoked. As model size increases, image-based renderer is faster than model renderer, resulting acceleration of rendering process.

2.3. Compression

High compression ratios are achieved by reducing the transmission of redundant interframe (temporal) and intraframe (spatial) information. The reference image is the starting point in the IBRAC system, and the motion prediction encodes its approximate temporal evolution. The smoothness assumption in the intersection tests implicitly encodes spatial similarity. The low-confidence possible-hit case indicates where new information is needed. These pixels are computed by the model renderer (R_M) and stored into a residual image as an encoding process. This residual image replaces the need to transmit a complete new image because client also indicates where new information is needed. To achieve high compression ratios, the residual image should contain few pixels or low variations of pixel values. In IBRAC, the number of pixels in the residual image depends on the threshold of smoothness (i.e., user specified error tolerance) that determines whether surface depth variations warrant a possible-hit intersection test classification as explained in the previous section. This threshold provides a convenient control over the PSNR of the compressed images.

IBRAC computes pixel motion and the residual image in image-order. High confidence pixels have their color and depth interpolated from reference images. Pixels with low-confidence motion estimates are computed by the model renderer and included in the residual image. Since the IBRAC encoder and decoder use the same reference image (i.e., a precomputed frame or the prior frame), no motion estimates need to be transmitted. Also, since the sequence of residual pixels is known to both the encoder and decoder, there is no need to transmit address or index of the pixels in the residual image. These unique advantages of IBRAC dramatically raise the compression ratio.

Comparing IBRAC residual data with JPEG (~75% quality) or GIF encodings of the same residual images, shows an order of magnitude or more difference. IBRAC compression ratios improve over MPEG-2 by a factor of 2-10 in many cases.

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**" identifies a renderer and "**I**" denotes an image. The data flow is as illustrated in figure 3. Reference images or *I-frames* ($\mathbf{I_I}$) are generated from the model renderer (Eq. 3). These images contain color, and depth at each pixel. *Extracted Images* ($\mathbf{I_E}$) are created by the *Image-Based Renderer* ($\mathbf{R_I}$) from a given I-frame (Eq. 4). 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* ($\mathbf{I_E}$). *Residual Image* ($\mathbf{I_R}$) pixels are computed by the model renderer ($\mathbf{R_M}$) (Eq. 5). The *derived Image* ($\mathbf{I_D}$) combines an extracted image $\mathbf{I_I}$ and a residual image $\mathbf{I_R}$ (Eq. 6).

$$\mathbf{I_I}[j] = \mathbf{R_M}(Data_{3D}) \quad \text{(Eq. 3)}$$
$$\mathbf{I_E}[i] = \mathbf{R_I}(\mathbf{I_I}[j]) \quad \text{(Eq. 4)}$$
$$\mathbf{I_R}[i] = \mathbf{R_M}(Data_{3D}) \cap \mathbf{I_E} \quad \text{(Eq. 5)}$$
$$\mathbf{I_D}[i] = \mathbf{I_E}[i] \cup \mathbf{I_R}[i] \quad \text{(Eq. 6)}$$

The encoder (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 derived images ($\mathbf{I_D}$). The decoder (client) needs only the image-based renderer ($\mathbf{R_I}$) and one or more I-frames ($\mathbf{I_I}$), initially received from the

server. The decoder generates derived images ($\mathbf{I_D}$) 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 encoder and decoder are identical ($\mathbf{I_D}$[i] at encoder = $\mathbf{I_D}$[i] at decoder). The compression is lossless in this sense. Thus, derived images could be used as reference images for the next frame. However, the images generated by the model renderer are not identical to those generated by the total system ($\mathbf{I_D}$[i] != $\mathbf{I_I}$[i]), due to the errors introduced by sampled geometry and shading in the image-based renderer.

The reference and residual images at the encoder are compressed before transmission to the decoder. Residual images have no offset or index values and lossless compression is possible with an entropy coder like with gzip, an implementation of Lempel-Ziv encoding [Ziv77]. This compression further increases the compression ratio.

3. Results

Current implementation of IBRAC is that graphics server is composed of model renderer based on ray casting and image-based renderer in C++ on an SGI Onyx2 R10K 195 MHz CPU. Test data includes polygon data of varied complexity and volume data upto 64 MB, and is limited by the capacity of the model renderer in server, not by the image-based renderer. We generated animations by moving the camera within the scene. The test sequences exhibit rapid viewpoint changes over several different data types (Fig. 9*). Fig. 5 compares compression ratio of residual images (Fig.8*c, 8f, 8i, 8l) in different encodings (lossless JPEG, lossy JPEG at 75% quality, GIF and IBRAC) and complete images (lossless JPEG, lossy JPEG at 75% quality, and GIF). Use of the IBRAC encoding yields a compression ratio over ten times higher than the alternatives for the same residual images (strength of IBRAC format). It also compares compression of the residual image and complete images, showing how sparse the residual is (strength of IBRAC prediction capability). Therefore, transmitting a frame by IBRAC is 30 - 200 times better than simply transmitting a frame (as a complete image, not a residual) by lossless JPEG and 10 - 70 times better than lossy JPEG at 75% quality. This is indicative of the overall system performance.

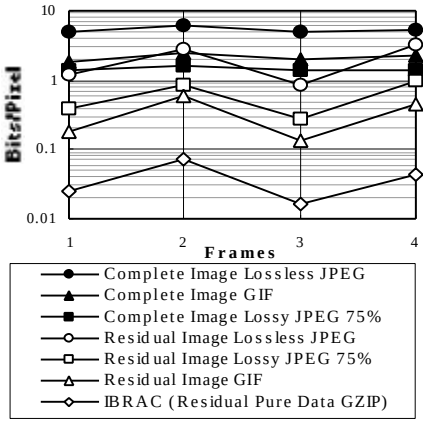


Fig. 5 - Compression ratio comparison of a residual image (lossless JPG/ lossy JPG at 75% quality, GIF, and IBRAC) and complete image (lossless JPG/ lossy JPG at 75% quality, GIF).

We also compared the image sequence shown in Fig. 9* with a lossy MPEG-2 encoding (using a Berkley encoder, version 1.2, at default quality). The animations were encoded using the data of Table 1a. The IBRAC method achieved a per frame compression ratio of 2 to more than 10 times better than the MPEG2 encoding (Fig. 6a). As shown in image sequences (Fig. 9*), 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 motions because of the knowledge of 3D positions. The residual images remain sparse as long as the camera remains trained in the vicinity of a reference image, regardless of zoom, rotation, or translation.

High compression leads to high acceleration since only residual pixels are rendered by the model renderer, and the image-based renderer is much faster and independent of data size.

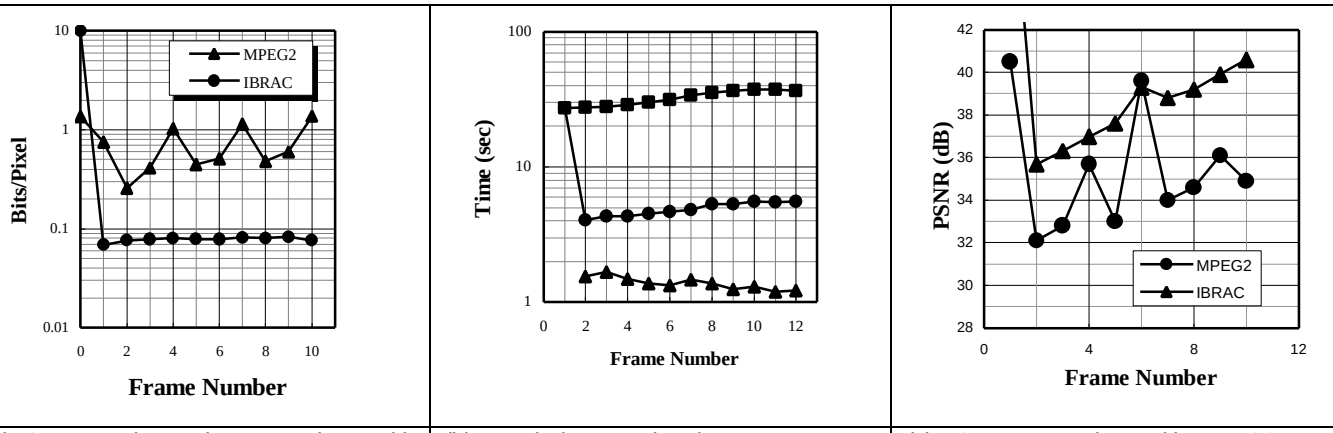
	Type	Size	Image size	Bits/pixel	PSNR
(a)	Volume	128x128x84x4	512x512	0.0256	41.78 dB
(b)	Volume	256x256x128x4	256x256	0.0707	31.8 dB
(c)	Polygons	897triangles	512x512	0.016	38.17 dB
(d)	Polygons	33264 triangles	400x400	0.0454	37.04 dB
(e)	Volume	64x64x64x4	256x256	0.077515	35.64 dB

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

Figure 6 illustrate IBRAC acceleration of rendering speed, compression ratio, and PSNR for test animation sequence using the model in Table 1e. In run time, the IBRAC method shows a speedup of three or more in encoding (server side) and thirty or more in decoding (client). Decoding (without transmission time) is faster since the residual is received rather than computed. As expected, the rendering time for the remote client side (decoding) is dependent on the image size, rather than the data set complexity. We tested the same volume data in three different sizes by interpolating additional points for a fair comparison. As shown in Fig. 7, the image-based renderer on the server side accelerates rendering speed by a factor of 3-9, reducing the dependency on data size, and the client image-based renderer shows execution time independent of data size, accelerating its rendering speed by a factor of 20-30.

As explained in Eq. 1 and 2, the smaller b is, the higher the compression and acceleration effects IBRAC achieves. Unfortunately most hybrid IBR techniques can not eliminate the $f(M)$ portion out of the overall running time, which still remains slight dependency on model size. But in overall performance, $(1-b) \bullet f(M)$ is much greater than $a \cdot \sqrt{n}$, resulting in acceleration. In table 2, we show performance of each item in range for animation sequences.

The image quality at the local renderer and the remote renderer are the same, as explained before. Compared to a stand-alone image-based renderer, our images are better since the hybrid system integrates model rendering to create the residual images that substantially improve image quality. However, IBRAC is a lossy compression method. The interpolated depth and color from reference images is different from what the model renderer would compute. Diffuse surfaces are more forgiving than specular ones. These problems are encountered with other image-based-rendering methods and discussed in [Chen93][McMillan95].

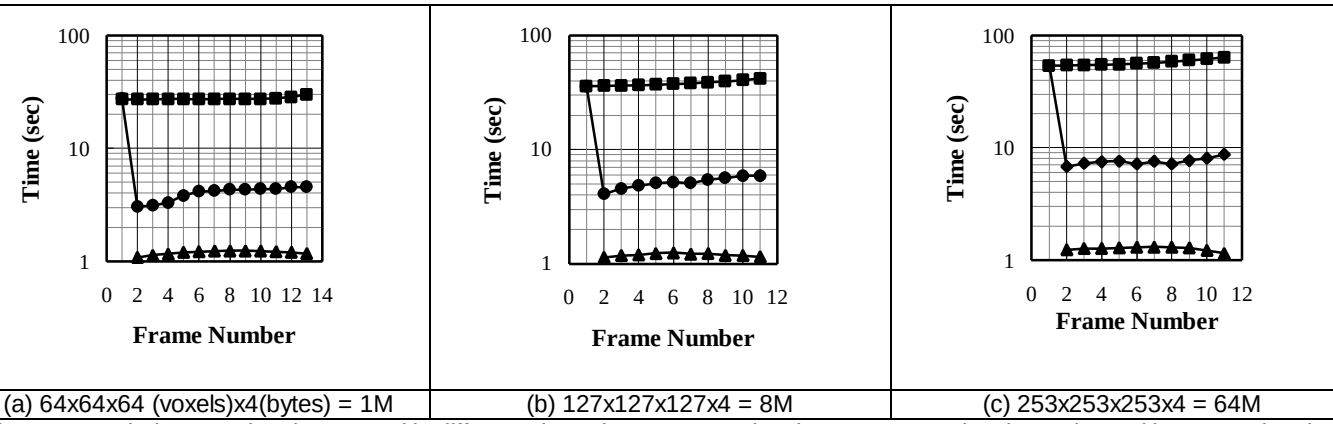


(a) Compression ratio comparison with MPEG2

(b) Rendering acceleration – Upper one is renderer without IBRAC, middle one is IBRAC server, lower one is client

(c) PSNR comparison with MPEG2

Fig. 6 – Measurement of IBRAC rendering time, compression and image quality using volume data rendering sequences.



(a) 64x64x64 (voxels)x4(bytes) = 1M

(b) 127x127x127x4 = 8M

(c) 253x253x253x4 = 64M

Fig. 7 - Rendering acceleration test with different data size. Rectangular dots represent the time taken without acceleration (using only model renderer). Circle dots represent time taken in the server image-based renderer, and triangular dots represent time taken in the client.

Unit : micro second	$a \cdot \sqrt{n}$ (cost of image-based renderer per pixel)	b (ratio of number of pixels in residual image over n^2)	$f(M)$ (the cost of model renderer per pixel)
Volume (1 Mbytes)	26 ~ 49	0.05 ~ 0.09	426 ~ 496
Volume (8 Mbytes)	28 ~ 55	0.05 ~ 0.09	576 ~ 667
Volume (64 Mbytes)	26 ~ 53	0.05 ~ 0.09	854 ~ 910
Polygon (33264 triangles)	20 ~ 42	0.11 ~ 0.2	1338 ~ 1495

Table 2 – Itemized performance

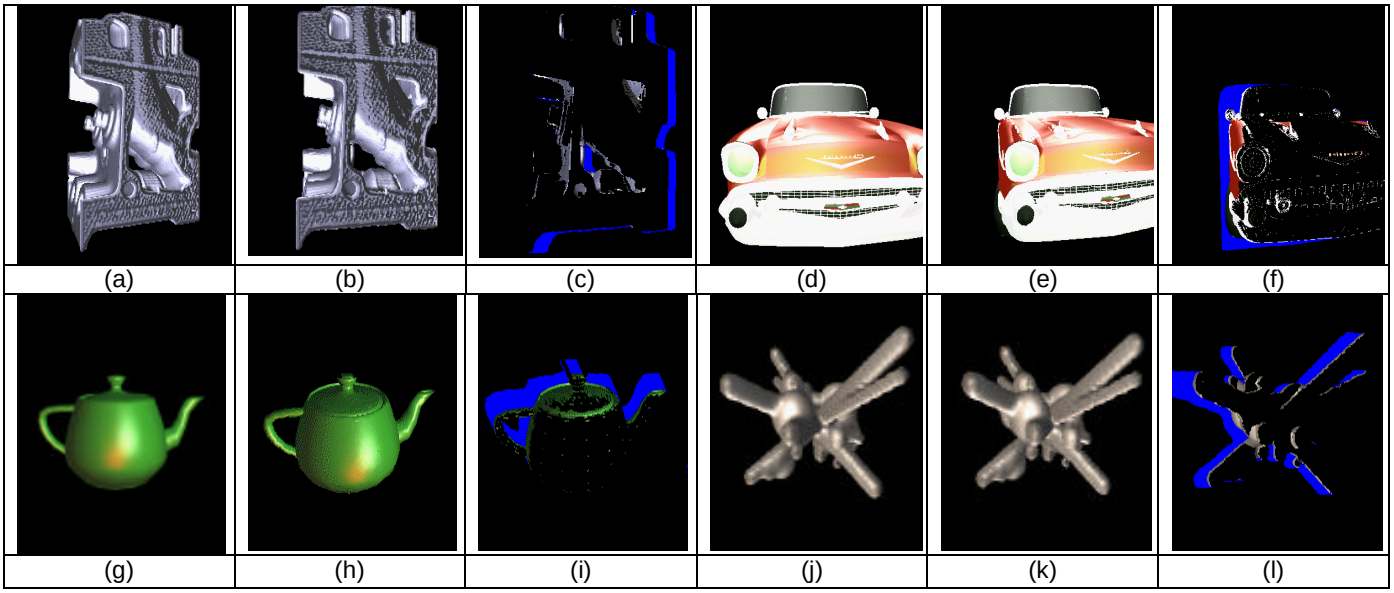


Fig. 8* - Images (a),(d),(g),(j) are used as reference images, and images (b),(e),(h),(k) are images generated from $\mathbf{R}_I$ (image-based renderer) and (c),(f),(l),(l) are residual images. The blue regions show pixels with low confidence motion estimates from image-based renderer that did not intersect objects or surfaces. The colored regions show low confidence pixels that were rendered by the model renderer.

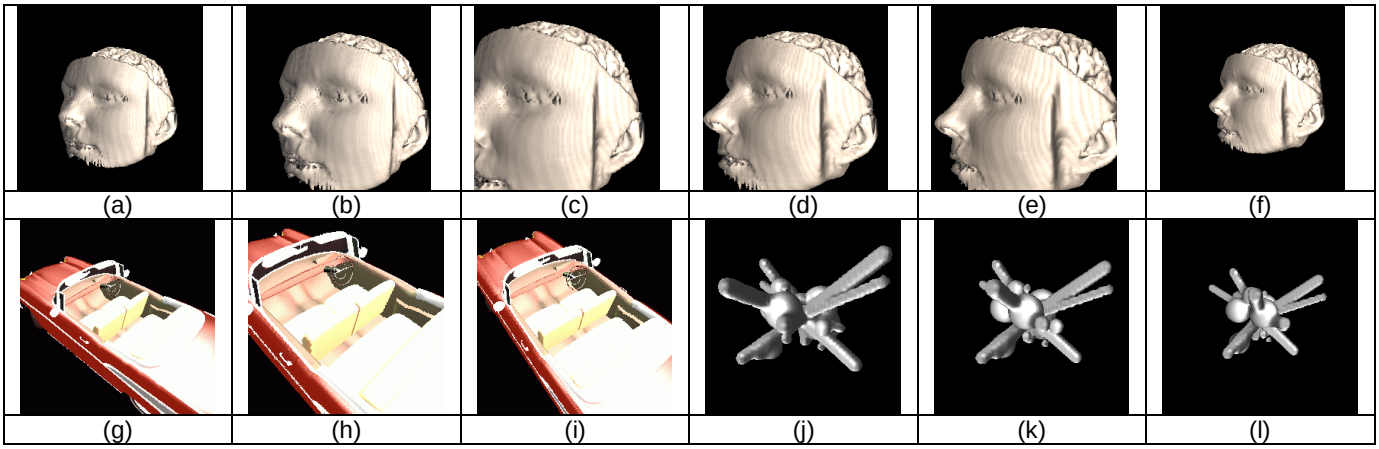


Fig. 9* - Images from the test animation sequences. Note the use of scaling and rotation. Mpeg animations made from the frames created by the IBRAC method are available at http://www-scf.usc.edu/~iyoon/html/research.html.

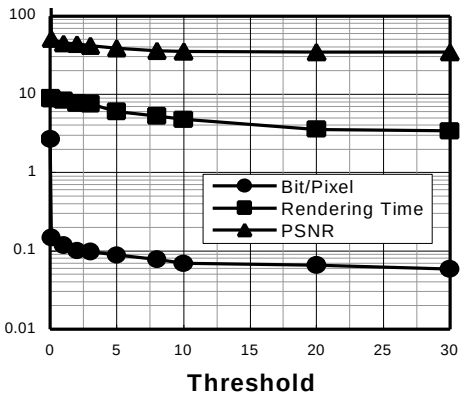


Fig. 10 - Threshold control of PSNR vs. compression (Bits/Pixel) & acceleration (seconds), using volume data described in Table1 (e).

However, we often find these artifacts to be subjectively minor, and objectively, the images exhibit a better PSNR (peak signal to noise ratio, calculated as 10 $\log_{10}(255^2/\text{MSE})$) than the MPEG-2 encodings in most frames. Figure 6c shows the PSNR for each frame of the animation sequences. Images in figure 9* and animations (found at http://www-scf.usc.edu/~iyoon/html/ research.html) show little visible degradation.

The initial reference image is compressed without loss, so it has infinite PSNR. The number of pixels in the residual images depends on the threshold of smoothness that determines whether surface depth variations warrant a possible-hit intersection-test classification as explained in the previous section. This threshold provides a convenient control over the PSNR of the compressed images as a trade off between image quality and compression ratio or acceleration (Fig. 10).

4. Discussion and future research

This paper presents a web-based rendering technique that utilizes and extends image-based rendering methods to compression. Using a reference image, 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 IBRAC method is attractive for web-based scientific visualization systems since it does not transmit model and is not limited by the format of volume data unlike VRML.

IBRAC Java applet (Fig. 11) is almost finished and model renderer in graphics server is being extended to handle larger data sets (up to several GB data sets)

The current implementation supports only a single reference image. The server may store more frames (reference images) than any one client since several clients can connect to the server at one time. The server may benefit from keeping several frequently accessed frames. These frames contribute to the enhancement of the compression

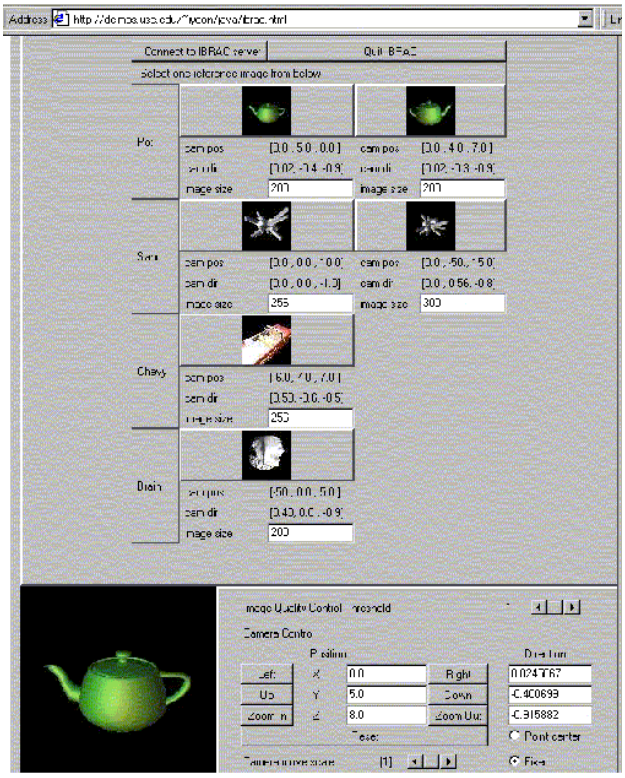


Fig. 11– IBRAC Java applet at "http://deimos.usc.edu/~iyoon/javaj/ibrac.html"

ratio and acceleration of the rendering in the server and client because multiple frames provide more information for motion prediction just as MPEG uses bi-directional prediction to enhance compression. Selection of a reference image from multiple frames can be done in a block basis. Selected pixels of each image block will be tested in low-resolution reference images. Future work will explore these topics and additional controls over quality and bandwidth tradeoffs.

5. Acknowledgements

This research has been funded by the Integrated Media Systems Center, a National Science Foundation Engineering Research Center, Cooperative Agreement No. EEC-9529152 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.

6. References

[Chen93] Chen, E., and Williams, L., "View Interpolation for Image Synthesis", Computer Graphics (Proc. SIGGRAPH 93, pp 279-288)

[Cohen-Or97] Cohen-Or, D., "Model-Based View Extrapolation for Interactive VR Web-Systems," Computer Graphics International, 1997.

[Djurcilov98] Djurcilov, S., Pang, A., "Visualization Products On-Demand Through the Web", VRML 98 Third Symposium on the VRML, pp 7 - 24

[Earnshaw97] The Internet in 3D Informarion, Images and Interaction / edited by Earnshaw, R. and Vince, J., Academic Press, 1997

[Levoy95] M. Levoy, "Polygon-Assisted JPEG and MPEG Compression of Synthetic Images", Computer Graphics, (Proc. SIGGRAPH 95 pp 21-28)

[Li98] J. Li, "Progressive Compression of 3D graphics," Ph.D Dissertaion, University of Southern California, May 1998

[Mann97] Mann, Y. and Cohen-Or, D., "Selective Pixel Transmission for Navigating in Remote Virtual Environments", Eurographics '97, Volume 16, Number 3.

[McMillan95] McMillan, L. and Bishop, G. ,"Plenoptic Modeling: An Image-Based Rendering System", Computer Graphics, (Proc. SIGGRAPH 95, pp 39-46)

[NAO] North Atlantic Ocean 3D datasets from LANL, www.acl.lanl.gov/climate/models/pop/simulations/atlantic/run_13/r13_offline/r13_off_frames.htm

[Trapp97] Trapp, J.C., and Pagendarm, " A prototype for a WWW-based Visualization Service", (Proc. Eurographics Workshop, Visualization in Scientific Computing '97, pp 21-30)
Available at http://www.sm.go.dlr.de/sm-sk_info/library/documents/EGSciVis97/VaWX5Fproto-1.html

[VHM] Visible Human Project web page http://www.nlm.nih.gov/research/visible/visible_human.

[VRML] ISO/IEC 14772-1, The Virtual Reality Modeling Language, http://www.vrml.org/VRML97/DIS/, 4 April 1997

[Ziv77] Ziv, J., Lempel, A., " A universal algorithm for sequential data compression," IEEE Transactions on Informaion Theory, IT-23:337-343, 1977