Compression of Lumigraph with Multiple Reference Frame (MRF) Prediction and Just-in-time Rendering

Cha Zhang[†*] and Jin Li[‡]

[†]Dept. of Electronic Engineering, Tsinghua University, Beijing 100084, China
[‡]Microsoft Research China, 49 Zhichun Road, Haidian, Beijing 100080, China

ABSTRACT

In the form of 2D image array, Lumigraph captures the complete appearance of an object or a scene, and is able to quickly render a novel view independent of the scene/object complexity. Since the data amount of Lumigraph is huge, the efficient storage and access of Lumigraph are essential. In this paper, we propose a multiple reference frame (MRF) structure to compress the Lumigraph data. By predicting the Lumigraph view from multiple neighbor views, a higher compression ratio is achieved. We also implement the key functionality of just-in-time (JIT) Lumigraph rendering, in which only a small portion of the compressed bitstream necessary for rendering the current view is accessed and decoded. JIT rendering eliminates the need to predecode the entire Lumigraph data set, thus greatly reduces the memory requirement of Lumigraph rendering. A decoder cache has been implemented to speed up rendering by reusing the decoded data. The trade off between the computational speed and cache size of the decoder is discussed in the paper.

KEYWORDS: image-based rendering (IBR), Lumigraph, multiple reference frame (MRF) structure, data compression, just-in-time (JIT) rendering

1. INTRODUCTION

There has been great interest in image-based rendering (IBR) systems recently. By recording the intensity of light rays passing through every space location and shooting at every possible direction, over any range of wavelengths and at any time, Adelson and Bergen [1] stated that the 7D plenoptic function could completely represent a 3D dynamic scene. By ignoring time and wavelength, McMillan and Bishop [2] defined image-based rendering (IBR) as generating a continuous plenoptic function from a set of discrete samples. The Lumigraph [3] and Lightfield [4] presented a clever 4D parameterization of the plenoptic function if the object (or conversely the camera view) can be constrained in a bounding box. We refer the technology as Lumigraph in the following discussion, though the object/scene of both Lumigraph and Lightfield fits the description. By placing the object in its bounding box which is surrounded by another larger box, the Lumigraph indexes all possible light rays entering and exiting one of the six parallel planes of the double bounding boxes. The Lumigraph data is thus composed of six 4D functions, where the plane of the inner box is indexed with coordinate *(u,v)* and that of the outer box with coordinate *(s,t)*. Usually, it is discretized more precisely for the inner bounding box closer to the object, and more coarsely for the outer bounding box. Alternatively, the Lumigraph can be considered as six two-dimensional image arrays, with all the light rays coming from

[*] The work was performed when Mr. Zhang was an intern at Microsoft Research China. The contact information of the authors are: Email: jinl@microsoft.com, Web: http://research.microsoft.com/users/jinl/index.htm, Phone: +86 10 62617711-5793.

a fixed *(s,t)* coordinate forming one image. This is equivalent to set a camera at each coordinate *(s,t)* and taking a picture of the object with the imaging plane be the *(u,v)* plane. An example of the Lumigraph/Lightfield image array is shown in Figure 1. Note that the neighboring Lumigraph images are very similar to one another. To create a new view of the object, we just split the view into its light rays, which are then calculated by interpolating existing nearby light rays in the image arrays. The novel view is then generated by reassembling the split rays together.

Lumigraph is attractive because it has information of all views of the object/scene. With Lumigraph, a scene can be rendered realistically yet fast compared with a top-notch graphic rendering algorithm such as ray tracing, and regardless of the scene complexity. However, the data amount of Lumigraph is huge. Using the Lumigraph scene of the fruit plate (Figure 1) as an example, there are 32 sample points in each axis on the *(s,t)* plane, 256 sample points in each axis on the *(u,v)* plane, 3 color samples per light ray, and 6 parallel image planes bounding the object. For such a relatively low resolution Lumigraph (the object resolution is that of the *(u,v)* plane, which is only 256x256), the total raw data amount is *32x32x256x256x3x6=1.125GB.* It is huge for storage in harddisk and distribution on CD, not to say browsing over the Internet. Practical Lumigraph applications may call for even higher sampling density, and therefore, result in even larger data amount.

The importance of compression has been realized through the birth of Lumigraph. However, Lumigraph bears some unique characteristics, which have lead to new challenges in compression. Since Lumigraph consists of array of images, its compression resembles that of video. However, there is difference. On the one hand, Lumigraph is a 2D image array, and there is more correlation in Lumigraph than in the 1D video sequences. On the other hand, the distortion tolerance of Lumigraph is small, as each view of Lumigraph is static, yet the human visual system (HVS) is much more sensitive to static distortions than time-variant distortions. Since a rendered view of Lumigraph is a combination of the image rays, certain HVS properties such as spatial and temporal masking may not be used. Most important, a compressed image bitstream is usually decompressed to get back the decoded image, a compressed video bitstream is played frame by frame, however, a compressed Lumigraph bitstream should not be decompressed and then rendered. In fact, the decompressed Lumigraph data is so large that most hardware today has difficulties to handle it. It is therefore essential to maintain the Lumigraph data in the compressed form, and decode only the contents needed to render the current view. We call such concept the just-in-time (JIT) rendering. JIT rendering is a key to design the Lumigraph compression algorithm, and thus, the Lumigraph decoder should be reasonably fast to accommodate the real-time decoding need.

To accommodate JIT decoding, most technologies used in IBR compression involves only intraframe coding. That is, the IBR data are segmented into blocks/chunks, and each block/chunk is compressed independently of each other. Levoy and Hanrahan [4] proposed a vector quantization (VQ) approach to compress the Lightfield. Sloan and Cohen [3] proposed to use JPEG (block DCT with run-level Huffman coding) to compress the Lumigraph. Both VQ and JPEG are fast in decoding, however, the compression performance of both approaches is limited. Image quality is acceptable at a low compression ratio of 25-50:1, and the quality of scene degraded quickly thereafter. Considering the huge data amount and high redundancy in the Lumigraph scene, better compression performance is expected. Kiu *et al.*[9] and Magnor[10] have proposed the use of an MPEG like algorithm to compress the Lightfield. Their algorithm has achieved a high efficiency, however, they did not address the problem of JIT rendering, which is crucial in the Lumigraph application.

In this paper, we propose the use of multiple reference frame (MRF) structure to compress the Lumigraph image array. A special contribution of the paper is that we achieve JIT rendering for a highly compressed Lumigraph scene with predictive coding. MRF bears strong resemblance to the video coding standard such as MPEG or H.26x. The image array in the Lumigraph scene is classified into two categories – the anchor frame (A) that is regularly distributed and independently encoded, and the predicted frame (P) that is referred to a nearby anchor frame through motion compensation and predictively encoded. Considering the 2D image array structure of the Lumigraph, the P frame in MRF may refer to any one of its four neighbor A frames. We restrict that the P frame refers only to an A frame, not another P frame so that the access of an arbitrary frame in the rendering stage can be reasonably fast. The compressed Lumigraph bitstream is attached with a two-level hierarchy index structure to enable random access to the compressed bitstream. We have also implemented a decoder cache to avoid the most recently used contents being decoded over and over again. The proposed MRF coder not only greatly improves the compression performance of the Lumigraph, but also spares the decoder from buffering an entire decoded Lumigraph. It also fits well for Lumigraph browsing over the Internet, as the codec has a high compression ratio and only decodes the necessary contents to render the current view.

For simplicity, we focus on the compression and rendering of one of the six image arrays of the Lumigraph in the following part of the paper. The technology can be easily extended to a full 3D view of the Lumigraph object. The multiple reference frame (MRF) compression algorithm is presented in section 2, as well as the two-level hierarchy index structure of the compressed bitstream. Section 3 discusses the JIT rendering implementation with the decoder cache. Simulation results and conclusions are presented in section 4 and 5, respectively.

2. THE MULTIPLE REFERENCE FRAME (MRF) COMPRESSION

The framework of the multiple reference frame (MRF) structure is shown in Figure 2. Let the Lumigraph be a 2D image array indexed by coordinate *(s,t)*, with pixels inside each image indexed by coordinate *(u,v)*. The 2D image array is shown in the left part of Figure 2, where each box represents one image. We select certain images as the reference or anchor frames (A frames). The rest images are called predicted frames (P frames), which refer to one of the anchor frames through motion compensation. The A frames are independently encoded, while only the prediction residue of the P frames is encoded. In the current implementation, the A frames are located on a regular grid in the *(s,t)* plane, as shown by boxes marked with symbol "*" in Figure 2.

Both anchor and predicted frame are segmented into square blocks and each block is encoded independently into a unit bitstream. In the current implementation, each square block is of size 16x16, which we call a macroblock (MB), for its similarity with the macroblock used in JPEG and MPEG. The size of the macroblock is a compromise of access granularity, the overhead spent on each macroblock (motion vectors and index information), and motion compensation efficiency. Each anchor frame is encoded independently, macroblock by macroblock. The macroblock is further split into six 8x8 subblocks, with four of which luminance subblocks, and the other two chrominance subblocks which have been subsampled by a factor of 2 in both horizontal and vertical direction. The subblocks are transformed by a basis-8 discrete cosine transform (DCT), quantized by an intra Q-table with a quantization scale Q_A, and then entropy encoded by a run-level Huffman coder. The whole procedure of macroblock coding is exactly the same as MPEG I frame coding [5] . Though not the best in terms of compression performance, the DCT and Huffman algorithm can be quickly inversed so that the macroblocks can be decoded in great speed. The quantization parameter Q_A determines the bitrate and the quality of anchor frame coding. The larger the value Q_A, the higher the compression ratio, however, the poorer the quality of reconstructed anchor frames. The quantization and Huffman tables used in MRF coding are exactly the same as those in MPEG2.

Four neighbor anchor frames are established for each predicted frame. Because the macroblock of each predicted frame may refer to multiple frames, we name the approach multiple reference frame (MRF) prediction. In Figure 2 for example, the predicted frame "☺" has four references which are the anchor frames with arrows pointing to the current one. A predicted frame only refers to an anchor frame, not another predicted frame. The predicted frame is also split into macroblocks, and each macroblock is encoded using motion compensation. For each macroblock, we search in an area around the current position of the macroblock in its four reference frames a best matching macroblock. A true best match should minimize the coding length and distortion of the residue error. However, since such a search is computational expensive, we use more simple criterion, i.e., the minimum mean square error (MSE) criterion that minimize the energy of the difference between the current macroblock and the matching one. A reference vector is transmitted for each macroblock of the predicted frame, indicating the position of the matching macroblock and its reference frame. After that, the difference between the current macroblock and its matching one, or the prediction residue, is encoded again through 8x8 DCT, an inter Q-table quantization with controlling parameter Q_P, and run-level Huffman coding. Because only the residue is encoded, typically it costs much few bits to encode a predicted frame than an anchor frame. The operation is similar to MPEG P frame coding, except that MPEG P frame has only one reference frame, and the reference frame may be either an I or a P frame. By enabling multiple frames as references, we improve the prediction efficiency with a price of two additional bits per macroblock for index. The overhead for using the multiple reference frames may be reduced if we encode the index of the reference frame, because the nearest anchor frame is more probable to be the best reference. By referring the predicted frames only to the anchor frames, we reduce the number of accessed frames to render an arbitrary Lumigraph view. Such easy data access is critical for just-in-time (JIT) rendering of the Lumigraph scene.

There are more advanced motion models, such as the affine motion model [6] and the perspective motion model [7] . A two-stage motion compensation method may be used [8] , which consists of global motion and local affine motion compensation. They achieve better motion compensation with more coding overhead for the motion parameters and much more complexity in obtaining the parameters. But in the current implementation, we prefer the simpler translation-based motion model due to the concern of the rendering speed.

The Lumigraph compression engine with MRF prediction is shown in Figure 3. Given a Lumigraph image array and the quality control parameters Q_A and Q_P, we first encode all anchor frames. After that, we decompress the anchor frames and use them as references for P frame coding, as the decoder can only access the compressed anchor frames, not the original ones. The predicted frames are then encoded, with each macroblock MRF predicted and its residue compressed. All the compression results are then fed into a bitstream, with a two-level hierarchical index table designed for easy random access. The first level of the index table resides in the head of the compressed Lumigraph bitstream, and records the encoded bitstream length of each individual frame, for both the A and P frame. A second level index table is stored within the bitstream of each compressed frame, and records the compressed bitstream length of each individual macroblock. With the two-level index tables, we may locate and access the compressed bitstream of any macroblock in any frame very quickly. The overhead added by the two-level index table is not trivial, especially at high compression ratio. The current implementation incurs a table overhead of 10% of the entire bitstream at a compression ratio 100:1, which increases to 30% when the compression ratio reaches 160:1.

3. JUST-IN-TIME RENDERING

We do not decode the entire set of Lumigraph scene any time during the rendering. Only the data necessary to render the current view are accessed and decoded right in time, interpolated and rendered on the screen. The concept is termed just-in-time (JIT) rendering. JIT rendering not only spares the huge memory required to buffer the entire Lumigraph scene, but also speeds up the Internet browsing of the Lumigraph as only the compressed data corresponding to the current view needs to be streamed over the Internet during browsing. The implementation of JIT is rather straightforward for compression algorithms that involve only local block access and fixed length coding, such as the spatial domain vector quantization (VQ) or the block truncation coding (BTC)[11] . However, the compression performance of such algorithms is limited. We implement JIT first time for a high compression ratio Lumigraph codec with frame prediction. To facilitate JIT rendering, we have built the two-level hierarchy index table inside the compressed bitstream so that each macroblock can be accessed independently. Moreover, cache has been established for both the anchor (A) and predicted (P) frame so that constantly accessed area needs not to be decoded again and again.

The JIT rendering flow is driven by the rendering engine, and runs as follows. When the Lumigraph viewer is launched, the two-level hierarchy index table is first decoded from the bitstream. When a new request is sent by the user to render a view of the Lumigraph, the rendering engine splits the view into multiple rays, where each ray passes through two parallel planes *(u,v)* and *(s,t)*, and the intersecting coordinate locates the ray in the Lumigraph data set. Since the coordinate may not be integral, the ray is calculated through a 4D bilinear interpolation in the *(u,v,s,t)* space with at most 16 rays. Up till now, the operation is exactly the same for any Lumigraph viewer. These 16 rays are accessed from the Lumigraph decoder. As shown in Figure 4, for each accessed ray *(u,v,s,t)*, its associated macroblock is located and checked if it has already been decoded and stored in the cache. For instance, if the ray belongs to an A frame, the anchor frame cache is checked, otherwise, the predicted frame cache is checked. If the macroblock is in cache, the intensity of the ray is returned to the rendering engine. Otherwise, the macroblock is decoded from the bitstream with the assistance of the two-level index table, put in the cache and accessed. The macroblock of the anchor frame is directly decoded from the compressed bitstream. However, to decode the macroblock of a predicted frame, its referred macroblocks in the anchor frame must be decoded first. There may be up to four referred macroblocks as the motion vector may not point to the start position of a macroblock. We check if the referred macroblocks are in cache, and if they are not, they are decoded from the compressed bitstream first. After all the referred anchor frame macroblocks are available, the prediction residue of the current macroblock is decoded and added to the motion compensated macroblock, and the resultant decoded P frame macroblock is stored in the P frame cache for later access.

In the current implementation, the combined size of anchor and predicted frame cache is around 800KB, which holds roughly 8 YUV images at resolution 256x256. This is only a fraction of the entire Lumigraph raw data set, which is 200MB. Besides, the cache size grows slower relative to the size of the Lumigraph data. If the resolution doubles, we can expect that the Lumigraph data size will increase by 16 folds, i.e., double in each of the *u,v,s* and *t* axis. However, the cache size only needs to quadruple, as it is more a factor of the image resolution at the *(u,v)* plane. A random replacement cache strategy is implemented for the management of the cache. Any time the anchor or the predicted frame cache is full and a new macroblock is to be decoded, we randomly drop one of the macroblocks in cache to leave room for the new one.

4. IMPLEMENTATION AND EXPERIMENTAL RESULTS

We have implemented a Lumigraph encoder with the multiple reference frame (MRF) structure and a just-in-time (JIT) Lumigraph decoder. A running scene of the JIT Lumigraph viewer is shown in Figure 5. The JIT Lumigraph viewer can run real-time on a desktop with Pentium II 300 CPU and 64MB RAM, without specific optimization. The test Lumigraph scene is a head rendered from the visible human project. As mentioned above, the data set is only one of the six parallel planes of the Lumigraph. The sampling resolution of the Lumigraph is 256x256 in the *(u,v)* plane and 32x32 in the *(s,t)* plane.

In the first experiment, we compare the compression efficiency of the multiple reference frame (MRF) algorithm with baseline JPEG. The rate control is turned off in MRF compression, i.e., the predicted frame quantization scale Q_P is set to be equal to twice the anchor frame quantization scale Q_A: $Q_P=2Q_A=2Q$, where scale Q controls the MRF compression ratio and the quality. We compare only the compression performance of JPEG and MRF, and do not count the overhead of the two-level index table. Note that if random access functionality is implemented in JPEG compressed Lumigraph, a two-level index table is needed too, and thus ignoring the table is a fair comparison with same functionality. The subsampling distance of the anchor frame is 4, thus 1/16th of the frames are anchor frames. Shown in Figure 6, the horizontal axis is the compression ratio, and the vertical axis is the average peak signal to noise ratio (PSNR), which is calculated as follows:

$$PSNR = 10 \cdot \log_{10} \frac{255^2}{mse}, \qquad \text{with} \qquad mse = E\left\| f(u,v,s,t) - \hat{f}(u,v,s,t) \right\|^2$$

where *mse* is the mean square error, and $E(x)$ is the average operation. f and $\hat{f}$ are the original and decoded Lumigraph data set, respectively. A curve towards upper right corner of the figure indicates a larger PSNR at the same compression ratio, and thus superior compression performance. It is obvious from the figure that the MRF-based Lumigraph compression is superior to the algorithm with only intra frame coding. The compression ratio of MRF nearly doubles that of the JPEG compression, especially at low bit rate.

The optimal distance between anchor frames is investigated in Figure 7. The test Lumigraph scene is compressed with sampling distance 2, 4, 6 and 8, with all the other compression control parameters the same. The curves of compression ratio versus the PSNR are shown in Figure 7. It is observed that for the visible human head, a sampling distance of 2 is optimal for compression ratio below 80:1, and a sampling distance of 4 is optimal for compression ratio beyond 80:1. Since MRF will be used mostly for compression ratio above 80:1, a distance of 4 is selected for the other experiments.

In Figure 8, we compare the MRF Lumigraph compression with the single reference frame (SRF) approach, which is almost the same as MRF except that each macroblock of a predicted frame refers to only one of the closest anchor frame. In case there are more than one closest anchor frames with equal distance, the frame toward upper-left corner will be selected. MRF outperforms SRF for around 0.5dB in PSNR at the same compression ratio, or 5.6% in compression ratio at the same PSNR. Therefore, the multiple reference frame structure and the 2 additional bits used as reference (not entropy coded yet) are justified in Lumigraph compression.

The size of the macroblock cache versus the decoding speed is investigated in Figure 9. The horizontal axis is the size of the cache, in terms of the number of macroblocks. The axis is numbered with a base-2 logarithmic coordinate. Therefore, coordinate 5 stands for $2^5=32$ macroblocks. The vertical axis is the average number of macroblocks newly decoded while rendering a view. We use a designed sequence of views for this experiment. Two curves are drawn in Figure 9. The solid curve corresponds to a cache design with a larger anchor frame cache, and the dashed curve corresponds to a design with a larger predicted frame cache. The ratios between anchor and predicted cache for the two cases are 2:1 and 1:2, respectively. It shows that the number of average decoded macroblocks decreases steeply as the total cache size increases from 32 (coordinate 5) to 256 (coordinate 8) macroblocks, but the decrease slows down for cache size beyond 256 macroblocks. It is observed that with the same cache size, more predicted frame cache leads to less decoded macroblocks per view. Moreover, with a cache larger than 512 macroblocks, the rendering speed difference between more anchor and more predicted frame cache becomes insensible.

For global optimization, a cache size of 1024 macroblocks is used in our implementation, which occupies a memory of only 0.8MB.

5. CONCLUSIONS AND FUTURE WORKS

In this paper, a MRF Lumigraph compression scheme is proposed. The algorithm significantly outperforms the intraframe Lumigraph compression schemes such as VQ or JPEG, yet still provides real time rendering that is not supported in a video-like coder. It outperforms JPEG Lumigraph compression as much as two times. A two-level index table is inserted into MRF compressed bitstream so that the Lumigraph may be rendered just-in-time (JIT), with the contents needed to render the current view accessed and decoded in real time. A JIT Lumigraph viewer can run smoothly on a desktop PC with Pentium II 300 CPU and 64MB RAM. With better tuning, it is possible to run the Lumigraph viewer at an even lower speed.

The current work may be improved in a number of ways. Lumigraph consists of image frames formed through regular motion of camera. By modeling the camera motion, we can achieve a more accurate prediction of neighbor images. We may also use multi-resolution scheme, such as DWT (discrete wavelet transform) for the residue transform. The entropy coding may also be made better through the use of bitplane coding.

ACKNOLEDGEMENTS

The authors would like to thank Michael Cohen and Peter-Pike Sloan for the raw Lumigraph data and the source codes of a Lumigraph viewer, and Honghui Sun for the software support. We would also like to thank Ya-Qin Zhang and Shipeng Li for their helpful discussions, and Ye Lu and Guiwei Xing for their help in proofreading the paper.

REFERENCES

[1] E. H. Adelson, and J. R. Bergen, “The plenoptic function and the elements of early vision”, Computational Models of Visual Processing, Chapter 1, Edited by Michael Landy and J. Anthony Movshon. *The MIT Press*, Cambridge, Mass. 1991.

[2] L. McMillan and G. Bishop, “Plenoptic modeling: an image-based rendering system”, *Computer Graphics (SIGGRAPH’ 95)*, pp. 39-46, Aug. 1995.

[3] S. J. Gortler, R. Grzeszczuk, R. Szeliski and M. F. Cohen, “The Lumigraph”, *Computer Graphics (SIGGRAPH’ 96)*, pp. 43, Aug.1996.

[4] M. Levoy and P. Hanrahan, “Light field rendering”, *Computer Graphics (SIGGRAPH’ 96)*, pp. 31, Aug. 1996.

[5] J. L. Mitchell, W. B. Pennebaker, C. E. Fogg, and D. J. LeGall, “MPEG video: compression standard”, Chapman & Hall, 1996.

[6] H. Sanson, “Motion affine models identification and application to television image coding”, *SPIE symp. Visual Communications and Image Processing 91*, Vol. 1605, pp. 570-581, 1991.

[7] R. Y. Tsai and T. S. Huang, “Estimation three-dimensional motion parameters of a rigid planar patch”, *IEEE Trans. Acoust., Speech, Signal Processing*, Vol. ASSP-29, pp. 1157-1162, Dec. 1981.

[8] H. Jozawa, K. Kamikura, A. Sagata, H. Kotera, and H. Watanabe, “Two-stage motion compensation using adaptive global MC and local affine MC”, *IEEE Trans. on Circuits and Systems for Video Technology*, Vol. 7, No. 1, Feb. 1997.

[9] M. Kiu, X. Du, R. Moorhead, D. Banks and R. Machiraju, “Two-dimentional sequence compression using MPEG”, *in Visual communication and image processing (VCIP’ 98)*, pp. 914-921, Jan. 1998.

[10] M. Magnor and B. Girod, "Adaptive Block-Based Light Field Coding," *Proc. 3rd International Workshop on Synthetic and Natural Hybrid Coding and Three-Dimensional Imaging IWSNHC3DI'99*, Santorini, Greece, pp. 140-143, Sep. 1999.
[11] Edward J Delp & O Robert Mitchell, "Image compression using block truncation coding", *IEEE trans. Communication*, Vol. COM-27, No.9, pp.1335-1342, Sep. 1979.

FIGURES

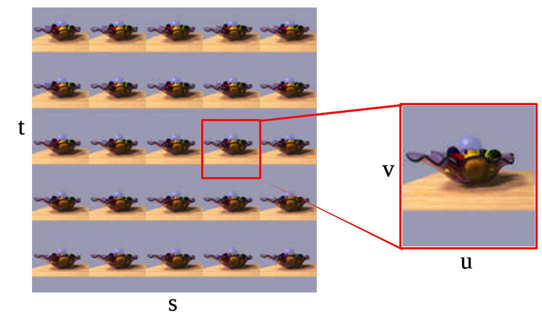

Figure 1 A sample Lumigraph image array: fruit plate

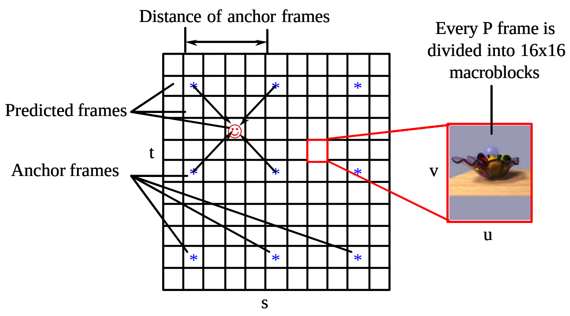
Distance of anchor frames
Every P frame is divided into 16x16 macroblocks
Predicted frames
Anchor frames
t
v
u
s

Figure 2 Anchor frame and predicted frame

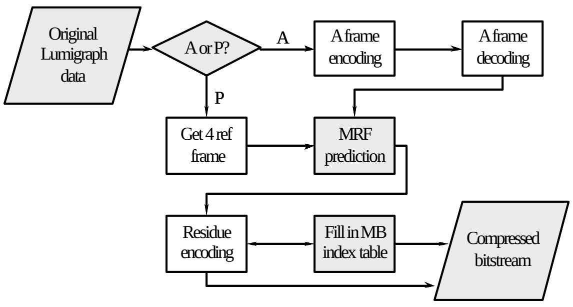

Figure 3 Framework of Lumigraph encoder with MRF prediction

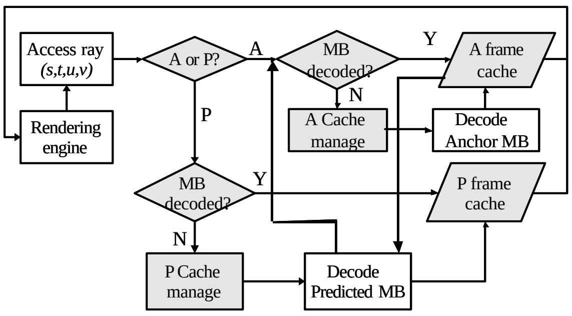

Figure 4 Just-in-time rendering flow

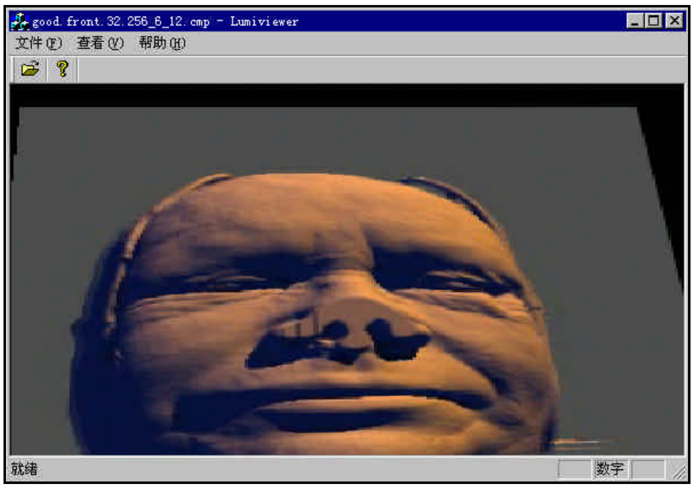

Figure 5 Running scene of the just-in-time Lumigraph viewer

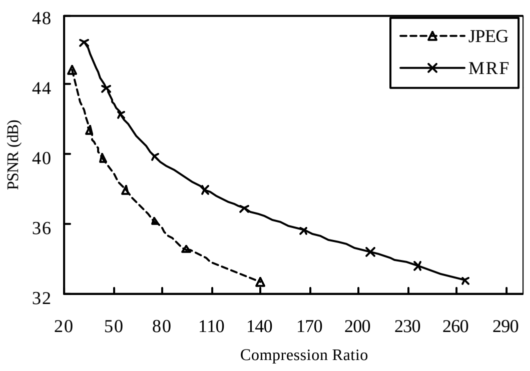

Figure 6 Compression performance of JPEG versus MRF

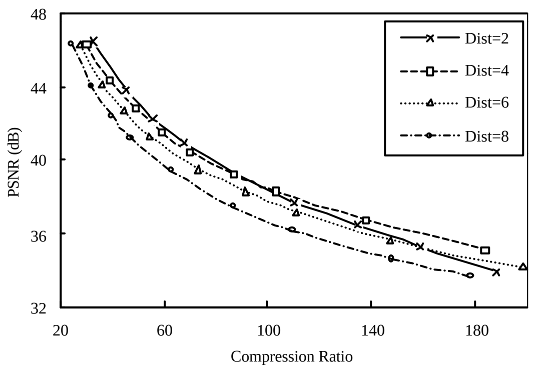

Figure 7 Compression performance with different anchor frame distance

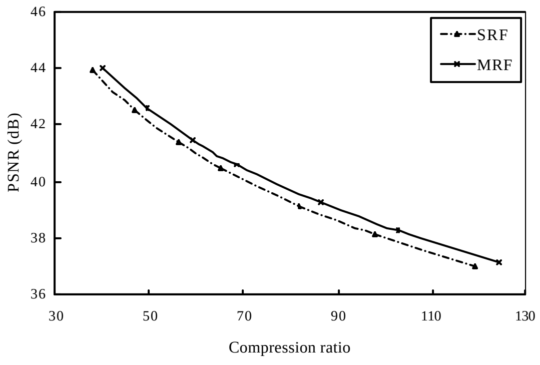

Figure 8 Compression performance of MRF versus SRF scheme

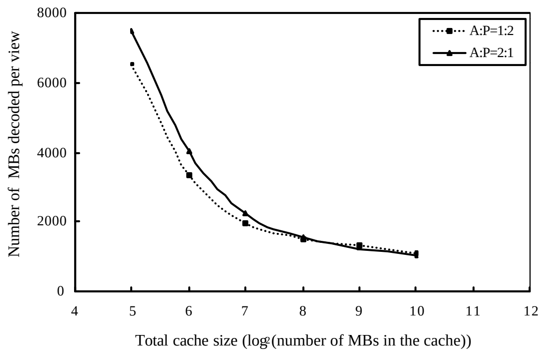

Figure 9 Total cache size versus the number of MBs decoded per view