Compression and Rendering of Concentric Mosaics with Reference Block Codec (RBC)

Cha Zhang[†*], Jin Li[‡]

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

ABSTRACT

Concentric mosaics have the ability to quickly capture a complete 3D view of a realistic environment and to enable a user to wander freely in the environment. However, the data amount of the concentric mosaics is huge. In this paper, we propose an algorithm to compress the concentric mosaic image array through motion compensation and residue coding, which we called reference block codec (RBC). A two-level index table is embedded in the compressed bitstream for random access. During the rendering, the entire compressed concentric mosaic scene is not fully expanded at any time. In stead, only the contents necessary to render the current view are decoded in real time. We denote such rendering scheme as just-in-time (JIT) rendering. Four decoder caches are implemented to speed up the rendering.

Keywords: Image-based rendering (IBR), concentric mosaics, reference block codec (RBC), data compression, just-in-time (JIT) rendering, cache management.

1. INTRODUCTION

Image-based rendering (IBR) has attracted much attention recently in realistic scene/object representation. Proposed by Adelson and Bergen [1] as a 7D plenoptic function, IBR models a 3D dynamic scene by recording the light rays at every space location, towards every possible direction, over any range of wavelengths and at any time. By ignoring time and wavelength, McMillan and Bishop [2] defined plenoptic modeling as generating a continuous 5D plenoptic function from a set of discrete samples. The proposals of Lumigraph [3] and Lightfield [4] made IBR more popular, as they provided a clever 4D parameterization of the plenoptic scene under the constraint that the object or the viewer could be constrained within a 3D bounding box. Even though most IBR scenes are synthetic, it is possible to capture Lumigraph/Lightfield of a realistic scene/object. However, there remain technical challenges, e.g., careful motion control of the camera array so that pictures can be taken from regular grid points parallel to the image plane. Shum and He [5] proposed concentric mosaics, a 3D plenoptic function restricting viewer movement inside a planar circle and looking outside. A concentric mosaic scene can be constructed very easily by rotating a single camera at the end of a round-swinging beam, with the camera pointing outward and shooting images as the beam rotates. At the time of rendering, we just split the view into vertical ray slits, and reconstruct each slit through similar slits captured during the rotation of the camera.

Compared with a top-notch graphic rendering algorithm such as ray tracing, the concentric mosaics can render a scene realistically yet fast, and regardless of the scene complexity. However, the data amount of the concentric mosaics is huge. As an example, the *Lobby* scene (shown in Figure 7) consists of *1351* frames with resolution *320x240* and occupies a total of *297* mega bytes. Compression is essential through the birth of concentric mosaics. High compression ratio is also achievable because of the strong correlation within frame and across frames of the concentric mosaics. Moreover, there are novel characteristics of the concentric mosaics that need special attention in the design of the compression algorithm. The rendering of the concentric mosaics never

---

[*] The work was performed when Mr. Zhang was an intern at Microsoft Research China.
Correspondence: Email: cha_zhang@hotmail.com; jinl@microsoft.com. Telephone: (86-10) 6261-7711 Ext. 5793.
Fax: (86-10) 6255-5337.

requires the entire dataset. In fact, each time a novel view is rendered, only a small portion of the dataset is used by the rendering engine. To save the system cost and reduce the rendering memory, a well-designed concentric mosaic codec should allow portion of the dataset to be randomly accessed and decoded from the compressed bitstream. We term such functionality as just-in-time (JIT) rendering or JIT decoding, i.e., to decode only the content needed for the rendering of the current view. The JIT decoding should also be reasonably fast to power the rendering engine.

A spatial domain vector quantizer (SVQ) was used in [5] to compress the concentric mosaic scene. The SVQ compressed bitstream can be fast decoded through a simple table lookup operation, and each SVQ lookup table index is of equal length, which makes the bitstream required for the current rendered view easy to access. However, SVQ is complex and time-consuming at the encoding stage, and the compression ratio of SVQ is limited. For example, SVQ in [5] achieves a compression ratio of 12:1, which leaves a compressed *Lobby* scene at 25 mega bytes. We may choose to compress each individual shot of the concentric mosaics with baseline JPEG or JPEG 2000. This may not be the most efficient way because correlation between multiple shots is not utilized. Moreover, during the rendering of new views, the concentric mosaic data set is accessed by slits rather than images. Therefore, a bitstream formed by concatenation of individually compressed images may not be easy to access. Video-based codec such as MPEG [6] is another choice for the compression of concentric mosaics. MPEG achieves a very high compression ratio by exploring the redundancy in neighboring frames. However, MPEG decoder is designed to access the compressed image sequentially and does not support random access. It thus may not be practical for use in the concentric mosaic environment. A 3D wavelet approach has also been proposed for the compression of the concentric mosaics [8] . The 3D wavelet algorithm achieves a good compression ratio, and may also access a portion of the compressed bitstream for a rendered scene with different resolution and quality through a progressive way. Such resolution and quality scalability is very useful in the Internet environment. However, scalable 3D wavelet decompression is highly complex and may not be feasible in the near future.

Even though MPEG and 3D wavelet codec can both achieve good compression efficiency, substantial computation resources must be devoted to decode the concentric mosaic data in real time. Alternatively, we may predecode the entire bitstream and render on the decoded data set. This is not feasible because it not only introduces a long delay at the beginning, but also requires a huge memory to hold the entire decoded environment. The challenge is thus to develop a codec which provides good compression performance, yet at the same time enables the view of the environment to be accessed and rendered in real time, with minimum decoder cache support. That is the essence of just-in-time (JIT) decoding.

In this work, we compress the concentric mosaics with a new scheme called reference block codec (RBC). For the first time, we realize the JIT rendering for a highly compressed concentric mosaic scene with predictive coding across frames. RBC splits the concentric mosaic images into blocks and predictively encodes them. Though RBC bears strong resemblance to MPEG, the frame structure of RBC is redesigned to improve the compression efficiency and to enable the bitstream to be easily random accessed. The shots or frames of the concentric mosaics are classified into two categories – the anchor (A) frame and the predicted (P) frame. The anchor frames are encoded independently, while the predicted frames are referred to an anchor frame by motion compensation and predictively encoded. We restrict any predicted frame to only refer to an anchor frame, not another predicted frame. A two-stage motion description is employed in RBC, so that the motion of each macroblock is a combination of the frame translation motion and the local refinement motion. Moreover, a built-in two level index table is included in the compressed bitstream for easy access at the rendering stage. Four elaborate caches are designed with special queue structure to avoid decoding contents over and over again. The access and decoding speed of RBC is fast enough to power real-time rendering of concentric mosaics.

The paper is organized as follows. The concentric mosaics and its rendering scheme are introduced in Section 2. Section 3 explains the details of the RBC codec, including the two-stage motion estimation, the frame structure

and the bitstream syntax. In Section 4, we discuss the JIT rendering implementation of RBC with the decoder cache. Simulation results and conclusions are presented in Section 5 and 6, respectively.

2. THE CONCENTRIC MOSAIC SCENE

Figure 1: Capturing device of concentric mosaics.

A concentric mosaic scene is captured by mounting a camera at the end of a rotating beam, and shooting images at regular intervals as the beam rotates, as demonstrated in Figure 1. The dataset of concentric mosaics is a shot sequence, which can be denoted as *c(n,w,h)*, where *n* indexes the camera shot, *w* and *h* represent the horizontal and vertical position of each ray within a shot, respectively. Let *N* be the total number of shots during the *360* degrees rotation, *W* and *H* be the width and the height of each shot. *N* is often set between *900* and *1500*, which means that the camera shots are very dense, typically *2.5* to *4* shots per degree. In [5] , the shot sequence is rearranged into concentric mosaic images, where each image is the combination of vertical slits with the same *w*. However, such a rearrangement is not necessary for either the compression or the rendering of concentric mosaics. In the proposed reference block codec (RBC), we use the original shot sequence for both compression and accessing.

Rendering concentric mosaics involves reassembling slits from existing shots. Shown in Figure 2, let *R* be the length of the beam, q_{FOV} be half of the horizontal field of view (FOV) of the camera, a concentric mosaic scene can render an arbitrary view with the same FOV within an inner circle of radius $r = R\sin q_{FOV}$. Let *P* be a novel viewpoint and *AB* be the field of view to be rendered. We split the view into multiple slits, and render each of them independently. For example, when the slit *PV* is rendered, we simply search for the slit *P'V* in the captured dataset, where *P'* is the intersection between ray *PV* and the camera path, as they are just the same because the intensity of the ray does not change along a line. Since the concentric mosaics consist of a discrete number of shots, and each shot consists of a discrete number of vertical slits, there may not be an exact slit for *P'V* in the dataset. The four sampled slits closest to *P'V* may be P_1V_1, P_1V_2, P_2V_3 and P_2V_4, where P_1 and P_2 are the two nearest captured shots on the two sides of the intersection point *P'* along the camera path, P_1V_1 and P_1V_2 are slits just beside P_1V in shot P_1, and P_2V_3 and P_2V_4 are slits beside P_2V in shot P_2. We may bilinearly interpolate the four slits to get the content of *P'V* (denoted as bilinear interpolation mode), or, if due to complexity concern, we may use the one slit closest to *P'V* to represent it (denoted as point sampling mode). It is obvious that the bilinear interpolation mode will result in a better rendering quality of the scene, with the price of a slower rendering speed. In either case, the content of the slit *P'V* is recovered, which is then used to render slit *PV* in the rendered view. The environmental depth information may be helpful to find the best approximating slits and alleviate the vertical distortion. More detailed description of concentric mosaics rendering may be found in [5] .

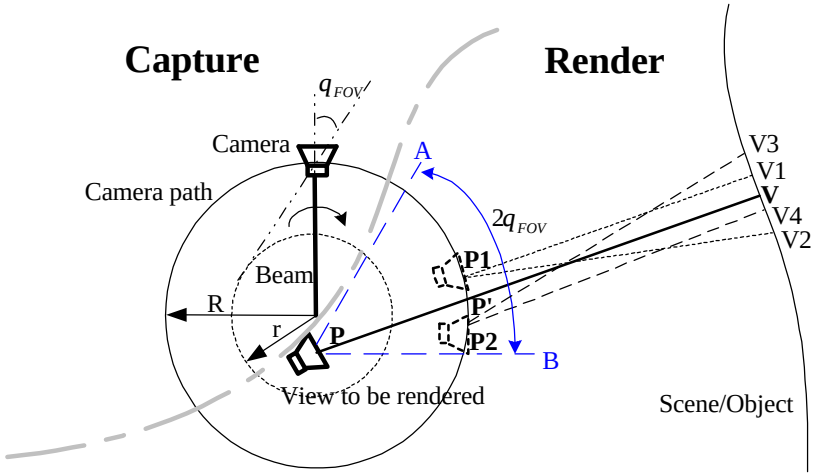

Figure 2: Concentric mosaics capturing and rendering.

It is observed in Figure 2 that when we are rendering a novel view from a camera at any position, only the captured shots within the horizontal field of view of the camera is involved. Moreover, even for the shots within the field of view, only a small portion of the vertical slits needs to be known. The number of the accessed vertical slits is inverse proportional to the distance between the viewpoint and the capture path. Therefore, it is essential to provide random access to a cluster of vertical slits. In the proposed RBC, 16 vertical slits are grouped and accessed as an elementary unit.

3. REFERENCE BLOCK CODING

The data structure of the reference block codec (RBC) is shown in Figure 3. The concentric mosaic shots are divided into two categories, the anchor (A) and the predicted (P) frames. The A frames are distributed uniformly across the concentric mosaic dataset and provide the anchor of access for the decoding operation. The A frames are encoded independently, while the P frames are predictively encoded with reference to the two nearby A frames. Let *D* be the ratio of P frames to A frames. The smaller the value of D is, the more the A frames are, and the easier the random access becomes. However, the compression efficiency will suffer, as the correlation between neighbor shots is not fully utilized. In the current implementation, *D* is set to be *7*, i.e., one out of eight frames is an A frame.

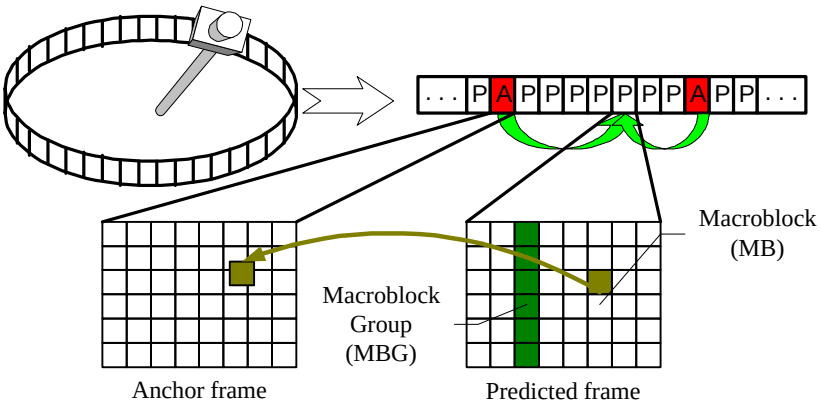

Figure 3: The data structure of the reference block codec.

Both A and P frame are segmented into square blocks of size *16x16*. We call it a macroblock (MB), for its similarity with the macroblock used in JPEG and MPEG. All MBs at the same vertical position of a shot are

grouped together and form a macroblock group (MBG), which is the smallest unit of accessing and decoding. The width of the MBG is a compromise between the access granularity and the coding efficiency. Without consideration of the coding efficiency, an access granularity of 1 slit is the optimum choice. However, it is shown [7] that image correlates over 20 neighbor pixels, thus coding the image slit by slit is not efficient. In this work, we cluster 16 vertical slits to form an MBG that is accessed and decoded together. Though redundant slits are accessed for an individual view, the rendering speed is satisfiable in interactive wandering due to the use of cache.

The MBs in A frame are independently encoded. Each MB is split into six *8x8* subblocks, with four of which luminance subblocks, and the other two chrominance subblocks which are sub-sampled by a factor of *2* in both the 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 with an A frame Huffman table. The compressed bit streams of all MBs belong to the same A frame MBG are then grouped together.

MBs in the P frames are predictively encoded with reference to a nearby A frame. The P frame may refer to two nearby A frames, however, for a single MB in the P frame, it only refers to one of the two. In fact, we restrict all MBs in a single MBG to refer to the same A frame. This restriction reduces the amount of accessed data when a slit in the P frame MBG is accessed. Since the concentric mosaic frames are shot by swinging a single camera mounting on a beam, the motion between two concentric mosaic images is predominantly horizontal translation, with little to none vertical motions. A two-stage motion estimation, including a global translation motion and a local refinement motion, is thus applied to calculate the motion vector of each MB. The camera motion between shots is modeled by a global horizontal translation motion. We do not use more complex model such as affine or perspective model, because the camera only moves a very small interval between shots with dominant translation motion, and more complex motion model is not justified. The dominant global horizontal translation vectors mv_1 and mv_2 of the P frame with regard to the two referring A frames are calculated and recorded. The vector *mv* will be used to reduce the search range and the entropy of the P frame MB motion vector. The individual refinement motion vector of the P frame MB is restricted to *+/-5* pixels of the global translation vector *mv* with half pixel accuracy, because we know most MBs just move along the underlying P frame. In fact, around half of the local refinement motion vectors of the P frame MBs are zeros. To encode a P frame MBG, we encode the MBG against both reference A frames. The MBG is split into a number of MBs. For each MB, its best match is searched in the two reference A frames. Since the search is restricted, it can be performed very fast. The prediction residue of the MB is then split into six subblocks, with each subblock transformed by a basis-8 DCT, quantized by scale Q_P, and then run-level Huffman coded with a P frame Huffman table. After all the MBs in the MBG are encoded, the rate and distortion of MB codings with reference to the two nearby A frames are compared. The one that offers a better rate-distortion trade off is selected as the reference frame for the MBG. One bit is encoded for each MBG to identify the reference A frame. After that, the motion vector and the prediction residue of the MBs are grouped together and encoded.

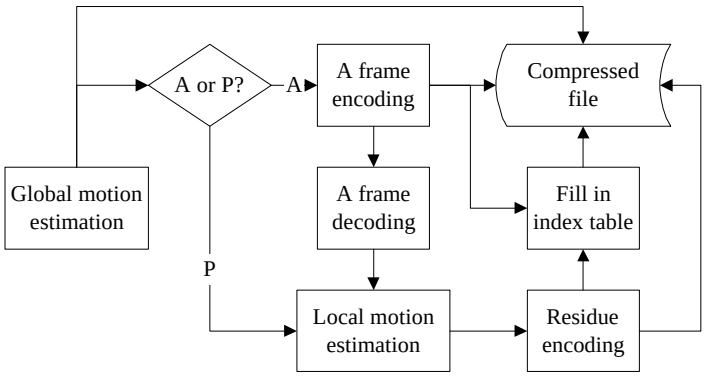

Figure 4: Flow of the RBC encoder.

The flowchart of the RBC encoding is shown in Figure 4. Given a concentric mosaic scene, the global translation vector of each P frame is calculated and encoded with a Huffman coder. After that, we encode all A frames. The A frames are decompressed immediately and used as reference for motion compensation, because the decoder can only access the compressed A frames, not the original ones. The P frames are then encoded, with each MB predicted and its residue compressed. Typically, it costs more bits to encode an A frame than a P frame, because the entropy of the A frames is much higher than that of the residue of the P frames.

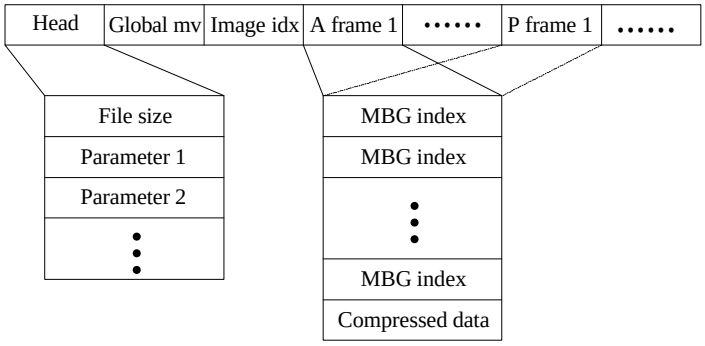

Figure 5: Syntax of the compressed concentric mosaics file.

The bitstream syntax of RBC compressed concentric mosaics is indexed by a two-level hierarchical table, and can be shown in Figure 5. After the file header, the global translation motion vectors are stored. It is followed by a first level index table, which records the encoded bitstream length of each A and P frame. The second level index table is located within the bitstream of each frame, and records the bit-stream length of each MBG. The overhead of the two-level index table is about 1% of the entire compressed bitstream and is thus very light. The two-level index table is loaded into the memory whenever the concentric mosaics are accessed.

The RBC coding scheme bears a strong resemblance to MPEG. In fact, the MB of an A frame and the MB prediction residue of a P frame are encoded exactly the same way as those in MPEG. However, RBC has a very different frame structure, motion model, and bitstream syntax from that of MPEG. In contrast to MPEG, which is a general-purpose video codec, RBC is tuned specifically for the compression of the concentric mosaics. Unlike MPEG, where a predicted P frame can refer to another predicted P frame, the P frame in RBC only refers to an A frame. MPEG allows strong motion for each MB, while the motion model in RBC is predominantly global horizontal translation, with only small local variation for individual MB due to the parallax. The two-level hierarchical index table is also unique for RBC. The modifications enhance the compression performance of RBC and enables the RBC compressed bitstream to be random accessed at the rendering stage.

4. JUST-IN-TIME RENDERING

Care must be taken not to decode the entire compressed concentric mosaic at any time, as a concentric mosaic scene can be very large. Another requirement for the decoder engine is a low computation load, so that decoding does not consume all computer resources, which are also needed for the rendering and user interaction. With the two-level index structure provided by the RBC, it is possible to access and decode only the data necessary to render the current view. When a user selects a new view of the environment, a series of slit access requests are sent by the rendering engine. It is the task of the decoder engine to provide the slits quickly. To reduce the computation load of the decoder, a number of caches have been implemented, so that recently decoded content need not be redecoded again. The key is to balance between the computation load and memory requirement.

The structure of the decoder is shown in Figure 6. There are altogether four caches, which hold the slits (in RGB space), the A and P frame MBGs (in YUV space), and the compressed concentric mosaic bitstream. The rendering engine accesses the data by slits. Whenever a slit is not in the slit cache, it will be further accessed from the A or the P frame cache, depending on where the slit is located. The engine checks if the MBG in which

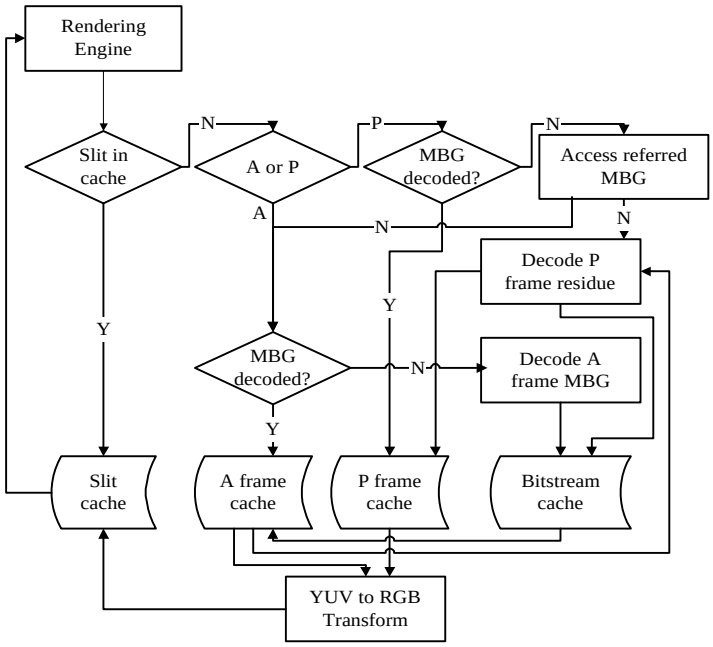

Figure 6: Just-in-time rendering flow of the reference block coding.

the slit is located is in cache. If it is, the accessed slit is converted from YUV to RGB, and it will be put in the slit cache. Otherwise, the corresponding MBG is decoded. If the MBG belongs to an A frame, it is decoded directly from the compressed bitstream with the aid of the two-level index structure. If the MBG belongs to a P frame, we must first check if all the referred A frame MBGs are in the A frame cache. The referred A frame MBG will be decoded if it is not in cache. The prediction residue of the P frame MBG is then decoded, added to the motion compensation, and stored in the P frame cache. Accessing the compressed bitstream is through the bitstream cache.

The slit, A and P frame caches are managed with a FIFO cache management strategy. A doublelink is established for each cache. Whenever a slit/MBG is accessed, the accessed content is moved to the head of the link. Whenever a slit/MBG not in cache is decoded, it is also added to the head of the link, and if the memory allocated to the cache is full, the content at the end link is dropped. The memory allocated to each cache should be large enough to cover the rendering of the current view as well as the most common movement of the user. The rendering engine typically requires that the slit cache holds *2048* slits, and the bitstream cache holds the entire compressed concentric mosaics. In the current design, the A and P frame caches hold 500 and *200* MBGs, respectively.

5. SIMULATION RESULTS

To demonstrate the effectiveness of the reference block codec (RBC), we compare the compression performance of RBC versus that of MPEG-2. Note that MPEG does not offer random access and is thus not a suitable coding tool for the concentric mosaics rendering. Nevertheless, we use the popular MPEG-2 as a benchmark to demonstrate the effectiveness of RBC. We obtain the MPEG-2 encoder from http://www.mpeg.org. In the MPEG-2, the first frame is independently encoded as I frame, and the rest frames are predictively encoded as P frames. The test concentric mosaic scenes are *Lobby* and *Kids*, as shown in Figure 7 and Figure 8, respectively. The scene *Lobby* is shot with *1351* frames at resolution *320x240*, and the scene *Kids* is shot with *1463* frames at resolution *352x288*. The *Kids* scene has more details, and is much more difficult to compress than the *Lobby* scene.

Figure 7: Concentric mosaic scene *Lobby*.

Figure 8: Concentric mosaic scene *Kids*.

The objective peak signal-to-noise ratio (PSNR) is measured between the original concentric mosaic scene and the decompressed scene. The PSNR is calculated by:

$$PSNR = 10\log_{10}\frac{255^2}{\frac{1}{N\cdot W\cdot H}\sum\left[c(n,w,h)-c'(n,w,h)\right]^2}, \quad (1)$$

where *c(n,w,h)* and *c'(n,w,h)* are the original and reconstructed concentric mosaic scene, respectively. We compress the *Lobby* scene at ratio *120:1* and *60:1*, and the *Kids* scene at ratio *100:1*, *60:1* and *40:1*. In the current implementation, no rate control is used in RBC, i.e., the quantization scale *Qa* and *Qp* is the same throughout the compression. While in MPEG2, the rate control is turned on in order to get the best performance. The two-level index table of RBC is counted in the comparison, thus the bitstream of RBC bears the property of random access, but that of MPEG-2 does not. The results are listed in Table 1. It is observed that for the Y component (luminance) that is most important in evaluation, RBC beats MPEG-2 for *0.4* to *1.7dB*, with an average gain of *1.1dB*. Considering that MPEG-2 is a highly optimized coder, the gain is significant. The rendered *Kids* scenes compressed at ratio *40:1* and *100:1* are shown in Figure 12. At compression ratio *40:1*, the scene shows very little distortion. Artifacts such as ringing and blur show up when we raise the compression ratio to *100:1*, however, the quality of the rendered view is still pretty good.

Table 1. Compression performance of RBC and MPEG2.

Scene / Approach	*Lobby*		*Kids*		
	(0.2bpp)	(0.4bpp)	(0.24bpp)	(0.4bpp)	(0.6bpp)
MPEG2 (dB)	Y: 32.2 U: 38.7 V: 38.1	Y: 34.8 U: 39.9 V: 39.1	Y: 28.3 U: 34.8 V: 34.9	Y: 30.1 U: 36.6 V: 36.7	Y: 31.9 U: 38.0 V: 38.1
RBC (dB)	Y: 32.8 U: 39.7 V: 40.5	Y: 36.1 U: 40.7 V: 41.8	Y: 28.7 U: 37.1 V: 36.6	Y: 31.5 U: 39.3 V: 38.9	Y: 33.6 U: 40.9 V: 40.5

In the second experiment, we investigate the rendering speed of RBC. The comparison codec is the spatial domain vector quantization (SVQ) used in [5] with compression ratio *12:1*. We simply replace the SVQ codec of [5] by the RBC codec. The test scene is the *Kids* scene, with original data size *424* mega bytes. The compression

ratio of RBC is *60:1*, i.e., five times that of SVQ. The bitstream cache of RBC holds *7.2* mega bytes. The slit, A and P frame caches hold *2048* RGB slits, *500* and *200* YUV macroblock groups (MBGs) with size of *1.7*, *3.6* and *1.3* mega bytes, respectively. The SVQ codec needs to hold the entire compressed file into the memory, thus requires a bitstream cache of *35.3* mega bytes as well as a slit cache of *1.7* mega bytes. The entire memory cache of RBC render is 13.8 mega bytes versus the 37.0 mega bytes of SVQ.

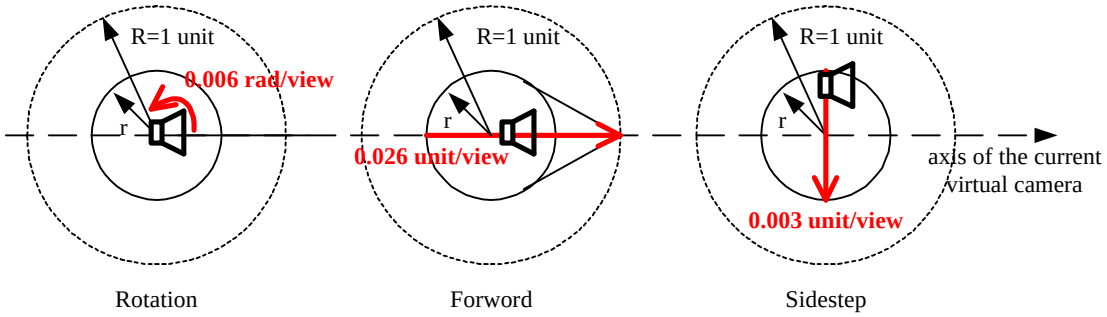

Figure 9: Three kinds of motion mode: rotation, forward, and sidestep.

The experimental platform is a Pentium II PC running at *400 MHz* with system memory large enough to accommodate all the caches. The rendering engine uses both point sampling and bilinear interpolation. Three motion passes of the viewer is simulated, i.e., rotation, forward, and sidestep modes, as shown in Figure 9. In the rotation mode, the viewpoint is at the center of the circle and rotates *0.006* radians per view. Altogether *1000* views are rendered to calculate the average frame rate. In the forward mode, the viewpoint starts at the edge of the inner circle and moves forward along the optical axis of the camera. A total of *500* views are rendered. In the sidestep mode, the viewpoint moves sidestep perpendicular to the optical axis of the camera. A total of *200* views are rendered. Sidestep is the most time-consuming mode in rendering, as it generates more new slits and causes more cache miss. The average number of frames rendered per second is shown in Table 2. Two rendering sizes *352x168* and *800x372* are used. The rendered view is shorter than the original concentric mosaic shots to compensate the depth variation [5] .

Table 2. Rendering speed of RBC and SVQ. (frame per second)

Rendering setting / Mode / Algorithm		800x372		352x168	
		Point sampling	Bilinear interpolation	Point sampling	Bilinear interpolation
Rotation	SVQ	17.6	14.6	76.9	47.6
	RBC	16.1	13.2	52.8	37.9
Forward	SVQ	17.0	14.2	71.4	45.5
	RBC	15.6	13.1	49.6	36.7
Sidestep	SVQ	15.8	13.2	53.4	37.1
	RBC	11.4	9.9	23.0	19.3

Due to higher complexity in decoding, the rendering speed of RBC is a litter slower than SVQ, especially in the sidestep mode. However, the frame rate difference between RBC and SVQ is insignificant, while the compression ratio of RBC is 5 times as much as that of SVQ, with about one third of cache memory requirement of the renderer. The rendered concentric mosaic scene looks comfortable and smooth under RBC. We have profiled the RBC codec, and the four most time-consuming components of RBC are the motion compensation, inverse DCT, YUV to RGB color transform and rendering. The first 3 components may be further accelerated if MMX instruction sets are used.

Experiments have also been performed to investigate the relationship between the cache size and the rendering speed of RBC. Since further reduction of the bitstream cache and the slit cache complicates the system design,

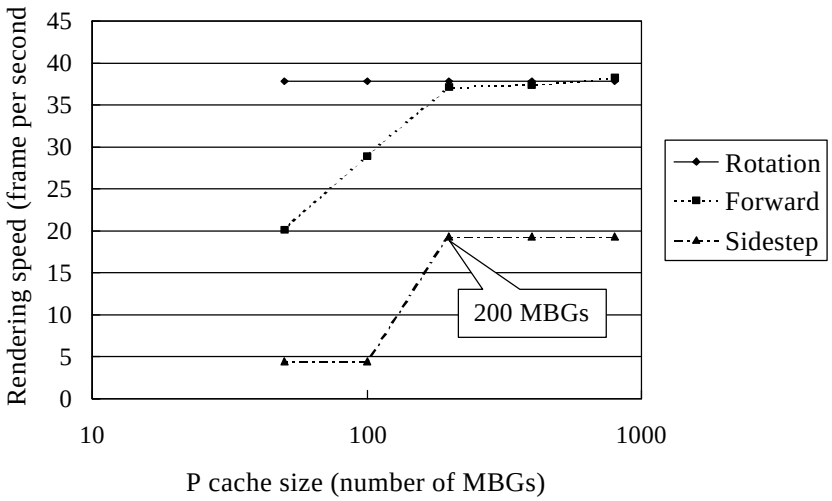

Figure 10: RBC rendering speed versus P cache size. (bilinear interpolation with A cache size 600 MBGs)

we mainly investigated the effects of A and P cache size. First, we vary the size of the P frame cache. The rendered scene is the *Kids* scene with compression ratio *60:1* at resolution 352x168. The size of the A frame cache is fixed at 600 MBGs, which is large enough to hold all the referred A frame MBGs. In Figure 10, the rendering speed with P frame cache size is plotted, with the solid, dashed and dot-dashed curves corresponding to the rotation, forward and sidestep mode, respectively. It is observed that the rendering speed is insensitive to the cache size in the rotation mode. This is because when the view rotates, a large portion of the rendered slits in a new scene are the same as those in the last rendered view, and can be taken from the slit cache directly. To provide quick rendering in the forward and sidestep mode, we find that an optimal design parameter for the P frame cache size is *200* MBGs, as a size above it shows no advantage for the rendering speed, and a size below it will cause significant speed penalty.

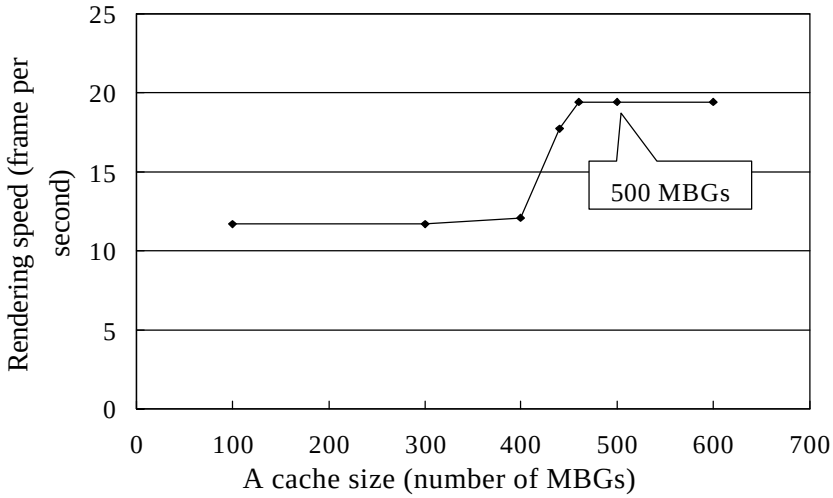

Figure 11: RBC rendering speed of sidestepping versus A cache size. (P cache size is 200 MBGs)

Figure 11 shows the relationship between A cache size and the rendering speed. The size of the P cache is fixed at *200* MBGs, and only the sidestep mode is tested because it is the slowest rendering mode and largely determines the final interactive wandering speed. It is observed that a good choice of the size of the A frame cache is around 450-500 MBGs.

6. CONCLUSIONS

A reference block codec (RBC) approach for the compression and just-in-time (JIT) rendering of the concentric mosaic scenery is presented in this paper. The algorithm not only outperforms the video coding standard MPEG-2 in compression efficiency, but also offers the capability to randomly access and decode the compressed bitstream. The rendering speed of RBC is fast enough for real time rendering of the compressed concentric mosaics.

ACKNOWLEDGEMENTS

The authors would like to thank Harry Shum, Honghui Sun and Minsheng Wu for the raw concentric mosaic data and the source codes of the SVQ concentric mosaic viewer, and Chunhui Hu for the tuning of the decodering engine.

REFERENCE

[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. 43-54, Aug.1996.

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

[5] H.-Y. Shum and L.-W. He. "Rendering with concentric mosaics", *Computer Graphics (SIGGRAPH'99)*, pp. 299-306, Aug. 1999.

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

[7] A. K. Jain, *Fundamentals of digital image processing*, Englewood Cliffs, NJ: Prentice Hall, 1989.

[8] L. Luo, Y. Wu, J. Li and Y. Zhang, "Compression of concentric mosaic scenery with alignment and 3D wavelet transform", *SPIE: Image and Video Communication and Processing, Vol. 3974, No. 10*, San Jose CA, Jan. 2000.

Figure 12 Rendered view of the *Kids* scene at compression ratio 40:1 (upper) and 100:1 (lower).