On the Compression and Streaming of Concentric Mosaic Data for Free Wandering in a Realistic Environment over the Internet

Cha Zhang[†*] and Jin Li[‡]

[†] Dept. of Electrical & Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213, USA
[‡]Microsoft Research, One Microsoft Way, Bld. 113, Redmond, WA 98052, USA

Email: czhang@andrew.cmu.edu, jinl@microsoft.com

ABSTRACT

In this paper, we describe a system for wandering in a realistic environment over the Internet. The environment is captured by the concentric mosaic, compressed via the reference block coder (RBC), and accessed and delivered over the Internet through the virtual media (Vmedia) access protocol. One of the key contributions of the paper is the proposal of the RBC concentric mosaic coder. The RBC coder not only compresses the huge dataset of the concentric mosaic very efficiently, but also produces a well-organized bitstream that can be accessed just-in-time (JIT). To reconstruct a virtual view, only a portion of the RBC bitstream needs to be accessed and decoded. This greatly reduces the memory and computation requirement of the viewer compared with first decoding the entire concentric mosaic data set and then rendering from the decoded data. Our second contribution is the employment of the Vmedia protocol to deliver the compressed concentric mosaic bitstream just-in-time over the Internet. Only the bitstream segments corresponding to the current view are streamed over the Internet. The delivered bitstream segments are managed by a local Vmedia cache, so that frequently used bitstream segments do not need to be streamed over the Internet repeatedly, and a RBC bitstream larger than the memory capacity can be easily handled. Combining RBC and Vmedia, a concentric mosaic interactive browser is developed through which the user can freely wander in a realistic environment, e.g., rotate around, walk forward/backward and sidestep, even under a tight bandwidth.

KEYWORDS: image based rendering (IBR), concentric mosaic, reference block coder (RBC), virtual media (Vmedia) access protocol, just-in-time (JIT) rendering, interactive browsing, cache.

1. INTRODUCTION

Multimedia greatly improves user experiences on the Internet, where people can now listen to news/music, watch TV, play network games, meet with friends through IP phone/videoconference, and so forth. With faster communication links and better personal computers, there has been an increasing demand to deliver a realistic 3D environment over the Internet. Rather than looking at the environment through static photos, the experience of free wandering in a 3D environment is more attractive and has many potential applications such as real-estate sale

[*] This work was done when Mr. Cha Zhang was an intern at Microsoft Research China and a student at the Dept. of Electronic Engineering, Tsinghua University, Beijing 100084, China.

on-line, virtual reality/chat room, Internet game, e-commerce, virtual museum, etc.

Traditionally, a virtual environment is represented as a collection of 3D geometric models, where the primary entities are polygons. The Virtual Reality Modeling Language (VRML) [1], which is rapidly becoming the standard file format for the delivery of 3D content across the Internet, uses polygonal models to represent the 3D content. Although the 3D modeling approach fits well the existing hardware and can be efficiently transferred and rendered, it is difficult to create the polygonal model for a complex realistic environment. Even if such a model were available, it may easily reach millions of polygons, causing difficulties in both storage and rendering.

An alternative approach to represent a 3D environment is through image-based rendering (IBR), which represents the environment with photo sets. An earlier form of the IBR approach is found in the branch movies, in which segments of movies corresponding to different spatial navigation paths are concatenated together at selected branch points, and the user is allowed to switch to a different path at these points. Examples of the branch movies are the movie-map [8], the Digital Video Interactive (DVI) demonstration [9], the "Virtual Museum" [10], etc. The branch movies are easy to shoot and quick to render. However, they require every displayable view/path to be captured and stored in the authoring stage, thus restricting the freedom of the movement and the interaction between the user and the application. Moreover, switching at the branch point is jerky, as the two adjacent movie segments are usually shot at different time, with slightly different camera setup. A huge amount of storage space is also required to store all the movie segments.

Another major thread of IBR approaches is rooted from the plenoptic function. Proposed by Adelson and Bergen [11] as a 7D function, the plenoptic function models a 3D dynamic environment 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 [12] defined plenoptic modeling as the process of generating a continuous 5D plenoptic function from a set of discrete samples or a set of images shot at different positions and towards different directions. New virtual views can be rendered from the captured images through various methods, e.g., non-physically based image mapping, mosaicking, interpolation from dense samples, and geometrically-valid pixel reprojection [13]. For a realistic environment, mosaicking and interpolation from dense samples are the two most effective techniques, as they can render very complex scenes without the knowledge about the scene geometry.

A famous mosaicking system is Apple's QuickTime™ VR [14], a virtual reality extension to the QuickTime digital multimedia framework. QuickTime VR uses multiple environment maps, i.e., the cylindrical panoramic images, to represent a scene. An environment map allows the user to stay at certain virtual point and look around in arbitrary direction through reprojection or warping. With multiple environment maps, the user may hop around the scene as well. Currently, the QuickTime VR system is the most widely available virtual reality representation on the Internet. However, in QuickTime VR the movement of the user is limited to panning, zooming or hopping. Free wandering in the environment is difficult unless the capturing points are very dense, which causes a huge increase in the captured data amount.

The Lightfield [15] and the Lumigraph [16] were the first practical approaches to interpolate novel views from dense sampled photos. They provided a clever 4D parameterization of the plenoptic scene under the restriction that the object or the viewer could be constrained within some 3D bounding boxes. Shum and He [17] proposed the concentric mosaic, which could be considered as a 3D plenoptic function. One attractive advantage of the concentric mosaic is that it can construct a representation of the realistic environment very easily: we just rotate a single camera at the end of a level beam, with the camera pointing outward and shooting images as the beam rotates. The movement of the user is restricted on a plane coincident with the rotating plane of the beam, which is the usual movement of the user as he/she wanders freely in the environment. Rendering virtual views in a scene represented by the concentric mosaic is straightforward. The virtual view is split into vertical ray slits, and each slit is reconstructed through close-by captured ones. One concentric mosaic can render views at arbitrary points and looking at arbitrary directions within a planar circle. To enable the user to smoothly wander around in a large environment, multiple concentric mosaics may be concatenated [23]. Though there are weaknesses in the approach, e.g., vertical distortion caused by incorrect constant-depth assumption of the current concentric mosaic setup [17], the view of the concentric mosaic is assembled from real photos, the sense of reality augmented by complex parallax and lighting changes is unparallel compared with a geometric model based representation.

Representing a complex environment is hard; delivering the experience of wandering in a complex environment over the Internet is even harder. A realistic scene representation involves huge amount of data, which is slow to be delivered over the Internet. MPEG-4 has developed algorithms to compress the polygonal models and deliver them over the Internet. Work has been done to reduce the VRML file size and progressively stream the

file over the Internet [2][3][4][5][6]. The VRML Dream Project [7], a streaming VRML entertainment project with a running time of more than two minutes, proves that it is possible to use VRML to broadcast the 3D real-time animation over the Internet. However, for a realistic environment, the number of polygons is so large that it is impractical to deliver them over the Internet, especially as most end users are still using a modem link.

The amount of raw data required for IBR representation is even greater. As an example, the concentric mosaic scene *Lobby* (shown in Figure 12) consists of 1350 frames at resolution 320×240 and occupies a total of 297 megabytes. Luckily, the data set is highly correlated and can be efficiently compressed. Moreover, to render a certain view of the environment, only a small portion of the data is needed. With the careful handling of the above two issues, we can bring the experience of wandering in a realistic environment constructed by the concentric mosaic over the Internet. One of our key contributions in the paper is a concentric mosaic compression algorithm, the reference block coder (RBC). RBC not only is highly efficient in the compression performance, but also is able to decode and render the view just-in-time (JIT). The compressed bitstream of the concentric mosaic is organized through a two-level hierarchical table. The first level of the hierarchical table indexes the compressed bitstream of the image shots, and the second level indexes macroblock groups (MBG) within the image shots. The image shots are classified into two categories: the anchor (A) frame which is independently encoded, and the predicted (P) frame which is encoded through motion compensation with reference to an A frame. With such an arrangement, only portion of the compressed RBC bitstream needs to be accessed and decoded to render a specific virtual view of the scene, and that portion of the bitstream can be quickly located through the two-level hierarchical table.

Compared with downloading the entire compressed concentric mosaic, decompressing it, and then rendering views from the decompressed concentric mosaic dataset, the JIT decoding and rendering approach obviates the need for the renderer to hold an uncompressed concentric mosaic in memory, which is usually impractical even though the computer today is much more powerful than its predecessors. Moreover, it enables quick delivery over the Internet as only portion of the RBC bitstream related to the current view needs to be downloaded. This is the key that enables the wandering in an environment over the slow Internet.

Our second contribution is to deliver the bitstream segments required for the rendering of the current view just-in-time over the Internet. A new media delivery protocol termed the virtual media (Vmedia) access protocol is

developed. The Vmedia protocol also caches and manages the delivered bitstream segments so that the same segment is not delivered repeatedly over the Internet. With the RBC and Vmedia protocol, we build an efficient 3D web-browser system. We enable the user to freely wander in a realistic environment constructed by the concentric mosaic even when the network bandwidth is tight. Although work has been done to compress the Lightfield or Lumigraph in support of Internet streaming [20][21], this is so far the first obtainable system on delivering compressed interpolation-based IBR bitstream over the real Internet.

The paper is organized as follows: the concentric mosaic and prior compression approaches are briefly reviewed in Section 2. The compression of the concentric mosaic by the reference block coder (RBC) is explained in Section 3. The Vmedia access protocol is described in Section 4. The implementation of the environment browser is explained in Section 5. Extensive simulation results are presented in Section 6. The conclusions and future work are given in Section 7.

2. THE CONCENTRIC MOSAIC AND PREVIOUS WORK ON ITS COMPRESSION

A concentric mosaic can be easily captured by mounting a camera at the end of a round-swinging beam, and shooting images at regular intervals as the beam rotates [17]. The capturing device of the concentric mosaic can be shown in Figure 1. The resultant dataset is a video sequence. For a typical realistic environment with large depth variation, 900 to 1500 shots have to be captured in a circle to render the scene properly without aliasing. This leads to a dataset of several hundred megabytes.

Rendering the concentric mosaic involves reassembling slits from existing photo shots. No 3D or depth information of the scene is needed to reconstruct a novel virtual view of the environment. 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 looking at any directions within an inner circle of radius $r = R\sin q_{FOV}$. Shown in Figure 2, let P be a novel viewpoint and AB be the field of view to be rendered. We split the view into multiple vertical slits, and render each slit independently. Let the slit PV be a rendered slit. We simply search for the slit $P'V$ in the captured dataset and use the content of $P'V$ to replace PV, where P' is the intersection between ray PV and the camera path. The basic hypothesis here is that the intensity of any light ray does not change along its path unless blocked. Therefore, what is rendered at PV can be recovered from $P'V$. Since the concentric mosaic

consists of a finite number of shots, and each shot consists of a group of vertical slits, the exact slit $P'V$ may not be captured in the discrete dataset. Let the four slits closest to $P'V$ be P_1V_{11}, P_1V_{12}, P_2V_{21} and P_2V_{22}, 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_{11} and P_1V_{12} are the two slits on the two sides of ray P_1V in shot P_1, and P_2V_{21} and P_2V_{22} are the two 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 and network bandwidth constraints, we may use the one slit closest to $P'V$ to represent it (denoted as point sampling mode). In either case, the content of the slit $P'V$ is recovered, which is then used to render slit PV. A slit can be rendered as long as its reference slits, i.e., the four slits in the bilinear interpolation mode or the closest one slit in the point sampling mode, are available. A view can be rendered when all the reference slits are available. Strictly speaking, vertical distortion is perceptible as there is no vertical parallax when the viewer moves forward and backward. However, since the view of the concentric mosaic is assembled from real photos, the sense of reality augmented by complex horizontal parallax and lighting changes is unparallel compared with the geometric model based representation.

There has been work concerning the compression of the concentric mosaic. In [17], the concentric mosaic is split into small blocks, which are then compressed by spatial domain vector quantization (SVQ). The compressed bitstream consists of a large SVQ codebook and the VQ indexes for each block of the concentric mosaic. SVQ is simple and fast to decode. The SVQ indexes are of fixed length and can be flexibly accessed and error resilient. However, SVQ is time-consuming at the encoding stage, and the compression ratio of SVQ is limited, e.g., only 12:1 in [17]. Moreover, in order to render a virtual view from the SVQ compressed concentric mosaic, the SVQ codebook must be delivered first, which results in a long initial wait if the compressed scene is browsed over the Internet. Since the concentric mosaic consists of a sequence of images, it is possible to compress the concentric mosaic with an existing image/video coding standard. We may compress each individual shot of the concentric mosaic separately with the still image compression standard JPEG. However, the compression will not be efficient because correlation across the shots is not used. Video-based coder such as MPEG-2 achieves good compression ratio, however, the compressed bitstream is coupled compactly and has to be downloaded and decoded before any view of the concentric mosaic can be rendered, which is slow and awkward. 3D wavelet approaches have also been proposed for the compression of the concentric mosaic [18][19]. The 3D wavelet algorithms achieve high

compression ratio, and may choose portion of the compressed bitstream to access with selected resolution and quality level. However, 3D wavelet coding and rendering system is complex to implement and computationally much more expensive.

3. THE REFERENCE BLOCK CODER (RBC)

We compress the concentric mosaic with a new scheme called the reference block coder (RBC). RBC bears strong resemblance to the MPEG video coder. It segments image shots into macroblocks, which are then encoded through motion compensation and residue coding. However, the frame structure of RBC is redesigned to improve the compression efficiency and to enable a virtual view of the concentric mosaic to be rendered with the access of a small portion of the bitstream.

The data structure of the reference block coder (RBC) is shown in Figure 3. The concentric mosaic frames 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 serve as the anchor to be accessed for the decoding operation. We encode the A frames independently, while encode the P frames predictively with reference to the two nearby A frames. The ratio of the number of P frames to the number of A frames is a tradeoff between compression efficiency and random accessibility. In the current implementation, the ratio is set to be *7*, i.e., one out of eight frames is an A frame.

The A and P frames are further segmented into macroblocks (MB), with size 16x16. Since vertical slit is accessed to render a view of the concentric mosaic, all MBs at the same horizontal position of a shot are grouped together and form a macroblock group (MBG). MBG is the smallest unit for encoding and delivering over the Internet.

The flowchart of the RBC encoder is shown in Figure 4. 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 horizontal and vertical directions. 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 bitstreams of all MBs belonging 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, a single MB in the P frame 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 data to be decoded when a slit in the P frame MBG is accessed. A two-stage motion estimation, including a global translation motion and a local refinement motion, is then applied to calculate the motion vector of each MB. The dominant global horizontal translation vectors *mv1* and *mv2* of the P frame with regard to the two reference A frames are calculated and recorded to account for the swinging motion of the camera. The global translation vector effectively reduces the search range of the local refinement vectors of the P frame MBs, which is restricted to *+/-5* pixels horizontally and vertically with half pixel accuracy. Experiments show that around half of the local refinement motion vectors of the P frame MBs are zeros. Therefore, most MBs just move along the underlying P frame. To encode a P frame MBG, we encode the MBG against both reference A frames. For each MB in the MBG, its best match is searched in both reference A frames. Since the search is restricted, this can be performed very quickly. After the motion search, the prediction residue is calculated by subtracting the decompressed A frame from the P frame, as the decoder can only access the decompressed A frames, not the original ones. 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 coding 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. The compressed bitstream of MBG is formed by the reference identification bit, the compressed local refinement vector and the prediction residue of the MBs.

The compressed bitstream of RBC is organized with an index structure so that an arbitrary MBG can be easily accessed and decoded. Shown in Figure 5, the RBC bitstream is led by an information header, which includes crucial information of the concentric mosaic scene, such as the size of the scene, the coding and rendering parameters. After the information header, a thumbnail of the panorama of the environment is stored. The thumbnail is compressed as an A frame, and provides a quick overview of the whole environment. After the thumbnail, it is the compressed bitstream of the global translation vectors of P frames, which are encoded with differential pulse code modulation (DPCM) and a Huffman coder. A two-level hierarchical index table follows the

compressed motion vectors, which records the encoded bitstream length of each A and P frame in the first-level index and that of each MBG in the second-level index. The information header, the thumbnail, the compressed global translation vectors and the two-level hierarchical index table form the file header of the compressed RBC bitstream. The size of the file header is small, typically 1-2% of the entire bitstream. The file header must be downloaded before any browsing of the environment can be performed. After the file header, it is the compressed bitstream of the A and P frames.

RBC is balanced between the compression efficiency and fast decoding. The basic access unit of RBC is the MBG, which is larger than the access unit (the slit) of the rendering engine. Therefore, redundant slits are accessed and decoded for the rendering of an individual view. Nevertheless, grouping slits into MBGs greatly improves the compression efficiency and reduces the percentage of the overhead required by a basic access unit, such as the bitstream index, motion vectors, etc.

The RBC coding scheme bears a strong resemblance to the MPEG. In fact, the MB of the A frame and the MB prediction residue of the 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, all of which are tuned specifically for the compression of the concentric mosaics (or other IBR images). Unlike MPEG, where a P frame can refer to another predicted P frame, the P frame in RBC only refers to an A frame, thus reducing the number of frames to be accessed when a view is rendered. MPEG allows strong motion for each MB, while the motion model in RBC is predominantly global horizontal translation, with only small local variations for the individual MBs. This matches with the camera panning motion in the creation of the concentric mosaic. The two-level hierarchical index table is also unique for RBC. These modifications improve the compression performance of RBC and enable a small portion of the RBC compressed bitstream to be accessed for rendering a specific view of the concentric mosaic.

4. THE VIRTUAL MEDIA (VMEDIA) ACCESS PROTOCOL

When a RBC compressed concentric mosaic scene is browsed over the Internet and a virtual view of the scene is rendered, the rendering engine only uses a small portion of the slits, and thus, only the compressed bitstream segments referred by these slits need to be delivered over the Internet. Rendering a novel view of the concentric mosaic requires a new set of bitstream segments. Since there are reusable bitstream segments between

the old and the new views, it is essential to cache and manage the downloaded bitstream segments so that the same bitstream segment is not downloaded repeatedly over the Internet.

A virtual media (Vmedia) access protocol is developed to provide the just-in-time bitstream delivery, as well as the cache and management of the downloaded bitstream segments. Rather than treating the entire media file as a whole, or streaming the media file sequentially, Vmedia enables accessing and streaming only the needed bitstream segments. The framework of Vmedia can be shown in Figure 6.

In the framework, a Vmedia server and several Vmedia clients are present. Most of the functionalities of Vmedia lie in the Vmedia client. The application calls the Vmedia client to access the remote media. Upon connection, the Vmedia client mirrors the remote media as a virtual local file (hence the name Vmedia), and manages it with a local Vmedia cache. The virtual local copy looks exactly the same in structure as the remote media; however, only a portion of the remote media may be accessible depending on the content in cache. Through the Vmedia client, the application can access segments of media, say with start position *pos* and length *len*. Vmedia checks if the accessed media segment is already in cache. If it is, the content is returned to the calling application. If it is not, a network request is queued to stream the missing segment from the Vmedia server. The media segment is accessed with a priority and an importance tag, which aid the streaming and cache management, respectively.

Vmedia provides the following functionalities:

1. Flexible media access
2. Cache and management of the delivered media segment

The use of cache prevents the same media segment from streaming over the network repeatedly. It also enables the client to access a compressed media much larger than its memory limitation. Though file cache has been widely used by the Internet Explorer™ and Netscape™, there is no implementation of file segment cache.

3. Prioritized delivery

High priority segments, such as those of A frames, are delivered first over the Internet to enable a low quality rendering before the rest bitstream segments arrive.

4. Media segment packaging

Small media segments are automatically packaged into a larger one, and large media segments are broken

into small packets according to the maximum transmission unit (MTU) of the network path. Packet loss and retransmission are also handled.

Through a group of unified APIs, Vmedia hides most of the above chores. The client media application simply accesses the remote media as a virtual local file, and issues media segment access requests with priority and importance. We explain the workflow of Vmedia concentric mosaic browser in Section 4.1. The two key components of the Vmedia, the Vmedia cache and the media segment delivery, are briefly explained in Section 4.2 and 4.3, respectively.

4.1 THE WORKFLOW

The application flow of the Vmedia concentric mosaic browser is shown in Figure 7. The browser calls the Vmedia client to access the remote media. Upon the receipt of the call, the Vmedia client contacts the Vmedia server, establishes the connection, and verifies the existence of the remote media. During the connection phase, the structure of the media, in this case the RBC file header that includes the information header, the thumbnail, the compressed global translation vectors and the two-level hierarchical index table in Figure 5, is downloaded to the Vmedia client to aid the subsequent just-in-time access. The Vmedia cache is also initialized to be empty at the connection phase. With the RBC file header, the browser has all the information for positioning the bitstream segments. To render each concentric mosaic view, the browser locates the bitstream segments required for the current view, and issues an access request for each bitstream segment. For each access request, the Vmedia client first checks whether the requested segment is in the Vmedia cache. If the entire segment is in the Vmedia cache, it is returned to the browser. If only part or none of the segment is in the cache, a pending network request is generated and sent to the Vmedia server to download the missing segments.

Vmedia works in an asynchronous mode, where the media segment request is served on a best effort basis. Vmedia always returns the control immediately to the browser regardless of the outcome of the request. To best utilize the Vmedia protocol, the browser also performs the rendering on a best effort basis. It repeatedly queries and renders using whatever data available. A partial view can be quickly rendered with a minimum amount of arriving data. The view can then be gradually improved as more and more media segments are received.

4.2 VMEDIA CACHE

A cache is used to hold the received media segments and to identify portions of the media that can be

accessed immediately. There are two major cache operations in Vmedia:

1. Cache hit detection.

The operation checks if a requested segment is in cache, and returns that segment if it is available.

2. Cache update

When a new media segment arrives from the network, memory needs to be allocated to store the arriving segment. If there is no memory left, certain less frequently used and less important media segments need to be thrown out.

In the Vmedia, we use fixed size media unit to improve the speed of cache hit detection and reduce the memory overhead needed to hold the Vmedia cache. As shown in Figure 8, the minimum memory allocation and cache management unit is a media unit (MU). Each MU is further broken into L sub media units (SMU), which is the smallest element for delivery. In the current implementation, $L = K = 32$. Each SMU is composed of K content bytes and a one-bit validity flag. In additional to the component SMUs, each MU also maintains an importance tag, a hit count and a lock flag. The MUs are indexed by a lookup table. If a particular MU has been allocated in cache, its lookup table entry points to the allocated memory. If the MU has not been allocated, the lookup table entry is empty.

With the MU/SMU structure, cache hit detection can be performed very quickly. To check whether a portion or all of an arbitrarily positioned media segment is located in the Vmedia cache, we break the media segment into MUs and further into SMUs and check the validity of each SMU. This can be performed by first referring to the MU lookup table and then, if the lookup table entry is not empty, further checking the validity tag of the SMU. The cache can also be easily updated. Whenever a new media segment is received, we again break the media segment into SMUs. We first check if the MU associated with the media segment is allocated, i.e., if its lookup table entry is valid. If it is, we simply store the SMUs into their appropriate places and turn on the validity bits of the SMUs. If it is not, a memory is allocated for the MU of the newly arrived segment. In case all cache memory is used up, some less frequently used (smaller hit count value) and less important (smaller importance tag value) MUs are thrown away, and their memory are reused to store the MUs of the arriving media segment.

4.3 DELIVERY OF THE MEDIA SEGMENTS

Each media segment access request is attached with a priority tag, which is used to prioritize the media

delivery. In case one or more SMUs of the requested media segment are not available, they are scheduled for delivery from the Vmedia server with the priority of the requested media segment. Higher priority SMUs are scheduled to be delivered earlier, and lower priority SMUs are scheduled to be delivered later. To prioritize the delivery requests and handle the packet loss, Vmedia maintains two queues: a pending queue which is the collection of the SMUs whose requests have not been sent to the server yet, and a sent queue containing the SMUs whose requests are sent to the server but the requested SMUs have not been received by the client yet. The pending queue consists of a number of sub queues holding pending SMUs of different priorities. Once there is a cache miss for a SMU, it is checked whether the corresponding SMU is already scheduled for delivery in the pending queue or the sent queue. This avoids the issuing of the same request twice, though the priority of the request may be adjusted to the highest priority issued. Once a SMU request is sent to the server, the corresponding SMUs are time stamped and moved to the sent queue. Whenever a SMU is received from the server, the Vmedia clears the corresponding request in the sent queue, and stores the received SMU in cache.

In the event of view change, namely a radically different set of media segments are accessed, a clear function call is issued by the browser application to the Vmedia client to eliminate all requests in the pending and sent queue. The Vmedia client also contacts the Vmedia server to cancel all the requests not processed by the server. The rational is that canceling all existing requests in both the pending queue and the server speeds up the delivery of the contents associated with the new view. SMUs that are already in route for delivery are still processed by the Vmedia client and stored in the Vmedia cache upon arrival.

Vmedia may access a normal HTTP server, in which case the request is in the form of a partial GET HTTP request. Alternatively, a special Vmedia server can be setup, and communicate with the Vmedia client through the UDP protocol to deliver the SMU requests and the returned SMU segments. This not only reduces the network overhead associated with the requests and returned SMU segments, but also reduces the load of the server. For efficiency, the Vmedia client bundles multiple SMU requests into a single UDP packet and sends it to the server. Similarly, the server also bundles multiple SMUs into a single UDP packet and sends it back to the Vmedia client.

The functionality of the Vmedia server is rather simple: it handles the SMU request coming from the Vmedia client, and returns the requested SMUs. The prioritization and caching are all handled by the Vmedia client. Such design puts more computation load at the client, and reduces the burden of the server. Though in this paper, the

Vmedia is developed for the interactive concentric mosaic browsing, it can be used with any type of media, e.g., interactive image browsing [22]. Remote media is simply treated as an unstructured, one dimension file, from which portions of the media are accessed with priority.

5. THE ENVIRONMENT BROWSER

With the Vmedia access protocol and the reference block coder (RBC) presented above, we develop a 3D environment browser over the Internet. The workflow of the Vmedia concentric mosaic browser can be illustrated in Figure 9. Caches are used extensively to speed up the rendering. Four caches are involved in the system: the slit cache which holds the vertical slits to be rendered (in the RGB color space), the A and P frame caches which hold the A and P frame MBGs (in YUV color space), and the Vmedia cache which holds the compressed RBC bitstream segments. The key of designing the caches is to balance between the computation load and the memory consumption. During the rendering of a view of the environment, the rendering engine accesses the vertical slit from the slit cache. If the slit is not in the slit cache, it will be further accessed from the A or the P frame cache, depending on which frame the slit is located. If the MBG containing the slit is located in cache, the slit is copied from the frame cache, converted from YUV to RGB space, and then put in the slit cache. If the slit belongs to an A frame and it is not in the A frame cache, the corresponding MBG bitstream is located via the two-level hierarchical index table, accessed through the Vmedia API, decoded and put into the A frame cache. The accessed slit is again converted to the RGB space and put in the slit cache. If the MBG bitstream is not available, a request is generated by the Vmedia to deliver the bitstream segment over the Internet. To decode a P frame MBG, both the prediction residue of that MBG and all the reference A frame MBGs are needed. The reference A frame MBGs are accessed from the A frame cache as described above. The compressed bitstream of the prediction residue is also located via the two-level index table and accessed through the Vmedia. If the bitstream segment is available, the prediction residue is decoded and added to the referred A frame MBG to recover the P frame MBGs. If the bitstream segment is not available, a similar request is generated by Vmedia. At the end of frame rendering, a synchronization function call is issued by the browser so that the Vmedia can prioritize all the requests, bundle them, and send them to the Vmedia server, which sends back the requested bitstream segments. The bitstream segments of the A frame MBGs are given higher priorities and importance than those of the P frame MBGs, so

that they are delivered earlier over the network and stay longer in the cache. A pseudo-code of the whole slit access process is listed below.

```
Function BYTE* access_slit (framenum, slitnum)
{
    if (slit in slit cache)
        return memory pointer of the slit;
    end
    switch (slit type)
    {
    case A frame slit:
        if (corresponding MBG is in A cache)
            copy slit to the slit cache;
            return memory pointer of the slit;
        else
            call Vmedia to retrieve related bitstreams;
            decode corresponding MBG to A cache;
            copy slit to the slit cache;
            return memory pointer of the slit;
        end
    case P frame slit:
        if (corresponding MBG is in P cache)
            copy slit to the slit cache;
            return memory pointer of the slit;
        else
            call Vmedia to retrieve all the related bitstreams;
            decode referred A MBGs to A cache;
            decode corresponding MBG to P cache;
            copy slit to the slit cache;
            return memory pointer of the slit;
        end
    }
}
```

The above task of rendering, slit accessing, MBG decoding, bitstream segment accessing is repeatedly performed by the Vmedia concentric mosaic browser. In each step of the iteration, the browser renders a current view on a best effort basis according to the available bitstream segments. At first, none of the bitstream segments is available, and a blank view is rendered. When some bitstream segments arrive, the corresponding A or P frame MBGs are decoded, and slits in the MBGs are rendered in the view. The slits that are unavailable are still rendered as blank. An example of an intermediate rendered concentric mosaic view is shown in Figure 10, where the slits in the right part of a view are unavailable and rendered as blank. According to the status bar, 11.7 kilobytes of compressed bitstream is still to be delivered from the server before the view can be completely rendered. As more and more bitstream segments arrive, the blank area becomes smaller and smaller, and the quality of the view

gradually improves. Although it might be possible to render some interpolated slits when the data have not arrived, rendering with blank for the missing data gives the user an explicit sign that data are missing there. Moreover, subjective test shows that blank slits are actually less annoying than interpolated slits.

The slit, A and P frame caches are managed with a least recent used (LRU) cache management strategy. A double link is established for each cache. Whenever a slit/MBG is accessed, the accessed slit/MBG 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 of the cache is full, the slit/MBG at the end of the 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. For rendering views at resolution 800×372, we typically set the slit cache to hold 2048 slits, and set the A and P frame cache to hold 500 and 200 MBGs, respectively. Detailed experiments on the cache sizes are described in the next section.

6. SIMULATION RESULTS AND SYSTEM PERFORMANCE

In this section, we demonstrate the effectiveness of the system by showing the compression performance of the RBC coder, its just-in-time rendering capability and its capability to browse concentric mosaic represented environment over slow Internet link.

We compare the compression performance of RBC versus that of MPEG-2 and the 3D wavelet codec [18]. The test concentric mosaic scenes are Lobby and Kids, as shown in Figure 12 and Figure 13, respectively. The scene Lobby has 1350 frames with resolution 320×240, and the scene Kids has 1462 frames with resolution 352×288. The Kids scene has more details, and is much more difficult to be compressed than the Lobby scene. The Lobby scene is compressed at ratio 120:1 and 60:1; and the Kids scene is compressed at ratio 100:1, 60:1 and 40:1.

Note that MPEG-2 does not offer random access and is thus not a suitable coding tool for the concentric mosaic. Nevertheless, it is used as one of the benchmark because it is a popular tool for compressing a sequence of images. We obtain the MPEG-2 encoder from http://www.mpeg.org/. In the MPEG-2 encoding of the concentric mosaic, the first frame is independently encoded as I frame, and the rest frames are predictively encoded as P frames. We turn on the MPEG rate control in order to achieve the desired bitrate. We have not

implemented the rate control in RBC. Therefore, we use a constant quantization scale Q_a and Q_p in RBC coding, and manually tune the Q_a and Q_p to hit the desired compression ratio. The 3D wavelet codec in [18] aligns and then encodes the concentric mosaic with a state-of-the-art wavelet codec. We use it as another benchmark codec. Since it is an embedded coder, it can reach the desired coding bitrate easily by simply truncating the resultant bitstream to satisfy the desired compression ratio.

To measure the quality of compression, the objective peak signal-to-noise ratio (PSNR) is measured between the original concentric mosaic scene and the decompressed scene as:

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

where *c(n,w,h)* and *c'(n,w,h)* are the original and the reconstructed concentric mosaic scenes, respectively. *N* is the number of images in the scene; *W* and *H* are the width and the height of each image. We measure the PSNR of the Y, U and V component, but mainly comment on the results of the Y component, because it has the biggest influence on the subjective quality and consumes most of the coding bits.

The results are listed in Table 1. RBC outperforms MPEG-2 for 0.4 to 1.7dB (Y component), with an average gain of 1.1dB. We do observe that the RBC also outperforms MPEG-2 significantly in U, V component coding. Considering that MPEG-2 is a highly optimized coder for image sequence, the gain is significant. Compared with the 3D wavelet coder, RBC outperforms by 0.9-2.1dB (Y component) in PSNR. Looking at the compressed concentric mosaic subjectively, we may draw the conclusion that the RBC is better in preserving sharp details than the 3D wavelet scheme. However, the blocking artifacts are more visible in RBC than 3D wavelet. The rendered RBC compressed Kids scenes with ratio 40:1 and 100:1 are shown in Figure 18. With the compression ratio of 40:1, the scene shows very little distortion. Artifacts such as ringing and blur show up when we increase the compression ratio to 100:1. However, the quality of the rendered view is still pretty good with the compression ratio of 100:1.

In the second experiment, we investigate the just-in-time rendering capability of the RBC by measuring its rendering speed and the requested cache. We put the entire compressed concentric mosaic in the Vmedia cache, thus no network request is issued and the application runs locally. The comparison coder is the 3D wavelet coder. Since MPEG-2 does not have the just-in-time rendering capability, as it has to decode the entire compressed

concentric mosaic and then render, it is not included in this experiment. For more comparison, we also include the spatial domain vector quantization (SVQ) used in the original concentric mosaic paper [17] with the compression ratio of 12:1. The test scene is the Kids scene, with original data size 424 megabytes. The compression ratios of RBC and the 3D wavelet coder are 60:1, i.e., five times that of the SVQ. The bitstream cache of RBC holds 7.2 megabytes. 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 megabytes, respectively. Altogether, the RBC viewer requires 13.8 megabytes cache space. The SVQ coder needs to hold the entire compressed file into the memory, thus it requires a bitstream cache of 35.3 megabytes as well as a slit cache of 1.7 megabytes. The implementation of the 3D wavelet coder is described in [18]. It requires over 40 megabytes in memory during the browsing operation. A quick summary of the various experimental conditions is given in Table 2.

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 patterns of the viewer are simulated, i.e., rotation, forward, and sidestep, as shown in Figure 14. In the rotation mode, the viewpoint is at the center of the concentric mosaic circle and the viewing direction rotates 0.006 radians per view. Altogether 1000 views are rendered to calculate the average rendering 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 sidewise 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 requires more new slits and causes more cache miss. The average number of frames rendered per second is shown in Table 3. Two rendering sizes 352×168 and 800×372 are used. The rendered view is shorter than the original concentric mosaic shots to compensate for the vertical distortion [17].

We observe that the walkthrough of the scene is comfortable and smooth under RBC. The rendering speed of RBC is slightly slower than the 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, and the cache memory for RBC is one third that of SVQ. Compared with the 3D wavelet scheme, the RBC has a higher rendering speed and a lower memory requirement. Notice that the advantage is significant for the sidestep mode, because sidestep requires the decoding of many new slits and the 3D wavelet scheme suffers from the

complex cache operations. Currently, the entire RBC browser is written in C++. We have profiled the RBC browser. The four most time-consuming components are the motion compensation, the inverse DCT, the YUV to RGB color transform and the rendering. The first 3 components can be greatly accelerated if MMX instruction sets are used.

Experiments have also been performed to investigate the trade-off 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, we mainly investigate the effects of the A and P cache size. First, we vary the size of the P frame cache. The rendered scene is the Kids scene with the compression ratio of 60:1 at resolution 352×168. 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 15, the relationship between the rendering speed and the 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 viewer 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. It shows no significant improvement for the rendering speed as the size increases, and slows the rendering speed significantly as the size decreases.

In Figure 16, the relationship between the A frame cache size and the rendering speed is shown. The size of the P frame 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. We thus select 500 MBG as the A frame cache size in our rendering system.

Finally, we describe the experience of just-in-time environment browsing. The browser embedded in a web page showing the concentric mosaic scene Lobby is illustrated in Figure 17. The scene is RBC compressed with the compression ratio of 120:1, and the compressed bitstream is 2.5 megabytes in size. The file header of the bitstream includes a small information header, a compressed thumbnail view of the environment (about 4.6 kilobytes), the global translation vectors and the two-level hierarchical index table (about 40 kilobytes). The entire file header occupies a total of 45 kilobytes. An ActiveX Vmedia concentric mosaic viewer is then used to access

the compressed concentric mosaic through the Internet. The experiment is performed with the Internet connection speed set at 33.6 kbps (kilobits per second), the typical modem speed per ITU v.34 standard. With higher Internet connection speed, the speed of virtual environment browsing will be proportionally faster.

During the connection phase, the information header and the thumbnail are downloaded from the Vmedia server. This takes around 2 seconds. The thumbnail is then displayed in the browser window to give the user an overview of the environment, as shown in Figure 11. The user may select an entry of the scene by double click a region of interest, which determines the initial view. The rest of the file header, i.e., the global translation vectors and the two-level index table (see Figure 5), is then downloaded from the Vmedia server before any virtual view of the environment is rendered, as they are indispensable for the accessing and decoding of the MBGs. This takes around 10 seconds. After that, the current view is rendered according to the position of the viewpoint and the selected viewing directions. A set of slits necessary to render the current view is accessed from the slit cache, which in turn triggers the access of the MBGs in the A or P frame cache, and the access of the compressed bitstream segments from the Vmedia API. Currently, we render the unavailable slits as black. Only the bitstream segments necessary to decode the slits used in the current view are accessed and decoded. This greatly improves the response time when a view is rendered. Typically, in the bilinear interpolation mode, 30 to 100 kilobytes of compressed bitstream segments are accessed to decode a novel view. Even with a limited network bandwidth of 33.6 kbps, the entire novel view can be rendered in 7 to 25 seconds. To render a subsequent view during the wandering, i.e., rotate, walk forward/backward and sidestep, about half of the bitstream segments will be already available in the Vmedia cache due to the rendering of the previous views. The unavailable bitstream segments are thus only about 15 to 50 kilobytes. A subsequent view can usually be completely rendered within 5 seconds. The RBC browser may operate in the point sampling mode. However, we observe that though far fewer slits are accessed in the point sampling mode, the requested compressed bitstream segments are about the same as the bilinear interpolation mode. That is because each four slits accessed in the bilinear interpolation mode are clustered so close to each other that they are frequently in the same MBG and do not require the access of additional segments. An example of an intermediate rendered concentric mosaic view is shown in Figure 10. At the bottom of the browser window, there is a progress bar showing the amount of bitstream segments to be streamed for the current view.

We have also compressed the Lobby scene with the compression ratio of 200:1. The compressed bitstream is reduced to 1.5 megabytes. The file header becomes smaller, too. The thumbnail takes up 3.5 kilobytes, and the index table takes up 35 kilobytes. A novel view now occupies 15 to 40 kilobytes. The response time of the browser improves significantly at the cost of low quality of the rendered view. The user can freely wander in the environment, rotate, walk forward/backward or sidestep as he/she wishes. The browser quickly responds to the request of the user, and renders the views that are actually seen in the real environment. A pleasant remote virtual environment experience is offered, even when the bandwidth of the connection is as low as 33.6kbps.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we developed a system that allows a user to wander in a realistic environment over the Internet. The environment was captured by the concentric mosaic, compressed via the reference block coder (RBC), and accessed and delivered over the Internet through the virtual media (Vmedia) access protocol. For the first time, we allow the user to wander in a realistic environment through the Internet, even with limited network bandwidth.

The current Vmedia concentric mosaic browser can be improved in several aspects. The response time of the browser is still not fast enough, especially during the start-up stage. We may encode each macroblock group (MBG) progressively. At the rendering time, the bitstream segments of the first quality layer can be delivered first, and then those of the second and third quality layer. With this strategy, a coarse quality view can be quickly rendered, and be refined as more and more bitstream segments arrive. The user movement is currently restricted within a planar circle. To move freely in a large environment, several concentric mosaics can be concatenated [23].

8. ACKNOWLEDGEMENTS

The authors would like to acknowledge the following individuals: Harry Shum and Hong-Hui Sun for the raw concentric mosaic data, the source codes of the SVQ concentric mosaic viewer, and many helpful discussions; Chunhui Hu and Jianping Zhou for the software support; Howard Leung for proofreading the paper.

9. REFERENCE

[1] R. Carey, G. Bell, and C. Marrin, "The Virtual Reality Modeling Language". Apr. 1997, *ISO/IEC DIS 14772-1*. [Online]: http://www.web3d.org/Specifications/.

[2] M. Deering, "Geometry compression", *Computer Graphics (Proc. SIGGRAPH'95)*, pp. 13-20, 1995.

[3] H. Hoppe, "Progressive Meshes", *Computer Graphics (Proc. SIGGRAPH'96)*, pp. 99-108, 1996.

[4] G. Taubin, W. P. Horn, F. Lazarus, and J. Rossignac, "Geometry Coding and VRML", *Proceedings of the IEEE*, pp. 1228-1243, Vol. 86, No.6, Jun. 1998.

[5] J. K. Li and C.-C. J. Kuo,, "Progressive Coding of 3D Graphics Models", *Proceedings of the IEEE*, Vol. 86, No. 6, June 1998.

[6] A. Khodakovsky, P. Schröder, and W. Sweldens, "Progressive Geometry Compression", *Computer Graphics (Proc. SIGGRAPH'2000)*, 2000.

[7] S. N. Matsuba and B. Roehl, "Bottom, Thou Art Translated: The Making of VRML Dream", *IEEE Computer Graphics and Applications*, pp.45-51, March/April, 1999.

[8] A. Lippman, "Movie Maps: An Application of the Optical Videodisc to Computer Graphics", *Computer Graphics (Proc. SIGGRAPH'80)*, pp. 32-43, 1980.

[9] D. G. Ripley, "DVI-a Digital Multimedia Technology", *Communications of the ACM*. 32(7): 811-822, 1989.

[10] G. Miller, E. Hoffert, S. E. Chen, E. Patterson, D. Blackketter, S. Rubin, S. A. Aplin, D. Yim, J. Hanan, "The Virtual Museum: Interactive 3D Navigation of a Multimedia Database", *The Journal of Visualization and Computer Animation*, (3): 183-197, 1992.

[11] 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.

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

[13] S. B. Kang, "A Survey of Image-based Rendering Techniques", *SPIE International Symposium on Electronic Imaging: Science and Technology*, Vol. 3641, pp. 2-16, San Jose, CA, 23-29 Jan. 1999.

[14] S. E. Chen, "QuickTime VR – An Image-Based Approach to Virtual Environment Navigation", *Computer Graphics (Proc. SIGGRAPH'95)*, PP29-38, 1995.

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

[16] S. J. Gortler, R. Grzeszczuk, R. Szeliski and M. F. Cohen, "The Lumigraph", *Computer Graphics (SIGGRAPH'96)*, pp. 43-54, Aug.1996.

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

[18] L. Luo, Y. Wu, J. Li and Y. Zhang, “3D wavelet compression and progressive inverse wavelet synthesis rendering”, *IEEE Trans. On Image Processing*, Vol. 11, No. 7, pp. 802-816, Jul. 2002..

[19] Y. Wu, C. Zhang and J. Li, "Smart rebinning for the compression of concentric mosaic", IEEE Trans. on Multimedia, vol. 4, no. 3, pp. 332-342, Sept. 2002.

[20] M. Magnor and B. Girod, “Hierarchical coding of light fields with displarity maps”, *Proc. International Conference on Image Processing (ICIP-99)*, Kobe, Japan, pp. 334-338, Oct. 1999.

[21] C. Zhang and J. Li, “Compression of Lumigraph with multiple reference frame (MRF) prediction and just-in-time rendering”, *IEEE Data Compression Conference (DCC’2000)*, Snowbird, Utah, Mar. 2000.

[22] J. Li and H. Sun, “A virtual media (Vmedia) access protocol and its application in interactive image browsing”, *SPIE Multimedia Computing and Networking 2001 (MMCN'01)*, San Jose, CA, Jan. 2001.

[23] M. Wu, H. Shum, J. Li, G. Xu, Y. Li, C. Zhang, T. Nakayama, and Y. Zhang, “Wandering in large environments using concatenated concentric mosaics”, *submitted to IEEE Trans. On Multimedia*.

Figure 1: Capturing device of the concentric mosaic.

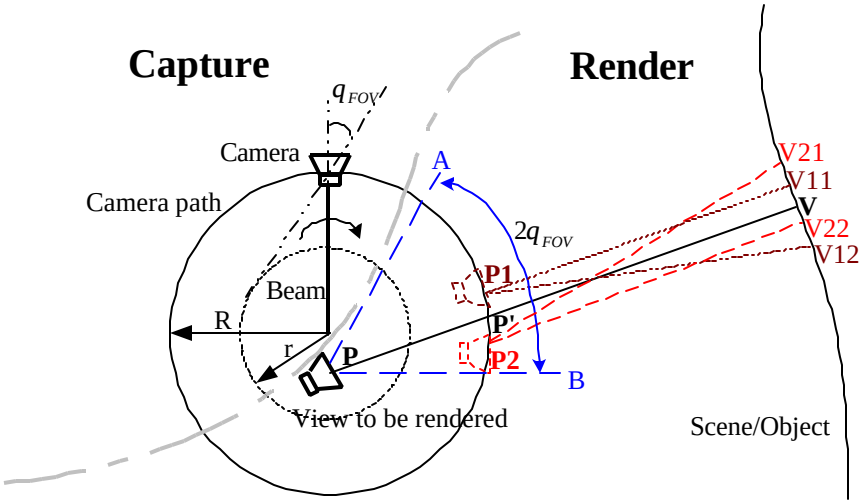

Figure 2: Concentric mosaic capturing and rendering.

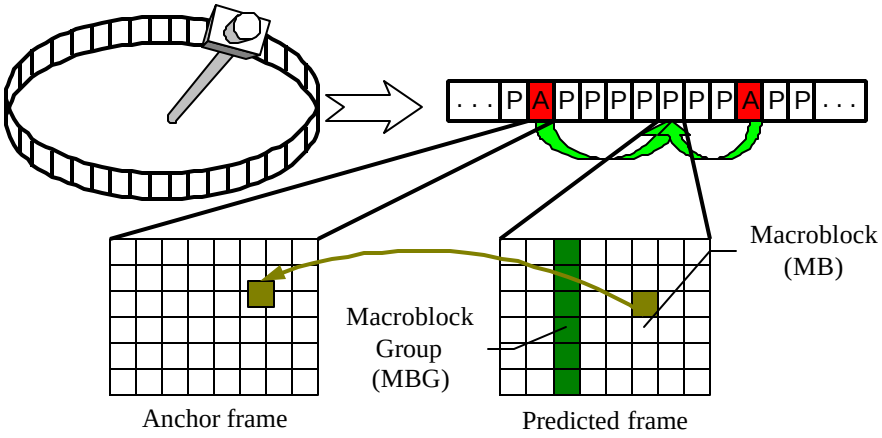

Figure 3: The data structure of the reference block coder (RBC).

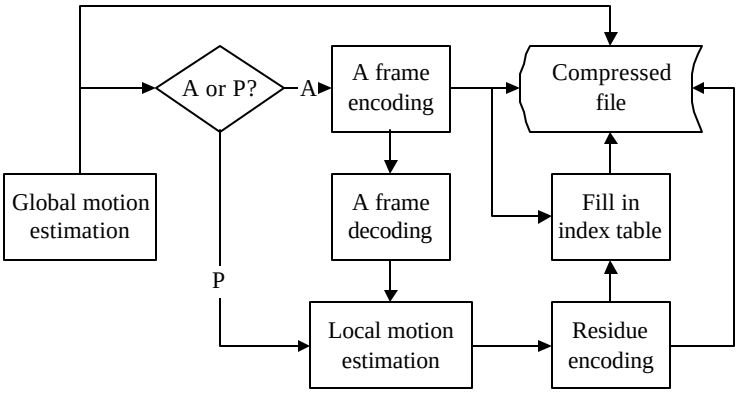

Figure 4: Flowchart of the RBC encoder.

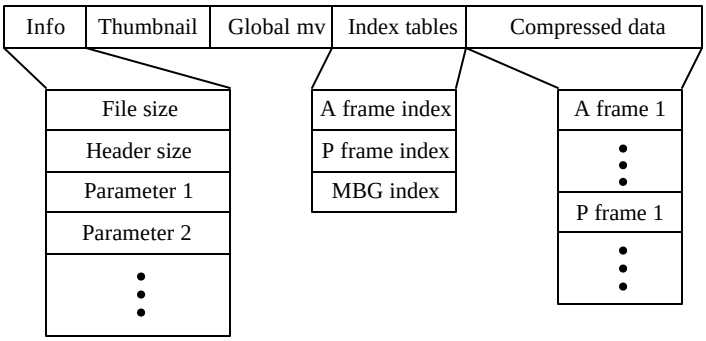

Figure 5: Syntax of the compressed RBC file.

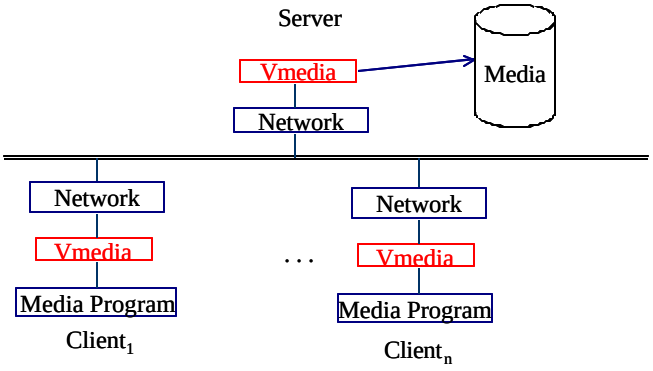

Figure 6: Vmedia framework.

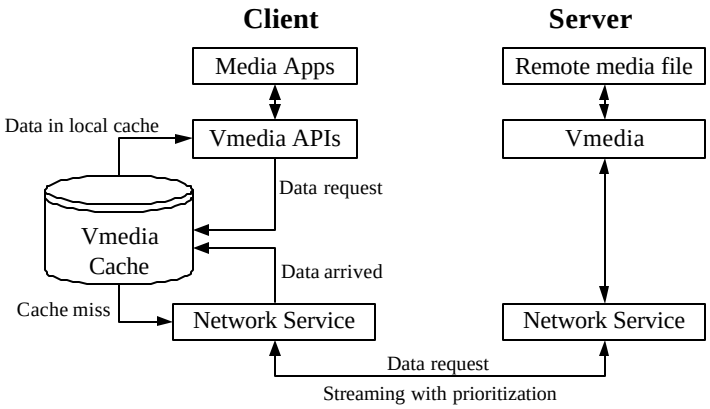

Figure 7: Workflow of a Vmedia application.

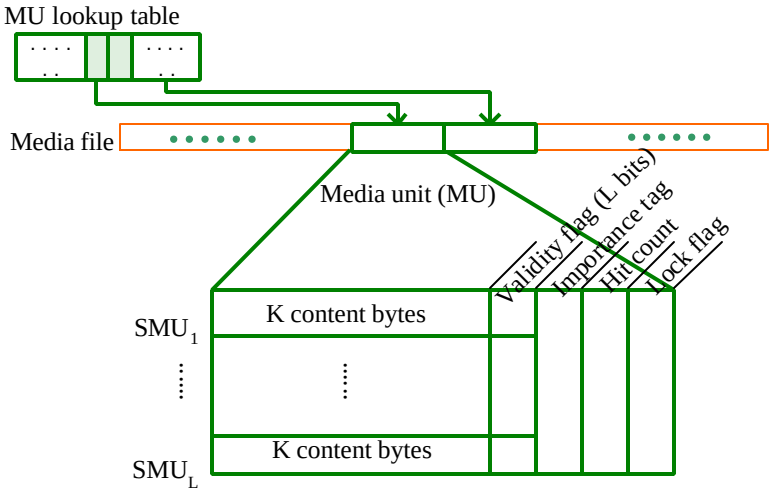

Figure 8: Media unit (MU) and sub media unit (SMU).

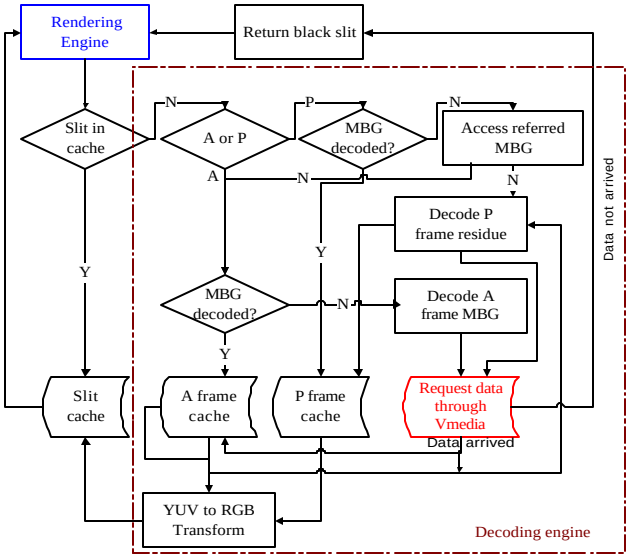

Figure 9: Rendering flow of the client.

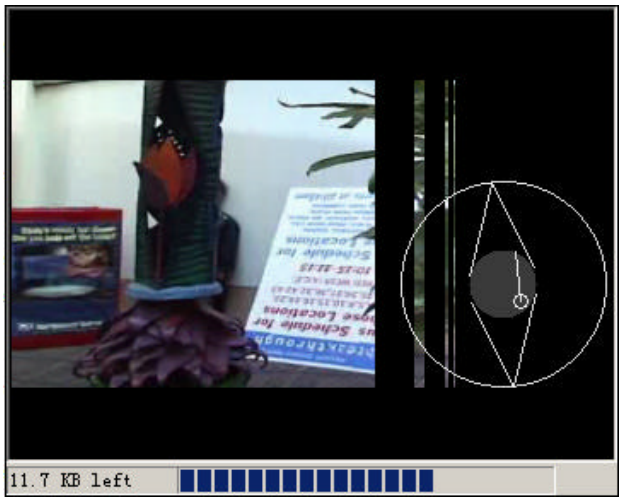

Figure 10: An intermediate rendered concentric mosaic view. (The unavailable slits are rendered as black.)

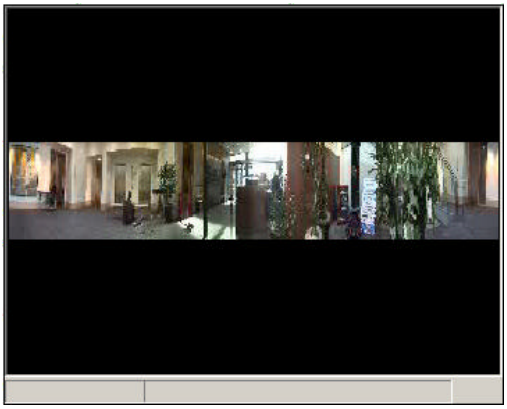

Figure 11: Thumbnail of the concentric mosaics.

Figure 12: Concentric mosaic scene *Lobby*.

Figure 13: Concentric mosaic scene *Kids*.

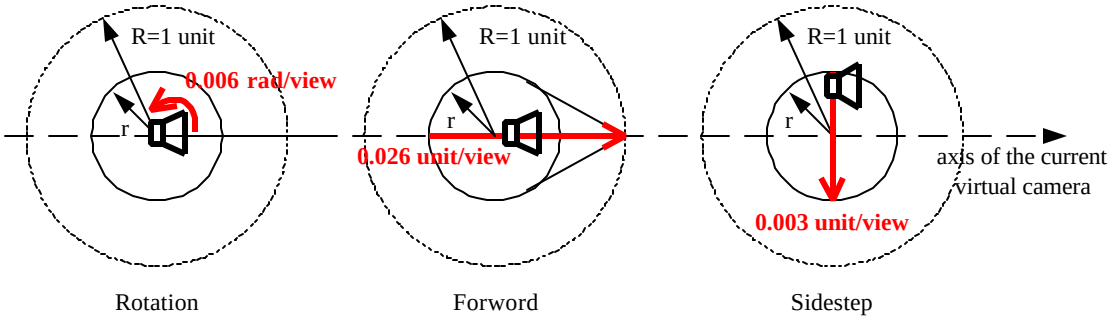

Figure 14: Three motion modes: rotation, forward, and sidestep.

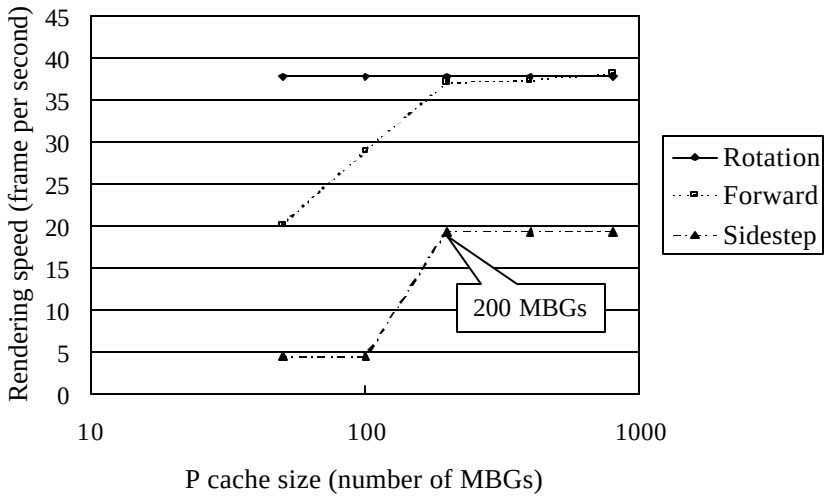

Figure 15: RBC rendering speed vs. P cache size. (Bilinear interpolation mode, 600 MBG A cache)

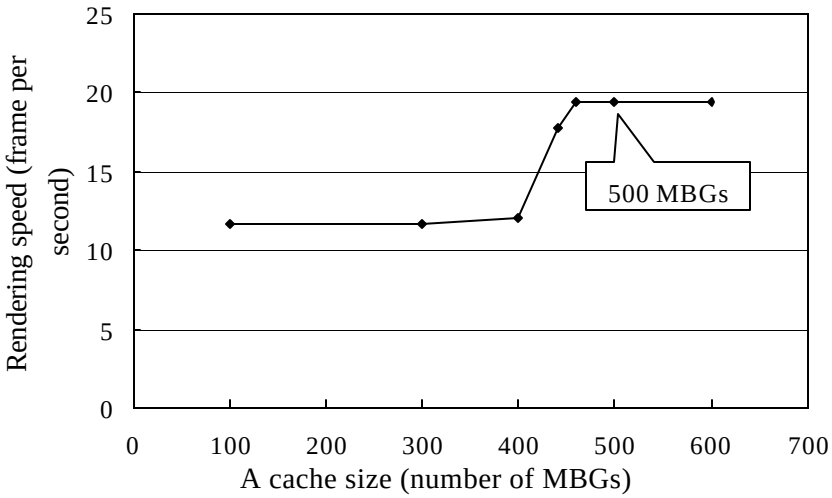

Figure 16: RBC rendering speed vs. A cache size. (Bilinear interpolation mode, sidestep motion, 200 MBG P cache)

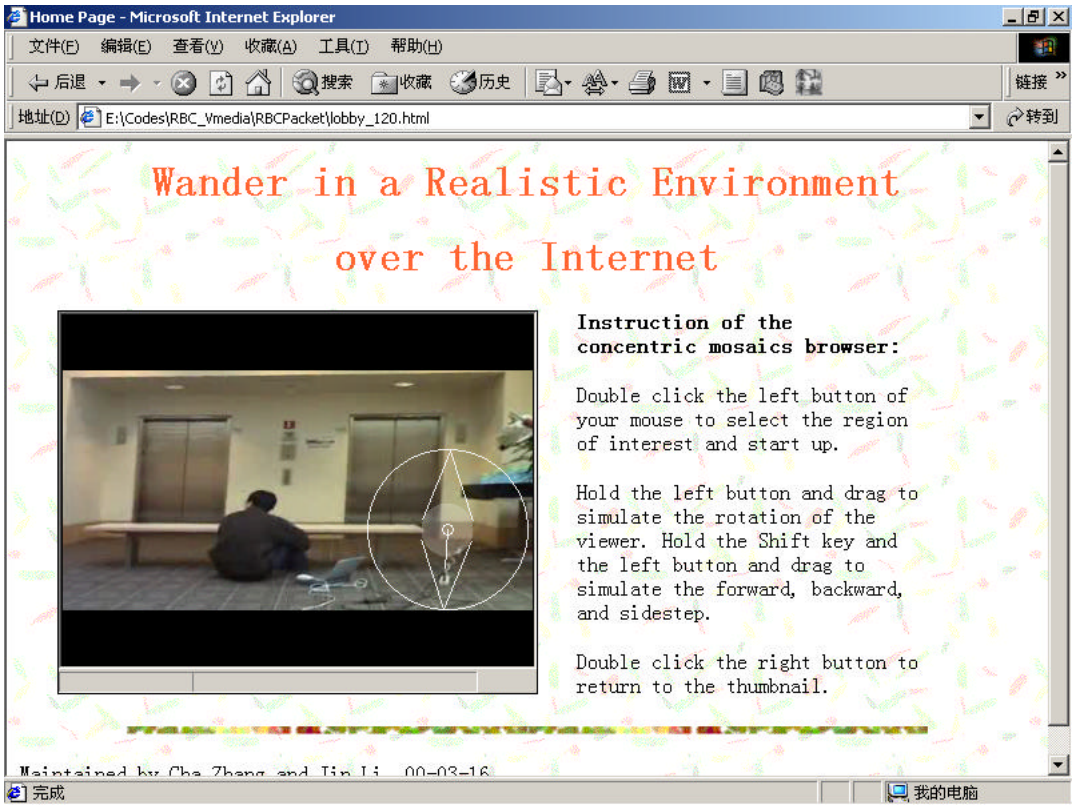

Figure 17: RBC ActiveX plug-in embedded in a web page.

Figure 18 Rendered view of the *Kids* scene with the compression ratio of 40:1 (upper) and 100:1 (lower).

Table 1. Compression performance (PSNR) of RBC, and MPEG-2 and the 3D wavelet approach (dB).

Approach \ Scene	*Lobby*		*Kids*		
	(0.2bpp)	(0.4bpp)	(0.24bpp)	(0.4bpp)	(0.6bpp)
MPEG-2	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	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
3D Wavelet	Y: 31.9 U: 40.3 V: 39.9	-	-	Y: 29.4 U: 36.5 V: 37.2	-

Table 2. Experimental conditions for comparing the rendering speed.

	Compression ratio	Cache size	Rendering complexity
SVQ	12:1	37.0 MB	Low
RGC	60:1	13.8 MB	Medium
3D Wavelet	60:1	40+ MB	High

Table 3. Rendering speed of RBC, SVQ and the 3D wavelet approach. (frame per second)

Mode / Algorithm \ Rendering setting		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
	3D Wavelet	16.9	13.6	-	-
Forward	SVQ	17.0	14.2	71.4	45.5
	RBC	15.6	13.1	49.6	36.7
	3D Wavelet	15.0	12.7	-	-
Sidestep	SVQ	15.8	13.2	53.4	37.1
	RBC	11.4	9.9	23.0	19.3
	3D Wavelet	7.2	6.6	-	-