SPIE Visual Communication and Image Processing 2001 (VCIP 2001), SPIE Vol. 4310, No. 51, San Jose, CA, Jan 2001.

Interactive Browsing of 3D Environment over the Internet

Cha Zhang[†*] and Jin Li [‡]

[†]Dept. of Electrical and Computer Engineering, Carnegie Mellon University, Pittsburgh, PA 15213.
[‡]Microsoft Research China, 49 Zhichun Road, Haidian, Beijing 100080, China
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. Capturing the environment through the concentric mosaic is easy. We mount a camera at the end of a level beam, and shoot images as the beam rotates. The huge dataset of the concentric mosaic is then compressed through the RBC, which is specifically designed for both high compression efficiency and just-in-time (JIT) rendering. Through the JIT rendering function, only a portion of the RBC bitstream is accessed, decoded and rendered for each virtual view. A multimedia communication protocol – the Vmedia protocol, is then proposed to deliver the compressed concentric mosaic data over the Internet. Only the bitstream segments corresponding to the current view are streamed over the Internet. Moreover, the delivered bitstream segments are managed by a local Vmedia cache so that frequently used bitstream segments need not be streamed over the Internet repeatedly, and the Vmedia is able to handle a RBC bitstream larger than its memory capacity. A Vmedia concentric mosaic interactive browser is developed where the user can freely wander in a realistic environment, e.g., rotate around, walk forward/backward and sidestep, even under a tight bandwidth of 33.6 kbps.

Keywords: virtual reality, 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 experience on the Internet, where people can now listen to news/music, watch TV, play network game, 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 stationed photos, the experience of free wandering in a 3D environment gives a more realistic feel of the environment and is very useful in applications such as real-estate sale on-line, virtual reality/chat room, Internet game, e-commerce, virtual museum, etc.

Traditionally, a virtual environment is modeled as a collection of 3D geometrical entities, 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 the hardware today well and can be efficiently transferred and rendered, it is difficult to create the polygonal model for a complex real environment. Even the model can be created, it can easily reach millions of polygons, which causes problem in 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 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 shot and quick to render. However, they require every displayable view/path to be captured and stored in the authoring stage, which restricts the freedom of the movement and the interaction between the user and application. Moreover, switch at the branch point is jerky, as the two movie segments are usually shot at different times, with slightly different camera setup. A large amount of storage space is also required to store all the movie segments.

Another major thread of IBR approach 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,

---

[*] 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.

McMillan and Bishop [12] defined plenoptic modeling as 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 describe very complex scene with no geometrical knowledge of the scene required for the rendering.

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 compose 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. Currently, the QuickTime VR system is the most widely available virtual reality representation on the Internet. However, the QuickTime VR limits the movement of the user to a few particular points. Smooth wandering in the environment is impossible unless the capturing points are very dense, which causes a huge increase in the captured data amount.

The Lightfield [15] and the Lumigraph [16] are the first practical approaches to interpolate novel views from dense sampled photos. They provide 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] propose the concentric mosaic, which can be considered as a 3D plenoptic function. One attractive advantage of the concentric mosaic is that a representation of the realistic environment can be constructed 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 coinciding with the rotating plane of the beam, which is the usual movement of the user as he/she wanders around in the environment. Rendering virtual views in a scene represented by the concentric mosaic is straightforward. The view is split into vertical ray slits, and each slit is reconstructed through similar slits captured during the rotation of the camera. 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 [26]. Since the view is assembled from real photos, the sense of reality augmented by complex parallax and the lighting change is unparallel compared with a geometric based representation.

Representing a complex environment is hard, and delivering the experience of wandering in a complex environment over the Internet is harder. A realistic scene representation involves huge amount of data, which is slow to be delivered over the Internet. MPEG-4 has developed compression algorithms to compress the polygonal models and deliver them over the Internet. Works have 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 data of an IBR representation is even greater. As an example, a concentric mosaic scene Lobby (shown in Figure 10) consists of 1350 frames at resolution 320x240 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. Carefully handling the above two issues, we bring the experience of wandering in a real environment constructed by the concentric mosaic over the Internet. First, a concentric mosaic compression algorithm, the reference block coder (RBC), is developed to reduce the data amount in the concentric mosaic. RBC is not only highly efficient in compression performance, but also 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 compressed bitstream of the image shot, and the second level indexes macroblock groups (MBG) within the image shot. 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, for a specific virtual view of the concentric mosaic, only portion of the compressed RBC bitstream needs to be accessed and decoded, 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.

In order to deliver the bitstream segments associated with JIT rendering of the current view efficiently, and to cache and manage the delivered bitstream segments so that the same segment is not delivered repeatedly over the Internet, a new media delivery protocol termed the virtual media (Vmedia) access protocol is developed. With the RBC and Vmedia, we build up

an efficient 3D web-browser system. Caches are implemented so that the required virtual view of the environment can be decoded and rendered in real-time with a PC today. We are able to freely wander in a realistic environment constructed by the concentric mosaic even when the connection speed of the network is as slow as 33.6kbps.

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. A short conclusion is given in Section 7.

2. THE CONCENTRIC MOSAIC

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 alias. This leads to a dataset of several hundreds of 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, θ_{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 $r = R\sin\theta_{FOV}$. 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 ray does not change along a straight line unless blocked. Therefore, what is rendered at PV can be recovered from that is observed in $P'V$. Since the concentric mosaic consists of a finite number of shots, and each shot consists of a group of vertical slits, there may not be an exact slit $P'V$ 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 constraint, 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 referred 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 slits referred in the view are available.

There have been works concerning the compression of the concentric mosaic. In [17], the concentric mosaic is split into small blocks, which is compressed by spatial domain vector quantization (SVQ). The compressed bitstream consists of a large SVQ codebook, and VQ indexes for each block of the concentric mosaic. SVQ is simple and fast to decode, and since the SVQ indexes are of fixed length, they can be flexibly accessed. However, SVQ is time-consuming at the encoding stage, and the compression ratio of SVQ is limited, e.g., at 12:1 in [17]. Moreover, 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 very high compression ratio, and may choose portion of the compressed bitstream to access with selected resolution and quality level [20]. However, 3D wavelet coding and rendering system is more complex to implement and more computational expensive.

According to the survey of the authors, although works have been done to compress the Lightfield or Lumigraph in support of the Internet streaming [21][22], there is no existing system on delivering compressed image based rendering scene over the Internet.

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 the bitstream to be easily indexed and delivered over the network.

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 of access for the decoding operation. We encode the A frames independently, while predictively encode the P frames 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 number of A frames is, 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.

The A and P frame are further segmented into square blocks of size 16x16. We call it a macroblock (MB), for its similarity with the macroblock used in MPEG. 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), which 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, while those in the P frames are predictively encoded with respect to a nearby A frame. We restrict the MBs in the P frame to refer to one of the nearby A frame, which 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 MBs in the A frame and the motion compensated MBs in the P frame are then DCT transformed, quantized and Huffman encoded. We refer to [23] for details of the RBC algorithm.

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, RBC bitstream is lead 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 anchor 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 vector, 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 vector and the two-level hierarchical index table form the file header of the compressed RBC bitstream. The size of the file header is trivial, 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 A frames. The A frames are compressed and decompressed before any P frame coding, because the decoder can only access the decompressed A frames, not the original ones. The P frame bitstream follows. Typically, it costs more bits to encode an A frame than a P frame, because the entropy of the A frame is much higher than that of the residue of the P frame.

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 vector, etc.

The RBC coding scheme bears a strong resemblance to the 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 mosaic. 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. The two-level hierarchical index table is also unique for RBC. These modifications improve the compression performance of RBC and enable the RBC compressed bitstream to be randomly accessed in the rendering stage.

4. THE VIRTUAL MEDIA (VMEDIA) ACCESS PROTOCOL

When a RBC compressed concentric mosaic is browsed over the Internet and a virtual view of the concentric mosaic is rendered, only a small portion of the compressed bitstream segment is accessed. Such set of bitstream segments need to be delivered efficiently over the Internet. Rendering a new view of the concentric mosaic requires a new set of bitstream segments. Since there are reusable bitstream segments between the old and the new view, it is essential to cache and manage the downloaded bitstream segments so that the same bitstream segment is not downloaded repeatedly over the Internet.

We deliver the bitstream of the RBC compressed concentric mosaic through a virtual media (Vmedia) access protocol we have developed. Vmedia is a multimedia communication protocol in a server-client environment. Rather than treating the entire media file as a whole, or streaming the media file sequentially, Vmedia enables flexible accessing to the media, and streaming just the bitstream related to the current region of interest, namely the compressed bitstream of the current view.

Though used specifically for the RBC compressed concentric mosaic, Vmedia is useful for many other Internet browsing applications [24]. The framework of Vmedia can be shown in Figure 6.

There are a Vmedia server and a Vmedia client component. Application calls the Vmedia client to access the remote media. Upon connection, the Vmedia client mirrors 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 Explore™ and Netscape™, there is no implement for segment cache.

3. Prioritized delivery

High priority segment, such as the A frame MBG, can be delivered first over the Internet to improve the response time.

4. Media segment packaging

Small media segments are automatically packaged into a larger one, and a large media segment is broken into several 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 media application simply accesses remote media as a virtual local file, and issues media segment access requests with priority and importance. We explain the typical workflow of Vmedia in Section 4.1. The two key components of the Vmedia, which are the Vmedia cache and the media segment delivery, are explained in Section 4.2 and 4.3, respectively.

4.1. The workflow

A typical application flow using Vmedia is shown in Figure 6. The application 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 is downloaded to the Vmedia client to aid the access of segments of the media. The Vmedia cache is also initialized at the connection phase. After that, the Vmedia client has all the information for positioning the bitstream segments, and is ready to send out access request issued by the application. The access request is attached with a priority tag, an importance tag and an optional lock, which are used to prioritize streaming and aid cache management. For each access request, 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 calling application. If only part of the segment is in the cache, a continuous head portion is returned. If none of the segment is in the cache, no information is returned. In the latter two cases, a pending network request is also generated and sent to the Vmedia server to download the missing segments. The request is not sent immediately over the network because the requests need to be packetized and prioritized for delivery. A synchronization function call is issued by the application to inform the Vmedia that there are no more pending requests, so that pending requests in queue can be sorted and sent to the Vmedia server.

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 application regardless of the outcome of the request. To best utilize the Vmedia protocol, the application should also render the media on a best effort basis. A usual strategy is that the application repeatedly queries and renders the ROI using whatever data currently available. A partial/coarse 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.

When the application closes the Vmedia connection, the current content of the Vmedia cache can be optionally dumped as a permanent file to the disk, so that the next time the same remote media is accessed, there is no need to download the media segments that have already been received in the last session. The functionality is similar to the temporary Internet files implemented in the Internet Explorer™ and Netscape™, though instead of an entire file, segments of a file are dumped.

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 used and less important media segments need to be thrown out.

There are several approaches to implement the Vmedia cache. We may allocate a continuous memory of the size of the remote media, and use validity tags to indicate which part of the media is received. However, such an approach is very memory consuming and cannot browse media larger than the memory capacity of the client. Another approach is to allocate media segment on demand, with arbitrary length and start point. The approach may severely fragment the memory. It is also slow to detect whether a particular segment is already in the Vmedia cache.

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. An arbitrary positioned media segment is round up into a set of sub media unit (SMU), which is the minimum size for delivery and cache management. SMU has a fixed size of K bytes and starts at K byte boundaries. L SMUs form a media unit (MU), which is the unit for memory allocation. In the current implementation, K=L=32. The MU and SMU structure can be shown in Figure 7. MU maintains tags for cache management; while SMU only carries a validity bit. All MUs are indexed by a lookup table. The memory overhead introduced by the MU/SMU is only 10 bytes per 1024 byte MU, which is about 1%.

With the MU/SMU structure, cache hit detection can be performed very quickly. The media segment is first broken into SMUs. The validity of the MU containing the SMU can be checked by the table lookup. A bit-wise AND operation can then be used to further validate the accessed SMUs. The cache can also be easily updated. Whenever a new media segment is received, we check if the MU associated with the media segment is allocated. If it is, the media segment is stored in the memory, and the validity bits of the associated SMUs are turned on. 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 and less important MUs are thrown away.

As shown in Figure 7, each MU maintains one importance, one hit count and one lock tag. There is also a validity bit for each SMU. The importance tag is provided by the application, with value 255 for the most important, and value 0 for the least important. A highly important MU is less likely to be thrown away in case that the cache memory is used up. A MU may be assigned with several importance values through different function calls. In such a case, the highest importance value among all calls is assigned to the MU. The MUs may also be temporarily locked to ensure that certain media segments are not thrown away. The lock is frequently used on the media segments associated with the current ROI to ensure newly arrived bitstream segments of the current view is not accidentally swapped out. In case that the current view changes, an unlock function is called to release all locks placed on the Vmedia.

The validity tag indicates whether a specific SMU of the Vmedia is accessible. The hit count tag records how many times a MU has been visited. Unlike the importance value and the lock, which are provided by the application, both validity and hit count tags are maintained internally. In case the cache memory is used up, the MU with the smallest importance plus a weighted hit count is thrown away to make room for the newly arrived media segments.

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. The media segment with a higher priority value is delivered earlier over the network. The highest priority value is 255 and the lowest is 0. Vmedia maintains two categories of queues: pending queue where the request is issued by the application but have not been sent over the network yet, and the sent queue where the request is sent over the network but the requested media segment has not been received. The pending queue consists of a list of queues with each queue holding media segments of different priority. Once a pending media segment request is generated due to a cache miss, it is checked whether the corresponding media segment is already in queue scheduled for delivery. It will not be necessary to issue the same media request twice, though the priority of the request may be adjusted to the highest priority issued. Once a media segment request is sent to the server, the request is bound with a timer and moved to a sent queue. Whenever a media segment is received from the server, the Vmedia clears the request in the sent queue, and stores the received media segment in cache. Vmedia may or may not report the arrival of the media segment. A good strategy is to report only when a certain number of media segments have arrived, or all requested media segments above a certain priority level have arrived. If a sent request is not fulfilled for a long time, i.e., the associated timer of the request runs out, it will be reinserted into the pending queue. However, instead of appending to the tail of the queue as a cache miss generated request, the time out request is inserted into the head of the pending queue. Similarly, if an error or a packet loss is detected, the corrupted/lost media segment is also reinserted into the head of the pending queue for redelivery.

In the event of ROI change, namely a radically different set of media segments are accessed, a clear function call can be issued to eliminate all requests in the pending and sent queue. The Vmedia client also contacts the Vmedia server to cancel all 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 content associated with the new ROI. Requests that are already in route to the client are still processed by the Vmedia client and stored in the Vmedia cache when they arrive.

Vmedia may access a normal HTTP server, where request is delivered as a partial GET HTTP request. In such a case, the accessed media segment is returned by the HTTP server. Alternatively, a special Vmedia server can be specifically designed to handle the Vmedia request, and to reduce the network overhead associated with the request and returned media segment. UDP protocol is used to deliver the request and the media segment. Considering the overhead of the UDP and IP header, multiple requests are bundled into a single UDP packet and sent to the server. Similarly, the server also bundles multiple small media segments into a single UDP packet and sends it back to the Vmedia client. The Vmedia server also breaks down a media segment larger than the maximum transmission unit (MTU) into multiple UDP packets, and then sends them back separately to the client.

The functionality of the Vmedia server is rather simple: it handles media access request coming from the Vmedia client. Media segment 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 associated with any type of media, e.g., interactive image browsing [24]. 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 8. 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 space), the A and P frame caches which hold the A and P frame MBGs (in YUV 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 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 referred A frame MBGs are needed. The referred 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 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 stayed longer in cache.

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 with a best effort based on 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 9, where the slits in the right part of a view is unavailable and is 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.

The slit, A and P frame caches are managed with a least recent used (LRU) cache management strategy. A doublelink 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. 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.

6. SIMULATION RESULTS AND SYSTEM PERFORMANCE

To demonstrate the effectiveness of the reference block coder (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 mosaic. 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 10 and Figure 11, respectively. The scene Lobby is shot with 1350 frames at resolution 320x240, and the scene Kids is shot with 1462 frames at resolution 352x288. The Kids scene has more details, and is much more difficult to compress than the Lobby scene.

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:

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

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 Q_a and Q_p are constants throughout the compression. While in MPEG-2, 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. In term of the Y PSNR which is the most important in subjective evaluation, RBC beats MPEG-2 for 0.4 to 1.7dB, 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, the gain is significant.

In the second experiment, we investigate the rendering speed of RBC. 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 spatial domain vector quantization (SVQ) used in [17] with a compression ratio 12:1. We simply replace the SVQ decoder of [17] by the RBC decoder. The test scene is the Kids scene, with original data size 424 megabytes. The compression ratio of RBC is 60:1, i.e., five times that of 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 [23]. 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 entire memory cache of the RBC render is 13.8 megabytes versus the 37.0 megabytes of SVQ.

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 12. 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 vertical distortion [17].

Due to higher complexity in decoding, the rendering speed of RBC is a litter 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, with about one third of cache memory requirement of the renderer. The rendered concentric mosaic walkthrough looks comfortable and smooth under RBC. We have profiled the RBC coder, and the four most time-consuming components of the RBC decoder and render are the motion compensation, inverse DCT, YUV to RGB color transform and rendering. The first 3 components can be greatly accelerated if MMX instruction sets are used.

An environmental browser embedded in a web page showing the concentric mosaic scene Lobby is illustrated in Figure 13. The scene is RBC compressed at a 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 Internet connection speed is set at 33.6 kbps (kilobits per second).

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. 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 can be rendered, as they are indispensable for the accessing and decoding of 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. In case of the slit cache miss, the MBGs in the corresponding A or P frame are retrieved, which in turn accesses bitstream segments from the Vmedia API. Only the bitstream segments necessary to decode the slits used in the current view are accessed and decoded. This greatly improves the response time that 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. Under the network bandwidth of 33.6 kbps, the entire view can be rendered in 7 to 25 seconds. To render a view during the wandering, i.e., rotate, walk forward/backward and sidestep, the unavailable bitstream segments is only about half that amount, i.e., 15 to 50 kilobytes. The rest bitstream segments are usually associated with prior views and are already stored in the Vmedia cache. A view can usually be completely rendered within 5 seconds. Though far fewer slits are accessed in the point sampling mode, the requested data amount is 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 media segments. Currently, we render the unavailable slits as black. Shown in Figure 9, a progress bar is displayed to show the amount of bitstream segments to be streamed for the current view.

In an alternative experimental setting, we compress the Lobby scene at a compression ratio of 200:1. The compressed bitstream reduces 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 end 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 WORKS

In this paper, we develop a system that enables a user to wander in a realistic environment over the Internet. The environment is captured by the concentric mosaic, compressed via the reference block coder (RBC), accessed and delivered over the Internet through the virtual media (Vmedia) access protocol. For the first time, we enable the user to wander in a realistic environment through the Internet, even when the network bandwidth is as low as 33.6 kbps.

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 such a strategy, a coarse quality view can be quickly rendered, and be refined as more and more bitstream segments arrive. Technologies developed in video error concealment [25] can also be employed to render the unavailable slit by interpolating from the available neighborhood slits. The user movement is currently restricted within a planar circle. To move freely in a large environment, several concentric mosaics can be concatenated[26].

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.

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.-Q. 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.
[19] Y. Wu, C. Zhang, J. Li, and J. Xu, "Smart-rebinning for Compression of Concentric Mosaics", *accepted by ACM multimedia 2000*, Los Angeles, CA, Oct, 2000.
[20] Y. Wu, L. Luo, J. Li and Y.-Q. Zhang, "Rendering of 3D Wavelet Compressed Concentric Mosaic Scenery with Progressive Inverse Wavelet Synthesis (PIWS)", *SPIE: Visual Communication and Image Processing (VCIP 2000)*, Perth, Australia, Jun. 2000.
[21] 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.
[22] 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.
[23] C. Zhang and J. Li, "Compression and rendering of concentric mosaic scenery with reference block codec (RBC)", *SPIE Visual Communication and Image Processing (VCIP 2000)*, Perth, Australia, Jun. 2000.
[24] J. Li and H. Sun, "A virtual media (Vmedia) access protocol and its application in interactive image browsing", to appear in *SPIE Multimedia Computing and Networking 2001 (MMCN'01)*, San Jose, CA, Jan. 2001.
[25] Y. Wang, Q. F. Zhu, "Error Control and Concealment for Video Communication: A Review", *Proceedings of the IEEE*, pp. 974–997, Vol. 86, No. 5, May 1998.
[26] 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*.

Table 1. RBC versus MPEG-2 (performance)

Scene / Approach	*Lobby*		*Kids*		
	(0.2bpp)	(0.4bpp)	(0.24bpp)	(0.4bpp)	(0.6bpp)
MPEG-2 (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

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

Figure 1: Capturing device of the concentric mosaic.

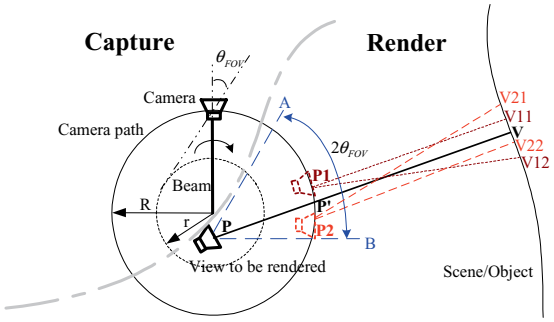

Figure 2: Concentric mosaic capturing and rendering.

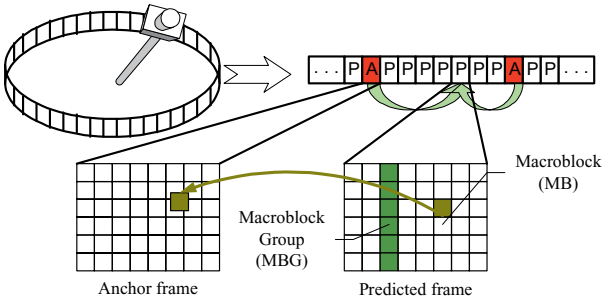

Figure 3: The data structure of the reference block coder (RBC).

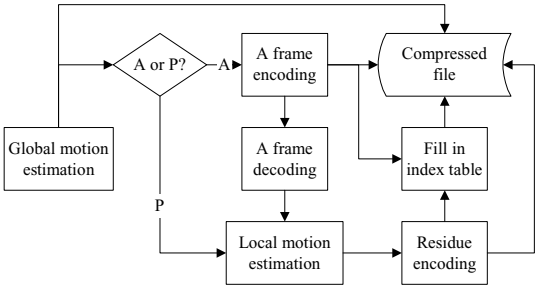

Figure 4: Flowchart of the RBC encoder.

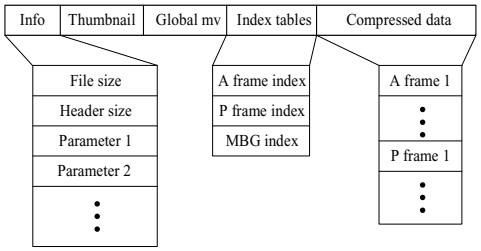

Figure 5: The syntax of the compressed RBC file.

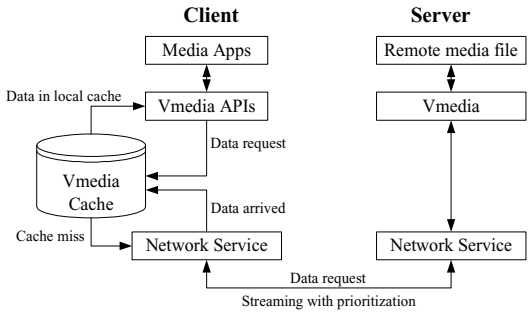

Figure 6: The workflow of a Vmedia application.

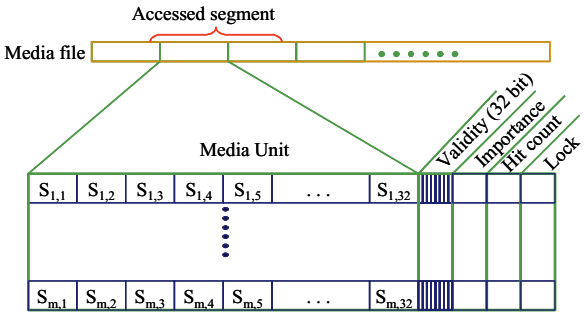

Figure 7: Media unit (MU) and sub media unit (SMU).

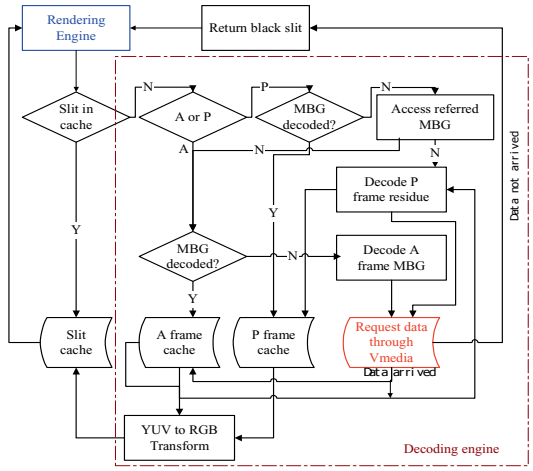

Figure 8: Rendering flow of the client.

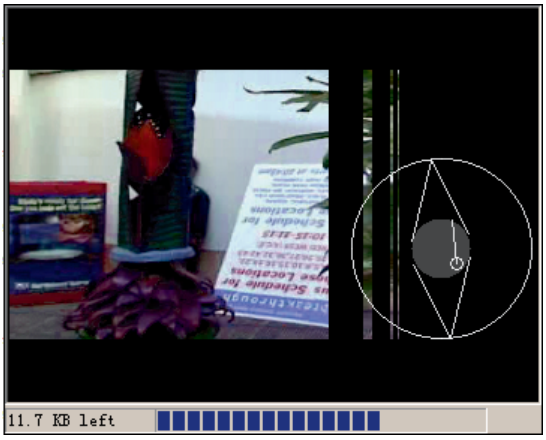

Figure 9: An intermediate rendered concentric mosaic view. (unavailable slits rendered as black)

Figure 10: Concentric mosaic scene *Lobby*.

Figure 11: Concentric mosaic scene *Kids*.

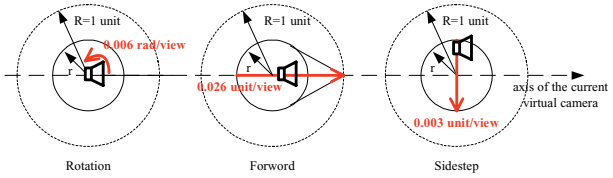

Figure 12: Three kinds of motion mode: rotation, forward, and sidestep.

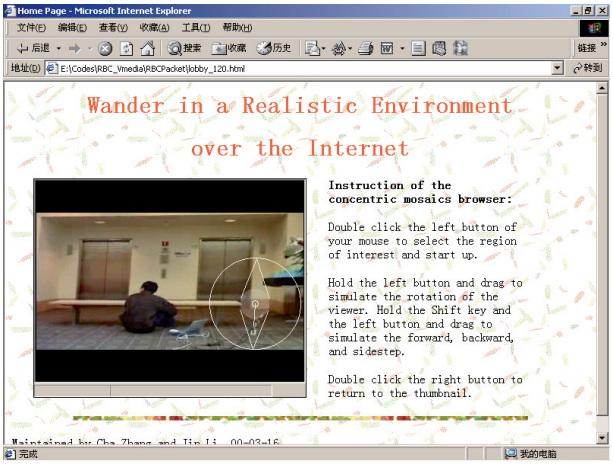

Figure 13: RBC ActiveX plug-in embedded in a web