Data Compression and Transmission Aspects of Panoramic Videos

King-To Ng, *Member, IEEE*, Shing-Chow Chan, *Member, IEEE*, and Heung-Yeung Shum, *Senior Member, IEEE*

***Abstract*—Panoramic videos are effective means for representing static or dynamic scenes along predefined paths. They allow users to change their viewpoints interactively at points in time or space defined by the paths. High-resolution panoramic videos, while desirable, consume a significant amount of storage and bandwidth for transmission. They also make real-time decoding computationally very intensive. This paper proposes efficient data compression and transmission techniques for panoramic videos. A high-performance MPEG-2-like compression algorithm, which takes into account the random access requirements and the redundancies of panoramic videos, is proposed. The transmission aspects of panoramic videos over cable networks, local area networks (LANs), and the Internet are also discussed. In particular, an efficient advanced delivery sharing scheme (ADSS) for reducing repeated transmission and retrieval of frequently requested video segments is introduced. This protocol was verified by constructing an experimental VOD system consisting of a video server and eight Pentium 4 computers. Using the synthetic panoramic video *Village* at a rate of 197 kb/s and 7 f/s, nearly two-thirds of the memory access and transmission bandwidth of the video server were saved under normal network traffic.**

***Index Terms*—Image-based rendering, panoramic video, video coding and transmission, video-on-demand (VOD), virtual reality.**

I. Introduction

IMAGES and videos are effective means to represent objects and scenes. With increasing demand for better user experience in interactive applications such as virtual walkthrough, computer games, and medical simulation, virtual reality techniques have become increasingly more important. Image-based rendering (IBR) using the plenoptic function [1] has recently emerged as a simple yet powerful photo-realistic representation of real-world scenes [2]–[6]. Its basic principle is to render new views of a scene using rays that were previously captured in densely sampled pictures of the scene. In its most general form, the plenoptic function is a seven-dimensional (7-D) function allowing one to reconstruct any novel view at any point in space and time. Due to the difficulties in capturing and storing the function, various simplifications have been advocated. By ignoring time and wavelength, the dimension of the plenoptic function can be reduced from seven to five [3]. Using the two planes parameterization in a free space, one can further simplify the plenoptic functions to four dimensions, leading to the four-dimensional (4-D) light field [4] and the 4-D lumigraph [5]. More recently, Shum and He [6] have proposed a new three-dimensional (3-D) plenoptic function representation, the concentric mosaic, by restricting the viewer movement inside a planar circle. The capturing and rendering of concentric mosaics are very simple due to the lower dimensionality. If the viewpoint is also fixed and only the viewing directions and camera zoom can be altered, the plenoptic function simply becomes the two-dimensional (2-D) panorama (cylindrical [2] or spherical [7]). Panoramas are relatively simple to construct by stitching together a set of images taken at different angles along a given axis. During rendering, part of the panoramic image is reprojected onto the screen to emulate the effect of panning and zooming.

Most image-based representations reported so far in the literature deal with static scenes. It is largely attributed to the logistical difficulties in capturing and transmitting dynamic representations, which involve huge amounts of data. This has stimulated considerable research interests in the efficient compression of various image-based representations such as light fields, lumigraphs, concentric mosaics and panoramas [4], [8]–[14], [41]. For example, vector quantization (VQ) has been used in [2], [4], [6] and it has the advantage of fast and simple decoding using table lookup operations. It also simplifies the important random access problem of image-based representations in concentric mosaics and light fields. However, VQ encoding is complex and time consuming, and its compression ratio is somewhat limited. In [6], a compression ratio of 12:1 has been reported. Since image-based representations are usually highly correlated, codecs using JPEG or MPEG-2-like algorithms have been proposed for the compression of light fields/lumigraphs [11]–[13], and concentric mosaics [8]–[10]. The MPEG-2-like algorithm in [10] achieves a very high compression ratio by exploring the redundancy in adjacent image frames of concentric mosaics. Moreover, the MPEG-2 algorithm can be modified to support random access of the compressed image sequence. Another approach that is more complicated includes the 3-D wavelet coding proposed in [14]. It is envisioned that data compression will continue to be an important issue in IBR applications. More recently, panoramic videos have been proposed to capture dynamic environment maps for applications such as tele-presence and autonomous vehicles [15]–[17]. A panoramic video is a sequence of panoramas taken at different time instants. It can be used to capture dynamic scenes at a stationary location or in general along a path, which is also known as a dynamic or time-varying environment map. It is basically a video

Manuscript received May 14, 2002; revised May 6, 2003. This work was supported in part by the AoE, Information Technology, Hong Kong Research Grant Council. This paper was recommended by J. Ostermann.

K. T. Ng and S.-C. Chan are with the Department of Electrical and Electronic Engineering, University of Hong Kong, Hong Kong (e-mail: ktng@graduate.hku.hk; scchan@eee.hku.hk).

H.-Y. Shum is with Microsoft Research Asia, Beijing 100080, China (e-mail: hshum@microsoft.com).

Digital Object Identifier 10.1109/TCSVT.2004.839989

with 360 degrees of viewing freedom. Another application of panoramic videos is to implement virtual walkthrough applications where a series of panoramas of a static scene along a given path is captured. Therefore, it is a static environment map where one can freely navigate along the predefined paths and freely change their viewpoints. Much emphasis has been put on the construction of panoramic videos and how they can be constructed and rendered [15]–[19], [33]. Although the amount of data associated with panoramic videos is significantly reduced when compared to other possible dynamic image-based representations, it can still be very high, thereby posing a number of practical problems when good resolution and interactive response are required. To illustrate the severity of this problem, let us consider a 2048×768 panoramic image without compression. It will occupy about 4.5 MB of storage. A 25 f/s video at this resolution would require 112.5 MB/s of digital storage or transmission bandwidth. Another problem of high-resolution panoramic videos is the high computational complexity in softwareonly real-time decoding.

In this paper, we are concerned with efficient methods for the compression and transmission of high-resolution panoramic videos for both dynamic environment maps and virtual walkthrough applications. For dynamic environment map applications, a high-performance MPEG-2-like compression algorithm, which takes into account the random access requirement in changing one's viewing angle and the redundancy of panoramic videos, is proposed. For virtual walkthrough applications, the indexing structure proposed in [10] is employed to support random access for individual panoramic images so that the user can freely change his viewing position in the path as well as his viewing angle. The transmission of panoramic videos over cable networks, local area networks (LANs) and the Internet are also briefly discussed. In particular, we describe in detail a video-on-demand (VOD) system that delivers panoramic videos to users over high-speed networks such as high-speed LANs. It is a very challenging problem because typically a VOD system has to serve many users and it requires a very high disk bandwidth [i.e., the data transfer rate of the disk storage, e.g., in a redundant array of inexpensive disks (RAID)] and transmission bandwidth. If fixed amount of server resources and transmission bandwidth are allocated to each user, the VOD system is only able to support a small number of users and it will be very expensive. As some videos are more popular than others, different methods for sharing video streams among users via batching [20]–[23] or broadcasting [24]–[31] were proposed. In batching, a user requests for a video and waits for the availability of the server channel. The server then selects a batch of users to whom the video will be multicast according to certain policies in order to maximize the possible sharing of video streams [20]–[23]. In the broadcasting method [24]–[31], channels are reserved to broadcast frequently requested videos. The broadcasting protocols further improve the efficiency of the systems by reducing the transmission bandwidth for videos that are simultaneously watched by many users. More precisely, a video is partitioned into a number of segments and each segment is repeatedly broadcast on a different channel. The play-out latency depends on how frequently the first segment is broadcast. Subsequent segments might be received before they are actually being played back. Thus, memory buffers are needed to store these video segments. Essentially, these broadcasting schemes take advantage of the resources (e.g., disk) at the user side so as to guarantee a latency independent of the number of requests.

In order to reduce the disk bandwidth of the server and the transmission bandwidth of the network required for dynamic situations other than simply broadcasting, a new video sharing scheme called advanced delivery sharing scheme (ADSS) is developed. The ADSS is equipped with an efficient protocol that allows users to specify in advance those video data that are useful in the future. In so doing, the multimedia server can effectively determine whether the current video segment retrieved is also useful to other users. By broadcasting or multicasting this segment to other intended users, the number of unnecessary accesses and repeated transmission of the same segment of video data can be minimized. An efficient scheduling algorithm for the server to support the ADSS is also proposed. In order to exploit the possible sharing of video data using the ADSS, the users need to have a relatively large memory buffer and receiving bandwidth. As fast and low-cost secondary storages (such as hard disks) and high-speed networks will be widely available in the nearest future, it is envisioned that these requirements can be easily satisfied. Besides, the cost of memory buffers and network resources can be shared with other applications in the set-top box, such as web browsing, video games, and other communication functions. Furthermore, the proposed scheme can be easily extended in a hierarchal manner to include distributed servers.

The rest of this paper is organized as follows. The principle of panoramic videos, their construction and rendering are discussed in Section II. Section III is devoted to the proposed compression and rendering algorithms. The transmission aspects of panoramic videos are briefly discussed in Section IV. Details of the proposed ADSS VOD system and its implementation are described in Section V. Finally, concluding remarks are given in Section VI.

II. Construction of Panoramic Videos

A. Panoramic Videos

Panoramas belong to the family of the plenoptic function. In [1], the plenoptic function, $P(.)$, is defined as the intensity of light rays passing through the camera center at every location (V_x, V_y, V_z) at every possible elevation and azimuth angles (θ, ϕ), for every wavelength λ and at every time t. Hence, it is a 7-D function given by

$$P(V_x, V_y, V_z, \theta, \phi, \lambda, t). \tag{1}$$

The basic idea of IBR is to reconstruct a continuous representation of the plenoptic function from its observed samples.

By dropping the time variable t (i.e., static environment) and the wavelength of light λ, McMillan and Bishop [3] introduced plenoptic modeling using the following five-dimensional (5-D) plenoptic function

$$P_5 = P(V_x, V_y, V_z, \theta, \phi). \tag{2}$$

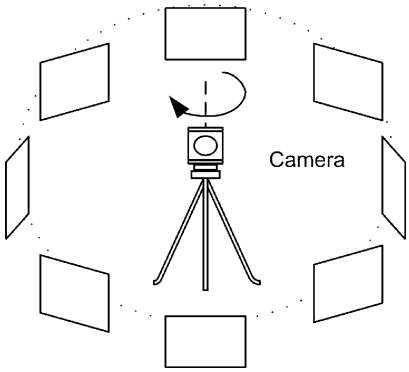


Fig. 1. Construction of a panoramic mosaic.

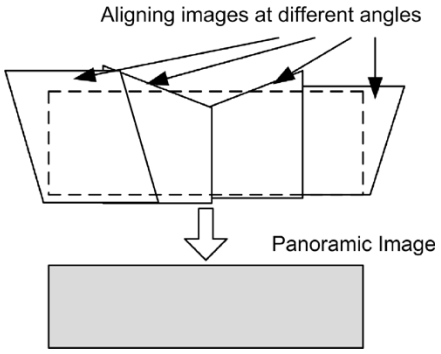


Fig. 2. Mapping of images onto a cylinder to generate a panoramic image.

The simplest plenoptic function is a 2-D panorama (cylindrical [2] or spherical [7]) where the viewpoint is fixed. A panoramic mosaic can be obtained by projecting a series of images (after registering and stitching) on a cylindrical or spherical surface. Figs. 1 and 2 show the construction of a panoramic mosaic. Since it is obtained by stitching several images together, its resolution is usually very large (e.g., 2048×768). Several algorithms for constructing such mosaics or panoramas were previously reported in [2], [7], [32], and [41]. Using the panorama, it is possible to emulate "virtual camera panning and zooming" by projecting appropriate portions of the panorama onto the user's screen [2]. Different projections can be used to map the environment map to 2-D planar coordinates. The cylindrical projection is the most popular for general applications since it is very easy to be captured. One drawback of the cylindrical projection, however, is the limited vertical field of view as compared to the spherical projection. The cubic projection [34] is another efficient representation of environment maps. The captured environment map is projected onto the sides of the cube. Therefore, each environment map consists of 6 images each associated with one face of the cube, making it very simple to manipulate. A panoramic video refers to a sequence of panoramas captured at different time instants. Although panoramic videos are very compact when compared to other possible dynamic simplifications of the plenoptic function, the amount of storage and transmission bandwidth can still be very large as compared with conventional videos. Next, we consider methods for capturing panoramic videos.

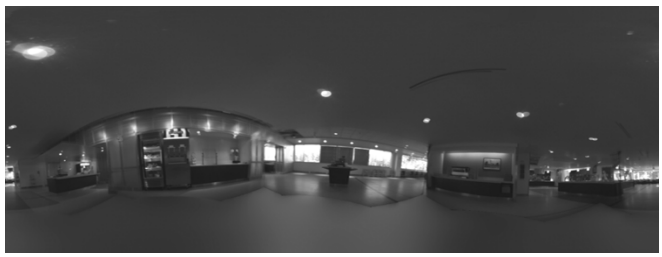


Fig. 3. Frame 8 of the *Cafeteria* panoramic video sequence.

B. Capturing of Panoramic Videos

A time-varying environment map can be obtained by taking panoramas at regular time interval either at a given location or along a trajectory. Such time-varying environment map or panoramic video closely resembles a video sequence with very high resolution. There are different methods to capture a panoramic video [15], [16], [19], [33], [35]. For example, in the FlyCam system [15], multiple cameras are mounted on the faces of an octagon with each side equal to 10 cm. In the system reported in [16], the camera is fitted with a mirror to produce panoramic videos. Specialized hardware for capturing panoramic videos has also been reported in [19], where six closely spaced charge coupled devices (CCDs) are assembled together to minimize parallax. Each CCD is used to capture an image pointing at one of the six faces of a cube. Their outputs are synchronized and streamed directly to disk for storage.

Both real-world and synthetic panoramic videos are considered in this work. For real-world scenes, we use panoramic videos captured by the omni-directional setup proposed in [35]. It comprises a catadioptric omni-directional imaging system [36] with a 1300×1100 pixel camera, all placed on a movable cart. To capture a panoramic video, four video streams of the omni-directional video are taken at different camera orientations (front, left, back, right) along the same path. This arrangement is used because each omni-directional image has blind spots in the middle, and has only about 200 degree field of view from side to side. The resulting panoramic video (with a frame resolution of 2048×768) is created by stitching these four video streams frame by frame. The panoramic video consists of 381 panoramic images. Fig. 3 shows a typical panorama of the *Cafeteria* panoramic video sequence.

For the synthetic scene, the mosaic images of the environment map were rendered using 3-D Studio Max. Cubic projection is used for storing the panoramic video. Each panorama has six input images with a resolution of 256×256 and there are altogether 2910 images. Fig. 4 shows a typical cubic environment map of the synthetic panoramic video sequence *Village*.

C. Rendering of a Novel Video View

Fig. 5 is a flow chart showing the decoding of panoramic videos. At the viewer side, the compressed videos are decoded and rendered to create a scene at a given viewing angle. As the resolution of the panoramic video is usually very large, the decoding or transmission of the whole panoramic video is very often time-consuming. This problem can be remedied by reducing the resolution of the decoded video and/or decoding only

Fig. 4. Typical cubic environment map of the synthetic environment.

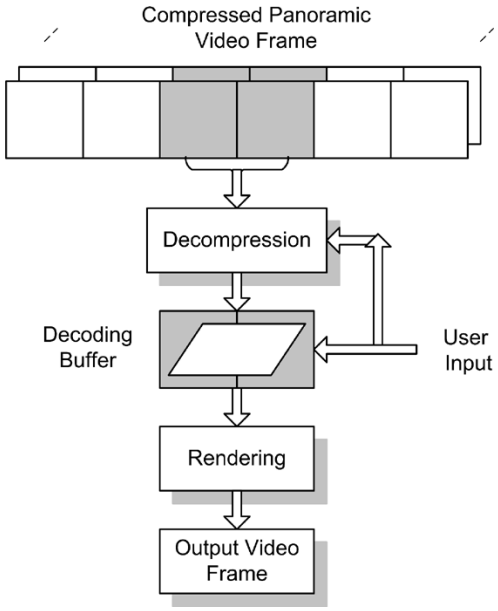


Fig. 5. Rendering of panoramic video.

a given portion in the whole video frame. In virtual walkthrough applications, it is unnecessary to decode the entire video frame because only a fraction of the panorama will be used for rendering the novel view. Because of this reason, the panorama is usually divided into tiles to simplify decoding and data transfer from slower devices such as CD ROM [2].

For a panoramic video sequence with 2-D planar images, like the real panoramic video *Cafeteria*, each panoramic video frame is divided into six vertical tiles as shown in Fig. 5. If the whole panorama has a view of 360 degrees, the maximum viewing angle of each tile is 360/6 = 60 degrees, which is sufficient for most applications. It is therefore only necessary to concurrently decode at most two tiles at a time. Based on the current viewing angle, the tiles involved (the shaped ones) are decoded and placed in the decoding buffer. Appropriate portion of the panorama inside the buffer is used to render the novel view. Tile switching might happen when the user changes his/her viewpoint during the playback of the panoramic video. Therefore, additional mechanism must be provided in the compressed data stream to provide fast tile seeking. This issue is discussed in the following section on the compression of panoramic video.

III. Compression and Rendering of Panoramic Videos

As mentioned earlier, a panoramic video can be used to capture dynamic scenes at a stationary location or along a given path. It can also be used to provide seamless walkthrough by constraining the virtual camera location to a predefined path for image acquisition. Both of these applications are investigated as follows.

A. MPEG-2 Video Coding of Subtiles for Dynamic Environment Map

Similar to traditional videos, successive panoramic images have significant amount of temporal and spatial redundancies. These can be exploited for data compression by video coding techniques such as motion estimation in video coding.

Also mentioned in Section II, each mosaic image is usually divided into smaller tiles to avoid decoding the whole panoramic video and to reduce the data transfer requirement when slower secondary devices are used. It is therefore natural to treat each of these tiles as a video sequence and compressed these tiles individually. If a panoramic video with a resolution of 2048×768 is divided into six nonoverlapping tiles, it yields six video sequences each of which has a resolution of 352×768. To provide functionalities such as fast forward/backward and to make the panoramic video compatible to most decoders, one can employ the commonly used MPEG-2 video coding standard [37] to compress each of these video streams. Another advantage of MPEG-2, as we shall see in Section III-D, is that it is very efficient in compressing high resolution panoramic videos with a compression ratio of more than 100 times, yet with reasonably good reconstruction quality. For applications involving frequent editing of the videos, separate coding of the mosaic images might be desirable. Under these circumstances, the use of still image coding techniques such as JPEG2000 are desirable. Next, we shall consider the organization of compressed video streams to provide efficient access to individual tile during decoding.

1) Selective Decoding Problem (Tile Seeking): For transmission and storage of panoramic videos for dynamic environment map, individual tiles must be organized in an efficient manner in order to support fast switching between tiles during decoding. Fig. 6 shows the format of a tile or video stream encoded using the MPEG-2 standard. Consecutive image frames of a given tile are arranged in groups called group of pictures (GOP). In each GOP, the image frames are encoded as I-, P-, or B-pictures. I-pictures are intracoded and are used as references for predicting the next P- and other B-pictures in between using motion estimation. P-pictures are predicted using motion estimation from the previous I- or P-pictures. B-pictures are bidirectionally predicted from the nearest reference pictures. The arrows in Fig. 6 show the interdependency of this prediction method between various pictures in a GOP. In the proposed coder for dynamic environment map, there are seven pictures in each GOP, which consists of one I-picture, two P-pictures, and four B-pictures. Also shown in Fig. 6 is the sequence order of the compressed image frames to be transmitted. Note that the

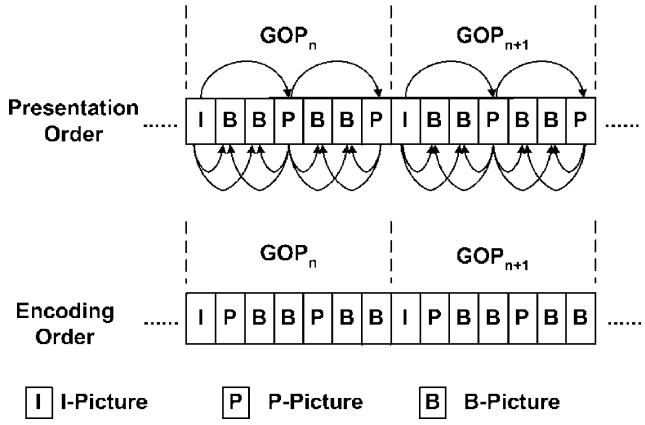


Fig. 6. GOP setting in MPEG-2 bitstream.

reference pictures are transmitted before the B-pictures because they must be decoded before the B-pictures. They serve as references for reconstructing the B-pictures in between.

Fig. 7 illustrates how the six tiles (video streams) of the panoramic video are multiplexed in the proposed method. Each tile is encoded by the MPEG-2 standard with the same GOP structure shown in Fig. 6. The compressed data of the tiles in the same panoramic video frame are packed together. This allows the decoder to locate very quickly the corresponding I-pictures when decoding the required tiles. An individual picture in each tile can be accessed randomly by searching for the appropriate picture header. During decoding, the viewer can selectively decode the tiles required by the user, for example, streams 1 and 2 in Fig. 7. The novel view can then be generated by remapping appropriate pixels in the tiles onto the user's screen.

When the viewing angle is changed in such a way that some of the required pixels are no longer in the tiles currently being decoded, switching to the new tile(s) has to be performed. If this happens during the decoding of P- and B-pictures in a GOP, tile switching can only begin in the next GOP. It is because the I-pictures of the new tiles in the current GOP might not be available. (In practice, previous data of new tiles is usually not buffered.) Hence, the separation of I-pictures in panoramic video streams should not be very large. Otherwise, it would introduce unacceptable delay in switching from one stream to another. As mentioned earlier, there are seven images in each GOP. At a frame rate of 25 f/s, the maximum delay during tile switching is therefore 0.28 second, which is quite acceptable. Other values can be chosen according to one's tradeoff between the compression performance and the response time delay. The synchronized I-pictures also allow us to preserve the fast forward and fast backward capability in the MPEG-2 standard. Notice that the number of P- and B-pictures in GOPs from different tiles can be different (as well as GOP from the same tile), provided that their *I*-pictures are synchronized. It helps to improve the compression performance, but at the expense of more complicated encoding and decoding processes.

B. Modified MPEG-2 Video Coding for Virtual Walkthrough Over Static Scenes

For virtual walkthrough applications, users are allowed to move along a given path and change freely their viewpoints. Therefore, a slightly different GOP structure to be described below is employed. The compressed panoramic video bitstream is usually stored in local storage or downloaded from the network before decoding. The image frames of the panoramic video are then accessed on demand for rendering according to the user's viewing position. In fact, it is very time-consuming to retrieve the image frames if the bitstream does not support any mechanism for random access. Therefore, we modify the MPEG-2 algorithm in order to support random access to individual image frames. In Fig. 8, a set of pointers to the starting locations of each image frame in the compressed data is first determined and stored in an array prior to rendering. Alternatively, the pointers can be embedded in the compressed bitstreams to avoid creating the pointer arrays when new panoramic videos are loaded into the memory at the expense of slightly lower compression ratio. During rendering, the compressed data for the required image frame can be located very quickly. For an I-picture, the pointer structure mentioned earlier can be used to access the compressed data. If B-pictures are added, for higher compression ratios, the pointer structure would only allow us to efficiently decode the motion vectors and the prediction residuals. The two reference I-pictures are required to be decoded first. Furthermore, if P-pictures are employed, then when a user moves backward along a path, a number of previous P-pictures have to be decoded due to their interdependence. Therefore, for efficient rendering, we do not employ P-pictures in the proposed compression algorithm for static scenes. As shown in Fig. 8, each GOP has one I-picture and six B-pictures. For simplicity, no rate control algorithm is applied and a uniform quantizer is used for the I- and B-pictures.

C. Rendering of Panoramic Videos

For dynamic environment maps, the panoramic videos are streamed from the server. The panoramic video viewer of the proposed system is implemented using the Microsoft DirectShow and Direct3D application programming interfaces (APIs) [38]. The DirectShow API is a media-streaming architecture for the Microsoft Windows platform, which provides high-quality capture and playback of multimedia streams. The basic building block of DirectShow is a software component called a *filter*. A filter generally accepts a multimedia stream as its input and performs a single operation on it to produce the output. For example, a filter for decoding MPEG-2 videos has its input an MPEG-encoded stream and the output is an uncompressed RGB video stream. Fig. 9 shows the filter graph of the panoramic video viewer for each user. Multiple data streams associated with a single panoramic video are retrieved from local storage devices or from the video server. Each data streams are then decoded using the Elecard MPEG-2 Multiplexer and Video Decoder filter [39]. The decoded video frames are copied to the texture buffer of the Panoramic Video Renderer filter for rendering. For fast rendering speed, we also make use of Direct3D to render and display the output images in the Panoramic Video Renderer filter. More precisely, the decoded panoramic image is projected onto a geometry model, which can be cylindrical, spherical or cubical. Subsequent rendering of the scene at different viewing angles is handled by Direct3D APIs. The viewer

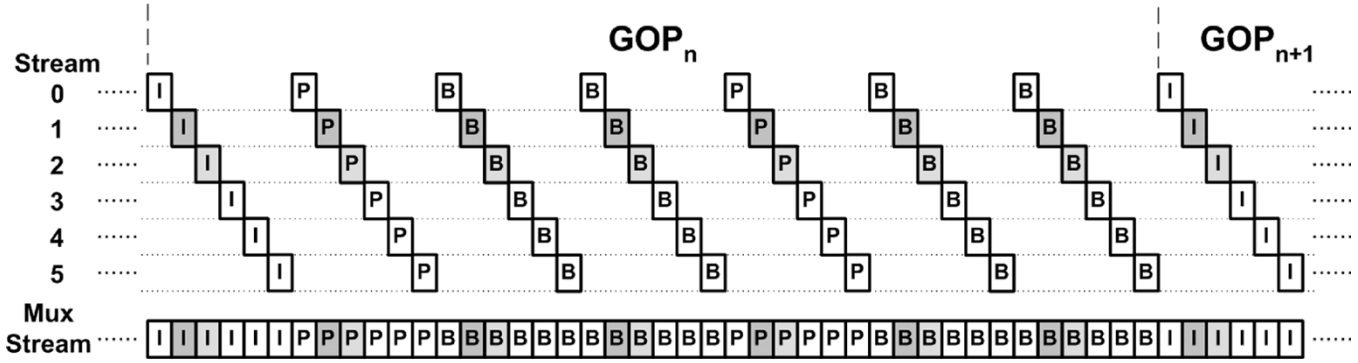


Fig. 7. Multiplexing of the tiles (streams) in the MPEG-2 compressed panoramic video.

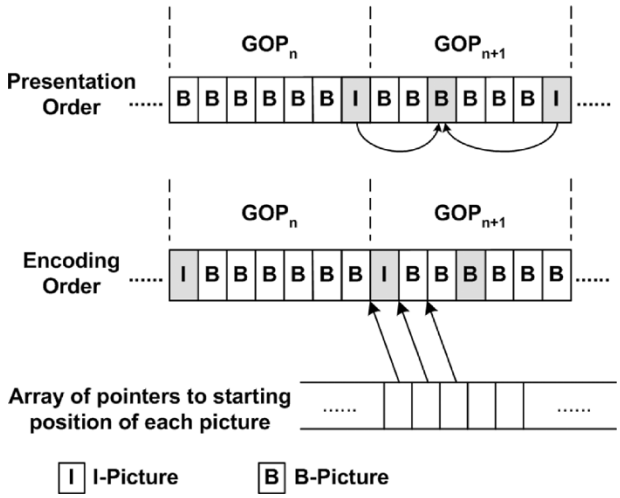


Fig. 8. GOP setting in MPEG-2 bitstream for virtual walkthrough over static scenes.

allows the user to pan, zoom and navigate interactively in the video by choosing his/her rotation angle of the viewing camera.

For the virtual walkthrough application, the modified MPEG-2 video decoder retrieves the panoramic images from the compressed bitstream. The rendering and display are also implemented using Direct3D APIs. The user interface for the virtual walkthrough application has two windows: the viewport and the plan map of the scene. The viewport renders the virtual camera view at the current location. The plan map indicates the current position of the virtual camera and the current viewing direction. The user can freely navigate in the static enviroment map or change its location along the path by clicking at the desired destination on the plan map.

D. Experimental Results

The *Cafeteria* panoramic video sequence described in Section II was compressed using the proposed coding algorithms for dynamic scene environment application. Although the *Cafeteria* sequence was captured from a static scene, it is used for simplicity to illustrate our algorithm in the dynamic situation. The six tiles of the panoramic video were encoded using the MPEG-2 video coding standard with the TM5 model. Each stream has a GOP consisting of seven image frames with two B-pictures between successive I- or P- pictures as illustrated in Fig. 6. Table I shows the compression performance of the panoramic video sequence using the proposed algorithm at different bit rates (target bit rate of 1 and 1.5 Mb/s per tile). Figs. 3 and 10 show respectively a typical panorama and the decompressed tiles of the panorama. The results show good quality reconstruction with a compression ratio of 108. When the compressed data is streamed from a remote PC throught a LAN, the rendering speed of the viewer is about 7 f/s (neglecting network congestion) using a Pentium 4 1.8 GHz PC with 256 MB memory.

For the virtual walkthrough (static scene) experiment, we used the synthetic panoramic video sequence *Village*. For simplicity, it was projected onto a cubic geometric model. Each environment map therefore consists of six images, one for each face of the cube. The image sequence of each face was compressed as a video stream. The appropriate image frames, according to the current viewing angle, were decoded during rendering. Table II shows the compression performance of the synthetic panoramic video sequence. Example screenshots of the synthetic environment during the virtual walkthrough experiment are shown in Fig. 11. The perceptual quality is quite good with a compression ratio of 30. The low compression ratio of the synthetic scene as compared with the real scene is due to its lower resolution, complicated textures, and sharp edges, which make coding more difficult. The overall results demonstrate that panoramic videos are efficient means for providing impressive 3-D visual experience to the users. For real-time rendering, we can achieve 20 f/s from raw data and 15 f/s from compressed bitstream using a Pentium 4 1.8 GHz PC with 256 MB memory. It is expected the frame rate can be increased after further optimization/enhancement of the C++ source program. Next, we briefly outline the transmission aspect of panoramic video over cable networks, LANs and the Internet.

IV. Transmission of Panoramic Videos

In order to deliver the interactive virtual walkthrough experience offered by panoramic videos, the compressed data stream can be broadcast or transmitted using VOD systems over, for example, the Internet, LANs, or cable networks. For broadcasting applications, say, over cable networks, the whole panoramic video can be transmitted through a few cable TV channels with each channel carrying one or more tiles of the video streams. The set-top box can be configured according to the user input so

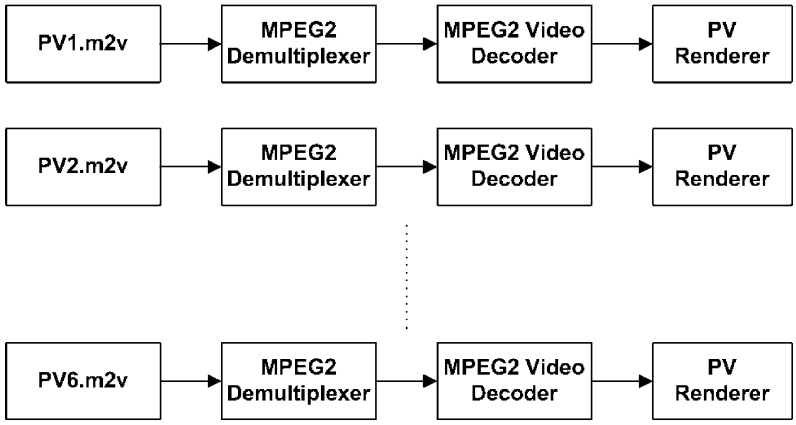


Fig. 9. Filter graph of panoramic video viewer.

TABLE I
COMPRESSION PERFORMANCE OF THE PANORAMIC VIDEO SEQUENCE *CAFETERIA*

Bit-rate (Mb/s)	Compression Ratio	Mean PSNR (dB)		
		Y	U	V
5.727	162	42.16	45.69	44.74
8.596	108	45.40	47.10	46.60

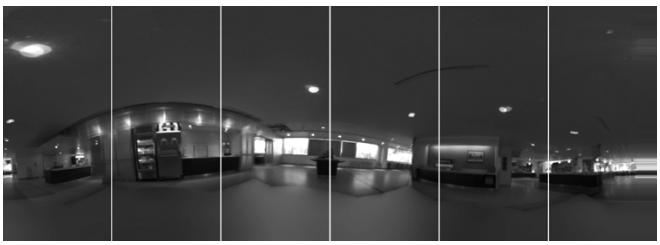


Fig. 10. Frame 8 of the decompressed panoramic video sequence *Cafeteria* at the bit rate of 1.5 Mb/s per tile.

TABLE II
COMPRESSION PERFORMANCE OF THE SYNTHETIC PANORAMIC VIDEO SEQUENCE *VILLAGE*

Compression Ratio	Mean PSNR (dB)		
	Y	U	V
30.49	36.13	39.54	41.40
35.02	35.26	39.10	41.03

Fig. 11. Example screenshots of the synthetic environment during the virtual walkthrough.

that the appropriate tiles in the panoramic video will be decoded. Since the panoramic videos are divided into tiles, only a limited number of tiles, two in the proposed system, have to be decoded. Additional hardware is required to render novel views from the decoded video streams. For broadcasting over LANs, the decoding and rendering are most likely performed by a workstation or PC. With present-day technology, real-time rendering and decoding of panoramic videos do not present significant problems. In applications where the channel has limited and/or dynamic bandwidth such as communications over the Internet, the tiles can be transmitted on an "on-demand" basis, where only the required video streams are transmitted. Further reduction of bandwidth for transmission can be achieved by creating a scalable bitstream using, for example, multiresolution techniques.

The performance of such a VOD system is usually limited by the transmission bandwidth of the network and the capability of the server. The latter in turn is limited by the relatively slow access time or disk bandwidth of the secondary devices in the server and its scheduling algorithm. In the following section, a concept called "advanced delivery sharing" for reducing the transmission and disk bandwidth required in the VOD system is introduced.

V. ADVANCED DELIVERY SHARING SCHEME

A. Basic Principle

Fig. 12 shows the architecture of a distributed VOD system where video servers with video archives are connected through a high-speed network such as SONET. Each video server supports the VOD requests from its local users through a LAN such as Gigabit Ethernet. If the videos requested by the users are not available at the local server, the requests will be forwarded to other servers in the network that possess these video data. Apart from forwarding requests for video data to other servers, the major task of the server is to schedule the requests, both from its local users and from other servers, and deliver the required video data that it has archived. The motivation of using the proposed scheme is to reduce disk and/or transmission bandwidths when multiple copies of the same video stream are simultaneously retrieved from these servers. Because of the large bandwidth requirement of panoramic videos, it is very likely, as in other applications, that the bottleneck will be the disk bandwidth of the servers. Fig. 13 shows a simple scenario where two identical video streams are retrieved from a server. User 1 starts the video at time t_1 while another user, User 2, starts viewing the same video at a later time t_2. If t_1 and t_2 are close to each other, it might be possible to delay User 1 so that the two streams can be merged together to reduce disk and transmission bandwidths.

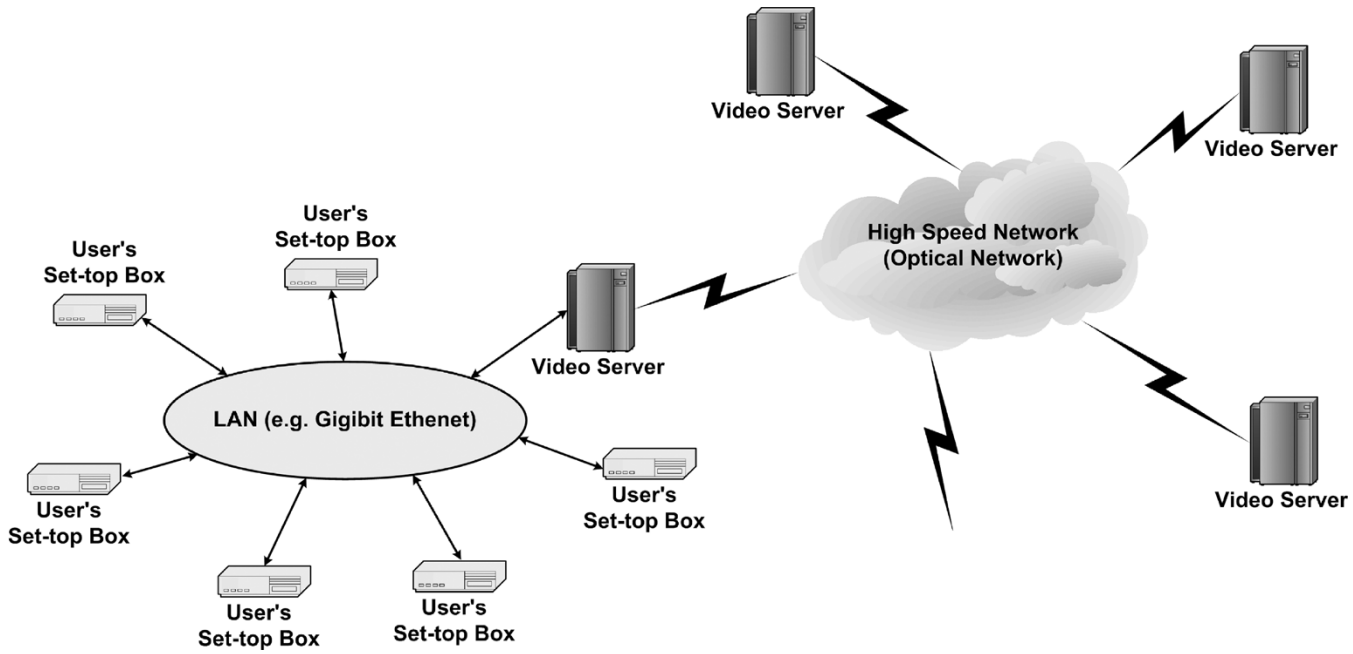


Fig. 12. System model of a distributed VOD system.

There are two practical problems associated with this simple approach. First, the time difference has to be small enough so that the two streams can be merged together without introducing excessive delay to User 1. Second, such merging is no longer feasible when some of the users perform fast forward/backward operations on the video at a later time. It might also require complicated hardware or algorithms to detect whether two or more streams can be merged together, and to delay them appropriately. Merging is, however, useful to reduce the number of new but identical video streams. It can be accomplished by treating adjacent requests as a batch. For example, one can merge requests for a given video that occur within a given interval T together into a single new video stream. A disadvantage, though quite acceptable in practice, is that some of the users might experience a worst-case delay time of T as a result of merging.

Although it is somewhat difficult in the previous example to merge the two streams together if t_2 is much larger than t_1, the video data retrieved from the server for User 1 is obviously useful to User 2 at a later time. Let us examine it more carefully and assume that the video data stream is divided into N segments: $M_i, i = 1, \ldots, N$, each of equal size. Suppose further that at time t_2, the server retrieves the segment M_k for User 1 as shown in Fig. 13. This segment and the subsequent ones, though not immediately requested by User 2 and other users, are useful to them at a later time. If these segments are also sent to other users who will be using them later, the disk bandwidth of the server and probably the network transmission can be drastically reduced. It is because segment M_k, while retrieved once, is able to serve many other users requesting it at a later time. The above discussion is the basic idea of the proposed ADSS. To achieve the sharing of video data, an efficient protocol that allows users to specify in advance those video data that they will be using in the future has to be developed. It can be seen that the effectiveness of this scheme is improved by increasing the cache size of the viewers, the network bandwidth and its capability for supporting multicasting. With the rapid development of high-speed secondary storage technology over the past few years, it is envisioned that set-top boxes and personal computers in the nearest future will be equipped with considerable amount of reasonably fast secondary storage. Secondary storage for video caching in the order of 500 MB to even 1 GB is therefore quite affordable. This additional storage also helps to smooth out the video traffic during fast forward/backward operations. Similar kind of caching scheme can also be used at the server level to reduce frequent requests for video data that are not stored locally or to reduce the cache size at the user's set-top box.

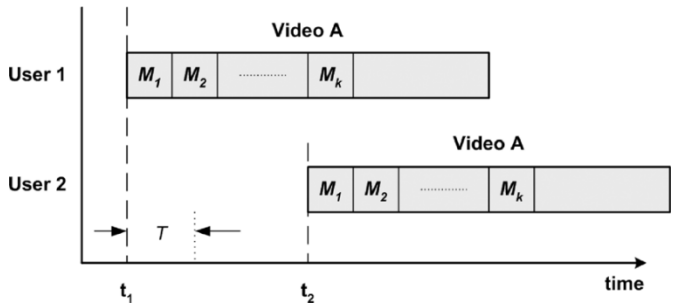


Fig. 13. Simple scenario where two identical video streams are retrieved from the server.

In what follows, we shall propose a simple but efficient protocol to support the ADSS together with a scheduling algorithm of the server.

B. ADSS Protocol (ADSP)[1]

We will discuss the protocol at the user's side followed by the scheduling algorithm of the server.

1) Users: The computer terminal or set-top box at the user side consists of two major parts: the video player and the cache

[1]The ADSP was first studied by the second author and a preliminary system was implemented in the report [40].

manager. The video player decodes the multimedia data and displays the output on the user's screen. The cache manager communicates with the server to ensure proper delivery of data and forward the data sequence to the video player for decoding. In the ADSS, the user terminal is assumed to have a cache of certain size, say, C. The cache manager is responsible for making requests to the server 1) to maintain the normal real-time playback of the video and to avoid buffer overflow, for instance, due to fast forward operation of the user, and 2) to inform the server what additional future video data it would like to receive. To carry out the latter, the cache manager has to perform simple cache management such as deleting some video segments that have not been used for a long time. We call request 1) and request 2) the *primary request* and the *secondary request,* respectively.

A primary request is associated with requests for video segments which should be handled immediately in order to maintain the continuous playback of the video. As a simple example, consider the situation where a user first requests for a new video. The cache manager then sends a primary request to the server consisting of the name of the video and the time of initial viewing (say 15 min) measured in terms of the number of video segments, say, W_P. After a sufficient amount of data is received (for example, a certain fraction of W_P), the cache manager informs the video player to start playing the video. The cache manager then issues a similar primary request to the server when the amount of data left in the data buffer to maintain continuity of the video is less than a certain level, say, $\gamma \cdot W_P$. Furthermore, when the rate of data consumption is greater than the normal one, as a result of user's interaction such as fast forward or fast backward operations or transmission delay, the priority level of the primary request, *Pri_level*, can be increased accordingly. In its simplest form where only two levels of priority are used, *Pri_level* can simply be set to 1 to indicate to the server that the current request should be served prior to other normal primary requests. A possible way to measure the data consumption rate is to examine the decoding buffer at regular intervals and compare it with the bit rate of the video. Other possible variation of this basic scheme includes sending reminder messages to the server if the primary request has not yet been served (resulting in severely limited data in the playback buffer). The choice of W_P and various other parameters are tradeoffs of communication overheads between the users and the server, initial delay, and memory requirement.

Secondary requests are primarily used to support sharing of active video streams as mentioned in Section V-A. In the previous example, in making the first primary request, the cache manager might also inform the server that it would like to receive video segments up to W_S, judging from its current cache size. It is possible that other users have already sent primary requests for some of these additional segments. Consequently, the server can multicast or broadcast copies of these segments to the current user and other similar users who have specified them in previous secondary requests. Since secondary requests are mainly associated with future but useful video segments, they are of lower priority than primary requests and might not be immediately entertained by the server. Apart from reducing the disk bandwidth of the server and probably network traffic, secondary requests also help to smooth out their momentary fluctuations resulting from user's interaction such as fast forward/backward operation, if the additional segments are already inside the user's buffer as a result of advanced delivery through secondary requests. The pseudocode of the cache manager is summarized in Fig. 14.

```
start video(video_id);
send primary request(video_id, W_p);
send secondary request(video_id, W_s);
while ( (timer == checking_period) and
  (receiver video segment( ) != all_video_segment_received) )
{
      if (user interaction( ) == true)
            handle user interaction( );
      check buffer status(buffer_status);
      if ((timer == update_request_period))
      {
            if (buffer_status != normal)
                  send reminder message( );
            send primary request(video_id, segment info);
            send secondary request(video_id, segment, info);
      }
      cache management( );
}
end video( );
```

Fig. 14. Pseudocode of the cache manager.

2) Scheduling Algorithm of the ADSS Server: In addition to the logistic function of transmitting the video segments to the users, another important function of the ADSS server is to schedule appropriately these data retrivals based on the requests from the users, so that buffer underflow in the user side can be minimized. When a request (primary and probably secondary) for a new video stream is received, the server tries to merge similar requests for the same video that occurs within a period of time T so as to reduce the actual number of streams being served. The server also performs admission control. When the system resources exceed a limit (for instance, the maximum number of streams, after merging, has been reached), the users may be blocked from using the VOD services or queued in the system waiting for these services, depending on certain admission policies [20]–[22]. If the request is accepted, the server appends the primary request to the end of its *task list* as depicted in Fig. 15. The *task list* is the list of video streams, in descending order of their serving priority, that the server is currently serving. Each node in the list contains the identity of the video, the index of the next video segment to be served, the users requesting it as primary requests (user list) and secondary requests (secondary request list). Fig. 15 shows an example of the *task list*. Each node in the *task list* contains the following fields: *video_id*—index of the video associated with this task, *segment_id*—segment index of the video to be served, *pointer to user_list*—pointer to the list of users supported by this video stream, and *pointer to secondary_request_list*— pointer to the users having secondary requests for the current video.

Each node in the *user list* contains the following fields: *user_id*—identity of the user supported by this video stream, and *segment_list*—list of video-segment requests by the corresponding user in his/her primary request.

Normally, the server serves the first node in its *task list* by sending the video segment specified in the *segment_id* of the video, which in turn is specified in the *video_id* field, to those users that are specified in the *user list*. If the current segment is the last segment of a user in the *user list*, it is removed from

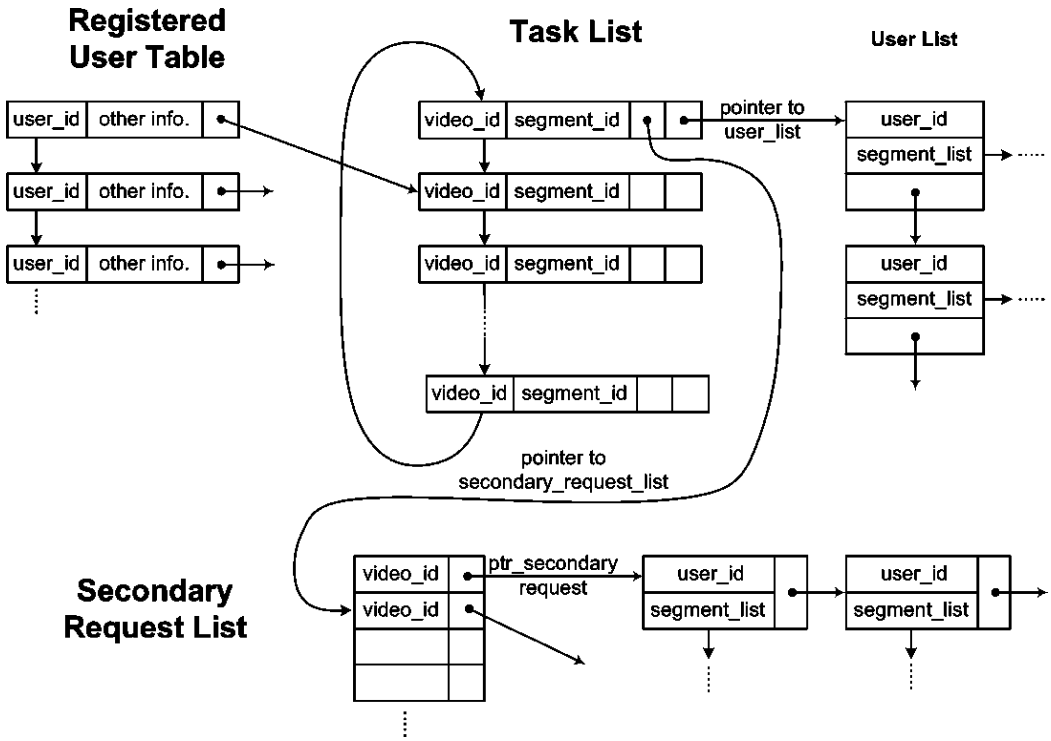


Fig. 15. Scheduling algorithm of the ADSS server.

the *user list*. Also, the segment index *segment_id* is updated accordingly. For example, it will be increased by one if the video is playing in the forward direction and vice versa. The *segment_list* is also updated. If the served video segment is the last video segment of the entire node, the current node is removed from the *task list*. Otherwise, the served node is moved to the end of the list with the lowest priority. Effectively, a round robbin type of priority scheme is used. As mentioned earlier, there might be situations where primary requests or reminders from the user with high priority are received due to transmission delay or other users' interaction such as fast forward/backward (which results from running out of buffered data at the user side). In this case, these higher priority primary requests or reminders are served by the server immediately, after serving the current node in its *task list*. The server can use its *user list* to locate the corresponding node of this urgent user in its *task list*. The *user list* is a list recording all the users in the system as shown in Fig. 15. Each node contains the user identity and a pointer to the corresponding task node in the *task list*. After serving it, the server resumes its normal serving order. A user can readily extend its period of viewing by sending one or more primary requests to the server. (If previous secondary requests hit, i.e., successful, it is unnecessary to do so.) The server then locates its corresponding node in the *task list* and updates its *segment_list* field. Other modifications, such as stop viewing the video (removing it from the *user list*, etc.) can also be made to the *task list*.

The ADSS protocol can also multicast the current video segment being served to other users who have specified it in their secondary requests. To support this, the server needs to maintain a *secondary request list* indexed by the identity of the videos, *video_id*, as indicated in Fig. 15. Each node of the *secondary_request_list* has two fields: *video_id*—identity of the video in the achive, and *ptr_secondary_request*—pointer to the list of secondary requests from users for the video with identity *video_id*.

Each node of the list contains the following two fields: *user_id*—identity of the user associated with this secondary request, and *segment_list*—list of video segments specified for this secondary request.

When the server processes a given node in its *task list*, it uses the *video_id* field in the node and the *secondary request list* to locate those secondary requests related to the current video. If the *segment_id* lies within the range specified in the *segment_list* of any of these secondary requests, the server will multicast a copy of the current video segment to the corresponding users. The *segment_list* field of the node is updated accordingly. Since the video segment is retrieved once, the segments are shared by all users (both primary and secondary). Hence, the read-write access and/or the network traffic can be reduced. Further modifications of secondary requests from users can readily be achieved by using the data structure considered here.

3) Extensions to Panoramic Videos: Stream sharing in panoramic videos is more complicated than just ordinary videos. The reason is that even if two users are watching the same panoramic video, they might not use the same set of tiles. One might be using tiles 1 and 2 and the other tiles 3 and 4. To simplify the handling and exploration of tile sharing, the primary requests for the same panoramic video at the very beginning are still kept in the same task node. An additional field, *tiles*, in the node of the *user list* is added as follows: *user_id*— identity of the user supported by this video stream, *segment_list*—the list of video segments requests by the corresponding user in its primary request, and *tiles*—tiles that the current user is using.

When this task node is served, all the tiles required by the users as specified in the *tiles* field are served. Those tiles in

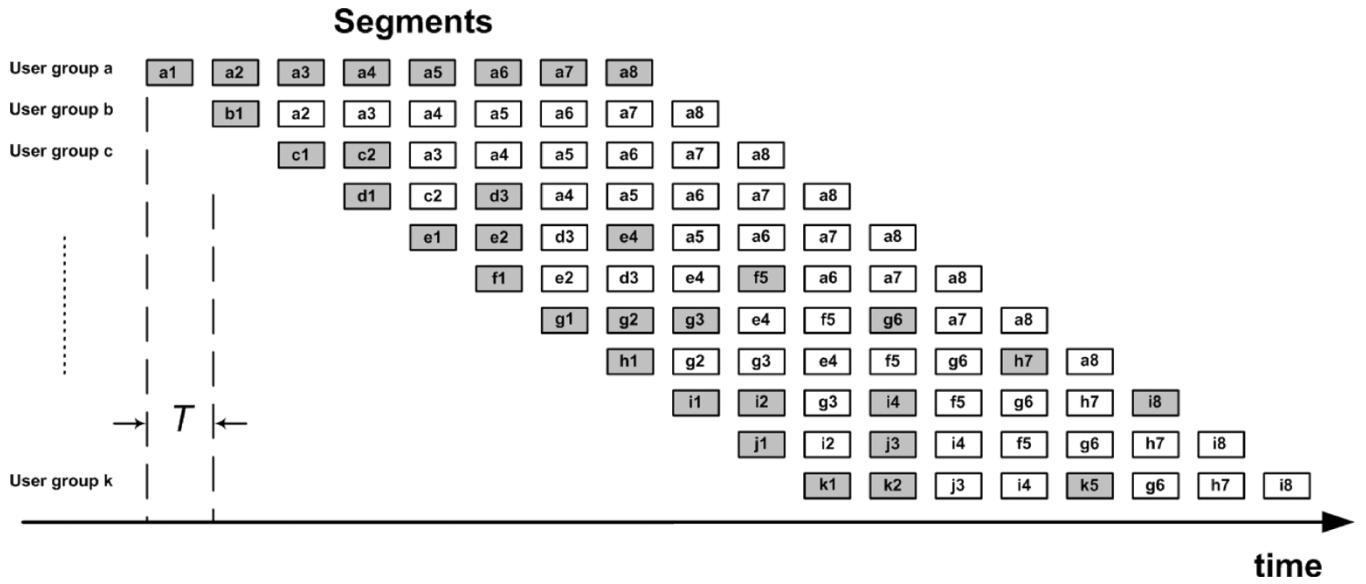


Fig. 16. Operation of the ADSS in a broadcasting scenario.

common are retrieved only once and transmitted either through multicasting or broadcasting. Since it is very difficult to predict which tiles a given user will be using in the future, we do not distinguish tiles in secondary requests. That is, if the current video segment was specified in some secondary requests of some users, any tiles within this segment will be multicast or broadcast to those users. As the ADSS is a highly flexible framework for providing stream sharing, other modifications such as different admission policies or incorporation of special videos for supporting fast forward/backward can also be used to further improve its performance. A simple system was developed to verify the proposed ADSS protocol in a network of PCs connected by a 100 BaseT LAN. The details will be given at Section V-D.

C. Performance Comparison With Other Broadcasting Schemes

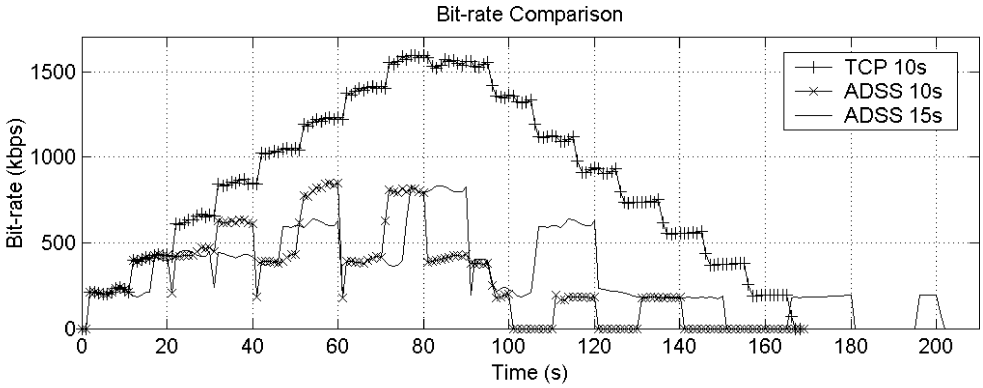
Considering a special configuration of the ADSS, where a group of new users requesting for the same video within an interval of time T are merged together. The first batch of users is denoted by *group a*, while the second group is *group b*, and so on, as shown in Fig. 16. The situation is similar to a broadcasting scenario, where users arrive at regular time interval and the maximum waiting time is T. As shown in Fig. 16, the server serves the users in *group a* by sending one video segment every time interval T to maintain the continuous playback of the video. The video segments are denoted by **a1**, **a2**, etc. As the users are assumed to be the only users requesting for that video in the system and there is no sharing with other users, these video segments have to be retrieved continuously and are denoted in gray color. For users belonging to *group b*, who request the same video in the next interval of time T, the video server has to send them a video segment **b1** because no sharing with *group a* users is possible. It is also marked in gray color to indicate the situation of no sharing. However, because of their secondary requests at the very beginning, it is unnecessary for the server to send all the subsequent video segments to *group b* users (marked with white color). They receive them simply by multicasting or broadcasting. For users in *group c*, who request the same video in the next time period T, the first two segments of the video have to be sent to them by the server (**c1** and **c2** in gray color), and all subsequent video segments can be shared with *group a* users via secondary requests. Fig. 16 also illustrates the situation where other user groups up to group k enter into the system regularly. All the shared video segments as a result of secondary requests are marked in white. The segments that cannot be shared by the protocol are marked in gray. Suppose that the bandwidth to retrieve each video segment from the disk (e.g., a RAID disk) is b and there are n segments in the video, then the average disk bandwidth $B(n)$ to support all users is

$$B(n) = b + \frac{b}{2} + \frac{b}{3} + \cdots + \frac{b}{n} = \sum_{i=1}^{n} \frac{b}{i} = b \sum_{i=1}^{n} \frac{1}{i} = bH_n \quad (3)$$

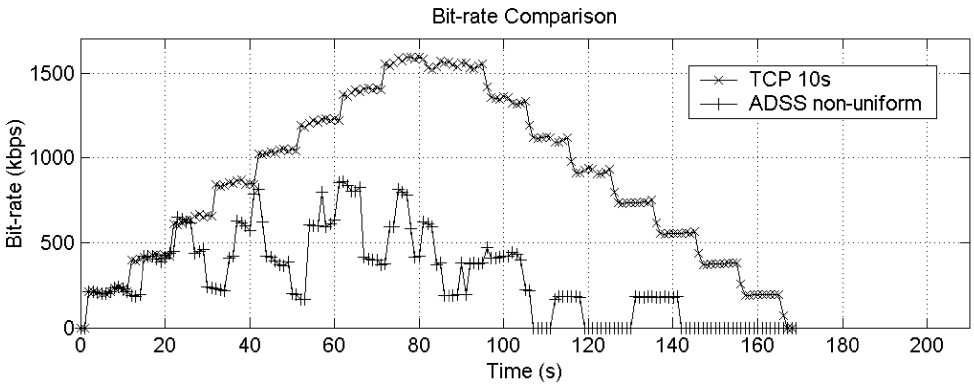
where H_n is the harmonic number of n. Here we have assumed for simplicity that the disk cache of the user is large enough to maximize the sharing. In practice, the transmission bandwidth is larger than $B(n)$. If the multicast transmissions are simply implemented as broadcasting, (3) also represents the average transmission bandwidth over the broadcasting channels. Interestingly enough, this situation is identical to the one proposed in the harmonic broadcasting scheme [28], which has been shown to achieve the minimum bandwidth requirement for the same waiting time [31]. For more general situations, it is expected that the ADSS protocol is able to support efficiently more dynamic user requests and to achieve significant bandwidth savings resulting from possible sharing of video segments.

D. Experimental Results

An experimental system with nine computers was built to verify the proposed ADSS protocol. An Intel XEON 1.8-GHz workstation with 512 MB memory was used as the video server. A 100 BaseT hub was employed to connect eight other Pentium 4 1.8 GHz user PCs each with 256 MB memory. The proposed ADSS protocol is implemented in C++ programming language. For simplicity, we assume that the server will merge the



(a)



(b)

Fig. 17. Disk bandwidth required for four different scenarios (a) TCP 10 s, ADSS 10 s, and ADSS 15 s. (b) TCP 10 s and ADSS nonuniform.

same video request within a period of 1 s. The cache size allocated to the video segments in the user terminals was 20 MB. In order to demonstrate the sharing among users, only the *Village* panoramic video was used. Its bit rate is 197 kb/s at 7 f/s and the total duration is 95 s. The lower frame rate was chosen so that the network will not be overloaded. As a result, other factors such as traffic congestion will not affect the performance measurement of the proposed ADSS protocol. All six tiles of the panoramic video are transmitted to the users. Each user then selectively decode the panoramic images from the compressed bitstream using a software MPEG-2 video decoder. The rendering and display of the panoramic video were implemented using Direct3D APIs. The rendering speed of the viewer is 7 f/s and the maximum latency time measured in normal operating condition (i.e., without network congestion) is 2 s. Higher frame rate can be obtained by further code optimization. Fig. 17 shows the disk bandwidth measured for four different scenarios.

1) TCP 10 s: The eight PCs request the panoramic video successively and the time difference between two successive requests to the server is 10 s. This is similar to the batching or broadcasting situation where users enter the system at regular interval. The server opens a TCP socket for each user and sends the whole video to the users periodically.
2) ADSS 10 s: The sequence of user requests is identical to 1) except that the server is running the ADSS and it broadcasts data segments to the users using UDP[2] sockets according to the scheduling algorithm described in Section V-B2. In order to prevent the loss of UDP packets, each user will send a reply to the server after the data is received. There are occasionally collisions of the replies from different users and the network performance will be slightly degraded.
3) ADSS 15 s: Same as 2) except that the time difference between successive requests is increased to 15 s. This is to test the effect of increasing the time interval between successive requests on the performance of the proposed sharing scheme.
4) ADSS nonuniform: Same as 2) except the time difference between successive requests are chosen randomly between 5–15 s (the mean arrival time is 10 s). This is to test the effect of nonuniform arrivals of the request on the performance of the proposed sharing scheme.

From Fig. 17, it can be seen that the ADSS provides significant savings in disk bandwidth over direct streaming of the videos. The bandwidth required for 2) is slightly lower than 3), since shorter arrival time allows greater sharing between the video streams. The bandwidth required for 4), the nonuniform case, is similar to 2) and 3), indicating the effectiveness of the proposed scheme in handling time-varying traffic. Finally, Table III shows the bit rate in terms of memory access, which is the same as the transmission bit rate, of the video server for the four scenarios. It can be seen that the ADSS reduces the transmission bandwidth to one-third of the original unshared value. Again, it can be seen that the sharing among users becomes smaller when the time between successive requests increases. Also, the performance of the ADSS is not too sensi-

[2]UDP is chosen to demonstrate the saving of transmission bandwidth in the ADSS protocol by means of multicast. TCP/IP can also be used.

TABLE III
MEMORY ACCESS AND TRANSMISSION BANDWIDTH FOR PANORAMIC VIDEO SEQUENCE *VILLAGE*

	Bandwidth (kbps per user)
TCP 10 s (uniform arrival)	196.98
ADSS 10 s (uniform arrival)	66.81
ADSS 15 s (uniform arrival)	85.41
ADSS ~10 s (non-uniform arrival)	65.77

tive to nonuniform arrival of users. If UDP is used to transmit the data packets, error concealment techniques can be used to handle the packet loss.

VI. CONCLUSION

In this paper, we have presented new compression and transmission techniques for panoramic videos. A panoramic video allows users to change their viewpoint interactively in a static or dynamic scene along a predefined path or trajectory. In particular, we have presented a high performance MPEG-2-like compression algorithm, which takes into account the redundancies of panoramic videos and the random-access requirements. Data pointer arrays were proposed to support effective access of the compressed data and the GOP is chosen so as to avoid possible interdependency during decoding. It helps to reduce the amount of storage and transmission bandwidth of high-resolution panoramic videos and simplify their real-time software-only decoding. Transmission aspects of panoramic videos over cable networks, LANs and the Internet have also been discussed. An efficient ADSS for reducing repeated transmission and retrieval of frequently requested video segments was introduced. The protocol and the scheduling algorithm of the ADSS were described in detail. They were implemented and verified in an experimental VOD system, which consists of a video server and eight Pentium 4 computers. Considerable savings in memory access and transmission bandwidth of the video server were measured under normal network traffic.

ACKNOWLEDGMENT

The authors would like to thank Dr. S. B. Kang of Microsoft Research, Redmond, WA, for providing them the panoramic video *Cafeteria* which was studied in this work.

REFERENCES

[1] E. H. Adelson and J. Bergen, "The plenoptic function and the elements of early vision," in *Computational Models of Visual Processing*. Cambridge, MA: MIT Press, 1991, pp. 3–20.
[2] S. E. Chen, "QuickTime VR – An image-based approach to virtual environment navigation," in *Proc. Computer Graphics (SIGGRAPH'95)*, Aug. 1995, pp. 29–38.
[3] L. McMillan and G. Bishop, "Plenoptic modeling: An image-based rendering system," in *Proc. Computer Graphics (SIGGRAPH'95)*, Aug. 1995, pp. 39–46.
[4] M. Levoy and P. Hanrahan, "Light field rendering," in *Proc. Computer Graphics (SIGGRAPH'96)*, Aug. 1996, pp. 31–42.
[5] S. J. Gortler, R. Grzeszczuk, R. Szeliski, and M. F. Cohen, "The lumigraph," in *Proc. Computer Graphics (SIGGRAPH'96)*, Aug. 1996, pp. 43–54.
[6] H. Y. Shum and L. W. He, "Rendering with concentric mosaics," in *Proc. Computer Graphics (SIGGRAPH'97)*, Aug. 1999, pp. 299–306.
[7] R. Szeliski and H. Y. Shum, "Creating full view panoramic image mosaics and texturemapped models," in *Proc. Computer Graphics (SIGGRAPH'97)*, Aug. 1997, pp. 251–258.
[8] W. H. Leung and T. Chen, "Compression with mosaic prediction for image-based rendering applications," in *Proc. IEEE Int. Conf. Multimedia and Expo*, vol. 3, Jul. 2000, pp. 1649–1652.
[9] J. Li, H. Y. Shum, and Y. Q. Zhang, "On the compression of image based rendering scene," in *Proc. IEEE Int. Conf. Image Processing*, vol. 2, Sep. 2000, pp. 21–24.
[10] H. Y. Shum, K. T. Ng, and S. C. Chan, "Virtual reality using the concentric mosaic: Construction, rendering and data compression," in *Proc. IEEE Int. Conf. Image Processing*, vol. 3, Sep. 2000, pp. 644–647.
[11] C. Zhang and J. Li, "Compression of lumigraph with multiple reference frame (MRF) prediction and just-in-time rendering," in *Proc. IEEE Data Compression Conf.*, Snowbird, UT, Mar. 2000, pp. 254–263.
[12] M. Magnor and B. Girod, "Adaptive block-based light field coding," in *Proc. 3rd Int. Workshop Synthetic and Natural Hybrid Coding and Three-Dimensional Imaging*, Santorini, Greece, Sep. 1999, pp. 140–143.
[13] ——, "Model-based coding of multi-viewpoint imagery," in *Proc. SPIE Visual Communications Image Processing (VCIP'2000)*, vol. 4067, Perth, Australia, Jun. 2000, pp. 14–22.
[14] L. Luo, Y. Wu, J. Li, and Y. Q. Zhang, "Compression of concentric mosaic scenery with alignment and 3-D wavelet transform," in *Proc. SPIE Image and Video Communications and Processing*, San Jose, CA, Jan. 2000, Paper no. SPIE 3974–10.
[15] J. Foote and D. Kimber, "FlyCam: Practical panoramic video and automatic camera control," in *Proc. IEEE Int. Conf. Multimedia and Expo*, vol. 3, 2000, pp. 1419–1422.
[16] J. Baldwin, A. Basu, and H. Zhang, "Panoramic video with predictive windows for telepresence applications," in *Proc. IEEE Int. Conf. Robotics and Automation*, vol. 3, 1999, pp. 1922–1927.
[17] T. Boult, "Remote reality demonstration," in *Proc. Conf. Computer Vision Pattern Recognition*, 1998, pp. 966–967.
[18] Be Here Technologies [Online]. Available: http://www.behere.com
[19] iMove Inc. [Online]. Available: http://www.imoveinc.com
[20] A. Dan, D. Sitaram, and P. Shahabuddin, "Scheduling policies for an on-demand video server with batching," in *Proc. ACM Multimedia*, San Francisco, CA, Oct. 1994, pp. 15–23.
[21] S. Sheu, K. A. Hua, and W. Tavanapong, "Chaining: A generalized batching technique for video-on-demand systems," in *Proc. IEEE Int. Conf. Multimedia Computing and Systems*, Jun. 1997, pp. 110–117.
[22] W. Liao and V. O. K. Li, "The split and merge protocol for interactive video-on-demand," *IEEE Multimedia*, vol. 4, pp. 51–62, Oct.-Dec. 1997.
[23] K. A. Hua, Y. Cai, and S. Sheu, "Patching: A multicast technique for true video-on-demand services," in *Proc. ACM Multimedia*, New York, Sep. 14–16, 1998, pp. 191–200.
[24] S. Viswanathan and T. Imielinski, "Pyramid broadcasting for video on demand service," in *Proc. IEEE Multimedia Computing and Networking Conf.*, San Jose, CA, 1995, pp. 66–77.
[25] ——, "Metropolitan area video-on-demand service using pyramid broadcasting," *Multimedia Syst.*, vol. 4, pp. 197–208, Aug. 1996.
[26] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "A permutation-based pyramid broadcasting scheme for video-on-demand systems," in *Proc. IEEE Int. Conf. Multimedia Systems*, Hiroshima, Japan, Jun. 1996, pp. 118–126.
[27] K. A. Hua and S. Sheu, "Skyscraper broadcasting: A new broadcasting scheme for metropolitan video-on-demand systems," in *Proc. SIGCOMM*, Cannes, France, Sep. 1997, pp. 89–100.
[28] L. Juhn and L. Tseng, "Harmonic broadcasting for video-on-demand service," *IEEE Trans. Broadcast.*, vol. 43, no. 3, pp. 268–271, Sep. 1997.
[29] L. S. Juhn and L. M. Tseng, "Enhanced harmonic data broadcasting and receiving scheme for popular video service," *IEEE Trans. Consumer Electron.*, vol. 44, pp. 343–346, 5 1998.
[30] C. C. Aggarwal, J. L. Wolf, and P. S. Yu, "Design and analysis of permutation-based pyramid broadcasting," *ACM Multimedia Syst.*, vol. 7, no. 6, pp. 439–448, 1999.
[31] Z. Y. Yang, L. S. Juhn, and L. M. Tseng, "On optimal broadcasting scheme for popular video service," *IEEE Trans. Broadcast.*, vol. 45, no. 3, pp. 313–322, Sep. 1999.
[32] R. Szeliski, "Video mosaics for virtual environments," *IEEE Comput. Graph. Applicat.*, vol. 16, no. 2, pp. 22–30, Mar. 1996.
[33] C. Geyer and K. Daniilidis, "Omnidirectional video," *Vis. Comput.*, vol. 19, no. 9, pp. 405–416, 2002.
[34] N. Greene, "Environment mapping and other applications of world projections," *IEEE Comput. Graph. Applicat.*, vol. 6, no. 11, pp. 21–29, Nov. 1986.
[35] S. B. Kang, "Catadioptric self-calibration," in *Conf. Computer Vision and Pattern Recognition*, vol. 1, 2000, pp. 201–207.

[36] S. Nayar, "Catadioptric omnidirectional camera," in *Conf. Computer Vision and Pattern Recognition*, 1997, pp. 482–488.
[37] "Generic Coding of Moving Pictures and Associated Audio Information: Video,", ITU-T Rec. H.262-ISO/IEC 13 818-2, 1994.
[38] Microsoft document, "Creating compressed textures," in *DirectX SDK Documentation*: MSDN Library.
[39] [Online]. Available: http://www.moonlight.co.il
[40] K. K. Chu and H. C. Lau, "The transmission of multimedia object over high speed LAN," Dept. Elect. Electron. Eng., Univ. Hong Kong, Hong Kong, 1998.
[41] C. Granheit, A. Smolic, and T. Wiegand, "Efficient representation and interactive streaming of high-resolution panoramic views," in *Proc. IEEE Int. Conf. Image Process.*, vol. 3, Sep. 2002, pp. 209–212.

King-To Ng (S'96–M'03) received the B.Eng. degree in computer engineering from the City University of Hong Kong, Hong Kong, in 1994, and the M.Phil. and Ph.D. degrees in electrical and electronic engineering from the University of Hong Kong, in 1998 and 2003, respectively.

In 2004, he worked as a Visiting Associate Researcher at Microsoft Research Asia, Beijing, China. Currently, he is a Postdoctoral Fellow in the Department of Electrical and Electronic Engineering, University of Hong Kong. His research interests include visual communication, image-based rendering, and video broadcast and transmission.

S. C. Chan (S'87–M'92) received the B.Sc.Eng. and Ph.D. degrees from the University of Hong Kong, Hong Kong, in 1986 and 1992, respectively.

He joined the City Polytechnic of Hong Kong, in 1990 as an Assistant Lecturer and later as a University Lecturer. Since 1994, he has been with the Department of Electrical and Electronic Engineering, University of Hong Kong, and is now an Associate Professor. He was a Visiting Researcher at Microsoft Corporation, Redmond, WA, and at Microsoft China, Beijing, in 1998 and 1999, respectively. His research interests include fast transform algorithms, filter design and realization, multirate signal processing, communications signal processing, and image-based rendering.

Dr. Chan is currently a member of the Digital Signal Processing Technical Committee of the IEEE Circuits and Systems Society. He was Chairman of the IEEE Hong Kong Chapter of Signal Processing from 2000 to 2002.

Heung-Yeung Shum (SM'01) received the Ph.D. degree in robotics from the School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, in 1996.

He worked as a Researcher for three years in the Vision Technology Group at Microsoft Research, Redmond, WA. In 1999, he moved to Microsoft Research Asia, Beijing, China, where he is currently a Senior Researcher and the Assistant Managing Director. His research interests include computer vision, computer graphics, human computer interaction, multimedia systems, pattern recognition, statistical learning, and robotics.

Dr. Shum was the General Co-Chair of 9th International Conference on Computer Vision (ICCV), Beijing, 2003. He currently serves as an Associate Editor for IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS FOR VIDEO TECHNOLOGY.