On Interactive Browsing of Large Images

Jin Li, *Senior Member, IEEE,* and Hong-Hui Sun

***Abstract*—A new effective mechanism is proposed for the browsing of large compressed images over the Internet. The image is compressed with the JPEG 2000 into one single bitstream and put on the server. During the browsing process, the user specifies a region of interest (ROI) with certain spatial and resolution constraint. The browser only downloads the portion of the compressed bitstream that covers the current ROI, and the download is performed in a progressive fashion so that a coarse view of the ROI can be rendered very quickly and then gradually refined as more and more bitstream arrives. In the case of the switch of ROI, e.g., zooming in/out or panning around, the browser uses existing compressed bitstream in cache to quickly render a coarse view of the new ROI, and in the same time, request a new set of compressed bitstream corresponding to the updated view. The system greatly improves the experience of browsing large images over the slow networks.**

***Index Terms*—Cache, interactive browsing, JPEG 2000, media access, Virtual media (Vmedia), streaming.**

I. Introduction

THERE is an old saying that an image is worth a thousand words. Image is an increasingly popular media on the Internet. High resolution image of the deep space is provided on the NASA site. Microsoft terra server allows you to check the satellite/aero photographs of your neighborhood through the Internet. Attached image in an auction on the Ebay convinces the buyer the validity of the product, and enables a transaction between two previously unknown parties. More and more people window shop over the Internet by checking image of the merchandise they want to buy. Museums are digitizing the collectibles and moving them online. Even individuals are sharing personal photos over the web. Watching a high quality high resolution image is an enjoyable experience; however it is painful to download the huge compressed bitstream of the image, especially if the Internet connection is slow.

The primary barrier caused by the slow download is the large compressed bitstream of the image. It is true that the Internet environment keeps improving: the speed of the server and client computer is faster and faster, efficient cache and delivery systems, such as the proxy server, are gradually installed, and more and more people are having high bandwidth connections. Moreover, new image compression algorithms have been developed with better and better compression efficiency, culminated in the recent JPEG 2000 image coding standard [2]–[4]. However, in the meantime, the number of images and the resolution of the image grow as well. New image types, e.g., the 360° panorama surrounded image, also lead to effectively large image size. Efficient delivery of large compressed image is crucial to provide enjoyable experiences on the Internet.

In the early days, compressed images were downloaded entirely from the network before its content was rendered. A long delay was inevitable in such a first download then render model. Newer version of the browser supports progressive JPEG [1] streaming. By streaming we mean that the image browser does not wait for the entire compressed bitstream to arrive. Instead, a coarse quality view is rendered as soon as a head portion of the bitstream is available. The view is then refined and rendered repeatedly as more and more bitstream arrives. Progressive JPEG streaming improves the experience of image browsing over the Internet. However, if the image is of high resolution, the size of the compressed bitstream can still be too large to be delivered efficiently.

A better way to browse a large image is to use the interactive browsing. That is, the user interacts with the browser and specifies the interested region and resolution. One may choose an overview covering the entire image at low resolution, or may choose a small area at high resolution. We call the current viewable image region with both spatial and resolution constraint as a region of interest (ROI). The user may change the ROI by panning around (changing the spatial access region), and/or zooming in and out of the image (changing the access resolution). Only the bitstream related to the ROI needs to be delivered to the client, this reduces the bandwidth required for the image browsing. The interactive browsing is highly useful in case that the image resolution is higher than the display resolution, such as in the handheld device or the browsing of large aero photos. It is also ideal when the viewing window covers only a portion of the image, such as in the browsing of a 360° panorama.

There are several ways to achieve the interactive browsing experience. The server may do all the work, and after receiving the request of the ROI, it may compress the ROI image and deliver a dedicated ROI bitstream to the client. However, this approach is very computationally intensive on the server side, and may limit the amount of the clients that a server can handle. It is much more efficient if we can store a master compressed bitstream of the image at the server, and per each ROI request, extract the compressed bitstream of the ROI as a subset of the master bitstream. A sample of the ROI interactive browsing is the Flashpix [5], [6]. The Flashpix generates a multiresolution view of the original image, where each resolution image is further segmented into fixed tiles at size 64×64 and compressed independently with standard JPEG. According to the ROI's request, the Flashpix browser locates the tiles needed for the ROI and downloads only the compressed bitstream of the ROI. With

Manuscript received December 14, 2000; revised July 8, 2002. The associate editor coordinating the review of this paper and approving it for publication was Prof. Shih-Fu Chang.

J. Li is with the Communication, Collaboration, and Signal Processing Group, Microsoft Research, Redmond, WA 98052 USA.

H.H. Sun is with Microsoft Research Asia, Beijing, China.

Digital Object Identifier 10.1109/TMM.2003.813284

the ROI concept, Flashpix greatly speeds up the browsing of the large image. However, there are shortcomings. Flashpix needs to store multiple resolution images at the server, which increases the amount of information needs to be stored at the server side. There is no relationship between bitstreams at different resolutions. Therefore, the response time of the browser is slow when switching from one resolution to another. Each ROI is browsed in a nonprogressive format. If the ROI becomes larger, the time for retrieving the ROI bitstream can still be significant.

In this work, a new interactive image browsing system is developed. The proposed browser is far superior to the FlashPix, in the sense that it not only supports the ROI functionality, but delivers the ROI bitstream in a prioritized way and manages the delivered ROI bitstream. We compress the image with JPEG 2000, which not only significantly outperforms the JPEG and Flashpix in terms of the compression ratio, but also offers a modular bitstream. We implement for the first time, the decoder region of interest access capability of the JPEG 2000. Once the request of the ROI is received, we locate a set of bitstream segments from the compressed master bitstream of the JPEG 2000. Only the compressed bitstream segments necessary to decode the current ROI are accessed and streamed over the Internet; and they are streamed in a prioritized fashion so that a low quality view of the current ROI can be rendered very quickly after the connection is established. As more and more bitstream segments arrive, the quality of the rendered ROI improves. Vmedia—a virtual media access protocol, is developed to support the access, cache and management of segments of the compressed bitstream. After all bitstream segments of the ROI arrived, the decoded ROI is exactly the same as if we decode the entire image and cut out the ROI.

The paper is organized as follows. We describe the Vmedia protocol that supports the cache and delivery of the bitstream segments in Section II. The implementation of the JPEG 2000 interactive image browser is described in Section III. Section IV and Section V show the experimental results and the conclusion, respectively.

II. Virtual Media Access Protocol

A. Framework

Vmedia stands for the virtual media access protocol. It is a protocol we developed to support interactive media browsing through access to a portion of the media. Though used specifically for the JPEG 2000 image browsing, Vmedia is useful for many other Internet browsing applications as well. One of such samples is the browsing of three-dimensional (3-D) environment [11], in which a 3-D scene is compressed into a master bitstream, and depending on the viewing angle and viewing point, a small portion of the compressed bitstream necessary to recover the current view is accessed through the Vmedia, decoded, and rendered. In either the interactive image browsing or the interactive 3-D environment browsing, the compressed media is stored as a master bitstream at the server side. Depending on the current ROI (in the case of image browsing) or the current viewing angle and viewing point (in the case of 3-D environment browsing), a subset of the compressed bitstream is accessed, decoded and rendered. Vmedia protocol is developed to facilitate such access.

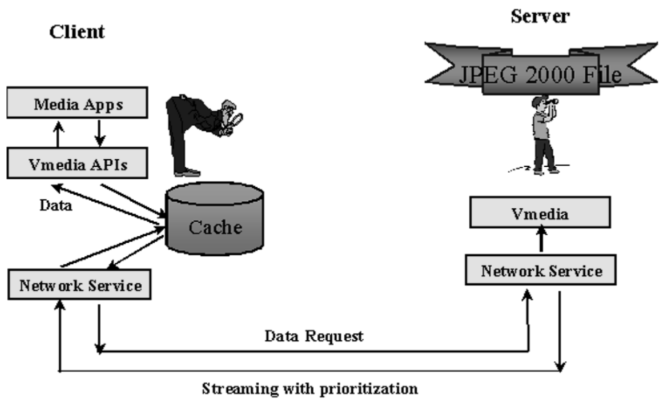


Fig. 1. Dataflow of a typical Vmedia application.

The framework of the Vmedia is shown in Fig. 1. There are a server and a client component in Vmedia. 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 virtual media), and manages it with a local cache. The virtual local copy is exactly the same in structure as the remote media. However, only a portion of the remote media may be in cache and directly accessible. Through the Vmedia client, the application can access segments of media bitstream with start position *pos* and length l. Vmedia checks if the media segment is already in cache. If it is, the requested segment 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.

To support interactive browsing, Vmedia provides the following functionalities:

1) *Flexible media access:* Vmedia enables the access of the remote media in the form of bitstream segment with start position *pos* and length l.
2) *Cache 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 implementation of the file segment cache in existing protocols.
3) *Prioritized delivery:* A compressed media bitstream is usually consisted of segments with different priorities. In the case of JPEG 2000 image browsing, the compressed bitstream segments associated with the first few quality layers are of higher priority and should be delivered first.
4) *Media segment packaging:* Small media segments should be automatically packaged into a larger one, and a large media segment should be broken into several small packets according to the maximum transmission unit (MTU) of the network path. Packet loss and retransmission also need to be handled.

Through Vmedia, the media application simply accesses remote media as a virtual local file. We explain the typical workflow of Vmedia in Section II-B. The two key components of the Vmedia, the Vmedia cache and the media segment delivery are explained in Section II-C and II-D, respectively.

B. Dataflow

We show in Fig. 1 the dataflow of a sample application using Vmedia. 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 (the JPEG 2000 file and packet header in the case of interactive image browsing) is downloaded to the Vmedia client in the form of a companion file. The Vmedia cache is also initialized to be empty at the connection phase. After that, segments of media, indicated by the start position *pos* and length l, can be accessed. The access request is attached with a priority tag, an importance tag and an optional lock, which are used to prioritize streaming and aide cache management. For each access request, Vmedia first checks whether the requested segment is in the Vmedia cache. If the entire segment is in 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 generated. We do not send the request immediately over the network because there may be subsequent requests of higher priority. Moreover, a number of requests may be packed and sent together.

Vmedia works in an asynchronous mode. The media segment request received by the Vmedia client is served on a best effort basis and the Vmedia client 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 coarse view of the ROI may be quickly rendered with a minimum of arriving data. The quality of the view can then be gradually improved as more and more media segments are received.

When the connection is closed, the current content of the Vmedia cache can be optionally dumped to a permanent file, so that 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[1] and Netscape[2], though instead of an entire file, segments of a file are dumped.

C. Vmedia Cache

A Vmedia 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 memory allocation.*

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

There are several approaches to implement Vmedia cache management. 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. Though simple, this approach is memory consuming and cannot browse media larger than the memory capacity. Another approach is to allocate memory for each media segment received, with arbitrary length and start point. The approach may severely fragment the memory and may also be slow to detect whether a particular segment is already in the cache.

Our implementation of the Vmedia cache management uses fixed size unit to improve the speed of cache hit detection and reduces the memory overhead needed to hold the Vmedia cache. An arbitrary positioned media segment is round up into a set of submedia unit (SMU), which is the minimum unit for delivery and cache management. SMU has a fixed size of K bytes, and aligns with K byte boundaries. L SMUs form a media unit (MU), which is the minimum unit for cache memory allocation. In the current implementation, $K = L = 32$. The MU and SMU structure can be shown in Fig. 2. MU maintains tags for cache management; while SMU only carries a validity bit. The memory overhead introduced by the MU/SMU is only 10 bytes (4 bytes for validity, 1 byte for importance, 2 bytes for hit count, 1 byte for lock, 2 bytes for table index) per 1024 byte MU, which is about 1%.

With the MU/SMU structure, the cache hit detection operation can be performed very quickly. The media segment is broken into SMUs, which are further grouped by the MUs. The validity of the MU containing the SMU can be checked by the lookup table. A subsequent bit-wise AND operation can then be used to further validate the SMUs within the MU. The cache memory allocation operation can also be easily performed. Whenever a new media segment is received, we check if the MU associated with the media segment is allocated. If it is not, memory is allocated for the MU. Then, the validity bits of the delivered SMUs are turned on. In case all cache memory is used up, some less frequently used and less important MUs are swapped out to make room for the newly arrived media segments.

As shown in Fig. 2, each MU maintains one importance, one hit count and one lock tag. There is also a validity bit for each SMU. When calling the Vmedia client, the application specifies the importance and the priority of the media segment. The priority is used for media delivery, while the importance is used for cache management. A highly important MU is less likely to be thrown away in case that the cache memory is used up. If a MU is associated with several importance values through different function calls, the highest importance value among all calls is assigned to the MU. The application may temporarily lock the MUs to ensure that certain media segments are not thrown away. The lock is frequently used on the media segments associated

[1]Internet Explorer is a trademark of Microsoft Corporation, Redmond, WA.

[2]Netscape is a trademark of Netscape Communications Corporation, Mountain View, CA.

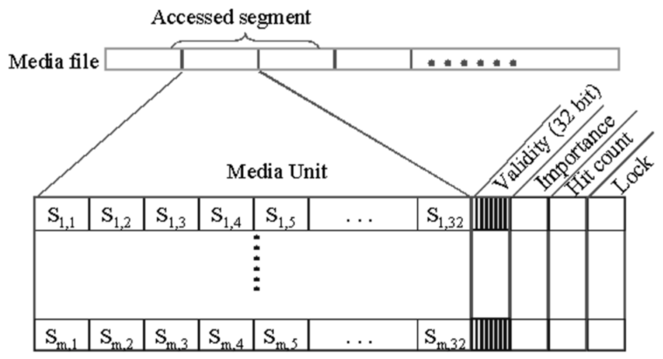

Accessed segment
Media file
Media Unit
Validity (32 bit)
Importance
Hit count
Lock
$S_{1,1}$ $S_{1,2}$ $S_{1,3}$ $S_{1,4}$ $S_{1,5}$ ... $S_{1,32}$
$S_{m,1}$ $S_{m,2}$ $S_{m,3}$ $S_{m,4}$ $S_{m,5}$ ... $S_{m,32}$

Fig. 2. Media unit (MU) and submedia unit (SMU).

with the current ROI to ensure the bitstream segments of the current view is not accidentally swapped out. We refer to Section III-B for an example of the lock usage. In case that the ROI 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.

D. *Delivery of Media Segments*

When the application calls the Vmedia client to request the media segment, a priority tag is attached to the request and is used to prioritize the media delivery. The media segment with a higher priority value is delivered earlier over the network. Vmedia maintains two categories of queues: pending queue where the request is issued by the application but has 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 received, 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 associated request is cleared from the sent queue. Vmedia may or may not report the arrival of the media segment to the application through callback function. A good strategy is to report to the application 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 and the associated timer of the request runs out, the request will be reinserted into the head of the pending queue for redelivery. 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, the application sends a clear function call to the Vmedia client, which clears 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. Canceling 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 [10]. Alternatively, a special Vmedia server can be designed. By reducing the network overhead associated with the request and the returned media segment, Vmedia server can be much more efficient in the delivery of media segments. The functionality of the Vmedia server is rather simple: it handles media access requests coming from the Vmedia client. Since prioritization and caching are all handled by the Vmedia client, the load of the Vmedia server is low. The dedicated communication between the Vmedia server and client is through the UDP protocol. Vmedia client bundles multiple small requests into a single UDP packet and sent it to the server. Alternatively, the Vmedia 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.

III. Interactive Browsing of JPEG 2000 Compressed Image

A. *JPEG 2000*

Established in December 2000, JPEG 2000 is the current state-of-the-art in still image compression [2]–[4]. We first briefly review the framework of the JPEG 2000. JPEG 2000 is a wavelet image compression algorithm. First, the image is transformed from space domain into the wavelet domain with a multiresolution wavelet decomposition. The wavelet transform compacts the energy of the image into a few coefficients so that they may be efficiently compressed by the following entropy coding step. Moreover, the multiresolution structure of the decompositions enables spatial scalability, so that by throwing away the high resolution coefficients and their associated compressed bitstream along with decoding only the low resolution coefficient bitstream, a lower resolution view of the image can be recovered. Each resolution subband is segmented into fixed size coefficient blocks. The default block size is 64×64, however, other block size may be used as well. The block partition enables spatial locality, so a view of a portion of the image can be decoded by only accessing the coefficient block covering the portion. An example of the wavelet transform and block partition is shown in Fig. 3, where the 512×512 Lena image is decomposed by a three-level wavelet transform. There are 16 coefficient blocks per subband in resolution 3 (the finest resolution), four coefficient blocks per subband in resolution 2, and one coefficient block per subband in resolution 1 (the coarsest resolution).

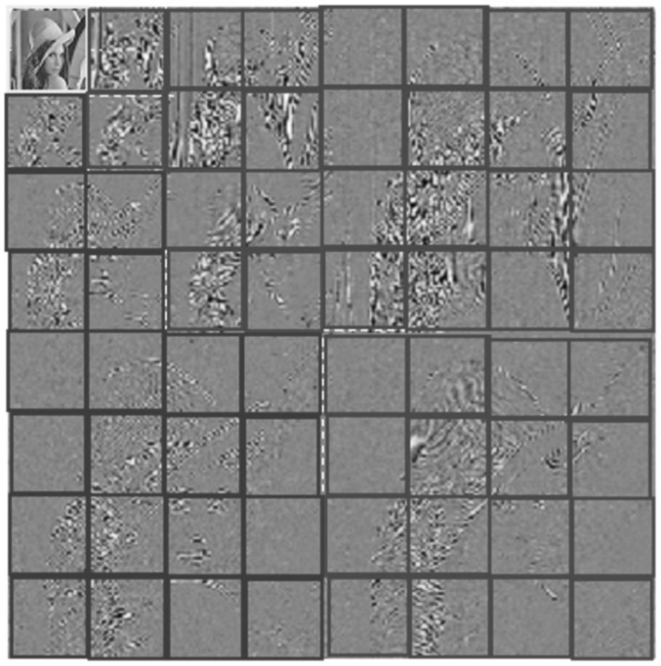


Fig. 3. Wavelet transform and block partition in JPEG 2000. The sample image is Lena, with a three-level wavelet transform.

Each block is then separately encoded by a block coder termed embedded block coding with optimized truncation (EBCOT) [7], where the more important information is placed at the head of the bitstream, and the less important information is placed at the tail of the bitstream. This way, the bitstream can be truncated at any place and still be decoded, albeit with lower quality. Since in such bitstream organization, the lower rate bitstream is embedded in the higher rate bitstream, the bitstream is called to have the embedding property. It should be noted that the current state-of-the-art embedded coding achieves the embedding property without sacrifice of the compression performance, i.e., the decoding quality of the block of the truncated bitstream is very close to optimally coding the block directly to the truncation bitrate.

The coded bitstreams of all coefficient blocks are then assembled together into a JPEG 2000 compressed file. The embedded bitstream of each block is segmented into a number of quality layers, 1 to L. The guideline for the segmentation process is that at each segmentation point of layer l, the rate-distortion (R-D) slope (distortion decrease per bit spent) of all blocks should be the same, whatever the spatial location and resolution level that the block is located. All block bitstreams of the same quality layer l and resolution j are assembled to form a packet of resolution j and quality layer l. Attached to each packet is a packet header, which records the length (number of bytes) of the chopped block bitstream for each coefficient block. In the final JPEG 2000 bitstream, the packets are ordered first by quality layer, and then by resolution. If there are multiple components (color image) and multiple tiles (for huge image), the packets are ordered first by tile, then by quality layer, next by resolution, and finally by components. When such bitstream is truncated, the bitstream of the first few quality layers of all coefficient blocks are retained, we may thus decode all blocks, and thus the entire image, albeit at a lower quality. The JPEG 2000 compressed bitstream contains a file header, which stores global information such as the image size, the method of transform and coding, the number of color components and tiles, the bit depth of the image, and the author of the image. The syntax of a JPEG 2000 bitstream can be illustrated in Fig. 4. For a more detailed description of the JPEG 2000 bitstream, we refer to document [7, Section X].

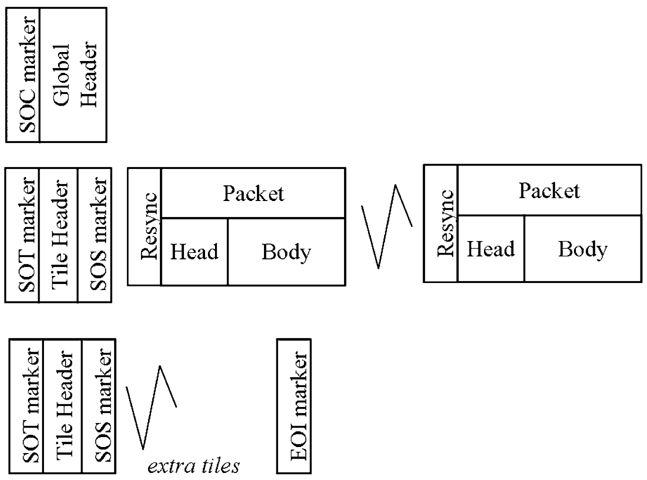
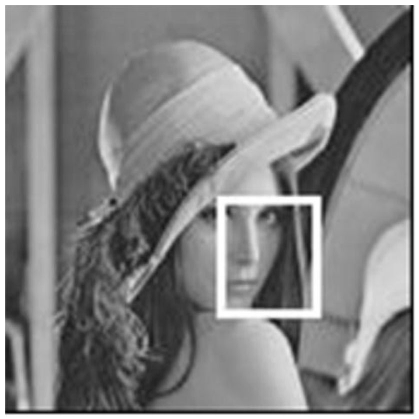


Fig. 4. JPEG 2000 file structure.

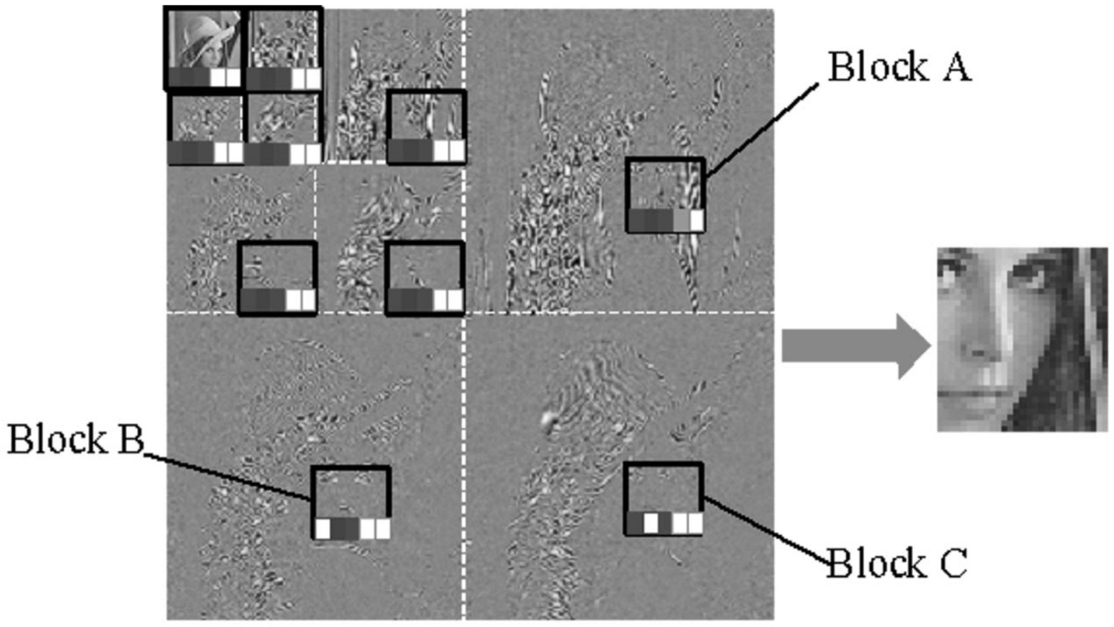
B. Interactive JPEG 2000 Image Browsing With Vmedia

The bitstream syntax of the JPEG 2000 provides the possibility to decode a region of interest (ROI) with both spatial and resolution constraint without accessing and decoding the entire JPEG 2000 compressed bitstream. However, such feature is not implemented with the existing JPEG 2000 verification software [9]. We are the first to implement the decoder ROI access functionality of the JPEG 2000. Based on that, a JPEG 2000 interactive image browser is further developed. In the interactive image browser, the user specifies the current region of interest (ROI), which may be the entire image at low resolution, or a detailed region at high resolution. The user may change the ROI interactively by panning around and zooming in/out of the image. The image browser responds by accessing the bitstream segments covering the current ROI, streaming them from the Vmedia server if they are not available in cache, decoding the bitstream and rendering the current ROI.

The workflow of the JPEG 2000 interactive image browser is described below. For easy discussion, we describe the operation of the image browser with two examples. The image used in the example is a JPEG 2000 compressed 512×512 Lena image with three-level wavelet decomposition and 64×64 coefficient block (shown in Fig. 3). We assume that each block bitstream is chopped into five quality layers. We also assume that the image is of a single tile and a single color component (gray image). However, the discussion can be easily extended to image with multiple color components and tiles. For the color image, we simply access and decode each component independently. For the multiple-tile image, each tile is treated as an independent image, accessed and decoded independently.



(a)



(b)

Fig. 5. (a) Sample region of interest (ROI) of Vmedia image browsing. The Lena image is of size 512 × 512. The ROI is at (256, 256) with size (128, 128). (b) Accessed coefficient blocks. The black box identifies the coefficient blocks accessed by the Vmedia. A shaded bar identifies an available bitstream segment, and a white bar identifies an unavailable bitstream segment.

After the Lena image is compressed with the JPEG 2000, the compressed bitstream is parsed and the file header and packet header are identified and stored in a companion file. Both the compressed bitstream and the companion file are put on the Vmedia server. The functionality of the companion file is to store the media structure information so that it can be quickly identified and streamed. In the beginning of the browsing process, the companion file is downloaded to the Vmedia client. By decoding the companion file, the Vmedia image browser is able to locate the position of the bitstream segment for every quality layer of every coefficient block. Without the companion file, the Vmedia image browser has to first stream the file header from the server, then decode the file header to locate and stream the packet header, only after that, the interactive browsing can be performed. The companion file eliminates the need of several round-trip communications between the client and server, thus greatly improves the initial response time in the wide area network.

In the first ROI browsing example, the ROI is of size 128 × 128, with its upper left corner located at (256, 256), as shown in Fig. 5(a). A total of ten wavelet coefficient blocks need to be accessed to decode the ROI. The accessed coefficient blocks include all the blocks of resolution 1, the lower right blocks of resolution 2, and the row 3 and column 3 blocks of resolution 3, for all directional subbands. They are marked by black boxes and shown in Fig. 5(b). Since each block bitstream has five quality layers, a total of 50 bitstream segments are accessed by the interactive image browser. We call those bitstream segments as the ones that covering the current ROI. The interactive image browser accesses the bitstream segments with different priority and importance. We order the delivery priority first with the quality layer, and then with the resolution. Thus the bitstream segment of earlier quality layer and coarser resolution level will be delivered in higher priority. We also assign a higher importance value to coarser resolution block bitstreams, so that in case the cache memory is tight, the coarser resolution block is more likely to stay in cache for later view. All bitstream segments covering the current ROI are also locked in cache, which makes sure that the memory assigned to these bitstream segments are not released.

Since the Vmedia immediately returns with the available bitstream segments that are in cache, the interactive image browser may proceed to render a coarse view of the ROI with the available compressed bitstream. All bitstream segments of the same coefficient block are put together. Since each block is embedded encoded, it can be decoded up to the last segmentation point of the available bitstream. Occasionally, a higher quality layer bitstream segment of a certain block is available, while the lower quality layer bitstream segment of the same coefficient block is not because of the loss, transmission jitter or corruption of the lower quality layer bitstream segment or its request sent to the server. In such a case, the existing higher quality layer bitstream segment is discarded because it cannot be used in the decoding process. The embedded coder can decode a truncated bitstream, it cannot, however, utilize part of the bitstream after a missing segment.

Back to our example in Fig. 5(b), at a certain instance, most of the quality layer 1, 2, and 3 bitstream segments of the ten accessed blocks have arrived. Quality layer 4 bitstream segment has arrived for block A, while quality layer 1 and 2 bitstream segments are missing for block B and C, respectively. Block B is decoded as all zeros because its first bitstream segment is not available. The browser may decode block A to quality layer 4, block C to quality layer 1, and the rest blocks to quality layer 3. After all coefficient blocks covering the ROI are decoded, the coefficient blocks are inversely quantized and inversely wavelet transformed. The resultant ROI is then rendered on the browser device. It is not the full quality image of the current ROI, as some bitstream segments of the ROI have not arrived yet. Nevertheless, it is the best available view of the ROI based on the received bitstream segments.

The above process of the bitstream segment accessing, assembling, decoding, inverse quantization, inverse wavelet transform and rendering is repeated again and again by the interactive image browser. With each repetition, more bitstream segments are received from the Vmedia server, and the quality of the ROI becomes better and better.

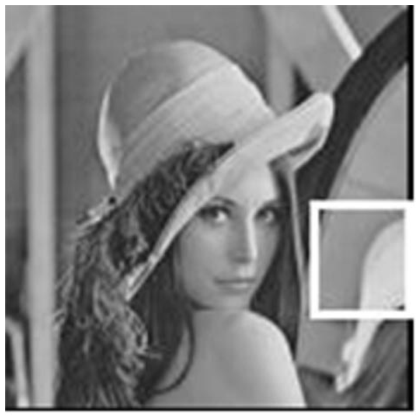
The user may interact with the browser and change the ROI. Suppose right after the moment of Fig. 5, the user pans right to view the ROI with upper left corner at (384, 256), as shown in Fig. 6(a). Receiving the change of ROI request, the interactive image browser first unlocks all bitstream segments locked in the Vmedia cache during the visit of the last ROI. A clear function call is also issued to cancel all requests in the pending queue of the Vmedia client and the requests pending execution at the Vmedia server, so that the network bandwidth can be freed for the delivery of the bitstream segments of the new ROI. The coefficient blocks corresponding to the new ROI and their bitstream segments are located and accessed. In this example, the coefficient blocks accessed in the resolution 1 and 2 are exactly the same as those last accessed, only the coefficient blocks accessed in resolution 3 are different, as the row 3 and column 4 blocks are accessed for all directional subbands of resolution 3, as shown in Fig. 6(b). The browser again accesses all bitstream segments associated with the coefficient blocks through the Vmedia. In the first iteration after the switch of ROI, no bitstream segments of resolution 3 blocks are available. However, the quality layer 1, 2, and 3 bitstream segments of resolution 1 and 2 coefficient blocks are already in the Vmedia cache and can be accessed. The browser can thus decode resolution 1 and 2 coefficient blocks up to quality layer 3, and fill in zeros for resolution 3 coefficient blocks. After inverse quantization and inverse wavelet transform, a somewhat blurred view of the ROI (due to the loss of one resolution) is shown. The rest of the bitstream segments are then gradually streamed from the Vmedia server. First, the bitstream segments of the quality layer 1 resolution 3 are streamed, then those of the quality layer 2, after that those of the quality layer 3. As the first few quality layer bitstream segments of the resolution 3 coefficient blocks arrive, the resolution of ROI improves quickly. The Vmedia image browser again locks all bitstream segments associated with the current ROI to secure the cache memory. After quality layer 3 bitstream segments for all coefficient blocks have arrived, the quality layer 4 bitstream segments are streamed, first for resolution 1, then for resolution 2 and resolution 3. The quality of the ROI keeps improving as more and more bitstream segments arrive.

IV. Experimental Results

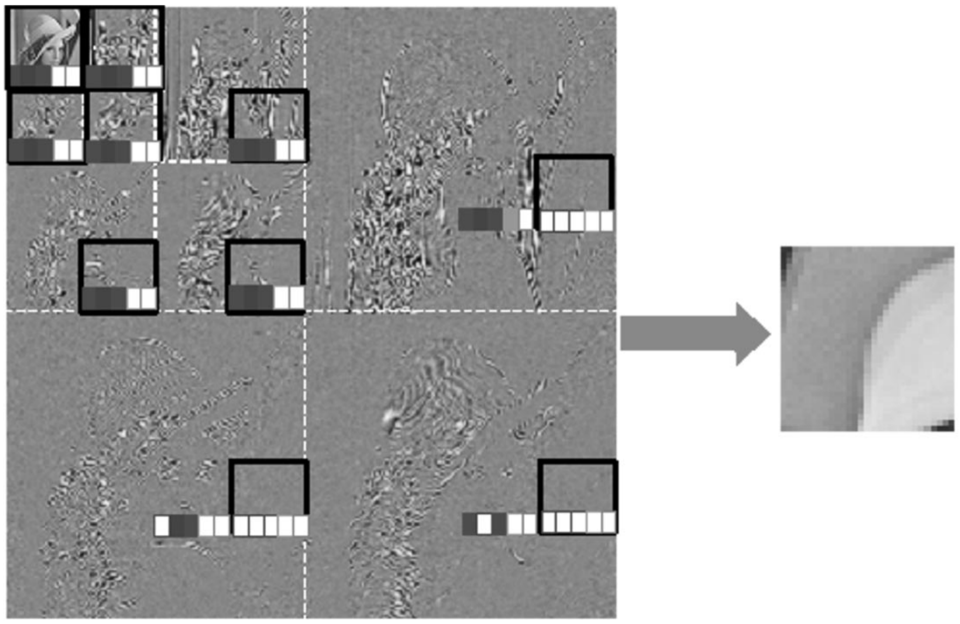
The interactive image browser can either be used as an ActiveX plug-in embedded in a web page, or be implemented as a stand-alone application. We show the use of the interactive image browser in web in Fig. 7. The browsed image is Aerial1, the largest image in the standard JPEG 2000 test set. The original image is of size $14\,565 \times 14\,680$. It is compressed to 1.0 bpp or 26 101 KB. The interactive browse window is of size 1020×630.

We evaluate the performance of the interactive image browser, especially on the slow connection link. To prepare the images for the interactive browsing application, they are encoded by JPEG 2000 VM 5.2 [9]. Seven images (shown in Table I), including five gray images and two color images from the JPEG 2000 standard test set are encoded. All images except Aerial1 are decomposed by a five-level biorthogonal 9–7 tap wavelet transform, compressed to a final bit rate of 1.0 bpp (bit per pixel), with the bitrate of the intermediate quality layers set at 0.0625, 0.125, 0.25, 0.375, 0.5, and 0.75 bpp (a total seven quality layers). The image Aerial1 is decomposed by a seven-level wavelet transform, with all the other coding parameters the same as above. The JPEG 2000 bitstream is then parsed and the file header and packet header are recorded in a companion file. Both the JPEG 2000 compressed bitstream and the companion files are put on a Vmedia server. The Vmedia interactive image browser is then used to interactively browse the image over a modem link at 33.6 kbps.

It takes a long delay (from 2.5 min to 103.5 min) if we first download the compressed bitstream and then render. The delay is intolerable even if the bitstream is streamed to the client. However, with the Vmedia image browser, large image can be browsed quickly with little delay, even when the bandwidth is tight (33.6 kbps) and the compressed bitstream is huge (from 511 KB to 26 MB). The interactive image browser responds to our request very quickly. We can pan around (change the locality of the ROI), zoom in and out of the image (change the resolution of the ROI) with ease. A low quality blurred view of the current ROI is usually shown immediately after the change of ROI. The



(a)



(b)

Fig. 6. (a) ROI pans right to (384, 256). (b) Accessed coefficient blocks (black box), the available (shaded bar) and unavailable (white bar) bitstream segments of the ROI.

quality of the ROI also quickly improves, as the first few quality layer bitstream segments of the ROI can be streamed very fast even over a slow network. A visually lossless view of the current ROI is often reached within 10 seconds for a ROI of size 480×480.

In the following, we show some quantitative experiments on the speed of the interactive image browsing. We first investigate the overhead for the interactive browsing. Before the interactive browsing can be performed, we need to download the companion file, which contains the structure information of the JPEG 2000 bitstream. The size of the companion file and its percentage versus the total compressed bitstream are listed in columns 5 and 6 of Table I, respectively. The companion file ranges from 0.83% to 1.67% of the total compressed bitstream, with an average of roughly 1%. It is pretty compact considering that it provides index to all the bitstream segments. The streaming of the companion file takes 1–3 s for relatively small images such as Aerial2, Café, Cats, Bike, and Woman. It takes as long as 15 s for Cmpnd2 and 54 s for Aerial1 on a 33.6 kbps modem. We can cut the initial connection time by building an index over the structure information and stream first only the packet headers of coarse resolution layers. The other packet headers can then be streamed on demand.

We then investigate the size of the compressed bitstream covering a specific ROI, which is denoted as the bitstream covering size of the ROI. In Table II, we list the bitstream covering size (in kilobytes) for zoom out ratio from 1:1 to 16:1. The ROI is of size 480×480 and at the top-left corner of the image, i.e., coordinate (0, 0). Note that when we zoom out more than 8:1, the ROIs completely cover the smaller test images Aerial2, Café, Cats, Bike, and Woman. Therefore, the covering sizes are also the size of the compressed bitstream for the whole image at that resolution. It is observed that though the compressed bitstream of the image can be very large, the covering size of a ROI is usually small, ranging from 11 KB to 61 KB for the original resolution or 88 KB to 143 KB for zoom out ratio 4:1. In fact, the average covering size of an original resolution ROI is 33 KB (1.18 bpp), pretty close to the image compression bitrate. At coarser resolution, the covering size of a ROI increases. However, at such resolution, the high quality layer bitstream seg-

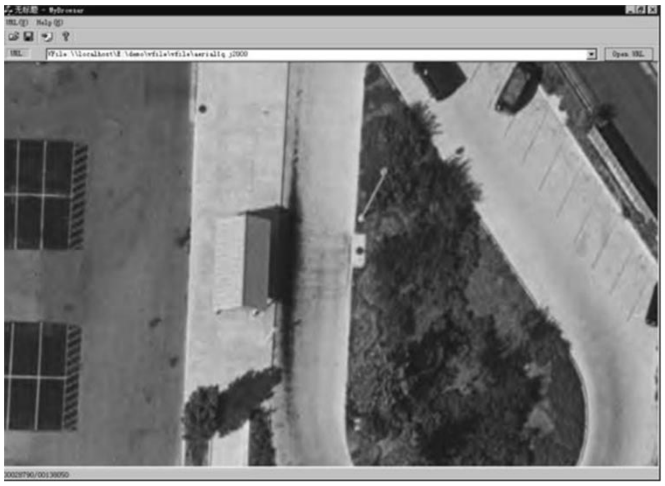


Fig. 7. Vmedia interactive image browser.

TABLE I
TEST IMAGE (COMPRESSED BY JPEG 2000 AT 1.0 bpp)

Image	Size	Format	Compressed size (KB)	Media structure	Overhead Percent
Aerial1	14565x14680	Gray	26,101	216	0.83%
Aerial2	2048x2048	Gray	511	5	0.97%
Café	2048x2560	Gray	640	8	1.21%
Cats	3072x2048	Gray	768	6	0.78%
Cmpnd2	5120x6624	Gray	4,139	61	1.47%
Bike	2048x2560	Color	640	11	1.67%
Woman	2048x2560	Color	640	9	1.30%

ments are for pixel depth beyond 8 bits, and are thus not sensitive for the visual quality improvement of the ROI. In fact, an intermediate resolution ROI achieves visual lossless when the size of the available bitstream segments reaches around 28–56 KB (1.0–2.0 bpp). Though the rest bitstream segments play an important role in the rendering of a finer resolution view, they are not visually critical for the rendering of the coarse resolution ROI.

In the third experiment, we investigate how many bitstream segments remain useful in cache after the switch of ROI. We continuously zoom in from 16:1 to 1:1, again with the ROI located at (0, 0) with window size 480×480. We define the size of the compressed bitstream segments that are available at the time of the resolution switch as the initial covering size. We first view the ROI at 16:1 resolution and wait for all its bitstream segments to arrive. We then zoom in to 8:1 and immediately record the initial covering size of the new ROI. The number is listed in column 2 of Table III. The same operation is repeated as we continuously zoom in until the original image resolution is reached. The initial covering sizes for resolution switches of 4:1, 2:1, and 1:1 are recorded in columns 3-5 of Table III. We observe that during the zoom in operation, roughly 80% of the bitstream segments are available in cache when we zoom in from 2:1 to the original image resolution. Alternatively speaking, on average only 8 KBs of bitstream segments are to be streamed for this type of ROI switch. It shows that the Vmedia cache is effective, and can greatly reduce the response time. The percentage of the initial versus total covering size decreases for the coarse resolution ROI, and becomes 47% if we zoom in from 4:1 to

TABLE II
BITSTREAM COVERING SIZE OF A 480×480 ROI AT DIFFERENT RESOLUTION LEVEL

Image	Covering size (KB) at zoom out rate				
	16:1	8:1	4:1	2:1	1:1(Original resolution)
Aerial1	236	189	143	95	43
Aerial2	13	44	134	83	37
Café	14	44	101	42	17
Cats	11	33	88	41	11
Cmpnd2	108	181	134	99	61
Bike	30	78	95	55	37
Woman	22	51	105	64	28

TABLE III
INITIAL COVERING SIZES WITH ZOOM IN OPERATION

Image	Initial covering size (KB) when the zoom ratio changes:			
	16:1 → 8:1	8:1 → 4:1	4:1 → 2:1	2:1 → 1:1
Aerial1	58	45	39	32
Aerial2	13	44	41	29
Café	14	34	17	14
Cats	11	26	14	11
Cmpnd2	58	53	53	42
Bike	30	58	33	28
Woman	22	41	33	24

2:1, 38% for 8:1 to 4:1, and 34% for 16:1 to 8:1. However, since the tailing bitstream segments do not contribute much to visual improvement, the actual ROI still clears up very quickly. In the zoom in experiment, we observe that the ROI usually reaches visual lossless within 4 s.

V. CONCLUSION

In this paper, an interactive image browser is developed. The user browses the image through a region of interest (ROI) with both spatial locality and resolution constraint. Using the modularity of the JPEG 2000 bitstream, only a subset of the compressed bitstream associated with the ROI needs to be accessed and delivered, which greatly reduced the amount of the compressed bitstream needs to be accessed. Through Vmedia, a virtual media protocol, the compressed bitstream of the ROI is delivered in a prioritized fashion, so that a coarse view of the image can be viewed quickly, and gradually refined as more and more bitstream segments arrive. Vmedia also caches and managed the delivered bitstream, so that when ROI changes, part of the delivered bitstream segments can be reused. Through the above mechanism, the proposed interactive image browser is able to efficiently browse a large compressed image (as large as 26 MB), even when the network connection speed is slow (33.6 kbps modem).

ACKNOWLEDGMENT

The authors would like to acknowledge the following individuals: Dr. X. Lin, for his contribution in securing the resource of the Vmedia project, Dr. Y.-Q. Zhang, for his managerial support and naming of Vmedia, H. Li, for codes of Vmedia client, Dr. Q. Zhang, for programming the network interface of the Vmedia client and server, and J. Zhou, for packaging the Vmedia image browser.

REFERENCES

[1] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*. New York: Van Nostrand Reinhold, 1993. ISBN 0-442-01 272-1.

[2] B. E. Usevitch, “A tutorial on modern lossy wavelet image compression: Foundations of JPEG 2000,” *IEEE Signal Processing Mag.*, vol. 18, pp. 22–35, Sept. 2001.

[3] A. Skodras, C. Christopoulos, and T. Ebrahimi, “The JPEG 2000 still image compression standard,” *IEEE Signal Processing Mag.*, vol. 18, pp. 36–58, Sept. 2001.

[4] M. W. Marcellin and D. S. Taubman, *Jpeg2000: Image Compression Fundamentals, Standards, and Practice*. Norwell, MA: Kluwer. International Series in Engineering and Computer Science, Sec. 642.

[5] Live Picture [Online]. Available: www.livepicture.com

[6] C. S. Jao and D. B. Hier, “The display of photographic-quality images on the web: A comparison of two technologies,” *IEEE Trans. Inform. Technol. Biomed.*, vol. 3, pp. 70–73, Mar. 1999.

[7] D. Taubman, “High performance scalable image compression with EBCOT,” *IEEE Trans. Image Processing*, vol. 9, pp. 1158–1170, July 2000.

[8] *JPEG 2000 Verification Model 5.2 (Technical Description)*, July 1999.

[9] *JPEG 2000 Verification Model 5.2 (Source Code)*, July 1999.

[10] *Hypertext Transfer Protocol—HTTP/1.1*, June 1999. W3C/MIT.

[11] C. Zhang and J. Li, “Interactive browsing of 3D environment over the Internet,” in *Proc. SPIE: Visual Communication and Image Processing (VCIP’2001)*, vol. 4310, San Jose, CA, Jan. 2001.

Jin Li (S’94–A’95–M’96–SM’99) received the B.S, M.S, and Ph.D. degrees in electrical engineering, all from Tsinghua University, Beijing, China, in 1990, 1991 and 1994, respectively.

From 1994 to 1996, he served as a Research Associate at the University of Southern California (USC), Los Angeles. From 1996 to 1999, he was a Member of Technical Staff at the Sharp Laboratories of America (SLA), Camas, WA, and represented the interests of SLA in the JPEG2000 and MPEG4 standardization efforts. He was a Researcher/Project Leader at Microsoft Research China, Beijing, from 1999 to 2000. He is currently a researcher at Microsoft Research, Redmond, WA. Since 2000, he has also served as an Adjunct Professor with the Electrical Engineering Department, Tsinghua University. He has published 15 journal papers and over 40 conference papers in a diversified research field, with interests cover image/video compression and communication, audio compression, virtual environment and graphic compression. He holds five issued U.S. patents, with many more pending. He is an Area Editor for the *Journal of Visual Communication and Image Representation* (Academic Press).

Dr. Li is an active contributor of the ISO JPEG 2000/MPEG4 project. He was the recipient of the 1994 Ph.D. thesis award from Tsinghua University and the 1998 Young Investigator Award from SPIE Visual Communication and Image Processing.

Hong-Hui Sun received the B.S. degree from Zhejiang University, Hangzhou, China, and the M.S. degree from Beijing University, Beijing, China.

He is a Senior Research Software Engineer at Microsoft Research Asia, Beijing. His interests include image and video processing/compression and network communication.