A Virtual Media (Vmedia) Access Protocol and Its Application in Interactive Image Browsing

Jin Li and Hong-Hui Sun

Microsoft Research China
3F, Sigma Center, 49 Zhichun Road, Haidian, Beijing 100080, P. R. China
Tel. + 86 10 6261 7711 – 5793, Email: jinl@microsoft.com

ABSTRACT

In this work, we propose a virtual media (Vmedia) access protocol for interactive image browsing over the Internet. Vmedia is a multimedia communication protocol in a server-client environment. It enables the access and delivery of segments of media, and manages the delivered media segments with a local Vmedia cache. An interactive JPEG 2000 image browser is implemented with Vmedia. The image is compressed with JPEG 2000 into one single bitstream and put on the server. In the interactive browsing process, the user specifies a region of interest (ROI) with a certain spatial and resolution constraint. The Vmedia browser only downloads the media segments covering 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 bits arrive. In the case of switch of ROI, e.g., zooming in/out or panning around, the Vmedia browser will use existing media segments in cache to quickly render a coarse view of the new ROI, and in the same time, request a new set of media segments corresponding to the updated ROI. Vmedia greatly improves the browsing experience of large images over slow networks.

Keywords: virtual media (Vmedia), JPEG 2000, media access, interactive browsing, streaming, cache

1. INTRODUCTION

With the booming of the Internet, more and more high-resolution images are brought online. NASA is putting its collection of space photos online [1]. The Microsoft terra server provides access to high-resolution satellite/aero photographs through the Internet [2]. Museums are digitizing the collectibles and moving them online [3]. Even individuals are sharing personal photos over the web [4]. It is enjoyable to watch a high-resolution image on screen; but it is equally painful to download the huge bitstream (even compressed) over the slow Internet. Although the backbone of the Internet keeps improving, the content and the number of users of the Internet also grow exponentially. Efficient delivery of large image is thus 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 [5] 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 is still 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). Besides reducing the bandwidth requirement, the interactive browsing is also ideal in case the image resolution is higher than the display resolution, such as in the handheld device. Live Picture Inc has developed a Flashpix viewer [6] for the interactive browsing of large image with the Flashpix format. The Flashpix generates a multi-resolution view of the original image, where each resolution image is further segmented into fixed tiles at size 64x64 and compressed independently with standard JPEG. The Flashpix browser downloads only the compressed bitstream of the ROI, so it can browse the image quickly. However, since each tile is compressed separately, and multiple compressed images at different resolutions need to be stored at the server, the compression efficiency of the Flashpix format is poor. 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,.

In this work, we implement an interactive image browser using JPEG 2000. We choose JPEG 2000 because it not only significantly outperforms JPEG in terms of compression ratio, but also offers a number of useful features, one of them is the decoder ROI access. For a specific ROI, it is possible to locate a set of bitstream segments which can be decoded to reconstruct the ROI, and the reconstructed ROI is exactly the same as if we decode the entire image and cut out the ROI. We refer to [8] for in-depth information of the JPEG 2000 standard.

Though enabled by the bitstream syntax, the decoder ROI access/interactive browsing functionality is not implemented in JPEG 2000[9]. Moreover, delivery of bitstream associated with the ROI is not trivial, especially as the bitstream segments needs to be delivered in a prioritized fashion, cached and managed. We develop Vmedia, a virtual media access protocol, to support media segment access in the interactive browsing application. A JPEG 2000 interactive media browser is further implemented, which according to the user requests, locates the bitstream segments corresponding to the current ROI, accesses the bitstream segments through Vmedia, and renders the ROI. Only the compressed bitstream segments necessary to decode the current ROI are accessed and streamed over the Internet. Moreover, the bitstream segments 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 media segments arrive, the quality of the rendered ROI improves. Vmedia greatly improves the experience of browsing large images over the Internet.

The paper is organized as follows. We describes the framework, functionality and implementation of Vmedia in section 2. The implementation of the JPEG 2000 interactive image browser is described in section 3. Section 4 and Section 5 show the experimental results and the conclusion, respectively.

2. VIRTUAL MEDIA ACCESS PROTOCOL

2.1. The framework

Vmedia stands for the virtual media access protocol. It is a multimedia communication protocol in a server-client environment. Rather than treating the entire media file as a whole, or streaming the media file sequentially, Vmedia enables flexible accessing to the media, and streaming just the compressed bitstream of the current view. Though used specifically for the JPEG 2000 image browser, Vmedia is useful for many other Internet browsing applications, such as the browsing of 3D environment[16]. The framework of Vmedia can be shown in Figure 1.

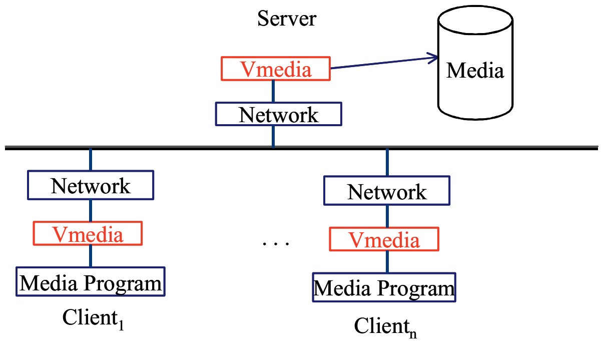


Figure 1 Vmedia framework

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

The following functionalities are provided by Vmedia:

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

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

3. Prioritized delivery
High priority segment, such as the bitstream segment associated with the first quality layer, can be delivered first over the Internet to improve the response time.
4. Media segment packaging
Small media segments are automatically packaged into a larger one, and a large media segment is broken into several small packets according to the maximum transmission unit (MTU) of the network path. Packet loss and retransmission are also handled.

Through a group of unified APIs, Vmedia hides most chores of interactive browsing from the programmer. The media application simply accesses remote media as a virtual local file, issues media segment access requests with priority and importance. We explain the typical workflow of Vmedia in Section 2.2. The two key components of the Vmedia, the Vmedia cache and the media segment delivery are explained in Section 2.3 and 2.4, respectively.

2.2. The workflow

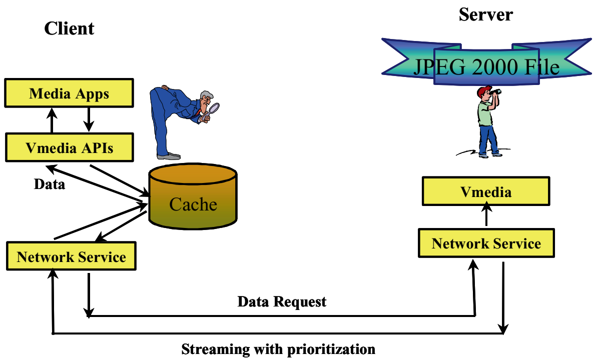


Figure 2 Workflow of a Vmedia media application

A typical application flow using Vmedia is shown in Figure 2. The application calls the Vmedia API to access the remote media. Upon the receipt of the call, the Vmedia client contacts the Vmedia server, establishes the connection, and verifies the existence of the remote media. During the connection phase, the structure of the media, in this case the JPEG 2000 file and packet header, is downloaded to the Vmedia client. The Vmedia cache is also initialized at the connection phase. After that, segment of media, indicated by the start position *pos* and length *l*, can be accessed by the Vmedia client. 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 and sent to the Vmedia server to download the missing segments. The request is not sent immediately over the network because the requests need to be packetized and prioritized for delivery. A synchronization function call is issued by the application to inform the Vmedia that there are no more pending requests. The requests still in queue are then sorted and sent to the Vmedia server.

Vmedia works in an asynchronous mode, where the media segment request is served on a best effort basis. Vmedia always returns the control immediately to the application regardless of the outcome of the request. To best utilize the Vmedia protocol, the application should also render the media on a best effort basis. A usual strategy is that the application repeatedly queries and renders the ROI using whatever data currently available. A 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 application closes the Vmedia connection, the current content of the Vmedia cache can be optionally dumped as a permanent file to the disk, so that the next time the same remote media is accessed, there is no need to download the media segments that have already been received in the last session. The functionality is similar to the temporary Internet files implemented in the Internet Explorer™ and Netscape™, though instead of an entire file, segments of a file are dumped.

2.3. 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 update

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. However, such an approach is very memory consuming and cannot browse media larger than the memory capacity. Another approach is to allocate media segment on demand, with arbitrary length and start point. The approach may severely fragment the memory. It is also slow to detect whether a particular segment is already in the Vmedia cache.

In the Vmedia, we use fixed size media unit to improve the speed of cache hit detection and reduce the memory overhead needed to hold the Vmedia cache. An arbitrary positioned media segment is round up into a set of sub media unit (SMU), which is the minimum 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 Figure 3. 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 per 1024 byte MU, which is about 1%.

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

As shown in Figure 3, each MU maintains one importance, one hit count and one lock tag. There is also a validity bit for each SMU. The importance tag is provided by the application, with value 255 for the most important, and value 0 for the least important. A highly important MU is less likely to be thrown away in case that the cache memory is used up. A MU may be assigned with several importance values through different function calls. In such a case, the highest importance value among all calls is assigned to the MU. The MUs may also be temporarily locked to ensure that certain media segments are not thrown away. The lock is frequently used on the media segments associated with the current ROI to ensure newly arrived bitstream segments of the current view is not accidentally swapped out. We refer to Section 3.2 for an example of lock usage. In case that the ROI changes, an unlock function is called to release all locks placed on the Vmedia.

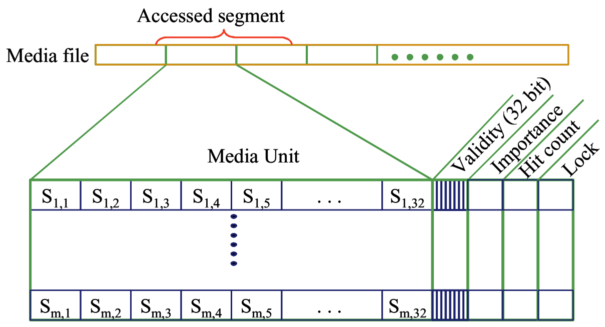


Figure 3 Media unit (MU) and sub media unit (SMU).

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.

2.4. Delivery of media segments

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

In the event of ROI change, namely a radically different set of media segments are accessed, a clear function call can be issued to eliminate all requests in the pending and sent queue. The Vmedia client also contacts the Vmedia server to cancel all requests not processed by the server. Canceling all existing requests in both the pending queue and the server speeds up the delivery of the content associated with the new ROI. Requests that are already in route to the client are still processed by the Vmedia client and stored in the Vmedia cache when they arrive.

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

The functionality of the Vmedia server is rather simple: it handles media access request coming from the Vmedia client. Media segment prioritization and caching are all handled by the Vmedia client. Such design puts more computation load at the client, and reduces the burden of the server. Though in this paper, the Vmedia is developed for the interactive image browsing, it can be associated with any type of media, e.g., interactive environment browsing[16]. Remote media is simply treated as an unstructured, one dimension file, from which portions of the media are accessed with priority.

3. INTERACTIVE BROWSING OF JPEG 2000 COMPRESSED IMAGE

3.1. JPEG 2000

In this section, we briefly review the JPEG 2000 coder and bitstream syntax. JPEG 2000 is a wavelet coder. The image is first transformed into the wavelet domain with a multi-resolution wavelet decomposition. Each resolution subband is segmented into fixed size coefficient blocks. The default block size of JPEG 2000 is 64x64, however, other block size may be used as well. As an example shown in Figure 4, the 512x512 Lena image is decomposed by a 3-level wavelet transform. There are 16 coefficient blocks per subband in resolution 3 (the finest resolution), 4 coefficient blocks per subband in resolution 2, and 1 coefficient block per subband in resolution 1 (the coarsest resolution). Each block is encoded with an embedded bit plane coder, 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. Pioneered by the embedded zerotree wavelet (EZW) coder[11], the embedded coded bitstream can be truncated at any place while still achieving a fair quality image. The coder in JPEG 2000, termed embedded block coding with optimized truncation (EBCOT) [7], further extends the embedding functionality to individual block bitstream. The embedded coded bitstreams of all coefficient blocks are then assembled together into a JPEG 2000 file. There are several possible assemble schemes. The most common one is the progressive by quality assembling. It is also called progressive by SNR (signal-to-noise ratio) assembling, as SNR measures the quality. A set of decreasing rate-distortion slopes $\lambda_1, \lambda_2, \cdots, \lambda_n$ are determined, and block bitstream with rate-distortion slope between λ_{i-1} to λ_i is chopped to form SNR layer *i*. We denote SNR layer *1* bitstream segment as the first segment, and layer *n* segment as the last. All block bitstreams of SNR layer *i* and resolution *j* are assembled to form a packet of resolution *j* and SNR layer *i*. Attached to each packet is a packet header, which records the number of bytes for each coefficient block, and provides index within the packet. The packets are ordered first by SNR 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 SNR layer, next by resolution, and finally by components. It is proven that the assembling order above provides an optimum performance if the assembled bitstream is truncated later [12]. Usually in encoding, the value of the layer *i* rate-distortion slope λ_i is not specified explicitly. Rather, we adjust the slope λ_i so that the coding bitrate up to the SNR layer *i* reaches a specific bitrate value. Before the first packet, there is a file header, which contains 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 Figure 5. For a more detailed description of the JPEG 2000 bitstream, we refer to document [8]section X.

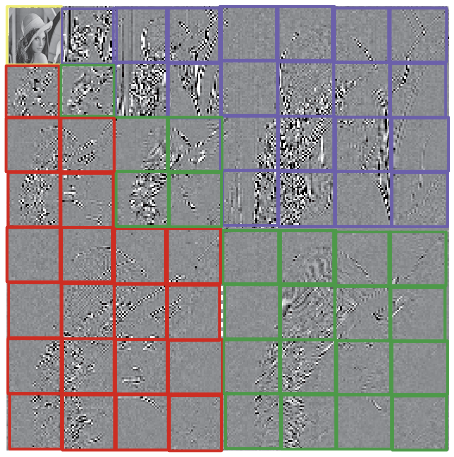


Figure 4 Wavelet transform and block partition in JPEG 2000. The sample image is Lena, with a 3-level wavelet transform.

Such bitstream syntax provides the possibility to decode a region of interest (ROI) with both spatial and resolution constraint without decoding the entire JPEG 2000 compressed bitstream. The wavelet decomposition provides the access by resolution. The coefficient block provides the access by spatial locality. And the embedded block bitstream provides the access by quality (progressive streaming). The smallest access unit, i.e., the bitstream segment, in the JPEG 2000 bitstream is the block bitstream of one SNR layer. Though enabled by syntax, the existing JPEG 2000 verification software [9] does not provide functionality to decode a ROI from the compressed bitstream. There is also no scheme to deliver segments of JPEG 2000 coded bitstream.

3.2. Interactive JPEG 2000 image browsing with Vmedia

In this section, the workflow of an interactive Vmedia JPEG 2000 image browser is described. In the Vmedia image browser, the user specifies a current region of interest (ROI), which may be the entire image at low resolution, or a detailed region at high resolution. The user changes the ROI interactively by panning around and zooming in/out of the image. The Vmedia image browser responds by accessing the bitstream segments covering the current ROI, streaming them from the Vmedia server if they are not locally available, decoding and rendering the current ROI.

For easy discussion, we describe the operation of the Vmedia image browser with two examples. The sample image is a JPEG 2000 compressed 512x512 Lena image with 3-level wavelet decomposition and 64x64 coefficient block (shown in Figure 4). We assume that each block bitstream is chopped into 5 SNR layers. We also assume that the image is of a single tile and a single color component (gray image). However, the discussion below can be easily extended to images 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.

After the Lena image is compressed with JPEG 2000, the compressed bitstream is parsed and the file header and packet header are identified and stored in the companion file. We use the companion file in the Vmedia so that the media structure information can be quickly identified and streamed, which enables the subsequent interactive browsing. Both the compressed bitstream and the companion file are put on the Vmedia server. In the beginning, the Vmedia image browser calls the Vmedia client to establish the connection with the Vmedia server, and download the companion file. By decoding the companion file, the Vmedia image browser is able to locate the position of the bitstream segment for every SNR layer and coefficient block.

In the first ROI browsing example, the ROI is of size 128x128, with its upper left corner located at (256,256), as shown in Figure 6(a). It is easily calculated that a total of 10 wavelet coefficient blocks need to be accessed to decode the ROI[1]. 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 Figure 6(b). The bitstream segments of all 5 SNR layers of the 10 blocks are accessed by the Vmedia image browser. We call those bitstream segments as the ones that covering the current ROI. The bitstream segments are accessed with different priority and importance. We order the delivery priority first with the SNR layer, and then with the resolution. We also assign a higher importance value to coarser resolution block bitstreams, so that in case the cache memory is tight, the coarser resolution

[1] Considering the boundary of the wavelet filter, the accessed region is actually a little larger. Though ignored in the discussion for simplicity, the boundary adjustment is implemented in the actual Vmedia interactive image browser.

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.

For a coefficient block, the bitstream segments that are already in Vmedia cache and accessible are assembled for decoding. Since each block is embedded encoded, if some of the tailing bitstream segments are not available, they are simply discarded. Occasionally, a higher SNR layer bitstream segment is available, while certain lower SNR layer bitstream segment of the same coefficient block is not because of the loss, transmission jitter or corruption of the lower SNR layer bitstream segment or its request sent to the server. In such a case, the existing higher SNR 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 a bitstream after a missing segment.

Back to our example in Figure 6(b), at a certain instance, most of the SNR layer 1, 2 and 3 bitstream segments of the 10 accessed blocks have arrived. SNR layer 4 bitstream segment has arrived for block A, while SNR layer 1 and SNR layer 2 bitstream segments are missing for block B and C, respectively. Block B is decoded as all zeros because its 1st bitstream segment is not available. The browser may decode block A to SNR layer 4, block C to SNR layer 1, and the rest blocks to SNR 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 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 ROI based on the received bitstream segments.

The above process of bitstream segment accessing, assembling, decoding, inverse quantization, inverse wavelet transform and rendering is repeated again and again by the Vmedia 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 Figure 6, the user pans right to view the ROI with upper left corner at (384, 256), as shown in Figure 7(a). Receiving the change of ROI request, the Vmedia 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 the bitstream segment 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 is different, as the row 3 and column 4 blocks are accessed for all directional subbands of resolution 3, as shown in Figure 7(b). The browser again accesses all bitstream segments associated with the coefficient blocks. In the first iteration after the switch of ROI, no bitstream segments of resolution 3 blocks are available. However, the SNR 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 SNR 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 bitstream segments are then gradually streamed from the Vmedia server. First, the bitstream segments of SNR layer 1 resolution 3 packet are streamed, then those of SNR layer 2, after that those of SNR layer 3. As the first few SNR layer bitstream segments of 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.

4. EXPERIMENTAL RESULTS

We have implemented the Vmedia protocol and the Vmedia interactive image browser. The browser wraps around a modified version of JPEG 2000 VM 5.2 decoder software [9]. The Vmedia browser can either be used as an ActiveX plug-in embedded in a web page, as shown in Figure 8, or be implemented as a stand alone application, as shown in Figure 9. The browser enables the user to interactively wander around the image and view the image through an underlying window. At each instance, the user specifies a region of interest (ROI), which may cover the entire image at low resolution, or may be a specific region at high resolution. The user may pan around and zoom in/out of the image, and change the ROI accordingly. In this section, we evaluate the performance of the Vmedia image browser, especially on the slow connection link.

To prepare an image for the interactive browsing application, the image is encoded by JPEG 2000 VM 5.2 [9]. Seven images (shown in Table 1), 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 5 level biorthogonal 9-7 tap wavelet transform, compressed to a final bit rate of 1.0bpp (bit per pixel), with the bitrate of the intermediate SNR layers set at 0.0625, 0.125, 0.25, 0.375, 0.5, and 0.75bpp (a total 7 SNR layers). The image Aerial1 is decomposed by a 7 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.

Table 1 Test image (compressed by JPEG 2000 at 1.0bpp)

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%

A live Vmedia interactive image browser will be demonstrated at the conference. It takes a long delay (from 2.5 minutes to 103.5 minutes) if we first download the compressed bitstream and then render. The delay is also 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 511KB to 26MB). The Vmedia image browser responds to our request very quickly. We can pan around, zoom in and out of the image with ease. A low quality view of the current ROI is usually shown immediately after the change of ROI. The quality of the ROI also quickly improves, as the first few SNR 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 480x480.

Before the interactive browsing can be performed, we need to download the companion file, which contains the structure information of the JPEG 2000 bitstream. Fortunately, the overhead of the structure information is pretty light in JPEG 2000. The size of the companion file and its percentage versus the total compressed bitstream are listed in columns 5 and 6 of Table 1, 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 of JPEG 2000. The streaming of the companion file takes 1-3 seconds for relatively small images such as Aerial2, Café, Cats, Bike and Woman. However, it can take as long as 15 seconds for Cmpnd2 and 54 seconds for Aerial1 on a 33.6kbps modem. There are ways to cut the initial connection time, for example, by building an index over the structure information and stream the structure information on demand. Since only the index of the structure information needs to be put in the companion file and downloaded at the connection, the connection time can thus be reduced.

Table 2 Bitstream covering size of a 480x480 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

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 2, we list in number of kilobytes the covering size for zoom out ratio from 1:1 to 16:1. The ROI is of size 480x480 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 whole test images Aerial2, Café, Cats, Bike and Woman. Therefore, the covering sizes are also the size of the compressed bitstream for the resolution image. It is observed that though the compressed bitstream of the image can be very large, the covering size of a ROI is usually stable, ranging from 11KB to 61KB for the original resolution or 88KB to 143 KB for zoom out ratio 4:1. In fact, the average covering size of an original resolution ROI is 33KB (1.18bpp), pretty close to the image compression bitrate. At coarser resolution, the covering size of a ROI increases. However, at such resolution, the high SNR layer bitstream segments are for pixel depth beyond 8 bits, and are thus not useful in substantially improving the visual quality of the ROI. In fact, an intermediate resolution ROI achieves visual lossless when the size of the available bitstream segments reaches around 28KB-56KB(1.0-2.0bpp). 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 after the switch of ROI. We continuously zoom in, from 16:1 to 1:1. We define the size of the compressed bitstream segments that are available at the switch of ROI as the initial covering size. The ROI is still of size 480x480, and lies at the top-left corner of the image. 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 3. The same operation is repeated as we continuously zoom in until the original image resolution is reached. The initial covering sizes for resolutions 4:1 to 1:1 are recorded in columns 3-5 of Table 3. We observe that during the zoom in operation, roughly 80% of the bitstream segments is available 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 number of the initial versus total covering size decreases for the coarse resolution ROI, and becomes 47% if we zoom in from 4:1 to 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 seconds.

Table 3 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

5. CONCLUSIONS

In this work, we develop a Vmedia JPEG 2000 image browser. It is the first to provide the decoder region of interest (ROI) access and interactive browsing functionality for the JPEG 2000 compressed image. A virtual media (Vmedia) access protocol is proposed to manage and deliver the bitstream segments associated with the JPEG 2000. With the Vmedia interactive image browser, the user may interactively browse a large compressed image through the ROI. Only the compressed bitstream needed by the ROI is streamed and decoded. Moreover, the streaming is performed in a quality progressive fashion. A low quality view of the ROI can be rendered very quickly, and gradually improved as more and more bitstream segments arrive. The browsing experience is very pleasing, and compressed image as large as 15kx15k (compressed to 26MB) can be browsed smoothly over a modem line of 33.6 kbps. For a 480x480 ROI, a visually lossless ROI is usually reached within 10 seconds.

6. ACKNOWLEDGEMENT

The author would like to acknowledge the following individuals: Dr. Xiaoning Lin, for his contribution in initializing the Vmedia project, Dr. Ya-Qin Zhang, for his managerial support and naming of Vmedia, Hu Li, for codes of Vmedia client, Dr. Qian Zhang, for programming the network interface of the Vmedia client and server, Jianping Zhou, for packaging the Vmedia image browser.

7. BIBLIOGRAPHY

[1] NASA, http://nssdc.gsfc.nasa.gov/photo_gallery/.
[2] Microsoft, http://terraserver.microsoft.com/default.asp.
[3] http://www.museumca.org/usa/alpha.html.
[4] Kodak, https://pvind.photonet.com/.
[5] W. B. Pennebaker and J. L. Mitchell, "JPEG Still Image Data Compression Standard", Van, Nostrand Reinhold, 1993, ISBN 0-442-01272-1.
[6] Live picture, www.livepicture.com.

[7] D. Taubman, "EBCOT: Embedded Block Coding with Optimized Truncation", ISO/IEC JTC1/SC29/WG1/N1020, Los Angeles, CA, Nov. 1998.
[8] VM adhoc, "JPEG 2000 verification model 5.2 (technical description)", ISO/IEC JTC1/SC29/WG1N1422, Vancouver, BC, Jul. 1999.
[9] VM adhoc, "JPEG 2000 verification model 5.2 (source code)", ISO/IEC JTC1/SC29/WG1N1423, Vancouver, BC, Jul. 1999.
[10] J. Li, "Visual progressive coding", in *SPIE Visual Communication and Image Processing (VCIP'99), Vol. 3653, pp. 1143-1154,* San Jose, CA, Jan. 1999.
[11] J. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. on Signal Processing*, Vol. 41, No.12, pp.3445--3462, Dec. 1993.
[12] J. Li and S. Lei, "An embedded still image coder with rate-distortion optimization", *IEEE Trans. On Image Processing*, Vol. 8, No. 7, pp. 913-924, Jul. 1999.
[13] Network working group, "Hypertext transfer protocol – HTTP/1.1", W3C/MIT, June, 1999.
[14] C. Zhang and J. Li, "Compression of Lumigraph with multiple reference frame (MRF) prediction and just-in-time rendering", *IEEE Data Compression Conference (DCC'2000),* Snowbird, Utah, Mar. 2000.
[15] L. Luo, Y. Wu, J. Li and Y. Zhang, "Compression of concentric mosaic scenery with alignment and 3D wavelet transform", SPIE: Image and Video Communication and Processing, Vol. 3974, No. 10, San Jose CA, Jan. 2000.
[16] C. Zhang and J. Li, "Interactive Browsing of 3D Environment over the Internet", SPIE: Visual Communication and Image Processing (VCIP'2001), Vol. 4310, No. 51, San Jose, CA, Jan. 2001.

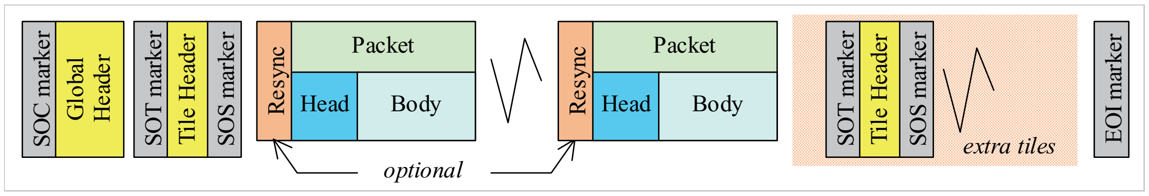


Figure 5 JPEG 2000 file structure.

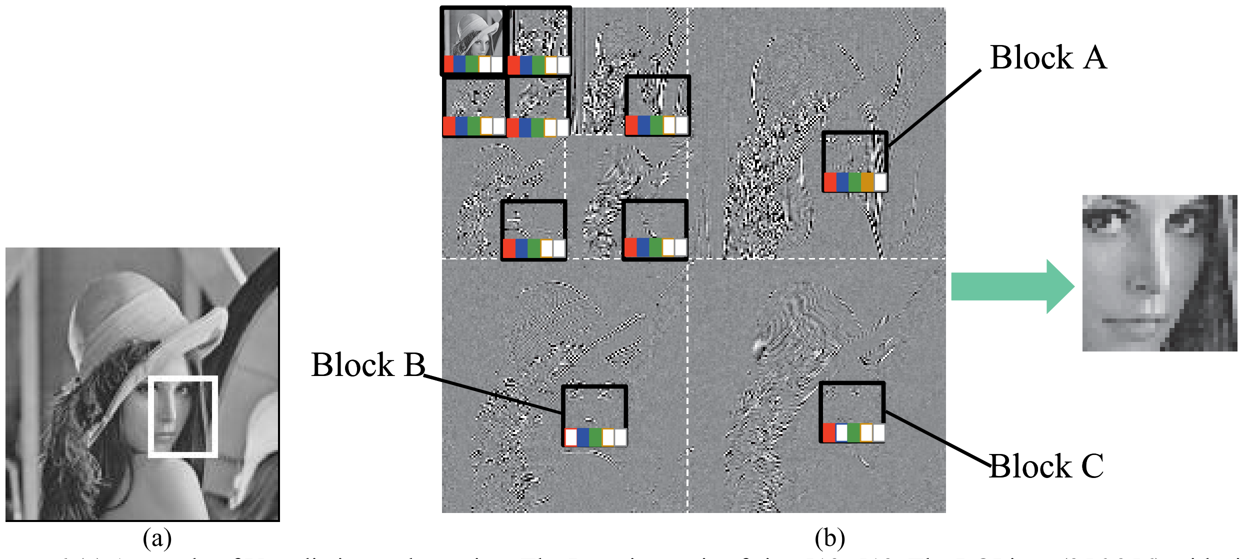
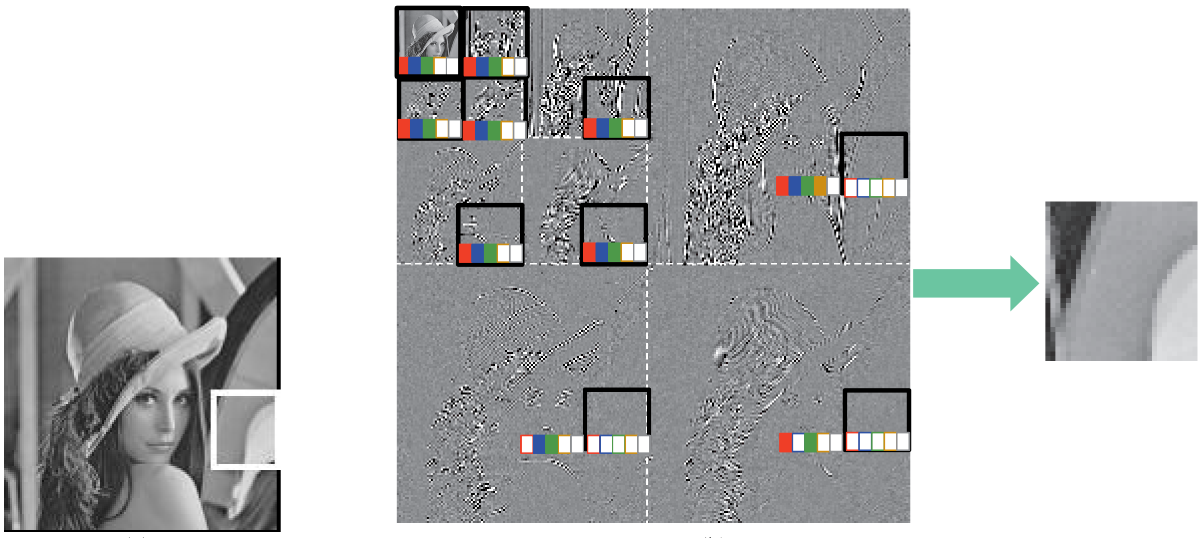
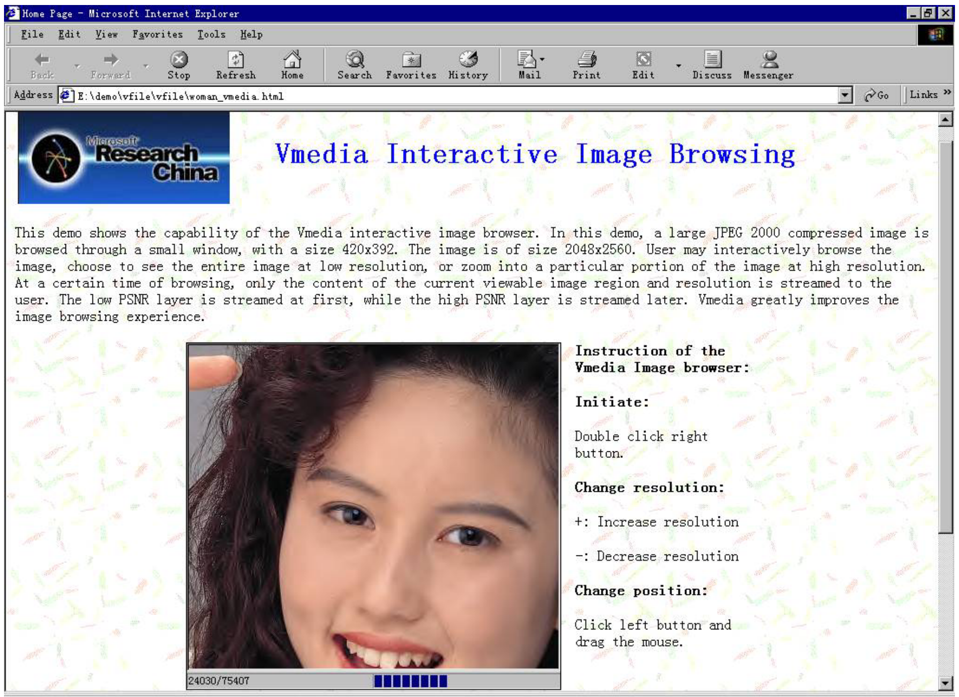


Figure 6 (a) A sample of Vmedia image browsing. The Lena image is of size 512x512. The ROI is at (256,256) with size (128,128). (b) The accessed bitstream segments. The black box identifies the coefficient blocks accessed by the Vmedia. A color bar identifies an available bitstream segment, and a white bar identifies an unavailable bitstream segment.



(a) (b)

Figure 7 (a) The ROI pans right to (384,256). (b) The accessed bitstream segments of the ROI. The black boxes identify the accessed coefficient blocks. The color and white bars identify the available and unavailable bitstream segments, respectively.


Home Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Forward Stop Refresh Home Search Favorites History Mail Print Edit Discuss Messenger
Address E:\demo\vfile\vfile\woman_vmedia.html
Go Links
Microsoft Research China
24030/75407

Vmedia Interactive Image Browsing

This demo shows the capability of the Vmedia interactive image browser. In this demo, a large JPEG 2000 compressed image is browsed through a small window, with a size 420x392. The image is of size 2048x2560. User may interactively browse the image, choose to see the entire image at low resolution, or zoom into a particular portion of the image at high resolution. At a certain time of browsing, only the content of the current viewable image region and resolution is streamed to the user. The low PSNR layer is streamed at first, while the high PSNR layer is streamed later. Vmedia greatly improves the image browsing experience.

Instruction of the Vmedia Image browser:

Initiate:

Double click right button.

Change resolution:

+: Increase resolution

-: Decrease resolution

Change position:

Click left button and drag the mouse.

Figure 8 Vmedia interactive image browser embedded in a web page. The image is Woman, a color image in the standard JPEG 2000 test set. The original image is of size 2048x2560. It is compressed to 1.0bpp or 640 KB. It is interactively browsed through a window of size 420x392.

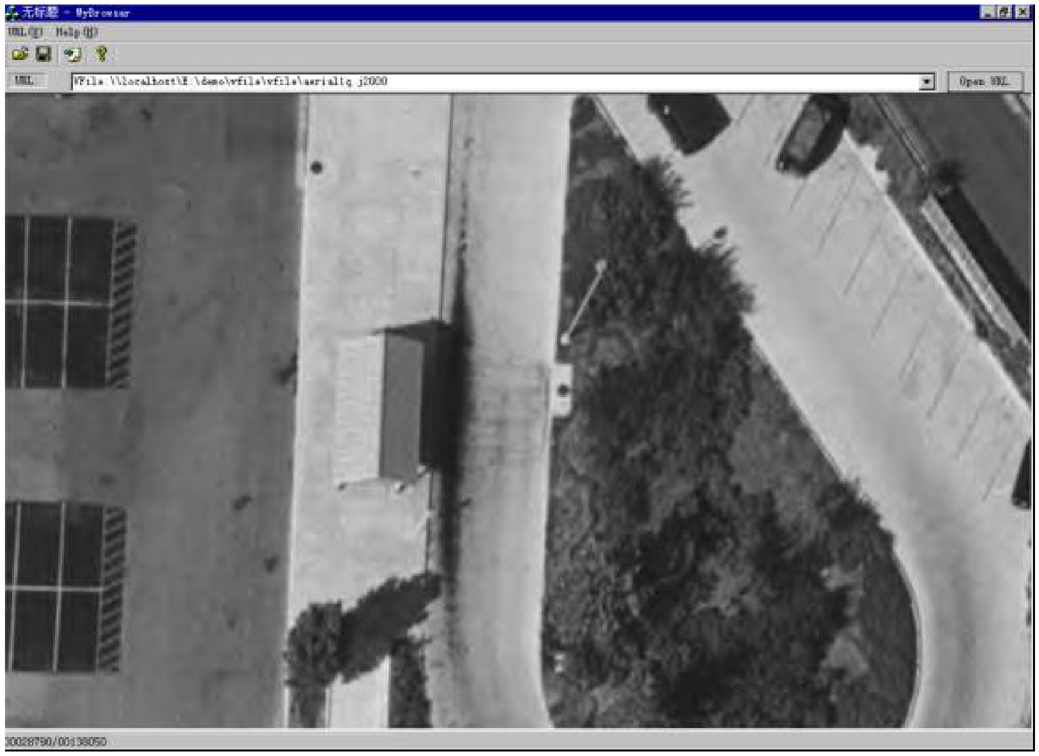


Figure 9 Vmedia interactive image browser as an independent application. The browsed image is Aerial1, the largest image in the standard JPEG 2000 test set. The original image is of size 14565x14680. It is compressed to 1.0bpp or 26,101 KB. The interactive browse window is of size 1020x630