MITSUBISHI ELECTRIC RESEARCH LABORATORIES
http://www.merl.com

Server-Driven Progressive Image Transmission of JPEG 2000

Derek Schwenke, Anthony Vetro, Toshihiko Hata

TR2008-005 March 2008

Abstract

Progressive transmission of images is an important functionality for communicating high resolution images over limited bandwidth networks. By encoding the image data in an accessible and hierarchical format, the JPEG 2000 standard supports many types of image progessions, e.f., based on quality, resolution, component and position. This paper considers a progressive transmission scheme in which codestream ordering and transmission decisions are drive entirely by the server, which is useful for classes of applicaitons that employ image analysis at the server and perform streaming based on the results of this analysis. The proposed system aims to minimize signaling overhead and allow for incremental decoding and display with minimal processing delay. It also aims to fully exploit the various styles of progression that are enabled by the JPEG 2000 coding format. The performance of our proposed scheme is reported in terms of signaling overhead, complexity and visual effectiveness.

SPIE Conf on Visual Communications and Image Processing

Server-Driven Progressive Image Transmission of JPEG 2000

Derek Schwenke[a], Anthony Vetro[a], Toshihiko Hata[b]
[a] Mitsubishi Electric Research Laboratories, 201 Broadway, Cambridge, MA, USA 02139
[b] Mitsubishi Electric Corporation, Amagasaki, Hyogo 661-8661, Japan

ABSTRACT

Progressive transmission of images is an important functionality for communicating high resolution images over limited bandwidth networks. By encoding the image data in an accessible and hierarchical format, the JPEG 2000 standard supports many types of image progressions, e.g., based on quality, resolution, component and position. This paper considers a progressive transmission scheme in which codestream ordering and transmission decisions are driven entirely by the server, which is useful for classes of applications that employ image analysis at the server and perform streaming based on the results of this analysis. The proposed system aims to minimize signaling overhead and allow for incremental decoding and display with minimal processing delay. It also aims to fully exploit the various styles of progression that are enabled by the JPEG 2000 coding format. The performance of our proposed scheme is reported in terms of signaling overhead, complexity and visual effectiveness.

Keywords: JPEG 2000, progressive transmission, scalability, streaming, server

1. INTRODUCTION

Scalable image compression schemes such as JPEG 2000[1,2] are designed so that different qualities, resolutions, components and positions of the compressed image could be easily accessed, transmitted, decoded and displayed. Progressive transmission of such images refers to the process by which more important portions of the coded data are transmitted first. For instance, progressive quality is achieved by transmitting coarsely quantized samples of the image first, followed by refinements of the image samples. When network bandwidth is limited, this effectively allows the client to display a lower quality version of the image in a short amount of time, and then as more coded data is received, the image quality improves over time.

A practical need for progressive transmission exists for the remote browsing of large images over networks with limited bandwidth[3]. In this particular application scenario, the desire to interact remotely with the content mandates the need for a protocol between the client and server. The standardized solution to this problem is JPIP, which is specified as Part 9 of the JPEG 2000 standard[4,5]; this solution is essentially a request-response protocol that allows clients to select portions of an image for transmission from a server. A key advantage of this approach is the flexibility that is provided to each client. Specifically, each client makes one or more HTTP GET requests. The requests contain the name of the image resource and the query fields to request. The fields describe many possible image features such as layer, resolution, component, precinct, viewing window, session, cached data, etc.

Beyond remote browsing of images, progressive transmission is also useful in other application environments. For instance, in our previous work, a scalable video streaming system for surveillance applications that is based on JPEG 2000 has been developed[6]. In this system, a stored JPEG 2000 codestream is transcoded in the compressed-domain using a low-complexity adaptation technique. Various types of streaming and display modes are supported that capitalize on the inherit scalability offered by JPEG 2000 and image analysis techniques that detect regions-of-interest (ROI's) such as pedestrian and face regions. It is important to emphasize that in this system the server has knowledge of important parts of the scene and is responsible for determining the image quality to a given region and the appropriate transmission policy; a system with such properties is referred to as server-driven.

The architecture and design philosophy for a server-driven progression image transmission scheme differs from that of a client-driven scheme such as JPIP that operates based on requests from each client. While the server-driven approach does not offer any level of interaction or feedback between client and server, it is suitable for certain applications and has some advantages. For one, signaling overhead and delay are expected to be less in the server-driven approach. Also, since the required processing on the client is substantially simplified, reduced complexity is achieved. Since these approaches are fundamentally different, a direct comparison with JPIP is not included in this paper.

The rest of this paper is organized as follows. In the next section, a brief overview of the server-driven architecture is provided. In section 3, a method for generating ordered segments of the codestream to achieve the desired progressive transmission is elaborated on; the proposed header format for encapsulated segments of the codestream is also discussed. Experimental results are reported in section 4, and concluding remarks are given in section 5.

2. SERVER-DRIVEN ARCHITECTURE

An overview of the proposed progressive image transmission system is shown in Figure 1. The system includes a server that transcodes an input codestream into a transcoded codestream. During the transcoding process, the transcoder also generates decomposition metadata that indicates descriptive information about each packet such as the layer, resolution, component and precinct, the number of bytes, as well as if a particular packet is part of an ROI. From this, codestream bursts are produced according to control data that implements the progressive transmission policy. This control data is essentially a mapping of all the JPEG 2000 packets into particular codestream bursts, and determines the order that codestream data is transmitted to the client. This control data may also be used by the transcoder to optimize the progression order of the codestream.

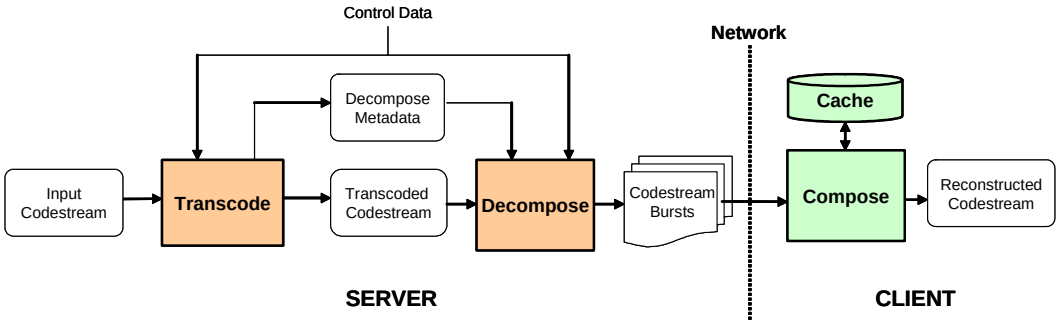


Fig. 1. Overview of server-driven progressive image transmission scheme.

The transcoded codestream may be identical to the input or altered in some way, e.g., to reduce the data rate, crop the image, or change the progression order. As will be discussed later, changing the progression order could be very useful in minimizing signaling overhead. It is also noted that the codestream bursts are non-compliant portions of the transcoded codestream. Each codestream burst contains one or more added headers that provide descriptive information about the particular codestream burst.

At the client side, the compose operation reorders the received codestream bursts and inserts empty packets as needed to construct a compliant codestream. The compose process is invoked each time a new codestream burst is received to generate a new reconstructed codestream that includes the new codestream burst. The newly reconstructed codestream is then decoded and displayed to give the effect of progressive reconstruction.

An illustrative example of this architecture is a distributed surveillance system with remote cameras acting as the servers and a central office as the client where images are received and displayed. In such an application scenario, consider the case in which face detection is preformed at each remote camera using the full image to determine the location and presence of the ROI's of the image. The ROI should be initially transmitted at a reduced quality, with progressive quality improvements of the ROI region over time. The background should then be transmitted next with similar quality progressions over time. To achieve such an ordering with the conventional JPIP approach would require the client to first access to the entire image to determine the ROI location and the desired ordering of the codestream to be transmitted by the server to the client, which is not practical. Alternatively, the client would need to first receive the ROI information for the image, which adds round trip delay time. In this example, the face-first ordering is encoded without any response to a client-server request, and the proposed system is able to achieve a face-first transmission order that reduces transmission time and the bandwidth used.

3. CODESTREAM BURST GENERATION

This section outlines the specific techniques that are utilized to generate codestream bursts according to a given progressive transmission policy. In particular, the structure of the encapsulated codestream bursts is described.

The decompose process sequentially scans through all the packets in the codestream. Based on the corresponding decompose metadata and the control data, each packet is assigned to a particular codestream burst. For each continuous scan of packets in a given codestream burst (assuming raster scan order), a burst header is also written to describe this data. The format of the codestream burst including burst header and codestream data is illustrated in Figure 2.

The fields of the burst header are optional, i.e., all or some fields may be omitted if they are not needed or can be predicted by the client. The optimization of these fields is not discussed at this time. The first field is the "Type", which signifies the type of burst header (i.e., long 12-Byte format or short 6-Byte format used in this work) and also indicates any omitted header fields. The "ID" is the L-R-C-P index into the codestream data; "Count" contains the number of sequential IDs that follow in the codestream data segment; and "Size" is the number of bytes of the codestream data segment that follows.

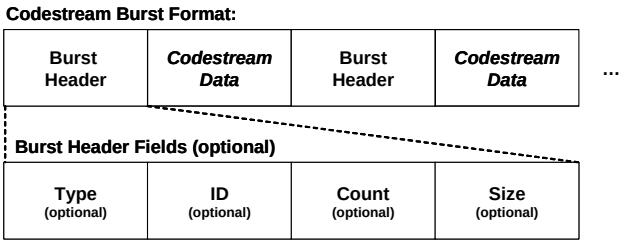


Fig. 2. Syntax for codestream bursts and burst header fields.

The decompose process described above is a general solution for representing any reordering of packets. Using this method, the order of progressive transmission and display could be changed, and packets corresponding to a particular quality layer, resolution level or component could be removed or transmitted at a later time. In this way, progressive transmission of ROI's could also be achieved.

For example, a position-ordered source can be reordered into quality-layer bursts as shown in Fig. 3. In this figure, the decompose process is illustrated. The input stream at the top of the figure includes four positions and three quality layers for each position as indicated by the different shadings of the respective packets. The source contains all packets for position 0, followed by all packets for position 1, position 2, and position 3. The figure shows the formation of each transmission (burst) layer. In this example, it is shown that after the burst header, the main JPC header is transmitted, and that burst headers are inserted into the stream wherever the source stream ordering is broken. In this example, all packets associated with quality layer 0 are inserted into the first burst, packets associated with quality layer 1 into the next burst, and so on. This example is a trivial case for the purpose of explanation. The added headers will expand the stream depending on how many are inserted, which is highly dependent on the ordering of the source input and desired ordering of the target output. More complex cases that allow mixed progression orders are described later and the signaling overhead will also be addressed.

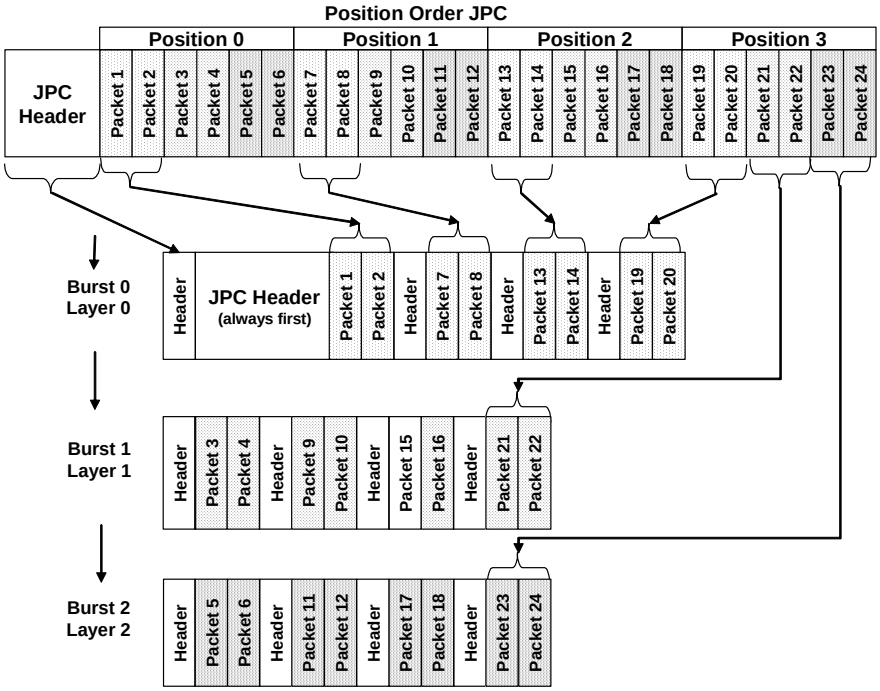


Fig. 3. Example decompose operation from position-ordered codestream to quality-layer bursts.

To better understand how a particular transmission policy leads to codeblock bursts, consider the simple example shown in Fig. 4. Assume that the codestream has a PCRL progression order and the policy is to transmit the ROI with highest quality first (labeled as region 1), followed by the non-ROI regions (labeled as regions 2-5). In this diagram, the each arrowhead represents one added header. In particular, the first codeblock burst would contain 7 burst headers: one for JPEG 2000's main header, and 6 others for the continuous set of packets in each row of the ROI. The codeblock bursts corresponding to regions 2 and 3 would contain 6 headers each, which correspond to the continuous set of packets in each row of the respective regions. Finally, the codeblock bursts corresponding to regions 4 and 5 would contain 1 header each since all the packets in these regions could be continuously scanned.

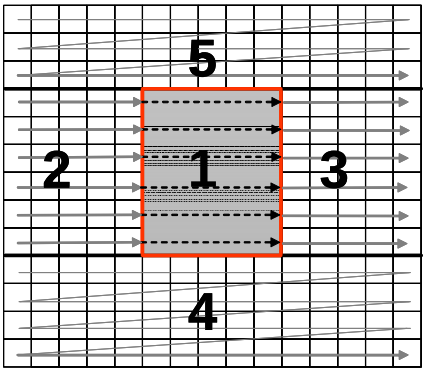


Fig. 4. Example of a specific scan that orders ROI first, followed by scans for each non-ROI region.

Many additional transmission orders can be achieved using different scans. Scans can mix progression orders of position, quality, resolution, and component in the same image. Positions can be defined by one or more ROIs, a motion weighting based on object movement over a sequence of images, or other scene characteristics such as the number and position of detected faces or the position of activity detected by sensors. Some example progressions are briefly shown in Fig. 5. For instance, Fig. 5(a) represents an image with a defined ROI region which is transmitted first in progressive quality layers until the ROI is completely sent, and then the background is transmitted in quality layers until the full image is complete. Figs. 5(b) and 5(c) repeat the same ROI-first progression with resolution layers or component layers. Figure 5(d) extends the defined ROIs to divide the image into 3 regions (top, middle, bottom) with the middle ROI region transmitted first, followed by the other regions. Figure 5(e) is a version of the progression presented in Fig. 4, where the image is divided into 5 regions. Figure 5(f) represents a ROI-first, quality horizon transmission order, where the ROI image is first transmitted in quality layers, and then quality layers are added to the background regions along a horizontal line until the horizon line is complete. Next, quality layers are added for any quality layers that have not been transmitted surrounding the quality horizon region until the quality horizon fills the full image. Another progression is presented in Figure 5(g) and referred to as Quality ROI Box. In this scheme, the ROI region is transmitted in quality layers until the ROI is complete. Next, all of the remaining positions that are adjacent to a transmitted-ROI region are sent. This progression builds out from the defined ROI region, until the image has been fully transmitted. Figure 5(h) is the same as 5(g) with the exception that quality alway builds out from the geometric center of the image. Figure 5(i) is the same as 5(h) where the first image sent is the full image at the lowest quality layer. Next, additional quality layers are sent for the ROI region, and then for the remaining background regions with the order shown in 5(h).

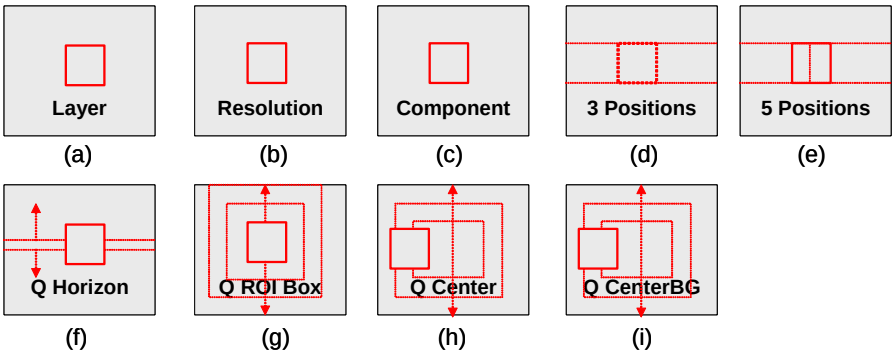


Fig. 5. Various other progression order scans based on layer, resolution, component and position.

4. EXPERIMENTAL RESULTS

To demonstrate the effectiveness of the proposed progressive transmission scheme, an evaluation of the signaling overhead for different types of progressions and codestreams is conducted. The complexity and visual effect of the proposed scheme is also reported and discussed. Experiments are conducted with a JPEG 2000 encoded image of 640x480 with 3 quality layers and 3 resolution levels with LRCP progression, and precinct sizes set as 64x64, 32x32, 16x16. The total size of the encoded image is 38 KB, and contains 2881 packets.

4.1 Signaling Overhead

Table 1 summarizes the broad range of the signaling overheads for different progressive orderings of the input codestream and output codestream bursts in terms of number of headers for a sample input image with and without ROI. It is observed that for each transmission policy, there exists a progression order of the codestream that yields a minimum signaling overhead, which could be determined at the server. In these cases, the overhead of the proposed approach is extremely low and offers support for a wide range of progressive transmission policies. It is also noted from the results in Table 1 that the presence of an ROI will create more discontinuities in the scan, and hence the need for more headers in general. Fig. 6 expands on this point and plots the percentage of signaling overhead in bytes for ROI with varying dimensions assuming a quality progression with LRCP codestream. As noted earlier, the current system uses headers that are either 12 or 6 bytes each. These results show that although there is some increase in the overhead, the total is kept to

a minimum. It is important to emphasize that the graph in Fig. 6 shows multiple values for one ROI area; this variation in the overhead is a result of the ROI position and dimension in a scene, which determines if the scan is broken and a header needs to be inserted. For instance, with a PCRL progression there will be increasing overhead for an ROI with increasing vertical dimension since it breaks the order in a greater number of scan lines.

Table 1. Evaluation of signaling overhead in codestream bursts based on number of headers. Results are provided for different transmission policies and progressive orderings of the input codestream, as well as with and without ROI.

Transmission Policy	No ROI					With ROI				
	LRCP	RLCP	RPCL	PCRL	CPRL	LRCP	RLCP	RPCL	PCRL	CPRL
Quality	**3**	9	2160	2160	2160	**111**	117	2160	2160	2160
Resolution	9	**3**	**3**	720	720	117	111	**15**	720	720
Component	27	27	720	240	**3**	135	135	720	240	**15**
Position 5	135	135	15	**5**	15	298	298	34	**12**	34
Position 3	81	81	9	**3**	9	82	82	10	**4**	10
Q Horizon	135	135	720	240	**15**	235	235	720	240	**27**
Q ROI Box	495	495	474	158	**55**	1081	1081	672	2160	**121**
Q Center BG	1081	1081	121	**41**	121	1189	1189	151	**51**	133
Q Center	729	729	456	152	**81**	793	793	465	155	**89**

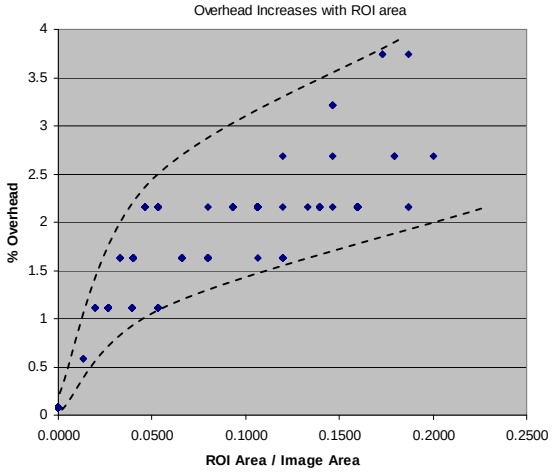


Fig. 6. Evaluation of signaling overhead in codestream bursts based on number of headers. Results are provided for different ROIs with varying dimensions and plotted as a function of total ROI area.

4.2 Complexity

In this subsection, the complexity of both the decompose operation at the server, as well as the compose operation at the client is considered. The complexity of the transcode operation has been addressed in previously published work[7], where it has been shown that proposed transcoding techniques are suitable for implementation on very low-cost processors.

As part of the decompose operation, each header inserted into a burst requires calculating its position or offset and size. Thus, the complexity of the algorithm is linear and related to the signaling overhead described above. These calculations require an addition/subtraction, and can be performed as each burst is written to the buffer. All bursts can be generated in a single scan of the JPC codestream and multiple outputs can be written simultaneously, or a multi-pass approach may be used. As a result, the required computation to implement the proposed scheme is minor, thereby making is suitable for platforms with limited computation.

To reconstruct and display the sequence of codestream bursts, the compose operation is applied. Initially, the first burst containing the JPC main header is read. Using information in the main header and the added burst headers, empty packets are inserted for any missing burst data that has not yet been received. An insertion sort was used in our implementation as the burst is read so that this operation would be efficient. This allows the burst data to be received in any order; however, only certain orders result in uniquely decodable and compliant images. Every burst can be used to

generate a new JPC stream that can be decoded and displayed. Thus, the decoding time needs to be considered. Since there is no efficient way to add the contribution of one new packet to a decoded image, a new compressed image is generated in our system for each incoming burst. However, applications can control the rate at which reconstructed codestreams are decoded and displayed. This essentially provides some control on the complexity at the client.

Table 2 shows example complexity measured on a 2.8 GHz Pentium with 2GB RAM. The same test stream as used in previous experiments is considered with the ROI progression shown in Fig. 4, i.e., 5 bursts. The experiment used Kakadu Software Version 6 to measure decoding times. The transcode time is optional; as discussed earlier, this step can provide a codestream with an optimal ordering for the decompose stage. The time to write the bursts includes both file I/O and burst header generation since these steps are performed at the same time. The compose operation includes the time to read the bursts and compose a compliant JPEG 2000 codestream. The results clearly indicate that the additional decompose and compose steps that have been introduced in this paper incur minor additional computations considering the total processing time. A further breakdown of the decoding times for each burst is plotted in Fig. 7. It is shown that the decode time for each burst increases approximately linearly with the size of the composed codestream.

Table 2. Sample breakdown of processing times for encoder and decoder operations.

Server (encoder) operation		
	1.48 ms	Transcode
	7.02 ms	Decompose (average for each burst: 1.40 ms)
Client (decoder) operation		
	7.28 ms	Compose (average for each burst: 1.46 ms)
	78.5 ms	Decode (average for each burst: 15.70 ms)

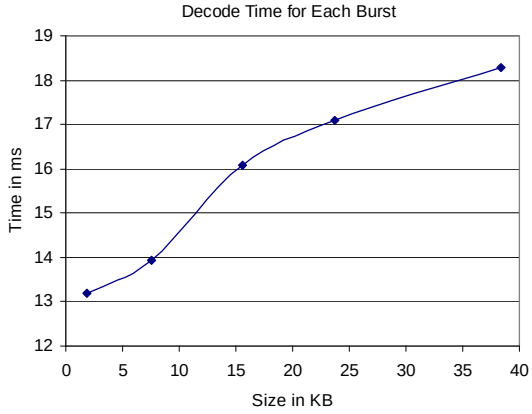
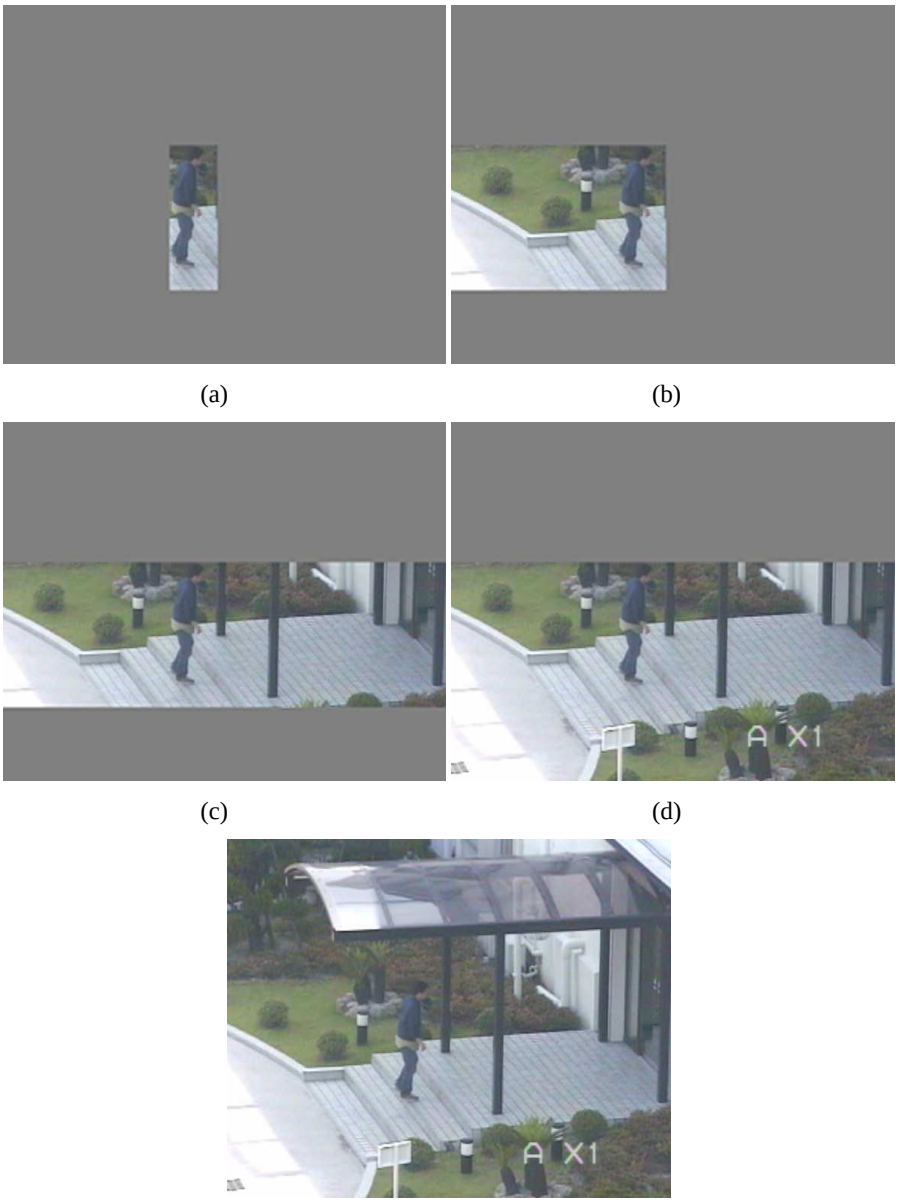


Fig. 7. Sample decode times of the five composed images following the progression order shown in Fig. 4.

4.3 Visual Effect

To illustrate the visual effect of the system, two progressive ordering configurations are provided. These examples are simply intended to demonstrate the visual reconstruction at various stages of the progression, and to also show the ability to support mixed progressions.

Fig. 8 shows the resulting images that are reconstructed at the client using the ROI scan defined in Fig. 4. This is an example of a simple progression order based on the position. The resulting decoded images associated with each burst are shown in Figs. 8(a)-(e). The total number of bytes corresponding to each burst is also indicated in the caption. From these bytes values and an assumed bit-rate for the channel, it is straightforward to calculate the transmission time for each burst. The method is easily extended for multiple ROI.



(a) (b)

(c) (d)

(e)

Fig. 8. Resulting decoded images after each successive burst from the ROI progression shown in Fig. 4. (a) First burst corresponding to 1854 total bytes, (b) second burst corresponding to 7568 total bytes, (c) third burst corresponding to 15620 total bytes, (d) fourth burst corresponding to 23721 total bytes, (e) fifth burst corresponding to 38340 total bytes.

Fig. 9 shows a progression of mixed position and quality. Fig. 9(a) shows the initially transmitted background of the image at a low quality level. Fig. 9(b) adds quality only to the ROI region. Fig. 9(c) then adds quality in the background based on the position near the horizon. Finally, Fig. 9(d) adds the remaining quality information which can be done in as many or few bursts as needed. Several progression steps have been omitted to save space, but generally speaking the technique can produce a wide number of progressive steps.

(a) (b) (c) (d)

Fig. 9. Resulting decoded images after each successive burst from an ROI-first quality progression order. (a) first burst for background corresponding to 4647 total bytes , (b) second burst for enhanced ROI corresponding to 8191 total bytes, (c) third burst for enhanced quality horizon corresponding to 13017 total bytes, (d) final burst for full quality image reconstructions corresponding to 38340 total bytes.

5. CONCLUDING REMARKS

A server-driven progressive image transmission scheme based on JPEG 2000 has been presented for applications that employ content-aware transmission techniques and do not require interaction with a client. Based on the transmission policy, codestream bursts are generated with inserted header information so that the codestream could be reconstructed in a progressive manner at the client side. It has been shown that minimal signaling overhead is required by this scheme, and that support for a wide range of progressive transmission policies could be provided.

REFERENCES

1. ISO/IEC 15444-1, “Information technology – JPEG 2000 image coding system – Part 1: Core coding system,” 2000.
2. D. Taubman and M. Marcellin, “JPEG 2000: Image Compression Fundamentals, Standards and Practice,” Kluwer Academic Publishers, Boston, 2002.
3. D. Taubman, “Remote Browsing of JPEG 2000 Images,” Proc. IEEE Int’l Conf. Image Processing, vol. 1, pp. 229-232, Rochester, NY, Sept. 2002.
4. ISO/IEC 15444-9, “Information technology – JPEG 2000 image coding system – Part 9: Interactivity tools, APIs and protocols,” 1st Ed., 2005.
5. D. Taubman and R. Prandolini, “Architecture, philosophy and performance of JPIP: internet protocol standard for JPEG2000,” Proc. SPIE Conf. on Visual Communications and Image Processing, San Jose, CA, Jan. 2003.
6. T. Hata, N. Kuwahara, D. Schwenke and A. Vetro, “Surveillance System with Mega-Pixel Scalable Transcoder,” Proc. SPIE Conf. on Visual Communications and Image Processing, San Jose, CA, Jan. 2007.
7. A. Vetro, D. Schwenke, T. Hata, N. Kuwahara, “Scalable Video Streaming Based on JPEG2000 Transcoding with Adaptive Rate Control,” Advances in Multimedia, Vol. 2007, Article ID 62094, doi:10.1155/2007/62094.