

# Adaptive Storage and Retrieval of Large Compressed Images\*

John R. Smith, Vittorio Castelli and Chung-Sheng Li  
IBM T.J. Watson Research Center  
30 Saw Mill River Road  
Hawthorne, NY 10532  
{jrsmith,vittorio,csli}@watson.ibm.com

## ABSTRACT

*Enabling the efficient storage, access and retrieval of large volumes of multi-dimensional data is one of the important emerging problems in databases. We present a framework for adaptively storing, accessing, and retrieving large images. The framework uses a space and frequency graph to generate and select image view elements for storing in the database. By adapting to user access patterns, the system selects and stores those view elements that yield the lowest average cost for accessing the multi-resolution sub-region image views. The system uses a second adaptation strategy to divide computation between server and client in progressive retrieval of image views using view elements. We show that the system speeds-up retrieval for access and retrieval modes such as drill-down browsing and remote zooming and panning and minimizes the amount of data transfer over the network.*

## KEYWORDS

Indexing and data organization, system optimization for search and retrieval, storage hierarchy, compressed-domain feature extraction and analysis, scalable approaches in storage and network delivery.

## 1 INTRODUCTION

There has been a recent interest in deploying image storage and retrieval systems that serve large images. The systems make available many types of images including satellite images,<sup>1</sup> aerial photographs,<sup>2</sup> medical images and high-resolution documents and maps. In many of these applications, the images have great spatial size ( $> 10K \times 10K$  pixels) and require significant storage space (10 MB to 1 GB per image). The large volume of data greatly complicates the handling of these images by the database systems. As a result, specialized solutions are needed for storing, accessing and retrieving these images.

The systems typically provide facilities for compressing, storing, accessing, analyzing, retrieving and querying the images. As shown in Figure 1, the applications need to provide access to multi-resolution sub-region views of the images. This requires that the systems be able to efficiently extract the views from the stored compressed data. Potentially, the methods used for partitioning, compressing and storing the images affect the efficiency of this type of access. The partitioning and compression schemes also affect the ability to do progressive retrieval. In these applications, the client applications retrieve, browse, zoom and pan around the images and remotely access views from across the network. Potentially, the capabilities of the client to cache image view data and synthesize the image views locally affects the efficiency of progressive retrieval. The system also typically provides query facilities, such as content-based query methods, that allow the users to search for images in the database.<sup>1</sup>

---

\*IS&T/SPIE Symposium on Electronic Imaging: Science and Technology - Storage & Retrieval for Image and Video Databases VII, Jan. '99

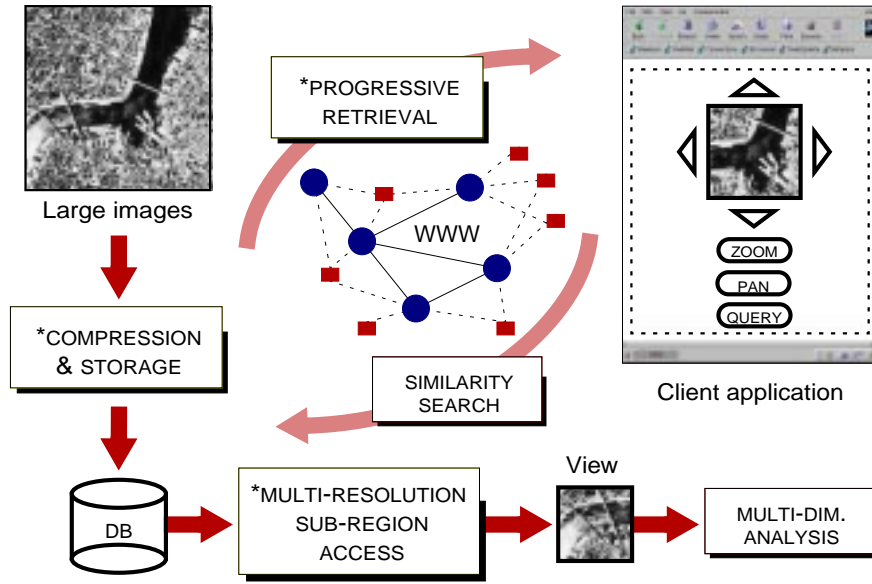


Figure 1: Large image storage and retrieval environment.

### 1.1 Image partitioning schemes

The partitioning of the image data is an important factor in determining the functionality and efficiency of the large-image storage and retrieval systems. For one, by breaking the images into smaller, more manageable units, the system more easily compresses, stores, accesses and retrieves the image data. However, no single partitioning scheme is optimal for all image storage and retrieval applications. Several partitioning schemes have been investigated that use wavelets, wavelet packets,<sup>3</sup> spatial grids, spatial quad-trees,<sup>4</sup> and the space and frequency graph (SFGraph).<sup>5</sup> Each offers different advantages for storage, access and retrieval.

Wavelets partition the images in spatial-frequency into subbands that are logarithmically spaced. The low-frequency wavelet subband serves as a coarse version of the image. On the other hand, spatial grids partition the images in space into equally sized tiles, where each tile corresponds to a spatial portion of the image. Flashpix combines the methods by partitioning the images into equal-volume tiles at different resolutions.<sup>6</sup> However, one drawback of Flashpix is that it expands the amount of data by 33%. The SFGraph partitions the images symmetrically in space and frequency. The SFGraph view elements correspond to wavelet subbands of portions of the images. The SFGraph embeds many image representations, such as the multi-resolution pyramid, wavelet, tiled-wavelet, wavelet packet, spatial quad-tree, and Flashpix, as shown in Figure 2.

The partitioning of the images affects the ability to access the image views. Since wavelets partition the images in spatial-frequency and not in space, they do not provide direct access to image sub-regions. Andresen, et al.<sup>7</sup> developed a method for extracting sub-regions from wavelet images by extracting and processing the sub-regions of the spatial-frequency subbands. This approach saves on computation, but the full subband data still needs to be accessed and de-compressed. Furthermore, the method introduces error at sub-region boundaries since the border extension used in the synthesis from sub-regions generally differs from the border extension used in the analysis of the whole image. This error has great potential to accumulate in progressive retrieval and synthesis of sub-regions at higher resolutions. By tiling the

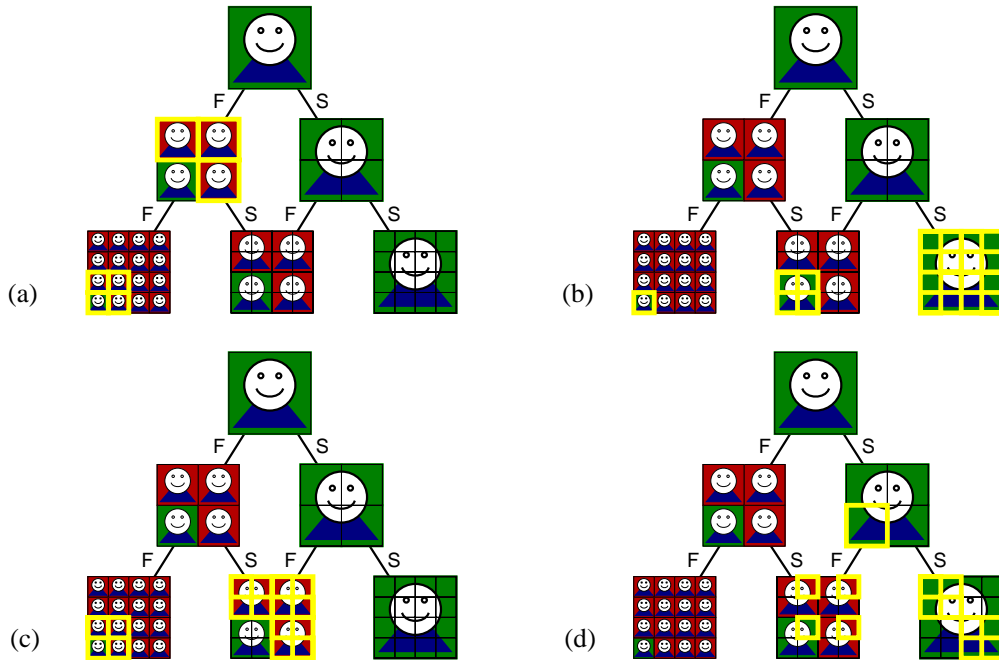


Figure 2: Image representations embedded in the space and frequency graph (SFGraph): (a) wavelets, (b) Flashpix, (c) tiled-wavelet, and (d) example complete and non-redundant set of view elements.

image simultaneously in space and spatial-frequency, the SFGraph provides more direct access to the multi-resolution sub-regions.

The partitioning of the images also affects the ability to progressively retrieve the image views. Since wavelets decompose images in their entirety, they allow the progressive retrieval of whole images. However, it is more difficult to use wavelets for progressive retrieval of sub-regions. On the other hand, Flashpix allows the re-use the image tiles at the client, but only for panning. Flashpix tiles cannot be used at the client to synthesize views at higher-resolutions. The SFGraph provides a more complete framework for progressively retrieving image views by retrieving and caching view elements and synthesizing the views at the client and server.

Since the SFGraph provides the most general image partitioning system, we use the SFGraph in an adaptive framework that optimizes the access and progressive retrieval of the image views. The framework adaptively partitions the images based on the patterns in which the users access the image views. The system selects the view elements from the SFGraph the give the lowest average view access cost. The framework also adaptively divides the work between server and client in progressive retrieval of image views. By caching SFGraph view elements at the client, the system determines the optimal view element retrieval and processing steps at the server and client to generate each requested image view.

### 1.2 SFGraph adaptive framework

In this paper, we present a new storage and retrieval system that adaptively stores, accesses and retrieves large images. Our objective is to manage large images in database applications that provide multi-resolution sub-region image access and progressive retrieval such as drill-down browsing, and

remote zooming and panning. The most significant advantages of the proposed framework are that it:

1. utilizes a multi-dimensional data partitioning scheme based on a space and frequency graph that embeds many other representation such as wavelets, wavelet packets, Flashpix, spatial quad-trees, and spatial gridding schemes;
2. enables rapid extraction of multi-resolution sub-region image views from the stored compressed view element data by adapting the partitioning to the access patterns;
3. enables image views to be efficiently retrieved in a progressive fashion by caching view element data at the client, and by adaptively dividing the view synthesis computation between the server and client.

### 1.3 Outline

We present the framework for the adaptive storage, access and retrieval of large images based on the space and frequency graph (SFGraph). In section 2, we describe an adaptive mechanism by which the SFGraph adapts to image view access patterns. We show that the adaptive framework achieves a great reduction in view access costs by adapting to known modes of image view access and to arbitrary access patterns. In section 3, we describe the process for adaptive view synthesis for progressive retrieval of image views. We show that the adaptive framework achieves great speed-ups in the progressive retrieval of image views and reduces the data transmission over the network.

## 2 IMAGE VIEW ACCESS

The system uses the SFGraph in an adaptive image partitioning process that adapts to patterns of image view access. We consider two cases: (1) when the modes of image view access are controlled by the system, and (2) arbitrary image view access. In the first case, the access modes are dictated by the application’s user-interface. For example, the user application may provide tools for zooming and panning and by default retrieve and display a low-resolution view of each image. The storage system uses this knowledge to estimate the access frequencies of the different multi-resolution sub-region image views. In the second case, access patterns are not known *a priori*.

The system observes the multi-resolution sub-region image view access patterns and reconfigures the image partitioning to adapt. In this case, the system initially stores a complete and non-redundant set of view elements that compresses the image well, as developed in.<sup>5</sup> In response to subsequent view requests, the system retrieves view elements from the database, and processes them in order to construct and deliver the view. Periodically, the system reconfigures the partitioning of the image in order to better match the access patterns and reduce the processing costs. Although we do not investigate it here, potentially, the optimization of the partitioning can be combined with optimization of the allocation of the view elements to different storage facilities, as in.<sup>8</sup>

### 2.1 View element intersection

The SFGraph provides the mechanism for constructing views from the compressed view elements stored in the database. Each view element corresponds to a tile in space and spatial-frequency and has a spatial location and size  $R_s = (x_s, y_s, w_s, h_s)$ , and a spatial-frequency location and size  $R_f = (x_f, y_f, w_f, h_f)$ . As such, each requested view corresponds to a region in space and spatial-frequency:  $R = \{R_s, R_f\}$ . The system uses this information to determine which stored view elements are needed to construct a requested view. For each image, the stored view elements form a complete and non-redundant tiling in space and spatial-frequency. Therefore, if a stored view element intersects with a requested view, it must be used in synthesizing the requested view. Given the request for a view



2. For each other view element  $v_i$  in the SFGraph, compute the processing cost  $C_{i,k}$  for supporting view  $v_k$ . The system derives this cost from the volume of the intersection of  $v_i$  and  $v_k$  in space and spatial-frequency.
3. Then,  $C_i = \sum_k p_k C_{i,k}$  gives the total support cost of each view element  $v_i$ .

### 2.3 View element selection

Given the assignment of support cost  $C_i$  to each view element  $v_i$ , the system then selects the complete and non-redundant set of view elements from the SFGraph that minimizes the total access cost for the population of queries. As illustrated in Figure 4(a), the selection of view elements from the SFGraph involves a three-way decision at each non-leaf view element as follows: (1) choose  $F$  (frequency partition), (2) choose  $S$  (spatial partition), or (3) choose self. The system carries out the selection procedure as follows:

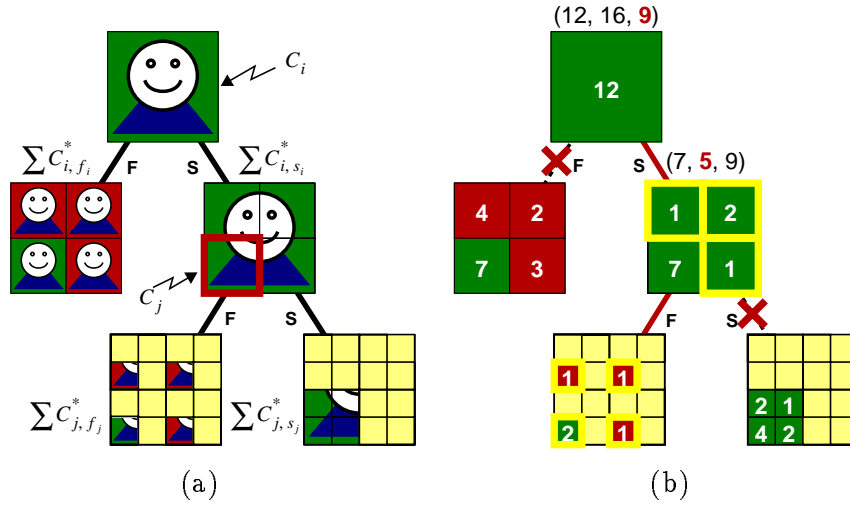


Figure 4: View element selection: (a) cost assignments, and (b) example minimum cost view element selection.

ALGORITHM 2 (VIEW ELEMENT SELECTION). *The complete and non-redundant set of view elements of least total cost is given by*

1. Assign cost  $C_i$  to each view element  $v_i$  using Alg. 1.
2. Start from the root view element  $v_0$  in the SFGraph, and recursively at each child view element  $v_i$ , choose the least cost path and let the optimal cost  $C_i^*$  corresponding to that path be given by:

$$C_i^* = \min(C_i, \sum C_{i,f_i}^*, \sum C_{i,s_i}^*),$$

where  $\sum C_{i,f_i}^*$  is the optimal total cost of the  $F$  child path,  $\sum C_{i,s_i}^*$  is the optimal total cost of the  $S$  child path, and  $C_i$  is the cost of  $v_i$ .

3. Mark which choice has lowest cost  $L_i = \operatorname{argmin}(C_i, \sum C_{i,f_i}^*, \sum C_{i,s_i}^*)$ .
4. Start again from the root view element  $v_0$  and follow the paths according to the marked choices  $L_i$ .

5. The set of terminal view elements encountered in the traversal form the complete and non-redundant set of lowest cost.

Figure 4(b) shows an example of the view element selection process, where the boxed view elements are selected as the ones with the least cost. Starting from the root view element, the  $S$  path has the minimum cost of 9. Along this path, replacing one of the children by its grandchildren along its  $F$  path gives a lower cost of 5. This recursive process prunes the SFGGraph and selects the least cost non-redundant and complete set of view elements.

#### 2.4 Access adaptation evaluation

We evaluate the access adaptation strategy as follows: we simulated the access modes illustrated in Figure 5 by randomly accessing views according to the corresponding access frequencies.

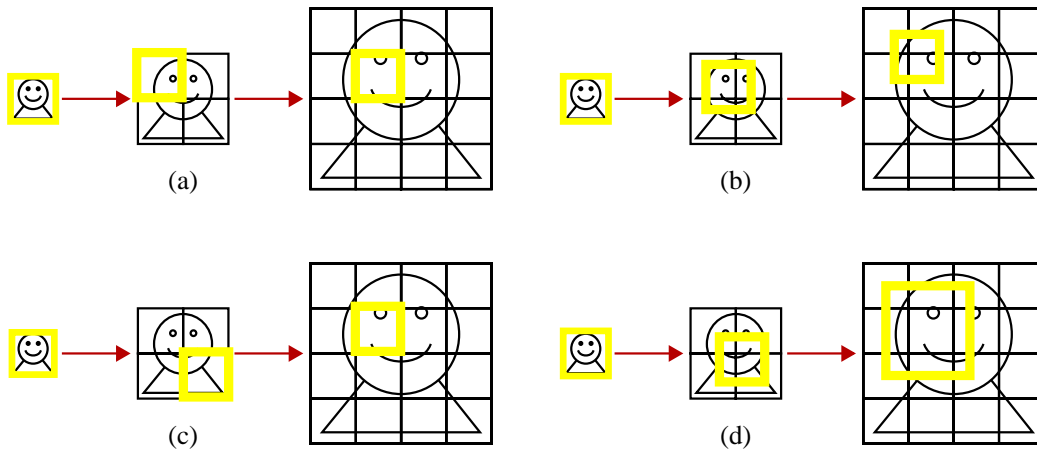


Figure 5: Four examples of view access modes for large images: (a) fixed-spatial grid drill-down, (b) arbitrary spatial drill-down, (c) equal probability view access, and (d) arbitrary multi-resolution access.

We repeated the experiments for the SFGGraph of different depths. The adaptation strategy achieved a significant reduction in view access cost using the adaptive approach for large images, as shown in Figure 6. We compared the view access performance to that of other image partitioning and storage schemes based on segments, blobs and wavelets. We did not consider Flashpix since it does not involve any processing in order to construct the views. Flashpix supports multi-resolution sub-region access by storing redundant views in the database. This eliminates view access costs but has penalties in view retrieval over a network, as we describe later.

The access cost reductions were obtained as follows:

1. **Fixed-spatial grid drill-down.** In this mode of access, as illustrated in Figure 5(a), the user drills-down from low- to high-resolution, accessing views that are restricted to a fixed-spatial grid. Each view at depth  $d$  in the SFGGraph has a frequency of retrieval of  $f = 0.25^d / (d + 1)$ . With this mode of access, the low-resolution image views are the most frequently accessed. The system adapts to this access pattern by storing a tiled-wavelet set of view elements (see Figure 2(c)). Figure 6(a) shows an access cost that is 0.4% to 4.5% of the other methods. Wavelets are more suited for this type of drill-down browsing than segments and blobs. However, the system determines that the tiled-wavelet set of view elements in the SFGGraph gives better access performance.

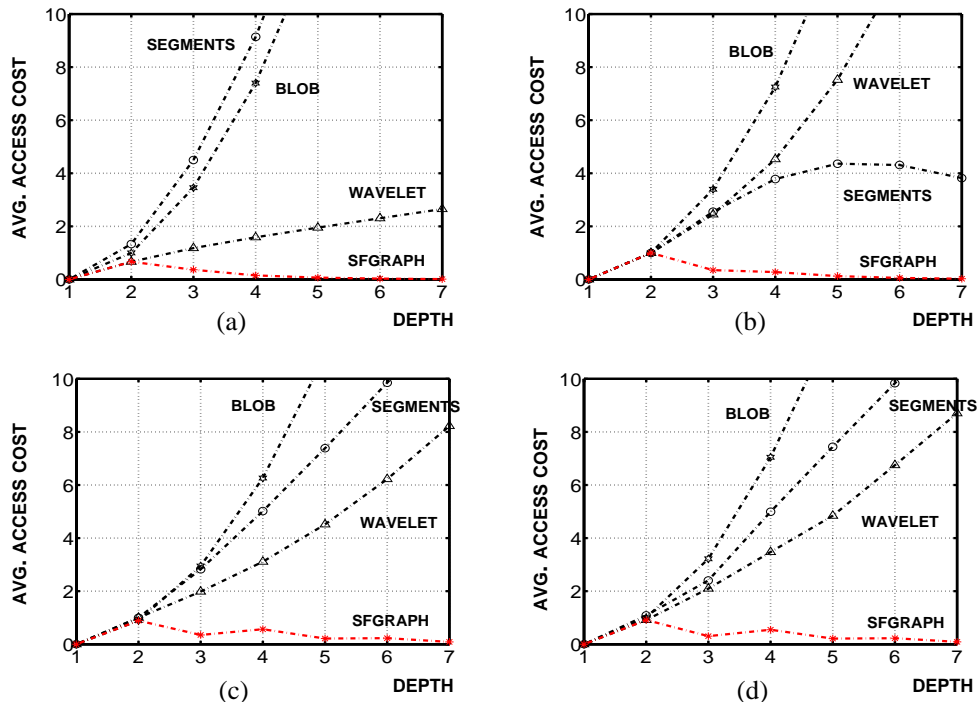


Figure 6: Evaluation of view access costs using adaptive SFGraph view element selection.

2. **Arbitrary spatial drill-down.** In this mode of access, as illustrated in Figure 5(b), the user drills-down from low- to high-resolution, accessing views of arbitrary location. Each image view at depth  $d$  has a frequency of retrieval of  $f = 0.5^d / (d + 1)$ . With this mode of access, the low-resolution image views are accessed with slightly lower frequency. The system adapts to this access pattern also by storing a tiled-wavelet set of view elements. Figure 6(b) shows an access cost that is 0.9% to 6.8% of the other methods. Storing spatial segments is somewhat better for this type of drill-down than wavelets. However, SFGraph performs significantly better than both methods.
3. **Equal probability view access.** In this mode of access, as illustrated in Figure 5(c), the user accesses all views of the image with the same frequency. Given an image with  $N$  views, each view has a frequency of retrieval of  $f = 1/N$ . When all views are equally likely to be accessed, the system, as shown in Figure 6(c) gives an access cost that is 7.2% to 10.7% of the other methods. Wavelets, segments and blobs are not well suited for supporting this type of access.
4. **Arbitrary multi-resolution access.** In this mode of access, as illustrated in Figure 5(d), the user drills-down into the images, but varies the spatial size and location of the zoom. In this case, we empirically measure the view retrieval frequency by observing the access pattern, as follows:  $f = n_k / (\sum n_i)$ , where  $n_k$  is the number of times view  $v_k$  is accessed. Given arbitrary access patterns, the system gives an access cost that is 3.4% to 10.2% of the other methods, as shown in Figure 6(d). To create these access patterns, we repeated several drill-down trials with random zooming and panning.

Figure 6 shows that the SFGraph adaptive partitioning framework achieves greater cost reductions

as the SFGGraph depth increases. This results from the SFGGraph providing more sets of view elements with increasing depth, and smaller view elements available are cheaper to process.

### 3 PROGRESSIVE RETRIEVAL

We extend the adaptation strategy to the progressive retrieval of image views. This differs from the access case in that both the server and client process the view elements. We also assume that the client has a cache for storing view elements. The problem of optimizing progressive retrieval comes down to determining the source for view elements (server database or client cache) and the location for processing (server and/or client).

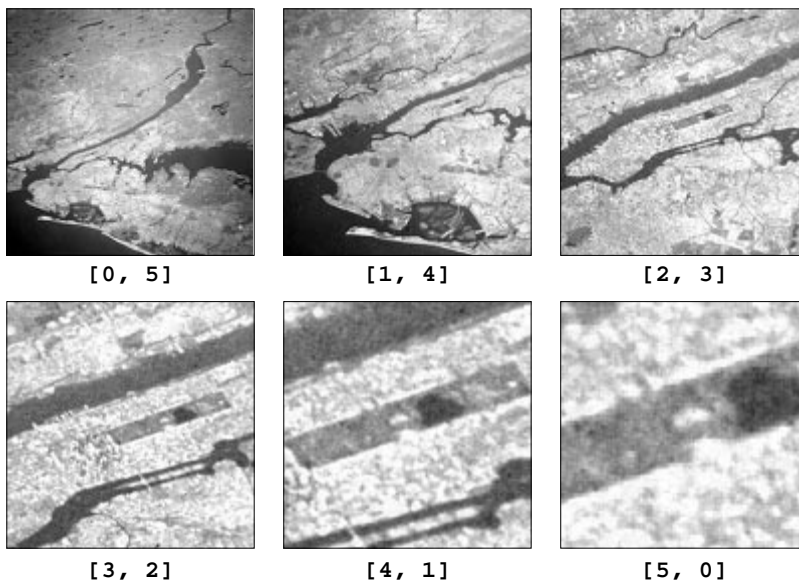


Figure 7: Example progressive retrieval in which the user zooms-in on a spatial location.

For each image, the database stores a complete and non-redundant set of view elements. The user issues a request to the server for an image view, say view  $[0, 5]$  in Figure 7. Initially, we assume the client cache is empty. The server responds by retrieving the appropriate view elements from the database. There are two options for delivering the view. The server can process the view elements locally and transmit the view. Alternatively, the server can transmit the view elements directly, and the client can synthesize the view. The system chooses the option that results in the fastest retrieval of the view. As the user clicks on areas of the image and zooms-in, the system chooses among the different alternatives for accessing, caching, retrieving the view elements and synthesizing the views.

#### 3.1 Client caching and synthesis

In the case that the client caches view elements and synthesizes the views locally, the server needs only to transmit the necessary residual view elements in each request. For example, if the user requests a new view, say view  $[1, 4]$  in Figure 7 by zooming-in, the client re-uses the cached view elements from view  $[0, 5]$ . As illustrated in Figure 8, the client retrieves these from the cache, and processes them in combination with new view elements retrieved from the server to synthesize  $[1, 4]$ . The user subsequently drills-down into the image to retrieve views  $[2, 3] \dots [5, 0]$ . Performing the view synthesis at the client minimizes the amount of data transmission over the network. For example, in the three zoom-in steps

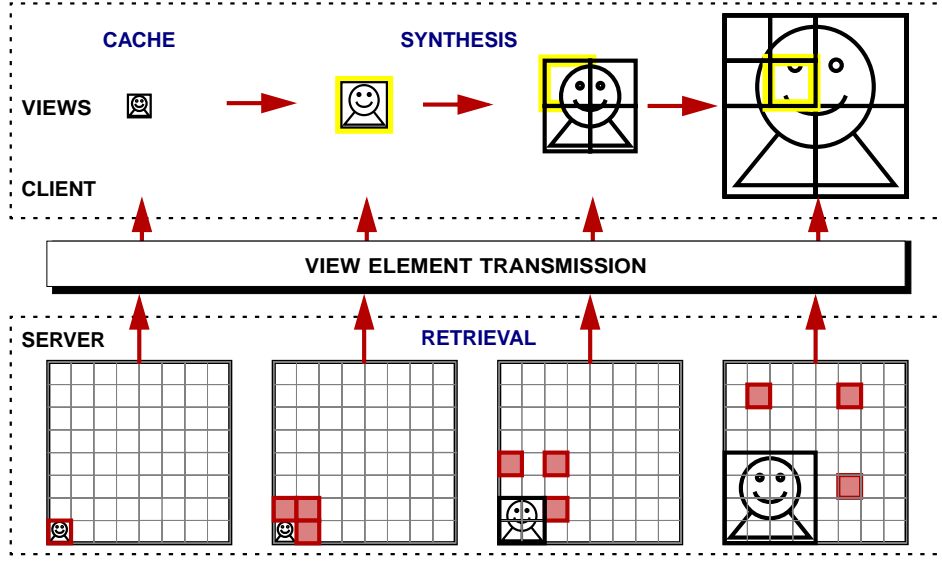


Figure 8: Progressive retrieval with caching of view elements and view synthesis at the client.

illustrated in Figure 8, this approach achieves a 25% data transmission reduction, compared to retrieving the views directly. Further data reduction is realized in practise when accounting for compression since the view elements compress relatively better than the view data.

### 3.2 Cooperative view synthesis

In general, both the server and client participate in the view synthesis process. In this case, the server potentially pre-processes some of the view elements before transmitting them to the client. The optimal division of the work depends on the relative processing power of the server and client and the network bandwidth. We determine the optimal division of work between client and server for each client request of view  $v_k$  as follows:

**ALGORITHM 3 (PROGRESSIVE RETRIEVAL ALGORITHM).** *The steps for dividing the work between the server and client in progressive retrieval of image views are:*

1. At the server ( $s$ ), compute the access cost  $C_n^s$  of all view elements  $v_n$  using Alg. 1. Compute the cost  $C_n^t$  for transmitting each view element  $v_n$  to the client from its volume and network bandwidth.
2. At the client ( $c$ ), assign an access cost  $C_n^c = 0$  for each of view element  $v_n$  in the client cache, otherwise  $C_n^c = \infty$ .
3. Then, at the client, starting from the root view  $v_0$  in the SFGGraph, and recursively for each child view element  $v_i$ , choose the least cost synthesis path and assign the optimal cost  $\bar{C}_i$  as follows:

$$\bar{C}_i = \min(C_i^c, \sum \bar{C}_{i,f_i}, \sum \bar{C}_{i,s_i}, C_i^t + C_i^s),$$

where  $\sum \bar{C}_{i,f_i}$  is the total optimal cost of using view elements on the  $F$  children path to synthesize view  $v_i$ , and  $\sum \bar{C}_{i,s_i}$  is the total optimal cost of using view elements on the  $S$  children path,  $C_i^t + C_i^s$  is the cost of accessing  $v_i$  from the server, and  $C_i^c$  is the cost of accessing  $v_i$  from the client cache.

4. Mark which choice has lowest cost  $L_l = \operatorname{argmin}(C_l^c, \sum \bar{C}_{l,fi}, \sum \bar{C}_{l,si}, C_l^t + C_l^s)$ .
5. Start again from the root view  $v_0$  and follow the paths according to the marked choices  $L_l$ . Retrieve and process the selected view elements accordingly to synthesize the view  $v_k$ .

### 3.3 Progressive retrieval evaluation

We evaluate the retrieval adaptation strategy by simulating the random zooming and panning of the large images by a remote client. We vary the relative processing power of the server and client, and vary the transmission bandwidth. Figure 9 shows the comparison of the adaptive strategy (“adapt”) where both client and server participate in synthesizing views to strategies where only the client (“client”) or server (“server”) synthesize the views.

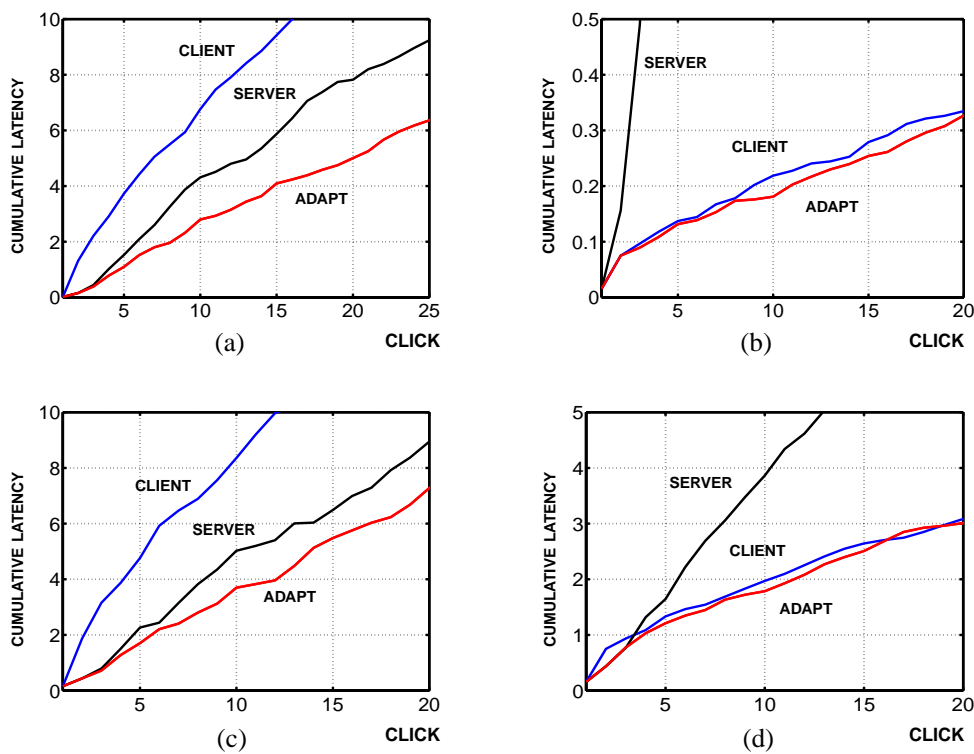


Figure 9: Progressive retrieval with caching of view elements at the client and adaptive partitioning of view element synthesis between server and client: (a) thin client, (b) powerful client, (c) think client and low bandwidth, (d) normal client and low bandwidth.

In each user click shown in Figure 9, the user randomly zooms-in, zooms-out or pans up, down, left or right. The results in Figure 9 show that the adaptive strategy minimizes the cumulative latency in progressively retrieving the views over the network. The first experiment, Figure 9(a) involves a thin client with  $1/10^{\text{th}}$  the processing power of the server. Performing all of the processing at the client is not optimal. On the other hand, performing all of the work at the server does not take full advantage of the view elements in the client cache. The optimal strategy adaptively partitions the SFGraph view synthesis between server and client. The next experiment, Figure 9(b), involves a powerful client with  $10\times$  the processing power of the server. In this case, it is much better to perform synthesis at the client than the server. In this case, the partitioning scheme adapts to the situation and provides the

lowest cumulative latency. Figure 9(c) shows the results for a thin client with low-bandwidth, where the adaptive scheme favors synthesis at the server. Figure 9(d) shows the results for a normal client with low bandwidth, where the adaptive scheme favors synthesis at the client.

We also consider the overall data transmission required for different image partitioning schemes. In this experiment, we used the adaptive progressive retrieval scheme to optimally divide the work between the server and client. We measured the amount of data transmitted over the network for each click in the random zooming-in, zoomin-out and panning by the user. The SFGraph required an average of only 450 bytes of data transmission per click in a trial of twenty clicks. Storing the images using a wavelet partitioning required an average of 7,966 bytes per click. Storing the whole image as a blob required an average of 12,025 bytes per click. Storing the Flashpix representation at the server required an average of 4,505 bytes per click.

#### 4 SUMMARY

We presented an adaptive system for the storage, access and progressive retrieval of large images. The framework utilizes a space and frequency graph for generating and selecting view elements to be stored in the database. The system adaptively selects the view elements that yield the lowest average multi-resolution sub-region view access cost and the fastest progressive retrieval to remote clients. We showed that by integrating the adaptation strategies using the space and frequency graph across the facilities for storage, access and retrieval, we improve access and retrieval performance for large images.

#### 5 REFERENCES

- [1] V. Castelli, L. D. Bergman, C.-S. Li, J. R. Smith, and A. Thomasian. Progressive content-based retrieval of satellite image database through Internet. In *I.E.E.E. Tyrrhenian Intern. Workshop on Digital Comm.*, Ischia, Italy, September 1998.
- [2] T. Barclay, R. Eberl, J. Gray, J. Norlinger, G. Raghavendran, D. Slutz, G. Smith, and P. Smoot. Microsoft terraserver white paper. <http://www.terraserver.com>, 1998.
- [3] R. R. Coifman and M. V. Wickerhauser. Entropy-based algorithms for best basis selection. *IEEE Trans. Inform. Theory*, 38(2), March 1992.
- [4] E. Shusterman and M. Feder. Image compression via improved quadtree decomposition algorithms. *IEEE Trans. Image Processing*, 3(2):207 – 215, 1994.
- [5] J. R. Smith and S.-F. Chang. Joint adaptive space and frequency graph basis selection. In *IEEE Proc. Int. Conf. Image Processing (ICIP)*, Santa Barbara, CA, October 1997.
- [6] Eastman Kodak Company. *Flashpix format and Architecture White Paper*, July 17 1996.
- [7] D. Andresen, T. Yang, D. Watson, and A. Poulakidas. Dynamic processor scheduling with client resources for fast multi-resolution www image browsing. In *Proc. Intern. Parallel Processing Symposium (IPPS)*, 1997.
- [8] S. Prabhakar, S. Agrawal, A. El Abbadi, A. Singh, and T. R. Smith. Browsing and placement of multiresolution images on secondary storage. Technical Report TRCS96-22, UCSB, 1996.
- [9] J. R. Smith, V. Castelli, A. Jhingran, and C.-S. Li. Dynamic assembly of views in data cubes. In *Proc. ACM Principles of Database Systems (PODS)*, pages 274–283, June 1998.