Performance Analysis of Algorithms for Retrieval of Magnetic Resonance Images for Interactive Teleradiology

M. Stella Atkins, Robert Hwang and Simon Tang,
School of Computing Science, Simon Fraser University, Burnaby, BC V5A 1S6, Canada.

ABSTRACT

We have implemented a prototype system consisting of a Java-based image viewer and a web server extension component for transmitting Magnetic Resonance Images (MRI) to an image viewer, to test the performance of different image retrieval techniques. We used full-resolution images, and images compressed/decompressed using the Set Partitioning in Hierarchical Trees (SPIHT) image compression algorithm. We examined the SPIHT decompression algorithm using both non-progressive and progressive transmission, focussing on the running times of the algorithm, client memory usage and garbage collection. We also compared the Java implementation with a native C++ implementation of the non-progressive SPIHT decompression variant.

Our performance measurements showed that for uncompressed image retrieval using a 10Mbps Ethernet, a film of 16 MR images can be retrieved and displayed almost within interactive times. The native C++ code implementation of the client-side decoder is twice as fast as the Java decoder. If the network bandwidth is low, the high communication time for retrieving uncompressed images may be reduced by use of SPIHT-compressed images, although the image quality is then degraded. To provide diagnostic quality images, we also investigated the retrieval of up to 3 images on a MR film at full-resolution, using progressive SPIHT decompression. The Java-based implementation of progressive decompression performed badly, mainly due to the memory requirements for maintaining the image states, and the high cost of execution of the Java garbage collector. Hence, in systems where the bandwidth is high, such as found in a hospital intranet, SPIHT image compression does not provide advantages for image retrieval performance.

1. INTRODUCTION

Teleradiology systems facilitate electronic transmission of medical images from one location to another, for the purposes of interpretation and/or consultation [1]. In a typical client-server teleradiology system, a radiologist uses the client computer to view images that have been transmitted over a network from a server. Hence a client/server teleradiology system must allow the client to efficiently retrieve and display medical images. Radiographic images vary widely in their sizes, and large data sets such as are encountered in magnetic resonance images (MRI) may take a long time to transmit over a computer network with limited bandwidth. Some teleradiology systems utilize image compression to solve this problem [18][13][6].

In this project, we studied the performance of the Set Partitioning in Hierarchical Trees (SPIHT) image compression algorithm for interactive client-server teleradiology. We chose the SPIHT algorithm as it has been shown to produce high quality compressed medical images [10][3], and the algorithm also has the property of progressive reconstruction, whereby the decompression algorithm can truncate the data stream at any point to generate an approximation of the original image. Furthermore, the quality of the image can be refined by using additional data from the stream (Fig. 1).

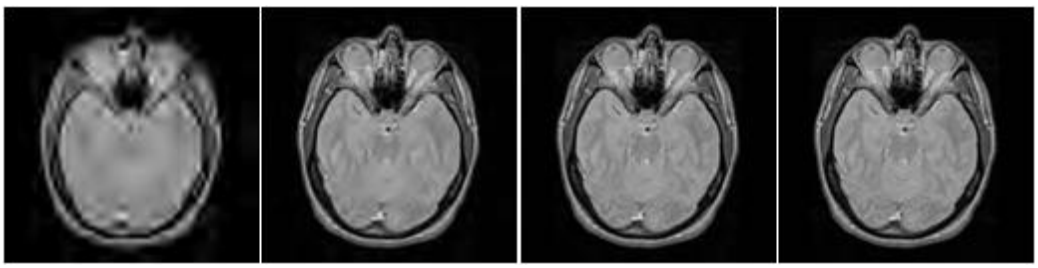


Figure 1. A progressively transmitted image. These images were generated from a single data stream using 512, 2048, 4096, and 6144 bytes from left to right.

Visualization, Display, and Image-Guided Procedures, Seong Ki Mun, Editor,
Proceedings of SPIE Vol. 4319 (2001) © 2001 SPIE · 1605-7422/01/$15.00

2. RELATED WORK

This work is concerned with the application of the SPIHT compression algorithm to teleradiology. The inventors of the algorithm give performance results for the compression and decompression of images on a single computer [14]. Manduca et al. [10] and Ericksen et al. [3] give favourable results for applying the algorithm to medical images.

Aside from the SPIHT algorithm, there are many wavelet-based image compression algorithms, and reports of these algorithms often include some computational performance results, but they all use a single computer for such evaluations. Examples include a space-frequency segmentation method to compress ultrasound images [2], and a comparison of the AWIC algorithm to the JPEG algorithm [8]. Wu et al. [18] give round-trip results for a teleradiology system that uses images compressed with a readily-available program called *gzip*. Gzip uses a lossless, dictionary-based compression technique. The underlying network was Fast Ethernet (100Mbps). Reponen et al. [13] analyze the performance of a mobile teleradiology system where the client is connected to the server using a cellular modem. The test images were compressed using JPEG. Maldjian et al. [9] use a simple wavelet-based compression technique to compress MR images, and give round-trip performance results for retrieving these images using a modem. The author also gives results for using the wavelet compression technique, followed by compression using gzip. Png et al. [12] study the performance of using an Integrated Services Digital Network (ISDN) connection for transmitting uncompressed CT and MR images. Nagata et al. [11] use JPEG compression, and measure the performance over the Internet between sites that are separated by various distances and interconnected by a communications network with varying capabilities. Hercus et al. [4] consider the computation and communication tradeoffs in a client/server image retrieval system, comparing the UNIX utility *compress*, the lossless JPEG algorithm, and the CALIC algorithm [17] to compress satellite images in their study. They build a model of a client/server system and consider the performance of the algorithms using various network bandwidths.

None of these systems have studied the particular demands of viewing MR images in a Java client, particularly with a progressive decompressor. This research is based on our earlier study [15] using the same Java-based MR image viewer, but the earlier work used a non-Java, non-progressive SPIHT image decompressor.

3. METHODS

We analyzed the performance of the SPIHT image compression algorithm in a teleradiology application, using a server to provide MRI data sets, and a client-side decompressor written in Java. We used the MR image viewer developed by [16], augmented to be a client/server application by [15]. This viewer uses a detail-in-context algorithm to solve the problem of limited screen real-estate [7]. The MR image viewer initially displays the images of a volume as a tiled data set at a reduced scale, and the user may expand on one or more of these images as desired (Fig 2) by selecting it. For convenience, we refer to the first, scaled-down image as the *initial image*, and the selected image as the *expanded image*.

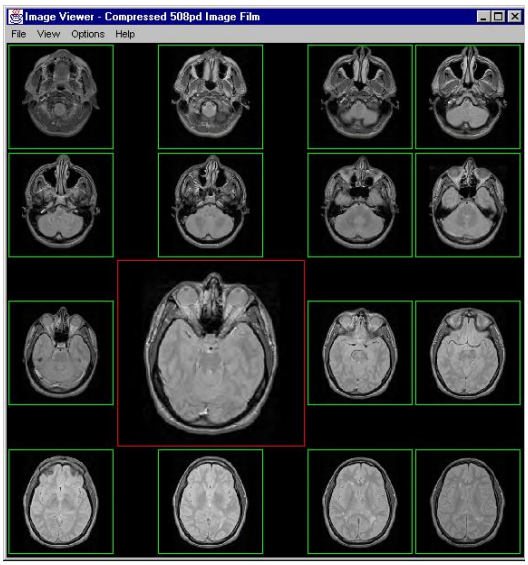


Figure 2. The Image Viewer after selecting a focal node (enlarged).

The client/server system may be implemented in several ways, where:

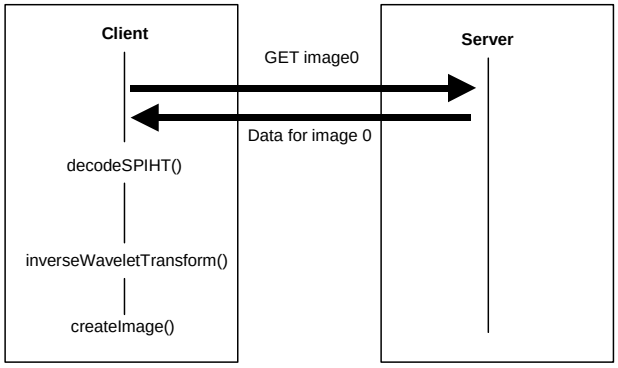
1. The server sends the uncompressed image data set for the client to scale to the appropriate size for display at the size of the initial image.
2. The server sends a complete pre-compressed non-progressive SPIHT image for each initial image, which has to be scaled by the client as above, after decompression.
3. The server scales the images and sends them at the appropriate size to the client. When a user expands an image, the server sends the full, uncompressed image for that expanded image to the client. The client must then decompress and display the expanded image.
4. The server sends a progressive SPIHT image stream, where the initial image is generated from the first N bytes, where N is determined by some criteria to be a sufficient number of bytes to reproduce an acceptable scaled-down image. If the expanded image is requested, an additional M bytes starting from byte N+1 of the stream are used to reconstruct the expanded image. Again, M is determined by some subjective criteria, described in [5].

3.1 Implementation

We implemented the SPIHT algorithm using the Java language as a set of classes suitable for encoding and decoding an image. The client was connected over a dedicated 10Mbps Ethernet to the server and the images were requested using the hypertext transfer protocol (HTTP). The client uses an HTTP GET request to retrieve each image from the web server. One thread of control made sequential image requests, so multiple requests did not overlap. Full details are given in [5].

We used identical PC computers at both client and server. Although their speed was only 133 MHz, tests on a faster computer (550 MHz) showed an almost linear improvement in cpu speed for SPIHT decompression with cpu power, while the network components remained constant. Both computers had 32 Mbytes RAM.

3.2 SPIHT compression



We measured the sequence of events for retrieving an individual image using SPIHT image compression, shown in Fig 3.
Figure 3. Sequence of HTTP method calls for SPIHT image decompression.
The response to the HTTP GET request is read into a buffer. The *decodeSPIHT()* method decodes the data, and returns a two dimensional array of wavelet coefficients. The 2D IDWT is performed on this array to reconstitute the image data. This data requires some additional processing (such as conversion from floating point to integer values) to be converted into an object of the *java.awt.Image* class. The Java environment renders this image object to the screen.

3.3 Progressive SPIHT compression

As shown in Figure 4, the client uses an additional thread of control for each image that it loads. When a specified number of bytes are read from the embedded stream, the data is passed to the SPIHT decoding thread. The number of bytes is configured on the server and is chosen to provide enough compressed image data to reconstruct an image with adequate fidelity (see [5] for details on how we chose the number of bytes).

After processing the data, the thread is suspended. The main thread performs a 2D IDWT on the result from the SPIHT decoding to produce the image data. Additional data from the embedded stream may be passed to the SPIHT decoding

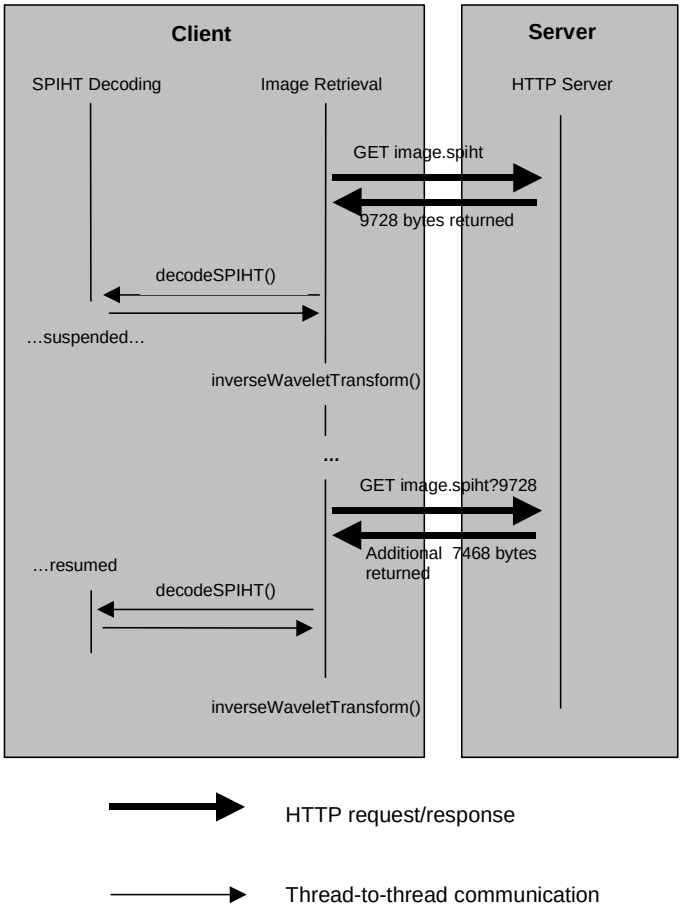


Figure 4. HTTP and intraprocess method call sequence for progressive SPIHT image decompression.

thread, which resumes processing the data from the point when it was suspended. Once it completes processing the additional data, the thread is suspended and a 2D IWT is performed to produce the updated image.

3.4 Performance Evaluation

We measured several system components, detailed in [5], by instrumenting the web server and the client to report execution times for 10 iterations of each display of SPIHT-compressed images, with a cache flush between each iteration. We also tested the Java system with and without garbage collection. For progressive SPIHT reconstruction, we obtained timings for loading the images initially, and also for expanding 1,2 or 3 focal images. Finally, we measured the memory usage during execution of the Java decoder.

4. RESULTS AND DISCUSSION

4.1 Pre-compressed SPIHT images

The total time required to retrieve and reconstruct 16 MR images compressed with the SPIHT algorithm is given in Fig 5. The individual components are itemized. The main points to note are that the client-side garbage collection becomes increasingly significant as the compressed image stream size increases, and, as expected, SPIHT decoding time increases linearly with image size.

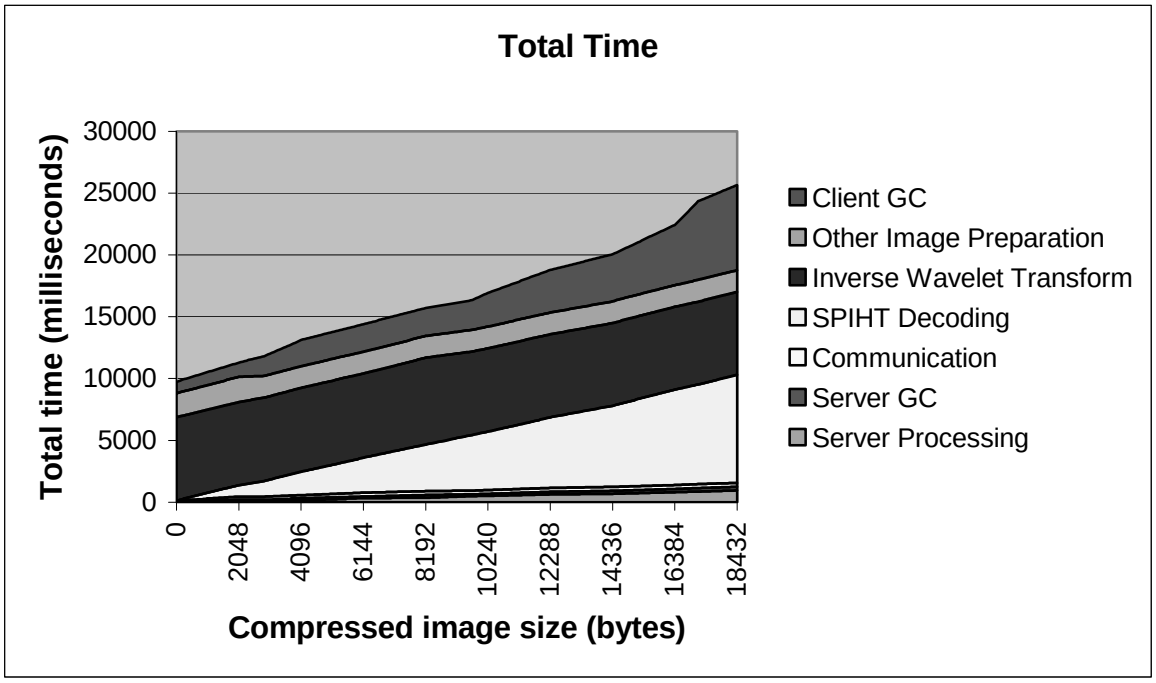


Figure 5. Total time required to retrieve and reconstruct 16 MR images decompressed with the Java SPIHT algorithm.

4.2 Progressively Transmitted SPIHT images

The detail-in-context algorithm displays MR images scaled down to the viewing area, and expands an image or a set of images, when requested by the user. We used this property as a vehicle for experimenting with the progressive transmission capability of the SPIHT algorithm. That is, we use an initial portion of the compressed stream to view the initial scaled-down image and we retrieve an additional portion of the stream to render an expanded image.

We initially intended to perform this experiment using 16 images. However, the SPIHT algorithm requires that we utilize a substantial amount of memory in order to use the progressive transmission capability; this is a limitation of this technique. As a result, the operating system on the client machine frequently swapped memory to secondary storage. We found that the effects of paging distorted the results substantially and therefore we reduced the memory requirements by performing the experiment using eight images instead of sixteen.

Surprisingly, we see that by using progressive image compression, we have not saved any time (Table 1). In fact, the total amount of time is substantially higher. Furthermore, for zero foci the amount of time to load eight images at 1.19 bpp (9728 bytes) using progressive image transmission is almost the same as the amount of time to load eight images at 2.09 bpp (17152 bytes) using SPIHT compression without the progressive transmission capability.

Table 1. Comparison of times in milliseconds progressive and non-progressive SPIHT compression using 8 images and between 0 and 3 foci.

Number of Foci	SPIHT Compression		*SPIHT Progressive Compression*	
	Mean	**Standard Deviation**	**Mean**	**Standard Deviation**
0	12870	1590 (12.3% of Mean)	12250	520 (4.2% of Mean)
1			13170	350 (2.7% of Mean)
2			15340	170 (1.1% of Mean)
3			17170	670 (3.9% of Mean)

Figure 6 shows the amount of time required for each step of the image reconstruction process. Note, client-side garbage collection accounts for the largest proportion of the total time for zero, one, two, or three foci. For each image, we must retain a large amount of data to maintain the state of the SPIHT algorithm between each successive progressive decompression call. In contrast, this memory may be disposed of when we do not need to support progressive decompression.

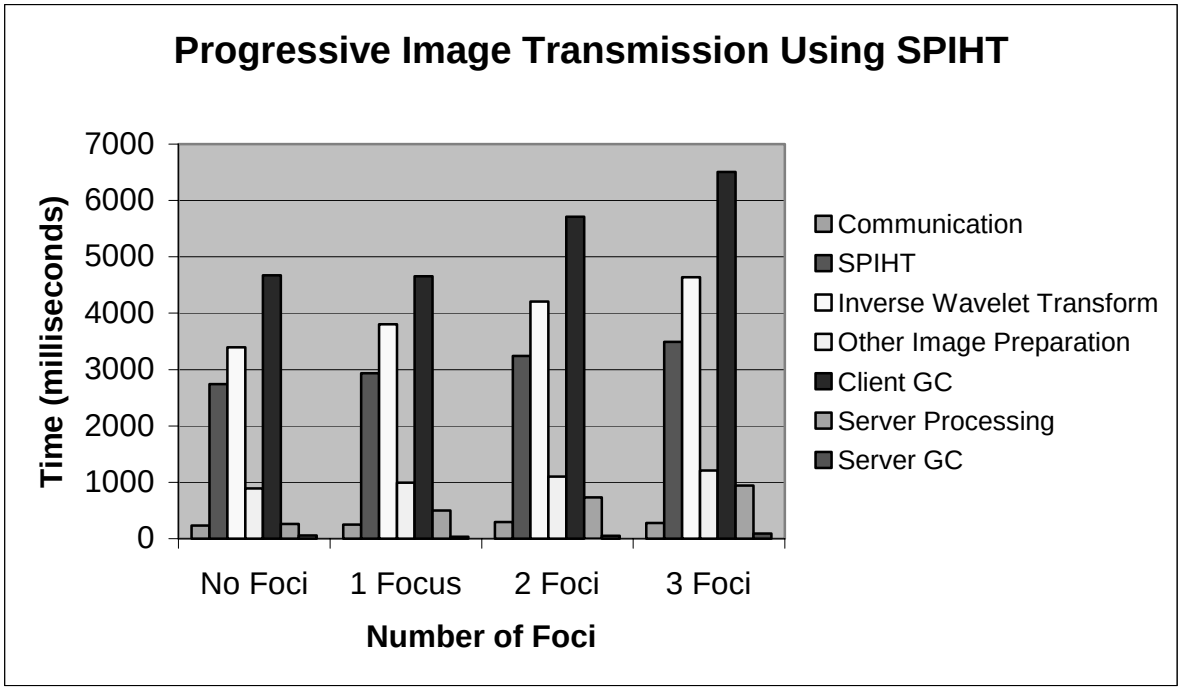


Figure 6. Comparison of factors affecting total time to retrieve and reconstruct 8 images using the SPIHT compression algorithm. We use progressive image transmission to improve the quality of an image when one, two, or three foci are expanded by the user.

4.3 Comparison with other techniques

We compare the performance of this Java SPIHT algorithm against the full-resolution, scaled, and native SPIHT technique results detailed in [15]. The timings are directly comparable because identical hardware configurations were used.

Table 2 compares [15] against the SPIHT image compression technique without making use of progressive image transmission. The table shows the total time required to retrieve and reconstruct sixteen MR images, and also the total time required to expand between one and three foci. Only the values in the "Scaled" column vary depending on the number of foci as it is the only technique in this table which requires additional communication with the server. Not surprisingly, this Java SPIHT algorithm underperforms all of these techniques from [15]; the Java version is almost twice as slow as the native (C++) implementation. Also note that the native implementation includes an additional arithmetic compression step which is not performed with the Java implementation. This figure is consistent with the 57% speed-up found by Bolin [Bolin97] when comparing the performance between C++ and Java programs. However, the convenience of running a Java decompressor in a web browser may mean that this option is still viable, especially with the huge increase in processor speeds (10 times) likely cheaply available.

Table 2. Comparison of image transmission times in milliseconds for SPIHT image compression against three techniques from [15].

# of Foci	Full-Resolution Uncompressed		Scaled		Native SPIHT (0.5 bpp)		Java SPIHT (0.5 bpp)	
	Avg.	Std.	Avg.	Std.	Avg.	Std.	Avg.	Std.
0	9355	803	4418	264	6994	82	13110	433
1			5260	238				
2			5441	296				
3			5894	246				

4.4 Progressively Transmitted SPIHT images

We compare the Progressive SPIHT image compression technique against [15] in Table 3. Since we reduced the total number of images in a film to eight for the Progressive SPIHT technique, we have recalculated the totals from [15] to only include the timings for the same eight images.

Table 3. Comparison of image transmission times in milliseconds for Progressive SPIHT image compression against two techniques from [15].

# of Foci	**Full-Resolution Uncompressed**		*Scaled*		**Progressive SPIHT 1.19 bpp for 0 foci 2.09 bpp for expanded foci**	
	Avg.	**Std.**	**Avg.**	**Std.**	**Avg.**	**Std.**
0	4509	630	2180	250	12249	519
1			3022	318	13171	351
2			3203	258	15339	172
3			3656	259	17167	674

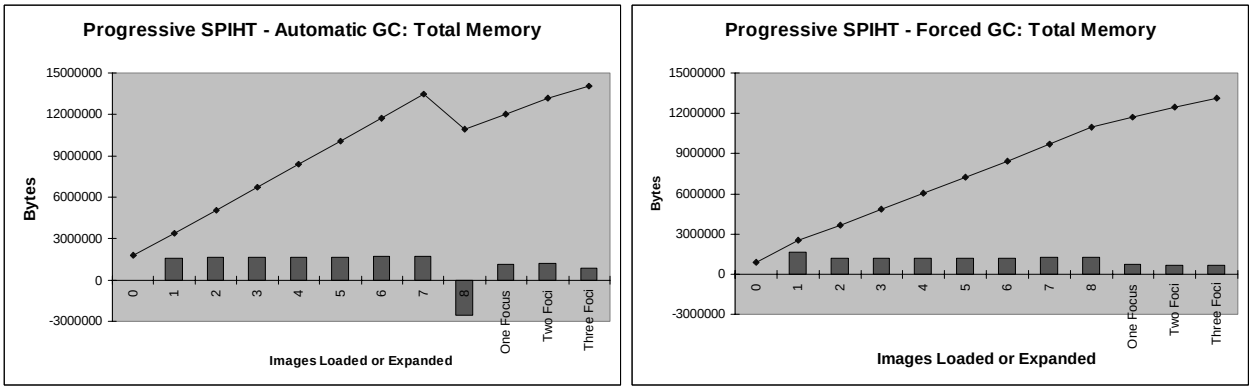


Figure 7. Average total memory usage and change in memory usage per image. Eight images were initially loaded at 1.19 bpp (9728 bytes) and then three images were expanded to a bitrate of 2.09 bpp (17152 bytes). We used automatic garbage collection (left) and forced garbage collection (right).

4.5 Memory Usage

Using the progressive SPIHT decoding technique with 8 initial images and expanding 3 images, we obtain memory usage results in Figure 7. When we use automatic garbage collection, our data shows that the total memory used decreases after loading the 8th image. We attribute this to a garbage collection phase that reclaimed some memory.
With forced garbage collection, the 8 initial images each require an average of 1,260,000 bytes. Loading the first image appears to result in a noticeably larger allocation of memory, which may be due to startup costs. Disregarding the first image, the average memory requirement of loading one image is 1,210,000 bytes.

Expanding the three foci requires an average of 709,000 bytes per focus. The total memory used when we expand the last focus is approaching 14 megabytes, which is the maximum size of the heap that we allowed for our experiments. This limit was chosen because larger heap sizes resulted in substantial hard drive activity due to memory swapping.

We see the progressive SPIHT decoding algorithm consumed a substantial amount of memory. In an environment where garbage collection is used for memory management, this can decrease overall performance due to additional processing by the collector. Even in an environment where this is not the case, an implementation of the algorithm must deal with the memory consumption issue. Some explanations for these results are given in [5]. It may be possible to optimize the data structures in ways that can reduce memory usage.

Alternatively, we believe that instead of saving the state, we may be able to reconstruct the state from the decompressed image. We could perform a forward wavelet transform on an image and execute the SPIHT encoding algorithm to reconstruct the state of the decoding algorithm at the point when decompression was halted. This would require that we remember the number of bits that were used to decompress the initial image and that we use a wavelet transform that does not suffer from round off errors such as the integer wavelet transform.

5. CONCLUSIONS AND FUTURE WORK

We implemented the SPIHT image compression algorithm in the Java programming language for a client/server MR image viewing application. We performed experiments and found a number of factors that affect the performance of the system. In order to take advantage of progressive image decompression, we needed to maintain intermediate state information about the algorithm. This state information consumed a substantial amount of memory, leading us to modify one of our experiments to use fewer images. Software developers should be aware of this when considering the progressive SPIHT algorithm in their design, unless huge amounts of local memory is available.

In an environment that uses garbage collection, the large amount of memory that is required to maintain state also results in longer garbage collection phases. In our progressive image compression experiments, garbage collection was the largest

factor in the overall time required to reconstitute an image. Since there are a variety of garbage collection algorithms, performance will differ depending on the environment.

The SPIHT algorithm is computationally intensive. Although we found that system performance was poor under our experiment's conditions when compared to the performance of the application when using uncompressed images, we note that if the underlying communication network is slower (as is common in teleradiology applications) and/or computation power is increased, then the SPIHT algorithm should perform better relative to using uncompressed images.

Future Work

Some of the garbage collection and memory usage problems with progressive image compression may be alleviated by reconstructing the state of the SPIHT algorithm when needed instead of maintaining it in memory. It would be useful to examine this further.

Another possible technique for improving system performance is to make use of the communication link while it would otherwise be idle. For example, after the Image Viewer application has loaded and rendered the initial images using the Progressive SPIHT technique, the communication link is not used until the user expands an image. During this idle time, the Image Viewer could preload and/or decompress additional image data from the server in advance of the user's selection and expansion of images. Other teleradiology applications could employ a similar technique, depending on their user interface, if they use the Progressive SPIHT algorithm.

With respect to the general applicability of progressive image compression to teleradiology, we have examined the performance of the SPIHT algorithm and by examining its limitations, we suggest a constraint which we might apply to determine how suitable other algorithms are for this application. In particular, the image compression algorithm should allow progressive decompression without excessive memory requirements and should maintain a minimal amount of state between successive reconstruction of images. Future work can be done to examine other progressive image compression techniques with the goal of determining criteria for teleradiology applications. These criteria could form the basis for the design of future image compression algorithms.

REFERENCES

[1] American College of Radiology, "ACR Standard for Teleradiology", http://www.acr.org, 1998.

[2] Chiu, E., Vaisey, J., Atkins, S., "Joint Space-Frequency Segmentation, Entropy Coding and the Compression of Ultrasound Images", Proceedings of ICIP 2000, Vancouver Sept. 2000, to appear.

[3] Erickson, B.J., Manduca, A., Palisson, P., Persons, K.R., Earnest IV, F., Savcenko, V., Hangiandreou, N.J., "Wavelet Compression of Medical Images," *Radiology*, 1998, pp. 599-607.

[4] Hercus, J.F., Hawick, K.A., "Compression of Image Data and Performance Tradeoffs for Client/Server Systems", Technical Report DHPC-029, Department of Computer Science, University of Adelaide, Australia.

[5] Hwang, R. "An analysis of the SPIHT image compression algorithm for real-time teleradiology", M.Sc. Thesis, Simon Fraser University, 2000. http://www.cs.sfu.ca/~stella/papers/RobertHwangMSc.ps

[6] Kim, N.H., Yoo, S.K., Kim, K.M., Kang, Y.T., Bae, S.H., Kim, S.R., "Development of a Medical Record and Radiographic Image Transmission System Using a High-Speed Communication Network", MEDINFO: Proceedings of the World Conference on Medical Informatics, Volume 9 part 1, 1998, pp. 282-285.

[7] Kuederle, O., Atkins, M.S., Inkpen, K.M., Carpendale, M.S.T., "Evaluation of Viewing Methods for Magnetic Resonance Images", to appear in SPIE Medical Imaging, Volume 4319-64, February 2001.

[8] Lepley, M.A. and Forkert, R.D., "AWIC: Adaptive Wavelet Image Compression", MITRE Technical Report, MTR 97B0000040, The MITRE Corporation, 1997.

[9] Maldjian, J.A., Liu, W.C., Hirschorn, D., Murthy, R., Semanczuk, W., “Wavelet Transform-Based Image Compression for Transmission of MR Data”, American Journal of Roentgenology, 169(1), pp. 23-26.

[10] Manduca, A., Said, A., “Wavelet Compression of Medical Images with Set Partitioning in Hierarchical Trees”, Proceedings of Medical Imaging 1996: Image Display, SPIE Volume 2702, pp. 192-200.

[11] Nagata, H. and Mizushima, H., “A remote collaboration system for telemedicine using the Internet”, Journal of Telemedicine and Telecare, 4(2) 1998, pp. 89-94.

[12] Png, M.A., Kaw, G.J.L., Tan, K.P., Koh, B.H., Ang, L., Filut, J., “Remote consultation for computerized tomography and magnetic resonance studies by means of teleradiology – experience at the Singapore General Hospital”, Journal of Telemedicine and Telecare, 3 Supp 1 1997, pp. 54-55.

[13] Reponen, J., Ilkko, E., Jyrkinen, L., Karhula, V., Tervonen, O., Taitinen, J., Leisti, E-L., Koivula, A., Suramo, I., “Digital wireless radiology consultations with a portable computer”, Journal of Telemedicine and Telecare, 4(4) 1998, pp. 201-205.

[14] Said, A., Pearlman, W., “A New Fast and Efficient Image Codec Based on Set Partitioning in Hierarchical Trees”, IEEE Transactions on Circuits and Systems for Video Technology, Vol. 6, June 1996.

[15] Tang, S., “Towards Real-Time Magnetic Resonance Teleradiology over the Internet”, M.Sc. Thesis, Simon Fraser University, 1999. ftp://fas.sfu.ca/pub/cs/theses/1999/SimonTangMSc.pdf

[16] van der Heyden, J., Atkins, M.S., Inkpen, K., and Carpendale, M.S.T., “MR image viewing and the screen real estate problem”, Proceedings of the SPIE-Medical Imaging 1999, 3658: 370-381, Feb. 1999.

[17] Wu, X., Memon, N., Sayood, K., “A context-based, adaptive, lossless/nearly-lossless coding scheme for continuous-tone images”, ISO/IEC JTC 1/SC 29/WC 1 document No. 202, July 1995.

[18] Wu, T-C., Lee, S-K., Peng, C-H., Wen, C-H., Huang, S-K., “An Economical, Personal Computer-based Picture Archiving and Communication System;, Radiographics 1999, 19, pp. 525-530.