Two Approaches for Applet-based Visible Human Slice Extraction

Sebastian Gerlach*, Roger D. Hersch*
Ecole Polytechnique Fédérale de Lausanne

ABSTRACT

Real-time interactive slicing is a tool of choice for exploring 3D anatomic data sets such as the Visible Human. We offer real-time slicing on the Web by partitioning the application between a Java applet (the client) and the Web server. Two approaches for partitioning the work between the client and the server are presented and compared. In the first approach, we transfer complete compressed slices from the server to the client. In the second approach, we successively build a local cache on the client by transferring small subvolumes of increasing resolution from the server to the client. The client is responsible for extracting the displayed slices from the local data cache. The Web-based Real-time Visible Human navigator can be accessed at our Visible Human Web site at http://visiblehuman.epfl.ch. A high-bandwidth network connection is recommended. The web server offering real-time interactive slicing together with anatomic structure labeling opens new perspectives for teaching anatomy to paramedical and medical professions.

Keywords: Web visualization, volume graphics, client-server work partitioning, Visible Human, real-time navigation

1. INTRODUCTION

Anatomic structures are often visualized by cross-sections similar to the ones printed in an anatomic atlas [5]. The Visible Human dataset, produced by the National Library of Medicine's Visible Human Project [1], provides an excellent resource for generating digital cross-sections. It consists of transverse CT, MRI, and cryosection imagery of a man and woman. The cryosection datasets provide high-resolution full-color photographs of transverse sections of the human body, representing 13 GB of data for the Visible Man, and 40 GB of data for the Visible Woman. For the Visible Man, this dataset is a volume of 2048 x 1212 x 1871 voxels, each voxel representing 0.33 x 0.33 x 1 mm. The Visible Woman dataset has a higher resolution, with voxels 0.33 x 0.33 x 0.33 mm in size. The other datasets (CT, MRI) provide lower resolution grayscale volumes.

The Visible Human dataset is used by many institutions for research and teaching purposes [9]. However, working with the full dataset on a workstation is cumbersome and requires advanced programming skills. By offering access services to the Visible Human data on the Web, a much larger public of students, professionals and researchers can benefit from this instructive anatomical resource.

The first Web application to allow extraction of slices perpendicular to the main axes was the NPAC Visible Human viewer applet [6]. The more recent Visible Human Slice and Surface Server [3] provides access to arbitrarily oriented and positioned slices and surfaces, as well as to slice sequence animations. These applications require that the user first defines the position and orientation of the slice he wishes to view before getting the resulting cross-section or animation a few seconds later.

We have recently presented a first approach for enabling real-time interactive slicing on the web [10]. This Java application provides the possibility to navigate within the Visible Human dataset by continuously extracting slices at a speed of several slices per second. It uses a custom protocol based on TCP sockets, which allows us to carry out time budget allocation and image quality management strategies.

In order to offer real-time navigation capabilities, we created a client-server protocol where the quantity of visual information sent from the server to the client can be adjusted depending on the network data rate and the desired frame display rate. Transmitted visual information is reduced by compression and sampled at a resolution that enables the information to fit the target size of the reply message and therefore respect as much as possible the desired frame display rate. In this paper, we show that faster navigation capabilities can be provided if the client also incorporates a

* Sebastian.Gerlach@epfl.ch; Roger.Hersch@epfl.ch; http://visiblehuman.epfl.ch

local cache containing multiresolution volumic information. This article presents the original approach, as well as the new client cache based approach and evaluates the results obtained with both techniques.

In section 2, we describe the principles of slice extraction from the Visible Human data set. In sections 3 to 6, we describe solutions for partitioning the navigation application between a Web client and a Web server. In sections 7 to 9, we evaluate the performance of the proposed solutions and in section 10, we describe the server's behavior when serving multiple clients simultaneously. The conclusions are drawn in section 11.

2. VISIBLE HUMAN SLICE EXTRACTION

The original Visible Human datasets are provided as collections of axial cross-section bitmaps. In the present section we show how this data is stored and processed in order to provide arbitrarily positioned and oriented slices for real-time navigation.

2.1. DATA STORAGE

The Visible Human dataset is stored as small subvolumes called extents. This data subdivision was selected in order to ensure that a constant amount of data has to be loaded for a given slice independently of its orientation. The size and compression of the extents is adapted to the storage and transfer requirements of the application. In addition to the standard full resolution dataset, multiple resolution versions of the Visible Human are also stored, at reduction factors of 2, 4, 8, 16, and 32. The number of bytes per extent is kept constant across all resolutions. Lower resolution extents therefore represent a larger volume of the data set. On disk, extents are stored sequentially, with the lowest resolution extents (highest reduction factor) stored first.

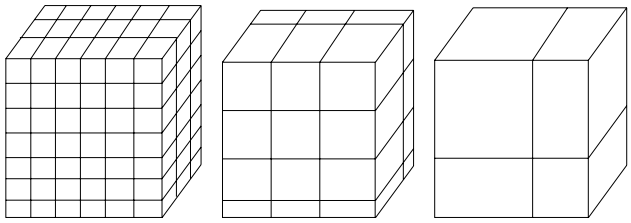

Figure 1. Subdivision of a volume dataset into extents of decreasing resolution

2.2. SLICE EXTRACTION ALGORITHM

The slice extraction algorithm must be able to produce an arbitrarily oriented and positioned slice from the dataset. The slice to be rendered is defined by three vectors as illustrated in Fig. 2.

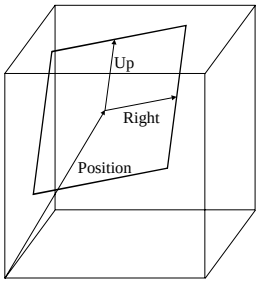

Figure 2. Definition of a slice in 3d space using 3 vectors

The slice resampling is carried out using an incremental fixed-point algorithm. The rendering starts at the top-left corner of the slice, and the 3D coordinates of the corresponding point in the dataset are evaluated. The nearest voxel (nearest neighbor interpolation) or surrounding voxels (trilinear interpolation) are retrieved from the dataset. The current coordinate is then incremented using the right and up vectors of the current slice visualization parameters in order to fully traverse the requested slice.

To provide acceptable speed, the extents that have been used are stored in a memory cache. Extent caching is effective since the position and orientation of slices varies in small increments during interactive navigation.

3. CLIENT-SERVER APPLICATION PARTITIONING

We present two approaches for dividing the application between a Java client applet and the Web server. In the first approach, the server extracts slices, compresses them, and streams them to the client. This is a 'fat server' approach, since the client only has to decompress and display the arriving slices. In the second approach, the server streams complete compressed extents to the client, which decompresses and stores them in its local cache and carries out the slice extraction. This is a 'fat client' approach, since the client does all the computations required for slice extraction, and the server essentially transfers complete extents to the client.

Possible compression standards that were considered for slice, respectively extent compression are the emerging JPEG 2000 standard based on wavelet compression and the older JPEG standard. Although JPEG 2000 could provide higher quality at an identical compression ratio [2], its processing requirements are much higher. In addition, standard JPEG compression is easy to use, libraries are available for encoding, it can be decoded quickly with pure Java code, and, due to its independent macroblock structure, it provides a simple mechanism for transfering small or partial images.

The JPEG header information containing the Huffman tables and other static information is transmitted only once at application startup. This header is 574 bytes in size. During the rest of the application, all image data is transferred as blocks of 16x16 pixels. This corresponds to groups of 6 macroblocks in JPEG, with four macroblocks for the luminance channel at full resolution, and two macroblocks for the two chrominance channels at half resolution.

These 'JPEG blocks' are compressed from 768 bytes to 37 bytes on average. This compression ratio of 20 was reached with the Independent JPEG Group's libjpeg JPEG compression library [4] by using a quality of 75 on the IJG JPEG quality scale (ranging from 0 to 100). The source data was the complete Visible Human data set, with all areas fully outside the body removed. The effective compression ratio that is obtained depends on the source data, since the compressor strives for constant quality rather than constant compression ratio.

The decompression is carried out on the client using a custom JPEG decompressor written in Java. This JPEG decompressor is based on code developed at USC [7], and was modified to accommodate the JPEG block-based streaming decompression required by this application. Tests have shown that the quality loss due to compression is not noticeable for real-time navigation within the Visible Human data set.

4. BASIC CLIENT-SERVER INTERACTION SEQUENCE

The client-server interaction sequence must enable the client to display slices according to the user's positioning requests as quickly as possible and with a constant display frame rate. To ensure optimal use of the available network bandwidth and to avoid receiving out of date information, the server must transmit exactly the amount of data that can be transferred in the time between two displayed frames on the client. The basic client-server interaction pipeline is illustrated in Fig. 3.

The client emits requests periodically (1). The interval between requests corresponds to the desired frame display rate. These requests contain the parameters for the slice currently requested by the user, i.e. the 3 vectors defining the slice position and orientation as well as an identifier and the viewport parameters. If the request is identical to the previous one, a shorter slice request with only the request identifier is sent to the server. The request also contains the maximal allowed message size for the reply. The client adjusts the allowed reply size in order to provide the best trade-off between response time and image quality.

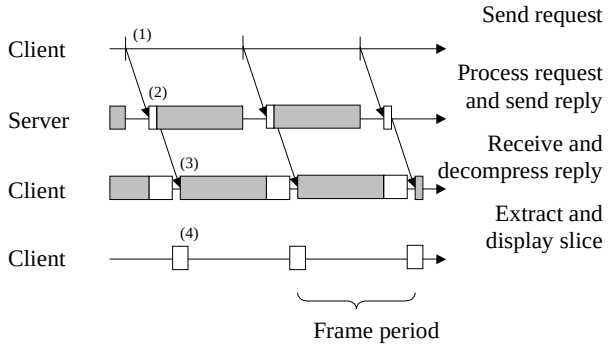

Figure 3. Client-server interaction pipeline (in gray: time for sending data, in white: processing time)

The server receives a request from the client, processes it, and starts sending data back as soon as possible (2). The response should contain no more bytes than the maximum fixed in the client's request. Any excess data sent would arrive after the client has sent its next request. This would produce a backlog of data in the client's input buffers and an additional delay between the client request and the display of the corresponding slice. To prevent data transfer backlogs, the client stops sending requests when there are more than two unanswered requests in circulation.

The client starts decompressing the received data in a separate thread (3), and displays it after processing is finished (4).

5. SLICE BASED SERVER-CLIENT TRANSFER

Transferring compressed slices between server and client is extremely simple, since all processing steps are carried out on the server. When the server receives a request, it extracts the slice from the dataset, compresses it, and sends it back to the client. The client decompresses the slice and displays it.

On the server, the dataset is available on disk stored in extents of 32 x 32 x 16 voxels, at resolutions ranging from full resolution to a resolution reduced by a factor of 32. At this lowest resolution, the complete dataset fits into 2 x 1 x 4 extents and voxels are 10.56 x 10.56 x 16 mm in size. If the required extents for the requested slice are not yet in the cache, they are loaded, and the slice is extracted by applying trilinear interpolation. The resolution of the extracted slice is selected according to the type of request sent by the client. If it is a new request (i.e. the user has moved the slice frame and the slice vectors have been sent), the slice is extracted at a resolution allowing the compressed version to fit into the maximum allowed reply size. If it is a continuation request (i.e. no user interaction has taken place), a full resolution slice is extracted, and sent back in parts as successive reply messages. The client always displays whatever data has been received after decompression. Therefore the final high-resolution slice is progressively built up. If a request for a new slice is received before this transmission of parts is complete, the transfer of the full resolution slice is aborted, and a new low resolution slice is transmitted.

The slice is sent as a collection of JPEG blocks, with a header indicating to which request it corresponds. When the slice has to fit into a single reply message (i.e. when the slice frame is moving), its resolution is computed based on an estimated compression ratio of 14:1, yielding a JPEG block size of 55 bytes. Assuming a square aspect ratio, the formula for calculating the transmitted slice size is:

$$slicesize = 16 \cdot \mathrm{floor}\left(\sqrt{\frac{replysize}{55}}\right) \qquad (1)$$

For example, for a square aspect ratio image and a reply size of 4000 bytes, the transmitted slice has a resolution of 128x128 pixels. The JPEG block size used for slice size computation is somewhat above average; the effective size obtained oscillates between 35 and 65 bytes depending on the content of the slice[1].

[1] In the rare cases where the average compressed JPEG block size for a slice is higher than 55 bytes, the reply will be longer than requested, and the client's effective display frame rate will be slightly reduced.

When the users stops moving the slice frame, a full resolution image is transmitted, taking as many reply slots as required. For a full resolution slice of 384 x 384 pixels and a reply size of 4000 bytes, the complete transmission of the full frame typically takes up to 6 reply slots: the client sees the high resolution image building up over 6 frames. The response time is the sum of the delays induced by the system (network, server processing, client processing).

6. EXTENT BASED SERVER-CLIENT TRANSFER

Instead of transferring slices, complete extents can be transferred from the server to the client. The client needs to cache the extents and extract slices, therefore requiring higher processing power and an increased memory size.

To make this model viable on lower bandwidth network connections, the extents have to be kept small in order to avoid transferring too much data that is not immediately required for the current slice. With JPEG block compression, the size of horizontal extent slices must be a multiple of 16 x 16 pixels. In order to retain an equilibrated aspect ratio and dimensions of powers of two, we choose an extent size of 16 x 16 x 8 voxels. This extent size corresponds to a subvolume of 5.33 x 5.33 x 8 mm in the Visible Human dataset.

When the server receives a request from the client, it evaluates all the extents intersected by the slice at all the available resolution levels. It then builds a list of all required extents that are not yet in the client's cache, sorted with the lowest resolutions first. The extents from the beginning of the list are then sent to the client. The number of extents that are sent is limited by the maximal allowed reply size, which in turn limits the quality of the image that the client can produce after decompression and subsequent slice extraction. Fig. 4 illustrates this transfer process for three successive slices with a slice frame movement between the first two requests.

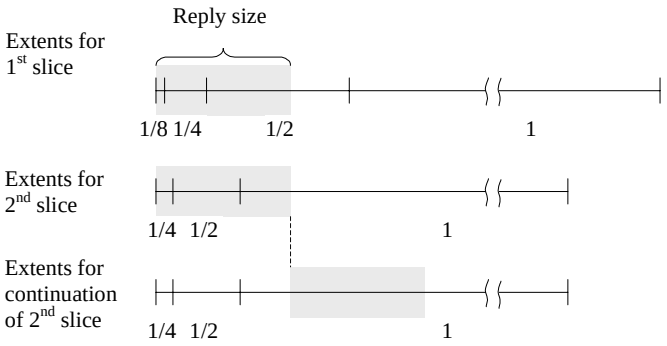

Figure 4. Extent transfers at different resolutions (fractions of full resolution) for three successive slice requests with a slice frame movement between the first two requests

The second slice requires less lower resolution extents to be transferred, since they are already available from the previous slice and can be reused. As opposed to the slice-based transfer, the quality on the client is not necessarily consistent from one slice to the next; it heavily depends on the content of the local extent cache, i.e. how many extents can be reused from the previous slice requests.

The server is responsible for managing the client's extent cache. This allows avoiding transmitting extents twice, and also simplifies the processing of incoming extents on the client, since the cache slot to be occupied is already indicated by the server.

Since the client has a local cache of extents, it can extract and display slices at any arbitrary moment. A slice can always be produced, even after very large jumps in the dataset, since the extents at the lowest resolution, i.e. at a reduction factor of 16, are always kept in the client's cache. When looking for the voxels corresponding to a given pixel in the slice, the highest resolution extent available in the client's cache is used. The highest-quality result is obviously obtained when the slice is displayed after reception and decoding of the extents of the current reply message. In this case we have the same latency penalty as for the slice based transfer. Thus, a compromise can be made between the quality of the displayed slice and the interaction latency. If the slice is extracted and displayed before all extents in the reply have been received, the remaining extents are still decompressed and stored, since the following slice will probably also intersect them.

With extent-based data transfer between the server and client, the time taken to produce a full resolution slice on the client can be fairly long, since transferring full resolution extents may require 30 times more data than transferring a simple compressed slice. However, when performing small slice frame movements, as is common when searching for small anatomical structures, extents are already in the client's cache and the slices are immediately available at full resolution.

Many Java Virtual Machines limit the amount of memory an applet may allocate to a few megabytes. This prevents the extent-based applet from functioning correctly on some system configurations, since it requires more memory for its local extent cache.

7. IMAGE QUALITY METRICS

To evaluate the image quality of slices produced with the Web-based Visible Human navigator, we propose a simple metric based on the resolution of the source voxels used for generating the slice. The following equation gives the quality of an extracted slice:

$$Quality = 1 - \frac{\sum_{j=1}^{h}\sum_{i=1}^{w} psr_{i,j}}{4 \cdot w \cdot h} \qquad (2)$$

where *psr* is the base 2 logarithm of the reduction factor[1] of the voxel that served as source for the given pixel, i.e. reduction factor = 2^{psr} . Variables *w* and *h* are the width and height of the displayed slice. Thus, a slice display with half resolution would have a quality of 0.75, quarter resolution would give a quality of 0.5, one-eighth resolution a quality of 0.25, and one sixteenth of the original resolution would obtain a quality of 0. This correspondence between the quality metric and slice resolution gives a direct indication of the size of structures that can be identified at a given quality level.

This quality metric can be evaluated without having to carry out a comparison with a reference image, and is also very easy to compute. It can thus be evaluated in real-time in the navigator. This is not the case for traditional metrics such as RMS error, which require a reference image and are much more compute intensive.

8. PERFORMANCE OF THE SLICE-BASED TRANSFER APPROACH

The performance of the slice-based transfer model is straightforward. For a given network data rate *ndr* (the sustained throughput available on the client's network connection in the server-client direction) and a desired frame rate *fps* (the frame rate requested by the client), it is possible to evaluate the reply size *rs*:

$$rs = \frac{ndr}{fps} \qquad (3)$$

The resolution of the image produced during interactive slice frame movement, which fits into the reply message size, is computed according to Eq. 1. Eq. 1 and 3 show that there is a fundamental compromise between the image quality (depending directly on the reply size *rs*) and the frame rate *fps*. The network data rate *ndr* cannot be controlled, since it depends on the network infrastructure between client and server. The tradeoff between frame rate and image quality is left to the user with a slider allowing him to control the reply size between 1 KB and 32 KB. Corresponding slider endpoints are characterized by 'Highly interactive' and 'High quality'. Fig. 5 shows the expected frame rates for various network data rates and reply sizes (the frame rate has an upper bound of 10 frames per second).

[1] In slice-based server-client transfer, the image size is computed so as to fit the maximum message size. The reduction factor is given by the transferred image resolution divided by the full image resolution

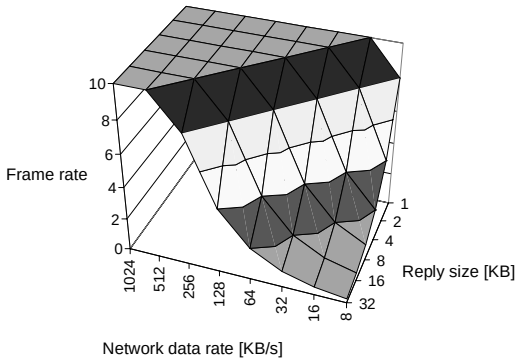

Figure 5. Frame rate for various network data rates and reply sizes

The effective frame rate obtained on the client is limited by one of three factors:

- The network bandwidth between client and server
- The time required to decompress a frame, which depends on the processing power of the client
- The desired frame display rate set on the client

The interaction response time on the client is the sum of the following time intervals:

1. Network latency for sending a request, ranging from less than 1 ms on a local area network, to over 100 ms on slow connections, typically 70-80 ms for long distances[2].

2. Transfer of the request from the client to the server, which depends on the effective network throughput. Request size is 48 bytes, typically requiring 2-3 ms with a data rate of 128 KBits/s.

3. Processing of the request (slice extraction and compression), which depends on the server processing power and disk throughput, typically 10-20 ms for serving a request from cache on a Pentium III at 733 MHz.

4. Network latency for sending the reply, same as 1.

5. Transfer of the reply from the server to the client, which depends on the network throughput. The reply size is adjusted to make the transfer time last about as long as the frame display period. Decompression starts as soon as the first data chunk arrives, i.e. the client does not wait for the full reply before starting the decompression. Decompression time is dependent on the client's processing power. A Pentium III at 733 MHz can decompress 1000 JPEG blocks per second, which corresponds to roughly 40 Kbytes of incoming data per second (depending on the compression ratio). Since the decompression is generally faster than the network data transfer, the decompression can occur simultaneously with the transfer.

Experiments have shown that latencies of 250 ms still allow a very good interaction for navigating in the Visible Human dataset. Since frames are displayed after slice arrival and decompression and since no more than two requests may be pending at any time, the effective frame rate adapts itself to the instantaneous network throughput.

Table 1 shows the obtained image quality when navigating at a fixed speed of 3.33 mm/s across the Visible Human dataset for different connection types and reply sizes (slice size is 384x384 pixels). The RMS error gives the distance between the displayed slice and a full resolution uncompressed slice.

[2] These times were evaluated by measuring the round-trip time from a modem connection in Switzerland to various servers in the US.

Network data rate [Kbit/s]	Reply size [bytes]	Frame rate [fps]	Displayed slice size [pixels]	Quality [Eq.2]	RMS error [scale 0..255]
128	4000	4	128x128	0.60	13.00
1000	8000	16	192x192	0.75	8.87
1000	32000	4	384x384	1.00	3.30

Table 1. Image quality during navigation across the Visible Human dataset

The frame rates shown in Table 1 can only be reached if the client processor is fast enough to decompress the slices as they arrive.

9. PERFORMANCE OF THE EXTENT-BASED TRANSFER APPROACH

Evaluating the performance and image quality provided by the extent-based server-client transfer is more complex, since the transferred extents are reused for the display of several slices.

The quality of the displayed slice depends on two parameters. As for the slice-based server-client transfer, the reply size plays an important role, since it determines directly how many extents can be transferred before a new slice is drawn. The other intervening factor is the number of useful extents already available in the client cache. This second factor depends on the user's navigation trajectory as well as on the speed with which the user moves the slice frame.

The quality and frame rate have been measured for a slice sequence moving at a speed of 3.33 mm/s along an axis inclined by 15 degrees from the z-axis. Figures 6 and 7 show the frame display intervals and quality that have been measured. For these measurements, the desired frame rate was set to 6 fps, which corresponds to a display period of 166 ms.

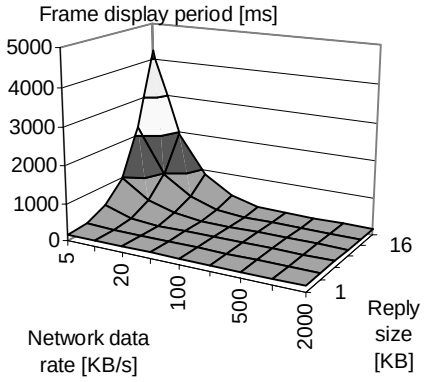

Figure 6. Frame display period as a function of network data rate and reply message size for extent-based server-client transfer

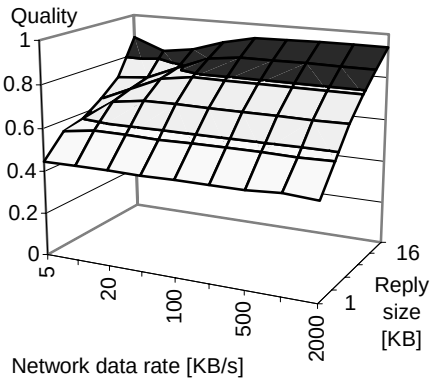

Figure 7. Image quality during navigation as a function of data rate and reply message size for extent-based server-client transfer

Fig. 6 shows that the effective display period increases severely when the reply size becomes too large to be transmitted in the desired frame display period. When this occurs, requests are sent at a higher frequency than the arrival of replies. The limitation of a maximum of two unprocessed requests prevents further requests from being sent, thus lowering the effective frame rate. Eq. 3 dictates the effective frame rate that can be achieved for a given reply size and network data rate. The irregularities in quality (Fig. 7) at low bandwidths (5-20 KB/s) with medium reply sizes are directly linked to this low effective frame rate. The slice frame moved to a new location before the next request was sent, thus causing more high resolution extents available in the client cache to be useless.

Latency and frame rate limitations for the extent-based transfer are similar to what has been described for the slice-based transfer. If the reply messages have the same size, they will contain the same number of JPEG blocks in both transfer systems. The decompression time will be identical. The major difference is that the extent-based transfer requires much more data to be transferred for a single high resolution slice. However, if the slice frame is moved slowly and continuously, the total amount of data that has to be transferred is similar for both methods.

10. MULTIPLE CLIENT PERFORMANCE

The server for the real-time Visible Human navigator must be able to serve several clients at the same time. This section discusses the performance issues of the slice and extent based systems when more than one client is connected at one time.

The first limitation on the number of clients that can be served is the available bandwidth. If the users move slices in small increments through the data set, the bandwidth requirements of both transfer methods are roughly equal. On the other hand, if arbitrary slices are requested at full resolution and must be displayed rapidly, the bandwidth required for the extent-based transfer is much higher. This is due to the fact that a large part of the data populating the transferred extents is never used.

The slice-based system places a very high load on the server for each connected client. When the client is moving the slice frame continuously requesting 6 slices per second, the processor load on the server is 17% for a single Pentium III clocked at 733 MHz. Such a server can accommodate at most 5 clients simultaneously.

Other factors that can influence server performance in multi-client scenarios are the total available bandwidth on the server's network connection, the server's memory cache size, as well as its disk throughput. The disk throughput and cache size are critical since the complete dataset cannot be loaded into memory at one time (out-of-core data access). In order to avoid having to reload data from disk at every transferred slice, the server must be able to hold at least enough extents for each slice the clients are currently viewing. A typical slice at 384x384 pixels requires from 160 to 300 extents, which corresponds to a maximum of 15 MB of memory. Thus 5 clients can be served with 100 MB of server cache memory.

The extent-based system places a much lower load on the server. The server only evaluates which extents have to be transmitted and manages the client's extent cache. Since the source dataset is stored in compressed form, the disk and cache memory requirements are also much lower. The complete full resolution compressed Visible Human dataset fits into 180 MB of memory. The processor usage in order to serve a single client is 10% for the server configuration mentioned before: 10 clients can be served simultaneously.

The number of clients that can be accommodated simultaneously with the current server software is rather limited. Therefore the total bandwidth requirement for the server is not critical for existing high-throughput institutional network connections.

11. CONCLUSIONS AND FUTURE WORK

Real-time navigation across a large 3D volume is highly desirable. It enables to quickly position oblique slices on given anatomic structures. This paper presented two different approaches for client-server partitioning. Both approaches are based on a similar network protocol. The size of replies sent from the server to the client is adjusted as a function of the available bandwidth and the desired frame rate, in order to maximize the usage of the available network bandwidth and to minimize the reply delay. The resolution of the transferred data, either at the slice or extent level, is adapted to provide the highest possible slice quality within this limited reply size.

The extent-based transfer solution has the following advantages when compared to the slice-based solution. The latency is perfectly controlled, since slice extraction is always carried out locally on the client. The server load is low, since the data is fully preprocessed and just has to be sent to the client. Immediate high slice quality is achieved when performing small movements (all data is already in client cache). The disadvantages are very high memory requirements on the client side, not supported by some Virtual Machines. In addition, the client needs high processing power. Furthermore, when carrying out large slice movements and stopping, the quality of the resulting slice only slowly improves due to the time taken to load all required highest resolution extents to the client cache.

Overall, despite some advantages of the extent-based approach, the slice-based service is better adapted to Web clients, since requirements on the client side are lower and the compatibility with various Java Virtual Machines is better.

REFERENCES

1. M. Ackerman, "The Visible Human project", *Proceedings of the IEEE*, Vol. 86, No. 3, March 1998, pp 504-511
2. M. Charrier, D. Santa Cruz, M. Larsson, JPEG 2000, "the Next Millenium Compression Standard for Still Images", *IEEE Multimedia Systems'99*, Volume 1, pp131-132
3. R.D. Hersch et al., "The Visible Human Slice Web Server: A First Assessment", *Proceedings IS&T/SPIE Conference on Internet Imaging*, SPIE vol 3964, pp 253-258
4. Independent JPEG Group, libjpeg, http://www.ijg.org
5. F. H. Netter, *Atlas of Human Anatomy*, Ciba-Geigy Corp., 1989
6. C. North, B. Schneiderman, C. Plaisant, "User Controlled Overviews of an Image Library: The Visible Human Explorer", *The Visible Human Conference Proceedings*, October 1996, http://www.npac.syr.edu/projects/vishuman/ VisibleHuman.html
7. A. Ortega, S. Breslin, K. Lengwehasatit, Variable Complexity Algorithm for JPEG decoding, http:// biron.usc.edu/~lengweha/jpeg/jpg.html
8. W. Richard Stevens, *TCP/IP Illustrated, Volume 1: The Protocols*, Addison-Wesley, 1994
9. A complete bibliography related to the Visible Human Project can be found at http://www.nlm.nih.gov/pubs/ cbm/visible_human.html
10. S. Gerlach, R.D. Hersch, "A Web-based Real-time Visible Human Navigator", *IEEE Internet Computing*, in press, 2002