# A Real-time Navigator for the Visible Human

Adapting data transfer to network throughput enables real-time interactive Web-based navigation of large 3D anatomical data sets.

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

The Visible Human data set, produced by the National Library of Medicine's Visible Human Project,[1] provides researchers with digital cross-sections of the human body. Many institutions use the Visible Human for research and teaching purposes. However, working with the full data set on a workstation is cumbersome and requires advanced programming skills. Storing the data on Web servers and offering online access allows many more students, professionals, and researchers can benefit.

The first Web-based application that let researchers explore the images was the Northeast Parallel Architectures Center's (NPAC) Visible Human Viewer applet,[2] developed in 1995, which allowed extraction of slices perpendicular to the human body's main axes. The Visible Human Slice and Surface Server, which went online in 1999, added access to arbitrarily oriented and positioned slices and surfaces, as well as to slice sequence animations.[3] These applications require the user to define the position and orientation of each slice; each application takes a few seconds to present the results.

To test the feasibility of letting users interact with the data set in real time over the Internet, we recently developed the Visible Human Navigator (visiblehuman.epfl.ch). We used a client-server architecture for the application because the full data set is too large (10 Gbytes) to transfer to the client in a Web-based application. The prototype can display several slices per second on a standard PC connected to the Internet. It also extracts and incorporates anatomic labeling information from the Classified and Labeled Visible Human data set (www.gsm.com/docs/products/segclass.htm) into the slices extracted at any position or orien-

  

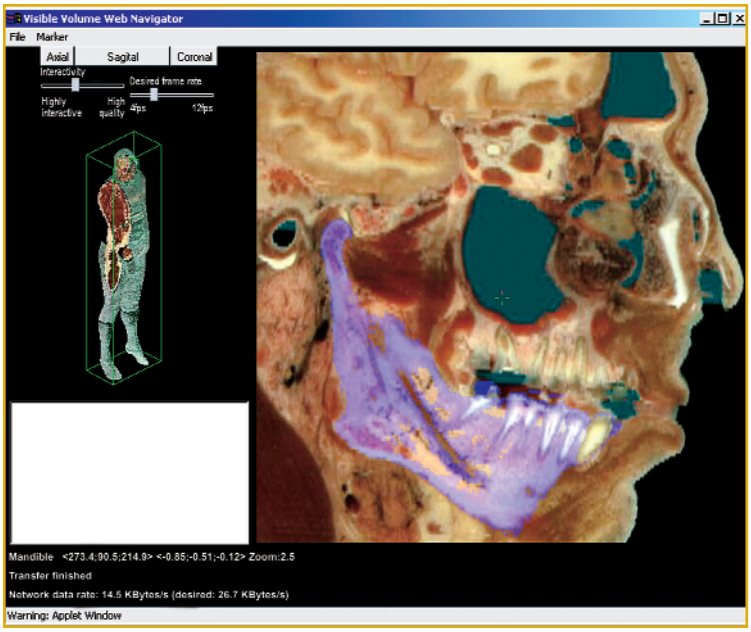


*Figure 1. User interface of the Visible Human navigator. The left side displays the navigation pane, and the right side shows an oblique view across the mandible in the current slice view.*

*Figure 2. Oblique view of the right foot. This slice view shows a highlighted metatarsal bone.*

tation. Because standard HTTP uses a text-based request-response model that creates high overheads, we had to use a custom protocol based on TCP sockets. And because we wanted to let a large audience access the application, we developed the client in Java.

This article focuses mainly on network performance and data-encoding issues. We describe our implementation and the principles of slice and label extraction. Evaluating the proposed solution's performance and the server's behavior when serving multiple clients simultaneously points to several issues for further exploration.

## Navigator Interface

The user interface of the client applet, shown in Figure 1, divides information into two parts: the navigation pane and the current slice view. To facilitate orientation and navigation within the Visible Human data set, the navigation panel provides a 3D view of the human body from which the current slice is cut. Three buttons let users instantly return to planes with standard orientations (axial, coronal, and sagital), and two sliders control the desired frame rate and the resultant speed-to-quality trade off. The slice view displays the current slice and highlights the current structure with labeling data.

The status bar (below the slice view) shows the current position in the data set and the name of the highlighted structure. To verify the current network data rate and frame rate, the status bar also provides the current effective network data rate, the required network data rate (based on the desired frame rate), and other related information. The client applet updates these values in real time.

The navigator provides real-time interactive navigation using a standard mouse. Navigating a 3D data set requires three degrees of freedom for translation and three additional degrees of freedom for rotation around the main axes of the current slice's local coordinate system. The client interface also allows the user to freely zoom in and out of the slice.

To find specific anatomic structures, such as the metatarsal bone, shown in Figure 2, you first move along slices until part of the desired structure becomes visible. Then you can rotate, zoom, and make fine adjustments to the current position until you find the structure you're looking for.

## Visible Human Slice Extraction

The Visible Man data set — a subset of the Visible Human — is a collection of 1,871 color images, each of size 2,048 × 1,212 pixels. The stacked images form a volume of 682 × 404 × 1,871 millimeters. Each voxel (graphic information that defines a point in three-dimensional space) represents 0.33 × 0.33 × 1 mm of the body. The Visible Human server stores each piece of data as small cubic subvolumes called *extents*. As illustrated in Figure 3, each extent consists of a stack of color bytemaps of equal size extracted from full-size axial slices.

For our Web-based navigator, we set the size of the extents at 32 × 32 × 16 voxels to ensure that an approximately constant amount of data will be loaded for a given slice regardless of its orientation. In addition to the standard full-resolution data set, our project stores multiple-resolution versions at reduction factors of 2, 4, 8, 16, and 32, as shown in Figure 4 . This data set keeps the num-

ber of bytes for each extent constant across all resolutions. Lower-resolution extents therefore represent a larger volume of the data set.

Our slice-extraction algorithm can produce an arbitrarily oriented and positioned slice from the data set. As Figure 5 shows, three vectors define each slice. The system extracts the slice and resamples it using an incremental fixed-point algorithm.[4] The algorithm begins rendering at the top-left corner of the slice and evaluates for each pixel the 3D coordinates of the corresponding point. The algorithm then retrieves the nearest voxel (using nearest-neighbor interpolation) or surrounding voxels (using trilinear interpolation) from the data set.

The algorithm traverses the requested slice incrementally from the current coordinate using the right and up vectors of the slice visualization parameters. To provide acceptable speed, the system stores the extents in a memory cache. Caching this data turns out to be an effective technique because the position and orientation of slices varies in small increments during interactive navigation.

The classified and labeled Visible Human data set contains labeling information for each cryosection voxel. The extents store this labeling information in addition to the color data. Each label includes a 16-bit number to indicate which anatomical structure contains each voxel. We use the labels to highlight selected anatomical structures on the cross-section or to indicate the name of the structure over which the user is currently holding the cursor. In Figure 6 (next page), for example, our system uses labeling information to highlight the carotid artery leaving the aortic arch.

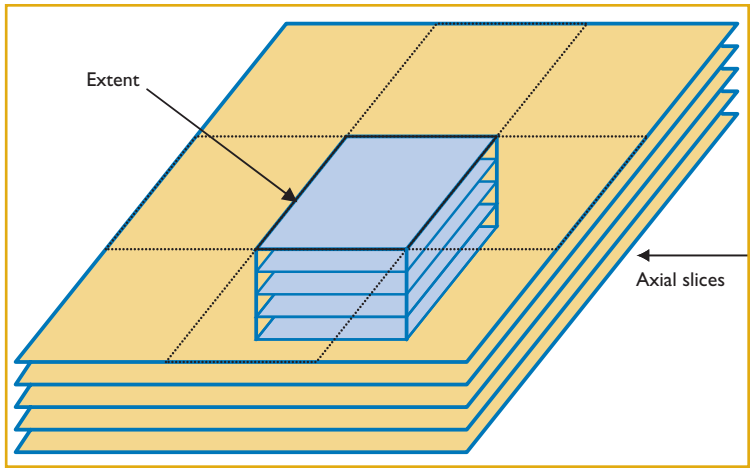


*Figure 3. Extents in the Visible Human project. Each piece of data is stored as an extent (small cubic subvolumes), which consists of a stack of equal-sized bytemaps extracted from full-size axial slices.*

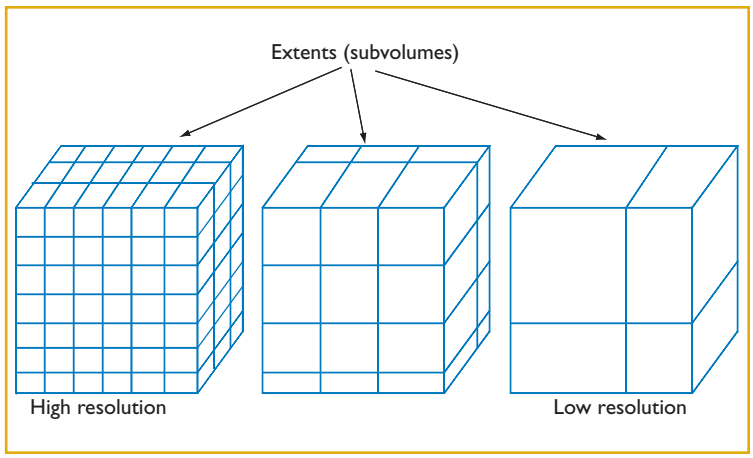


*Figure 4. Multiple-resolution versions of the images. Subdividing volume data sets into extents of decreasing resolution keeps the number of bytes for each extent constant across all resolutions.*

## Client-Server Application Partitioning

We considered several possible mechanisms for slice compression, including the JPEG and emerging JPEG 2000 (based on wavelet compression) standards. Although JPEG 2000 can provide higher quality at an identical compression ratio,[5] it also requires greater processing capacity. Standard JPEG compression is relatively easy to use, encoding libraries are readily available, and JPEG compression can be decoded quickly with pure Java. Also, JPEG's independent macroblock structure provides a simple mechanism for streaming small or partial images from server to client.

At application start-up, the server transmits the JPEG header information, which contains the Huffman tables and other static information and

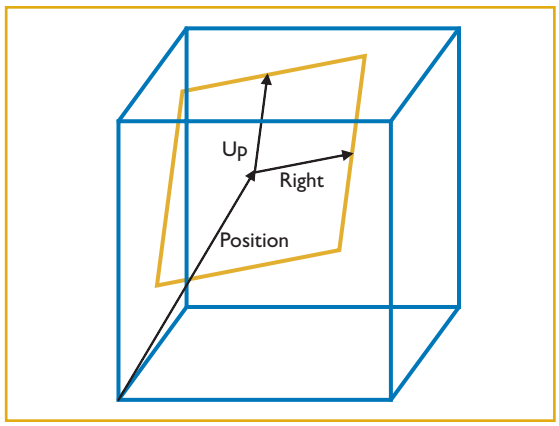


*Figure 5. Slice definition. Three vectors define the position and orientation of each slice in 3D space.*

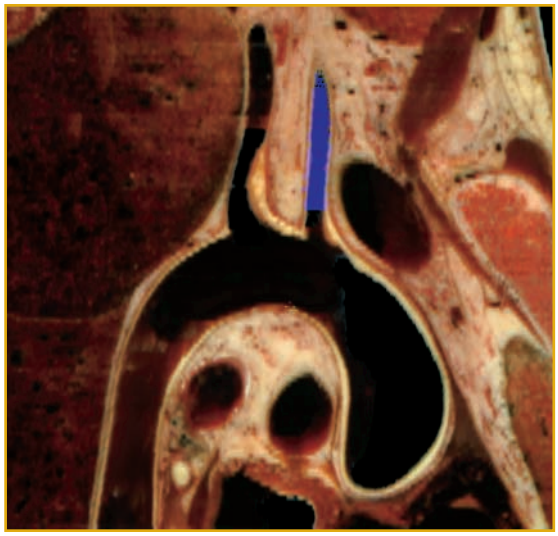

*Figure 6. Sample cross-section from the cryosection data set. The labeling information is used to highlight the carotid artery in blue.*

consists of only 574 bytes of data. The server transmits image data as blocks of 16 × 16 pixels. That corresponds to groups of six JPEG macroblocks with four 8 × 8 pixel macroblocks for the luminance channel at full resolution and two 8 × 8 pixel macroblocks for the two chrominance channels at half resolution.

The server compresses these blocks, on average, from 768 bytes down to 37 bytes. We reached this compression ratio of 20 with the Independent JPEG Group's (IJG) libjpeg compression library (www.ijg.org) by using a quality of 75 on the IJG JPEG quality scale (which ranges from 0 to 100). The instant compression ratio that we obtain depends on the source data because the JPEG compressor strives for constant quality rather than for constant compression ratio. The source data consists of the complete Visible Human data set with all areas outside the body removed.

Decompression happens on the client with a custom Java applet based on code developed at USC,[6] but which we modified to accommodate the JPEG block-based streaming decompression required by our application. Tests have shown that the quality loss caused by compression is not noticeable.

### Client-Server Interaction

Figure 7 shows the basic client-server interaction pipeline. The sequence must let the client display slices that match the user's positioning requests as quickly as possible and with a relatively constant frame rate. To guarantee optimal use of the available network bandwidth and to avoid sending outdated information, the server must transmit exactly the amount of data that it can transfer in the time interval between two displayed frames on the client.

The interval between requests corresponds to the user's desired frame rate. The requests contain the parameters for the currently requested slice, including the three vectors that define position and orientation, as well as an identifier and the viewport parameters. (The client issues a shorter request when no parameters have changed.) The request also contains the maximum allowable message size for the reply. The user can adjust the allowed reply size to provide the best trade off between response time and image quality.

Upon receiving the request, the server processes it and starts sending data back as soon as possible. The response contains only as many bytes as the maximum set by the client's request. Any excess data would arrive after the client had sent its next request, which 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. This mechanism lowers the frame display rate when network transfers cannot keep up with client requests.

The server carries out all transfers using TCP sockets. While UDP might be better suited for multimedia streaming applications — where low latency is more critical than reliability[7] — UDP is unreliable in unsigned Java applets. Also, firewalls do not generally let UDP traffic pass.

### Slice Transfer

Because the server carries out all processing, transferring compressed slices between server and client is extremely simple. When the server receives a request, it extracts the slice from the data set, compresses it, and sends it back to the client. The client then decompresses and displays the slice to the user.

The data set resides on the server in extents of 32 × 32 × 16 voxels at resolutions ranging from full to 1:32. At the lowest resolution, the complete data set fits into 2 × 1 × 4 extents, and individual voxels have a size of 10.56 × 10.56 × 16 mm. The server selects the extracted slice's resolution according to the client's request. For a new request, the server extracts the slice at a resolution set to fit into the maximum reply size. For a continua-

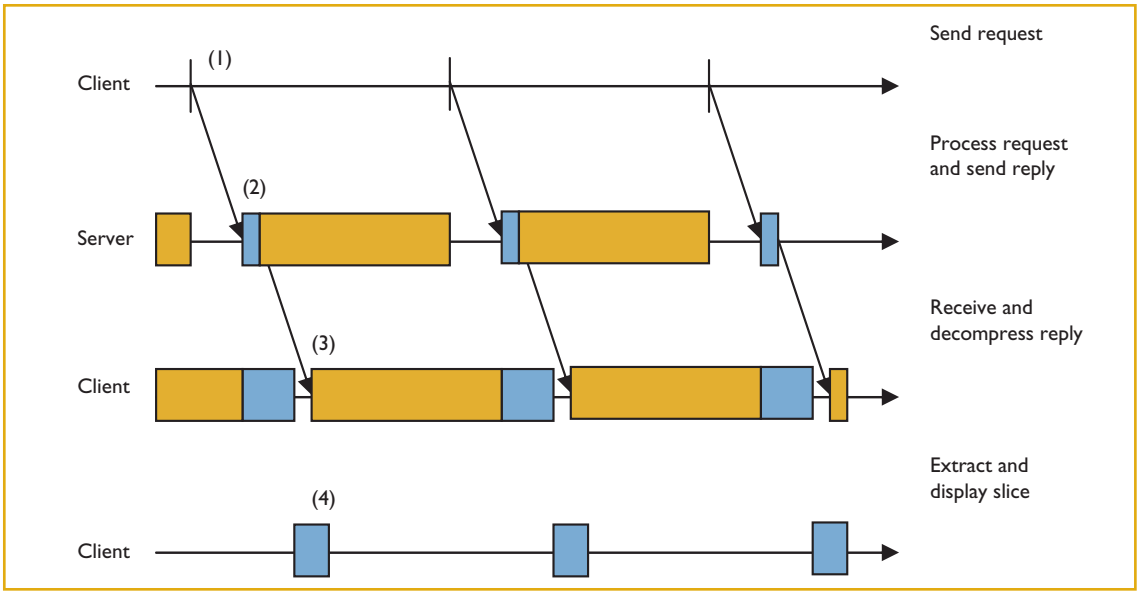


*Figure 7. Client-server interaction pipeline model. Yellow represents the time for sending data and blue represents the processing time. The arrows represent packets transmitted between client and server. Numbers indicate the sequence of operations.*

tion request, the server extracts a full resolution slice and sends it back in parts distributed across successive reply messages.

The client always displays whatever data it receives after decompression, which means that the final high-resolution slice is built progressively. If the server receives a request for a new slice before completing a transmission, it aborts the transfer of the full-resolution slice and instead transmits a new low-resolution slice.

The server sends all slices as collections of JPEG blocks; the headers indicate to which request each block corresponds. When the slice has to fit into a single reply message, the client computes its resolution based on an estimated compression ratio of 14:1, yielding a JPEG block size of 55 bytes. The size must also be a multiple of 16 pixels to ensure optimal use of the JPEG macroblocks. Assuming a square aspect ratio, the formula for calculating the transmitted slice size is

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

For a square aspect ratio image and a reply size of 4,000 bytes, for example, the transmitted slice has a resolution of 128 × 128 pixels. The Java client then scales this lower-resolution image to fill the user's window.

When the user stops moving the slice frame, the server transmits a full-resolution image that takes as many reply slots as required. For a full-resolution slice of 384 × 384 pixels and a reply size of 4,000 bytes, the complete transmission of the full frame typically takes six reply slots, so the client sees the high-resolution image building up over six frames. For the same quality factor, high-resolution slices achieve higher compression than low resolution ones.

This server-client transmission mechanism gives the shortest possible response time when the user moves the slice frame. When the user stops moving the slice, the highest resolution version immediately starts building up on the display. The response time is the sum of the delays induced by the system, which can include network latency, server processing, and client processing.

### Labeled Data

To make the labeled data available to the client, the server extracts the labels simultaneously with the high-resolution slice and compresses the label using the deflate-compression algorithm implemented in zlib (http://www.gzip.org/zlib/). This lossless algorithm provides a compression ratio of 50:1, resulting in a label slice size of approximately 6 Kbytes for a typical slice of 384 × 384 pixels.

The client decompresses the labeled slices using standard Java libraries. The server simply transfers the compressed label slice after the full resolution slice when the user has stopped moving the slice frame. When the server finishes transferring

**Table 1. Image quality during navigation across the Visible Human data set.**

| Network data rate (Kbit/s) | Reply size (bytes) | Frame rate (frames per second) | Extracted slice size (pixels) | RMS error (scale 0.255) |
|---|---|---|---|---|
| 128 | 4,000 | 4 | 128 × 128 | 13.00 |
| 1,000 | 8,000 | 16 | 192 × 192 | 8.87 |
| 1,000 | 32,000 | 4 | 384 × 384 | 3.30 |

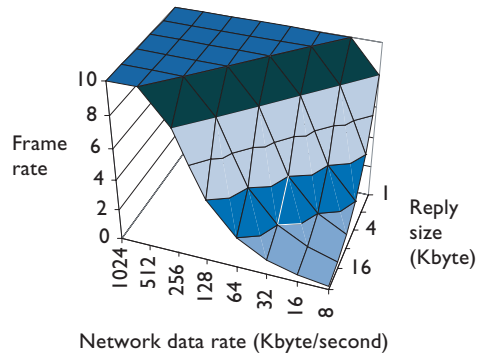


*Figure 8. Frame display rate for various network data rates and reply sizes.*

the labeled data, the highlighting features become available on the client's display.

## Performance

The performance of the slice-based 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 client's requested frame rate *fps*, we can evaluate the reply size *rs* as

$$rs = \frac{ndr}{fps} \tag{2}$$

We compute the resolution of the image produced during the interactive frame movement — which fits into the reply message size — according to Equation 1. Both Equations 1 and 2 show that the system has to make a fundamental compromise between image quality (depending directly on the reply size *rs*) and frame rate. The network data rate *ndr* cannot be controlled, since it depends on the network infrastructure between client and server. Our system leaves the trade-off to the user's discretion by providing a slider that lets the user specify a position between high interactivity (reply size 1 Kbyte) and high quality (reply size 32 Kbytes). Figure 8 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).

Three factors limit the effective frame rate users can achieve: the network bandwidth between client and server, the time required to decompress a frame, or the client's desired frame display rate. The interaction response time on the client is the sum of the following intervals:

- Network latency for sending the request and the reply. The interval ranges from less than 1 millisecond on a local area network to more than 100 ms on a slow connection (typically 70 to 80 ms for long distances).
- Request transfer from the client to the server. The interval depends on the effective network throughput (typically 2 to 3 ms for 48-byte requests with a data rate of 128 Kbps).
- Request processing (slice extraction and compression). The interval depends on the server processing power and disk throughput (typically 10 to 20 ms to serve a request from the server's memory cache on a Pentium III running at 733 MHz).
- Reply transfer from the server to the client. Again, the interval depends on the effective network throughput.

A Pentium III at 733 MHz can decompress 1,000 JPEG blocks per second (roughly 40 Kbytes of incoming data, depending on the compression ratio), and decompression starts as soon as the first data chunk arrives. Because decompression is generally quicker than network data transfer, decompression can occur simultaneously with the transfer.

Experiments have shown that even latencies of 250 ms still provide acceptable responsiveness levels. Because the client displays frames after slice arrival and no more than two requests can be pending at any time, the effective frame rate adapts itself to instantaneous network through-

put. Table 1 shows the image quality when navigating at a fixed speed of 3.33 mm/s across the Visible Human data set for different connection types and reply sizes. The root-means-squared (RMS) error denotes the distance between the actual displayed slice and a full-resolution uncompressed slice.

## Multiple-Client Performance

Because our slice-based system places a high load on the server, it currently supports a rather limited number of users. A client that moves the slice frame continuously, requesting six slices per second, for example, places a 17 percent processor load on the server for a single Pentium III at 733 MHz. Such a server can accommodate at most five clients simultaneously.

Other factors that can influence server performance in multiclient scenarios include the total available bandwidth on the server's network connection, the server's memory size, and its disk throughput. The total bandwidth requirement is not much of a problem for high-throughput institutional network connections, which generally provide between 35 and 155 Mbps. The disk throughput and memory size are critical, however, because the server cannot load the complete data set into memory at one time. To avoid having to reload data from disk with every transferred slice, the server must be able to hold at least enough extents for all the slices the clients are currently viewing. A typical 384 × 384 pixel slice requires from 160 to 300 extents, for example, or a maximum of 15 Mbytes of memory. The server can thus support all five clients with 100 Mbytes of RAM.

## Future Work

Interactive navigation makes it possible for researchers to follow complex, twisted anatomic structures and move through organs in any given direction. With real-time interaction, a user can position slices at odd angles across structures to find the optimal observation direction. The present implementation shows that usable real-time navigation is feasible on currently available networks. The performance remains lower than what we can obtain with a local data set on DVD.[8]

The next step in our research involves creating scalable server architectures using clusters of PCs as server systems. Such systems would be of particular interest for extracting information from terabyte data sets. To situate detailed slice information within larger 3D anatomic structures, we intend to combine the real-time navigation service with the 3D visualization of anatomic organs constructed from the labeled data set.

### References


1. M. Ackerman, "The Visible Human Project," *Proc. IEEE*, vol. 86, no. 3, Mar. 1998, IEEE Press, Piscataway, N.J., pp. 504-511.
2. C. North, B. Schneiderman, and C. Plaisant, "User Controlled Overviews of an Image Library: The Visible Human Explorer," *Proc. Visible Human Conf.*, Oct. 1996; http://www.nlm.nih.gov/research/visible/vhp_conf/vhpconf.htm.
3. R.D. Hersch et al., "The Visible Human Slice Web Server: A First Assessment," *Proc IS&T/SPIE Conf. Internet Imaging*, The Int. Soc. for Optical Eng., Bellingham, Wash., vol. 3964, 2000, pp. 253-258.
4. A. Kaufman and E. Shimony, "3D Scan-Conversion Algorithms for Voxel-Based Graphics," *Proc. 1986 Workshop Interactive 3D Graphics*, ACM Press, New York, 1986, pp. 45-75.
5. M. Charrier, D. Santa Cruz, and M. Larsson, "JPEG 2000: The Next Millennium Compression Standard for Still Images," *IEEE Multimedia Systems 99*, IEEE Computer Soc. Press, Los Alamitos, Calif., vol. 1, 1999, pp. 131-132.
6. A. Ortega, S. Breslin, and K. Lengwehasatit, "Variable Complexity Algorithm for JPEG decoding," http://biron.usc.edu/~lengweha/jpeg/jpg.html.
7. W. R. Stevens, *TCP/IP Illustrated: The Protocols*, Addison-Wesley, Reading, Mass., 1994.
8. S. Gerlach and R.D. Hersch, "The Real-time Interactive Visible Human Navigator," *Proc. Third Visible Human Conf.*, U.S. Nat'l Library of Medicine, 2000; http://www.nlm.nih.gov/research/visible/vhpconf2000/.



**Sebastian Gerlach** is an assistant and PhD candidate at Ecole Polytechnique Fédérale de Lausanne (EPFL). He received a diploma in microengineering from EPFL. His research interests include real-time 3D visualization and parallel application frameworks.

**Roger D. Hersch** is a professor at Ecole Polytechnique Fédérale de Lausanne (EPFL). He received an electrical engineering diploma from ETH Zurich and a PhD in computer science from EPFL. His current research interests include high-performance server applications (imaging servers, Web servers, and PC clusters) and novel imaging techniques (color prediction, color reproduction, artistic imaging, and anticounterfeiting).

Readers can contact the authors at {Sebastian.Gerlach, RD.Hersch}@epfl.ch.