Java Internet Viewer: a WWW Tool for Remote 3D Medical Image Data Visualization and Comparison

Chris A. Cocosco and Alan C. Evans

McConnell Brain Imaging Centre, Montreal Neurological Institute,
McGill University, Montreal, Canada
email: crisco@bic.mni.mcgill.ca

March 2001

Abstract. A powerful, robust, portable, and extensible Java application (JIV) was developed for visualization and side-by-side comparison of multiple 3-dimensional medical image datasets. It works well through the World Wide Web (WWW), it only requires a common Web browser, and can cope with slower networks and less capable workstations. Moreover, JIV provides features and a level of performance usually only found in traditional stand-alone applications. Design choices and considerations of interest to any WWW medical image visualization tool are presented.

1 Introduction

There is a growing need in the medical imaging community for Internet-capable tools that facilitate remote data dissemination and interaction. Large scale, multi-site, research projects and clinical trials require means for a geographically dispersed scientific and clinical community to interact and examine medical imaging data via the Internet [2]. Such 3-dimensional (3D) medical imaging data typically requires special-purpose, non-portable, software to be installed and maintained on the local workstations. The World Wide Web (WWW) technologies have potential for improving this [3, 9, 20]. Below we will present our work on Web tools for 3D medical image data visualization and comparison.

2 Background

The chief advantage of accessing the data and software directly from a remote server (via Internet) is the ease of propagating updates: the user always and transparently gets the most recent versions. There are three main elements in this model: 1. the *server*, where the data (and software) reside; 2. the *client workstation*, operated by the user; 3. the *network* connecting the server and the client (usually in different geographical locations). The main design decision for any remote imaging data visualization tool is how should the computing work be divided between the server and the client workstation; also, how should the

network data transfers be done. An important consideration is the network's performance; in practice the transfer rate can vary by up to three orders of magnitude (1000x). Furthermore, regular Internet connections currently cannot guarantee a certain performance (or QOS: "quality of service"). Another consideration is the client workstation capability: the available RAM and the processor speed can vary in practice by an order of magnitude.

2.1 An initial prototype

A convenient way to visualize 3D medical imaging datasets is by three orthogonal 2D slices (usually: transverse, sagittal and coronal) through the same location in the volume. When several image volumes are to be compared, it is desirable to visualize their slices side-by-side, all at the same position in the volume.

We implemented an initial prototype of a WWW medical image volume viewer using only HTML version3 — HTML "forms" and clickable images for user input, and a CGI application on the server. This design places practically all the computation on the server; the client needs only a basic Web browser. This prototype is available online [5], and a snapshot of its interface is in Fig. 1.

However, this solution has many limitations. First, the HTML forms have only basic user interface capabilities. Secondly, this approach is completely dependent on server response time and on network speed. Overall, this prototype is not interactive enough: the user cannot quickly "roam" through the 3D image volume. Its sole advantage is that it does not require many computational resources on the client workstation; any computer that can run a Web browser will suffice. However, this is becoming less and less important as ordinary PC-class workstations get more and more powerful.

The above limitations can be addressed by running a Java "applet" in the client Web browser (also shown by others[8]), and by employing a fast Web server implementation for providing the data (such as [22, 21], which uses a parallel cluster of servers and disks). Even so, as long as the images are downloaded only when actually required, and the client and the server are not inside the same Local Area Network (LAN), the network speed still limits proper interactive usage [11, 10].

2.2 Other work

Several Java applets for remote medical image and atlas visualization have been reported in the literature [11, 10, 22, 21, 12, 4, 7, 16, 15, 1, 13, 14]. Most of them have limited functionality and use little of the computational potential of a Java applet (notable exceptions are [11, 10], and [16, 15]). In all of these, the slice images are downloaded to the client workstation only when and if required — thus the interactive performance is unsatisfactory when used through remote Internet connections. Given modern PC workstations and Java execution environments, more computation can be moved from the server to the client. In addition, the server-intensive model does not scale well for a large number of concurrent users. In order to minimize the amount of data transfered through the network (and

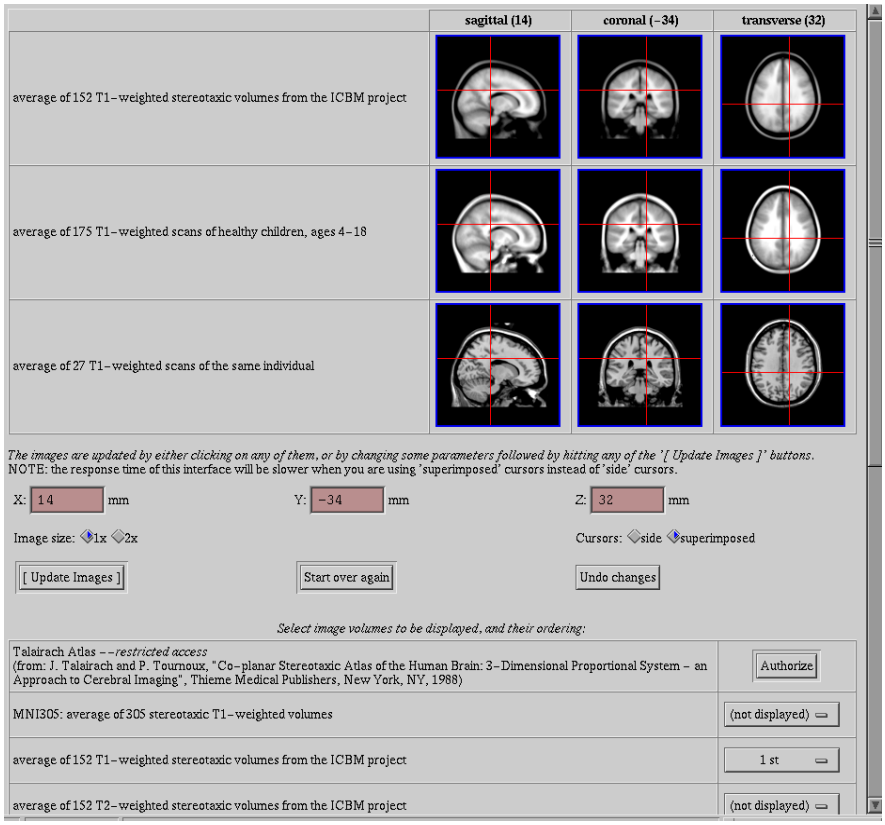

Fig. 1. Screenshot of the initial prototype (HTML-3 forms & CGI). All position cursors (cross-hairs) are synchronized/cross-referenced. The 3D volumes to be displayed, and their relative ordering, are selected from the table at the bottom of the web page

thus minimize the download time on a slow network), many of the above projects use JPEG compression for the 2D slice images. We believe this is not appropriate in a medical imaging context because the JPEG encoding is *lossy*: it does not preserve the original data intact, hence introducing artifacts in the image.

Overall, none of these applets offer capabilities similar to the typical stand-alone medical imaging visualization tools available for traditional (non-Web) local use. For example, none supports side-by-side simultaneous visualization of several image volumes, each with three cross-referenced orthogonal 2D slice images, nor do they offer useful features like colormap adjustment. It was the goal of the work we present here to try to address these shortcomings and produce a Web application that is comparable to traditional stand-alone tools.

3 Design & implementation

This section presents some design and implementation issues that arose during the development of our *Java Image Viewer (JIV)*. Java (version 1.1) was an easy choice for the implementation environment: to date, it is the most portable software development language; Java applets require no installation and no maintenance on the workstation; the vast majority of computers come with Web browsers, and most of these browsers support Java 1.1 . JIV's design places most of the computation on the client; the Web server only has to supply the original data files.

3.1 Data transfer

We use a simple and platform-independent data format for the 3D image data read by JIV: a stream of byte values (8-bit) for the voxel intensities, accompanied by a "header": a plain text file containing ancillary information (described in Table 1). Non-isotropic voxels are resampled at load time (in the applet) to an isotropic grid, by simple voxel replication. This minimizes the data to download, and it trades applet memory for increased interactive performance.

Table 1. Content of the 3D image data file's header (inspired by [19, 18])

(order of dimensions)	the order in which the file scans the data volume — e.g. (z, y, x) means the x coordinate changes fastest
$(size_x, size_y, size_z)$	volume size (number of voxels) along the x, y, z axes
$(step_x, step_y, step_z)$	step values (can be + or -) along each dimension axis
$(start_x, start_y, start_z)$	distance from origin to first voxel along each dim. axis
$(dircos_x, dircos_y, dircos_z)$	unit vectors of the positive direction of each dim. axis

The voxel data file is compressed using the standard (*lossless*) utilities `gnuzip` or `bzip2`. An approximate file size reduction achievable by `gnuzip` for typical volumes is: 20% for individual anatomical MRI, 35-50% for MRI population averages, 90% for segmented data. `bzip2` achieves a further 25-45% improvement in compression ratio over `gnuzip`. However, `bzip2` compressed data takes longer to uncompress and requires more working memory in the client[1]. It is important to note that these compression algorithms are not designed specifically for medical images; special-purpose algorithms should provide better compression.

The most important decision is how and when to download the 3D image data; the following three operation modes are supported by JIV:

1. *All up-front:* all of the data is downloaded and stored in client's memory before the user can view and interact with any of it.
2. *On demand:* download slice image data only when and if the user wants to view that particular slice number.

[1] The actual bzip2 performance impact is yet to be determined (work in progress).

3. *Hybrid (background download):* first download only the slices required for the initial cursor positions; then continue downloading all of the data in a background thread (as in (1)); if the user requests slice images which are not already downloaded, they will be downloaded with priority (as in (2)).

Mode (1) guarantees the best interactive response of the viewer; however, the user has to wait for all the data to download before the JIV interface becomes available – this can take a long time on a slow network link. Mode (2) minimizes the data downloads and amount of memory required by the applet (see section 4), but, like for previous work mentioned in sections 2.1 and 2.2, its interactive response time is completely dependent on the server and on the network.

Mode (3) is the best compromise for most situations. Our implementation does not temporarily freeze while waiting for data to arrive: instead, it displays a discernible pattern for the image areas it does not yet have [2].

3.2 Graphical User Interface

We used the same visualization approach introduced by [17], while extending the capability to an arbitrary number of volumes. For screen-shots of the JIV graphical user interface (GUI), see Figs. 2, 3, and 4.

The 8-bit image intensity data is displayed using a user-controlled colormapping, composed of a color-coding scheme (in between adjustable lower and upper limits), an "under" color, and an "over" color. An intuitive user input scheme (inspired by [17]) is used inside the 2D slice viewports — two mouse buttons combined with a modifier key is all that is needed to: relocate the position cursor, zoom and pan. The zoom factor can be adjusted to arbitrary values. There is also an in-slice distance measurement mode, completely mouse-controlled. Nearest-neighbor interpolation is used for image scaling (zoom), because it is fast to compute and it does not artificially improve the image data — for *discrete* data, smooth interpolation will show data that is not actually there in reality.

4 Results

JIV (available online [6]) can be run either as a Java applet (by means of a Web browser or applet-viewer), or as a stand-alone Java application. It is robust (unlike some of the other tools mentioned in section 2.2)[3]. It was tested and used on a variety of computer platforms: it works well with common Web browsers and with applet-viewers on Linux i386, on SGI IRIX, on MacOS (PowerPC), and on the various MS Windows (Win32) versions. The implementation follows closely the Java object-oriented, image/events passing models; this makes it easy to further expand and reuse the code.

[2] Modes (2) and (3) are work in progress as of this writing.

[3] The likely reason for this is the immaturity of current Java development tools and run-time environments

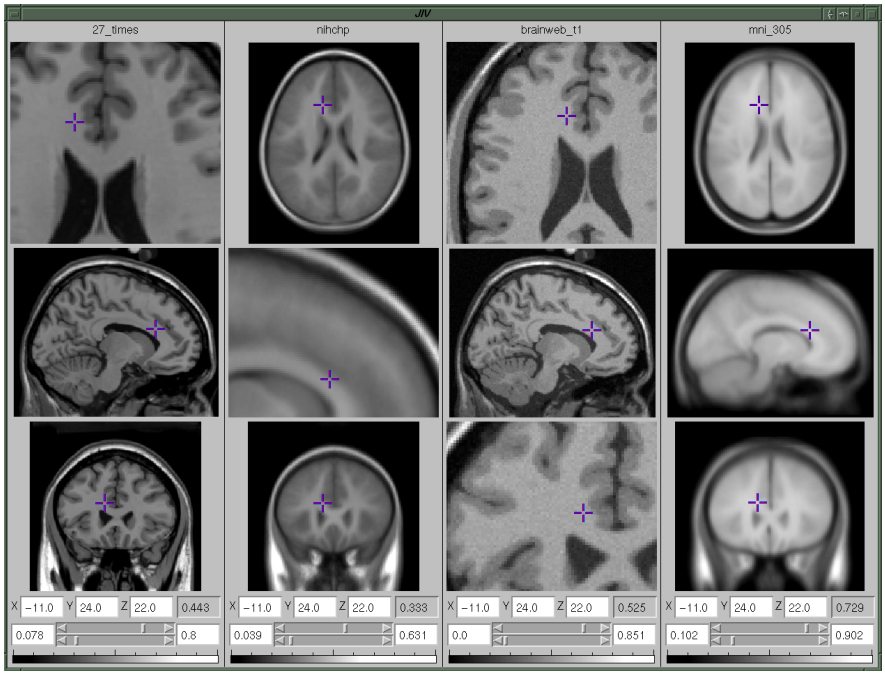

Fig. 2. Screenshot of the JIV (Java) interface. Four datasets are displayed side by side, each with the common three orthogonal 2D slices. The world-coordinates position of all the cross-hair cursors is synchronized. Some of the slices are zoomed-in (differently)

The interactive performance, defined as screen update time following user input, is good when JIV is running on recent PC hardware (as of this writing) and with enough memory (RAM) – see below for a memory requirement estimate. It is worth mentioning that various Java execution environments (JVM, for "Java Virtual Machine") span a wide range in terms of execution speed. We find the JIV interface (as shown in Figs. 2, 3, and 4) to be useful for efficient comparison of multiple 3D image datasets.

If all of the image data is to be downloaded and stored by the applet, JIV's main short-coming is its large resident memory (RAM) requirements: for example, for data volumes of $181 \times 217 \times 181$ voxels, every loaded volume requires 8–9 Mb of memory. Additional memory if required by the JVM, and by internal JIV screen buffers. However, modern PC-s usually have enough RAM for several data volumes. If this is not the case, a possible workaround is to configure JIV to only download image slices on an as-needed basis. If only a low-memory, slow, workstation is available, then the HTML/CGI application presented in section 2.1 might provide a workable solution. But both these approaches will likely hamper the interactive performance because regular Internet connections currently cannot guarantee a certain quality of service.

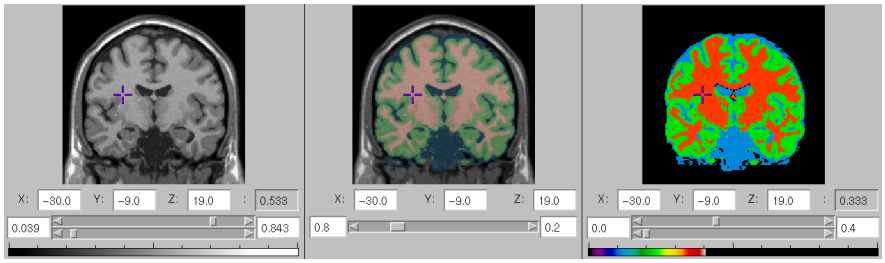

Fig. 3. Sample JIV application: evaluating the tissue classification (segmentation) of a simulated MRI image volume. The merged view (*center*) can be used to visualize the (semi-transparent) color-coded segmentation on top of the MRI image. The image merging is computed in RGB color space by: $color_{vol\_1} \times (1-\beta) + color_{vol\_2} \times \beta$, where β is in 0.0–1.0 (user-controlled by the slider underneath)

Applications: See Figs. 3 and 4 for sample uses of JIV. It is a convenient and platform-independent software for the remote visualization of 3D medical image data. JIV is useful in remote data processing, such as when data goes to a central, well-equipped, site for image processing and storage. Then various remote, geographically dispersed, users can use JIV from common PC-s to visualize, evaluate and otherwise use the data stored in the central image database. As an example, it is used routinely in a currently ongoing, NIH-funded, large-scale project to study normal human brain development (the McConnell Brain Imaging Centre in Montreal hosts the central image database). This software allows simultaneous visualization and comparison of a large number of image volumes, hence it is even used locally within our lab. Other applications are possible in scientific fields that deal with 3D image datasets: biology, geology, physics, and so on.

5 Conclusion

JIV provides a powerful, robust, portable, and extensible 3D image data viewer in the form of an applet which can be run straight from the WWW using a common Web browser. The software can be configured to work reasonably well even with slow networks and less than state-of-the-art workstations.

Acknowledgements Peter Neelin for many design suggestions and advice. Drs. Alex Zijdenbos and Louis Collins for useful suggestions on the interfaces.

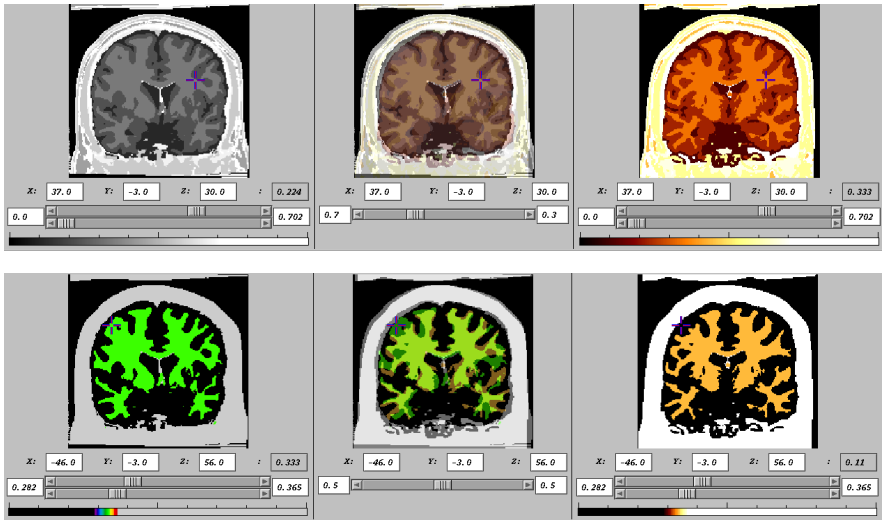

Fig. 4. Sample JIV application: evaluating the mis-registration of two (simulated) MRI-s, after tissue classification. The colormap limits sliders (window/level) can be used to restrict the visible tissue classes (*top* and *bottom* screenshots)

References

1. A. Ade, W. Meixner, and B. Athey. The visible human female world wide web browser. In *The Second Visible Human Project Conference Proceedings*, Oct 1998.
2. C. Alberola, R. Cardenes, M. Martin, M. A. Martin, M. A. Rodriguez-Florido, and J. Ruiz-Alzola. disnei: A collaborative environment for medical images analysis and visualization. In *Medical Image Computing and Computer-Assisted Intervention: Third International Conference*, Lecture Notes in Computer Science, pages 814–823, 2000.
3. E. Bellon, J. Wauters, J. Fernandez-Bayo, M. Feron, K. Verstreken, J. Van Cleynenbreugel, V. den Bosch B, M. Desmaret, G. Marchal, and P. Suetens. Using www and JAVA for image access and interactive viewing in an integrated PACS. *Medical Informatics*, 22(4):291–300, Oct-Dec 1997.
4. Y.-J. Chang, P. Coddington, and K. Hutchens. Viewing the visible human using Java and the web. In Y. Y. et al., editor, *Asia Pacific Web (APWeb) Conference*. International Academic Publishers, Sep 1998.
5. C. A. Cocosco. MNI ICBMView. http://www.bic.mni.mcgill.ca/icbmview/.
6. C. A. Cocosco. MNI JIV. http://www.bic.mni.mcgill.ca/users/crisco/jiv/.
7. P. Coddington, Y.-J. Chang, and K. Hutchens. The NPAC/OLDA visible human viewer. http://www.dhpc.adelaide.edu.au/projects/vishuman/.
8. A. P. Dagher, M. Fitzpatrick, A. E. Flanders, and J. Eng. Enhancing web applications in radiology with Java: estimating MR imaging relaxation times. *Radiographics*, 18(5):1287–93, Sep-Oct 1998.
9. J. Fernandez-Bayo, O. Barbero, C. Rubies, M. Sentis, and L. Donoso. Distributing medical images with internet technologies: a DICOM web server and a DICOM Java viewer. *Radiographics*, 20(2):581–90, Mar-Apr 2000.

10. MIT: AnatomyBrowser. http://www.ai.mit.edu/projects/anatomy_browser/.
11. P. Golland, R. Kikinis, M. Halle, C. Umans, W. E. Grimson, M. E. Shenton, and J. A. Richolt. Anatomybrowser: A novel approach to visualization and integration of medical information. *Computer Aided Surgery*, 4(3):129–43, 1999.
12. K. A. Johnson and J. A. Becker. Harvard university: The whole brain atlas navigator. http://www.med.harvard.edu/AANLIB/cases/java/case.html.
13. J. Lancaster, J. Summerln, L. Rainey, C. Freitas, and P. Fox. The talairach daemon, a database server for talairach atlas labels. In *Neuroimage*, volume 5(4), page S633, 1997.
14. J. L. Lancaster, P. T. Fox, S. Mikiten, and L. Rainey. Talairach daemon. http://ric.uthscsa.edu/projects/talairachdaemon.html.
15. Internet image viewer (iiV). http://james.psych.umn.edu/CNUViewer.
16. J. T. Lee and J. V. Pardo. iiV: A Java-based internet image viewer. In *NeuroImage*, volume 11(5), page S918, 2000.
17. D. MacDonald. MNI register+Display. ftp://ftp.bic.mni.mcgill.ca/pub/register+Display/.
18. P. Neelin. MNI MINC. ftp://ftp.bic.mni.mcgill.ca/pub/minc/.
19. P. Neelin, D. MacDonald, D. L. Collins, and A. C. Evans. The minc file format: from bytes to brains. In *Neuroimage*, volume 7(4), page S786, 1998.
20. A. Oka, Y. Harima, Y. Nakano, Y. Tanaka, A. Watanabe, H. Kihara, and S. Sawada. Interhospital network system using the worldwide web and the common gateway interface. *Journal of Digital Imaging*, 12(2 Suppl 1):205–7, May 1999.
21. EPFL: Visible human slice and surface server. http://visiblehuman.epfl.ch/.
22. S. Vetsch, V. Messerli, O. Figueiredo, B. Gennart, R. D. Hersch, L. Bovisi, R. Welz, and L. Bidaut. A parallel pc-based visible human slice web server. In *The Second Visible Human Project Conference Proceedings*, Oct 1998.