Journal of Digital Imaging

CAVASS: A Computer-Assisted Visualization and Analysis Software System

George Grevera,[1,2] Jayaram Udupa,[2] Dewey Odhner,[2] Ying Zhuge,[2] Andre Souza,[2] Tad Iwanaga,[2] and Shipra Mishra[2]

The Medical Image Processing Group at the University of Pennsylvania has been developing (and distributing with source code) medical image analysis and visualization software systems for a long period of time. Our most recent system, 3DVIEWNIX, was first released in 1993. Since that time, a number of significant advancements have taken place with regard to computer platforms and operating systems, networking capability, the rise of parallel processing standards, and the development of open-source toolkits. The development of CAVASS by our group is the next generation of 3DVIEWNIX. CAVASS will be freely available and open source, and it is integrated with toolkits such as Insight Toolkit and Visualization Toolkit. CAVASS runs on Windows, Unix, Linux, and Mac but shares a single code base. Rather than requiring expensive multiprocessor systems, it seamlessly provides for parallel processing via inexpensive clusters of work stations for more time-consuming algorithms. Most importantly, CAVASS is directed at the visualization, processing, and analysis of 3-dimensional and higher-dimensional medical imagery, so support for digital imaging and communication in medicine data and the efficient implementation of algorithms is given paramount importance.

KEY WORDS: Visualization, 3-dimensional imaging, software systems, image analysis

INTRODUCTION

Software development for 3-dimensional computer-aided visualization and analysis (CAVA) in our group started in the 1970s. In 1980, we brought out the first ever such package for medical 3-dimensional CAVA.[1] This software worked on a Data General minicomputer, which drove a Comtal image display frame buffer. In 1982, we brought out a significantly expanded version of this software package.[2] In spite of its high machine and display device dependency, this package was distributed to over 150 sites with source code worldwide long before the term "open source" was coined. This package was also incorporated into the General Electric (GE) computed tomography (CT)/T 8800 scanner.[3] We subsequently developed a more advanced package[4] for the GE 9800 CT scanner. GE distributed widely these on-the-scanner packages. Earlier, we implemented DISPLAY and DISPLAY82 at the Mayo Clinic, whose investigators used these packages until they started developing the Analyze system[5] around 1984–1985.

Around 1987, we started the development of a Unix-work-station-based software system named 3DVIEWNIX,[6] which was based on standard C programming language and a graphical user interface library developed by us based on X Windows. It also incorporated a multidimensional

[1]*From the Department of Mathematics and Computer Science, Saint Joseph's University, 5600 City Avenue, Philadelphia, PA 19131, USA.*

[2]*From the Medical Image Processing Group (MIPG), Department of Radiology, University of Pennsylvania, 423 Guardian Drive, 4th Floor Blockley Hall, Philadelphia, PA 19104-6021, USA.*

Correspondence to: George Grevera, Department of Mathematics and Computer Science, Saint Joseph's University, 5600 City Avenue, Philadelphia, PA 19131, USA; tel: +1-610-6601535; fax: +1-610-6603082; e-mail: ggrevera@sju.edu

Online publication 6 September 2007
doi: 10.1007/s10278-007-9060-5

generalization[7] of the 2-dimensional digital imaging and communication in medicine (DICOM) image representation standards. This issue of the need to handle a multidimensional vectorial image as a single entity and also to handle nonimage structure information such as surfaces is only now being looked into by the standards committees related to DICOM. These issues were addressed in 3DVIEWNIX in the early stage of its design during 1987–1990. 3DVIEWNIX has incorporated numerous advanced 3-dimensional (and higher-dimensional) CAVA operations including various methods of interpolation, filtering, segmentation, registration, algebraic and morphological operations, visualization methods for surfaces and volumes, interactive structure editing and manipulation, and scene intensity and structure-based quantitative analysis. Its binary version is available freely via the Internet and has been used by hundreds of sites, and the source-code-version has been distributed to more than 180 sites worldwide to date. We continue to maintain, distribute, and develop 3DVIEWNIX by incorporating into it all functions that we find useful after rigorously testing them in one or more of our ongoing applications. About 60 person years of work has gone into 3DVIEWNIX so far. Its design has stood the test of time and of over 15 applications pursued by us since its release.

Since the time 3DVIEWNIX was first released (1993), a number of significant developments have occurred. Most significantly, PC platforms (and the Windows OS) have gained in capability, accompanied by precipitous price reductions. They have supplanted traditional Unix-based work stations as the scientific work stations of choice. Second, network connectivity speed has greatly increased. Third, viable parallel processing standards have been developed and are now freely available for all popular platforms and operating systems. Fourth, platform-independent windowing application programming interfaces (APIs), some of which maintain the native look and feel, have been defined and implemented. Finally, toolkits such as Insight Toolkit (ITK) and Visualization Toolkit (VTK) have been developed and are freely available. Although not complete applications in themselves, these toolkits provide a breadth of techniques and can be employed as building blocks of applications.

Current Software Systems and their Limitations

During the past 10 years, the software development activity for CAVA has increased considerably, making several open-source systems available. In the rest of this section, we shall review the currently available software systems and examine their limitations that were considered in the design and implementation of CAVASS. Our survey here considered most of the well-known software systems, including Analyze,[5] Digital Data Viewer (DDV) (http://www.compgeomco.com), GNU Image Manipulation Program (GIMP) (http://www.gimp.org), Image/J (rsb.info.nih.gov/ij/), Interactive Data Language (IDL) (http://www.rsinc.com), ITK,[8] Java (http://www.javasoft.com), Khoros (http://www.khoral.com), Mathematica (http://www.wolfram.com), Matlab (http://www.mathworks.com), Open Data Explorer (OpenDX) (http://www.opendx.org), Photoshop (http://www.adobe.com), Volview (http://www.kitware.com/products/volview.html), VTK (http://www.vtk.org), Vision-something-Libraries (VXL) (vxl.sourceforge.net), and 3DSlicer (http//www.slicer.org). Analyze, IDL, Khoros, Mathematica, Matlab, Photoshop, and Volview are excellent commercial software packages. However, none of them are freely available or available as open source. Academic prices for these packages for a single user on a Microsoft Windows platform are typically subsidized. Platforms other than Windows are often more expensive, as are commercial licenses. These fees typically include 1 year of updates. After that period of time, additional fees are required to obtain updated software. Only a few of these vendors offer source code (for an additional fee). Additionally, IDL, Khoros, Mathematica, and MATLAB are not complete medical imaging applications but libraries of functions that are callable from their own respective proprietary computer programming languages. Note that, in the case of these proprietary languages, even experienced software developers who are typically already familiar with C++ must learn these programming languages. Matlab provides some support for libraries that are callable from C/C++ and FORTRAN. Analyze is a complete application with the Developer Add-On available for the programming of custom applications. Photoshop is exclusively oriented towards 2-dimensional

image processing and manipulation. Photoshop may be extended by user-written plugins. A DICOM plugin is available for Photoshop to enable it to read single DICOM image files (2-dimensional slices).

The DDV (free, open source, Windows only) software available was not afforded further consideration because it is primarily oriented toward EEG data and not 2-dimensional or higher-dimensional imagery. Another DDV software package, available from http://www.compgeomco.com/, is freely available as binary executables for a variety of platforms, including Windows, Linux, Unix, and Mac. Source code is not available, and even if source code were freely available, DDV uses Qt, which costs $2,330 per person, per development platform. It is primarily a complete application but is limited to the display of slice data (read in from TIFF, raw, and a proprietary format) and the creation and display of isosurfaces from manually segmented slice data. GIMP is freely available as open source on Unix/Linux only. There is no support for DICOM and it is exclusively oriented towards 2-dimensional image processing and manipulation. Image/J is a Java (and therefore, platform-independent) outgrowth of the NIH Image application that is available only for the Mac. It is primarily oriented towards 2-dimensional images but can combine 2-dimensional images into "stacks" of slices. Three-dimensional display is limited to surface plots only. Image/J can import DICOM data. Source code is freely available. ITK, funded by the National Library of Medicine, is also freely available as open source on a wide variety of platforms. It is a programmer's toolkit that is specifically geared towards medical image segmentation and registration. It requires a software developer with extensive C++ knowledge (even more so than VTK). As for VTK, no user interface is provided, but, in contrast with VTK, it does contain a rich variety of algorithms for image processing that are specific to medical imaging. CAVASS incorporates simple mechanisms to interface to ITK.

Our consideration now turns to the Java programming language (including the Java Advanced Imaging, Java2D, and Java3D APIs). Our experience shows that medical CAVA demands the utmost in speed and efficiency due to the voluminous higher-dimensional and/or multimodality data. Simple, prototype Java-based applications that we developed required inordinate amounts of memory and executed far more slowly than their C++ counterparts. We considered using the Java Native Interface (JNI), which allows one to combine Java and C++ code but that requires developers to be experts in two programming languages with no benefit over the solution that we propose below. In fact, a JNI version of 3DVIEWNIX may even be slower because of the conversion between Java and C++ data structures. In the future, Java compilers [such as Google Code Jam (GCJ)], which compile Java to native machine code, may make Java as efficient as C++, but GCJ is still in its infancy as it does not yet support Swing (the Java API for building user interface, which is responsible for drawing buttons, menus, windows, etc.). Similar to Matlab, IDL, VTK, and ITK, it does not provide a suite of medical imaging and visualization applications but is a general-purpose, higher-level programming language upon which these applications may be built. If Java was adopted, we would have had to rewrite all existing 3DVIEWNIX in a different programming language.

OpenDX is an X11-based, open-source application that is freely available for Unix platforms. It is not oriented towards medical imaging applications. It does not contain any segmentation or registration methods. It does, however, perform surface and volume rendering. The DICOM format is not supported. A stereo viewing module, DXStereo, has also been contributed, but, according to the documentation, this module runs only on IBM RS6000 and SGI R4000 platforms. Volview is an application that is primarily oriented towards volume rendering. One may also filter and annotate data. It supports a wide variety of input data formats and is available for a wide variety of platforms (except Mac). Volview is one of the few packages (other than 3DVIEWNIX and CAVASS) that interface with computer-aided design/computer-assisted manufacturing (CAD/CAM) packages. VTK is freely available as open source on a wide variety of platforms. It is not specifically oriented towards medical image processing or medical visualization but it can be used to develop such applications if one is a software developer with solid C++ and Tcl/Tk experience. As a toolkit, no user interface (let alone

one tailored to image processing or medical imaging) is provided. It does not include any medical image processing or visualization applications either. It would require a great deal of effort to use VTK as the basis of CAVASS because none of the existing 3DVIEWNIX applications would be directly transferable into its framework and, therefore, would have to be rewritten. Furthermore, because VTK does not provide a multidimensional (and multimodality) user interface dedicated to medical imaging and visualization as 3DVIEWNIX currently does, this would have to be developed as well. VXL is an open-source C++ library that grew out of TargetJr and Image Understanding Environment. It is primarily oriented towards the analysis of 2-dimensional surveillance satellite imagery with the goal of inferring 3-dimensional geometry. 3DSlicer is an open-source, freely available application for the analysis and display of 3-dimensional medical imagery. It also includes basic automatic registration and semiautomatic segmentation capabilities. It is primarily designed to be used for surgical planning.

Other recently introduced systems include the Medical Imaging Interaction Toolkit,[9] another system by the same name,[10] and Image-Guided Surgery Toolkit.[11] The Medical Imaging Interaction Toolkit[9] and the Medical Imaging Toolkit[10] are similar and independent open-source systems that reuse and extend the capabilities of VTK and ITK. The Image-Guided Surgery Toolkit[11] is an open-source software library that provides the basic components needed to develop image-guided surgery applications, providing functionality for tracking, reading, registering, and calibrating images based on the programs available in ITK and VTK. There is a dichotomy between commercial and noncommercial software systems. Because the availability of source in an open manner is of primary consideration for the theme of this paper, feature-filled and otherwise excellent commercial packages such as Analyze and IDL do not enter into our further discussion.

In the noncommercial group, considering the features as already discussed, the most prominent candidates that remain are 3DVIEWNIX, ITK, VTK, and 3DSlicer. VTK and 3DSlicer are not strong in image processing, manipulation, and analysis. ITK on the other hand is very rich in image processing functions but has no functions for visualization, manipulation, and analysis. Furthermore, it only caters to sophisticated software developers. 3DVIEWNIX has its own limitations. Although it has a variety of functions under image processing, ITK is by far richer in this category. On the other hand, many important image processing functions that are in 3DVIEWNIX and that have been of proven utility, such as shape-based interpolation,[38,39] digital surface detection (in n-dimensions),[49] live-wire segmentation,[12,13] MR image intensity standardization,[14] and various local scale-based processing strategies,[15–19] are not in ITK. The visualization tools in 3DVIEWNIX are as rich and efficient as in any (commercial/noncommercial) system. It has a rich collection of structure manipulation tools, including the ability to cut, move, reflect, and re-edit in 3-dimensional space structures defined in both hard and fuzzy manner. In addition to the common intensity-based and geometry-based analysis tools, it has advanced tools to analyze the morphology, architecture, and kinematics of object systems.

From our review and experience in developing CAVA software and in using (and contributing to) ITK, we conclude that there are four main groups of limitations that exist in open-source software for CAVA currently. (1) Lack of comprehensiveness: There simply is no software (open-source or otherwise) currently that covers all elements (image processing, visualization, manipulation, and analysis) of CAVA comprehensively. There are no systems that provide the basic functional support needed to realize inexpensive stereoscopic visualization and user interaction with such displays in a portable manner. (2) Lack of ease of use: Current systems should cater to such diverse users as CAVA technology developers, CAVA application developers, and users of CAVA in clinical research. (Open-source software activities are not suited for clinical end users of CAVA for day-to-day patient care due mainly to the safety, legal, and financial issues related to patient care.) (3) Lack of speed: Both interactive and noninteractive operations fall short of the speed needed to make many CAVA applications practical, from future considerations and at present, especially when dealing with large data sets. (4) Interfaceability with other systems (such as ITK). Our design goal for CAVASS is to overcome all of these limitations in a system that is useable by a wide variety of different users.

METHODS

CAVASS is an open-source system written entirely in C/C++ and is based on our years of experience with 3DVIEWNIX. It encompasses four groups of operations: image processing (including region of interest, interpolation, filtering, segmentation, registration, morphological operations, and algebraic operations), visualization (including slice, reslice, maximum intensity projection, surface rendering, and volume rendering), manipulation (for surgical planning and simulation), and analysis (various methods for extracting quantitative information).

CAVASS shares a single code base for all Windows, Unix, Linux, and Mac platforms by employing the portable, open-source wxWidgets library. wxWidgets is unique in that it provides a single API across all platforms while maintaining the native look and feel of each. This allows CAVASS to have a single code base for all platforms rather than separate code bases for each platform (which makes development and updates much more difficult). CAVASS also employs the open source local-area multicomputer (LAM) implementation of the message passing interface (MPI) parallel-processing standard. LAM MPI is part of the Linux distribution and is freely available for Unix, Mac, and Windows as well. CAVASS also integrates with popular toolkits such as ITK and VTK.

CAVASS retains much of the architecture of 3DVIEWNIX, which has proven to be very effective, efficient, and easy to maintain and expand (see Fig. 1). The program libraries are compartmentalized into four groups: (1) data interface, (2) graphical interface, (3) process interface, and (4) and CAVA functions. In the interest of brevity, only groups 1 and 4 will be described in detail. CAVA functions are further divided into four groups according to the four elements of CAVA: (a) image processing, (b) visualization, (c) manipulation, and (d) analysis. One may develop their own applications based on these libraries. In addition to these libraries, CAVASS also provides a sophisticated GUI, which, together, form a complete suite of medical imaging applications. The GUI is menu-driven with such main items as Preprocess, Visualize, Manipulate, and Analyze, as well as Port Data, which allows data to be ported into and out of CAVASS. In addition to DICOM support (as illustrated in Fig. 2), CAVASS also supports an *n*-dimensional generalization of the DICOM standard, as well as popular CAD/CAM formats such as stereo lithography (STL) (for the biomechanical analysis) and image format standards such as TIFF, PNM, and VTK.

Data Interface

The data interface library in 3DVIEWNIX[7,20] is designed for a data representation protocol, which is a generalization and an extension of the 2-dimensional DICOM standards. The data interface library contains functions for reading and writing

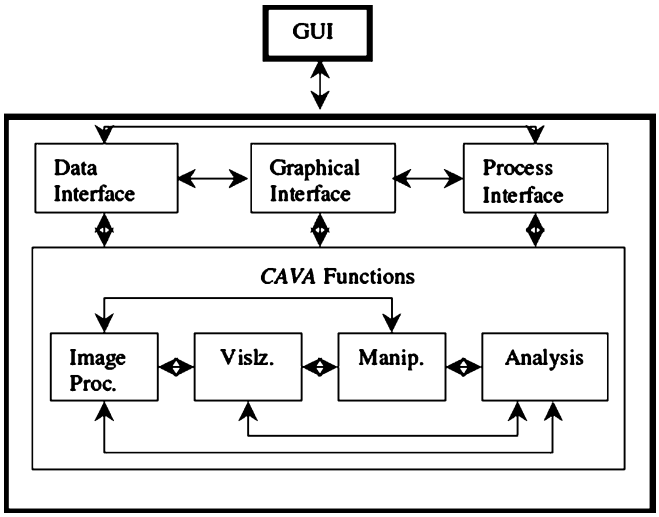

Fig 1. The architecture of CAVASS.

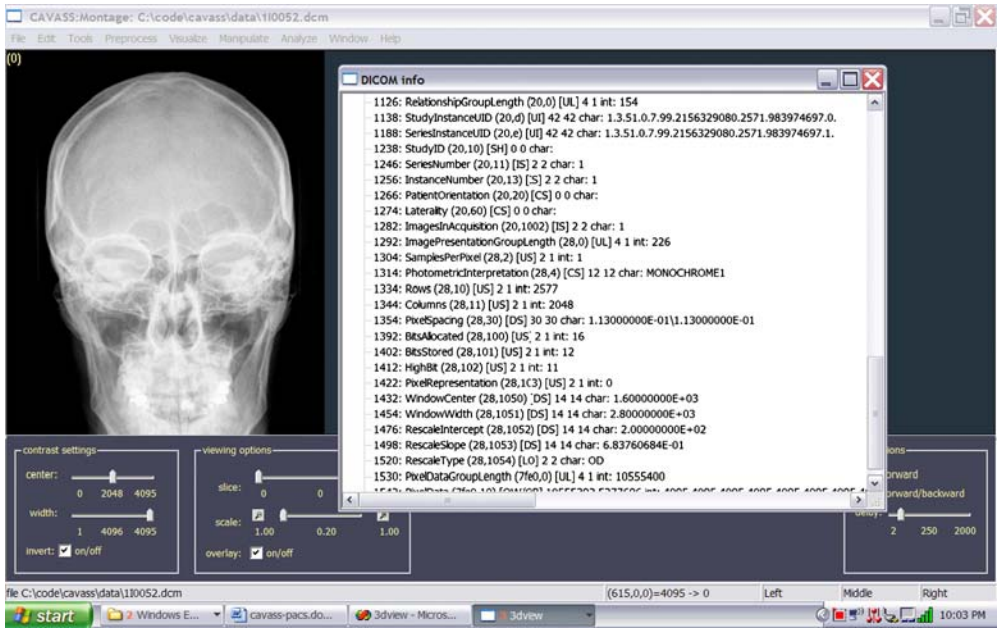

Fig 2. An example of the CAVASS DICOM header browser (sample DICOM image from http://www.agfa.com/en/he/support/doc_library/dicom/adc_dicom_images/index.jsp).

the various types of data handled in 3DVIEWNIX. DICOM[21] is a communication and representation standard for 2-dimensional images. In its current form, it cannot represent 3-dimensional and higher-dimensional image data as a single entity. Also, it has not dealt with issues related to the representation of nonimage data such as surfaces. In view of these lapses, we spent a considerable amount of time in the early phase of the design of 3DVIEWNIX on devising a multidimensional extension and generalization of DICOM. The data interface manual of 3DVIEWNIX[20] describes this generalization and all data types handled in 3DVIEWNIX in great detail. Because the generalization has been found to be very satisfactory, we adopted this in the development of CAVASS. There are three types of data handled by CAVASS: SCENE data, STRUCTURE data, and DISPLAY data. There are multiple subtypes under each category. The SCENE data type represents *n*-dimensional images—scalar, vector-valued, or binary with a regular (rectangular grid) or arbitrary sampling scheme. The STRUCTURE data type represents multidimensional nonimage structure information usually derived from SCENE data. These may be hard or fuzzy boundaries, represented by curves, digital surfaces of various forms, triangulated surfaces, or shells. There is a particular subtype of STRUCTURE data, which is very powerful and has been found to be useful in a variety of applications. This type allows us to represent a structure system, which is a collection of structures, in a manner that is very useful for its visualization and analysis. The structure system may contain any combination of rigid, deformable, static, and dynamic objects. The structure system is essentially a computer representation of an object (organ) system in the body. In the case of dynamic and deformable objects, multiple time samples of the objects are represented in the structure system or appropriate transformations (in the case of rigid, dynamic objects) are stored. When a structure system is rendered, the variable character of the objects is also portrayed – static objects remain static, dynamic objects are rendered with the dynamics – the adequacy of the time component of the portrayal depending on the speed of the rendering algorithm and of the host computer. Finally, the DISPLAY data type constitutes a visual representation of any information in the form of a picture ready to be displayed. This includes such subtypes as screen shots, rendered images, and movie sequences. Because the data types have been found to be very satisfactory and useful, we continue to utilize them in CAVASS.

In addition to supporting the DICOM file formats (both import and export), 3DVIEWNIX (and, hence, CAVASS) supports other common image file formats such as raw unformatted data, GIF, JPEG, TIFF, and PGM. Additionally, 3DVIEWNIX has been used to export structural data to CAD/CAM packages such as Fluid (for computational fluid dynamics) and Abaqus (for finite element modeling) via its ability to create files in the STL format. CAVASS also supports these formats. In addition, it also supports Matlab, Analyze, and Mathematica formats.

Rather than reinventing the wheel with regard to DICOM networking/image query and retrieve capability, CAVASS integrates with commonly available DICOM networking software such as the SimpleDICOM receiver (http://www.radiology.upmc.edu/software.html), which is available from the University of Pittsburg Department of Radiology (for the Windows platform only) or the Conquest DICOM server (http://www.xs4all.nl/%7Eingenium/dicom.html), which is available for both Windows and Linux with source code. Other options include the eFilm work station package (http://www.efilm.ca/), which includes a DICOM server (version 1.5.3 was the last free version), and DCMTK (dicom.offis.de/dcmtk.php.en), which is freely available for Linux, Unix, and Windows with source code.

CAVA Functions

Image Processing

The key image processing operations commonly employed in CAVA are interpolation, filtering, registration, segmentation, and miscellaneous other operations. The image processing operations included in CAVASS may be divided into the following seven groups. We will use $I=(I, f)$ to denote an (*n-dimensional*) *image* where I is the *image domain*, which is a rectangular array of volume elements (voxels), and f is an intensity function that assigns to each voxel v in I an intensity value $f(v)$. $f(v)$ is usually scalar-valued, but it may also be vectorial. In the following description, we assume that $I_i=(I_i, f_i)$ and $I_o=(I_o, f_o)$ denote input and output images, respectively.

1. Volume of interest: These operations are such that $I_o \subseteq I_i$ and f_o is a restriction of f_i to I_o. I_o may be selected interactively or by automatic means. The aim of these operations is to make subsequent operations more efficient and effective.
2. Interpolation: In these operations, the voxels in I_o can be of any size relative to those in I_i and both may be gray or binary images; f_o is some interpolant of f_i.
3. Filtering: The meaning of the term "filtering" is extremely variable as used in the literature. We consider filtering to be any operation such that $I_o=I_i$, I_o and I_i are both either gray or binary, and the intensities in I_o are modified from those in I_i. Operations that come under this category are image enhancement, noise/artifact suppression, and morphological and certain topological operations.
4. Segmentation: In these operations, the output is a binary or a gray image such that $I_o=I_i$ and $f_o(v)$ for $v \in I_o$ indicates the degree of membership of v in the object of interest. Alternatively, the output may also be a hard or fuzzy surface, which represents the boundary of the object.
5. Registration: These operations take two inputs, either images $\mathbf{I}_{i1}$ and $\mathbf{I}_{i2}$ or surfaces S_{i1} and S_{i2} and produce in the respective cases an image $I_o=T(I_{i2})$, which matches with I_{i1} or a surface $S_o=T(S_{i2})$ that matches with S_{i1}, where T is a geometric transformation. T may be a rigid (six-parameter), affine (9–12-parameter), or deformation (hundreds to thousands of parameters) transformation.
6. Image algebra: These operations take generally two input images, either of which may be gray or binary and produce an output gray or binary image. A variety of operations, such as addition, subtraction, multiplication, division, inverting, and certain types of algebraic expressions involving the input images, are permitted.
7. Miscellaneous: These operations allow converting one structure (surface) representation to another, structure-to-image representation, merging different structures into a single structure system, editing DISPLAY data for creating demos, annotation operations, etc.

Visualization

Our work on the surface rendering method of visualization dates back to the early days of CT and MR imaging.[1,22–24] We have devised digital

surface rendering algorithms[25] that run on PCs 16–31 times faster than methods based on rendering triangulated surfaces by using hardware rendering engines[26] and take about an order of magnitude less storage space. The simplicity and efficiency of these algorithms afforded by the simplicity of the geometry of digital surfaces can also be extended to triangulated surfaces and thereby achieve an 8–10-fold speedup in software on PCs over hardware rendering engines if the triangles are embedded in a digital grid, as in the output produced by the Marching Cubes family[27] of algorithms. This also affords compact storage of such surfaces. Due to this computational/storage efficiency, the need for triangle decimation methods currently pursued to reduce the number of triangles in the surface for overcoming computational bottlenecks is obviated.

For volume rendering, we developed a paradigm called shell rendering.[28] The basic idea of this approach is to represent tissue interfaces as shells and do volume rendering by projecting voxels in the shell in a back-to-front or front-to-back order onto the projection plane and performing, in the process, the basic operations of volume rendering, such as reflection, emission, and transmission. In one extreme, the shell may be very thin, just one voxel thick, in which case shell rendering reduces to the digital surface rendering method referred to above. In another extreme, the shell may include the whole foreground of a 3-dimensional image. In practice, the thickness of the shell is in between the two extremes. Recently, a method of volume rendering that has become popular is shear-warp rendering.[29] Like shell rendering, the shear-warp method can be used in both surface and volume mode. The speed of its surface mode is about the same as that of shell rendering in surface mode, but its volume mode is faster (about two times) than shell rendering,[30] although the shear-warp method requires about six to eight times more storage space than shell rendering. We have developed a new method, called shear-warp shell rendering, which combines the advantages of both methods[30] to achieve the speed of shear-warp and storage efficiency close to that of shell rendering. Figures 3 and 4 demonstrate overlaid slice display and t-shell surface rendering in CAVASS on the Windows operating system.

Manipulation

One of the earliest papers to suggest the use of structure information derived from images for surgery planning was that of Udupa.[31] 3DVIEWNIX

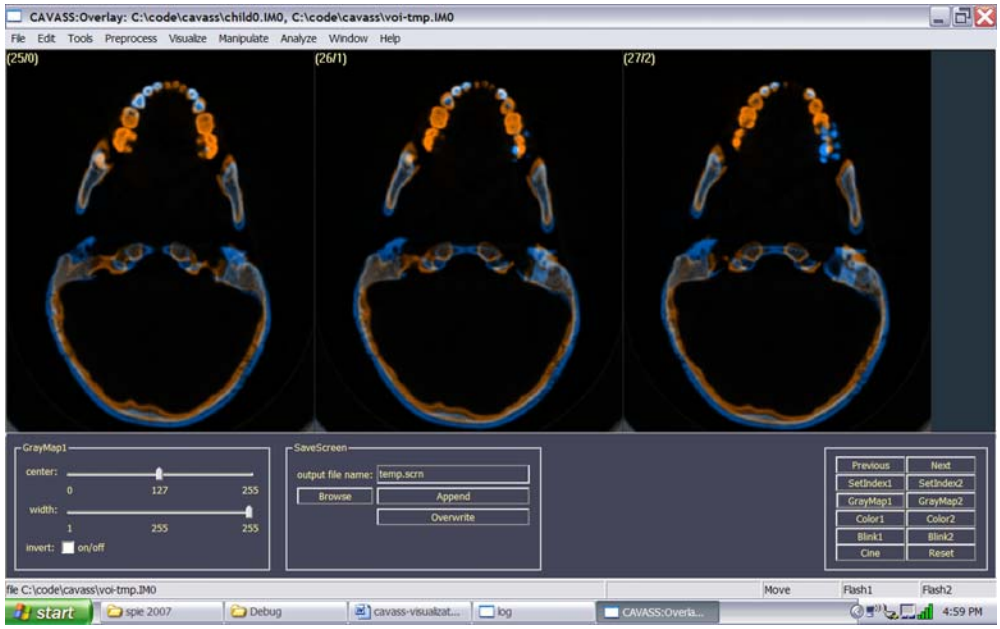

Fig 3. Example of overlaid slice display in CAVASS on the Windows operating system.

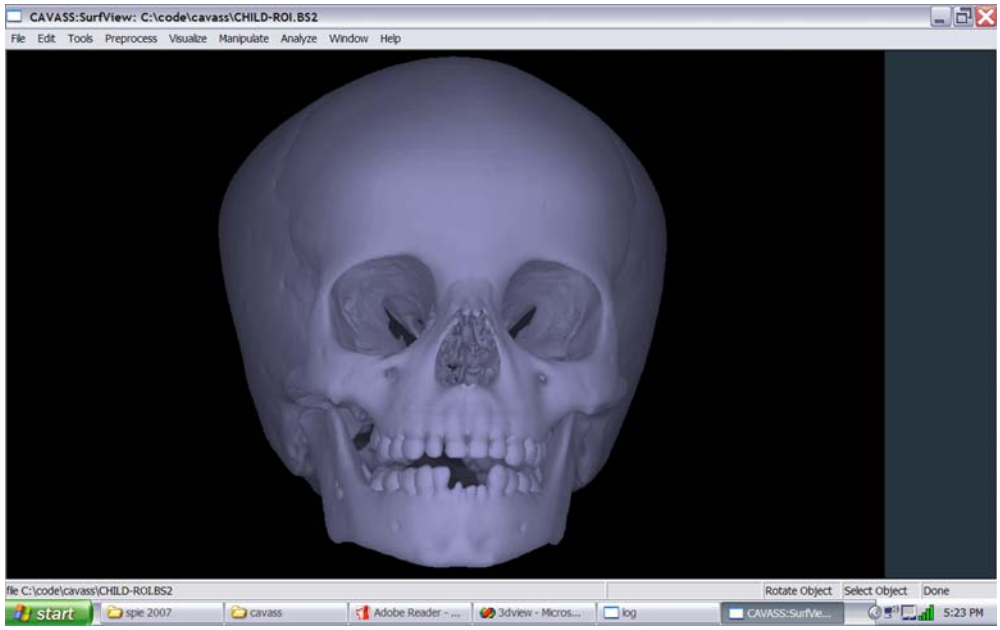

Fig 4. Example of triangulated shell (t-shell) rendering in CAVASS on the Windows operating system.

contains extensive tools for manipulating (cutting, separating, mirror reflecting, moving, or repositioning) structures interactively, all implemented without depending on specialized hardware, and to carry out these manipulative operations on structures defined in a hard, as well as a fuzzy, manner.[25,32]

Analysis

An early paper on the estimation of volume enclosed by and the area of a surface is another by Udupa.[33] Recently we have demonstrated that the volume enclosed by triangulated surfaces can also be estimated in the same simple way by table lookup as done for digital surfaces.[34] We have published methods to compute linear, curvilinear, and angular measurements on the surface or by utilizing landmark points on surfaces observable in their renditions.[3] Methods are also implemented in 3DVIEWNIX for various intensity-based measures.[6] We have developed methods for higher-level analysis of object systems by describing the morphology of individual objects through morphological parameters, the interrelationship among objects through parameters describing the architecture of the object system, and the way this interrelationship changes when the objects move through kinematic parameters.[35–37]

Parallelizing Key Operations in CAVASS

Before we set out to develop parallel implementations of key algorithms, we first devised an experiment to determine the need for such algorithms operating on data sets of common, practical sizes. We know that if we are given an image of high resolution, size, dimensionality, and pixel depth, then many computationally intensive algorithms are choked. We argue that we are already at a stage where it is just impractical to carry out some computationally intensive CAVA operations on top-of-the-line work stations and PCs with sequential algorithms. To justify this argument, we list in Table 1 the processing time for both 3DVIEWNIX and ITK for some key CAVA operations for (scalar) images of two sizes. The platform on which this was performed was a 2-GHz Pentium PC with 1 GB of RAM and 4 GB of swap-space running version 2.4 of Linux. In all cases, the images had 16 bits/pixel. The interpolation operation here created (from a size 512×512×193 input image) an output image whose sizes were, respectively, 512×512×296 and 1,023×1,023×591 for the two cases. (The larger image size simply choked – indicated by "error" – both systems.) Blanks indicate that the operation was not tested. NA stands for "not available." Although this is not at all the main point we wish to make

Table 1. Processing Time (in Seconds) for Some Key Operations in CAVASS and ITK

	512 × 512 × 296		1,023 × 1,023 × 591	
Operation	CAVASS	ITK	CAVASS	ITK
Interpolation (trilinear)	10	319	96	2,530
Filter (3D median)	54	310	517	3,480
Image algebra (difference)	16	52	211	896
Threshold	5	22	51	396
Reslice	14	323	410	2,668
Isosurface creation (digital[58])	10	NA	129	NA
Isosurface creation (triangles) (http://www.gimp.org; rsb.info.nih.gov/ij/)	12	NA	Error	NA
Distance transform (3D)	66	1,766	Error	Error
Registration (correlation)	679	330,336	Error	Error
Registration (mutual information) (http://www.javasoft.com)	3,410	Error	Error	Error
Fuzzy connectedness segmentation (http://www.compgeomco.com)	357	840	Error	Error
Volume rendering (http://www.khoral.com)	1	NA	4	NA
Surface rendering (http://www.wolfram.com)	1	NA	1	NA
Structure manipulation (http://www.wolfram.com)	1	NA	1	NA

from Table 1, note that the 3DVIEWNIX operations are more efficient than those of ITK. The lower efficiency of ITK is mainly due to its generality (often 2-dimensional and 3-dimensional images treated as *n*-dimensional), its class inheritance overhead, code developed at multiple centers, etc. In CAVASS, we keep the same level of emphasis on efficiency as in 3DVIEWNIX.).

From the perspective of the ease of parallelizability, CAVA operations may be divided into three groups, which we will call type 1, type 2, and type 3. Our general approach to parallelized implementation of key CAVA operations is to perform what we call chunking. A chunk is the data contained in a contiguous set of slices. A chunk may represent SCENE or STRUCTURE data. In the former case, it represents a set of contiguous slices of the given image. In the latter case, it represents structure data contained in a contiguous set of slices.

Parallelizing Type 1 CAVA Operations

There are many operations in CAVA, which work, or, which can be made to work, in a more-or-less "slice-by-slice", and hence in a "chunk-by-chunk", manner. In these operations, a slice (or chunk) worth of data needs to be accessed only once to complete the operation (or to complete one iteration of the operation) and produce the final output. Such operations are labeled type 1. Examples of such operations are image gray level slice interpolation methods (linear, spline-based methods),[38] shape-based (binary as well as gray-level) interpolation,[38–42] image-based registration (via mutual information/correlation),[43,44] diffusive filtering,[45–47] inhomogeneity correction,[48] all nonuser-steered slice-by-slice segmentation methods (such as clustering techniques), nonconnected isosurface detection, and structure manipulation.[25,32] The method of processing for type 1 operations may be summarized as follows:

Begin

Step 1 Divide the given image I_i into chunks.
Step 2 Assign the next set of chunks to be processed to the processors; one chunk per processor.
Step 3 In each processor, carry out the type 1 operation on the chunk assigned to it and send its result to the master processor.
Step 4 If there are chunks remaining to be processed, go to step 2.
Step 5 Else assemble results from all processors and output the resulting image I_o or structure.

End

The effect of parallelization here comes from step 3. In the above algorithm, the number of times the loop from step 2 to step 4 is executed depends on the size of I_i, the number of processors available, and the RAM on each processor. In this manner, load balancing is achieved automatically and there is no limit on the size of I_i that can be

handled irrespective of the number of processors available.

Parallelizing Type 2 CAVA Operations

There are other CAVA operations, which work (chunk-by-chunk) in the above sense, but some further operation is needed to combine the outputs produced by the chunks to yield the final output. Such operations are labeled type 2. These are more difficult to parallelize and implement than type 1 operations. Examples of such operations are various surface- and volume-rendering methods, particularly those that use some sort of a front-to-back or back-to-front splatting/projection strategy, such as shell and shear-warp rendering methods.[25,28–30]

Parallelizing Type 3 CAVA Operations

We label those CAVA operations that require each slice/chunk to be accessed more than once to complete the operation as type 3. These can be more difficult than type 1 and type 2 operations to parallelize the implementation. These operations can be characterized by graph traversal methods. The number of times a slice (chunk) is accessed depends on the shape of the objects represented in the image and on the orientation of the slices with respect to the objects. Examples of such operations are connected isosurface detection,[49–51] connected object segmentation in a hard or fuzzy manner,[52–62] and optimal path (graph cut) and fast marching (level set) methods of segmentation.[63,64] In connected isosurface detection,[49,50,65] for example, the average number of accesses of an axial slice in a 3-dimensional image of the human body is typically in the range 1.5–1.8.

A general parallelization scheme for type 3 operations is outlined below. The algorithm uses a queue Q_j (optionally) a list L_j associated with each chunk $\mathrm{I}_{\mathrm{i}}^{j}$ of the input image I_{i}. In the algorithm, π is a predicate whose exact form depends on the particular type 3 operation with which we are dealing.

Begin

Step 1 Divide the given image I_{i} into chunks $\mathrm{I}_{\mathrm{i}}^{j}, j = 1, ..., N$.

Step 2 Initialization. A set of voxels is identified for initializing the underlying type 3 operation. These voxels are placed in the queues associated with the chunks to which they belong.

Step 3 While any of the queues Q_j, j=1,..., N, is not empty, do steps 4–7.

Step 4 Find a free processor P_j and load it with $\mathrm{I}_{\mathrm{i}}^{j}$ and Q_j and L_j.

Step 5 While Q_j is not empty, P_j executes steps 6–7.

Step 6 Remove a voxel v from Q_j, evaluate $\pi(v)$, and place v in L_j, perform appropriate output operations.

Step 7 If $\pi(v)$ is true, place the appropriate neighbors of v in the queues they belong to if they are not already in their designated lists.

Step 8 Combine all outputs from all processors to output I_{o} or the output structure.

End

In the above algorithm, parallelism is achieved via steps 4–7. It is the task of the master processor to keep a watch on the processors whose queues become empty and who therefore may become idle. A processor may be activated because there are chunks whose queues are not empty. The entire process stops at a point when all queues become empty. In steps 6–7, the exact nature of the operations depends on the specific *type 3* operation being implemented. Step 7 also calls for interprocessor communication, which can be handled in several ways to keep it efficient. The method we have implemented is to allow one slice overlap between neighboring chunks and in the associated Q_j and L_j.

Algorithms for Parallelization

Our aim in CAVASS is to parallelize the implementation for the following 10 groups of key CAVA operations: gray-level slice interpolation, shape-based interpolation, image-based registration (via mutual information, correlation), diffusive filtering (scale-based and non-scale-based), inhomogeneity correction (scale-based), structure manipulation (hard and fuzzy[25,32]), surface and volume rendering (via shell and shear-warp techniques), connected isosurface detection (both digital and triangulated), and fuzzy connectedness segmentation. Another area where parallelism can be employed is in stereo rendering for

display. We modified the CAVASS surface/volume rendering implementation to render from not one but two different points of view (one for each eye) for each given position of the projection plane. Typically, the angle between the two nearby viewpoints is about 4°. In CAVASS, we leave this number as a parameter whose value can be modified according to an individual's vision characteristics. The graphics interface library and the GUI was modified to handle these stereo display hardware devices. Library functions were also developed to support all necessary interactions with the stereo display, including pointing to locations on the structures in their surface/volume renditions (we have previously published such algorithms[25,28]), interactively performing curved cuts, repositioning of segments, and making linear, angular, and curvilinear measurements interactively.

Parallel Implementation

Parallel algorithms are implemented in CAVASS using the MPI/OpenMPI standard, which is commonly and freely available for Linux, Unix, Mac, and Windows. MPI or OpenMPI should not be confused with MP or Open Specifications for Multi Processing (OpenMP) (http://www.llnl.gov/computing/tutorials/openMP/#Introduction, OpenMP; http://www.openmp.org/drupal/). OpenMP is a parallel processing standard for "multi-threaded, shared memory parallelism." (http://www.llnl.gov/computing/tutorials/openMP/#Introduction, OpenMP) OpenMP requires special compilers that recognize compiler directives embedded in the source code to control parallelism. Furthermore, "OpenMP is not meant for distributed memory parallel systems." (http://www.llnl.gov/computing/tutorials/openMP/#Introduction, OpenMP) Typically, OpenMP systems are expensive, tightly coupled, shared memory multiprocessor systems (MPS), such as the SGI Origin systems or the new SGI Altix 4700, which "supports up to 512 processors under one instance of Linux and as much as 128TB of globally shared memory" (http://www.sgi.com/products/servers/altix/4000/). Our approach uses inexpensive, commonly available "commodity" work stations/PCs. In particular, our cluster of work stations (COW) consists of six single-process systems (Dell single process 3.6-GHz Pentium systems with 3 GB RAM, hyperthreading enabled, Linux operating system version 2.6.9-1.667smp) interconnected by an inexpensive 1-gigabit (Gb) switch (Dell PowerConnect 2608, an 8-port 1-Gb Ethernet switch).

Another area where parallelism can be employed is in stereo rendering for displays such as those shown in Figure 5 (i-glassesonline.stores.yahoo.net/iglassespc-3d.html). The CAVASS stereo surface/volume rendering implementation renders from not one but two different points of view (one for each eye) for each given position of the projection plane. Typically, the angle between the two nearby viewpoints is about 4°. In CAVASS, this number is a parameter whose value can be modified according to an individual's vision characteristics. The graphics interface library and the GUI handle stereo display hardware devices such as the one in Figure 5. Library functions support all necessary interactions with the stereo display, including pointing to locations on the structures in their surface/volume renditions (we have previously published such algorithms[49,50]), interactively performing curved cuts, repositioning of segments, and making linear, angular, and curvilinear measurements interactively.

An Interface to ITK

We provide access to ITK within the GUI of CAVASS. In this manner, from the user's perspective, all ITK algorithms appear to be incorporated into CAVASS. A working example of the integration of a portable CAVASS prototype GUI with ITK was developed. Platform-independent software loads two SCENE files and displays these files. The user is allowed to change various parameters of Thirion's Demons deformable registration algorithm[66] as implemented in ITK. Once these parameters have been specified (or the default values are found to be acceptable), CAVASS executes the ITK registration algorithm. Results of the registration process are then provided to the user by CAVASS as an output SCENE file.

Portable User Interface

To implement a portable GUI, we considered Qt (http://www.prolltech.com), wxWidgets (formerly called wxWindows) (wxWidgets.org),[67–69] and Fast Light Tool Kit (FLTK) (http://www.fltk.org). Qt was eliminated from further consideration as it is proprietary/closed and requires fees [$2,330 for

Fig 5. Head-mounted display employed by CAVASS for stereo viewing.

one developer on a single platform (ie, Unix, Windows, or Mac OS); $4,660 for one developer on three platforms]. FLTK and wxWidgets both provide a common C++ API that one may use to develop portable GUIs. Both are freely available on a variety of platforms, are open-source, support OpenGL, and drag and drop and cut and paste in a platform-independent manner. Choosing between FLTK and wxWidgets is not simple, but we feel that wxWidgets is superior because it endeavors to maintain the native look and feel of the platform on which it is running. Furthermore, only wxWidgets supports printing in a platform-independent manner. For these reasons, we chose wxWidgets for implementing the GUI in CAVASS.

RESULTS

Many algorithms have already been implemented and tested in CAVASS. For example, interpolation of anisotropic data to isotropic data is a common medical imaging task. We did extensive comparisons on a variety of CAVA operations (type 1 and type 3) in both sequential and parallel modes in various configurations of the COW on a variety of data sets (Table 2). We also compared the speed of these operations as implemented in ITK in sequential mode and parallel mode (when available). The parallel implementations in ITK are in the MPS environment. Because MPS are very expensive, we acquired a dual-processor MPS for testing purposes only. Therefore, to make the comparison fair, in such instances, the COW was configured with only two single-processor CPU work stations. Our results are summarized in Table 3.

We note that considerable speed differences exist between CAVASS and ITK for the image processing operations. We attribute the higher speed of CAVASS to several factors. First, many of the implementations in ITK are very general on

Table 2. Description of Data Sets of Varying Sizes used in the Comparisons

Data Set Name	Voxel Size (mm)	Image Size	Data Size (MB)
Regular	0.98 × 0.98 × 3.00	256 × 256 × 46	6
Large	0.68 × 0.68 × 1.50	512 × 512 × 459	241
Super	0.24 × 0.24 × 0.50	1,023 × 1,023 × 417	873

various counts, such as image dimensionality, number of bits per pixel, and scalar versus vectorial. In CAVASS, we went for generality to the extent it is needed and in most common use. Second, implementations in CAVASS (many of which come from 3DVIEWNIX) are more tightly monitored, being an effort within a single group. Third, in ITK, because of its openness, and contributions of implementations coming from around the world, testing and optimization become really challenging.

The times reported in Table 3 represent the total operational time for each listed CAVA operation. Some of these operations include a mix of type 1 and type 3 algorithms in addition to other housekeeping operations such as input/output. We may note that, for pure type 1 operations (interpolation, scale computation), we achieve a speedup factor of 0.65–1.8 for parallelization. Here, the speedup factor is defined as $t_s/(t_p n_p)$, where t_s and t_p are the time taken for the sequential and parallel implementation of the same operation, and n_p is the number of processors used in parallel implementation. This factor is, quite understandably, lower, 0.56, for pure type 3 operations (eg, fuzzy connectedness). Among the operations listed in Table 3, registration is the most time consuming. In these operations, normalized mutual information was used to register the two images. The second image was created from the first by applying a known (rigid or affine) transformation. The speedup factor achieved in this instance is excellent. With a COW of about 10 PCs, therefore, we can expect to complete a 12-parameter affine registration of extremely large data sets in about 30 min. Parallelized deformable registration is currently being implemented in CAVASS.

The two major volume visualization methods of surface rendering and volume rendering have also been implemented and tested in CAVASS. We compared the implementations of sequential t-shell surface rendering (an example appears in Fig. 4) implemented entirely in software in CAVASS with hardware-assisted surface rendering using the Marching Cubes method as implemented in VTK (using the vtkImageMarchingCubes class). We also compared sequential and parallel volume rendering implemented entirely in software in CAVASS with two methods of volume rendering (ray casting and 2-dimensional texture mapping) implemented in VTK (using the vtkVolumeRayCastMapper and vtkOpenGLVolumeTextureMapper2D classes, respectively). The timing results in seconds per frame were obtained by applying the various visualization techniques to three data sets of varying sizes (regular, large, and super) as shown in Table 2. Results for sequential surface rendering and parallel and sequential volume rendering appear in Tables 4 and 5, respectively.

Table 3. Time in Seconds for the Various Operations for the Regular (6 MB), Large (241 MB), and Super (873 MB) Image Data Sets

		Regular (DS1)		Large (DS2)		Super (DS3)	
Operation	System	Seq	Parallel	Seq	Parallel	Seq	Parallel
	ITK	2.9	1.7	87.7	62.8 [2]	Failed	Failed
Interpolation	CAVASS	0.6	1 [2]	54.9	14.9 [2]	139.1	49.2
	ITK	57		2,026.6		Failed	
Anisotropic diffusive filtering	CAVASS	52.7		1,664.2			
	ITK	1.5		65.2		Failed	
Gaussian filtering	CAVASS	0.4		18.3		83	
	ITK	10.5		473.7		Failed	
Distance transform	CAVASS	18.7		916.5		3,882.4	
	ITK	0.3		11.4		340.6	
Thresholding	CAVASS	0.1		2.7		20.2	
	ITK	108.4		Failed		Failed	
Fuzzy connected segmentation	CAVASS	49.5	17.8	843.7	298.6 [5]	Failed	1,312.6 [5]
	ITK	57.2		Failed		Failed	
Registration (rigid)	CAVASS	56.1	8.6 [5]	1,860.6	301.6 [5]	3,863.4	1,089.1 [5]
	ITK	208.3		Failed		Failed	
Registration (affine—12 parameters)	CAVASS	155.3	25.1	3,602.4	1,018.6 [5]	13,111	3,662.2 [5]

The number of processors used is indicated in square brackets in case of parallel operations. No entries indicate that the particular operation was either not tested or not available.

Table 4. Surface Rendering Timing Comparison for CAVASS (Sequential Implementation with and without Antialiasing) and Surface Rendering as Implemented in VTK

Data Set Name	CAVASS seq/no aa	CAVASS seq/aa	VTK
Regular	0.03	0.06	0.29
Large	0.11	0.19	0.41
Super	0.16	0.26	1.38

Table 4 shows that sequential CAVASS shell rendering, entirely in software and without antialiasing, was more than 8.5 times faster than hardware-based rendering as implemented in VTK for the largest data set (super) in our test. With antialiasing, CAVASS shell rendering was more than five times faster. For volume rendering, Table 5 shows that the CAVASS implementation, entirely in software, was faster than both ray casting and 2-dimensional texture mapping as implemented in VTK for both the regular and large data sets. For the super data set, sequential CAVASS volume rendering was slower than volume rendering in VTK, but the parallel implementation of volume rendering in CAVASS was almost twice as fast as ray casting in VTK. Although VTK ray casting was able to render the largest data set, 2-dimensional texture mapping as implemented in VTK was unable to render the largest data set after more than 240 s. This is likely due to the limited amount of memory on the graphics card. When we compare VTK ray casting to VTK 2-dimensional texture mapping, we note the trend that VTK ray casting is consistently faster than VTK 2-dimensional texture mapping. Because CAVASS parallel volume rendering is consistently faster than both VTK ray casting and VTK 2-dimensional texture, we conclude that, even with additional video memory, CAVASS parallel volume rendering would be faster than VTK 2-dimensional texture rendering of the largest data set.

All sequential tests were performed on a Dell single-processor, 3.6-GHz Pentium system with 3 GB RAM and hyperthreading enabled under the Linux operating system version 2.6.9-1.667smp. The multithreaded tests were performed on a Dell dual processor, 3.4-GHz Xeon system with 4 GB of RAM and hyperthreading enabled under the Linux operating system version 2.6.9-1.667smp. The parallel visualization tests were performed on a cluster of six single-processor systems (Dell single processor, 3.6-GHz Pentium systems with 3 GB RAM and hyperthreading enabled under the Linux operating system version 2.6.9-1.667smp) interconnected by an inexpensive 1-Gb switch (Dell PowerConnect 2608, an eight-port, 1-Gb Ethernet switch). All systems had Nvidia Quadro NVS280 PCIe 64-MB video cards.

CONCLUSIONS

We described CAVASS, a new open-source, open-platform software system and the next incarnation of the previously established and widely distributed 3DVIEWNIX software system. We demonstrated the extremely efficient implementation of algorithms in sequential and parallel modes on COWs in CAVASS. CAVASS is the only freely available, open-source image processing, analysis, and visualization software system for multidimensional medical imagery that incorporates other open-source toolkits and provides for the efficient and parallel implementations of important algorithms. With regard to such common image processing tasks as interpolation, anistropic diffusive filtering, Gaussian filtering, thresholding, fuzzy connected segmentation, rigid registration, and affine registration, CAVASS sequential implementations were shown to be as much as 4.8, 1.2, 3.8, 4.2, 2.2, 1.0, and 1.3 times faster, respectively, than ITK sequential implementations. The only exception was the distance transform where the ITK implementation was as much as 1.9 times faster than the CAVASS implementation. With regard to parallel interpolation, CAVASS was as much as 4.2 times faster than ITK parallel interpolation. CAVASS also provides parallel implementations for fuzzy

Table 5. Volume Rendering Timing Comparison for Sequential and Parallel Implementations of CAVASS Volume Rendering, VTK Ray Casting, and VTK 2-Dimensional Texture Mapped Volume Rendering

	CAVASS		VTK	
Data Set Name	Sequential	Parallel	Ray Casting	2D Texture
Regular	0.56	0.06	1.09	1.20
Large	3.53	1.36	5.03	18.32
Super	9.77	3.66	6.94	>240.00

connected segmentation and rigid and affine registration, whereas ITK does not. These parallel implementations in CAVASS were shown to be 6.1, 6.7, and 8.3 times faster than the corresponding sequential ITK implementations. CAVASS was also able to deal with much larger data sets that made ITK fail.

With regard to visualization, surface rendering in CAVASS entirely in software was demonstrated to be more than 8.5 times faster than hardware-assisted surface rendering. For volume rendering, we demonstrated that sequential volume rendering in CAVASS entirely in software is faster for the regular and medium data sets in our test, and for the largest data set (super), parallel volume rendering in CAVASS was almost twice as fast as the fastest hardware-based method.

Finally, CAVASS may be used as a toolkit library or as a complete set of applications with an easy-to-use GUI that interfaces with other popular data formats and toolkits.

CAVASS is available from http://www.mipg.upenn.edu/~cavass.

ACKNOWLEDGEMENT

The authors gratefully acknowledge support for this work from DHHS grant EB004395. Portions of this work were previously published in Udupa et al.[70] and Grevera et al.[71,72] and appear here with permission.

REFERENCES

1. Udupa JK: DISPLAY—A System of Programs for Two- and Three-dimensional Display of Medical Objects from CT Data. Technical Report MIPG41, Medical Image Processing Group, Department of Computer Science, SUNY/Buffalo, Buffalo, 1980

2. Udupa JK: DISPLAY82—A System of Programs for the Display of 3D Information in CT Data. Technical Report MIPG67, Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Philadelphia, 1983

3. Chen LS, Herman GT, Meyer CR, Reynolds RA, Udupa JK: 3D83—An easy-to-use software package for three-dimensional display from computed tomograms. In: Proceedings of IEEE Computer Society International Symposium on Medical Images and Icons, Arlington, 1984, pp 309–316

4. Udupa JK, Herman GT, Margasahayam PS, Chen LS, Meyer CR: 3D9S: A turnkey system for the display and analysis of 3D medical objects. SPIE Proc 671:154–168, 1986

5. Robb RA, Hanson DP, Karwoski RA, Larson AG, Workman EL, Stacy MC: ANALYZE: A comprehensive, operator-interactive software package for multidimensional medical image display and analysis. Comput Med Imaging Graph 13:433–454, 1989

6. Udupa JK, Odhner D, Samarasekera S, Goncalves R, lyer K, Venugopal K, Furuie S: 3DVIEWNIX: an open, transportable, multidimensional, multimodality, multiparametric imaging software system. SPIE Proc 2164:58–73, 1994

7. Udupa JK, Hung HM, Odhner D, Goncalves R: Multidimensional data format specification: A generalization of the American College of Radiology National Electric Manufacturers Association Standards. J Digit Imaging 5(l):26–45, 1992

8. Ibanez L, Schroeder W, Ng L, Cates J: The ITK Software Guide. New York: Kitware, 2003

9. Wolf I, Vetter M, Wegner I, Böttger T, Nolden M, Schöbinger M, Hastenteufel M, Kunert T, Meinzer H-P: The Medical Imaging Interaction Toolkit (MITK). Med Image Anal 9(6):594–604, 2005

10. Zhao M, Tian J, Zhu X, Xue J, Cheng Z, Zhao H: Design and implementation of a C++ toolkit for integrated medical image processing and analyzing. Proc SPIE 5367:39–47, 2004

11. Gary K, Ibanez L, Aylward S, Gobbi D, Blake MB, Cleary K: IGSTK: An open source software toolkit for image-guided surgery. Computer 39(4):46–53, 2006

12. Falcao A, Udupa JK, Samarasekera S, Sharma S, Hirsch BE, Lotufo R: User-steered image segmentation paradigms: Live wire and live lane. Graph Models Image Process 60 (4):233–260, 1998

13. Falcao A, Udupa JK, Miyazawa FK: An ultra-fast user-steered image segmentation paradigm: Live-wire-on-the-fly. IEEE Trans Med Imag 19(1):55–62, 2000

14. Nyul LG, Udupa JK, Zhang X: New variants of a method of MRI scale standardization. IEEE Trans Med Imag 19 (2):143–150, 2000

15. Saha PK, Udupa JK: Scale-based image filtering preserving boundary sharpness and fine structure. IEEE Trans Med Imag 20(11):1140–1155, 2001

16. Nyul LG, Udupa JK, Saha PK: Incorporating a measure of local scale in voxel-based 3-D image registration. IEEE Trans Med Imag 22(2):228–237, 2003

17. Souza A, Udupa JK, Saha PK: Volume rendering in the presence of partial volume effects. IEEE Trans Med Imag 24 (2):223–235, 2005

18. Madabhushi A, Udupa JK, Souza A: Generalized scale: Theory, algorithms, and application to image inhomogeneity correction. Comput Vis Image Underst 101:100–121, 2006

19. Madabhushi A, Udupa JK: New methods of MR image intensity standardization via generalized scale. Med Phys 33 (9):3426–3434, 2006

20. Udupa JK, Hung HM, Odhner D, Goncalves R: The 3DVIEWNIX Software System, Data Format Specification: A Multidimensional Extension to the ACR-NEMA Standards. Version 1.0. Technical Report MIPG177, Medical Image Processing Group, Department of Radiology, University of Pennsylvania, Philadelphia, 1991

21. National Electrical Manufactures Association (NEMA): Digital Imaging and Communication in Medicine (DICOM) Part 1: Introduction and Overview. Washington, DC: NEMA, 1993

22. Altschuler MD, Censor Y, Eggermont PBB, Herman GT, Kuo YH, Lewitt RM, McKay MR, Tuy H, Udupa JK, Yau MM: Demonstration of a software package for the reconstruction of the dynamically changing structure of the human heart from cone-beam x-ray projections. J Med Syst 4 (2):289–304, 1980

23. Udupa JK: Display of 3-D information in discrete 3-D scenes produced by computerized tomography. Proc IEEE 71:420–431, 1983

24. Herman GT, Udupa JK: Display of 3-D information in 3-D digital images: Computational foundations and medical applications. IEEE Comput Graph Appl 3:39–46, 1983

25. Udupa JK, Odhner D: Fast visualization, manipulation, and analysis of binary volumetric objects. IEEE Comput Graph Appl 11(6):53–62, 1991

26. Grevera GJ, Udupa JK, Odhner D: An order of magnitude faster surface rendering in software on a PC than using dedicated rendering hardware. IEEE Trans Vis Comput Graph 6(4):335–345, 2000

27. Lorensen WE, Cline HE: Marching cubes: A high resolution 3D surface construction algorithm. Comput Graph 21(4):163–169, 1987

28. Udupa JK, Odhner D: Shell rendering. IEEE Comput Graph Appl 13(6):58–67, 1993

29. Lacroute P, Levoy M: Fast volume rendering using a shear-warp factorization of the viewing transformation. In: Proceedings of SIGGRAPH. New York: ACM Press, 1994, pp 451–458

30. Falcao AX, Rocha LM, Udupa JK: Comparative analysis of shell rendering and shear-warp rendering. SPIE Proc 4681:472–482, 2002

31. Udupa JK: Interactive segmentation and boundary surface formation for 3-D digital images. Comput Graph Image Process 18:213–235, 1982

32. Odhner D, Udupa JK: Shell manipulation: Interactive alteration of multiple-material fuzzy structures. SPIE Proc 2431:35–42, 1995

33. Udupa JK: Determination of 3-D shape parameters from boundary information. Comput Graph Image Process 17:52–59, 1981

34. Nystrom I, Udupa JK, Grevera GJ, Hirsch BE: Area of and volume enclosed by digital and triangulated surfaces. SPIE Proc 4681:669–680, 2002

35. Stindel E, Udupa J, Hirsch B, Odhner D, Couture C: 3D MR image analysis of the morphology of the rear foot: Application to classification of bones. Comput Med Imaging Graph 23:75–83, 1999

36. Stindel E, Udupa J, Hirsch B, Odhner D: A characterization of the geometric architecture of the peritalar joint complex via MRI: An aid to classification of feet. IEEE Trans Med Imag 18:753–763, 1999

37. Udupa J, Hirsch B, Samarasekera S, Hillstrom H, Bauer G, Kneeland B: Analysis of in vivo 3D internal kinematics of the joints of the foot. IEEE Trans Biomed Eng 45:1387–1396, 1998

38. Grevera GJ, Udupa JK: Shape-based interpolation of multidimensional grey-level images. IEEE Trans Med Imag 15 (6):881–892, 1996

39. Raya SP, Udupa JK: Shape-based interpolation of multidimensional objects. IEEE Trans Med Imag 9(l):32–42, 1990

40. Higgins WE, Morice C, Ritman EL: Shape-based interpolation of thin structures in three-dimensional images. IEEE Trans Med Imag 12(3):439–450, 1993

41. Herman GT, Zheng J, Bucholtz CA: Shape-based interpolation. IEEE Comput Graph Appl 12(3):69–79, 1992

42. Treece GM: Volume Measurement and Surface Visualisation in Sequential Freehand 3D Ultrasound. Ph.D. Thesis, Cambridge University, 2000

43. Wells WM III, Viola P, Atsumi H, Makajima S, Kikinis R: Multi-modal volume registration by maximization of mutual information. Med Image Anal 1(1):35–51, 1996

44. Lauchaud JO, Montanvert A: Continuous analogs of digital boundaries: A topological approach to iso-surfaces. Graph Models 62(3):129–164, 2000

45. Gerig G, Kubler O, Kikinis R, Jolesz FA: Nonlinear anisotropic filtering of MRI data. IEEE Trans Med Imag 11 (2):221–232, 1992

46. Saha PK, Udupa JK: Scale-based image filtering preserving boundary sharpness and fine structure. IEEE Trans Med Imag 20(11):1140–1155, 2001

47. Perona P, Malik J: Scale-space and edge detection using anisotropic diffusion. IEEE Trans Pattern Anal Mach Intell 12 (7):629–639, 1990

48. Zhuge Y, Udupa JK, Liu J, Saha PK, Iwanaga T: Scale-based method for correcting background intensity variation in acquired images. Proc SPIE 4684:1103–1111, 2002

49. Udupa JK: Multidimensional digital boundaries. CVGIP Graph Models Image Process 50(4):311–323, 1994

50. Udupa JK, Srihari SN, Herman GT: Boundary detection in multidimensions. IEEE Trans Pattern Anal Mach Intell 4:41–50, 1982

51. Boykov Y, Veksler O, Zabih R: Fast approximate energy minimization via graph cuts. IEEE Trans Pattern Anal Mach Intell 23:1222–1239, 2001

52. Udupa J, Samarasekera S: Fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation. Graph Models Image Process 58:246–261, 1996

53. Saha P, Udupa J, Odhner D: Scale-based fuzzy connected image segmentation: Theory, algorithms and validation. Comput Vis Image Underst 77:145–174, 2000

54. Saha P, Udupa J: Relative fuzzy connectedness among multiple objects: Theory, algorithms, and applications in image segmentation. Comput Vis Image Underst 82:42–56, 2001

55. Saha P, Udupa J: Fuzzy connected object delineation: Axiomatic path strength definition and the case of multiple seeds. Comput Vis Image Underst 83:275–295, 2001

56. Udupa J, Saha P, Lotufo R: Relative fuzzy connectedness and object definition: Theory, algorithms, and applications in image segmentation. IEEE Trans Pattern Anal Mach Intell 24:1485–1500, 2002

57. Jones T: Image-Based Ventricular Blood Flow Analysis. Doctoral Dissertation, University of Pennsylvania, 1998

58. Cutrona J, Bonnet N: Two methods for semi-automatic image segmentation based on fuzzy connectedness and watersheds. France-Iberic Microscopy Congress, Barcelona, 2001, pp 23–24

59. He R, Narayana P: Detection and delineation of multiple sclerosis lesions in gadolinium-enhanced 3D Tl-weighted MRI data. In: Proceedings of IEEE Symposium on Computer Based Medical Systems, 2000

60. Aldeliesten T, Niessen W, Vincken K, Maintz J, Jansen F, van Nieuwenhuizen O, Viergever M: Objective and reproducible segmentation and quantification of tuberous sclerosis lesions in FLAIR brain MR images. Proc SPIE 4322:1509–1518, 2001

61. Jin Y, Laine A, Imielinska C: An adaptive speed term based on homogeneity for level-set segmentation. Proc SPIE 4684(1):383–390, 2002

62. Henn S, Lemole MG, Ferreira MAT, Gonzalez FL, Schornak M, Preul MC, Spetzler RF: Interactive stereoscopic virtual reality: a new tool for neurosurgical education. J Neurosurg 96(1):144–149, 2002

63. Sethian J: Level Set Methods. Cambridge: Cambridge University Press, 1996

64. Raya SP, Udupa JK, Barrett WA: A PC-based 3D imaging system: algorithms, software, and hardware considerations. Comput Med Imaging Graph 14(5):353–370, 1990

65. Frieder G, Gordon D, Reynolds RA: Back-to-front display of voxel-based objects. IEEE Comput Graph Appl 5:52–60, 1985

66. Mockus A, Fielding RT, Herbsleb JD: Two case studies of open source software development: Apache and Mozilla. ACM Trans Softw Eng Methodol 11(3):309–346, 2002

67. Cochran S: wxWindows 2.2 offers cross-platform alternative to Java. Dr Dobb's J, August 2000

68. Zeitlin V: The wxWindows cross-platform framework. Dr Dobb's J, May 2001

69. Rampersad T: wxWindows for cross-platform coding. Linux J 2003(111):6, 2003

70. Udupa JK, Grevera GJ, Odhner D, Zhuge Y, Souza A, Mishra S, Iwanaga T: CAVASS: a computer-assisted visualization and analysis software system—image processing aspects. SPIE Proc 6509, 2007

71. Grevera GJ, Udupa JK, Odhner D, Zhuge Y, Souza A, Mishra S, Iwanaga T: CAVASS: a computer assisted visualization and analysis software system—visualization aspects. SPIE Proc 6509, 2007

72. Grevera GJ, Udupa JK, Odhner D, Zhuge Y, Souza A, Mishra S, Iwanaga T: Introducing CAVASS: a computer-assisted visualization and analysis software system. SPIE Proc 6516, 2007