

The architecture and performance of CAVASS

George Grevera^{*ab}, Jayaram Udupa^b, Dewey Odhner^b, Ying Zhuge^b, and Andre Souza^b

^aDepartment of Mathematics and Computer Science, Saint Joseph's University, 5600 City Avenue, Philadelphia, PA 19131;

^bMedical Image Processing Group (MIPG), Department of Radiology, University of Pennsylvania, 423 Guardian Drive, 4th Floor Blockley Hall, Philadelphia, PA 19104-6021

*ggrevera@sju.edu; phone 1 610 660-1535; fax 1 610 660-3082; www.sju.edu/~ggrevera

ABSTRACT

Our group has been developing medical image software systems since the early 1980s. Our latest system, CAVASS, is freely available, open source, integrated with popular toolkits, and runs on Windows, Unix, Linux, and Mac OS. The architecture of CAVASS incorporates parallel processing by exploiting inexpensive networks of workstations. CAVASS is directed at the visualization, processing, and analysis of nD medical imagery, so support for large medical imagery data and the efficient implementation of algorithms is given paramount importance. We describe the architecture of CAVASS, the parallelization strategy, and present the results of comparing the implementation of CAVASS algorithms with similar algorithms in ITK and VTK for a host of operations.

Keywords: Visualization; surface rendering; volume rendering; 3D imaging; software systems; image analysis.

1. INTRODUCTION

The development in our group (MIPG – Medical Image Processing Group at the University of Pennsylvania) of the concepts within CAVASS and its progenitor 3DVIEWNIX [1] dates back to the 1970s [2]. Since then, a series of software packages for Computer Assisted Visualization and Analysis (CAVA) of images have been developed and distributed by our group [2-5]. CAVASS, an open source system, is the latest in this series, and represents the next major incarnation of 3DVIEWNIX. The key features of CAVASS are: (1) most major CAVA operations incorporated; (2) very efficient algorithms/implementations; (3) parallelized algorithms for computationally intensive operations; (4) parallel implementation on a cluster of PCs; (5) interface to other systems such as CAD/CAM software, ITK [6], VTK [7], and statistical packages; (6) an easy to use GUI (Graphical User Interface). In general, one can identify four different user groups (UG) for CAVA software packages: (UG1) CAVA basic researchers/technology developers, (UG2) CAVA application developers, (UG3) users of CAVA methods in clinical research, and (UG4) clinical end users in patient care. CAVASS, in particular, is aimed at UG1 through UG3 (as these users have many needs in common whereas UG4 has much different needs).

CAVASS incorporates four groups of operations: (a) *image processing* for enhancing information about and defining an object system, (b) *visualization* for viewing and comprehending an object system, (c) *manipulation* for altering an object system (useful for virtual surgery), and (d) *analysis* for quantifying information about an object system. Image processing operations can be further dichotomized into scene operations and structure operations. The native CAVASS file format, inherited from 3DVIEWNIX, includes scene data, a multidimensional extension [8,9] to the DICOM format [10], as well as structure data for the representation and manipulations of object systems (most often extracted from scene data). Image processing operations on scene data include VOI (Volume Of Interest), Interpolate, Filter, Segment, Classify, Algebra, and Registration. Image processing operations on structure data includes surface normal calculation, the merging of structures, conversion to structures, and conversion to scenes. Visualization operations in CAVASS are divided into slice, surface, and volume. Slice visualization includes montage, reslice, cycle, and overlay. Surface and volume visualization includes the operations of view, measure, and create movie. Manipulate operations include select slice, measure, reflect, cut, move, and create movie. Analyze operations on scene data include density profile and ROI (Region Of Interest), and kinematics for structure data.

In what follows, we describe in detail the architecture, GUI, interface with other systems, and parallelization strategy of CAVASS which are based on our experiences in developing 3DVIEWNIX. We present the results of comparing the implementation of a host of common visualization and image processing operations in CAVASS with ITK. Our conclusions are based on assessing performance by utilizing regular (6 MB), large (241 MB), and super (873 MB) size 3D image data sets (see Table 1). The results of these tests indicate that CAVASS is considerably more efficient than ITK/VTK for computationally intensive operations. CAVASS can also deal with considerably larger data sets than ITK/VTK. It is easy to learn and is ready to use in applications since it provides an easy to use GUI. The users can easily build a cluster from ordinary inexpensive PCs and reap the full power of CAVASS inexpensively compared with expensive multiprocessing systems which are less efficient for CAVA operations.

Table 1. Description of datasets of varying sizes used in the comparisons.

dataset name	voxel size	image size	data size
regular	0.98 x 0.98 x 3.00 mm	256 x 256 x 46	6 MB
large	0.68 x 0.68 x 1.50 mm	512 x 512 x 459	241 MB
super	0.24 x 0.24 x 0.50 mm	1023 x 1023 x 417	873 MB

2. MATERIALS & METHODS

We now describe in detail the architecture of CAVASS. CAVASS, written in portable C++, provides a portable, cross-platform GUI for the interactive display and processing of large, 3D and higher dimensional medical imagery. CAVASS also provides extremely efficient implementations of the most common visualization and image processing operations. For extremely time consuming operations, CAVASS provides parallel implementations as well. CAVASS is also easily extensible in that it provides an interface to ITK and VTK and other popular CAD/CAM packages.

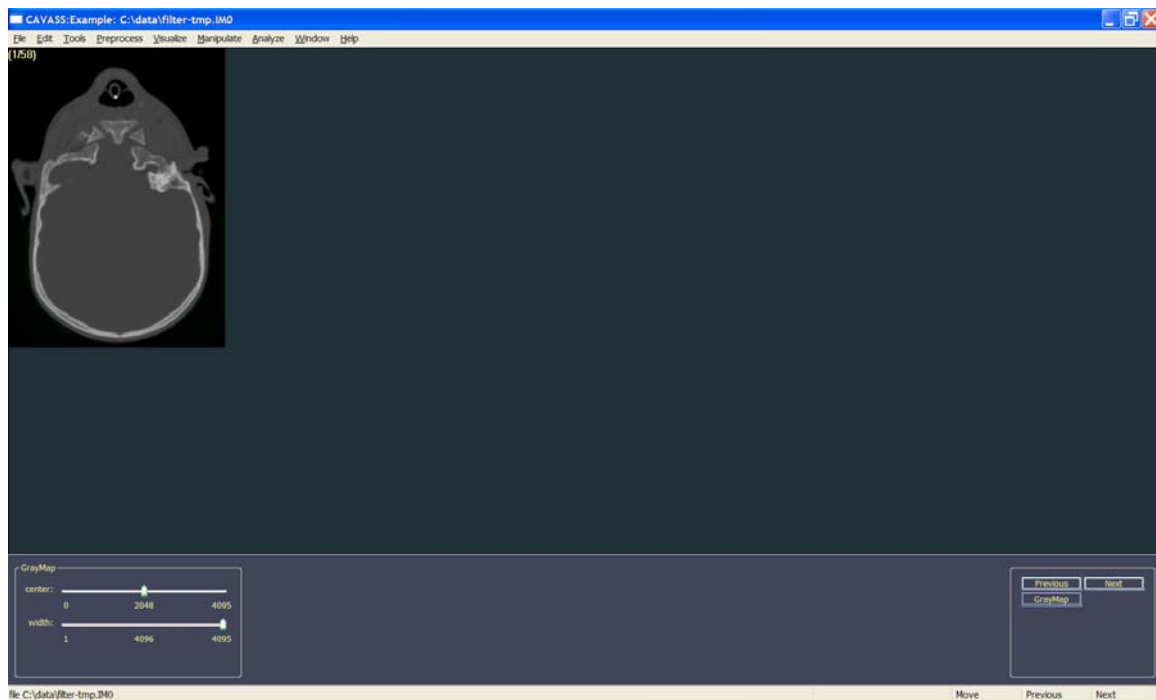


Figure 1. The CAVASS example module.

2.1 User interface

With regard to the user interface, CAVASS provides a cross-platform GUI based on our years of experience with 3DVIEWNIX. CAVASS employs the wxWidgets C++ programming library [11] which provides a single platform independent API. Using this API, one can develop applications that maintain the native look-and-feel of the various

operating systems such as Windows, Linux, Unix, and Mac OS but share a single source code base. Individual CAVASS applications are referred to as modules (because they do not function as standalone applications but contribute to a single, cohesive CAVASS application). For example, image interpolation and slice overlay are two modules. Modules are typically associated with a user interface window referred to as a frame and are subclasses of the MainFrame superclass. Each module inherits a standard menu bar and menu items from MainFrame. The frame for each module is divided in two separate areas: a (top) canvas or drawing/display area inherited from the MainCanvas superclass and a (bottom) control area inherited from MainFrame. The user may hide the control area at any time to devote the entire window content area to image display. Common controls such as an image contrast control (window width and level) are provided as C++ classes for reuse. The control area is further subdivided into two areas. Controls that are always displayed for a particular module appear towards the right and those controls that are more or less transient are displayed towards the left (and may be replaced by other transient controls and reappear later as necessary). A complete example module is provided as well (see Figure 1) which functions as introductory code for those wishing to write modules themselves for CAVASS. This ExampleFrame module and corresponding ExampleCanvas illustrates both the permanent and transient module controls. On the bottom right appear buttons that allow the user to scroll to the previous and next slices and a GrayMap button that, when pressed, causes sliders that allow the user to change the gray scale contrast to be changed via window center and width sliders to be present or absent. wxWidgets also provides platform independent methods of handling non-GUI related issues such as portable file name format handling, process creation and deletion, and the storage of user preferences.

As an aid in learning how to perform common operations in medical imaging, CAVASS includes tutorials. In a tutorial, all parameters are fixed but may include user interaction. For frequently used sequences of operations, CAVASS also provides *tasks* and *recipes*. A task is a useful sequence of operations that can be recalled and executed on multiple input data sets (e.g., thresholding, shape-based interpolation, filtering, and t-shell surface creation and rendering). Task parameters are not fixed and may include user interaction. A recipe is a collection of tasks and can be understood as a sequence of commands for accomplishing an application specific goal (e.g., the determination of the volume of a segmented tumor). Recipes may also include user interaction and the parameters may not be fixed as well.

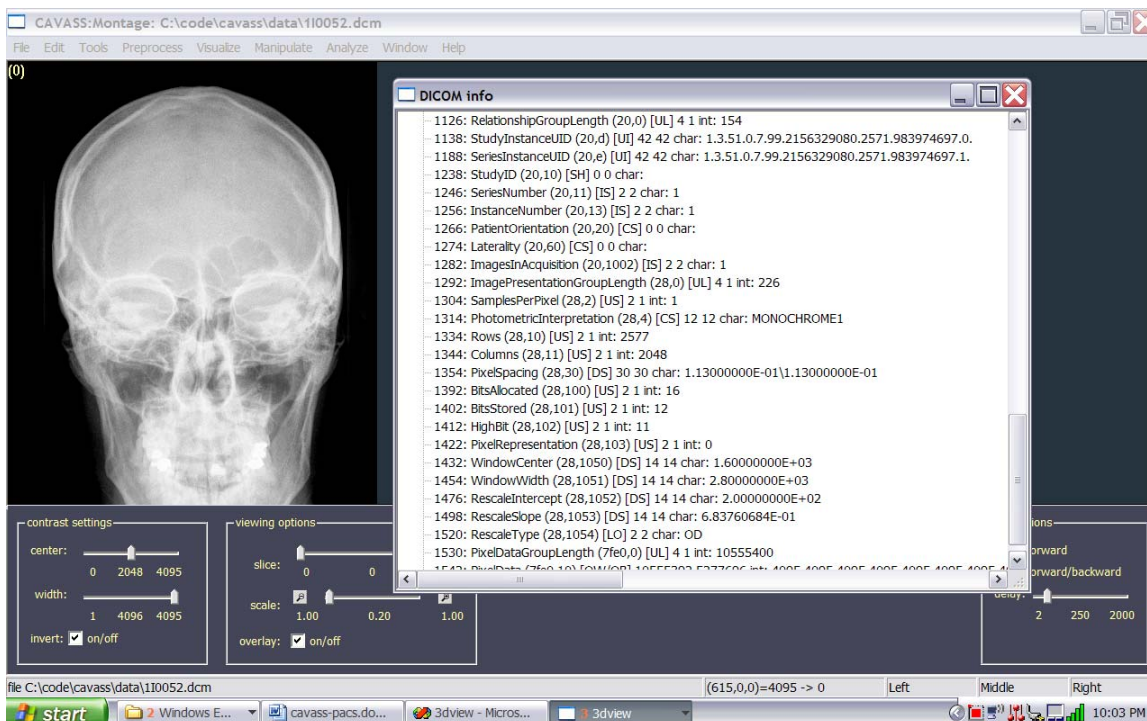


Figure 2. The CAVASS DICOM explorer.

2.2 Parallelization approach

From the perspective of parallelization, CAVA operations may be divided into three groups, which we 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. 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), shape-based (binary as well as gray-level) interpolation, diffusive filtering, inhomogeneity correction, and all non-user-steered slice-by-slice segmentation methods (such as clustering techniques). There are other CAVA operations, which work (chunk-by-chunk) in the above sense but some significant 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. We label those CAVA operations, which require each slice/chunk to be accessed more than once to complete the operation as Type-3. These are more difficult than Type-1 and Type-2 operations to parallelize. They 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 object.

The portable MPI (Message Passing Interface) [12-15] is used for parallelization in CAVASS. Rather than requiring the user to purchase an expensive shared memory multiprocessor systems (such as those supported by the OpenMP standard [16]), the MPI standard allows one to treat an inexpensive network of computers or COW (Cluster of Workstations) as a distributed memory, multiprocessor system. MPI is included in the standard Linux distribution and is available for Windows and Mac OS as well. The nodes that participate in the cluster, however, must be of the same general architecture (e.g., Pentium) and must be executing the same operating system type (e.g., Windows XP) although different processor speeds, RAM amounts, vendor and computer models, and OS versions may participate as nodes in the cluster.

To improve network performance, we also purchased and installed an inexpensive (\$150) switch which increased bandwidth by an order of magnitude (from 100mb/s to 1gb/s). This is an optional albeit significant yet inexpensive improvement.

Table 2. Surface creation file sizes (in bytes) for CAVASS and VTK.

	regular	large	super
VTK	17,243,606	154,114,983	264,811,271
CAVASS BIM	378,654	15,047,556	54,556,813
CAVASS BS0	1,538,092	11,609,160	22,475,066
CAVASS BS1	2,578,038	21,071,204	32,616,578
CAVASS BS2	5,332,709	40,358,005	63,662,354

2.3 Software engineering practices

With regard to software engineering practices employed by the CAVASS team, we use doxygen [17] to generate program documentation including class diagrams, inheritance diagrams, and *todo* lists. This is accomplished by embedding special comments within the source code. Since CAVASS is intended to execute on a variety of platforms/operating systems, we use CMake [18] to automatically generate Makefiles (Linux/Unix/Mac OS) and Visual C++ project files (Windows). For source code control, we use CVS [19] which allows us to track changes to the source code even when those changes are made by different developers at the same time. We have also adopted the methodology of testing our software on three different data sets of increasing size (regular, large, and super) with the largest data set intended to “stress test” our algorithms. Finally, for C++ memory leak detection, we use the method that is available in Visual Studio. Memory leaks occur when one allocates a block of memory and does not free the block when it is no longer used (references). As the program run, it consumes more and more memory and may eventually cause future memory allocations to fail and/or slow the system down a great deal.

Table 3. Surface creation times (in seconds) for CAVASS and VTK.

	regular	large	super
VTK	1.20	20.50	78.15
CAVASS BIM	0.03	1.07	3.83
CAVASS BS0	1.13	21.18	69.25
CAVASS BS1	1.32	24.07	73.40
CAVASS BS2	15.16	236.49	692.66

2.4 Interface to ITK

To provide interoperability with ITK, CAVASS can interface directly with ITK and provide a graphical user interface for it (see Figure 3). We developed a unique approach that allows us to seamlessly yet optionally integrate with ITK without directly linking to it and without being dependent upon it. We chose to not be dependent upon ITK for a number of reasons. (1) If we were to link directly to ITK, then we would have to require the user to install ITK as well or we would have to provide ITK as part of the CAVASS distribution. ITK is a large system and would significantly increase the size of the CAVASS distribution and the time required to build the entire system including ITK. If we were to not distribute ITK with CAVASS, the user would have to independently obtain and build it. In all likelihood, the version of ITK obtained in this manner would be different from the ITK version used for CAVASS testing. In this case, it would be difficult to us to insure compatibility between the two systems. (2) Linking directly with ITK also makes error handling much more difficult. For example, if there is a memory access violation/error in ITK, the entire application including CAVASS would be aborted by the operating system. (3) Linking directly with ITK would significantly increase the size of the CAVASS executable program. Large executables require more memory and require more time to initially begin execution. (4) CAVASS, based upon 3DVIEWNIX, contains very efficient implementations of the most commonly used algorithms. Therefore, CAVASS can function as a complete standalone application suite in its own right.

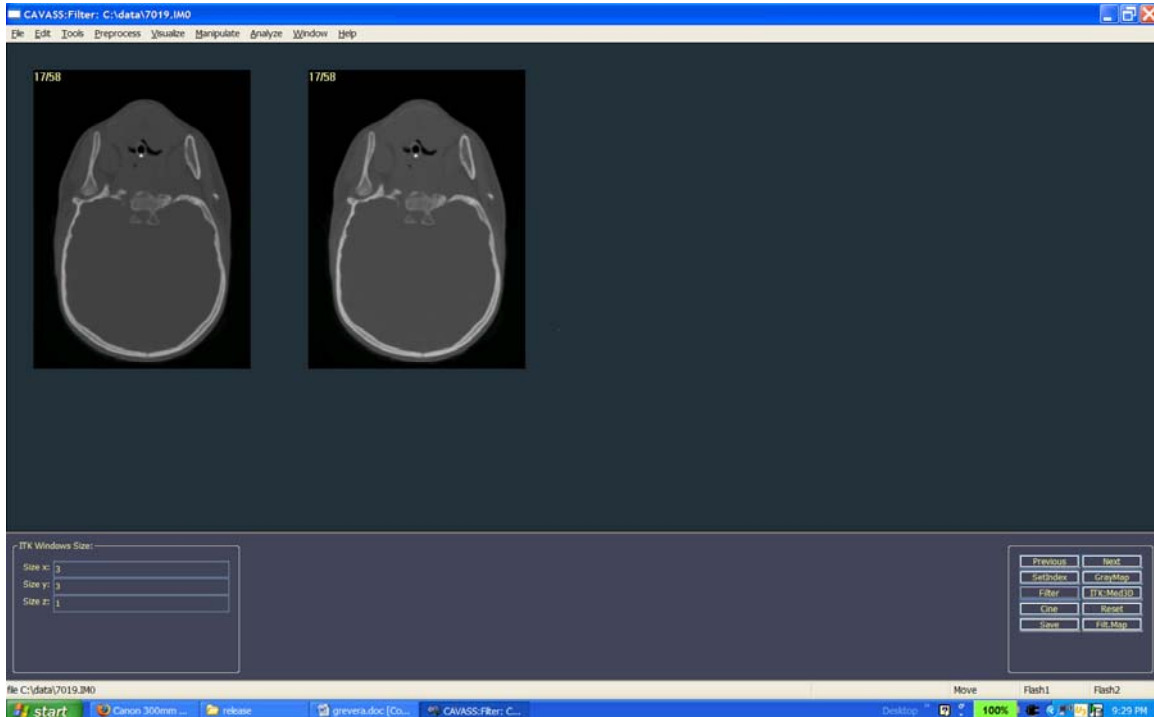


Figure 3. An example of CAVASS integration with ITK (ITK median filter example). The controls on the left, bottom side allow the user to set the spatial extent of the ITK filter. The controls on the right, bottom allow the user to view different slices in the volume, change the contrast of both the original and filtered slice, and process the entire volume with the ITK median filter.

Our solution to these issues is to provide an extendable user interface for ITK functions that allows the user to view a data slice and to graphically set the parameters that correspond to the ITK function. Once the parameters are set, the user may press a button to view the result of the ITK operation applied to a single data slice. The result is displayed next to the original slice. If the user is satisfied, the user may press a button that causes the ITK operation to be applied to the entire volume of data. The user may also optionally view other slices in the volume, readjust parameters, and process that slice and view the result or apply the new settings to the entire volume. The ITK function are not linked directly into CAVASS but are completely standalone command line programs that take input and output from files and parameters from either the command line or from a parameter file. The buttons in the user interface execute these command line programs on either the entire volume data file or on temporary files created from the displayed slice. In this way, potential errors in the ITK command line programs can be detected by CAVASS but will not cause CAVASS to abort. Additionally, CAVASS can execute the ITK programs as background processes and free the user interface program to perform other tasks. This is all accomplished in a portable manner using wxWidgets. To date, the following ITK programs have been integrated with CAVASS: itkBinaryDilateFilter, itkBinaryErodeFilter, itkBinaryMedianFilter, itkBinaryMorphFilter, itkBinaryThresholdFilter, itkBinomialBlurFilter, itkCannyEdgeDetectionFilter, itkCurvatureAnisotropicDiffusionFilter, itkCurvatureFlowImageFilter, itkDanielssonDistanceMapFilter, itkDerivativeFilter, itkDiscreteGaussianFilter, itkGradientAnisotropicDiffusionFilter, itkGradientMagnitudeFilter, itkGradientMagnitudeRecursiveGaussianFilter, itkGrayDilateFilter, itkGrayErodeFilter, itkMeanFilter, itkMedianFilter, itkRescaleFilter, itkSigmoidFilter, itkSmoothRecursiveGaussianFilter, itkThresholdFilter, itkVotingBinaryHoleFillingFilter. Using ITK's observer and progress events, we provide a class that monitors and reports the progress of ITK function. This information is reported to standard output which is monitored by a thread in the CAVASS process that initiated the ITK function. This information is reported to the user by the CAVASS process via a dialog box which also allows the user to cancel the operation if it is taking too long. In that way, CAVASS appears to be seamlessly integrated with ITK.

Table 4. Surface rendering timing comparison (in seconds) for CAVASS (sequential implementation with and without antialiasing) and surface rendering as implemented in VTK.

	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

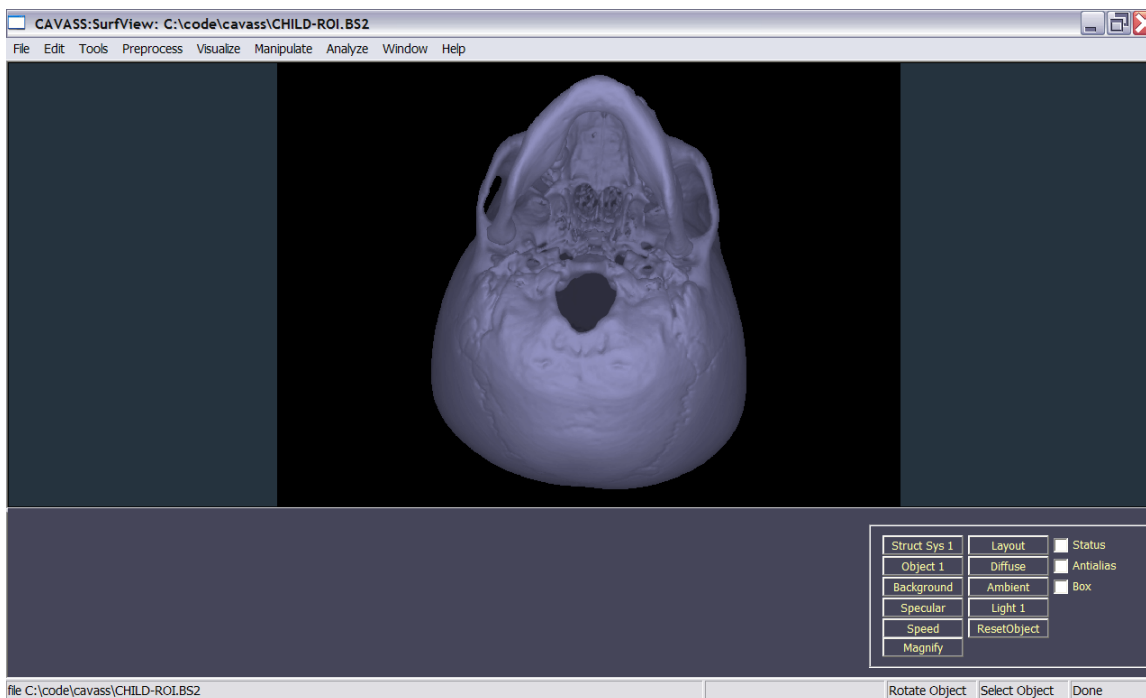


Figure 4. CAVASS t-shell surface rendering [31] example.

2.5 Supported data file formats

First and foremost, CAVASS supports the 3DVIEWNIX formats [8,9] which are, in general, an n-dimensional generalization of the DICOM standard [10]. In particular, (1) IMO files contain n-dimensional, possibly vector-valued scene data. (2) BIM files represent packed binary scenes. (3) SH0 represents structure data of type Shell0. Any Shell0 type shell, binary or non-binary, can be represented in these files. Shell0 structures are represented as sets of voxels wherein every voxel has a set of attributes associated with it. (4) SH1 files contain structure data of type Shell1. Any Shell1 type shell, binary or non-binary, can be represented in these files. Shell1 structures are represented as sets of voxel faces wherein every face has a set of attributes associated with it. (5) BS2 files represent triangulated surfaces (see Figure 4). (6) PLN files contain structure plans.

Additionally, CAVASS provides an interface that parses DICOM files and creates volumes from the individual DICOM files. CAVASS can also load and display individual DICOM files directly and display the contents of the DICOM header for the user (see Figure 2). In addition to the DICOM and 3DVIEWNIX formats, CAVASS can also import and export data in a wide variety of common formats such as VTK [20], PNM [21], TIFF [], JPEG [], GIF [], and STL (STereo Lithography - for CAD/CAM packages) [25].

All of these file formats are represented by a common C++ class, CavassData, so that the algorithms remain independent of the particular file format that was used for storage.

Table 5. Volume rendering timing comparison (in seconds) for sequential and parallel implementations of CAVASS shell rendering, VTK ray casting, and VTK 2D texture mapped volume rendering.

	CAVASS		VTK	
	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

3. RESULTS

The results of a performance comparison between CAVASS, VTK, and ITK are shown in Tables 2 through 6. All of these comparisons used our standard test suite of 3D medical image data sets (see Table 1). The regular and large data sets are of typical size while the super size data set is intended to stress and test the limits of the implementations.

The results of our first comparison are shown in Tables 2 and 3. A typical visualization operation is the creation/determination of a specific surface for subsequent surface rendering or analysis. Given the same threshold and input data set, VTK and CAVASS are assigned the task of creating surface files. We measured the time required to create these files as well as the subsequent sizes of the files. All formats are binary (as compared with ASCII) and none contain compression (i.e., LZW [26,27] or DCT [28]). CAVASS supports a variety of formats (BIM=binary, BS0=voxel, BS1=voxel faces, and BS2=triangulated surface) that can be directly rendered. These results demonstrate that CAVASS can quickly create BIM files that are very compact. BS0 and BS1 require approximately the same time to create as VTK files although they are much smaller. The BS2 file requires more time to create than the VTK files but is much more compact.

Table 4 contains the time required to render surfaces in CAVASS and VTK by a variety of methods. CAVASS is much faster in rendering surfaces entirely in portable software than VTK can render the same surfaces with hardware assistance using the shell rendering technique [29].

For the task for volume rendering, Table 5 illustrates that CAVASS sequential volume rendering (implemented entirely in software) can render at rates comparable to VTK ray casting and much faster than VTK texture mapping. CAVASS parallel volume rendering (again, implemented entire in software) can render at rates much faster than VTK with hardware assistance.

Finally, Table 6 illustrates that CAVASS is much faster than ITK for common image processing operations and CAVASS can process large data sets that cause ITK to fail. It is important to note that most ITK filters are inherently parallel (via multithreading) and take advantage of dual core/multi core processors. Note that although the vast majority of CAVASS operations are sequential (and therefore only use a single core), they were still faster than the corresponding ITK operations.

4. CONCLUSIONS

We described in detail the new architecture of CAVASS. This work has not been described elsewhere. CAVASS is an extremely efficient yet flexible architecture that can be used to develop cross-platform applications with a standard user interface. We also described the cross-platform architecture of CAVASS and have demonstrated that it is very flexible and efficient. We have tested it on a variety of data sets. The results of these tests indicate that CAVASS is capable of processing data sets at higher rates than other systems and that CAVASS can process data sets of sizes that make other systems fail.

More information regarding CAVASS is available from www.mipg.upenn.edu/~cavass [31].

Table 6. Time in seconds for the various operations for the regular (6 MB), large (241 MB), and super (873 MB) image data sets. 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.

	regular		large		super	
	ITK	CAVASS	ITK	CAVASS	ITK	CAVASS
gaussian 2d	1.01	0.2	27.89	7.42	?	28.76
gaussian 3d	1.28	0.22	38.67	9.2	?	36.45
seq linear interpolation	2.90	0.60	87.70	54.90	?	139.10
par linear interpolation	1.70 [2]	1.00 [2]	62.80 [2]	14.90 [2]	?	49.20 [2]
median 2d	1.25	0.75	45.24	28.44	132.76	70.38
median 3d	5.75	2.84	165.45	109.09	466.93	323.91
threshold	0.08	0.03	3.78	1.1	7.39	3.89
binary erode	1.98	0.06	63.53	2.26	210.84	8.03
binary dilate	3.15	0.06	137.13	2.34	538.22	8.39
gray dilate	3.34	0.13	99.03	5.19	352.35	16.94
gray erode	3.31	0.13	100.27	5.12	380.55	17.19
logical (&)	0.07	0.01	1.92	0.11	7.32	0.37
algebra (a+b)	0.11	0.28	3.78	11.23	?	41.11
algebra (aa+ab)	0.17	0.29	5.38	11.17	?	40.66
seq rigid MI registration	57.20	56.10	?	1860.60	?	3863.40
par rigid MI registration	NA	8.60 [5]	NA	301.60 [5]	NA	1089.10 [5]
seq affine MI registration	208.30	155.30	?	3602.40	?	13111.00
par affine MI registration	NA	25.10 [5]	NA	1018.60 [5]	NA	3662.20 [5]
anisotropic diffusion	41.67	13.49	1553.11	526.22		
fuzzy connectedness seg	108.4	49.5	?	843.7	?	?
fuzzy connectedness seg	108.4	17.8 [5]	?	298.6 [5]	?	1312.6 [5]

ACKNOWLEDGEMENTS

The authors gratefully acknowledge support for this work from DHHS grant EB004395.

REFERENCES

1. J.K. Udupa, D. Odhner, S. Samarasekera, R. Goncalves, K. Iyer, K. Venugopal, and S. Furuie: "3DVIEWNIX: an

- open, transportable, multidimensional, multimodality, multiparametric imaging software system," in SPIE Proceedings 2164:58-73, 1994.
2. J.K. Udupa: "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, New York, 1980.
 3. J.K. Udupa: "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, April 1983.
 4. L.S. Chen, G.T. Herman, C.R. Meyer, R.A. Reynolds, and J.K. Udupa: "3D83 - An easy-to-use software package for three-dimensional display from computed tomograms," Proceedings of IEEE Computer Society International Symposium on Medical Images and Icons, Arlington, Virginia, pp. 309-316, 1984.
 5. J.K. Udupa, G.T. Herman, P.S. Margasahayam, L.S. Chen, and C.R. Meyer: "3D83: A turnkey system for the display and analysis of 3D medical objects," SPIE Proceedings 671:154-168, 1986.
 6. www.itk.org.
 7. www.vtk.org.
 8. J.K. Udupa, H.M. Hung, D. Odhner, and R. Goncalves: "Multidimensional data format specification: A generalization of the American College of Radiology National Electric Manufacturers Association Standards," Journal of Digital Imaging 5(1):26-45, 1992.
 9. J.K. Udupa, H.M. Hung, D. Odhner, and R. Goncalves: "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, January 1991.
 10. National Electrical Manufactures Association (NEMA): Digital Imaging and Communication in Medicine (DICOM) Part 1: Introduction and Overview, NEMA, Washington, DC, 1993.
 11. www.wxwidgets.org.
 12. www.mpi-forum.org.
 13. www.lam-mpi.org.
 14. www.mcs.anl.gov/research/projects/mpich2/.
 15. www.open-mpi.org.
 16. www.openmp.org.
 17. www.doxygen.org.
 18. www.cmake.org.
 19. www.cvs.org.
 20. www.vtk.org/pdf/file-formats.pdf.
 21. www.fileformat.info/format/pbm/egff.htm.
 22. partners.adobe.com/public/developer/en/tiff/TIFF6.pdf.
 23. www.jpeg.org/jpeg/.
 24. www.w3.org/Graphics/GIF/spec-gif89a.txt.
 25. [en.wikipedia.org/wiki/STL_\(file_format\)](http://en.wikipedia.org/wiki/STL_(file_format)).
 26. Welch, T. A. (June 1984). "A technique for high-performance data compression." Computer. Vol. 17, pp. 8-19.
 27. Ziv, J., and Lempel, A.: Compression of Individual Sequences Via Variable-Rate Coding, IEEE Transactions on Information Theory, September 1978.
 28. Ahmed, N., Natarajan, T., and Rao, K.R.: "Discrete Cosine Transform", IEEE Trans. Computers, 90-93, Jan 1974.
 29. Udupa, J.K., and Odhner, D.: "Shell rendering," IEEE Computer Graphics and Applications 13(6):58-67, 1993.
 30. Grevera, G., Udupa, J., Odhner, D., Zhuge, Y., Souza, A., Iwanaga, T., and Mishra, S., "CAVASS: a Computer Assisted Visualization and Analysis Software System. " J. Digital Imaging 20(S1):101-118, 2007.
 31. Grevera, G.J., and Udupa, J.K.: "T-Shell Rendering and Manipulation." Proc. SPIE Medical Imaging 5744:22-33, San Diego, CA, 2005.