CAVASS: A Computer Assisted Visualization and Analysis Software System – Image Processing Aspects

Jayaram K. Udupa[*a], George J. Grevera[a,b], Dewey Odhner[a], Ying Zhuge[a], Andre Souza[a], Shipra Mishra[a] and Tad Iwanaga[a]

[a]Medical Image Processing Group - Department of Radiology
University of Pennsylvania - Philadelphia, PA, USA

[b]Department of Mathematics and Computer Science
Saint Joseph's University - Philadelphia, PA, USA

ABSTRACT

The development of the concepts within 3*DVIEWNIX* and of the software system 3*DVIEWNIX* itself dates back to the 1970s. Since then, a series of software packages for Computer Assisted Visualization and Analysis (*CAVA*) of images came out from our group, 3*DVIEWNIX* released in 1993, being the most recent, and all were distributed with source code. *CAVASS*, an open source system, is the latest in this series, and represents the next major incarnation of 3*DVIEWNIX*. It incorporates four groups of operations: IMAGE PROCESSING (including *ROI*, interpolation, filtering, segmentation, registration, morphological, and algebraic operations), VISUALIZATION (including slice display, reslicing, *MIP*, surface rendering, and volume rendering), MANIPULATION (for modifying structures and surgery simulation), ANALYSIS (various ways of extracting quantitative information). *CAVASS* is designed to work on all platforms. Its key features are: (1) most major *CAVA* operations incorporated; (2) very efficient algorithms and their highly efficient implementations; (3) parallelized algorithms for computationally intensive operations; (4) parallel implementation via distributed computing on a cluster of PCs; (5) interface to other systems such as *CAD/CAM* software, *ITK*, and statistical packages; (6) easy to use *GUI*. In this paper, we focus on the image processing operations and compare the performance of *CAVASS* with that of *ITK*. Our conclusions based on assessing performance by utilizing a regular (6 MB), large (241 MB), and a super (873 MB) 3D image data set are as follows: *CAVASS* is considerably more efficient than *ITK*, especially in those operations which are computationally intensive. It can handle considerably larger data sets than *ITK*. It is easy and 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 to expensive multiprocessing systems which are less efficient for *CAVA* operations.

***Keywords*:** visualization, 3D imaging, software systems, image analysis

1. INTRODUCTION

1.1 Background

A long-standing activity in medical imaging that is begging for a name (and an acronym) is what we refer to as *Computer Aided Visualization and Analysis* (*CAVA* for short). When images are acquired for a body region *B*, there is usually an object system *O* of study, which is simply a set of organs within the body region *B*. The purpose of image acquisition is usually to fulfill some combination of the following six objectives: (a) the diagnosis of a disease of *O*, (b) understanding the natural course of a disease of *O*, (c) understanding the function of *O*, (d) planning a treatment for a disease of *O*, (e) studying the effects of a treatment for a disease of *O*, and (f) medical education related to *O*. Given a set *S* of multidimensional, potentially multimodality, images acquired for a given body region *B*, to fulfill the above six objectives, the following four types of computerized operations are generally needed: (E1) image processing, (E2) visualization, (E3) manipulation, and (E4) analysis. The purpose of E1 is to *process* the images in *S* so as to enhance/define/extract model information about *O*. The commonly used operations include interpolation, filtering, registration, segmentation, morphological operations, and image algebra. The goal of E2 is to *visualize O* in its true form, shape, and function. Various forms of slice display, and surface and volume rendering come under this group. E3

*jay@mipg.upenn.edu, www.mipg.upenn.edu/□ *cavass*.

allows the object information obtained in E1 to be *altered*; for example, virtual surgical operations can be performed on the *structures,* which are computer representations of *O* derived from *S* via operations in E1. The aim of E4 is to derive *quantitative information* about the morphology, architecture, kinematics, mechanics, and other functions of the objects in *O*. The four groups of operations are interdependent for their effectiveness.

Some comments are in order regarding the choice of the phrase *Computer Aided Visualization and Analysis*, or *CAVA*, vis-à-vis another important area, which has come to be commonly known as *CAD*, or *Computed Aided Diagnosis*. In our view, *CAVA* is a much broader activity than *CAD* and predates *CAD*. While the latter focuses mainly on diagnosis, the former encompasses activities related to the development of new visualization, image processing, and quantitative image analysis tools to facilitate *CAD*, to facilitate new therapeutic strategies, and to help in basic clinical research, education, and training. Although there is some overlap, in terms of operations between *CAVA* and *CAD*, *CAVA* has its own unique challenges, requirements, and solutions.

The subject matter of this paper is a software system for *CAVA* developed in our group, which is on the verge of being completed and released. The name of the software system is *CAVASS*, which stands for "*CAVA* Software System". The methods, principles, and algorithms underlying *CAVASS* and the development of *CAVASS* itself dates back to the 1970s in our group [1]. Since then, a series of software packages [2-5] for *CAVA* came out from our group, 3*DVIEWNIX* released in 1993, being the most recent, and all research software packages [1-3] were distributed with source code. *CAVASS*, an open source system, is the latest in this series, and represents the next major incarnation of 3*DVIEWNIX*.

1.2 Requirements of CAVA Software

There has been a steady increase during the past 25 years and a rapid increase during the past 10 years in activities related to *CAVA*. With the increased focus on Molecular Imaging Techniques and on the efficacy (higher speed, higher spatial and temporal resolution, and lower dose) of x-ray *CT*, and because of the availability of diffusion/perfusion imaging techniques and combined *PET*/*CT*, the *CAVA* activities are likely to continue to increase. Keeping these exciting current developments in mind, we believe that any software system for *CAVA*, particularly open-source, should satisfy the following seven major requirements RQ1-RQ7. (RQ1) portability, (RQ2) ease of use, (RQ3) freedom from size restrictions, (RQ4) adequacy of speed of processing, (RQ5) comprehensiveness, (RQ6) easy interfaceability to other packages, and (RQ7) availability/cost.

With the currently available software engineering tools, portability (RQ1) is easy to provide. The design of *ITK* [6] reflects this fact well. However, the portability issue becomes more complex if the user interface is considered in the software. Ease of use (RQ2) has different implications depending on the target user group. We believe that there are four types of *user groups* in *CAVA*: (UG1) *CAVA* technology developers, (UG2) *CAVA* application developers, (UG3) users of *CAVA* in clinical research, (UG4) clinical end users of *CAVA* for day-to-day patient care. Open-source software activities are not suited for UG4, mainly due to the safety, legal, and financial issues related to patient care. For UG1, RQ2 implies that the software system should provide a rich set of basic library functions for handling image and non-image data structures, graphical operations, display operations, and low-level image computing operations, and it should be straightforward to implement users' own new algorithms for any of the elements E1-E4 by using these library functions. For UG2, what RQ2 means is that the software system should provide a rich set of high-level functions incorporating a rich variety of algorithms for *all* of E1-E4, and it should be straightforward to develop a new application by utilizing these functions within the environment of the software system. UG3 is a distinct group of users, which is currently rapidly growing in Radiology and in various other medical disciplines. The meaning of ease of use for this group is that it should be straightforward to utilize the software system in a clinical research project or trial, and tools to make whatever modifications are needed (such as scripting, creating a new procedure) should be available in the software system. RQ3 implies that, within reasonable limits, there should not be restrictions on the size of the images, dimensionality of the images, the number of components in the case of vectorial images, and on the number of bits allowed for each component value. Adequacy of speed of processing (RQ4) means that, for most key operations under E1-E4, by using well-known and important methods, the speed of processing on a commonly available computer platform of reasonable cost should be acceptable. "Acceptable speed" ideally means interactive response or better for visualization and manipulation tasks, and within a few minutes for other tasks. RQ5 implies that the software system should incorporate all key methods for all four elements E1-E4 of *CAVA*. RQ6 implies that the software should provide ready interface to other *CAVA* and non-*CAVA* systems. Availability/cost (RQ7) means that the software should be open-source/conditional open-source and as inexpensive as possible to obtain, install, operate, and update.

1.3 A Review of *CAVA* Software Development Activities

Although 2D images were available since the invention of x-rays, *CAVA* for 3D images started in the 1970s with the advent of x-ray *CT*. The earliest known software development [2] for *CAVA* also started in the 1970s. This software implemented a digital surface detection algorithm [7], linear interpolation, digital surface rendering [8], and some simple surface quantification techniques [9]. A vastly more streamlined and improved version of the software named *DISPLAY*82 [3] was released in late 1982 which removed image size restrictions, established a standard for image and structure format, and recognizing the difficulties in segmentation, incorporated a gesture-controlled interactive segmentation method [10]. This software, in spite of its machine dependency (Data General Eclipse mini-computer driving a Comtal display frame buffer), was distributed with source code to over 150 sites worldwide within the next 2 years. This was the first "open source" activity that we know of in *CAVA* in spite of severe handicaps due to the lack of software, graphics, and data standards for portability. Earlier, *DISPLAY* [2] was implemented in 1980 and 1981 on the Independent Physician Display Console of the *GE* 8800*CT/T* scanner, which *GE* demonstrated at *RSNA* in 1981. This seems to have been the first attempt to transfer the 3D *CAVA* technology to medical imaging industry and the first industrial entry of this technology. Subsequently all major *CT* scanner vendors, (Siemens, Philips, Picker, Technicare, Thompson CGR, Elsinct, Toshiba) started a 3D *CAVA* software development activity. The first commercial package for 3D *CAVA* was 3D83 [4]. It was based on *DISPLAY*82 and was marketed by *GE*. Although developed in the computer graphics world, there was another early package called Movie-*BYU* [11] developed at Brigham Young University, Provo, Utah, which took the approach of polygonal surface representation. It did not have tools to handle images. Slice-by-slice contour tiling methods [12], which, were in vogue then, were used to go from images to surfaces. Although there was ongoing research on *CAVA* at several centers at this time, the above description of software development for distribution constitutes the earliest historic phase of 3D *CAVA*.

There were several key developments during the mid 1980s, which cast a lasting impact on *CAVA* and related software development. The first key development was the introduction of workstations into the computing milieu, which provided more computing resources and also removed the problems related to the development of machine-dependent graphics interfaces to frame buffers. As a consequence, workstation-based software development became an active area. The development of several subsequently well-known software (and hardware) products started around this time, notable among these being Analyze [13], products from Contour Medical systems [14] (which subsequently changed name to *CEMAX*), Dimensional Medicine, Inc. [15], Phoenix Data Systems [16], *IBM* [17], Multi-Planar Diagnostic Imaging [18], Virtual Imaging [19], ISG Technologies [20], software developed by Vannier et al. [21] which was utilized in the early products of Siemens, and the software developed by Hohne's group which subsequently came to be known as Voxel Man [22]. All these products provided some elementary image processing operations (E1), voxel-based surface rendering and arbitrary plane (and curved surface) reslicing of 3D image data for visualization (E2), and some analysis techniques (E4).

The second important development was the advent of *MRI* in the 1980s. This gave rise to a spurt of activity in E1, particularly in image segmentation [23], image filtering [24], image inhomogeneity correction [25], image interpolation [26], and importantly, in image registration [27] to match among *CT*, *MRI* and *PET* images of the same subject. With the increasing data volume and demand for interactive speed of processing, solutions were sought for improving the speed of processing. For some reason, the focus on speed was always as related to visualization. Hardware accelerator graphics boards and specialized rendering engines were built not just for *CAVA* (or only for visualization in *CAVA*) with medical imaging in mind but to cater to other computer graphics and scientific visualization applications also. Notable among these developments were the *TAAC* and other boards from *SUN* Micro Systems, the Silicon Graphics rendering engines, the *AT&T* Pixel Machine, the workstations from Stellar and Ardent (the two subsequently merged and became Stardent), and the workstation from Dynamic Digital Displays.

The third key development was the advent of volume rendering [28, 29]. The key idea behind volume rendering was to render 3D regions (rather than just surfaces of structures) by using opacity and color to enhance or suppress various aspects of the objects as they manifest themselves in the 3D image. Pixar brought out a special workstation to perform rapid hardware-based volume rendering. Although its speed was nowhere near real time, it was much faster than software-based rendering on other powerful workstations at that time. Other vendors (Silicon Graphics, Stellar, Ardent, Stardent) subsequently provided hardware-assisted capabilities for volume rendering. A notable software system of the 1980s that focused mainly on image processing was Khoros [30].

From the 1990s till now, there were additional major developments, which brought the *CAVA* tools from research labs to the clinic and which also brought along with them new challenges. The major developments include the tremendous

advances made in *CT* technology, the emergence of a variety of newer *MRI* modes, tremendous increase in the power of personal computers, computer aided diagnosis, and a variety of new applications of improved *CT* and *MRI*. Multi-slice helical *CT* [31] has substantially improved image quality especially because of the considerable increase in the speed of data acquisition. This has come to the point where, in the near future, volume image acquisition for an entire organ in a fraction of a second will become a reality. The challenges this has brought are the need to handle large image data rates and alternative and more practical methods of visualizing these data than the traditional radiological method of viewing the slices, which has already become impractical. The newer modes of *MRI* such as perfusion and diffusion *MRI* and *fMRI* [32] have brought new challenges for processing, displaying, and analyzing these images. The PCs have become so powerful and inexpensive that they have made expensive scientific workstations a luxury item and inessential in *CAVA* labs.

1.4. Current Software Systems and 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 *CAVASS* development. Our survey here considered most of the well-known software systems including Analyze [13], *DDV* (Digital Data Viewer) [33], *GIMP* [34], Image/*J* [35], *IDL* (Interactive Data Language) [36], *ITK* [6] (Insight Toolkit), Java [37], Khoros [30], Mathematica [38], Matlab [39], OpenDX (Open Data Explorer) [40], Photoshop [41], Volview [42], *VTK* (Visualization Toolkit) [43], *VXL* (Vision-something-Libraries) [44], and 3D Slicer [45]. Analyze, *IDL*, Khoros, Mathematica, Matlab, Photoshop, and Volview are excellent commercial software packages. They are not freely available nor are they 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 one 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 2D 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 (2D slices).

The *DDV* (Digital Data Viewer, free, open source, Windows only) software available was not afforded further consideration because it is primarily oriented toward *EEG* data and not 2D 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 was freely available, *DDV* uses *Qt,* which costs $2330 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 iso-surfaces 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 2D 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 2D images but can combine 2D images into "stacks" of slices. 3D 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*). Similarly to *VTK*, no user interface is provided, but, in contrast with *VTK*, it does contain a rich variety of algorithms for image processing (E1) that are specific to medical imaging. *CAVASS* we propose incorporates easy mechanisms to interface to *ITK*.

Our consideration now turns to the Java programming language (including the Java Advanced Imaging (*JAI*), Java2D, and Java3D *API*s). 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 *GCJ*) which compile Java to native machine code may make Java as efficient as C++ but *GCJ* is still in infancy as it does not yet support Swing (the Java *API* for building user interface which is responsible for drawing buttons, menus, windows, etc.). And similarly to Matlab, *IDL*, *VTK*, and *ITK*, Java 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 *X*11-based, open source application that is freely available for Unix platforms. OpenDX 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 3*DVIEWNIX* and *CAVASS*) that interface with *CAD/CAM* (Computer Aided Design/Computer Assisted Manufacturing) 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. Further, since *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 (*IUE*). It is primarily oriented towards the analysis of 2D surveillance satellite imagery with the goal of inferring 3D geometry. 3D Slicer is an open source, freely available application for the analysis and display of 3D medical imagery. It also includes basic automatic registration and semi-automatic segmentation capabilities. It is primarily designed to be used for surgical planning.

Other recently introduced systems include *MITK* [46], another system by the same name [47], and *IGSTK* [48]. The Medical Imaging Interaction Toolkit [46] and the Medical Imaging Toolkit [47] are similar and independent open-source systems that reuse and extend the capabilities of *VTK* [43] and *ITK* [6]. The Image-Guided Surgery Toolkit [48] 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 non-commercial software systems. Since 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 non-commercial 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 E1, E3, and E4. *ITK* on the other hand is very rich in E1 (image processing functions) but has no functions for E2, E3, and E4. Further it caters to only the user group UG1. *3DVIEWNIX* has its own limitations. Although it has a variety of functions under E1, *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 [26, 49], digital surface detection (in *n*-dimensions) [50], live-wire segmentation [51, 52], *MR* image intensity standardization [53], and various local scale-based processing strategies [54-58] are not in *ITK*. The visualization tools (E2) in *3DVIEWNIX* are as rich and efficient as in any (commercial/non-commercial) system. It has a rich collection of structure manipulation tools (E3) including the ability to cut, move, reflect, re-edit in three-dimensional space structures defined in both hard and fuzzy manner. In addition to the common intensity-based and geometry-based analysis tools (E4), it has advanced tools to analyze the morphology, architecture, and kinematics of object systems.

From our review and experience in developing *CAVA* software and using (and contributing to) *ITK*, we conclude that there are four main groups of limitations that exist in open source software for *CAVA* currently. *(LM1) Lack of comprehensiveness*: This essentially means not fulfilling requirement RQ5. There simply is no software (open-source or otherwise) currently that covers all elements E1-E4 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. *(LM2) Lack of ease of use*: This means inadequately fulfilling requirement RQ2 in terms of catering to the needs of all three user groups UG1-UG3. *(LM3) Lack of speed*: Both interactive and non-interactive operations fall short of the speed needed to make many *CAVA* applications practical, from future considerations as well as at present, especially when dealing with large data sets. *(LM4) Interfaceability*: This relates to inadequately fulfilling requirement RQ6. Our design goal for the system CAVASS presented in this paper was to try to fulfill RQ1-RQ7 as best as possible and overcome limitations *LM1-LM4* in the best possible way.

2. METHODS

A simplified architectural diagram of *CAVASS* appears in Figure 1. The software design aspects of *CAVASS* are described in detail in a companion paper in these proceedings [60]. The aspects of visualization are covered in another paper [59]. In the present paper, we focus mainly on the image processing aspects (E1) indicated by the first of the four *CAVA* functions in Figure 1.

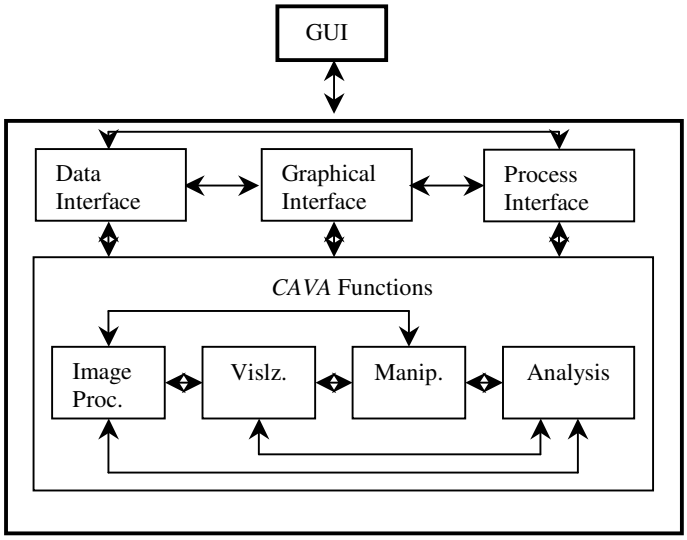

Figure 1: The architecture of *CAVASS*.

From the image processing perspective for *CAVA*, RQ4 and RQ5 become the most crucial requirements. Our approach in choosing and implementing image processing algorithms for *CAVASS* was guided by two principles: (i) not to go overboard with RQ5 but make most useful and commonly used algorithms available, (ii) give utmost importance to RQ4 (speed). The image processing operations included in *CAVASS* may be divided into the following seven groups. We will use $\mathbf{I} = (I, f)$ to denote an (*nD*) *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 $\mathbf{I}_i = (I_i, f_i)$ and $\mathbf{I}_o = (I_o, f_o)$ denote input and output images, respectively.

(1) Volume of Interest (*VOI*): 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, 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 $I_o = I_i$, $\mathbf{I}_o$ and $\mathbf{I}_i$ are both either grey or binary, and the intensities in $\mathbf{I}_o$ are modified

from those in $\mathbf{I}_i$. 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 I_{i1} and I_{i2} or surfaces S_{i1} and S_{i2} and produce in the respective cases an image $\mathbf{I}_o = T(\mathbf{I}_{i2})$ which matches with $\mathbf{I}_{i1}$ or a surface $S_o = T(S_{i2})$ that matches with S_{i1}, where *T* is a geometric transformation. *T* may be a rigid (6-parameter), affine (9-12 parameter) or a deformation (100s to 1000s 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 etc.

Many of the E1 operations listed above under the 7 groups achieve adequate speed (< 1-2 minutes) in sequential implementation on a single *CPU* modern *PC* even for large image data sets. Examples of such operations are rectangular *VOI*, small neighborhood based high/low pass filtering and morphological operations, thresholding, iso-intensity surface generation, image subtraction/addition/multiplication, and simple structure operations. However, some of the most useful E1 operations are very time consuming even for medium sized images and structures on single *CPU* machines. For large data sets, these operations are quite prohibitive in time and/or in memory requirements. The most crucial among such *CAVA* operations under E1 are: interpolation, non-linear and iterative filtering methods (particularly those based on diffusion, distance transforms), many 3D segmentation methods, and intensity based registration methods.

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. 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 [57], 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. We will focus here on *Type-1* and *Type-3* operations since they relate to E1.

2.1 Parallelizing *Type-1 CAVA* Operations

Our idea is to divide the input image data into as many chunks as there are processors in the multiprocessing system (we will come back to a discussion of the system environment later) and assign each chunk to a processor for executing the operation. The chunk size is made to fit the memory available on each processor. For some of the *Type-1* operations, the chunks can be created by partitioning the set of slices in the input data so that there is no need for overlap among chunks. For other *Type-1* operations (wherein neighboring slices are required), the chunks have to be defined with an overlap with the neighboring chunk(s). There is no inter-processor communication for *Type-1* operations. We note that chunking for 4D and higher dimensional data is more challenging than for 3D data. In the 4D case (which is the highest dimensionality of image data of practical value encountered in medical imaging), each 3D image volume corresponding

to a given instance of the 4th dimension is divided into chunks as above. The method of processing for *Type-1* operations may be summarized as follows.

begin

Step 1: Divide the given image $\mathbf{I}_i$ 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 $\mathbf{I}_o$ or structure.

end

The effect of parallelization comes here 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 $\mathbf{I}_i$, 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 $\mathbf{I}_i$ that can be handled irrespective of the number of processors available.

2.2 Parallelizing *Type-3 CAVA* Operations

Solutions to *Type 3 CAVA* operations can be characterized by optimal graph traversal algorithms. Although there are slight variations in the algorithmic expression [61] among seemingly vastly different methods such as graph cut, fuzzy connectedness, level-set, watershed, distance transforms, and iso-surface tracking, their algorithms have a similar character and behavior from the viewpoint of parallelized implementations.

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 $\mathbf{I}_i^j$ of the input image $\mathbf{I}_i$. In the algorithm, π is a predicate whose exact form depends on the particular *Type-3* operation we are dealing with.

begin

Step 1: Divide the given image $\mathbf{I}_i$ into chunks $\mathbf{I}_i^j$, $j = 1,....,N$.

Step 2: Initialization. A set of voxels are 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 $\mathbf{I}_i^j$ 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 $\mathbf{I}_o$ 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 inter-processor 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.

2.3 Multiprocessing Environment

At present, there are two choices available for parallel implementation – either in multiprocessor systems (*MPS*) via multi-threading or via distributed processing in a cluster of workstations (*COWs*). After extensive experiments with several *Type-1*, *Type-2*, and *Type-3* operations in *MPS* and in a *COW*, we have determined that, for *CAVA*, *COWs* offer significantly higher speed/dollar than *MPS*. In the following section, therefore, we report the results on a *COW*, except when results are compared with ITK, we used a dual processor system (3.4 *GHz*, 4*GB RAM*). Each workstation in the *COW* we constructed is a 3.4 *GHz* single *CPU* Pentium *PC* with 4*GB RAM*. The workstations are networked by a 1*G-bit*/sec connection.

3. RESULTS AND DISCUSSION

All E1 *CAVA* operations implemented in *CAVASS*, sequential and parallel, are tested with the following three regular, large, and super data sets. (*DS1*): a 3D *MR* image of the brain, size 256×256×46 (6.03 *MB*); (*DS2*): a 3D *CT* image of the torso, size 512×512×469 (240.65 *MB*); (*DS3*): a 3D *CT* image of the visible woman head, size 1024×1024×417 (872.8 *MB*). The regular data set *DS1* represents a typical *MRI* clinical study. *DS2* represents a typical, large clinical *CT* study of the thorax. *DS3* is a very large data set, artificially created by in-slice interpolation of the visible woman *CT* data set [62], employed to really push the algorithms to their limit.

We did extensive comparisons on a variety of E1 *CAVA* operations (*Type-1* and *Type-3*) in both sequential and parallel modes in various configurations of the *COW*. 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. Since *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* workstations. Our results are summarized in Table 1.

Operation	**System**	**Regular (*DS*1)**		**Large (*DS*2)**		**Super (*DS*3)**	
		seq	**parallel**	**seq**	**parallel**	**seq**	**parallel**
Interpolation	*ITK*	2.9	1.7	87.7	62.8 [2]	Failed	Failed
	CAVASS	0.6	1 [2]	54.9	14.9 [2]	139.1	49.2
Anisotropic Diffusive Filtering	*ITK*	57		2026.6		Failed	
	CAVASS	52.7		1664.2			
Gaussian Filtering	*ITK*	1.5		65.2		Failed	
	CAVASS	0.4		18.3		83	
Distance Transform	*ITK*	10.5		473.7		Failed	
	CAVASS	18.7		916.5		3882.4	
Thresholding	*ITK*	0.3		11.4		340.6	
	CAVASS	0.1		2.7		20.2	
Fuzzy Connected Segmentation	*ITK*	108.4		Failed		Failed	
	CAVASS	49.5	17.8	843.7	298.6 [5]	Failed	1312.6 [5]
Registration (rigid)	*ITK*	57.2		Failed		Failed	
	CAVASS	56.1	8.6 [5]	1860.6	301.6 [5]	3863.4	1089.1 [5]
Registration (affine – 12 parameters)	*ITK*	208.3		Failed		Failed	
	CAVASS	155.3	25.1	3602.4	1018.6 [5]	13,111	3662.2 [5]

Table 1: 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.

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 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 3*DVIEWNIX*) 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 1 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 house keeping 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 $t_s / \left(t_p n_p \right)$ 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 (e.g, fuzzy connectedness). Among the operations listed in Table 1, registration is the most time consuming. In these operations, normalized mutual information was used to register the two images [63]. 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 minutes. Parallelized deformable registration is currently being implemented in *CAVASS*.

In conclusion, we found, for *CAVA* operations, *COWs* are far more cost- and speed-effective than *MPS*. The recognition of the three types of *CAVA* operations helps in systematizing the parallelization efforts. This also enabled *CAVASS* to handle extremely large data sets. The implementations of the same *CAVA* image processing operations in *CAVASS* offer considerably greater speed than in *ITK*. *CAVASS* offers easy interface to a number of other software systems including *ITK*. Since it provides an easy to use GUI as in its predecessor 3*DVIEWNIX*, *CAVASS* can be readily used in applications. It is designed to serve all user groups (UG1-UG3) except clinical end users.

ACKNOWLEDGEMENT

The research reported in this paper is supported by 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, 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 (112 pages).

[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 (205 pages).

[4] L.S. Chen, G.T. Herman, C.R. Meyer, R.A. Reynolds, 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, C.R. Meyer: "3D98: A turnkey system for the display and analysis of 3D medical objects, SPIE Proceedings, **626**:474-482, 1986.

[6] L. Ibanez, W. Schroeder, L. Ng, and J. Cates: *The ITK Software Guide*, Kitware Inc., 2003.

[7] E. Artzy and G.T. Herman: "Boundary detection in three dimensions with a medical application," Computer Graphics **15**:92-123, 1981.

[8] G. Herman and H. Liu: "Three-dimensional display of human organs from computed tomograms," Computer Graphics and Image Processing **9**:679-698, 1979.

[9] J.K. Udupa: "Determination of 3-D shape parameters from boundary information," Computer Graphics and Image Processing **17**:52-59, 1981.

[10] J.K. Udupa: "Interactive segmentation and boundary surface formation for 3-D digital images," Computer Graphics and Image Processing, **18**:213-235, 1982.

[11] Movie *BYU*: *A general-purpose computer graphics system*, Department of Civil Engineering, Brigham Young University, (Provo, UT: Brigham Young University, 1981).

[12] H. Fuchs, Z.M. Kedem, and S.P. Uselton: "Optimal surface reconstruction for planar contours," Communication of ACM **20**:693-702, 1977.
[13] R.A. Robb, D.P. Hanson, R.A. Karwoski, A.G. Larson, E.L. Workman, and M.C. Stacy: "*ANALYZE*: A comprehensive, operator-interactive software package for multidimensional medical image display and analysis," Computerized Med. Imag. Graph. **13**:433-454, 1989.
[14] P. Dev, L.L. Fellingham, C. Lau, and J. Vogel: "Interactive graphics and 3-D modelling for surgical planning and prosthesis and omplant design," *NCGA*:132-142, 1986.
[15] Dimensional Medicine Co.: *RSNA* Scientific Program, p. 409, 1985.
[16] D.J. Meagher: "Interactive solids processing for medical analysis and planning," NCGA **2**:96-106, 1984.
[17] E.J. Farrel, W.C. Yang, and R.A. Zapulla: "Animated 3D CT imaging," IEEE Computer Graphics and Applications **5**:26-32, 1985.
[18] M.L. Rhodes: "An algorithmic approach to controlling search in three-dimensional image data," Proceedings of SIGGRAPH:*134-142*, 1979.
[19] Virtual Imaging Co.: *RSNA* Scientific Program, p. 435, 1986.
[20] *ISG* Technology Inc.: *RSNA* Scientific Program, p. 449, 1987.
[21] M.W. Vannier, J.L. Marsh, and J.O Warren: "Three-dimensional computer graphics for craniofacial surgical planning and evaluations," Computer Graphics **17**:263-274, 1983.
[22] K.H. Hohne and R. Bernstein: "Shading 3d images from CT using gray-level gradients," IEEE Trans. Med. Imag. **5**:45-47, 1986.
[23] D. Pham, C. Xu, and J. Prince: "Current methods in medical image segmentation," Annual Review Biomedical Engineering **2**:315-337, 2000.
[24] G. Gerig, O. Kubler, R. Kikinis, and F.A. Jolesz: "Nonlinear anisotropic filtering of MRI data," IEEE Trans. Medical Imaging **11(2)**:221-232, 1992.
[25] L. Axel, J. Costantini, and J. Listerud: "Intensity correction in surface coil MR imaging," American Journal Roentgenology **148**:418-420, 1987.
[26] S.P. Raya, and J.K. Udupa: "Shape-based interpolation of multidimensional objects," IEEE Transactions on Medical Imaging **9(1)**:32-42, 1990.
[27] C.A. Pelizzari, G.T.Y. Chen, D.R. Spelbring, R.R. Weichselbaum, and C.T. Chen: "Accurate three-dimensional registration of *CT*, *PET,* and *MR* images of the brain," J. Comput. Asst. Tomogr. **13**:20-26, 1989.
[28] R.A. Drebin, L. Carpenter, and P. Hanrahan: "Volume rendering," Comput. Graph. **22**:65-74, 1988.
[29] M. Levoy: "Display of surfaces from volume data," IEEE Comput. Graph. Appl. **8**:29-37, 1988.
[30] www.khoral.com
[31] T.O.J. Fuchs, M. Kachelriess, and W.A. Kalender: "Fast volume scanning approaches by X-ray-computed tomography," Proc. IEEE **91(10)**:1492-1502, 2003.
[32] Z.H. Chu, J.Pl Jones, and M. Singh: *Foundations of Medical Imaging*, New York, New York:Wiley, 1993.
[33] www.compgeomco.com.
[34] www.gimp.org.
[35] rsb.info.nih.gov/ij/.
[36] www.rsinc.com.
[37] www.javasoft.com.
[38] www.wolfram.com.
[39] www.mathworks.com.
[40] www.opendx.org.
[41] www.adobe.com.
[42] www.kitware.com/products/volview.html.
[43] www.vtk.org.
[44] vxl.sourceforge.net.
[45] www.slicer.org.
[46] I. Wolf, M. Vetter, I. Wegner, T. Böttger, M. Nolden, M. Schöbinger, M. Hastenteufel, T. Kunert, and H.-P. Meinzer: "The Medical Imaging Interaction Toolkit (MITK), Med. Imag Anal**. 9(6)**:594-604, 2005.
[47] M. Zhao, J. Tian, X. Zhu, J. Xue, Z. Cheng, and H. Zhao: "Design and implementation of a C++ toolkit for integrated medical image processing and analyzing," Proc. Of SPIE: Medical Imaging **5367**:39-47, 2004.
[48] K. Gary, L. Ibanez, S. Aylward, D. Gobbi, M.B. Blake and K. Cleary: "IGSTK: An open source software toolkit for image-guided surgery," Computer **39(4):**46-53, 2006.

[49] G.J. Grevera and J.K. Udupa: "Shape-based interpolation of multidimensional grey-level images," IEEE Transactions on Medical Imaging **15(6)**:881-892, 1996.
[50] J.K. Udupa: "Multidimensional digital boundaries," CVGIP: Graphical Models and Image Processing **50(4)**:311-323, 1994.
[51] A. Falcao, J.K. Udupa, S. Samarasekera, S. Sharma, B.E. Hirsch, and R. Lotufo: "User-steered image segmentation paradigms: Live wire and live lane," Graphical Models and Image Processing **60(4)**:233-260, 1998.
[52] A. Falcao, J. K. Udupa, and F. K. Miyazawa: "An ultra-fast user-steered image segmentation paradigm: Live-wire-on-the-fly," IEEE Transactions on Medical Imaging, **19(1)**:55-62, 2000.
[53] L.G. Nyul, J.K. Udupa, and X. Zhang: "New variants of a method of *MRI* scale standardization," IEEE Transactions on Medical Imaging **19(2)**:143-150, 2000.
[54] P.K. Saha, and J.K. Udupa: "Scale-based image filtering preserving boundary sharpness and fine structure," IEEE Transactions on Medical Imaging **20(11):**1140-1155, 2001.
[55] L.G. Nyul, J.K. Udupa, and P.K. Saha: "Incorporating a measure of local scale in voxel-based 3-D image registration," IEEE. Trans. Medical Imaging **22(2)**:228-237, 2003.
[56] A. Souza, J.K. Udupa and P.K. Saha: "Volume rendering in the presence of partial volume effects," IEEE Transactions on Medical Imaging **24(2)**:223-235, 2005.
[57] A. Madabhushi, J.K. Udupa, and A. Souza: "Generalized scale: Theory, algorithms, and application to image inhomogeneity correction," Computer Vision and Image Understanding **101**:100-121, 2006.
[58] A. Madabhushi and J.K. Udupa: "New methods of MR image intensity standardization via generalized scale, Medical Physics **33(9):**3426-3434, 2006.
[59] G. Grevera, J.K. Udupa, D. Odhner, Y. Zhuge, A. Souza, T. Iwanaga and S. Mishra: "Introducing CAVASS – A computer assisted visualization and analysis software system – Visualization aspects, Proc. of SPIE: Medical Imaging **6509**, 2007 (to appear).
[60] G. Grevera, J.K. Udupa, D. Odhner, Y. Zhuge, A. Souza, T. Iwanaga and S. Mishra: "Introducing CAVASS – A computer assisted visualization and analysis software system, Proc. of SPIE: Medical Imaging **6516**, 2007 (to appear).
[61] A.X. Falcao, J. Stolfi, and R. Lotufo: "The image foresting transform: Theory, algorithms, and applications," IEEE Pattern Analysis and Machine Intelligence **26(1)**:19-29, 2004.
[62] The Visible Human Project: National Library of Medicine, 1995. http://www.nlm.nih.gov/research/visible/visible_human.html.
[63] W.M. Wells III, P. Viola, H. Atsumi, S. Makajima, and R. Kikinis: "Multi-modal volume registration by maximization of mutual information," Medical Image Analysis **1(1)**:35-51, 1996**.**