The VAIDAK Medical Image Model Reconstruction Toolkit

Chandrajit L. Bajaj Malcolm C. Fields

Department of Computer Science, Purdue University

Keywords

Bio-Medical Computing, Image Processing, Geometric Modeling, Problem Solving Environments, Computational Science.

Abstract

The VAIDAK medical image model reconstruction toolkit manipulates medical image volume data and constructs accurate surface and solid models of skeletal and soft tissue structures. It takes CT (Computed Tomography), MRI (Magnetic Resonance Imaging), or Laser surface imaging data as input and incorporates both heuristic and exact methods for: contouring of image data, active thresholding, tiling, and polygon reconstruction. It also incorporates a scanner to view image data and interactively pick threshold values, a browser feature to modify the contours, an editor to edit boundary polygons of a reconstructed solid object, and a rendered display .

INTRODUCTION

VAIDAK is a X-11 windows and XS portable graphics library [3] based application that, given voxel data, such as that from CT (Computed Tomography), MRI (Magnetic Resonance Imaging), and laser imaging, produces a surface approximation of the scanned solid. The system produces this reconstruction by either computing serial cross sections of an object and reconstructing a surface of the object with respect to these contours, or by directly producing a surface approximation of the solid object. In both cases the approximated surface consists of triangular surface patches. The VAIDAK system makes contributions to scientific visualization, specifically medical imaging, as well as solid modeling. VAIDAK is part of a distributed geometric software environment called SHASTRA consisting of a growing number of individually powerful and interoperable (client–server) toolkits which support collaborative design sessions[2].

The functionality of VAIDAK is provided by four separate subsystems. The first consists of a CT/MRI to contour conversion subsystem. This subsystem provides a convenient user interface and an edge detection algorithm to compute polygonal contours that are implicitly present within the voxel data. The second subsystem consist of a contour browser and full function polygon editor. This subsystem provides the user a convenient mechanism for removing the errors that are present in the contours that are due to imaging noise. The third subsystem consist of a polyhedra browser and a three dimensional geometric editor. This subsystem provides the user a convenient mechanism for selecting and deleting extraneous polygons in three dimensions from a polygonal approximation of the solid object. The fourth subsystem consists of the surface and model reconstruction routines. Varying algorithms employing different surface metrics can be chosen by the user [9, 8, 5]. The whole system is controlled by a multi-window graphical user interface system that provides convenient methods for controlling the four subsystems.

In section 2, a brief technical background is presented which provides an overview of previous research in this area. The technical details of the surface reconstruction algorithms as well as surface fitting techniques are presented. In section 3 both the software architecture and the choice and rational of the algorithms and data structures used by VAIDAK are presented. This section also highlights the new contributions made in surface and model reconstruction from CT/MRI data. Section 4 outlines the user interface highlighting the functionality that is provided by VAIDAK. Section 5 examines the system internals. Finally, section 6 concludes this work and discusses future extensions.

TECHNICAL BACKGROUND

Skeletal model reconstruction from voxel data has been an active research area for many years. Schumaker [18] presents an excellent survey of the wide range of techniques that have been utilized. The majority of the reconstruction techniques produce planar C^0 approximations of the data set. Relatively little has been achieved in constructing smooth skeletal mod-

ACM-SAC '93/2/93/IN, USA
© 1993 ACM 0-89791-568-2/93/0002/0028...$1.50

els using curved surface patches.

There are primarily two classes of surface reconstruction techniques. One class of methods first constructs planar contours in each CT/MRI data slice and then connects these contours by a triangulation in three dimensional space. The triangulation process is complicated by the occurrence of multiple contours on a data slice (i.e. branching). Early contributions here are by Keppel [10], Fuchs, Kedem and Uselton [9], Christiansen and Sederberg [8] and Boissonnat[6].

The optimal algorithms due to Keppel [10] and Fuchs et. al [9] work by computing a graph in which each node represents a spanning arc and each edge in the graph represents a triangle defined uniquely by two spanning arcs that share a point. A shortest path algorithm is used to find the path that corresponds to the triangulation of minimum weight. Another algorithm, of a heuristic nature, due to Christiansen and Sederberg, performs a greedy walk around each adjacent pair of contours selecting segments. A segment on one contour and a point on the other define the triangle. The choice of the new segment is determined by the triangle with a shorter edge length. This algorithm has been implemented in VAIDAK.

The algorithm of Boissonnat computes a 2D Delaunay triangulation [17] of the planar contours and then a 3D Delaunay triangulation of the entire collection of triangles lying in parallel planes. Instead of planar contours one may compute a C^1 continuous piecewise curve approximation in each of the data slices. Examples of such techniques are [16] using conic splines and [15] using parametric B-splines. The stack of contours are then interpolated or least square approximated using piecewise tensor splines [12, 13] or nontensor piecewise smooth surfaces [7].

The other class of methods uses a hierarchical subdivision of the voxel space to localize the triangular approximation to small cubes [11]. This method takes care of branching, however the local planar approximation based on the density values at the corner of the subcube may sometimes be ambiguous. An extension of this scheme which computes a C^1 piecewise quadratic approximation to the data within subcubes is given in [14]

In [5] two algorithms for constructing C^1-smooth models of skeletal structures from CT/MRI voxel data is presented. The boundary of the reconstructed models consist of a C^1-continuous mesh of triangular algebraic surface patches. For definitions and basic manipulations of algebraic surfaces, the reader is referred to [4].

The first algorithm starts by constructing C^1-continuous piecewise conic contours on each of the CT/MRI data slices and then uses piecewise triangular algebraic surface patches, of degree at most seven, to C^1 interpolate the contours on adjacent slices. The other algorithm works directly in voxel space and replaces an initial C^0 triangular facet approximation of the model with a highly compressed C^1-continuous mesh of triangular algebraic surface patches of degree at most seven. Both schemes are adaptive, yielding a higher density of patches in regions of higher curvature. Both algorithms have been implemented in VAIDAK and the other geometry toolkits of the SHASTRA system. See also Figure 1 where geometric models of the human anatomy have been reconstructed from MRI voxel data.

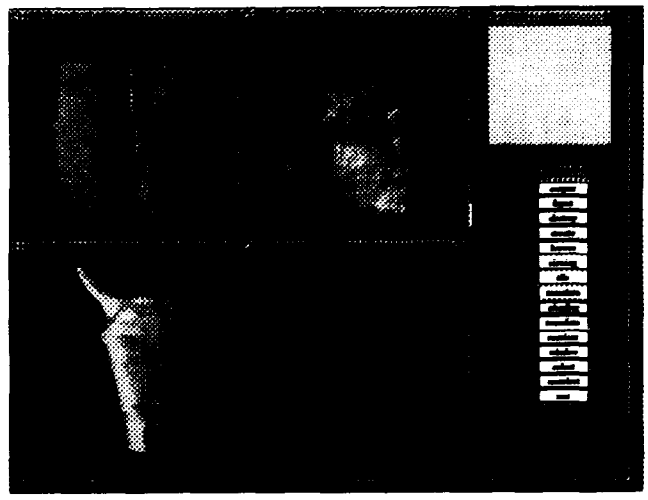

Figure 1: Human Anatomy Models Reconstructed using VAIDAK

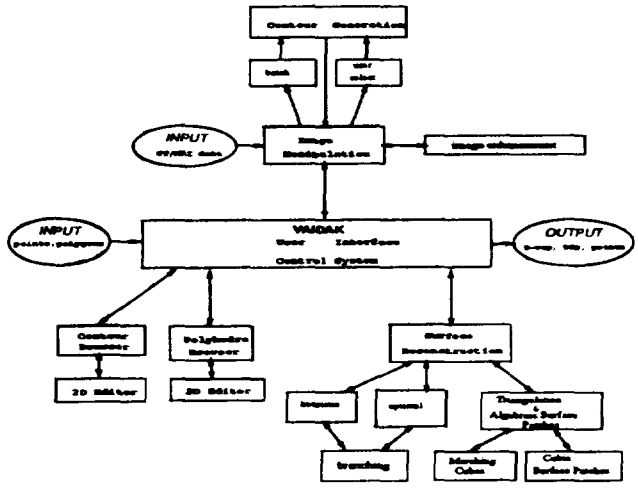

Figure 2: The VAIDAK Software Architecture

FUNCTIONALITY OF VAIDAK

Overview

VAIDAK provides a testbed for experimenting, using various metrics, with some of the previously described surface reconstruction methods, as well as providing a testbed for new methods. Figure 2 depicts the overall system architecture of the VAIDAK system. The functionality of VAIDAK is summerized as follows.

The density value of particular type of material varies with the calibration of the scanning machine. VAIDAK provides a voxel browser that will allow the user to browse through the voxel data, testing the voxel values of recognized material. In this way, a good approximation to the appropriate threshold

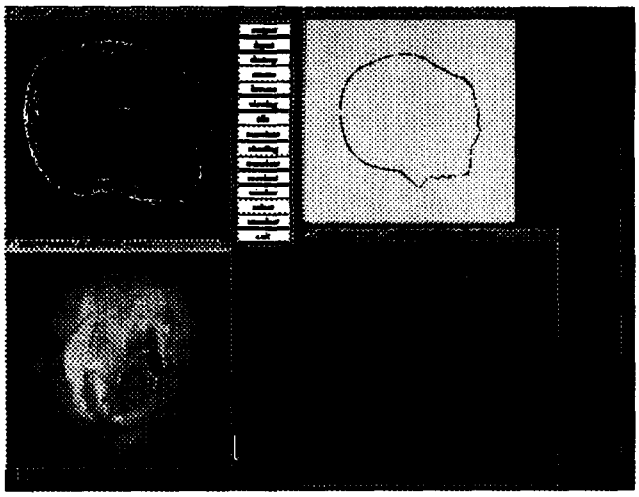

Figure 3: A Human Head reconstructed in VAIDAK

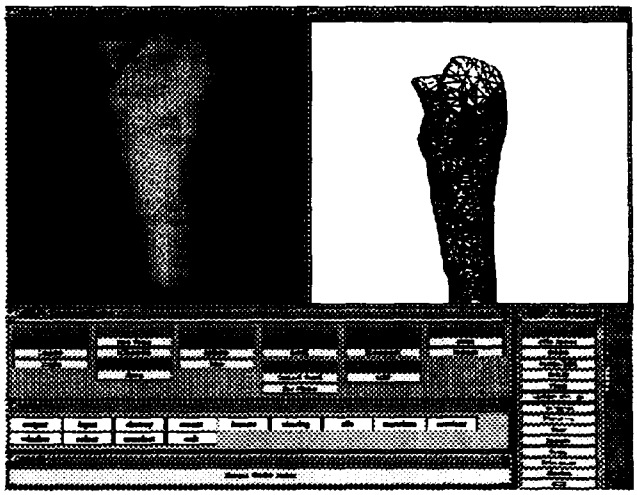

Figure 4: Part of a femur reconstructed in VAIDAK and communicated to SHILP

values can be deduced.

In addition, VAIDAK provides a contour generation algorithm which, given a threshold range and possible user interaction, computes polygonal contours from the voxel data. Currently, contour generation is being studied from an image processing point of view.

VAIDAK also provides a model construction and modification interface that is custom tailored for data in the form of numerous polygonal contours. Many problems existed previously in the design of a user interface that would give the user the ability to conveniently create, modify, and destroy contours in the presence of a possibly large number of other contours. This feature, for example, was used in the reconstruction of a human head of Figure 3 from dense MRI voxel data.

VAIDAK also provides functionality to model objects which have multiple contours per slice. This is done in a general way by allowing the user to define sub-objects. Sub-objects are defined in such a way that only one contour per slice is present. Therefore, each sub-object can be tiled as usual.

CT and MRI scanners have very limited travel capacities and therefore large studies have to be performed in stages. The specimen has to be moved with respect to the table after each stage. Therefore some method for adjusting the relative positions of the stages has to be supplied for the system to be useful. VAIDAK handles this by allowing multiple objects to be viewed in the same window and by supplying a utility for translating the relative position of objects.

Also, two stages of a large study seldom align perfectly. A small number of contours of two adjacent stages usually overlap. The overlapping contours differ in shape. VAIDAK provides merging operator which "averages" corresponding contours to produce a smooth transition.

VAIDAK provides a powerful testbed for experimentation with surface fitting algorithms. Computing algebraic surface patches which approximate the original surface compresses the size of the data storage requirements and provides a continuous surface.

Perhaps the most important contribution of VAIDAK is in its practical application to education and radiology. Veterinary schools are very interested in providing ways for their students to understand the interior structure of animals without destroying the animal in the process. VAIDAK provides just such a tool. Also, orthopedics and radiologists have found VAIDAK to be a very powerful tool for viewing the results of their studies.

To conclude, VAIDAK provides researchers with a testbed to experiment with existing and new surface reconstruction methods. Additionally, it represents a full functional scientific visualization tool that is of practical value to medical researchers and practitioners. Finally, because VAIDAK is part of the SHASTRA project, it provides a full function distributed system for computer aided design and scientific visualization[2]. The models generated by VAIDAK can be passed to the SHILP solid modeling system [1], via the SHASTRA protocol, for further manipulations in the pursuit of a accurate model of biological tissues. See Figure 4.

THE USER INTERFACE

Overview

In this section the VAIDAK user interface is briefly discussed. This includes a discussion of the look and feel of VAIDAK along with a general discussion of the command interface.

VAIDAK is a fully developed windowed system based on X. At startup, three windows are presented: the *command console*, the *main* window, and initially one *render* window. The main window is used by the browsing and editing subsystems.

The render window is where the rendered image is displayed. At startup, one render window is created. The user can create any number of render windows during the operation of VAIDAK. Multiple objects are maintained by VAIDAK and any or all of these objects can be referenced by any render

window. All render windows can be moved and resized at any time but will remain square.

Most user interaction will be with the buttons in the command console. All the subsystems are initiated from the command console. Each command provided in this window will be discussed in detail later.

Brief example

At this time, a quick example will be given. Once the windows are in place the user can begin driving the system by using the command console. One of the buttons in the command console window begins the file input process. When the *input* button is pressed, a popup menu is presented allowing the user to choose the desired file format. For example, if the user releases the button on the *points* entry then the system will assume a point data file is being input. A thorough discussion of file types and formats is presented later.

Popup dialogue windows are used for all textual interaction. In the case of file input, a dialogue is presented prompting the user for a path name and an object name. Each object managed by VAIDAK has a unique object name. If none is given the file name prefix will be used. After entering an object name and signaling the system that keyboard input is complete, by the use of a command button located within the dialogue, the system looks to the file system for the specified file.

If found, this file will be read and an object will be created with the specified name. On input, the render window that is labeled *CURRENT* will be given a reference to the newly created object. The method for manipulating these references, via the *select* and the *unselect* commands, will be discussed later. As a hint to the user, the title block of each render window contains a list of objects that are referenced by that window.

Objects displayed in a render window can be interactively rotated, transformed, and scaled by use of the mouse. In this way the user can interactively modify his or her perspective on the object.

As described earlier, VAIDAK allows the user to control the surface reconstruction methods and parameters that are used. Another command button is available in the command console to tile or reconstruct the surface of the object. Like many of command buttons, when pressed the *tile* command presents the user with a popup list of objects. The user must choose the object that is to be operated on. When the object is chosen, as before, a pop-up dialogue is presented. The user can control the method and metric used by setting the toggle buttons in this dialogue. By default, the heuristic method for computing a surface is used. As the reconstructed surface is being computed, the object is progressively displayed in the render window. See Figure 5.

A popup menu is available in the render window for changing the graphical surface characteristics of the objects referenced by the window. The *contour* command displays the original cross sections without any hint of the reconstructed surface. The *wire frame* command displays a wire frame view of the objects. The *mesh* command produces a lighted view of the object in which no smoothing has been done, i.e., the triangular facets are easily seen. The *lighted* command produces a smoothed and shaded image of the object in which the surface normals have been averaged. The surface can be reoriented with the *inside out* command. This essentially turns the surface of all the objects in the window inside-out telling the system it has confused the object's inside with the object's outside.

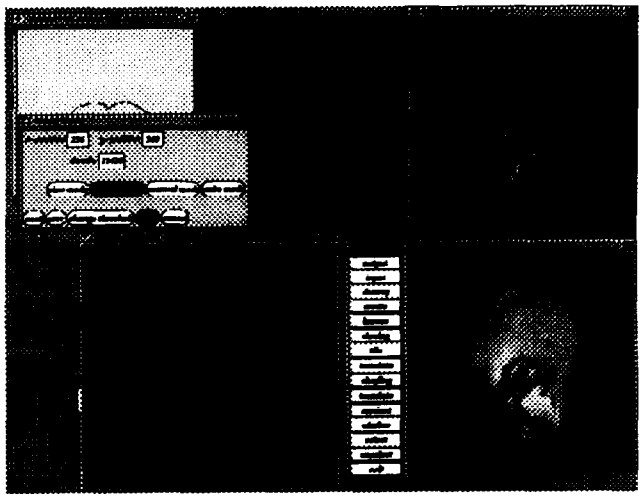

Figure 5: Surface and Solid Reconstruction from MRI Using Vaidak

This concludes the introductory example. The *quit* command terminates the execution of VAIDAK. Refer to the following subsections for a more complete explanation of the VAIDAK user interface.

The Command Console Window

The commands controllable from the command console window are examined in this section.

The *output* command raises a popup menu listing the object names of the currently managed objects. Selecting one of these objects directs the system to prepare to output data for the selected object. A dialogue is raised which allows the user to enter a file name. This dialog also allows the user to specify the file format.

The *input* command has three subcommands which allow the user to input data from different file types. The exact format of these files will be detailed in a later section. In any case, an object is added to VAIDAK's object list and a reference is added to this new object from the *CURRENT* render window.

The *destroy* command allows users to delete objects.

The *create* command presents the user with a list of object names. The list is prefixed by the special designator *NEW*. If the user selects an existing object name then the newly created contour will be added to the selected object upon completion of the editing. If *NEW* is selected, then a new object will be created and a reference to it will be added to the *CURRENT* render window. In this way, contours can be added to existing

objects, or entirely new objects can be built. This commands invokes a full function polygon cross section editor which can also be invoked through the browsing subsystem. If the object was created via the marching cubes algorithm, then this command does not apply.

The *browse* commands invokes a subsystem which allows the user to browse through the cross sections invoking additional commands on the chosen cross section. The browse subsystem will be examined in detail in a later in this section. This command also does not apply to marching cubes objects.

The *viewing* command raises a dialogue window which lists the modifiable render parameters for the *CURRENT* render window. Note that some of these parameters are also modified by mouse motion in the corresponding render window.

The *tile* command presents a popup menu and invokes the surface reconstruction subsystem on the selected object. Radio type buttons are used to allow the user to tailor the method used in tiling. The user can choose to optimize area or length when the optimal contour method is employed. If the *map coordinates* button is *ON* then, when length is optimized or the heuristic method is used, the coordinates will be mapped to a unit square prior to computation much like suggested in [8]. This command can be canceled at any time prior to the start of the contouring computation by pressing the cancel button.

The *tesselate* command initiates the marching cubes algorithm on a voxel data set. The marching cubes algorithm uses a vastly different approach for contour reconstruction. The reader is refered to [11] for a more thorough discussion of the technique. Solids generated by the marching cubes method are represented internally different from those based on contours.

The *windows* command presents the user with a list of window numbers; the list is prefixed by the special designator *NEW*. Each render window is designated in its title by a unique integer generated by the system. The title of each render window displays this integer. If an existing window number is selected from the popup list then that window will be made *CURRENT*. (To *select* or *unselect* objects into a window that window must be made current.) If the *NEW* designator is chosen, then a new render window will be created. This window will initially reference no objects. This new window will be made the *CURRENT* window.

The *translate* command is used to modify the position of an object in real world space. Also, the distance of separation between contours can be adjusted using this command. When multi-stage studies are performed, this command allows the user to adjust the relative positions of the stages in order to create one contiguous reconstructed object.

The *reorient* command allows the user to reorient objects, i.e., turn objects inside out. This command is needed if the user wishes to view the inside of the object or if the object is oriented incorrectly at the time it is created.

VAIDAK maintains one object list from which objects can be added or deleted. Each window can reference any or all of these objects, and one object can be referenced multiple times. When an object is brought into the system, either by input from the file system, creation by the editor, or computed from voxel data, a reference to it is added from the *CURRENT* render window. Using the *select* command, any window can reference any object. But note that if window n wants to reference an object it must be made current first using the *windows* command. It is important to note that when an object is modified, these changes will be reflected in every window that references the object. When a marching cubes object is selected, a window will appear and the selected object will be displayed.

The *unselect* command is the inverse of the select command. Likewise, to unselect an object from a window, that window must be *CURRENT*.

The *exit* command causes the system to exit. A positive response is required.

Render window commands and interaction

The render window also has a number of commands. These commands are activated via a popup menu. A description of the render commands follows:

The *contours* command sets the display mode to *contours* in which only the cross sections are displayed. All objects in the window are displayed in this way.

The *wireframe* command sets the display mode to *wire frame*. If a surface has been computed for a particular object since the last input or edit, then the wire frame of the computed tiling is displayed for that object in that window. Note that objects that have not been tiled are still displayed as contours only.

The *mesh* command sets the display mode to *mesh*. In mesh mode the lighting model is turned on and all triangular facets are filled and shaded. No averaging of normals is performed. Again if no tiling has been computed for the object, only the contours of that object will be displayed.

The *lighted* command display mode is set to *lighted* in which a smoothed and shaded image is displayed.

The *inside out* command reverses the outward normals of all objects referenced in the window. This is considered an active command because it effects the actual object, not just the reference to it. When this command is performed, if an object is referenced by two or more windows, then its image will change in every window that it is referenced.

The contour editor

The VAIDAK system comes with a full function cross section editor. The editor can be started by the *create* button on the command console or directly from the browser. It can be used to create new contours or edit existing contours. New contours can be part of an existing object or can become a new object. This section examines the commands and functionality of the editor.

The *add* command allows users to add points to the end of the list of points which make up the cross section. Because the cross section will always be implicitly closed by the system

no method is provided to add points to the beginning of the list. The *move* command allows the user to modify the geometry of the points of the contour. The *insert* command allows the user to insert points into cross sections. The *delete* command allows users to delete points of a contour.

The *edit z* command differs from the previous commands in that it does not perform a mode switch. This command allows the user to edit the z value associated with the contour. The current z value is always displayed in the main window title during editing and browsing.

The *close* command registers all the changes made to the cross section and exits the editor. The changes do not take effect until the *close* command is executed. If two or more segments in the cross section cross then the *close* command will abort with an error message. The *exit* command exits the editor disregarding all edits. This command requires a positive response to complete.

The Browser

The browse subsystem provides facilities for the user to interactively browse through the contour list and invoke commands on the selected contour. The user interacts with the browser in the main window only. The browser maintains a *current cross section*. This current cross section is highlighted in red in each render window that references the object. Pressing the left and middle mouse buttons move the current pointer backward and forward in the contour list respectively.

The browser commands are available via a popup menu. The description of these commands follow: The *first* command makes the first cross section in the list the current cross section. The *last* command makes the last cross section in the list the current cross section. The *edit* command invokes the cross section editor on the current cross section. Refer to the description of the edit subsystem commands contained in this paper. The *delete* command removes the current cross section from the list permanently. The *copy* command creates a exact copy of the current cross section and inserts it into the list of cross sections. Most often the user will then adjust the z value of the points on one of these identical contours with an editor command. The *method* command allows the user to *label* the current cross section as *heuristic*, *optimal*, or *default*. When surface reconstruction is performed in default mode the default method, via Christiansen et. al, is used unless the cross section is labeled heuristic or optimal. Refer to the *contour* command for further explanation. The *exit* command causes the browse subsystem to be exited.

The voxel browser and contour detector

VIADAK is equipped with an interactive voxel data browser and and contour detector. When the user directs the system to read voxel data, an auxiliary window is raised, along with the voxel data window, from which the user can control the contour searching process. From this auxiliary window, the voxel browser and contour detector are controlled.

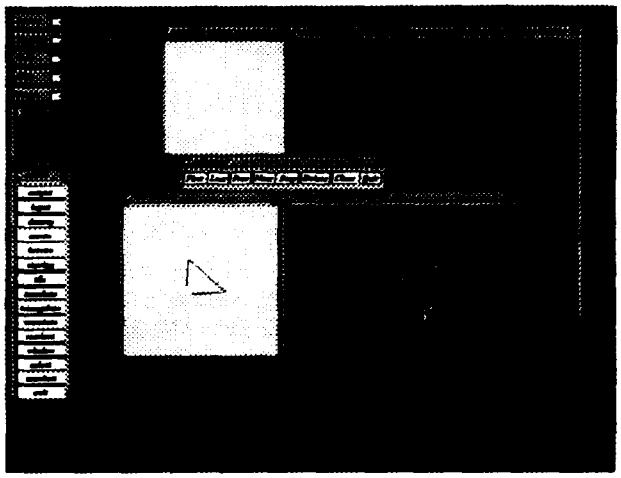

Figure 6: Editing and Browsing using the 3D Object Editor

Using the mouse, the user can interrogate the magnitude of any given voxel. In this way, the the appropriate threshold for contour searching can be deduced.

The contour detector can be driven interactively. Using the mouse, the expert user can provide the contour detection algorithm with a starting position. From this starting position, the algorithm constructs a connected contour by moving from voxel to voxel in the two-dimensional image.

Additionally, the contour can be sketched interactively directly on top of the image. In this case, the voxel image acts only as a reference. This mode is only used when the noise level in the image is so high that the contour searching algorithm cannot perform satisfactorily.

The 3D object editor and browser

When a marching cubes window is displayed, the editor can be invoked by clicking the Edit button described above. Only one editor is allowed at a time, thus if a new object is to be edited then any other marching cubes editor must be closed. All triangles displayed in the editor window, will be outlined in red in the object window. See Figure 6 for a displayed example.

The editor has the following functions:

First - Displays the first triangle

Last - Displays the last triangle

Prev - Displays the previous triangle from the current one.

Next - Displays the next triangle after the current one.

Step - Allows the user to skip triangles when the Prev or Next button is hit. So a step value of two has the following sequence: 1, 3, 5, 7, 9...

Delete - Deletes the current triangle.

Close - Quits the editor and saves all changes.

Quit - Quits the editor without saving any changes.

SOFTWARE IMPLEMENTATION ISSUES

In this section some implementation issues involving design and data representation are discussed. This section is partitioned into two subsections: graphics and file formats.

Graphics

VAIDAK is based on the X Athena widgets and on the XS platform independent graphics library. All dialogues and menus are implemented via the Athena widget toolkit. The Athena widget set was chosen for portability and functionality.

All graphics calls are to the XS graphics library. XS is a platform independent graphics library. XS provides a common library interface upon which graphics software can be built. Porting software from one platform to the next requires only a recompilation and linking with the appropriate version of XS library. Currently, an X windows and Silicon Graphics version of XS is available. A HP version is being implemented.

XS provides high level graphics calls modeled, in part, on the GL graphics library created by Silicon Graphics. One common interface for event handling is supplied regardless of the platform. More information on XS is available in [3].

File Formats

To interface effectively to VAIDAK, a thorough understanding of the various file formats is necessary. In this section these file formats will be examined.

VAIDAK provides input facilities for three different file types. Starting with the simplest, *point* files are text files containing the contours of the object only. The file is organized as a list of points organized into contours. The first line in the file contains an integer n describing the number of points in the first contour. The next n lines have three tab separated floating point numbers per line. This sequence of a count followed by the coordinates is repeated for each contour in the file.

Polygon files have a much more complicated format. Polygon files are text files that contain the explicit contours of the object, an implicit representation of the triangulated surface, and an explicit list of polygons.

The first line in the file contains a count m of the number of contours making up the object. For each of these m contours there is a count n of the number of points on the contour followed by n lines containing the tab separated floating point components for each point.

Following this, a polygon file will have an implicit representation of the triangulated surface of the object. As mentioned earlier, because the triangulated surface can be organized into *rings* spanning between adjacent contours, the triangular surface can to represented by a list of 3D points. The translation from point lists to contours is easy if one recalls that all the points on a contour share the same z-component, and therefore the contour to which a point belongs is easily computed.

The triangulated surface of an object is described by a number r of triangular rings, followed by r sets of: a line count n, followed by n lines, each containing the components of one point. Note that all adjacency information for these polygons can be computed without problems with the inaccuracies of floating point comparisons if the file data was derived from a manifold object in a deterministic manner.

The rest of a polygon file contains an explicit list of polygons. Each polygon is described by (1) the number n of vertices of the polygon, and (2) a list of n vertices and the associated normals at these vertices. This pattern of the number of vertices followed by the coordinates and normals of the vertices, is repeated until the end of the file. Note that the adjacency information is lost. This information is included only for ease of use by other systems. For a marching cubes object the *Poly* output is only a list of polygons, where each polygon is as described above.

The format of the CT/MRI data is much more difficult to decipher. Each slice of data is contained in a separate file. The file is in a binary format and therefore can not decoded without prior knowledge of the format. The format is as follows: the first 2 kilobytes are header information which is unused at this time by VAIDAK. The next 1K bytes are typically referred to as the circle map. It consists of 512 two byte integers, one integer for each scan line. The integer specifies one half the number of data values that are associated with the corresponding scan line. The data is assumed centered along the 512 cell scan line. For example, many CT scanners provide a 512 diameter circle of data centered within the 512 by 512 density matrix. The circle map describes, for each of the 512 scan lines, how much of the data corresponds to that scan line.

CT/MRI data comes in many formats. The user who wishes to use data in a different format must write special conversion routines to convert this data to VAIDAK form. A few of these routines have already been created; refer to the authors of this system for more explanation.

CONCLUSIONS AND OPEN PROBLEMS

The VAIDAK medical imaging and model reconstruction toolkit produces surface and solid geometric models from voxel data. A full-function user interface is provided that yields a system that is useful to medical researchers and practitioners. Additionally, because of VAIDAK's open architecture and platform independent construction, the VAIDAK system provides researchers with a convenient platform for experimenting with new and existing surface reconstruction techniques.

However, work on VAIDAK has not been completed. One open problem deals with objects that have multiple contours per cross section. VAIDAK provides a simple mechanism for reconstructing a surface of an object like this, i.e., allowing the user to define sub-objects which have only one contour per cross-section. Although this method for handling multiple contours per slice is effective, methods to automatically

compute surfaces of objects with multiple contours per cross-section are currently under investigation. The problems that arise are very difficult and have yet to be solved with much generality. In fact, the problem is not solvable in it most general form because of the lack of available information.

Acknowledgement
This research was supported in part by NSF grants CCR 90-00028, DMS 91-01424, AFOSR contract 91-0276 and industrial funds from AT & T and Nippon Steel Company. We also acknowledge the programming assistance of Vinod Anupam, Brian Bailey and Eric Price of Purdue University.

References

[1] V. Anupam, C. Bajaj, T. Dey, and I. Ihm. *The SHILP Solid Modeling and Display Toolkit*. Computer Science Technical Report, CAPO-91-29, Purdue University, 1991.

[2] V. Anupam, C. Bajaj, and A. Royappa. *The SHASTRA Distributed and Collaborative Geometric Design Environment*. Computer Science Technical Report, CAPO-91-38, Purdue University, 1991.

[3] V. Anupam, C.Bajaj, A. Burnett, M. Fields, A. Royappa, and D. Schikore. *XS: A Hardware Independent Graphics and Windows Library*. Computer Science Technical Report, CAPO-91-28, Purdue University, 1991.

[4] C. Bajaj. Geometric Modeling with Algebraic Surfaces. In D. Handscomb, editor, *The Mathematics of Surfaces III*, pages 3–48. Oxford Univ. Press, 1988.

[5] C. Bajaj. Electronic Skeletons: Modeling Skeletal Structures with Piecewise Algebraic Surfaces. In *Curves and Surfaces in Computer Vision and Graphics II*, pages 230–237, Boston, MA, 1991.

[6] J. Boissonnat. Shape Reconstruction from Planar Cross Sections. *Computer Vision, Graphics and Image Processing*, 44:1–29, 1988.

[7] L. Brevdo, S. Sideman, and R. Beyar. A Simple Approach to the Problem of 3D Reconstruction. *Computer Vision, Graphics and Image Processing*, 37:420–427, 1987.

[8] H. Christiansen and T. Sederberg. Conversion of complex contour line definitions into polygonal element mosaics. *Computer Graphics*, 13:187–192, 1978.

[9] H. Fuchs, Z. M. Kedem, and S. P. Uselton. Optimal Surface Reconstruction from Planar Contours. *Communications of the ACM*, 20:693–702, 1977.

[10] E. Keppel. Approximating Complex Surfaces by Triangulation of Contour Lines. *IBM J. Research and Development*, 19:2–11, 1975.

[11] W. Lorensen and H. Cline. Marching Cubes: A High Resolution 3D Surface Construction Algorithm. *Computer Graphics*, 21:163–169, 1987.

[12] R. Gmelig Meyling. Least Squares B-spline Surface Reconstruction in Tomography. *Manuscript*, 1984.

[13] R. Gmelig Meyling and P. Pfluger. B-spline Approximation of a Closed Surface. *IMA J. of Numerical Analysis*, 7:73–96, 1987.

[14] D. Moore and J. Warren. Approximation of dense scattered data using algebraic surfaces. In *Proc. of the 24th Hawaii Intl. Conference on System Sciences*, pages 681–690, Kauai, Hawaii, 1991.

[15] G. Nielson and T. Foley. Knot Selection for Parametric Spline Interpolation. In T. Lyche and L. Schumaker, editors, *Mathematical Methods in Computer Aided Geometric Design*, pages 261–271. Academic Press, 1989.

[16] V. Pratt. Techniques for Conic Splines. *Computer Graphics*, 19(3):151–159, 1985.

[17] F. Preparata and M. Shamos. *Computational Geometry, An Introduction*. Springer Verlag, 1985.

[18] L. Schumaker. Reconstructing 3D Objects from Cross Sections. In W. Dahmen, M. Gasca, and C. Michelli, editors, *Computation of Curves and Surfaces*, pages 275–309. Kluwer Academic Publishers, 1990.