The VAIDAK Medical Imaging and Model Reconstruction Toolkit

Brian Bailey
Chanderjit Bajaj
Malcolm C. Fields

Report Number: 91-066
THE VAIDAK MEDICAL IMAGING
AND MODEL RECONSTRUCTION TOOLKIT

Brian Bailey
Chanderjit Bajaj
Malcolm C. Fields

CSD-TR-91-066
August 1991
The VAIDAK medical imaging and model reconstruction toolkit*

Brian Bailey  Chanderjit Bajaj  Malcolm C. Fields
Department of Computer Science
Purdue University
West Lafayette, IN 47907

Abstract

The VAIDAK medical imaging and 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 for the boundary polygons of a reconstructed solid object and a render window to change lighting and display modes.

*Supported in part by NSF grant CCR 90-02228 and AFOSR contract 91-0276
CONTENTS

Contents

1 INTRODUCTION 3

2 TECHNICAL TOUR 3
  2.1 Surface Reconstruction Algorithms 3
  2.2 Functional Subsystems 5
  2.3 CT/MRI to Contour Conversion 7

3 User Interface and Functionality 10
  3.1 Start up 10
  3.2 Brief example 10
  3.3 The Command Console Window 12
  3.4 Render window commands and interaction 17
  3.5 The cross section editor 18
  3.6 The Browser 19
  3.7 The voxel browser and contour detector 20
  3.8 The 3D object editor and browser 21

4 Implementation Issues 22
  4.1 Graphics 22
  4.2 Data Structures 22
  4.3 File Formats 23

5 FUTURE EXTENSIONS AND PLANS 24
1 INTRODUCTION

VAIDAK is a X-11\textsuperscript{1} and XS\textsuperscript{2} based application toolkit that, given slices of voxel data such as from CT, NMR, LASER imaging, either computes serial cross sections of an object and reconstructs a surface of the object with respect to these contours or directly produces a surface approximation of the solid object. In both cases the approximated surface consists of triangular surface patches. The VAIDAK project makes contributions to scientific visualization, specifically medical imaging, as well as solid modeling.

VAIDAK consists of four 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 editing the voxel and edge detection noise that is present in the contours.

The third subsystem consist of a polyhedra browser and a 3D 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 controled by a multi-window graphical user interface system that provides convenient methods for controlling the three subsystems. VAIDAK as a whole is a subsystem of SHASTRA and will be integrated into the SHASTRA distributed front end.

The rest of this paper consists of the following. Section 2 presents the technical details of both the software architecture, and the choice and rational of the algorithms and data structures implemented. It also highlights the new contributions made in surface and model reconstruction from CT/MRI data. Section 3 may be treated as a user manual and details the functionality of the three subsystems in terms of the user interface. This section also includes an example with a tutorial. Section 4 examines the system internals and provides a programmer's guide. Finally the last section discusses future extensions and plans.

2 TECHNICAL TOUR

2.1 Surface Reconstruction Algorithms

Skeletal model reconstruction from voxel data has been an active research area for many years. Schumaker [17] presents an excellent survey of the wide range of techniques that have been utilized. Curiously, and as he points out, the majority of the reconstruction techniques produce only planar $C^0$ approximations of the data set. Relatively little has been achieved in constructing smooth skeletal models using curved surface patches.

\textsuperscript{1}X-11 is an MIT product

\textsuperscript{2}See [3]
There are primarily two classes of model reconstruction techniques. One class of methods first constructs planar contours in each CT/NMR 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 Useiton [9], Christiansen and Sederberg [8] and Boissonnat[6]. The optimal algorithms due to Keppel, Fuchs et. al 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. 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] we present two algorithms belonging to construct \( C^1 \)-smooth models of skeletal structures from CT/NMR voxel data. 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 which

---

Figure 1: Human Anatomy Models Reconstructed using VAIDAK
starts by constructing $C^1$-continuous piecewise conic contours on each of the CT/NMR 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 which 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.

### 2.2 Functional Subsystems

VAIDAK provides a testbed for experimenting with the performance of these methods when various metrics are applied and it provides a testbed for new methods.

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 density values of recognized material. In this way, a good approximation to the appropriate threshold 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 a computer vision 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.

VAIDAK provides functionality to model objects which have multiple contours per slice. This is done in a very 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 preformed 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. Work is currently being performed on a merging operator in VAIDAK 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 application to education and radiology. Veterinary schools are very interested in providing ways for their students to
Figure 2: The VAIDAK Software Architecture
understand the interior structure of animals without destroying the animal in the process. VAIDAK provides just such a tool. We have experimented with voxel data of a hound dog from which we have generated numerous models. Also, orthopedists and radiologists have found VAIDAK to be a very powerful tool for viewing the results of their studies.

Finally to conclude, as mentioned earlier, VAIDAK is part of the SHAASTRA project. SHAASTRA is 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] for further manipulations in the pursuit of an accurate model of biological tissues.

2.3 CT/MRI to Contour Conversion

VAIDAK provides the user with a convenient mechanism for extracting contours from voxel data. Voxel data typically is provided with one file per slice. The number of slices can range from 20 to 1000 or more. VAIDAK employs two different schemes for contour extraction: (1) batch mode in which every slice is automatically scanned for contours of a known density range, or (2) interactive mode where the user is allowed to select the area in which the object is located and thus where the scanning for a contour should begin. In either case the contour is computed automatically.

Also a pixel browser is supplied which allows the user to query the pixels in the density
image of the slice in order to deduce the correct threshold range. Once the correct threshold range is discovered, the range can also be used in other systems that input voxel data, such as marching cubes and some octtree implementations. More will be presented on the voxel browser later in this section.

VAIDAK makes a number of assumptions about the slice data that need to be addressed before providing a working example. VAIDAK assumes that the data for each slice is contained within a separate file and that these files are named with a common prefix and suffix. Commonly the file names are of the form ctf?.dat. ctf is the prefix, dat is the suffix, and the ? is replaced by an integer. For example, the files might be ctf1.dat, ctf2.dat, ctf3.dat, etc... Also, the format of the data within the file is assumed to be of a special form. Auxiliary programs are supplied to convert other commonly occurring formats to VAIDAK form. More on this in the programmer's manual.

Perhaps the best method for explaining the user interface of the CT/MRI conversion subsystem is to present a working example, explaining each step. Press the input button of the command console and let it up on the CT/MRI data selection of the popup menu. The scanner data dialogue window immediately appears. Provided is a text box for object name, file name prefix and suffix, starting and ending contour numbers (and thus files), minimum and maximum density, contour separation, smoothing error, and radio buttons for user select mode and viewing mode. The user, using the input boxes for minimum and maxi-
mum density, provides the range that the edge detection algorithm uses. These numbers constitute starting values and can be changed at any time. The contour separation parameter specifies the distance of separation of adjacent contours. The appropriate distance can only be found through experimentation or from some a priori knowledge of the voxel data. The separation distance can be modified at a later time by the translation command. The smoothing error parameter is used to “smooth” the detected contour. When the user select mode button is ON the user directs the edge detection algorithm. If the button is OFF the system is in batch mode. It scans from the edge of the density image and will usually find one or more exterior contours of the scanned object. If the viewing mode button is ON the interface becomes a pixel browser allowing the user to test the density of the pixels in the voxel image.

When all the parameters are set the user can press the Ok button to proceed. Pressing the Cancel button will abort the CT conversion subsystem.

When the dialogue is complete and the user presses the Ok button the dialogue window disappears and a the user is prompted to place a new window. This window is a grey scale map of, initially, the first slice. The user also must place a dialogue that provides a window for all interaction during the conversion process.

In conversion mode, the left mouse button is used to select within the density window. In viewing mode, every time the left button is pressed the density of the pixel under the mouse pointer is printed to the density text window. The user can experiment in viewing mode to discover what density range is most appropriate. When the user turns the contour mode on by using the radio buttons in the conversion control window, the edge detection algorithm is called at every left button down. The user should strike the button just below the edge of the desired object but always within the object. The system tries to compute a contour. This contour might be an exterior contour that surrounds the object or an interior contour that surrounds a void in an object. If successful, a green contour will be drawn in the window. This contour is a smoothed approximation to the original contour. The user can adjust the density values to get the desired contour. This process is repeated until no more slices remain. The result is a set of polygonal contours that approximates the original scanned object.

The algorithms used in the voxel to contour conversion subsystem also deserve mention. The raw voxel scan data is in the form of a two dimensional array of two byte integers, one array for each planar slice through the object. The value in each cell of the array is related to the density of the scanned object at that point in space. Each array may contain any number of cross sections, i.e., each slice may cut the scanned object in multiple places. To locate the cross sections within a two dimensional slice the following simple algorithm is employed: (1) scan for a cell on an initial edge, (2) starting at this cell hug the exterior of the cross section working from cell to cell and creating a list of two dimensional points until the beginning is reached or a dead end is found, (3) if a dead end is found backtrack, (4) if the path closes and the algorithm does not backtrack to the beginning point then smooth and compress the list of points if necessary. The question arises: What are the properties of an acceptable cell? This implementation employs the following heuristic rule: if the density
value in a cell \( c \) is within range and if the density values of all the cells surrounding \( c \) are within range, then the cell \( c \) is acceptable. The point list is smoothed and compressed by growing segments that are within a pre-described constant value of the original polyline.

3 User Interface and Functionality

In this section the VAIDAK user interface is discussed. This includes a general discussion of the look and feel of VAIDAK along with a thorough discussion of the command interface. Recall that VAIDAK is constructed from three basic subsystems; the discussion in this section reflects this partitioning.

VAIDAK is a fully developed windowed system based on X and XS. XS is a system independent graphics library [3] developed at Purdue University’s Department of Computer Sciences. VAIDAK’s window heirarchy supplies an outline for this section.

3.1 Start up

Upon starting a VAIDAK process, the user will be prompted to place three windows: the command console, the main window, and initially one render window. The main window is used by the browsing and editing subsystems. By default, it is sized at about three inches square; it can be resized at any time.

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.

A rectangular window is also presented upon startup. This will be called the command console. Most user interaction is with the buttons in this window. All the subsystems are initiated from the command console. Each command provided in this window will be discussed in detail later.

3.2 Brief example

At this time, a quick example will be given as an introduction to the functionality of the system.

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.
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.

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.

### 3.3 The Command Console Window

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

**output**

The output command raises a popup menu listing the object names of the currently managed objects. Sliding the pointer down and releasing the button on an entry directs the system to prepare to output data for the selected object. Immediately a dialogue is raised. The dialogue prompts the user to enter a file name. Along with this, four command buttons are presented along the top of the dialog; these buttons are labeled *points*, *polygons*, *DES*, and *B-rep*. Depending on the default X resources that are installed, usually the *points* button is the only one that is ON, i.e., that is highlighted. Any or all of these can be ON simultaneously. If the *a* button is ON then a file of the corresponding format will be created. If the *points* button is on, three dimensional points will be written organized in planar polygonal contours. If the *polygons* button is ON a specialized file containing the contours and the computed surface is output. A detailed explanation of this file format is included in the *programmers guide*. If the *DES* button is on then a standard DES file will be created containing only the polygonal contours. Finally, if the Brep toggle button is on then a file with a SHILP boundary representation in external form will be created [1]. The system defines several default file extensions which are *pts*, *poly*, *des*, and *brep*; these are self-explanatory.

VAIDAK handles two different kinds of objects, a contour object, and a marching cubes object. Both of these are separated internally, thus the user does not need to worry about distinguishing between them. This is done by checking how the object is created, and then adding separate callbacks based upon the type of object. Of course, not all functions are able to be applied to a marching cubes object. However, absolutely no functionality of VAIDAK has been lost. When a marching cubes object is chosen for output the only choices are: *Brep* and *Poly*. The *Poly* output is only a list of polygons (without contours) and the *Brep* is the standard SHILP output format.
3 USER INTERFACE AND FUNCTIONALITY

input

This command has three subcommands which allow the user to input data from different sources. The points subcommand is as was explained previously. The exact format of these files will be detailed in the programmer’s guide. All cross sections of an object must be oriented similarly. That is, they must wrap around the object in the same direction, clockwise or counter-clockwise.

Releasing the button on the polygon data entry of the popup menu produces the same dialogue presented by the points selection. However, in this case a file of a special format, containing polygonal data of the computed surface, is expected. Polygon files created by the output command will be in this format. All file formats will be examined thoroughly in the programmers manual.

The CT/MRI data subcommand invokes the voxel data converter. This subsystem is described in detail in the subsequent 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.

destroy

Like the output command, a popup list of object names is presented. After selecting the object to delete, VAIDAK will require a positive response before destroying the object and all references to it. Generating references to an object is done with the select command and will be discussed later. This command when applied to a marching cubes object, destroys all object windows with the selected object in them.

create

This command presents a popup 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. After selecting an object name or NEW, this command invokes a cross section editor. The cross section editor is a full function polygon editor which additionally allows for the creation of contours on the fly. It can also be invoked through the browsing subsystem. The user must create the cross sections so that the direction of wrap around the object is consistent among all cross sections. The editing commands will be discussed in a following section.

This command does not apply to a marching cubes object.

browse

This command 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 much detail in a later section.

This command does not apply to a marching cubes object.

viewing

The viewing command raises a dialogue window which lists the modifiable viewing pa-
3 USER INTERFACE AND FUNCTIONALITY

Parameters for the CURRENT render window. Note that some of these parameters are also modified by mouse motion in the corresponding render window. The function of each parameter should be clear from its name.

This command does not apply to a marching cubes object.

tile

This command presents a popup menu and invokes the tiling 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. When the heuristic method is used the optimization setting has no effect. 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. At this revision, no means are available for aborting the computation of the contours and thus if the object has a large number of points the user is advised to use the default method first and from this preliminary result label the troublesome cross sections as optimal. After labeling, the contours can be computed again and the optimal method will be employed on the labeled contours. Unless the object has more than 2000 points, it is suggested that the optimal method be used initially.

This command does not apply to a marching cubes object.

tessellate

This command initiates the marching cubes algorithm to a voxel data set. The marching cubes algorithm uses a vastly different approach for contour reconstruction. The reader is referred 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. Once clicked, an input window will appear.

In this window the following information is requested:

Object name - name of the object. Object names are always distinct regardless of the type of the object.

Min/Max Density - the density threshold which is used to determine if the object of interest.

File prefix - the pathname to the data set. It also includes the beginning of the file names. For example: a file called ctf4.dat in the directory /u27/bailey/data will have the prefix: /u27/bailey/data/ctf.

File suffix - Again using the above example, the file suffix would be: dat. (The program adds the period).

Begin file - the begining file in the data set.

End file - the ending file in the data set.
Number of cubes - number of cubes per row of the data set. Each data slice has 512X512 size. Thus each slice can be divided up into cubes (or squares). It is not necessary to use a power of two because a mapping is used from each vertex of the cube to the closest data cell.

On completing the above entries the user clicks the OK button. Another window will appear and at this time the object is being computed internally. Once this is finished, the object will be drawn in the window.

windows

When this button is pressed a list of window numbers is presented; the list is prefixed by the special designator NEW. Each render window is designated 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.

For marching cube objects, special display windows are tied to the objects. The windows command does not apply to these special windows. Instead these marching cubes object windows have the following functions:
Rotation: Any object can be rotated simply by clicking the left mouse button in the desired window, holding it down, and dragging it across the screen. Letting up on the button ceases rotation.

Resize: Any object can be resized by clicking the middle mouse button in the desired window, holding it down, and dragging it across the screen. Letting up on the button ceases resizing.

Menu: By clicking the right mouse button, a menu will appear with the following options:

- Wireframe - simply draws the wireframe of the object.
- Lighted - redraws the object using the default lighting model.
- Edit - brings up the marching cubes editor. Described later.
- Exit - removes the window, but the object is not affected.

translate
This 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. Pressing the translate button presents the user with a list of object names. After selecting an object name a dialogue is presented. The dialogue has three text boxes labeled \( x\text{-delta} \), \( y\text{-delta} \) and \( z\text{-delta} \). If non-zero values are entered into these fields then each time the apply button is pressed the selected object will be translated with respect to the delta values.

Also, a text box labeled contour separation is provided. The distance between contours is initially entered into this field by the system. If this value is changed, then the distance between contours is modified when the apply button is pressed. Note that a mirror image version of the object can be created by entering a negative number in this text field.

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.

This command does not apply to a marching cubes object.

reorient
After pressing the reorient command button and list of objects is presented. If the user chooses one of these object names, the object will be reoriented, that is, the system will be told that the object’s inside is its outside.

This command does not apply to a marching cubes object.

select
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.

**unselect**

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

This command does not apply to a marching cubes object.

**exit**

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

### 3.4 Render window commands and interaction

The render window also has a number of commands. These commands are activated via a popup menu which is generated by pressing the right mouse button. When new cross sections are created or read from the file system or when the user edits a cross section the objects must be retiled. If a tiling has not been computed since the last edit or since the cross sections were entered into the system the *wire frame*, *mesh*, and *lighted* commands are inactive. Also, note that the commands *contour*, *wire frame*, *mesh* and *lighted* affect only the way the object is viewed; it does not actively effect the object itself. Therefore, if an object is referenced by multiple windows, it can be viewed differently in different windows.

A description of the render commands follow:

**contours**

This 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.

**wireframe**

This command sets the display mode to *wire frame*. If tiling has been performed on 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.

**mesh**

This 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.

**lighted**

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

**inside out**
This 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.

Additionally, regardless of the display mode, the user can modify his or her perspective using the left and middle mouse buttons as described in a previous section. Finally, it should be noted that when the contours are computed the render display mode is always reset to wire frame.

3.5 The cross section 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.

If the editor is directed to create a new contour the user will be prompted initially for the z-axis value of the contour. All contours are assumed to lie on a plane which is perpendicular to the z-axis, and thus all points on a particular contour will have identical z values. Additionally, if a new object is being created, i.e., the NEW designator was selected in the create command, then the user will also be prompted for a new object name.

The commands examined in the rest of this section are selected from a popup menu. The right mouse button presents this menu.

add

This command puts the editor in add mode. In add mode the user is allowed to add points to the end of the list of points which make up the polygon. 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.

move

This command puts the editor in move mode. When the editor is in move mode clicking the left mouse button on or near a point and holding the button while the mouse is moved, moves the point once the button is released.

insert

This command puts the editor in insert mode. In insert mode a point is added between the two nearest adjacent points when the left mouse button is pressed. Some checking is performed to assure that unwanted insertions do not occur. For instance, if the left button is pressed and the pointer is nearer to a point in the polygon list than it is to a segment, then no action is taken. The editor considers this ambiguous. If the user desires to add a new point very near an existing point then it is suggested that the new point be inserted near the middle of the appropriate segment and then moved into place.
delete
This command puts the editor in delete mode where each time the left button is pressed the nearest point is deleted from the list.

edit \( z \)
Unlike the previous commands, this command does not perform a mode switch. This command raises a dialogue window which 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 command can be canceled without changing the current \( z \) value by pressing the cancel button in the dialogue window.

close
This 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.

exit
This command exits the editor disregarding all edits. This command requires a positive response to complete.

Additionally, pressing the middle mouse button and holding it while sliding the mouse, moves the cross section once the button is released. The only method for rescaling the cross section at this revision is by editing the size parameter (refer to the viewing command).

3.6 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 right mouse button initiates this menu. The description of these commands follow.

first
This command makes the first cross section in the list the current cross section, skipping all cross section in between. The current cross section is highlighted in red in the render window and is orthographically displayed in the main window.

last
This command makes the last cross section in the list the current cross section.

edit
This command invokes the cross section editor on the current cross section. Refer to the description of the edit subsystem commands contained in this document.
delete
This command removes the current cross section from the list permanently. No positive
response is solicited.

copy
This command creates an 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.

method
This command allows the user to label the current cross section as heuristic, optimal, or
default. When contouring is performed in default mode the default method is used unless
the cross section is labeled heuristic or optimal. Please refer to the contour command for
further explanation.

exit
The exit command causes the browse subsystem to be exited.

3.7 The voxel browser and contour detector
VIADAK is equipped with an interactive voxel data browser 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.
This subsection will discuss the commands available from this window.

The numerous voxel images of a single study are viewed one at a time by the system.
The next and prev commands of the auxiliary window allow the user to move through the
images.

Button-down events within the voxel image window are interpreted based on current
mode of the subsystem. Three modes are available: browse, contour, and manual. The mode
can be switched interactively by activating the appropriate toggle button in the auxiliary
window.

If the subsystem is in browse mode then upon every button-down event the x and y
position of the mouse pointer along with the density value of the voxel datum at the point
is displayed in the auxiliary window. In this way the user can interrogate the voxel data in
order to help deduce the appropriate threshold that should be used for contour searching.

In contour mode a contour searching algorithm is employed at each button-down event.
The mouse pointer position provides the searching algorithm with a starting position. Ini­tially
the algorithm searches for an edge. Once an edge is found, the algorithm constructs
a connected contour by moving from voxel to voxel in the two-dimensional image.

In manual mode the user can sketch out the contour interactively directly on top of the
voxel image. In this case, the voxel image acts only as a reference. Move the cursor to the
starting point of the contour. Click down on the left mouse button and drag the cursor
around the contour. Releasing the button will close the contour. The user does not need
to worry about connecting the starting and ending points, because the program will take
3 USER INTERFACE AND FUNCTIONALITY

Figure 7: Editing and Browsing using the 3D Object Editor

care of that. This mode is only used where the noise level in the image is so high that the
countour searching algorithm cannot perform satisfactorily.

Threshold values that are used during contour searching can be changed by activating
the threshold command button and entering the appropriate values into the maximum and
minimum text dialogues of the dialogue window.

The process can be terminated at anytime by the cancel command button in the auxiliary
window.

3.8 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.

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.

4 Implementation Issues

In this section the interesting implementation issues involving design and data representation are discussed. This section is partitioned into three subsections: Graphics, Data Structures, and File Formats.

4.1 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 irregardless of the platform. More information on XS is available in [3].

4.2 Data Structures

An heirarchical data structure is used to represent VAIDAK objects. Underlying each object is a set or list of planar polygon contours. The polygonal and curved reconstructed surfaces reference back to this contour data structure. Editing the contour points implicitly, but directly, affects the computed surface.

Based on this, an interesting method for representing the polygonal surface, computed by VAIDAK, is of some interest. A contiguous set of triangular facets is computed between each pair of adjacent contours. We have found that each set of facets can be represented by a single compacted list of references to the vertices of the contours. By maintaining a reference to the appropriate previous two vertices, a new triangle is defined by each new vertex in the list. Since the contour slices are assumed planar and perpendicular to the
4 IMPLEMENTATION ISSUES

z-axis, the last vertex on each of the two adjacent contours constitutes the appropriate previous two vertices.

Objects themselves are stored in a global object list. Windows, which are also stored in a list, contain a variable length list of references to the object list.

4.3 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.

5  FUTURE EXTENSIONS AND PLANS

VAIDAK provides a simple mechanism for reconstructing a surface of a object with multiple
objects. This is done by allowing the user to define sub-objects. The idea is that sub-objects
are defined in such a way that each sub-object contains only a single contour on each slice.
Although this method for handling multiple contours per slice is very effective, attempts
have been made to compute surfaces of branched objects automatically. This appendix
presents some of this work.

The problems that arise in trying to automatically compute a surface of an object
with branches 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 information
available. The information describing the surface of an irregular object is infinite whereas
this subsystem will attempt, from a finite amount of information to rebuild the object.
Clearly no algorithm can perform the task correctly every time.

In the most general case, multiple cross sections on both levels will be present with
one-to-many transitions possible. In this case a mapping of the cross sections from one
level to the next must be computed. If a one to many transition is present then a vertex
matching will be employed along with some method to introduce bridge points. Once the
vertex matching is completed and bridge points are introduced, cross sections can be built,
and the methods already implemented for single transitions will be used.

The key observation, and one that has not been given any attention in the literature,
is this: many methods for computing good contours between adjacent cross sections have
been presented and indeed two of these methods have been implemented in this system.
What has not been studied is applying this relatively mature technology to compute optimal
contours when multiple cross sections are present. The algorithm can no longer consider
only two cross sections. Also, the algorithm must consider the location and number of
bridging points when a one-to-many branch is encountered.

The concepts are the same but the solution is hauntingly elusive. In its simplest form,
a mapping can be produced which locally minimizes some metric. However, it seems inter­
esting to try and compute an optimal mapping based on some idea of distance or area,
i.e., to compute a tiling that is minimum over all tilings and mappings of cross sections. Metrics based on approximating each cross section by a bounding box or a convex hull and applying a distance metric to the approximation's center of gravity has been considered as a heuristic method but no results are implemented at this release.

References


