Reconstructing free form 3D objects from 2D images with PEX
by Xia You

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science
Montana State University

Abstract:
This thesis develops a software which reconstructs free form 3D objects from 2D images based on Motif/Pex. It allows the users to edit, transform and render random shape 3D objects interactively through a friendly and consistent Motif style Graphic User Interface.

This software displays a series of 2D images and also provides several image processing methods which make it possible to enhance the quality of the images. 3D bodies are reconstructed with B-splines after the user defines a three-dimensional polygon structure data.

As the result, 3D human bodies were successfully reconstructed from medical MRI images. It is also applicable for other purposes 3D free form bodies rendering.

RECONSTRUCTING FREE FORM 3D OBJECTS FROM 2D IMAGES WITH PEX

by

Xia You

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

November 1994

APPROVAL

of a thesis submitted by

Xia You

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage,format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

11/4/94 — Date

J Dunbigh Stanley — Chairperson, Graduate Committee

Approved for the Major Department

11/4/94 — Date

J Dunbigh Stanley — Head Major Department

Approved for the College of Graduate Studies

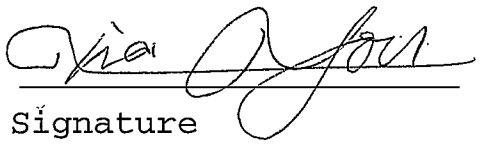
11/8/94 — Date

R L Bro — Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.



Signature

Nov 9, 1994

Date

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

This thesis develops a software which reconstructs free form 3D objects from 2D images based on Motif/Pex. It allows the users to edit, transform and render random shape 3D objects interactively through a friendly and consistent Motif style Graphic User Interface.

This software displays a series of 2D images and also provides several image processing methods which make it possible to enhance the quality of the images. 3D bodies are reconstructed with B-splines after the user defines a three-dimensional polygon structure data.

As the result, 3D human bodies were successfully reconstructed from medical MRI images. It is also applicable for other purposes 3D free form bodies rendering.

CHAPTER 1

INTRODUCTION

In recent years, with the big progress in computer hardware and software, especially the availability of low-cost visualization-compatible workstations, networks, and powerful graphics software, Computer visualization has become more and more popular. On the other hand, the rapid growth of large-scale computing in the basic sciences and the steady accumulation of high-bandwidth data sources (radio telescopes, medical scanners, etc.) has also increased the interest in scientific visualization as a computational technology. Scientific Visualization has become a tool for discovery and understanding, a tool for communication and teaching. Mathematicians want their equations to be visualized; engineers want their design to be visualized in order to make the designing cheaper and faster; In a word, visualization has become so popular that more and more scientists, engineers, physicians are involved in this new area.

There is a project called Boron Neutron Capture Therapy (BNCT), which is an international development aimed at curing certain cancers. Idaho National Engineering Laboratories (INEL), the University of Utah, and Montana State University have been responsible for the development of the Computer Software for this project. There is an existing software system called Bnct_edit. This software first shows a two dimensional medical image, then it can display a three dimensional representation of a human head, brain, and other objects in the head, to assist in the treatment and experimental planning of radiation therapies by creating accurate patient geometries.

This software was first developed for the Apollo environment using GPR at the University of Utah; then it was ported to X-Windows and significantly enhanced by INEL and Montana State University. In this thesis, I attempted to adapt the methods used in the BNCT project to the PEXlib/Motif environment. PEXlib is a programming library for 3D graphics. It's the lowest-level and most direct interface for drawing 3D pictures in the X Window System. Widely and freely available, efficient, and supported by many workstations and terminal manufacturers, it is a popular choice for the graphics

interface of 3D applications. PEX is the 3D extension to X. It adds over 200 X protocol requests for defining and displaying 3D pictures. PEX provides all the common features found in most modern 3D graphics systems, but provides them in a way that's seamlessly integrated with X. PEX is a high-level graphics library. It allows a programmer to describe a graphic image in terms of familiar objects and attributes, without having to deal with the details of producing that image in windows. All the details of producing the picture are handled by the PEX server. The user merely specifies the geometry, the location, and some appearance attributes for the objects. PEXlib and Motif have just worked together for a few months at the Department of Computer Science at Montana State University, Bozeman. The software is called "Show-mri". As the name indicates, it manipulates medical images, then reconstructs 3D bodies of patients. It is also applicable for rendering other 3D free form bodies without major changes. I will use medical images to describe how this program works through this thesis. After executing the program, it will show some medical images with a unique Z-value for each image. Users can then edit B-spline contours interactively by clicking the button of the mouse through a friendly motif-style

Graphics User Interface (GUI). Finally, 3D bodies will be reconstructed and displayed by interpolating those B-splines. Users can observe the 3D bodies in different orientations, and can change the color of each body.

The system was developed on a DEC alpha machine with the OSF/1 operating system. A PEX terminal, which has 4M bytes of code memory and 8M bytes of data memory, was used for the program development. Because of the high portability of the Unix program, this software can easily port to other PEX platforms such as HP and SUN workstations.

There are three major parts to this thesis: the first part is Motif-style Graphics User Interface, which will be presented in chapter 2; the second part is Graphics which includes B-spline drawing, picking, 3D rendering, etc, some of which were done by Pexlib, some by Xlib. In chapter 4, I will describe the details; the last part is image processing part including some image-processing utilities which are presented in chapter 4. This thesis concludes with chapter 5.

CHAPTER 2

THE GRAPHIC USER INTERFACE OF SHOW_MRI

In this chapter, I will show the Graphic User Interface of this package. I will not only present the appearance of the Graphic User Interface of this program, but also will discuss some detailed information about how it was implemented with Xlib/Motif.

"Show_mri" was designed through a consistent Motif-style Graphic User Interface. Motif has become an industry standard for displaying windows with text and graphics. Now after years of proprietary window interfaces, applications and their users can count on a consistent interface across almost all producers and models of computers. It is no longer necessary to produce separate versions of an application to run on different proprietary window systems, and it is no longer necessary for users to learn a different interface to an application simply because it's running on a different machine.

The Main window of "show-mri" system, shown in **figure 1**, is the only window after the software is

executed.

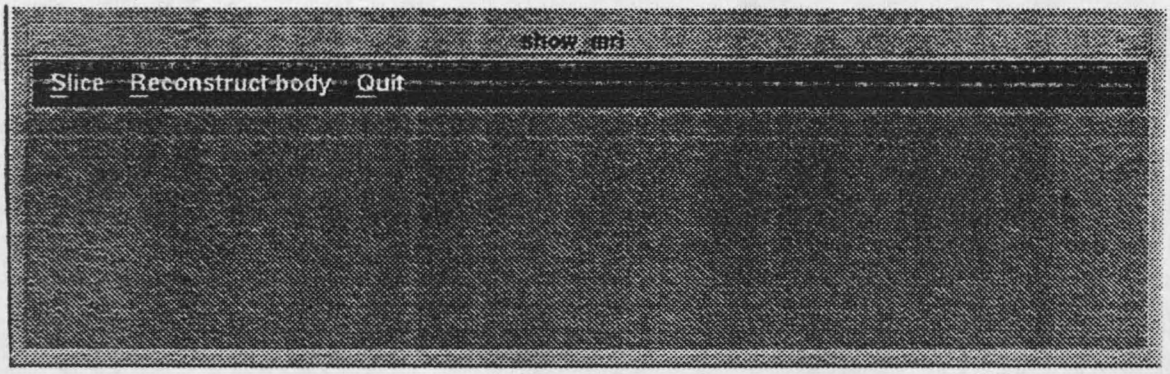


Figure 1. Main window of "Show_mri"

At the top of the window, there is a Title Bar with a Title Area at the center and two push buttons for Minimize and Maximize at the right corner. Also, there is a window menu push button at the left corner; by pushing that button users can get a pull down menu for some window control commands. Below title bar there is a menu bar which is designed by the application programmer, which are the functions that I have implemented in this program. The next area is the drawing area, which was used to display the histogram in this program.

The menu bar is the most important part in this window, and provides the functions that are needed to use the software. In designing this menu bar, I used a general function to generate these menus with the following data structure to specify each menu item.

Table 1. Record for general menus (struct _menu_item)

Type	Name	Notes
*char	label;	the label for the item
*WidgetClass	class;	pushbutton, label, separator..
char	mnemo	mnemonic; NULL if none
*char	accelerator;	accelerator; NULL if none
*char	accel_text;	to be converted to compound string
*void	(callback) ();	routine to call; NULL if none
XtPointer	callback_data;	client_data for callback()
*_menu_item	subitems	pullright menu items, if not NULL

By using this general function, new functions can be added quite easily, which gives this program a good flexibility for expansion.

The first item is "Slice." Users can use both mouse or "alt-S" to activate this function, which is used to generate a window to display images for operations.

The second item is "Reconstruct body" activated by a clicking of a button of the mouse or "alt-R," which is used to reconstruct bodies after drawing contours on the image windows.

The last item is "Quit" for quitting this program, activated by mouse or "alt-Q."

There may be several windows on the screen at the same time. Users can use the mouse to move around and focus on the desired window. Most jobs to operate this software can be done by using the mouse except on a few

occasions when you will need to use key stroking.

Generate Slices of Image

When the "Slice" button has been pushed, there will be a pulldown menu. Currently there is just one item: "Make New slice" on the menu. After that, a window of the image will be displayed on the screen after you choose a Z-value and an image-file name.

Figure 2 show the dialog-popup window for users to enter a Z-value, which is the Z value of the slice image normally within the range 0.0 to 1.0.

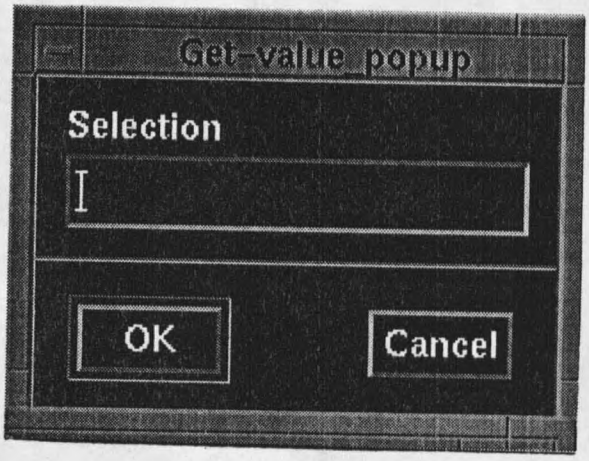


Figure 2. Dialog-popup Window for Entering the Z-value

Figure 3 is the dialog-popup window to choose the image file users want to display. Users can use this tool

to navigate in the file system to find the right image file to be shown.

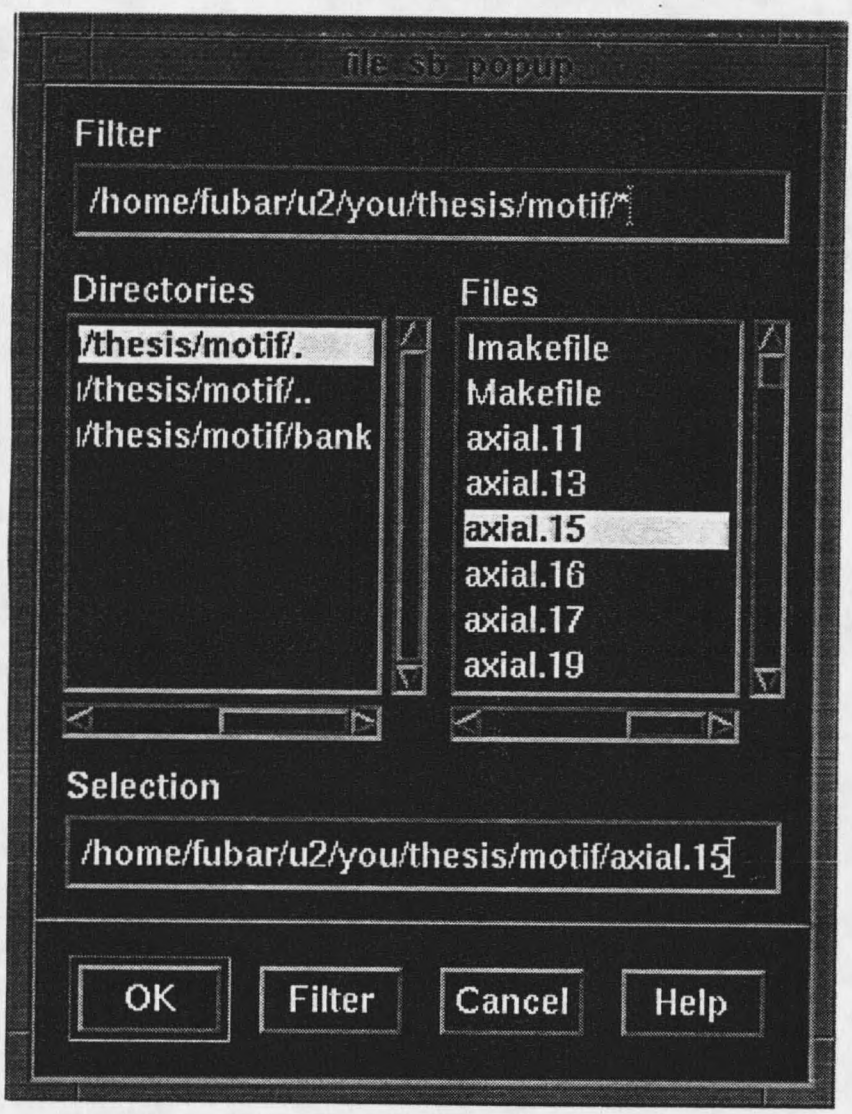


Figure 3. File Choosing Window

Windows of Image

After the z-value and the right x-image file have been chosen, a window of an image will be displayed. There are two items in the menu bar area. One is "Edit" which includes functions for the users to construct B-splines for bodies, while the other is "Option" with several image processing utilities for the users to enhance the picture quality, which I will discuss in detail in the next chapter.

Below the menu bar, there is a push button called "Locate" for switching whether or not to display the mouse's current position and pixel value.

At the bottom of the window, there are two assistant push buttons for editing B-splines. Figure 4 is the window of images with the "Locate" push button switch on.

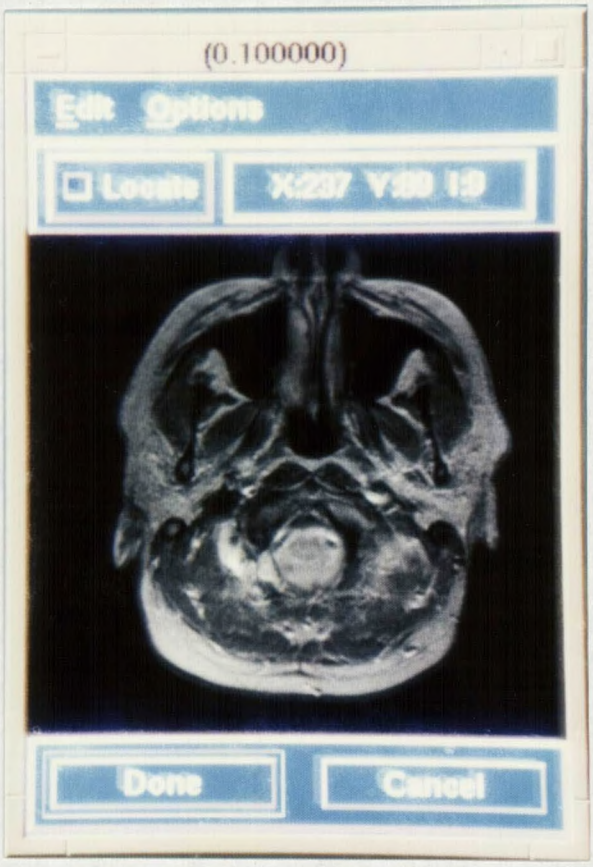


Figure 4. A Window of A Image

Each window of images is associated with a record which is used for the management of images. All the image records are organized in a single linked ring in this program. Table 2 is the data structure of the record for image windows.

Table 2. Record for image windows. (struct _slicering)

Type	Name	Note
Window	window	Image window
Widget	report_label	report_label of image window
*char	filename	image data file
*XImage	ximage	Ximage for the current window
int	nrows	rows of the draw area
int	ncols	columns of the draw area
float	zvalue	z value of image
*_slicering	next	pointer to other image node

Edit the B-splines

There are four functions in the "Edit" pull down menu. They are "New Body in Slice," "Copy Body to Slice," "Edit Body Slice," and "Add Flex to Body."

"New Body in Slice" is the first function which users should choose when they edit a new B-spline, which pops a window for users to enter a unique name for that body. Then one can use the mouse to define several flexes to construct a smooth B-spline curve, which is normally a contour of a body at a specific slice, before pressing the "Done" push button. Also, users can stop defining the current B-spline by pressing the "Cancel" button.

"Copy Body to Slice" is a simple function used to

copy a B-spline from one image window to another so that the bodies can be reconstructed by those sequential B-spline curves.

The other two functions are for users to interactively change a specific B-spline to fit the contour of the bodies more closely. Users can use "Edit Body Slice" to move one flex from one place to another place; and "Add Flex to Body" is to add a new extra flex in that B-spline when the users feel the current flex is not enough to describe the contour of the body.

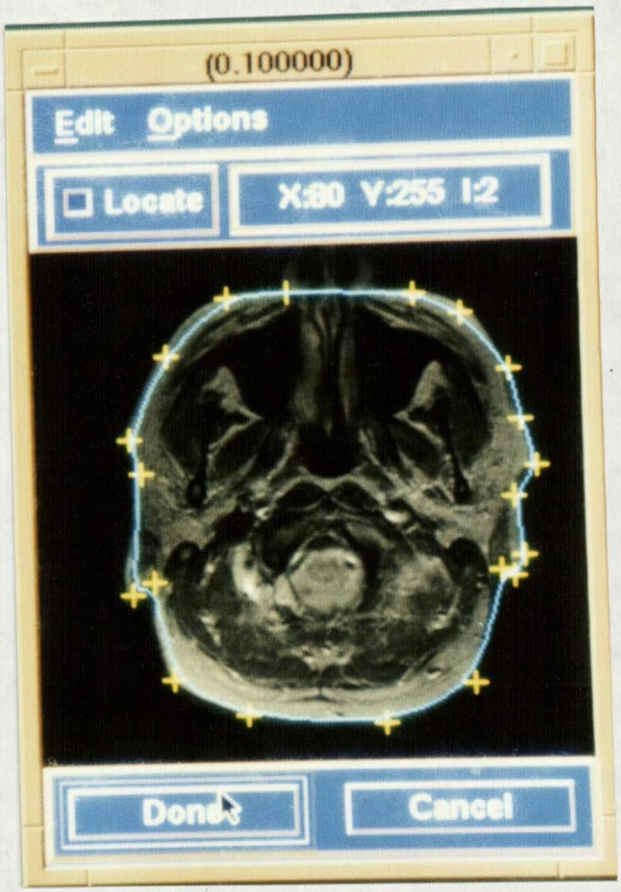


Figure 5. A Defined B-spline Curve

Display Window for Bodies.

Finally, the bodies can be displayed after the B-spline contour of each body has been defined (**figure 6**).

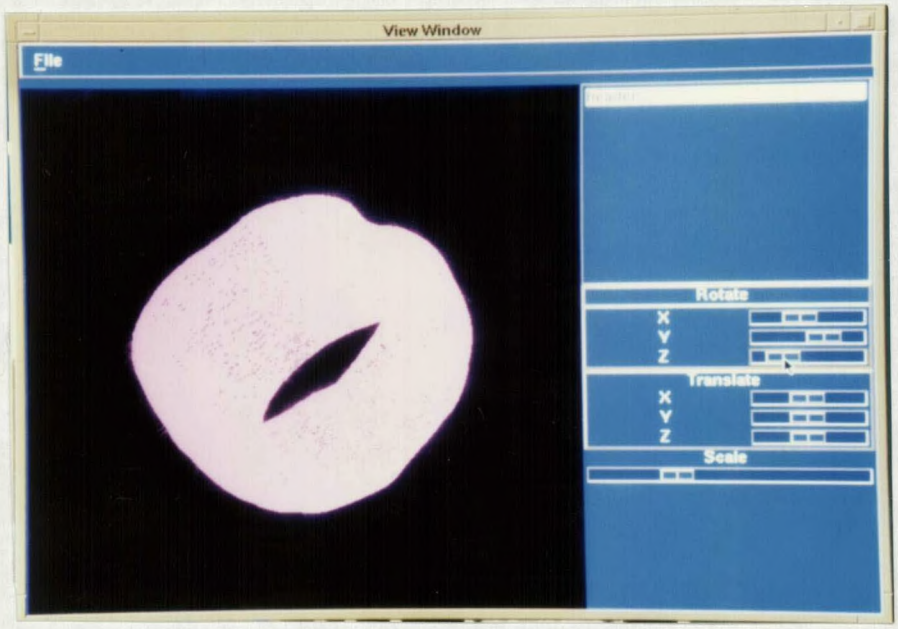


Figure 6. Reconstructed Dog Head with B-spline

There are seven little slide windows at the lower right, used to change the appearance of the bodies seen on the window. From top to bottom, they are three slides for transforming X, Y, Z, three slides for rotating around X, Y, Z axis, and a scaling slide.

At the top right, there is a list widget which shows the bodies' name in the graphic area. Users can change each body's color by clicking the body's name, which can generate a popup window, as shown in **figure 7**, to let users choose a new color for that body.

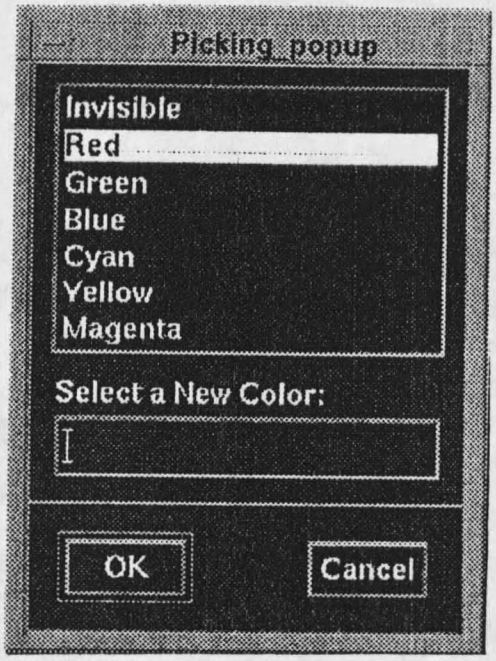


Figure 7. Popup Window for Changing Color

CHAPTER 3

UTILITIES FOR IMAGE ENHANCEMENT

Image Processing is a field which has grown rapidly over the last decade. Digital image processing can be used for improving pictorial information for human interpretation and extracting data autonomously.

In the program, I have implemented several image processing algorithms for the users to get a better look at the images. Users can use one function, or combine several functions to meet their requirements. Also, new functions can be added in a straightforward manner to this program if needed.

In the rest of this chapter I am going to discuss the image processing functions I used in this program. More detailed information can be found in reference [2].

Display The Image

This program takes the image file, generated by some other tools, like CT, for granted. The current PEX-

stations I used have 256 different colors in a colormap, of which I used pixel values from 46 to 255 as a grey colormap from black to bright, and reserved the follow pixel values for displaying bodies.

Table 3. Colors Reserved for Bodies.

Color	Pixel value
RED	40
GREEN	41
BLUE	42
CYAN	43
YELLOW	44
MAGENTA	45

All the other low pixel values are used for the Motif window's usage.

When reading a image file, the program first formalizes the image's value to fit the color range used, then generates an X-image structure for displaying. Users can find more detailed information about X-image structures and usage in reference [7].

Bump, Negative and Two_tone

These three methods for changing pixel values are quite easy but very useful, e.g., if the image shown is too black, users can use Bump to brighten it.

Bump adds or subtracts a number to all the pixel values on the image.

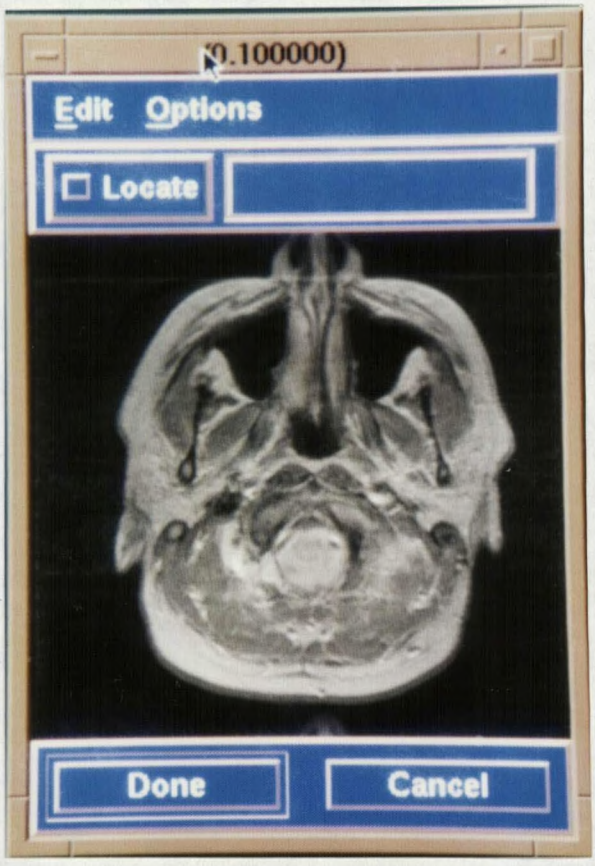


Figure 8. Image Bumped with 20

Negative replaces all the pixel values $X_{i,j}$ with Maximum_color + Minimum_color - $X_{i,j}$, which is like a negative film of the picture.

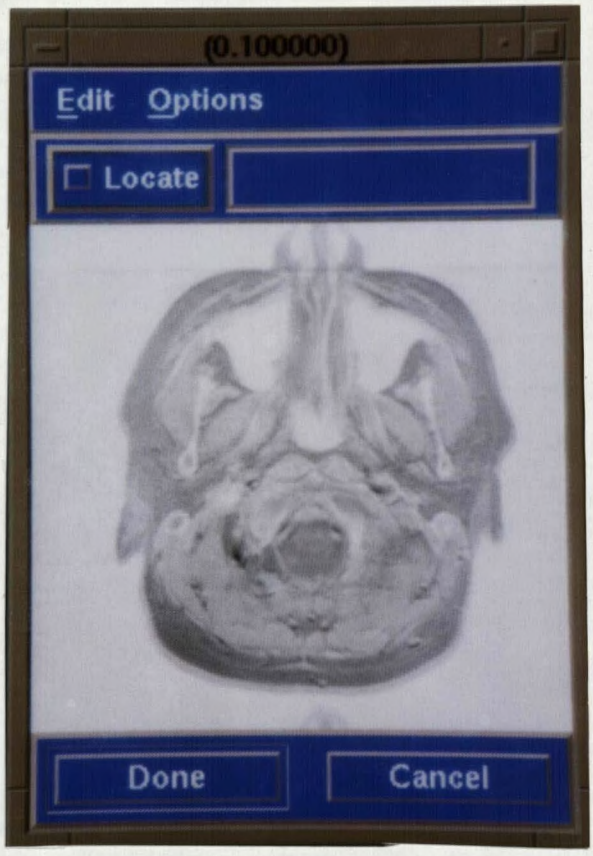


Figure 9. Image of Negative

Two_tone sets up a thresholding value, then replaces each value of a pixel on the image with Maximum_color if its original value is larger than the thresholding value; otherwise the original pixel value is replaced with Minimum_color.

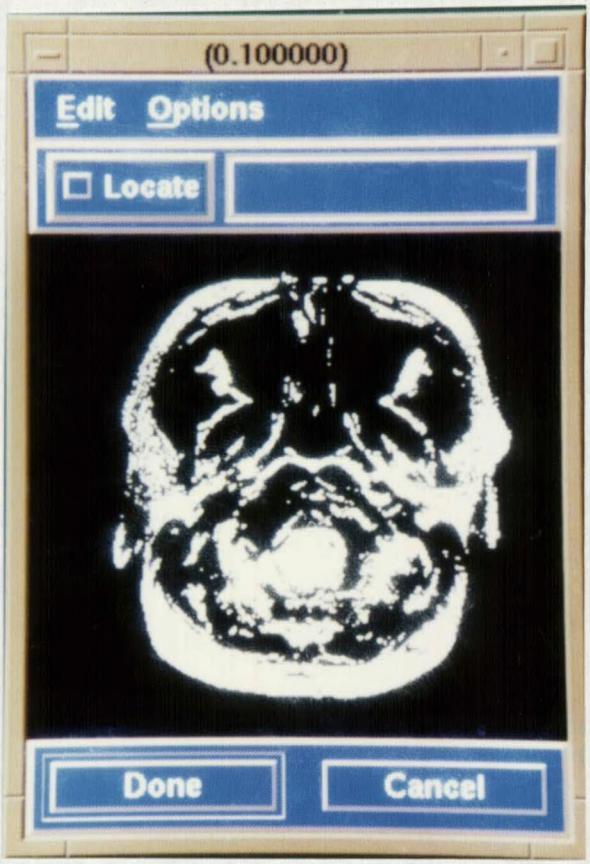


Figure 10. Image after Two_tone

Rotate and Flip

Rotate and flip are two functions for changing the orientation of the image; users can rotate the image by 90, 180 and 270 degree (**figure 11**), and flip the image horizontally or vertically (**figure 12**).

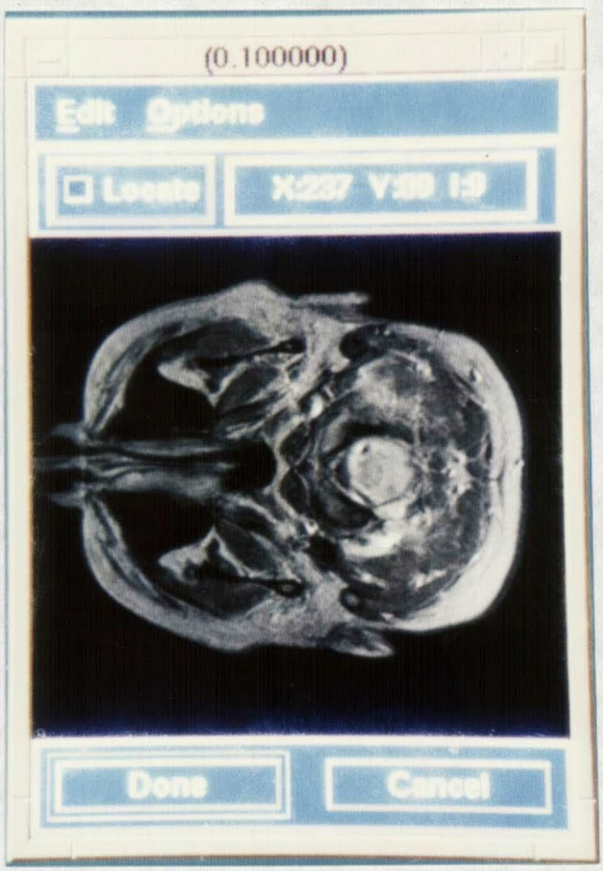


Figure 11. Image Rotated 90 Degree

Figure 12. Image Flipped Vertically

Histogram and Equalization

The histogram of a digital image with gray levels in the range [0, L - 1] is a discrete function $p(r_k) = N_k/n$, where r_k is the kth gray level, n_k is the number of pixels in the image with that gray level, n is the total number of pixels in the image, and k = 0, 1, 2, ..., L-1.

The concept of histogram is important in image processing, and there are several processing methods related to it. In this program, I included histogram generation, displaying, and equalization. **Figure 13** is a histogram of the image of **figure 4**.

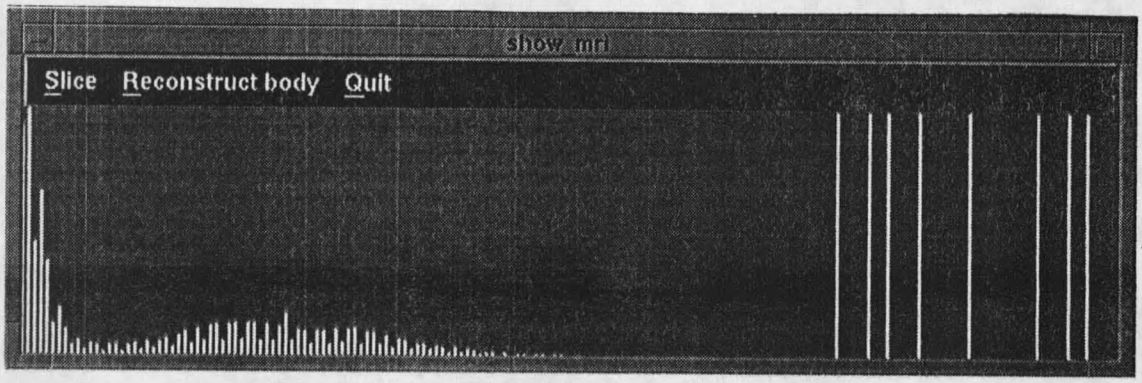


Figure 13. Histogram of The Original Image

Histogram equalization is useful when the dynamic gray level range of the original image is small, and many of the gray levels are wasted. After equalization, the entire pixel value is represented (**figure 14**). The corresponding histogram is shown in **figure 15**. The histogram equalization algorithm is described in the following steps:

1. Generate the histogram by enumerating each pixel falling in a given gray-level from black to white.
2. Calculate the probability density function of

this histogram. New values are from 0 to 1.
3. Calculate the cumulative density function by computing the accumulated probability for each level.
4. Create the transformation function by distributing the cumulative density function to the total range of possible pixel values.
5. Apply this transformation function to all the pixels in the image.

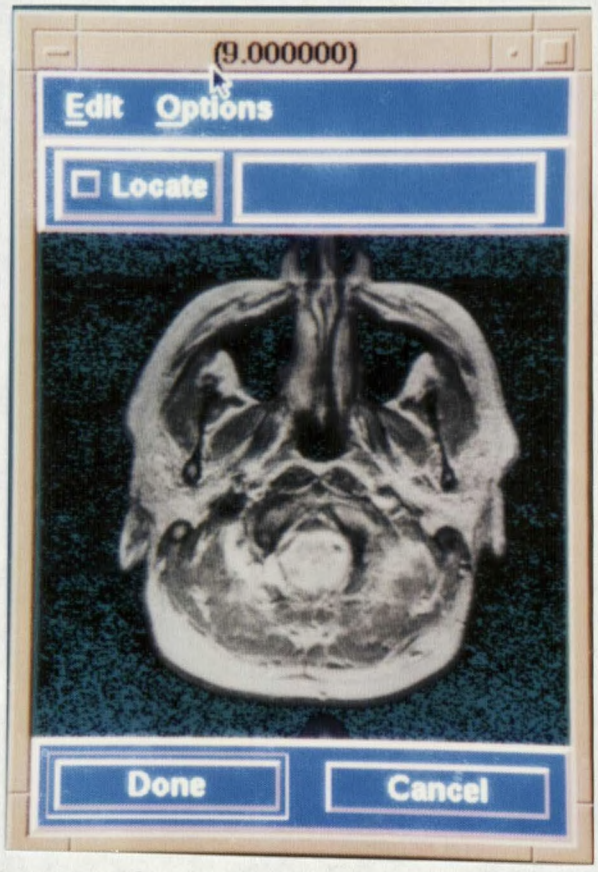


Figure 14. Image after Histogram Equalization

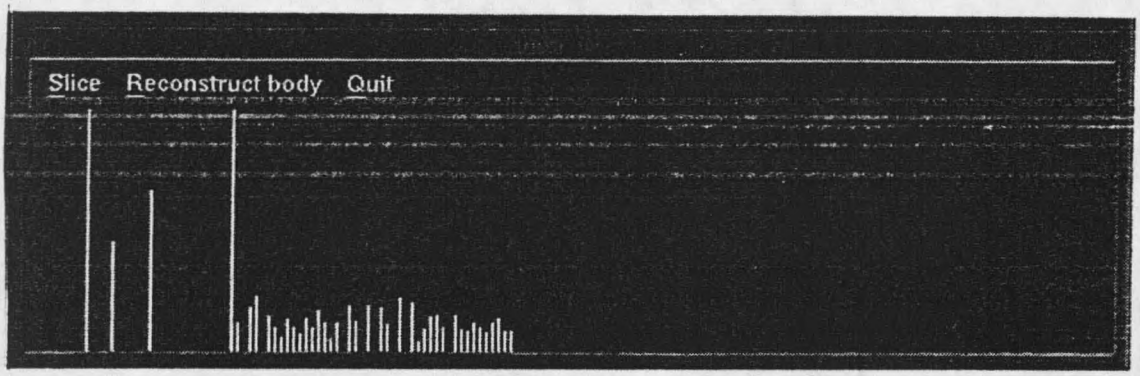


Figure 15. New Histogram after Equalization

Sharp and smooth

Smoothing is a effective way to remove the noise of the image. There are several smoothing algorithms. I chose and implemented the average smoothing algorithm in the 3x3, 5x5, and 7x7 neighborhood.

The algorithm is quite simple. It applies the following process to all pixels (except the pixel on the edge): replace the old gray level by the average of the gray level of the pixels in the indicated neighborhoods.

Figure 16. Image after 5x5 Average Smoothing

Sharping is used to emphasize edges and points in the image. There are also several methods. From them I chose and implemented the method using 3x3 Roberts cross-gradient operators.

The Roberts operators is shown below:

-2	-1	0
-1	0	1
0	1	2

Figure 17. Roberts Operators

Figure 18. Image after Sharping

Contrast Stretch and Area Contrast

Contrast stretch is affected like histogram equalization, but users have more power to manipulate the image by choosing the parameters.

Figure 19 is a typical transformation used for contrast stretching. The locations of points (r_1, s_1) and (r_2, s_2) control the shape of the transformation function, which users can set to reach the best result, depending on the original image. Figure 20 is the image after contrast stretching.

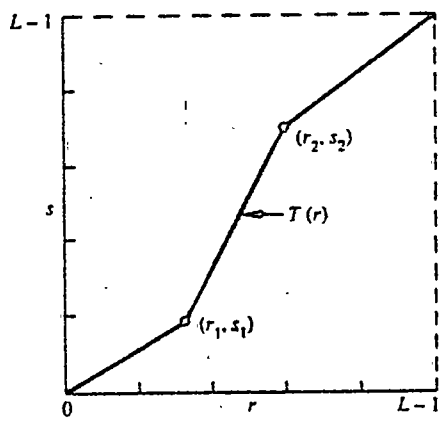


Figure 19. A Transformation Used for Contrast Stretching

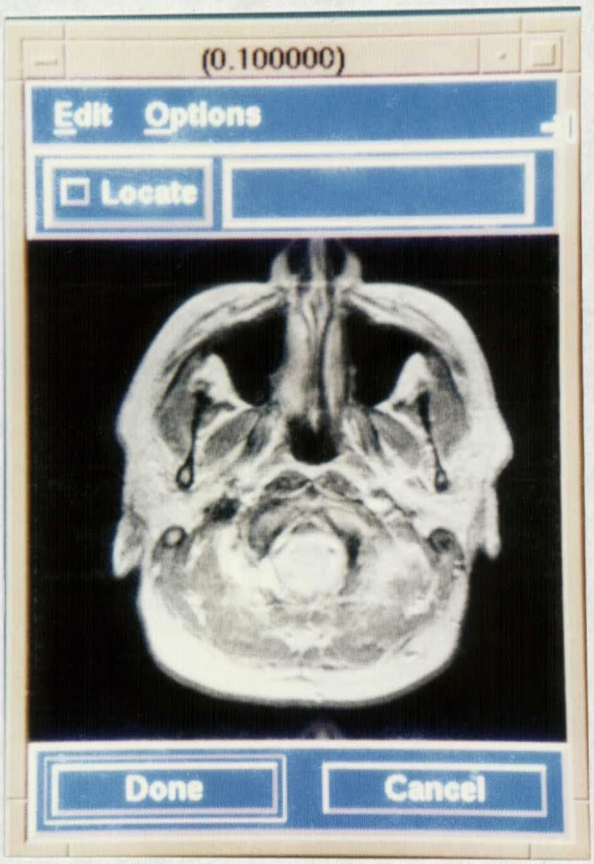


Figure 20. Image after Contrast Stretching

Area contrast is a derivation of the contrast stretch. There are just a few differences in the algorithms:

1. users use the mouse to define a rectangular area by setting two diagonal points.
2. the program will apply contrast stretch to this rectangular area.

Figure 21 is the image after Area Contrast.

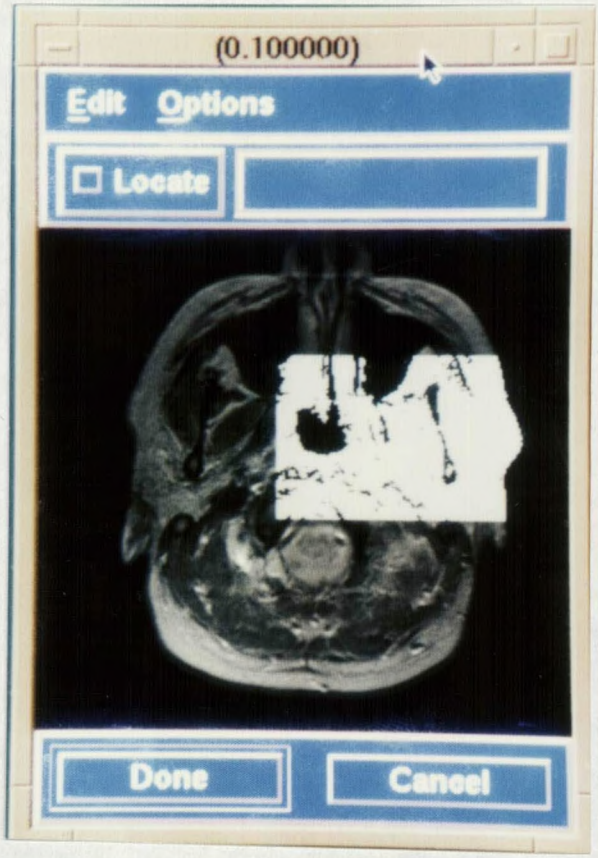


Figure 21. Image after Area Contrast

Local Enhancement

Local Enhancement is used to get better local enhancement, because other functions like histogram equalization are global in the sense that pixels are modified by a transformation function based on the gray-level distribution over an entire image. But when one

needs to enhance details over small areas, the number of pixels in these areas may have negligible influence on the computation of a global transformation. The solution is to devise transformation functions based on the graylevel distribution - or other properties - in the neighborhood of every pixel in the image.

Apply the following algorithm to each pixel in the image except those on the edge:

1. Define a square or a rectangular neighborhood and compute the histogram of that neighborhood.
2. obtain a histogram equalization transformation function of that area.
3. map the gray level of the pixel centered in the neighborhood, based on the function obtained.

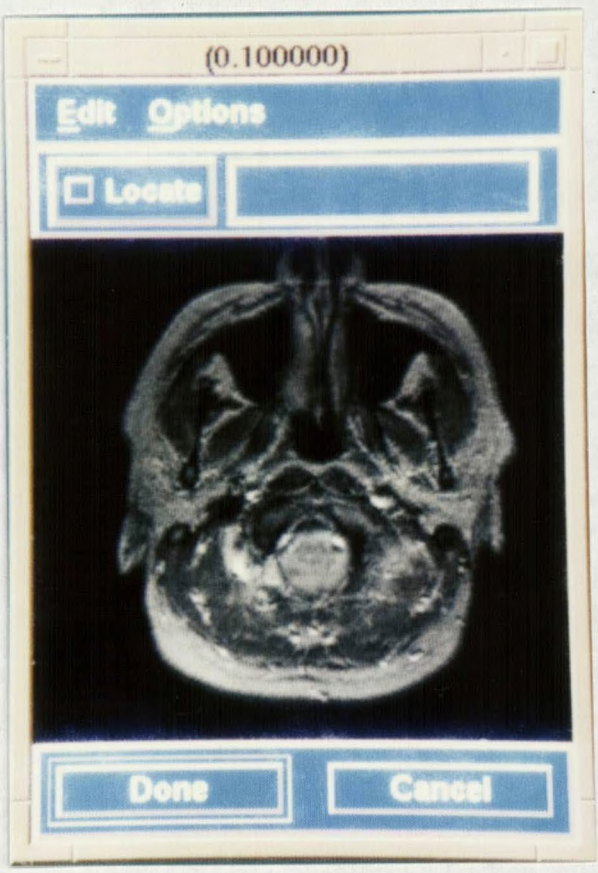


Figure 22. Image after Local Enhancement

CHAPTER 4

FREE FORM SYSTEM

In this chapter, I am going to present the major functions and methods I used in this project, which include picking, bodies (B-spline) generation, three dimensional transformation, and some others like z-buffer. By using these methods, users can reconstruct the bodies interactively through a serial of 2D images.

Picking

In the program, users first choose some controlling points to define a rounded B-spline curve to represent a slice of contour of the body, then they can copy it to another slice, or edit the controlling points, or add more controlling points to represent the contour of the body more concisely. In order to finish those functions, a way of picking a specific controlling point must be realized. There is a picking function in PEXlib which makes most 3D picking quite easy, but it does not work

well because of the following reasons:

A list of controlling points is drawn by the following PEXlib function call:

```
PEXMarkers (dpy, g_renderer,
            PEXOCRender, count, dots);
```

Where "count" is the number of the control points, and "dots" is an array of the coordinate of the controlling points.

If we do a picking, the PEX server will identify the statement to generate these points and pick all the control points in that list but not a specific one, because the points are generated by one statement.

So in the program, I used a fundamental method to pick the point instead of using the provided picking function as follows:

```
xtemp = (x_i - x);
ytemp = (y_i - y);
if ((-PICK_RANGE < xtemp) && (xtemp < PICK_RANGE) &&
      (-PICK_RANGE < ytemp) && (ytemp < PICK_RANGE))
```

PICK_RANGE is the criteria distance for picking. where x, y are the position of mouse. and x_i, y_i are the coordinate of a controlling point.

By comparison against all of the controlling points which were defined, a distinguishing point can be picked, or none of them will be picked if the PICK_RANGE is set

properly (2 pixels distance is used as PICK_RANGE in this program).

B-spline Curve and The Construction of Bodies

The major part of this thesis is to use B-spline to reconstruct bodies of the human brain.

In order to generate rounded smooth B-splines, I used the rational uniform B-spline generation function provided by PEXlib.

Rational curves are chosen because they are invariant under rotation, scaling, translating, and perspective transformations of the control points. Thus, the perspective transformation needs to be applied only to the control points, which can then be used to generate the perspective transformation of the original curve.

General rational cubic curve segments are ratios of polynomials:

$$x(t)=\frac{X(t)}{W(t)}, y(t)=\frac{Y(t)}{W(t)}, Z(t)=\frac{Z(t)}{W(t)}$$

Where X(t), Y(t), Z(t), and W(t) are cubic polynomial curves whose control points are defined in homogeneous coordinates.

```
PEXNURBCurve (dpy, g_renderer, PEXOCRender,
              PEXRational, order, bodyindex->knots,
              slice->count + 4, cpts, tmin, tmax);
```

When I used order 3, the weight of every control point was 1. The knots are generated by a subroutine as (0, 0, 0, 1, 2, 3....).

I used the following method to make the B-spline curve smooth and rounded by duplicating four points at the end of B-spline and setting "tmin" to the first knot plus 0.5 and "tmax" to the last knot minus 0.5 which will cut the B-spline into a smooth and rounded shape (**figure 6**).

Z-buffer

Because there is not always only one reconstructed body on the view window, the users should always see the closer object regardless of the order in which the primitives are created. In order to ensure that only unobscured portions of primitives are drawn, some kind of hidden line and hidden surface removal (HLHSR) method must be implemented. In this program, I used the Z-buffering provided by PEXlib.

Z-buffering is the most common method of performing

HLHSR which can be done very easily in PEX. Z-buffering works by associating a "depth" variable with every pixel of a window. The Z buffer is essentially a two-dimensional array of these variables. When a primitive is drawn, and causes some pixels to "light up," the depth variable for each pixel is touched but that primitive is given a value. This value indicates the distance of the corresponding point on the primitive from the front of the picture, when a subsequent primitive is drawn, and touches one of the same pixels. PEX compares the depth currently stored for that pixel with that of the new primitive. If the new point is closer to the viewer, then the new primitive overwrites the old primitive at that pixel, and the new depth is stored in the Z buffer. The pixel is set to the color of the new primitive. If the new point is further from the viewer, and therefore obscured by the previous primitive, then the pixel is not changed, leaving the pixel set to the color of the previous primitive. After all primitives have been processed, the picture is left showing only those portions of primitive that the viewer can see from his/her viewing position.

Transformation

In order to manipulate the reconstructed bodies to get different views, a transformation method must be used. I chose a method, which does transformtion to the objects in the global coordinate system, from the several methods provided by PEXlib.

All graphics packages use homogenous matrices for transformation. That is also true in PEX. PEX programmers do not need to know how those matrices are generated and operated. They just need to specify, for example, moving along the X axis by 5 pixels, rotating around the Y axis by 30°, etc. PEXlib will do all the rest.

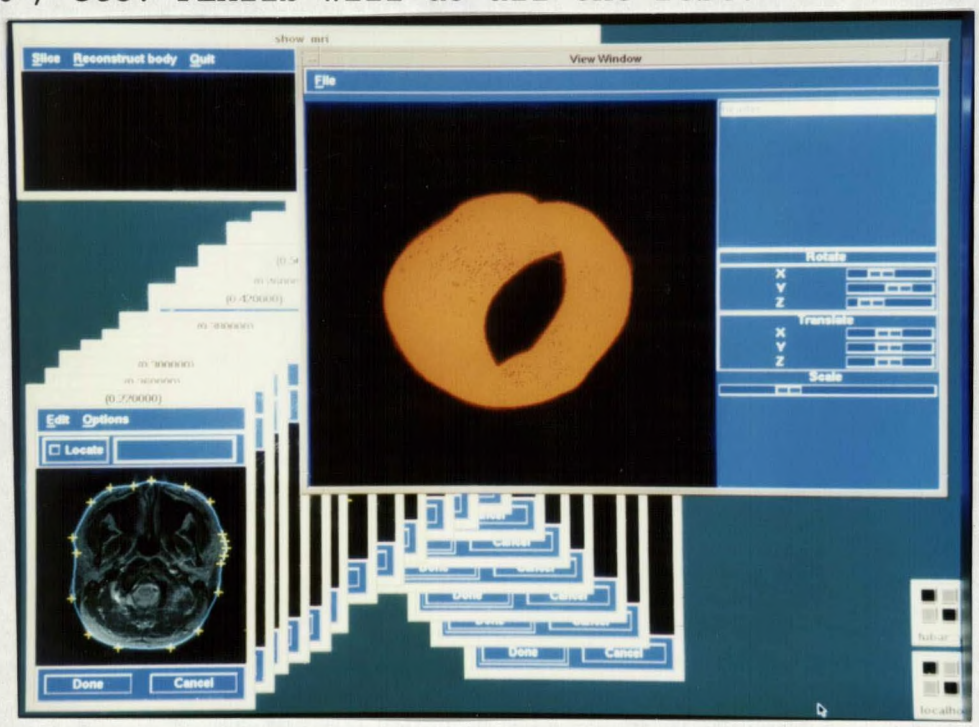


Figure 23. Reconstructed Dog Head

CHAPTER 5

CONCLUSIONS

The major goal of this thesis project was to develop a software system to reconstruct three dimensional bodies from a serial of 2D images with fast rendering and easy representation. This goal was achieved. A package with a friendly Motif-style Graphic User Interface was developed. Users can do most jobs with simple clicking of the mouse, with a few exceptions where the keyboard must be used. This friendly GUI makes it possible for people without computer science backgrounds to master this software quickly to reconstruct 3D bodies for their own purposes.

Several associated image processing methods were included for the users to change and enhance the 2D image provided, which makes it convenient for the users to process images of different styles and quality.

Fast reconstruction of 3D bodies with B-spline in PEX environment was used. Users can transform the 3D bodies in the screen by different orientations in a very short time. Also, the B-spline nature makes the data

structure to represent those 3D bodies quite small (We just need restore the control points of the B-spline instead of all the points of the B-spline).

Examples of how a dog head was reconstructed from 2D images was used in this thesis to show the ability of this software. But the software can also be used in other fields.

Future Research

Due to the time limitations and some problems on our new PEX Software, there are some functions which have not been done, but can be included in the future.

More image processing may be developed later for the users with special requirements. Also, an auto contour generation method will be useful to help define the control points of the B-spline contour.

A file input and output function to save the reconstructed 3D bodies will be very useful when this program is in real operation, because it is quite time consuming to reconstruct 3D bodies.

In this program, Interpolation was used to represent

the 3D bodies with B-spline curves. Other methods can be developed for better results.

REFERENCES

[1] DeFanti, Thomas A. and Brown, Maxine D., "Visualization Expanding Scientific and Engineering Research Opportunities" Visualization in Scientific Computing. IEEE Computer Society Press, 1990

[2] Gonzalez, Rafael C. and Woods, Richard E., Digital Image processing. Addison-Wesley Publishing Company, Inc., 1992

[3] Tiller, Wayne "Rational B-splines for Curve and surface Representation" IEEE Computer Graphics & Applications, September 1983

[4] Grakins, Tom, PEXlib Programming Manual, O'Reilly & Associates, Inc., 1992

[5] X Toolkit Intrinsic Programming Manual, O'Reilly & Associates, Inc., 1992

[6] Motif Programming Manual, O'Reilly & Associates, Inc., 1992

[7] Foley, James D., van Dam, Andries, Feiner, Steven K., Hughes, John F. and Phillips, Richard L., Introduction to Computer Graphics, Addison-Wesley Publishing company, 1994

Garcial Stubbering for plates

158-181