Interface for 4D Ultrasound Application

[redacted]

Abstract – This senior design project involved creating an improved software interface for an existing 4D ultrasound analysis application. The original application, provided by Professor Dove in the Department of Biomedical Engineering, was written in C and OpenGL with a basic interface implemented using the OpenGL Utility Toolkit (GLUT). Limitations of this basic interface included awkward drag and drop file opening, a non-resizable application window, the need to right click for a menu, and graphically primitive slider controls. The improved interface designed for this project used C++ and MFC to create a standard Windows application interface. Features of this interface included menu and toolbar file opening, completely resizable application window, standard Windows menu system, and toolbar-based slider controls.

1 Introduction

Any software application is only as useful as its interface allows. A well-designed interface is both intuitive and functional, allowing the user to perform complex tasks with minimal instruction or effort. In contrast, a poorly designed interface confuses or frustrates the user, requiring expert knowledge and clumsy execution of tasks. Unfortunately, interfaces for many OpenGL applications fall into the latter category, as the OpenGL Utility Toolkit (GLUT) provides a quick yet crude method for OpenGL interface design. OpenGL programmers often revert to using GLUT, thus sacrificing usability for timeliness in the creation of their applications. One such OpenGL application is a 4-dimensional ultrasound analysis program developed in Professor Dove's laboratory in the Department of Biomedical Engineering. This program, written in C language with OpenGL command calls, has a fully functional yet clumsy and non-intuitive GLUT interface. With this application, the typically simple matter of opening a file is complicated by the need to either drag an appropriate file onto the executable icon or type a known file name into an MS-DOS prompt. Additional annoyances include an inability to resize the application window and the need to right click to bring up a menu of commands. Also disappointing is the overall appearance of the application, which includes a set of primitive and distracting slider controls attached to the main window. With these limitations in mind, the goal for this project was to design an improved interface for this existing 4D ultrasound analysis application. The new interface, written with C++ and MFC, would provide standard Windows file opening and menu bars as well as a resizable window and more appealing slider controls. The MFC interface, with aforementioned improvements, would turn a good program with an awkward interface into an intuitive and useful application.

Achieving this project goal was a multi-phase process of learning MFC, understanding the original C, OpenGL, and GLUT code, preparing an OpenGL-ready MFC application template, modifying this template to display the original ultrasound images, and adding additional functionality with MFC. This process required a variety of resources, including several MFC and OpenGL books, the MSDN development library, an online tutorial, and the Microsoft Visual C++ development environment. The remainder of this paper will detail first the specific resources and tools used for the project and then the actual processes and functions used to meet the project goals. Finally, the conclusion will summarize the initial goals and final result, with acknowledgement to supporters of the project.

2 Tools

Each phase of this project required specific programming languages, learning references, existing programs or development tools. This section describes all of these resources in detail, providing a rough outline for the strategy followed in completing this project.

2.1 MFC Programming

The bulk of new code for this project was written using the Microsoft Foundation Class library (MFC), Microsoft's C++ class library for Windows programming. MFC version 6 contains about 200 classes, which can be used either directly or as base classes for other derived classes. An MFC program creates objects from MFC classes and calls member functions belonging to those objects. In addition, MFC helps define the structure of an application and handles many routine tasks automatically.

Three reference books were used to learn about MFC programming for this project. Proceeding from the least to most advanced, these books were *MFC Programming from the Ground Up* by Herbert Schildt, *Programming Windows with MFC Second Edition* by Jeff Prosise, and *The MFC Answer Book* by Eugene Kain, all pictured in Figure 1 below.

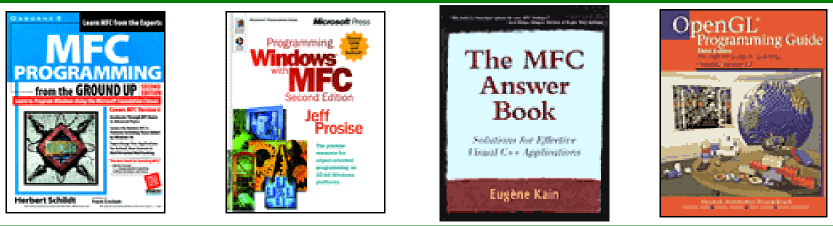


Figure 1: MFC and OpenGL/GLUT Programming References

2.2 Original Program

One obviously indispensable resource for the completion of this project was the original 4D ultrasound analysis program to be modified. Two screen shots demonstrating the appearance and functionality of this original program are included in Figure 2 below. As previously stated, this program uses the C language with OpenGL calls, including functions in the OpenGL Utility Toolkit (GLUT) for the interface design. More specifically, the application consists of twenty four .c files with C/OpenGL functions and nine .h header files containing function prototypes, variables, and structures. The main function is included in GL3us.c and roughly 200 global variables used throughout the program are included in globals.h. Most important to this project were the five .c files containing calls to GLUT functions: GL3us.c, Init.c, Display.c, Interface.c, and Mouse.c. All GLUT calls in these functions had to be replaced with equivalent MFC code in order to create a functional MFC interface for the existing application. The reference used to understand the purpose of these GLUT calls as well as any necessary OpenGL code was the *OpenGL Programming Guide*, pictured in Figure 1 above.

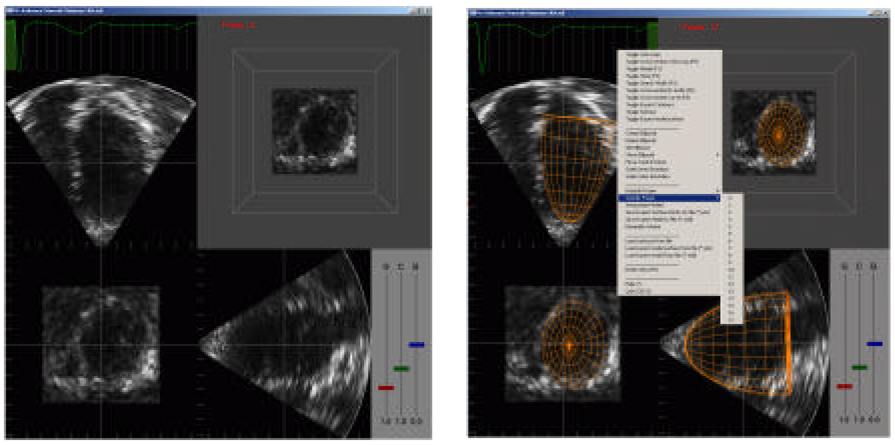


Figure 2: Original 4D Ultrasound Application

Nearly as important as the understanding of GLUT calls for replacement with MFC was the understanding of the screen coordinate system used in the original program for modification in the new application interface. By reading the original C/OpenGL code, particularly in the file

Display.c, the coordinate system shown in Figure 4 below was uncovered. Global labels for the six window areas are shown in yellow, while global variables for area widths and heights are shown in blue text.

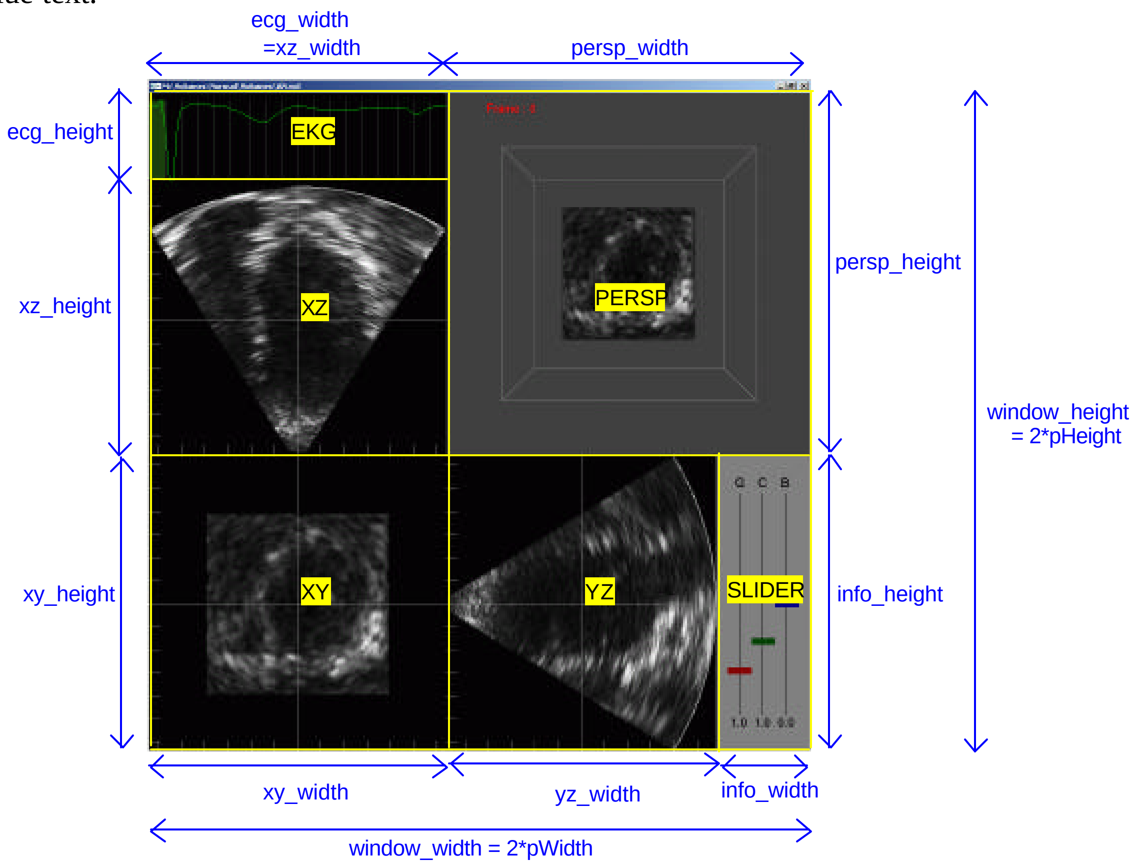
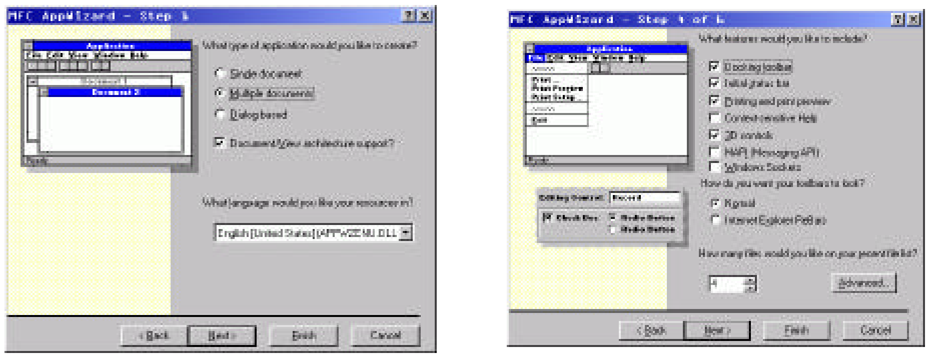


Figure 3: Original Program Screen Layout

2.3 Application Development

The starting point for the MFC interface design was to create an initial MFC application template consisting of a skeleton MFC program with an empty window display. The tool used to create this template was the Microsoft Visual C++ MFC AppWizard. MFC AppWizard, as shown in Figure 5 below, takes the user through a series of options to create an initial application. The starting template created for this project is shown in Figure 6 below.


MFC AppWizard - Step 1

What type of application would you like to create?

- ( ) Single document
- (•) Multiple documents
- ( ) Dialog based

[✓] Document/View architecture support?

What language would you like your resources in?

English [United States] (APPWZENU.DLL)

MFC AppWizard - Step 4 of 6

What features would you like to include?

- [✓] Docking toolbar
- [✓] Initial status bar
- [✓] Printing and print preview
- [ ] Context-sensitive Help
- [✓] 3D controls
- [ ] MAPI (Messaging API)
- [ ] Windows Sockets

How do you want your toolbars to look?

- (•) Normal
- ( ) Internet Explorer ReBars

How many files would you like on your recent file list?

4

Figure 4: MFC AppWizard

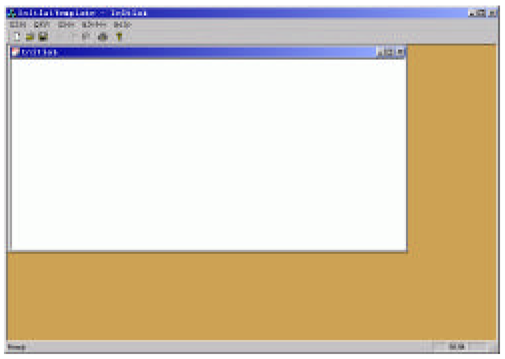


Figure 5: Initial Application Template

Once the initial MFC template was created, it needed to be enabled for OpenGL programming. Two resources were instructive in this enabling process: the *OpenGL I: Quick Start* tutorial in the MSDN Visual Studio 6.0 Library, and the online tutorial *Using OpenGL in Visual C++ Version 4.x* by Alan Oursland located at http://devcentral.iftech.com/learning/tutorials/mfc-win32/opengl/. After enabling for OpenGL the application still appeared exactly as in Figure 6 above.

Many other modifications to the MFC application were made in Microsoft Visual Studio either by hand or using the Class Wizard. Two different menus for the Class Wizard are shown in Figure 7 below. As these menus suggest, the wizard can help modify an existing class by adding variables or functions or can ease the work involved in creating a new class. All of these Class Wizard features were used during the course of this project.

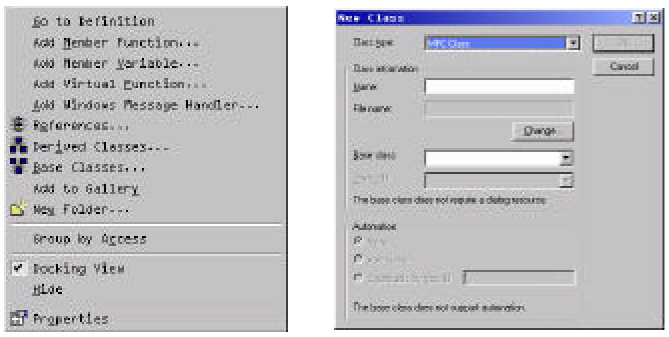


Figure 6: MFC Class Wizard

The final tools used in the MFC application interface design were the various resource editors included with Visual Studio. Two such resource editors are shown in Figure 8 below. These editors allow for simple editing of resources such as menus and toolbars. While these editors can simplify modifications to the appearance of resources, they cannot add functionality. Instead, this must be accomplished through the usual MFC message handling system.

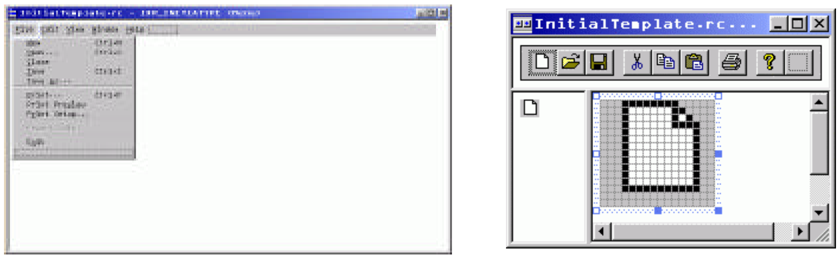


Figure 7: Menu and Toolbar Resource Editors

3 Results

Once the initial phases of learning MFC and understanding the original C, OpenGL, and GLUT code were completed, the next steps involved preparing an OpenGL-ready MFC application template, modifying this template to display the original ultrasound images, and adding additional functionality with MFC. These steps are discussed in the following subsections.

3.1 Application Template

Creating an OpenGL-ready MFC application template involved using the MFC AppWizard described in section 2.3 to generate a generic MFC template that followed the MFC document/view architecture as a multiple document interface (MDI) application. The document/view architecture is simply a programming model that separates an application's data (document) and appearance (view) into separate classes. An MDI application is an extension of this model where many views are allowed for one program. The 4D ultrasound analysis program seemed well suited to the MDI model given that future program updates could include separating the various plane ultrasound images into separate view objects. Thus following this model the AppWizard created the five basic MFC derived classes shown in Table 1 below. Enabling this basic application for OpenGL code required creating and overriding several functions in the view class. Following the tutorials listed in section 2.3, two new functions were written to set the window pixel format and create a rendering context for OpenGL. Additionally, the message handling functions OnCreate(), OnDraw(), and OnDestroy() were modified for displaying OpenGL graphics.

Derived Class	Base Class	Modifications for OpenGL
CMy4DUltrasoundApp	CWinApp	
CMy4DUltrasoundDoc	CDocument	

CMy4DUltrasoundView	CView	SetWindowPixelFormat(HDC hDC)
		CreateViewGLContext(HDC hDC)
		OnCreate(LPCREATESTRUCT lpCreateStruct)
		OnDraw(CDC* pDC)
		OnDestroy()

CMainFrame	CMDIFrameWnd	
CChildFrame	CMDIChildWnd	

Table 1: Initial Application Template Classes

3.2 Display Window

Once an OpenGL-ready MFC template existed, the next challenge was to open a file and display its contents in the same format as for the original C/OpenGL program. Typically, opening a file in a document/view application involves overriding the OnOpenDocument()

function in the document class and initializing the member variables of that class for display. This project involved a slightly modified approach to the typical solution as code for file opening and display was already included in the original C files. Replacing all of this code would have required rewriting a large portion of complex code and eliminating hundreds of global variables from the original program design. Thus for simplicity the original code was retained and all relevant functions were called as external C functions from the new MFC classes. A listing of MFC functions used for the window display, along with relevant C function calls and a description of functionality, is included in Table 2 below.

Class	MFC Function	C Function Call	Functionality
Document	OnOpenDocument	readimage()	Reads in file data

Document	OnOpenDocument()	varinit()
		persp_init()
		reset_view()
			Initialize C global variables

Child Frame	OnCreate()	varinitMFC()	Set initial position and size of window
View	OnDraw()	display()	Draws all window images
View	OnSize()	varinitMFC()	Sets C global variables based on window size
View	OnEraseBkgnd()		Eliminates white background flicker
Child Frame	OnWindowPosChanging()		Lock sizable aspect ratio

Table 2: Functions for Window Display

Together, the functions in Table 2 comprise a rather complex system for displaying the ultrasound images in the application window. Several unexpected problems were encountered and solved during this programming process. The first major problem was that part of the original display image was chopped off in the MFC window. The source of this error turned out to be a discrepancy between the coordinate systems of MFC and OpenGL. OpenGL draws images starting from the lower left hand corner of a given area, while MFC displays from the upper left hand corner. This mismatch was solved by writing a new version of the C function varinit() called varinitMFC(), that modified the C global variables for appropriate display.

The other major display problem caused by the switch from OpenGL to MFC was in window resizing. The original C/OpenGL program did not allow for any window resizing, but MFC provides resizing by default. The problem encountered was that the window display was created as a determinate square that would not expand upon resizing. Thus it was possible to resize a window and show an unattractive blank gray window background. The final solution to this problem was to override the OnWindowPosChanging() function and set the window width to be always equal to the window height. Thus the window could still be resized, but with a fixed width to height ratio, eliminating the possibility of a visible blank portion of the window. Screen shots from before and after this resizing problem was fixed are included in Figure 9 below.

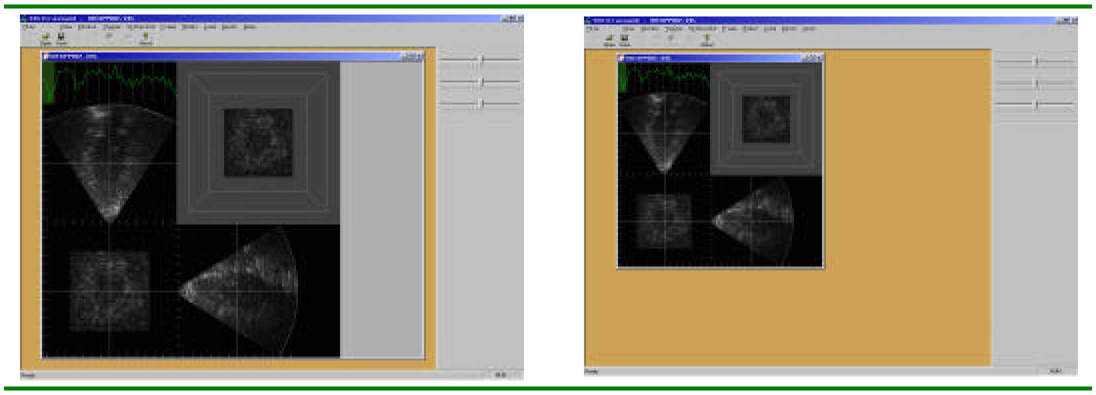


Figure 8: Window Resizing Problem – Before and After

Once all display problems were fixed, the new MFC interface performed as desired in terms of both file opening and window rendering. With the new MFC interface, not only was the display now resizable, but files could be opened by three different methods: choosing file – open on the menu bar, clicking the open button on the toolbar, or selecting one of four recently opened files in the most recent file list on the menu. Additional screen shots of this file opening functionality are included in Figure 10 below.

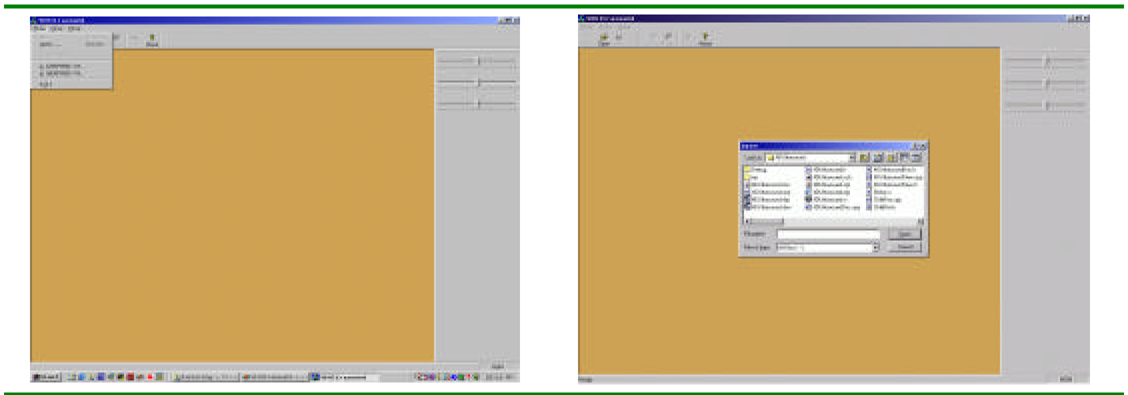


Figure 10: File Opening Options

3.3 Additional Features

Several other features, both minor and major, were added to the MFC interface to improve functionality. One minor improvement was to change the cursor style to the "wait" cursor as a file was being read. Depending on computer speed, the ultrasound .VOL file could take nearly thirty minutes to read in. Thus the "wait" cursor, implemented by calls to BeginWaitCursor() and EndWaitCursor() surrounding the readimage() call, indicated that some process was taking place.

A second minor improvement was adding labels to the main toolbar buttons. Comparing the toolbar generated by AppWizard in Figure 5 with the final toolbar in Figure 10 shows these additional labels. These labels were added by calls to the SetButtonText() member function of the toolbar class in the CMainFrm class.

The final and most significant improvement to the MFC interface was in the addition of a new set of slider controls to replace the original sliders visible in Figure 2. The new sliders are present in a dockable and undockable toolbar as shown in Figure 11 below. The implementation for this new toolbar required the derivation of a new class, CSlidersToolBar, derived from CToolBar. This class then contained three CSliderCtrl member variables representing the three slider controls on the toolbar. Finally, the toolbar was created in the OnCreate() function of CMainFrm, just as the original toolbar had been created with the MFC AppWizard.

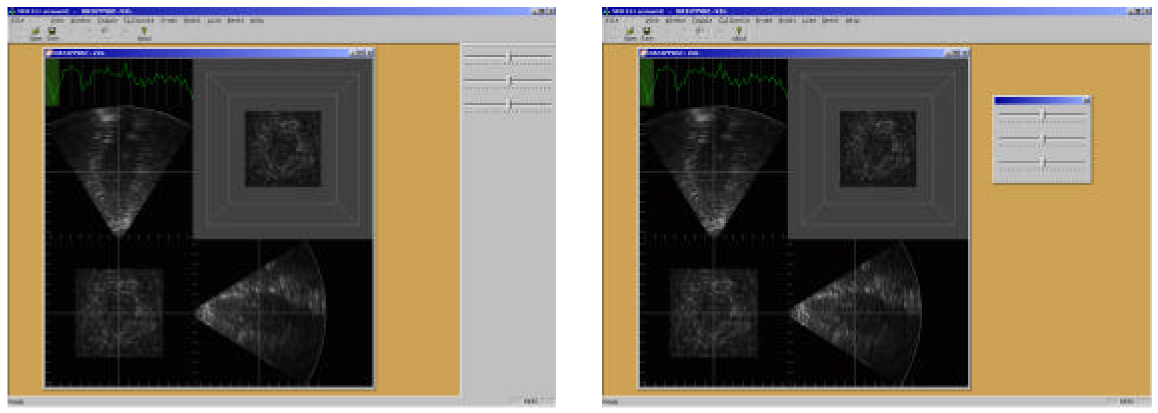


Figure 11: Dockable/Undockable Slider Control Toolbar

4 Conclusion

The goal for this senior design project was to develop an improved software interface for an existing 4D ultrasound analysis program. The original application, provided by Professor Dove in the Department of Biomedical Engineering, was written in C and OpenGL with a basic interface implemented using the OpenGL Utility Toolkit (GLUT). Limitations of this basic interface included awkward drag and drop file opening, a non-resizable application window, the need to right click for a menu, and graphically primitive slider controls. The improved interface designed for this project used C++ and MFC along with the original C and OpenGL code to overcome these functionality limitations. A variety of resources were required for the development of this interface, including several MFC and OpenGL reference books, the MSDN Development Library, an online tutorial, and a number of wizards and tools from the Microsoft Visual C++ 6.0 development environment. The final MFC interface design improved on the original application in several ways. File opening was modified to allow for opening of files from the menu open option, a toolbar open button, or a most recent files list. Additionally, the main window display was modified to allow for resizing with a locked aspect ratio. Finally, a standard Windows menu system was implemented and a new dockable/undockable toolbar was created with improved slider controls. Altogether, these design improvements turned a promising program with a confusing and awkward interface into a professional, intuitive, and useful application.