

**VISUALIZATION TECHNIQUES FOR A
PROTOTYPE ANTHROPOMETRIC DATABASE**

by

Adwait Ashok Joshi

February 2004

A thesis submitted to the Faculty of the Graduate School
of The State University of New York at Buffalo in partial
fulfillment of the requirements for the degree of

Master of Science

Department of Mechanical and Aerospace Engineering

ACKNOWLEDGEMENTS

The written word has an unfortunate tendency to degenerate genuine gratitude into stilled formality. However, this is the only way we have to permanently record our feelings.

I would like to express my deep sense of gratitude towards Dr. Roger Mayne, my advisor, for his constant support and inspiration. He was a constant source of new ideas during the early programming stages of this undertaking. I would also like to express sincere appreciation towards Dr. Victor Paquet for giving me an opportunity to work on this project and assisting in preparation of this manuscript.

In addition I would like to thank the members of the R1 team and the IDEA center staff whose help has been instrumental in the completion of this thesis. And finally I would like to apologize to all I must have forgotten to mention and who have helped me during this endeavor.

This research was partially supported with a grant provided by the Department of Education and National Institute on Disability and Rehabilitation Research through the Rehabilitation Engineering Research Center on Universal Design at Buffalo (Grant # H133E990005). The opinions expressed in this paper are those of the author and do not represent those of the Department of Education or those of the National Institute on Disability and Rehabilitation Research.

TABLE OF CONTENTS

LIST OF FIGURES.....	i
ABSTRACT.....	iii
GLOSSARY.....	v
1. INTRODUCTION.....	6
2. BACKGROUND.....	8
3. THE ANTHROPOMETRIC STUDY AND DATA BASING.....	11
3.1 Structural Measurements or Human Body Sizes.....	11
3.2 Functional Measurements.....	12
3.2.1 Grip Strength.....	12
3.2.2 Pinch Strength.....	13
3.2.3 Grip Span.....	14
3.2.4 Reach and object moving tasks.....	15
3.2.5 Maneuverability.....	17
3.3 Database design.....	19
4. DEVELOPMENT.....	21
4.1 Overview of AnthroDB.....	21
4.2 Functional details of AnthroDB.....	23
4.2.1 Changing the Screen Resolution.....	23
4.2.2 Splash Thread.....	25
4.2.3 Frames.....	27
4.2.4 Top Left Form.....	29
4.2.5 Top Right Form.....	36

4.2.6 Bottom left form.....	43
4.2.7 Bottom right form (The View class).....	44
4.2.8 The document class.....	55
4.2.9 Dialogs.....	63
4.2.10 Menus and Toolbars.....	69
4.2.11 Menu Navigation.....	70
4.2.12 Other functionality.....	71
5. WORKING	75
6. CONCLUSIONS	80
7. FUTURE WORK	82
APPENDIX A - Landmarks and Associated Considerations (Feathers, 2003).....	84
APPENDIX B - Anthropometric Measurements	89
APPENDIX C – Approximate location of some key landmarks	90
BIBLIOGRAPHY	91
INDEX.....	92

LIST OF FIGURES

Figure 1 : Structural Protocol.....	12
Figure 2 : Measuring grip strength using a grip dynamometer	13
Figure 3 : Measuring pinch strength using a pinch dynamometer - Lateral Pinch and Thumb Forefinger pinch	14
Figure 4 : Measuring grip span using a grip span cone	15
Figure 5 : Reach protocol - 0 degree reach and 90 degree reach.....	17
Figure 6 : Level Maneuverability - Straight Wall.....	18
Figure 7 : Level maneuverability - L Turn	18
Figure 8 : AnthroDB : A typical case.....	22
Figure 9 : Resolution change dialog.....	24
Figure 10 : Splash screen	25
Figure 11 : Top left form - Demographic Data	33
Figure 12 : Top left form - MSFlexGrid	36
Figure 13 : Top right form - Structural dimensions	37
Figure 14 : Top right form - 2D Reach graph.....	39
Figure 15 : Top right form - Multiple 2D graphs	39
Figure 16 : Top right form - 2D histogram	43
Figure 17 : Bottom left form - Distances from reference points	44
Figure 18 : Structural protocol - Front view.....	54
Figure 19 : Structural protocol - Side view	55
Figure 20 : Functional protocol - Reach envelopes.....	55
Figure 21 : Population characteristics dialog	64

Figure 22 : Data category dialog	64
Figure 23 : Select population dialog.....	65
Figure 24 : Select dimensions dialog.....	66
Figure 25 : Select reach dialog.....	66
Figure 26 : Progress dialog	68
Figure 27 : AnthroDB parent window	75
Figure 28 : AntorDB showing statistical data.....	76
Figure 29 : AnthroDB showing structural data.....	77
Figure 30 : AnthroDB showing reach data.....	78
Figure 31 : Reach data with all the 4 reach envelopes.....	79

ABSTRACT

This thesis deals with visualization of anthropometric data collected on wheelchair users for design of environments conforming to the concept of Universal Design. A computer package has been developed which would provide designers, architects and ergonomists with an interface to perform various analyses on the anthropometric data.

Traditional anthropometric measurements include use of anthropometers, a graduated rod with a sliding edge at a right angle and a graduated scale, to measure body dimensions. Most recent technologies include the use of laser technology to measure body points with an obvious disadvantage of not being able to measure body points not visible to the scanner. In our research project we make use of various instruments such as anthropometer, grip and pinch dynamometer, grip span cone and an electromechanical 3-D digitizer to measure the participant data. This data is then stored in a database so that it can be queried as required.

This thesis employs different types of visualization techniques for the anthropometric data. These techniques deal with the visualization of body and wheelchair sizes and participant's capabilities to perform different tasks. The interface provides one dimensional data in the form of a spreadsheet, two dimensional data in the form of histograms and three dimensional data in the form of a structural human model and reaching envelopes. It gives the users tremendous flexibility by allowing them to choose specific dimensions and variables. For example, users have an ability to import their 3D environments modeled with the help of 3D studio max, to illustrate the fit between their models and the 3D characteristics of wheelchair users, or export anthropometric data to excel for further statistical calculations. Due to the complex nature of AnthroDB, it employs menu navigation

to help the users make correct selections. The users can classify data on the basis of disability and demographic classification. A combination of all such techniques, a capacity to display audio visual data in the form of pictures and videos, individual data in the form of graphs and three dimensional rendering and statistical data in the form of histograms makes this computer package extremely powerful.

GLOSSARY

AnthroDB – Name of the package

Anthropometer – A graduated rod with a sliding edge at a right angle

Bony Landmarks – Landmarks on the human body which are measured in the structural protocol

Data type - Individual data or statistical data

DC – Device Context

DSN – Data Source Name

Functional study – A study related with a participant's capability to do a specific task

Grip Dynamometer – Equipment used to measure the grip strength

Grip span cone – Equipment used to measure the grip span

IDE – Integrated Development Enterprise

Individual/Participant – The subject on whom the study is carried out

MDI – Multi Document Interface

MFC – Microsoft Foundation Classes

Mid Sagittal Plane – Plane in which a human body can be symmetrically divided. i.e. a plane passing vertically from the head and perpendicular to the shoulder line

Pinch Dynamometer – Instrument used to measure pinch strength

Protocol – Type of study, structural or functional

R1 – prototype anthropometric database project

RC – Rendering Context

Structural study – A study related with the body sizes

User – The person using AnthroDB

SQL – Structured Query Language

1. INTRODUCTION

Anthropometric data provide the most rudimentary information needed to design fitting equipment and workspace environments. Body dimensions and functional abilities differ across people of different sex, different origin, different ages etc. Population or statistical data ranges of subjects, provides us with a wealth of information that can be used in the designing of environments. It is very important to be able to focus on data for populations as well as data for individuals that may fall in the extremes of the distribution of body sizes and function. This provides full flexibility to decide who should be excluded from the design or who should be included in it. For example a designer may attempt to accommodate a very narrow or very broad range of population.

Dave Feathers, a research assistant on the prototype anthropometric database project (R1 project), has been involved in developing and evaluating anthropometric methods that record 3-dimensional coordinates of body dimensions and the collection of dynamic as well as static anthropometric data that will be useful for universal design and support computerized modeling of wheelchair users (Feathers, 2002). This thesis mainly deals with the visualization of anthropometric data collected on wheelchair users with these methods. The structural information deals with individual body dimensions such as sitting height, eye height, bi-deltoid breadth, etc. The functional information deals with an individual's ability to do specific tasks. It consists of reaching abilities at different angles (with respect to his mid-sagittal plane; the plane in which the human body can be symmetrically divided), different weights and different heights along with the participant's grip and pinch strength, grip span and maneuverability. These data, when used in conjunction with structural data, provides us

with valuable information needed for design of environments. For example, a designer might want to see how far a person can reach in front of his body holding a particular weight and at a particular height, or he might want to look at a percentage of people that can do a particular task. The designers have an ability to query specific information from the database pertaining to human body dimensions. Since all the information is calculated on the fly, it caters to user specific requirements. The idea of this thesis is to develop a computer package that will help designers, architects and engineers interact with anthropometric in a very efficient and useful manner.

2. BACKGROUND

Anthropometry is the science that deals with measurement of the height and other dimensions of human beings, especially at different ages, or in different races, occupations, etc. It evolved from physical anthropology, in which the bones of the human body are measured and compared. Engineers are very much interested in applying anthropometric information to design, in order to match the body dimensions of people to the dimensions of the physical environment in which they perform tasks.

Structural body measurements are conventionally defined by two endpoints of the distance measured (Kroemer, *et al.*, 1994). For e.g. Bi-acromial breadth would be the distance between the acromion processes that are bony landmarks of the shoulders. The terms used in classical anthropometry are

Height – is a straight line, point to point vertical measurement

Breadth – is a straight line, point to point horizontal measurement running across the body or a segment

Depth – is a straight line, point to point horizontal measurement running fore and aft the body

Distance – is a straight line, point to point horizontal measurement between landmarks on the body

Curvature – is a point to point measurement following a contour, this measurement is usually neither closed nor circular

Circumference – is a closed measurement that follows a body contour, hence this measurement is not circular

Reach – is a point to point measurement following the long axis of the arm or leg

Classical measurement techniques include the Morant Technique, which makes use of a set of grids, usually attached to the inside corner of two vertical walls meeting at right angles (Kroemer, *et al.*, 1994). The subject is placed in front of the grids and projections of the body onto the grids are used to measure anthropometric variables. The software developed for this thesis, AnthroDB, makes use of the same concept to come up with the clipping plane feature in the reach study to calculate intersection points with the reach envelopes. Traditional measuring devices include an anthropometer, a graduated rod with a sliding edge at a right angle. The anthropometer provides a good approximation of distances. The grip strength and the pinch strength of an individual can be measured with the help of a grip and pinch dynamometer. Measurements such as grip span (the diameter around which the fingers can close) can be measured with the help of a tapered cone marked with readings. The Faro Arm is an electromechanical measurement device with 6 degrees of freedom. It provides 3D coordinate information rather than only dimensional information such as length, breadth etc. With the 3D information available in the form of X, Y and Z coordinates, it becomes easy to calculate all the dimensions on the fly. With the absolute coordinates of the bony landmarks, it is easy to render a 3D human figure without having to make any approximations of data points.

Most of the classical measuring techniques leave many of the body dimensions unrelated to each other in space. For example, when looking from the side, a person's stature, the eye height and the shoulder height are located in different undefined frontal planes. In the past, there has been limited effort to effectively visualize the anthropometric data. Data have been

displayed in the form of tables and graphs which is purely one and two dimensional information, and hence incomplete. Designers have even considered using photographic information, with the obvious drawback of lacking a third dimension. They have explored the areas of laser technology to efficiently capture and visualize the data with the fact that it cannot scan points that are not visible to the probe.

The idea of developing this computer package is to provide the user with a tool to interact with anthropometric data in an open ended manner. The unique combination of being able to interact with the virtual human being, interact with virtual reaching values as well as to be able to do the traditional tasks such as plotting histograms and carry out statistical analyses makes it powerful. This is a very important contribution to the field of anthropometry. Such measures allow realistic human models to be visualized directly and allow designers to understand how body postures (and not just body segment sizes) affect the amount of space required in activities of daily living. The different measurement protocols involved in this study are described in chapter 3.

3. THE ANTHROPOMETRIC STUDY AND DATA BASING

The overall idea of the research project R1 – Prototype anthropometric database project, (RERC on the Universal Design in the build environment at Buffalo) is to develop a prototype anthropometric database of large sample wheelchair users to serve as a model for anthropometric research in universal design. The study includes the development and evaluation of anthropometric methods that record 3-dimensional coordinates of body dimensions, collection of dynamic (functional), as well as static anthropometric data, the design of a database that will be useful for universal design, and support the development of the computer package capable of handling the 3D modeling of the wheelchair users. The different protocols involved in the anthropometric study are described below.

3.1 Structural Measurements or Human Body Sizes

In our study of wheelchair users the size of the chair as well as the individual is covered. There are basically three domains that we cater to: human modeling, design, and posture assessment (Feathers, 2003). Three-dimensional anthropometric measurements do not have exactly the same formal anthropometric definitions as the corresponding one- or two-dimensional measurements (Feathers, 2003).

The structural protocol involves capturing of over 150 3-dimensional locations on the participant's body and the wheelchair. The participant is generally oriented facing the Faro Arm and wheelchair locked, such that his mid-sagittal plane aligns with the YZ plane of the faro arm. The global coordinate system for each participant is slightly different because of the relative position of the participant and wheelchair with respect to the Faro Arm is

different across individuals. However, AnthroDB makes the necessary corrections (translations and rotations) so that the measurements can be compared across individuals or summarized for groups of individuals. Figure 1 shows a participant when his structural body dimensions are being measured.



Figure 1 : Structural Protocol

3.2 Functional Measurements

Functional measurement can be defined as a participant's ability to accomplish a specific task. The different type of functional measurements are described below

3.2.1 Grip Strength

The traditional grip strength, measured by the grip dynamometer, is defined as (Feathers, 2003) the maximum voluntary average grip force obtained over three trials with a hand dynamometer.

There are two categories in this protocol, first being with the arm out stretched and the other being with the arm flexed at 90 degrees. A total of three measurements are taken with sufficient time interval between them to avoid muscular fatigue, provided there are no outliers within that set (Outliers, in this sense, meaning any reading that is greater/lesser than 5 psi as compared to the other readings).



Figure 2 : Measuring grip strength using a grip dynamometer

3.2.2 Pinch Strength

There are two measurements that are taken for Pinch Strength: Lateral Pinch and Thumb-Forefinger Pinch. The Lateral Pinch tests the thenar musculature of the hand, acting to abduct the thumb (Feathers, 2003). This pinch is performed when the hand is in a neutral position (or with palm facing upwards). It tests the subject's ability to use a precision pinch for picking out objects from a neutral hand position. The thumb-forefinger pinch tests the participant's ability to press one's thumb down toward the forefinger. This abduction tells how strong the participant can pinch an

object while holding it in one hand. All the readings are taken with the help of a pinch dynamometer shown in figure 3.

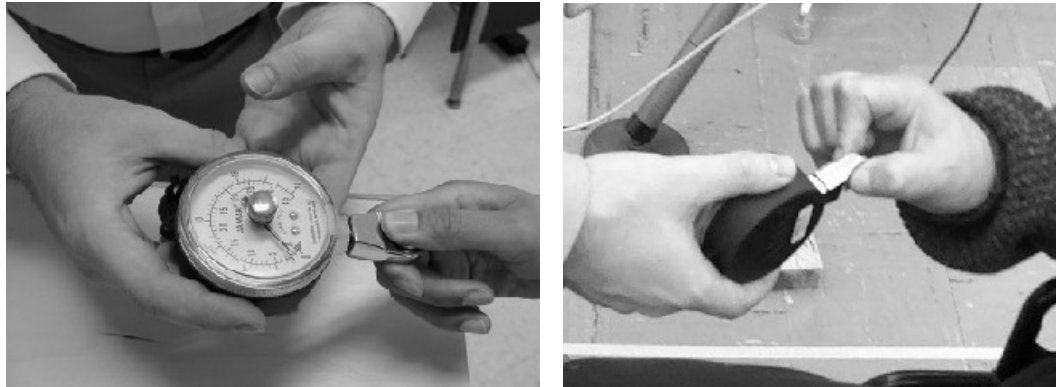


Figure 3 : Measuring pinch strength using a pinch dynamometer - Lateral Pinch and Thumb Forefinger pinch

3.2.3 Grip Span

Grip span is defined as the minimum circumference around which the human fingers can be wrapped (Feathers, 2003). Grip span is measured by an anthropometric grip span cone, shown in figure 4. A total of four readings are taken which consists of Digit I and Digit II; Digit I and Digit III; Digit I and Digit IV; Digit I and Digit V.



Figure 4 : Measuring grip span using a grip span cone

3.2.4 Reach and object moving tasks

Reach is defined as the maximum, unsupported reach from a seated position to a shelf that is placed at one of the five heights (described below) (Feathers, 2003). In our reach protocol, wheelchair users are asked to reach in three directions- forward, at 45 degrees, and laterally at five heights and four weight conditions. The study begins with measuring points on the human body which contribute towards the calculation of shelf heights used in reaching equipment. The four key measurements include

1. Acromion height – This is the height of the shoulder bone of the reaching arm from the floor.
2. Floor to center distance – This is the distance between the floor and the center of the palm of the reaching arm, when the arm is relaxed pointing the floor.

3. Overhead reach – This is the distance between the center of the palm of the reaching arm and the floor when the arm is comfortably stretched above his body, vertically upwards.

4. Arm length – It is the distance between the acromion and the center of the palm of the reaching arm, when the arm is comfortably outstretched, parallel to the ground.

The height of the first shelf is usually equal to the overhead reach, height of the middle shelf is the acromion height, the height of the bottom shelf is the floor to center distance and the second and the fourth shelf heights are calculated off of the four reach dimensions. Once the shelf heights are fixed, the participant is aligned with the faro arm facing his back to it. The shelving unit is then adjusted and placed in front of him, so that the shelves can be moved back and forth depending upon his reaching capabilities. The unit is usually placed at three different angles 0, 45 and 90 with respect to his mid-sagittal plane and the participant is asked to place the weight on a specific shelf height. The weights consist of 0 pound, 1 pound, 3 pounds and 5 pounds. A total of 60 readings are taken in this protocol (3 angles x 4 weights x 5 shelves).



Figure 5 : Reach protocol - 0 degree reach and 90 degree reach

3.2.5 Maneuverability

The maneuverability protocol is split into three sub-elements. These elements are door use (entrance and egress), level maneuverability and transfer maneuverability. Level maneuverability consists of U-turns, L-turns, 360 turns, level propulsion and a K-turn (3-point turn).

In the first task, a set of walls is placed at a specific distance from each other, and the participant is asked to maneuver around the wall as shown diagrammatically below.

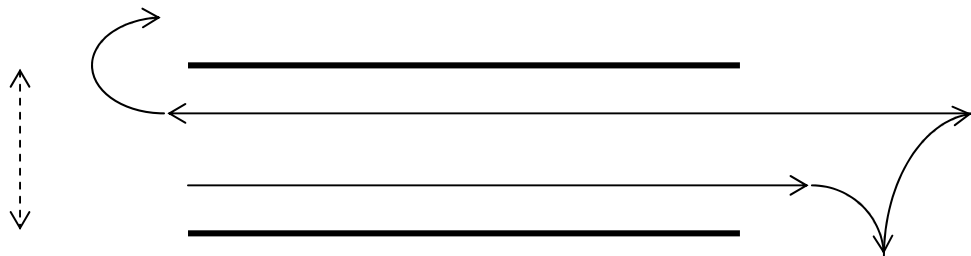




Figure 6 : Level Maneuverability - Straight Wall

The next task in the list is an L-Turn. The wall distance is adjusted (increased or decreased) till the participant comfortably passes through without bumping on to them. The third task is to make a 360 turn on the spot.



Figure 7 : Level maneuverability - L Turn

Door use utilizes three different levels of doors (each separately coded via the Usability Rating Scale and other subjective rating scales. These doors vary in

difficulty in a host of architectural conditions ranging from door thresholds, automatic door closers, wide doors with difficult pull phase, etc. (Feathers, 2003).

3.3 Database design

In the anthropometric study, hundreds of data points are measured which gives rise to a tremendous amount of data to be visualized as numbers. The Anthrocam Package produces a text file with 3-dimensional coordinates of each point measured in space. However this raw data is of no use as it cannot be imported into MS Access directly and neither can it be visualized. An Excel macro (written to cater our needs) helps in clearing unwanted characters from the text file, and does a pick and place so as to format it for MS Access imports. A simple transfer spread sheet command then allows the data to be imported into the MS Access tables. The relational database is categorized into following tables.

1. Demographics

Demographics, as the name suggests, stores the demographic information of the participant such as name, age, gender, handedness, disability type, years with disability etc. This demographic information helps the user select a subset of participant data, for detailed statistical study.

2. Grip and Pinch Strength

Grip and pinch strength tables store the values of the grip and the pinch protocols.

3. Participant Pictures and Videos

Participant pictures and videos, contains the information about audio visual data.

4. Reach Info

Reach info consists of information about a participants acromion height, overhead reach, floor to center distance and arm length used in determining the shelf heights in the reach protocol.

5. Actual Data

Actual Data table stores the 3D coordinate data collected with the Faro Arm which is later used to render a humanoid figure in 3D space and also visualize the reach protocol.

6. Wheelchair Data Sheet

Wheelchair data sheet consists of information about the wheelchair of the participant including parameters such as its make, model, condition, serial number etc.

A DSN driver is used to extract data from the database so as to render it in the package. The next chapter provides an overview of the package and explains in detail some of its functional aspects.

4. DEVELOPMENT

This chapter focuses upon the functional abilities of the computer package. AnthroDB is a windows application, developed with the help of Visual Studio. It provides the user with an interface so as to interact with the anthropometric and biomechanical data.

4.1 Overview of AnthroDB

AnthroDB has been developed using Visual C++ 6.0 and .net 2002 IDE's for providing the framework. The 3D rendering is carried out with the help of OpenGL. Visual studio interfaces very well with MS Access databases and makes the data retrieval easy. It also provides standard windows functionality by providing predefined classes and objects. OpenGL is platform independent and hence the part of the visualization that uses OpenGL can be used to develop multi platform applications.

Incorporating OpenGL in the Visual C++ environment is challenging. The default implementation of Visual C++ only supports DirectX as the primary renderer. In order to enable OpenGL there is a strict set of rules to be followed which are described in chapter 4.2.7.

AnthroDB has different modules for each protocol. There are 4 static window frames which control different views. A typical case is shown in figure 8.

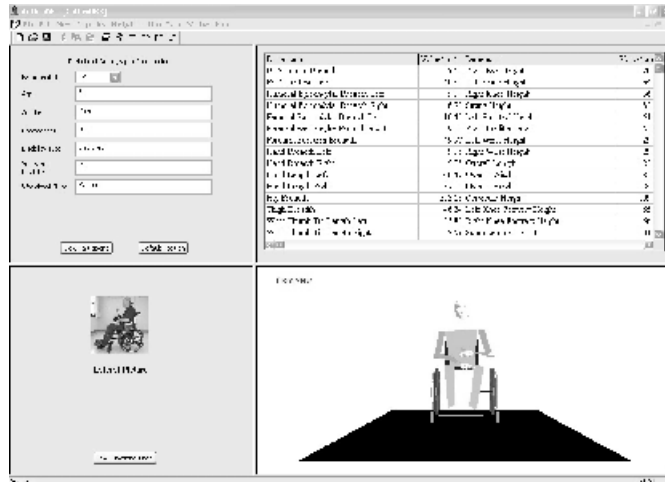


Figure 8 : AnthroDB : A typical case

AnthroDB makes use of various dialogs for navigational purposes. It supports all of the standard functionalities such as File I/O, Multi Document Interface (an implementation of multi threading), keyboard and mouse shortcuts etc. It also makes use of ActiveX components to support audio visual data, allow the use of an excel worksheet etc. Advance functionality includes exporting the data to an excel worksheet, importing of 3DS Max objects into the scene etc. Most of the designers and architects use 3DS Max for generating environments. With the ability to import these scenes, AnthroDB eliminates the need for conversion of the 3DS Max file to a universal format or making use of external libraries.

In the following section of the manuscript, all of the functionality of AnthroDB has been described in detail. The source code is accompanied with a detailed description of its purpose and working.

4.2 Functional details of AnthroDB

Document/View Architecture

The document/ view architecture is the backbone of AnthroDB. Functions and variables that are called from within other classes are a part of the document class. If a function or a variable is a member of the document class, it can be called from anywhere by declaring the appropriate document class pointer (shown below).

```
CAnthroDBDoc* pDoc = GetDocument();
```

Another purpose of the document class is the allocation of memory for the pointers. Since the document class constructor is called before anything else, the memory allocations and variable initializations are carried out here.

In MFC, it is mandatory to specify an active frame when using the window splitter. An active frame is the one from where all the menu functions are accessible. In this case it is the rendering window. The view class mainly handles the rendering, and hence controls this window. All the event handlers for events such as, mouse clicks, keyboard buttons, menu clicks, are defined in the view class.

4.2.1 Changing the Screen Resolution

When an instance of AnthroDB is first invoked, it checks if the user's computer is using the correct resolution. In order to better view the interface, an 1152 x 864 resolution is recommended. AnthroDB first checks the current resolution of the screen and stores the value in a member variable so that it can be restored later. If

the resolution is below 1152 x 864 it prompts the user to change it. The dialog for the resolution change is shown below

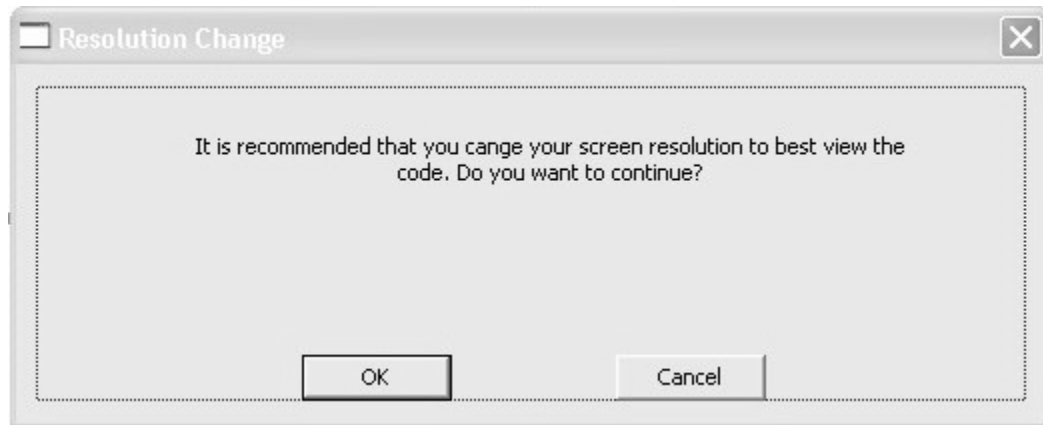


Figure 9 : Resolution change dialog

The code for resolution change is given below.

```
CResolutionChange DialogWindow;
DEVMODE dm;
dm.dmSize = sizeof(DEVMODE);
dm.dmDriverExtra = 0;

EnumDisplaySettings(NULL,ENUM_CURRENT_SETTINGS,&dm);

m_ScrHeight = dm.dmPelsHeight;
m_ScrWidth = dm.dmPelsWidth;

if(m_ScrHeight<864&& m_ScrWidth<1152)
{
    if(DialogWindow.DoModal()==IDOK)
    {
        dm.dmPelsHeight = 864;
        dm.dmPelsWidth = 1152;
        ChangeDisplaySettings(&dm,0);
    }
}
```

An object of the class *CResolutionChange* provides the required functionality. *DEVMODE* can capture the user's current display settings. *dmPelsHeight* and *dmPelsWidth*, member variables of the *DEVMODE* structure, store the information regarding the screen height and width respectively. When the *ChangeDisplaySettings* function is called a variable exchange takes place and the screen resolution is changed. After AnthroDB is closed, the original resolution is restored. This functionality is called from the document class destructor.

```
DEVMODE dm;  
dm.dmPelsHeight = m_ScrHeight;  
dm.dmPelsWidth = m_ScrWidth;  
ChangeDisplaySettings(&dm,0);
```

4.2.2 Splash Thread

After the resolution has been changed, a splash screen comes up showing the version, author, release date and a copyright notice. The splash screen is a bitmap image, which is displayed from the *CSplashThread* class.

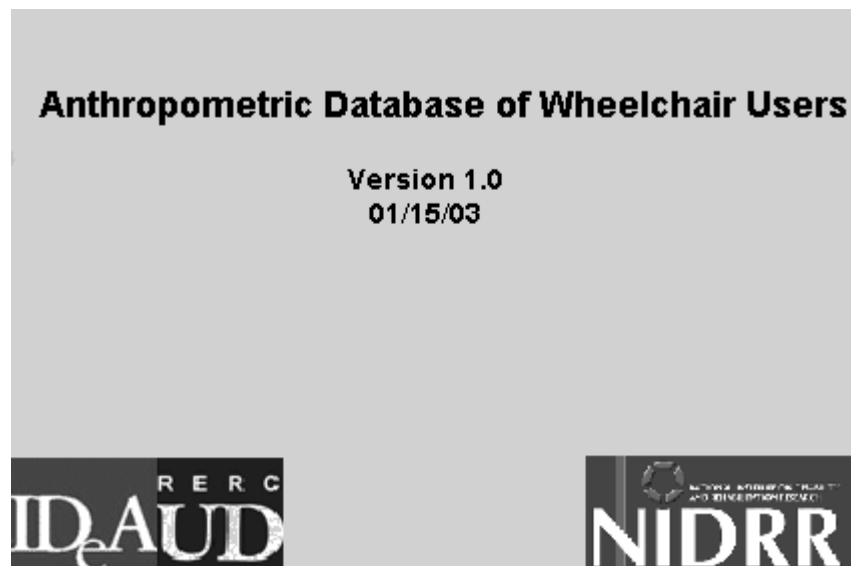


Figure 10 : Splash screen

The call to the splash screen is made from the *CAnthroDBApp* class which is called after the document class constructor. This class instantiates a separate thread so that the splash screen is shown at the same time while the other thread continues to load the application.

```
CSplashThread* pSplashThread = (CSplashThread*) AfxBeginThread (RUNTIME_CLASS (CSplashThread),  
THREAD_PRIORITY_NORMAL, 0, CREATE_SUSPENDED);
```

pSplashThread is a pointer to class *CSplashThread*. *AfxBeginThread* creates a new class thread. It takes a runtime class object as a parameter, which gets its runtime class structure from a C++ class. *RUNTIME_CLASS* returns a pointer to a *CRuntimeClass* structure for the class *CSplashThread*.

```
if(pSplashThread == NULL)  
{  
    AfxMessageBox (_T("Failed to create splash screen"), MB_OK | MB_ICONSTOP);  
    return FALSE;  
}
```

If *AfxBeginThread* returns a null pointer, a message box is displayed giving an error.

```
pSplashThread->SetBitmapToUse(IDB_SPLASH);  
pSplashThread->ResumeThread();  
Sleep(2000);
```

The function *SetBitmapToUse*, sets a bitmap to be displayed when the splash screen comes up. The *ResumeThread* function, resumes the thread which was initially started. The *Sleep* function puts the thread to sleep until the loading is done in the background. In the *AnthroDBApp* class after the *pMainFrame*->*UpdateWindow*()

function is called, the splash screen is hid by calling its *HideSplash* function. The hide splash function kills the thread which was previously initiated.

4.2.3 Frames

Since the program supports multiple documents, it has two different frames; main frame, the skeleton of the package & the child frame, the working frame.

Main Frame – The main frame is called in the *CAnthroDBApp* class just before the splash screen disappears. Following is the order of events that occur when a main frame is created.

```
CMainFrame* pMainFrame = new CMainFrame;  
    if (!pMainFrame->LoadFrame(IDR_MAINFRAME))  
        return FALSE;  
pMainFrame->ShowWindow(SW_SHOWMAXIMIZED);  
pMainFrame->UpdateWindow();
```

First, a pointer to the main frame class is declared. Then the function, *LoadFrame* loads the main frame with *IDR_MAINFRAME* as its default menu. *ShowWindow* displays the main frame when it is loaded. Usually different bits can be set here which define the window properties. In this case the *SW_SHOWMAXIMIZED* bit is set which maximizes the main frame window on start.

Child Frame – When the user starts AnthroDB, the main frame is instantiated and the child frame is suppressed by calling the *FileNothing* function.

```
if(cmdInfo.m_nShellCommand == CCommandLineInfo::FileNew)  
    cmdInfo.m_nShellCommand = CCommandLineInfo::FileNothing;
```

One of the important features of the child window is that it incorporates a window splitter. The main window is split into four parts to handle different scenes at the same time. This gives the user an option to change certain parameters in one window and view the effect immediately in the other windows. The window splitter is implemented in the *OnCreateClient* function of the *CChildFrame* class.

```

m_wndSplitter.CreateStatic(this,2,2);

m_wndSplitter.CreateView(0,0,RUNTIME_CLASS(CAnthroDBFormView),CSize(420,375),pContext);
m_wndSplitter.CreateView(0,1,RUNTIME_CLASS(CAnthroDBTopRightForm),CSize(727,375),pContext);
m_wndSplitter.CreateView(1,0,RUNTIME_CLASS(CAnthroDBBottomLeftForm),CSize(420,450),pContext);
m_wndSplitter.CreateView(1,1,RUNTIME_CLASS(CAnthroDBView),CSize(727,450),pContext);

SetActiveView((CView*)m_wndSplitter.GetPane(1,1));
((CAnthroDBDoc*)GetActiveDocument()->m_pAnthroDBView = (CAnthroDBView*)GetActiveView());
return TRUE;

```

m_wndSplitter is an object of the class, *CSplitterWnd*. The *CreateStatic* member function is used to create the static windows. *CreateStatic* signifies that throughout the application, the windows that will be created are static. In this case there will always be four windows irrespective of what they are displaying or in which mode the user is in. *CreateStatic* takes three parameters, a pointer to the current child window, number of rows and number of columns.

The *CreateView* member function, assigns different classes to each of the four windows. The windows are represented in the form of matrices with zero indexes. The different windows are as follows

R,C	Location	Class Associated with window
0,0	Left Top	CAnthroDBFormView
0,1	Right Top	CAnthroDBTopRightForm
1,0	Left Bottom	CAnthroDBLeftBottomForm
1,1	Right Bottom	CAnthroDBView

The *SetActivePane* function sets one of these four windows as active windows. An active window is the window from where all the menus and the keyboard short cuts are functional. The content of these windows is dynamic, depending upon the protocol and data type, and is explained in detail in the next few sections.

4.2.4 Top Left Form

The top left form is a reference form, and displays one of the following two things at a time

1. Demographic Data (Age, Gender, Handedness etc.)
2. Outliers from the statistical data

Depending upon what is relevant with the data displayed in the other windows, we cycle between one of the two things described above. When this form displays the demographic data, it provides the user with a drop down list of participant numbers. A user can select a desired participant and click a button at the bottom of the window, which then updates all of the four windows, now displaying the data of the new participant.

Data Retrieval

The demographic data is retrieved with the help of a SQL query. For this purpose, a *CDatabase* object is declared and then its *OpenEx* method is used to open the database. Following is the implementation of *OpenEx*.

```
m_AnthroDB.OpenEx(_T("DSN=MS Access" ),CDatabase::openReadOnly|CDatabase::noOdbcDialog );
```

The DSN parameter tells the object, which file DSN to use. The second parameter sets the bits which specify the properties of the database such as open it in the read only mode without displaying the ODBC dialog. Next step in extracting the data is setting a record set object. This is done with declaring a new *CRecordset* object by passing the address of object *m_AnthroDB* to it.

```
m_rs = new CRecordset(&m_AnthroDB);
```

This record set object *m_rs* is then used to construct the query and extract the data.

The extraction process is described in detail below

```
m_rs->Open (CRecordset::snapshot,_T(Query));
```

The *Open* member function of the record set object executes the query. *Query* is a string variable which contains the SQL command. The word snapshot means that it takes a snapshot of the database at the instance when the *Open* function is called. The complementary bit to this is *dynaset* which is used if the database is continuously changing.

When the information is extracted from the database, it is in the form of a record set or a linked list. In order to extract the specific information from the linked list, the *GetFieldValue* member function is used.

```
m_rs->GetFieldValue(index1,TempStore1);  
m_Form_Participant_Age.Format("%2.0d",atoi(TempStore1));
```

Two parameters are passed to the function *GetFieldValue*, the index of the value to be extracted and a *CString* variable where the extracted value will be stored. In order to display this in a textbox on the main form, a *CStringVariable* is used, which controls the value of a textbox. After all the data has been extracted and formatted for the display, the entire form is updated by calling *UpdateData*.

After the data has been extracted, the record set has to be closed. The record set object is recycled throughout the code and hence, if it is not closed before opening a new object, an exception would be thrown. The *Close* member function can be called to close any active record sets.

The participant IDs are shown in a drop down list. The drop down list is populated by extracting the participants from the database. In the *OnInitialUpdate* member function of the form class, this data extraction is carried out. *OnInitialUpdate* is only called once per life cycle of the window.

The drop down list is called as a combo box in MFC and has an associated class to control its behavior. The *AddString* member function adds entries to the drop down list.

```
m_ComboBox_PartID.AddString("Value");
```

When the user selects a different participant ID and clicks the *OK* button, all the windows are updated showing the data of the new participant. The event for the button click is as follows

```
CAnthroDBDoc* pDoc = GetDocument();
m_ComboBox_PartID.GetWindowText(pDoc->m_PartID);
if(pDoc->m_Protocol == 0 && pDoc->m_Data_Type == 0)
{
    pDoc->GetStructuralHumanData();
    pDoc->GetWCData();
}
else if(pDoc->m_Protocol == 1 && pDoc->m_Data_Type == 0)
{
    pDoc->GetFunctionalData();
    pDoc->GetReachHeightInfo();
}
GetDemographicData();
pDoc->GetPictureInfo();
pDoc->UpdateAllViews(NULL);
```

GetDocument initializes a document class pointer so that any document class functions or variables can be easily called. The *GetWindowText* function stores the value of the current selected participant in a document class variable, *m_PartID*. Depending upon whether the user wants to look at that participant's structural data or functional data, different functions are called. *GetDemographicData*, queries the demographic data of

the current participant and *GetPictureInfo* gets the corresponding participant pictures entries from the database. Once the data querying is complete, all the windows are updated by calling *UpdateAllViews*. When a NULL pointer is passed to this function, all the windows are updated regardless. A typical form is shown in figure 11.

Participant Demographic Information

Participant ID	S0
Age	52
Gender	Female
Handedness	Right
Disability Type	MS
Years with Disability	24
Wheelchair Type	Manual

View Participant GetFull Poster

Figure 11 : Top left form - Demographic Data

Flex Grid for Statistical Data

When the user is looking at the statistical data, this form shows outliers from the population. The excluded participants are decided, based on left and right exclusion bars on the histogram. All such excluded participants are shown in a MS Flex Grid which is an ActiveX object and functions similar to a excel worksheet. The data is populated in the flex grid with the help of a *PopulateFlexGrid* function. *m_Form_FlexGrid* is an object of the *MSFlexGrid* class which has been defined in MFC. Some of the important parts of *PopulateFlexGrid* have been outlined here.

```
m_Form_FlexGrid.Clear();
```

Each time the flex grid is populated with the new data, it is mandatory that the old data is removed by calling the *Clear* function.

```
m_Form_FlexGrid.put_TextMatrix(r,c,string);
```

The flex grid mainly shows the participant ID of the outlier excluded with the help of the bars. The *put_TextMatrix* function populates the data at r^{th} row, c^{th} column. If the user decides to choose an outlier and study him in detail, he can do so by selecting the participant ID and clicking the view participant button at the bottom of the window. As soon as the button is clicked, a new child window is opened showing the individual body data for that participant. This functionality is handled by the *OnBnClickedView* function. The different steps are shown below.

```
RowSel = m_Form_FlexGrid.get_RowSel();  
ColSel = m_Form_FlexGrid.get_ColSel();  
Buffer = m_Form_FlexGrid.get_TextMatrix(RowSel,ColSel);
```

The *get_RowSel* and *get_ColSel* identifies the row and the column of the selected cell. The *get_TextMatrix* function returns the value in that cell. This retrieved value is then stored in a temporary file in the local directory and is automatically deleted when its purpose is served. In the *OnFileNew* member function of the document class, there is a special case which handles this. The function checks if the temporary file exists. If the file is found, it is read and the necessary information is extracted and a new window is called with the *SendMessage* function. In all other cases a blank frame is opened.

```
AfxGetMainWnd()->SendMessage(WM_COMMAND,MAKEWPARAM(ID_FILE_NEW,  
CN_COMMAND), NULL);
```

If the user clicks a cell that is empty or it contains the value for a particular dimension, a message box is displayed telling the user to make a valid selection. *AfxMessageBox* produces a standard windows error box displaying the required message.

Hiding of window elements

Throughout the execution of the program, there are many window elements which are shown and hidden. For eg. When looking at a participants body sizes, his demographic information is shown and during the statistical analysis a flex grid is shown. This showing and hiding is done in the *OnUpdate* member function of the *CAnthroDBFormView* class. *OnUpdate* is called whenever an update message is passed to the window.

```
GetDlgItem(IDC)->ShowWindow(SW_SHOW);
```

A bit is set in the *ShowWindow* function which decides whether or not to show that particular window element. *GetDlgItem* gets the pointer to the window element with an id *IDC*. A typical form in statistical data mode is shown in figure 12.

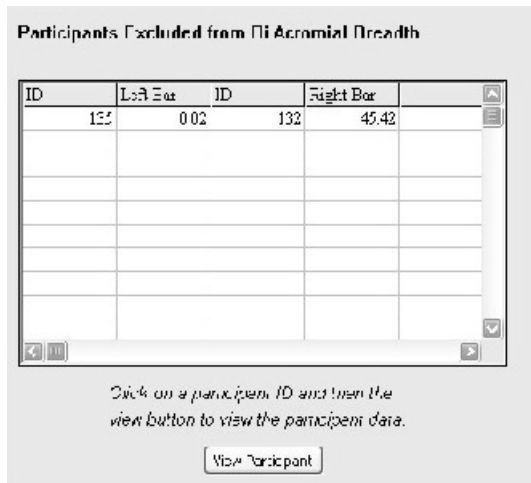


Figure 12 : Top left form - MSFlexGrid

4.2.5 Top Right Form

Top right form is associated with one and two dimensional information whose specifics are shown in the following table.

Type of Information	Information displayed by the form
Individual Body Sizes	Body Dimensions
Individual Reach Data	2D Graph showing reach on X axis and Shelf heights on Y axis
Individual grip and pinch strengths	Grip and Pinch Strength data along with some statistical information.
Statistical Data	2D Histogram of the selected population

The body dimensions are shown in an MSFlexGrid. The working of the MSFlexGrid is exactly same as described in chapter 4.2.4. A typical form in individual body sizes would look like the one shown in figure 13.

Dimension	Value(mm)	Dimension	Value(mm)	Dimension
B. Acromial Breadth	4 00	Right Eye Height	53.03	Elbow
B. Deltoid Breadth	0 00	Left Knee Height	114.58	Arm: Le.
Humeral Epicondylar Breadth Left	0 00	Right Knee Height	84.43	Arm: Le.
Humeral Epicondylar Breadth Right	5 12	Elbow Height	15.81	Forearm:
Femoral Epicondylar Breadth Left	0 00	Left Popliteal Height	152.87	Forearm:
Femoral Epicondylar Breadth Right	0 00	Right Popliteal Right	153.71	Wrist Br
Forearm Forearm Breadth	1 65	Left Wrist Height	78.38	Wrist Br
Hand Breadth Left	13 50	Right Wrist Height	88.38	Abdomin
Hand Breadth Right	36 14	Overall Length	0.00	Abdomin
Hand Length Left	153 40	Overall Width	0.00	
Hand Length Right	134 41	Overall Height	24.45	
Hip Breadth	18 43	Cervical Height	111.34	
Thigh Breadth	0 00	Left Knee Forearm Height	20.72	
Wrist Thumb Top Length Left	110 01	Right Knee Forearm Height	72.29	
Wrist Thumb Top Length Right	64 21	Suprasternale Height	140.59	

Figure 13 : Top right form - Structural dimensions

When working with the reach data, the user has to select a specific case before the 2D graph is shown. The different control variables (reach variables) are

1. Weight - Weight can be 0 pound, 1 pound, 3 pounds or 5 pounds.
2. Angle – Is the angle made by the shelving unit with the mid sagittal plane. Its value can be 0, 45 or 90 degrees.
3. Shelf – There are 5 shelves whose height depends upon individual body structure.
4. Reference point – The 2D graphs are drawn with respect to a reference point.

There are four reference points in our design - acromion of the reaching arm, distal knee point, anterior most point and the lateral most point.

If the user chooses to do an analysis on individual reaching capabilities, a series of dialogs are displayed before he actually sees the 3D reach envelope. The 2D graphs show more detailed information about a specific case. For example reach graph of one pound weight at 0 degrees with respect to the acromion. This 2D graph shows the actual reaching distance from the acromion on the X axis and the different shelf heights on the Y axis. The X coordinate of the origin is same as the X coordinate of the reference point. Hence if the reach point is behind the reference point, the graph is plotted on the negative side of the X axis. Points on the negative side of the X axis are shown in red and that on the positive side are shown in blue.

In AnthroDB a new concept of a clipping plane has been introduced. A clipping plane is a simple vertical plane which can be moved back and forth at the user's discretion. The significance of the clipping plane is that it can be placed at a specific distance from a reference point and all the intersection points with the reach envelope can be calculated. This provides the designer with all the points an individual can reach in front (or on the side) of his body at a particular distance. On the 2D graph the clipping plane is identified by a dark black vertical line. In AnthroDB a user can chose from two preset clipping planes, a plane parallel to the XZ plane, and a plane parallel to the YZ plane

A typical 2D graph with 0 pound weight, 0° angle and with respect to his acromion is shown in the following figure.

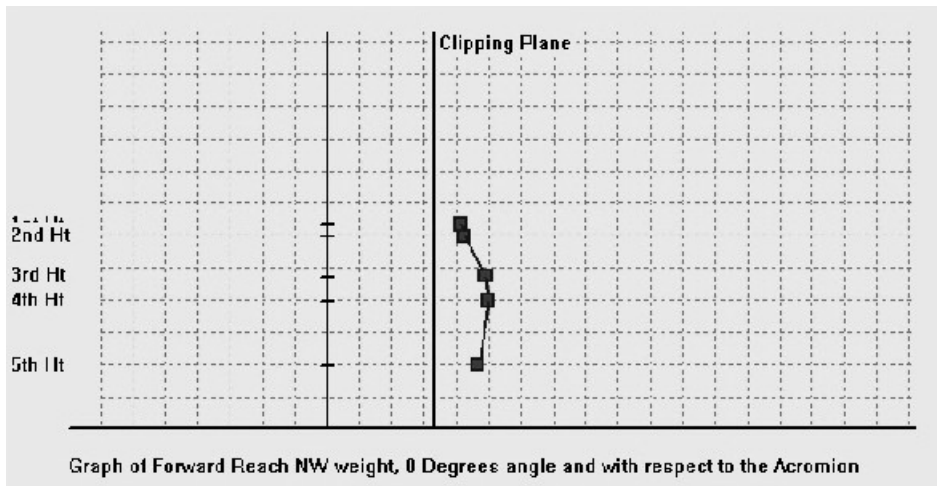


Figure 14 : Top right form - 2D Reach graph

If the user is interested in looking at more than one reach at a time, he can select the corresponding variables from the reach dialog. The following figure shows the 0 pound, 1 pound, 3 pound and 5 pound reach on the same graph.

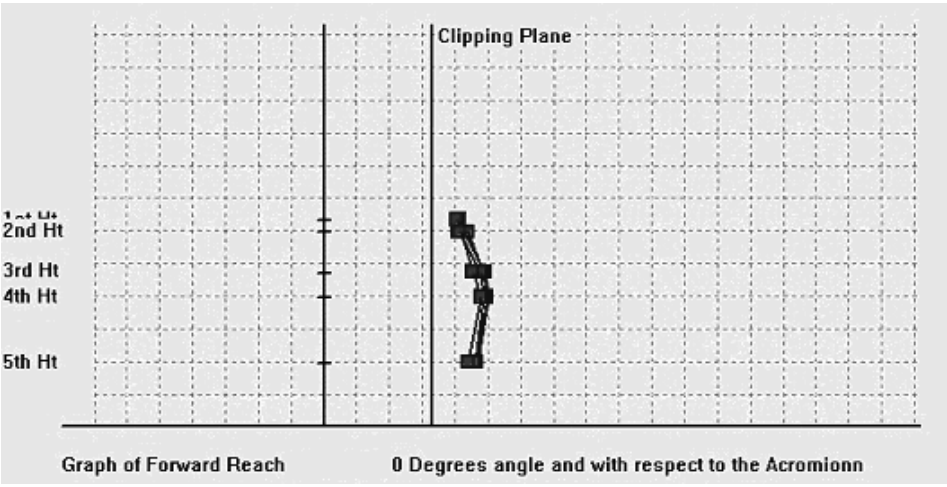


Figure 15 : Top right from - Multiple 2D graphs

When the user decides to do a statistical analysis on a particular dimension, this window shows a 2D histogram. As soon as the user wishes to switch the protocols he is again prompted with a series of dialogs. After the correct selections have been made a 2D histogram is shown with the statistical values on the top left corner of the window. A typical case scenario is described in the following paragraph which would further explain the working of the 2D histogram in detail.

Let's take an example that a user is designing aisles at a grocery store. It becomes obvious that he would be interested in seeing the variation of the overall width across the range of participants. The histogram is normalized between the smallest and the largest participant and then divided into 20 intervals at 5% each. Hence, the left most part of the histogram is the smallest individual (with smallest overall width) and the right most part is the largest individual (with the largest overall width).

Just as there was a clipping plane on the reach graph, there are exclusion bars on the histogram. These bars are responsible for excluding participants from the design and identifying outliers. The bars can only be moved at steps of 5%. The movement takes place with the help of sliders provided at the bottom of the window. The code for this is done in the *OnHScroll* event of the form view.

```
int iTemp;
float fMin,fMax;
int m_LeftScrollBarPos;

CAnthroDBDoc* pDoc = GetDocument();

m_ScrollBarItem[0] = GetDlgItem(IDC_LeftScrollBar);
m_ScrollBarItem[1] = GetDlgItem(IDC_RightScrollBar);
m_ScrollBarItem[2] = GetDlgItem(IDC_PerScrollBar);
```

```

for(int item=0; item<3; item++)
{
    if((CWnd*)pScrollBar == m_ScrollBarItem[item])
        break;
}

if(item !=3 )
    iTemp = pScrollBar->GetScrollPos();

switch(item){
case 0:
    fMin = 0; fMax = 20;
    break;
case 1:
    m_LeftScrollBarPos = m_LeftScrollBar.GetScrollPos();
    //m_LeftScrollBarPos=m_LeftScrollBarPos/5;
    fMin = m_LeftScrollBarPos+1; fMax = 20;
    //fMin = 0; fMax = 20;

    break;
case 2:
    fMin = 0; fMax = 100;
    break;
}

switch(nSBCCode){
case SB_LINELEFT :
    if(iTemp > (fMin )) iTemp --;
    else iTemp = (int)fMin;
    break;

case SB_LINERIGHT :
    if(iTemp < (fMax)) iTemp ++;
    else iTemp = (int)fMax;
    break;

case SB_THUMBPOSITION :
    iTemp=nPos;
    break;

case SB_THUMBTRACK :

```

```

        iTemp=nPos;
        break;
    }

    if(item!=3)
        pScrollBar->SetScrollPos(iTemp);

    switch(item){
    case 0:
        pDoc->m_LeftScrollBarPos = iTemp;
        break;
    case 1:
        pDoc->m_RightScrollBarPos = iTemp;
        break;
    case 2:
        pDoc->m_PercScrollBarPos = iTemp;
        break;
    }

    //OnUpdate(FALSE,FALSE,FALSE);
    pDoc->UpdateAllViews(NULL);

    CFormView::OnHScroll(nSBCode, nPos, pScrollBar);

```

The code makes use of a switch case in order to handle the different scroll events. The three scroll bars are characterized by an array *m_ScrollBarItem*. The scroll bars are divided into 20 intervals and are moved such that the exclusions bar move along with the movement of the scroll bars. The *OnHScroll* is called every time the scroll bar is used.

The statistical data shown in the top right corner of the window always excludes the outliers from the calculations. When a participant is excluded from the design, his participant ID and the value is shown in the flex grid on the top left form. The user

can then choose to look at that excluded participant in detail by clicking the view button (described earlier). A typical form with a histogram is shown in figure 16.

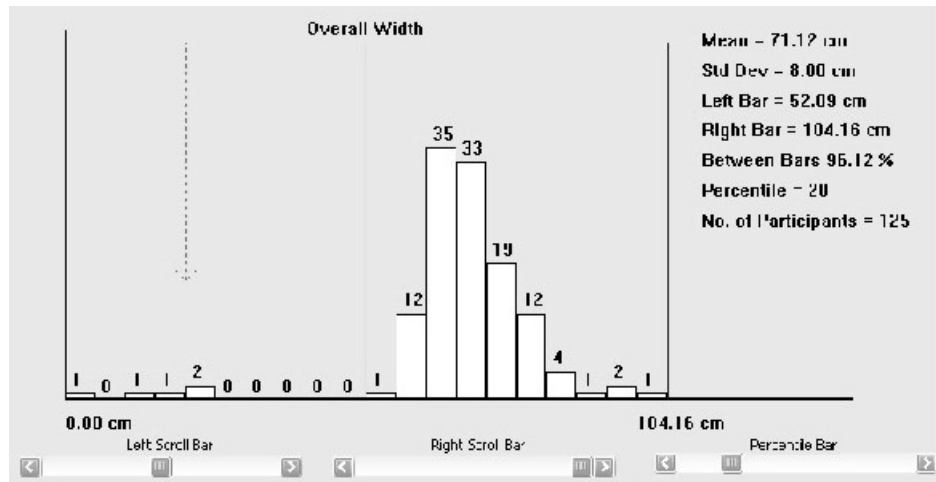


Figure 16 : Top right form - 2D histogram

4.2.6 Bottom left form

The bottom left form is mainly used for displaying audio visual. When a user is looking at a participant's body size, the front and lateral pictures are displayed as thumbnails with an ability to view a full blown image. A button at the bottom of this window, gives the user an option to view the maneuverability videos.

When a user is looking at the reach data, this window displays the distance of the clipping plane from the various reference points. It also gives the user instructions about using keyboard shortcuts to move the clipping plane. The selected reference is highlighted as shown in the following figure.

Distance of clipping plane from acromion	82. cm
Distance of clipping plane from distal knee Pt	26.04 cm
Distance of clipping plane from anterior most Pt	-13.69 cm

To move the clipping plane towards the reference point (out of screen) press + sign and to move away from reference point (inside the screen) press - sign

Figure 17 : Bottom left form - Distances from reference points

4.2.7 Bottom right form (The View class)

The view class is mainly associated with the OpenGL (3D rendering) part of the code. It has an *OnDraw* event which renders the scene. By default MFC doesn't support OpenGL. In order to enable OpenGL rendering, following functions are added to the view class, *SetWindowPixelFormat*, *OnCreate*, *CreateViewGLContext*, *OnDestroy*, *OnSize*. Let us look at them one by one (Wright, *et al.*, 1999):

```

BOOL CAnthroDBView::SetWindowPixelFormat(HDC hDC)
{
    PIXELFORMATDESCRIPTOR pixelDesc;
    pixelDesc.nSize = sizeof(PIXELFORMATDESCRIPTOR);
    pixelDesc.nVersion = 1;

    pixelDesc.dwFlags = PFD_DRAW_TO_WINDOW|PFD_DRAW_TO_BITMAP|PFD_SUPPORT_OPENGL|
    PFD_SUPPORT_GDI|PFD_STEREO_DONTCARE|PFD_DOUBLEBUFFER;
    pixelDesc.dwFlags = PFD_DOUBLEBUFFER;

    pixelDesc.iPixelFormat = PFD_TYPE_RGBA;
    pixelDesc.cColorBits = 32;
    pixelDesc.cRedBits = 8;
}

```

```

pixelDesc.cRedShift = 16;
pixelDesc.cGreenBits = 8;
pixelDesc.cGreenShift = 8;
pixelDesc.cBlueBits = 8;
pixelDesc.cBlueShift = 0;
pixelDesc.cAlphaBits = 0;
pixelDesc.cAlphaShift = 0;
pixelDesc.cAccumBits = 64;
pixelDesc.cAccumRedBits = 16;
pixelDesc.cAccumGreenBits = 16;
pixelDesc.cAccumBlueBits = 16;
pixelDesc.cAccumAlphaBits = 0;
pixelDesc.cDepthBits = 32;
pixelDesc.cStencilBits = 8;
pixelDesc.cAuxBuffers = 0;
pixelDesc.iLayerType = PFD_MAIN_PLANE;
pixelDesc.bReserved = 0;
pixelDesc.dwLayerMask = 0;
pixelDesc.dwVisibleMask = 0;
pixelDesc.dwDamageMask = 0;

m_GLPixelIndex = ChoosePixelFormat(hDC,&pixelDesc);
if(m_GLPixelIndex==0)
{
m_GLPixelIndex = 1;
if(DescribePixelFormat(hDC,m_GLPixelIndex,sizeof(PIXELFORMATDESCRIPTOR),&pixelDesc)=0)
{
return FALSE;
}
}

if(SetPixelFormat(hDC,m_GLPixelIndex,&pixelDesc)==FALSE)
{
return FALSE;
}
return TRUE;
}

int CAnthroDBView::OnCreate(LPCREATESTRUCT lpCreateStruct)
{
if(CView::OnCreate(lpCreateStruct) == -1)
return -1;
}

```

```

//TODO: Add your specialized creation code here
HWND hWnd = GetSafeHwnd();
HDC hDC = ::GetDC(hWnd);

if(SetWindowPixelFormat(hDC)==FALSE)
    return 0;
if(CreateViewGLContext(hDC)==FALSE)
    return 0;
return 0;
}

BOOL CAnthroDBView::CreateViewGLContext(HDC hDC)
{
    m_hGLContext = wglCreateContext(hDC);
    if(m_hGLContext == NULL)
    {
        return FALSE;
    }

    if(wglMakeCurrent(hDC,m_hGLContext)==FALSE)
    {
        return FALSE;
    }
    return TRUE;
}

void CAnthroDBView::OnDestroy()
{
    if(wglGetCurrentContext()!=NULL)
    {
        wglMakeCurrent(NULL,NULL);
    }
    if(m_hGLContext!=NULL)
    {
        wglDeleteContext(m_hGLContext);
        m_hGLContext = NULL;
    }
    CView::OnDestroy();
    //TODO: Add your message handler code here
}

```

```

void CAnthroDBView::OnSize(UINT nType, int cx, int cy)
{
    CView::OnSize(nType, cx, cy);

    //TODO: Add your message handler code here
    CAnthroDBDoc* pDoc = GetDocument();
    pDoc->m_Height=cy;
    pDoc->m_Width=cx;

    if (cy==0)
        pDoc->m_Aspect = pDoc->m_Width;
    else
        pDoc->m_Aspect = pDoc->m_Width/pDoc->m_Height;

    glViewport(0, 0, pDoc->m_Width, pDoc->m_Height);
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluPerspective(45,pDoc->m_Aspect,1,1000.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glDrawBuffer(GL_BACK);
    glEnable(GL_LIGHTING);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_BLEND);
    glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
}

```

SetWindowPixelFormat sets the window pixel formats for OpenGL. It enables certain options like double buffering, stereovision etc. This is one of the first things that are done in order to enable OpenGL in VC++. The pixel format describes how the graphics, that the window displays, are represented in memory. Parameters controlled by the pixel format include color depth, buffering method and supported drawing interfaces.

PIXELFORMATDESCRIPTOR contains all of the information defining a pixel format. *dwFlags* defines the devices and interfaces with which the pixel format is

compatible. Not all of these flags are implemented in the generic release of OpenGL.

dwFlags can accept the following flags:

PFD_DRAW_TO_WINDOW - Enables drawing to a window or device surface.

PFD_DRAW_TO_BITMAP - Enables drawing to a bitmap in memory.

PFD_SUPPORT_GDI - Enables GDI calls. Note: This option is not valid if

PFD_DOUBLEBUFFER is specified.

PFD_SUPPORT_OPENGL - Enables OpenGL calls.

PFD_GENERIC_FORMAT - Specifies if this pixel format is supported by the Windows GDI library or by a vendor hardware device driver.

PFD_NEED_PALETTE - Tells if the buffer requires a palette. This tutorial assumes color will be done with 24 or 32 bits and will not cover palettes.

PFD_NEED_SYSTEM_PALETTE - This flag indicates if the buffer requires the reserved system palette as part of its palette.

PFD_DOUBLEBUFFER - Indicates that double-buffering is used. Note that GDI cannot be used with windows that are double buffered.

PFD_STEREO - Indicates that left and right buffers are maintained for stereo images.

iPixelFormat defines the method used to display colors. *PFD_TYPE_RGBA* means each set of bits represents a Red, Green, and Blue value, while *PFD_TYPE_COLORINDEX* means that each set of bits is an index into a color lookup table. *cColorBits* defines the number of bits used to define a color. For RGBA it is the number of bits used to represent the red, green and blue components of the color (but not the alpha). For indexed colors, it is the number of colors in the table.

cRedBits, *cGreenBits*, *cBlueBits*, *cAlphaBits* are the number of bits used to represent the respective components. *cRedShift*, *cGreenShift*, *cBlueShift*, *cAlphaShift* are the number of bits each component is offset from the beginning of the color.

Once we initialize our structure, we try to find the system pixel format that is closest to the one we want. We do this by calling

```
m_hGLPixelFormatIndex = ChoosePixelFormat(hDC, &pixelDesc);
```

ChoosePixelFormat takes an *hDC* and a *PIXELFORMATDESCRIPTOR**, and returns an index used to reference that pixel format, or 0 if the function fails. If the function fails, we just set the index to 1 and get the pixel format description using *DescribePixelFormat*. There are a limited number of pixel formats, and the system defines what their properties are. If you ask for pixel format properties that are not supported, *ChoosePixelFormat* will return an integer to the format that is closest to the one you requested. Once we have a valid pixel format index and the corresponding description we can call *SetPixelFormat*. A window's pixel format may be set only once. Now that the pixel format is set, all we have to do is create the rendering context and make it current. We do this by adding a new protected member function to the View class called *CreateViewGLContext* and a protected member variable.

```
BOOL CreateViewGLContext(HDC hDC)  
HGLRC m_hGLContext
```

CreateViewGLContext creates and makes current a rendering context. *wglCreateContext* returns a handle to a RC (Rendering Context). The pixel format for the device associated with the DC you pass into this function must be set before you call *CreateViewGLContext*. *wglMakeCurrent* sets the RC as the current context. The DC passed into this function does not need to be the same DC you used to create the context, but it must have the same device and pixel format. If another rendering context is current when you call *wglMakeCurrent*, the function simply flushes the old RC and replaces it with the new one. You may call *wglMakeCurrent(NULL, NULL)* to make no rendering context current. Because *OnDestroy* releases the window's RC, we need to delete the rendering context there. But before we delete the RC, we need to make sure it is not current. We use *wglGetCurrentContext* to see if there is a current rendering context. If there is, we remove it by calling *wglMakeCurrent(NULL, NULL)*. Next we call *wglDeleteContext* to delete our RC.

Resizing of the OpenGL window

The *OnSize* member function handles the resizing of the window. Each time the window is resized, it sets the aspect ratio. If the height of the window is zero, the aspect ratio is equal to its width or else it is the ratio of the width to the height. *glViewport(GLint x, GLint y, GLsizei width, GLsizei height)* sets the view port. *x* and *y* are the coordinates of the lower left corner of the window, (0,0) being the default. *width* and the *height* are the top right corners of the window. When OpenGL context is first attached to the window, *width* and *height* are set to the dimensions of that window. *glMatrixMode(GLenum mode)*, specifies which matrix is the current matrix and applies subsequent matrix transformations to that matrix stack. The different modes

are *GL_PROJECTION*, *GL_MODELVIEW* and *GL_TEXTURE*. The *gluPerspective(GLdouble fovy, GLdouble aspect, GLdouble zfront, GLdouble zrear)* sets up the perspective projection matrix. *fovy* is the field of view angle in degrees in the y direction, in this case 45°. *aspect* is the aspect ratio and *zfront* and *zrear* are the front and the rear clipping planes. *glLoadIdentity* replaces the current matrix with the identity matrix. *glDrawBuffer* specifies the buffer to be used for painting while using double buffering. *GL_BACK* signifies that only the back right and back left color buffers are written. If there is no back right buffer, only the back left buffer is written. The functions *glEnable* and *glDisable*, enable or disable OpenGL capabilities. *GL_LIGHTING* specifies whether to use the OpenGL lighting whereas *GL_DEPTH_TEST* specifies whether to use depth testing.

Blending

In RGB mode, pixels can be drawn using a function that blends the incoming (source) RGBA values with the RGBA values that are already in the frame buffer (the destination values). This is done by enabling the blending with the help of *GL_BLEND*. Since the blending is enabled, we need to specify the operation of blending. The *glBlendFunc(GLenum sfactor, GLenum dfactor)* specifies pixel arithmetic. The *sfactor* specifies how the red, green, blue and alpha source-blending factors are computed. The *dfactor* specifies how the red, green, blue and alpha destination-blending factors are computed.

The reason blending is used here is that a user can see all the four (0, 1, 3, 5 pound) reach envelopes at the same time and compute their intersections with the clipping

plane. The blending model used for AnthroDB is explained in the following paragraph.

The source and destination color components are referred to as (R_s, G_s, B_s, A_s) and (R_d, G_d, B_d, A_d) . They are understood to have integer values between zero and (k_R, k_G, k_B, k_A) , where

$$k_R = 2^{m_R} - 1$$

$$k_G = 2^{m_G} - 1$$

$$k_B = 2^{m_B} - 1$$

$$k_A = 2^{m_A} - 1$$

(m_R, m_G, m_B, m_A) is the number of red, green, blue and alpha bitplanes. Source and destination factors are referred to as (s_R, s_G, s_B, s_A) and (d_R, d_G, d_B, d_A) . The scale factors are denoted as (f_R, f_G, f_B, f_A) , the values of which are in the range 0 and 1. For *GL_SRC_ALPHA* the value of scale factors is, $(A_s/k_A, A_s/k_A, A_s/k_A, A_s/k_A)$ and for *GL_ONE_MINUS_SRC_ALPHA* is $(1, 1, 1, 1) - (A_s/k_A, A_s/k_A, A_s/k_A, A_s/k_A)$. These two primitives are used for rendering anti-aliased points and lines in arbitrary order.

OnDraw vs. OnPaint

As a rule of thumb, the rendering functions are called in the *OnPaint* member function. When *OnPaint* is enabled, *OnDraw* is automatically disabled. In an MDI, *OnPaint* is only called for the newest child window, disabling the *OnPaint* of the previously open child windows. In order to avoid this, we call the OpenGL rendering functions through *OnDraw*. The *OnDraw* function makes the current

context the default Device Context for OpenGL, then it calls the *RenderScene()* function from the document class. AnthroDB makes use of double buffering to avoid screen flickers. The way double buffering works is that the scene is drawn in the rear buffer and just before updating the window, it is swapped with the front buffer by calling the *SwapBuffers* function.

Event Handlers

The view class also provides storage for the different keyboard and mouse events. When handling keyboard events, the *OnKeyDown(UINT nChar, UINT nRepCnt, UINT nFlags)* function is used. Different key strokes are identified by *nChar*. There are two ways to identify the key pressed. First, by using its ASCII value and second by using virtual keys defined in MFC.

Handling of the mouse events is complicated compared to the keyboard events. The *OnLButtonDown*, *OnRButtonDown*, *OnMiddleButtonDown*, *OnLButtonUp*, *OnRButtonUp*, *OnMiddleButtonUp* identify the different mouse events. The *OnMouseMove* identifies if the mouse is moved while keeping a button pressed. Different flags are set to true depending upon which button is pressed. There are different combinations possible. For example if the user presses the left button and moves the mouse while keeping the button pressed, the participant moves across the window as the mouse moves. If the user clicks the right mouse button and moves the mouse, the view is zoomed in or out depending upon the motion. In the button up events all the flags, previously set true, are set to false so that the motion doesn't continue to occur. The wheel of the scroll mouse is used to rotate the scene about the vertical axis depending upon if

the button is rotated clockwise or anticlockwise. The wheel events are handled by *OnMouseWheel*.

The menu events or toolbar events are identified by different event handlers. The important feature of the view class is that it can be updated independently from others. Updating the entire frame can cause problems like flickering, and computational efficiency can decrease. Due to this reason, all the event handlers are defined in the view class.

Following figures show the view class at different instances. First is the 3D human model rendered in OpenGL and the second is the 3D reach envelope with a clipping plane.

Front View

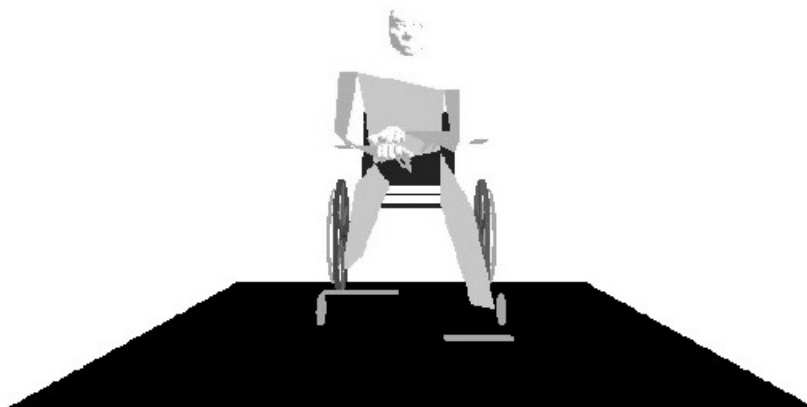


Figure 18 : Structural protocol - Front view

Left Side View

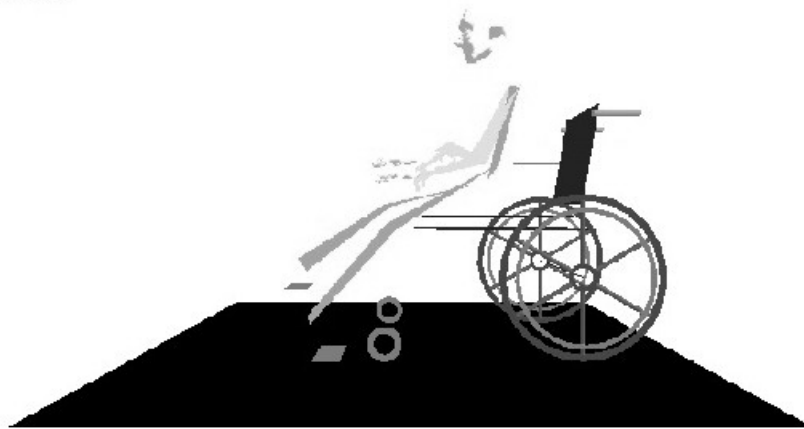


Figure 19 : Structural protocol - Side view

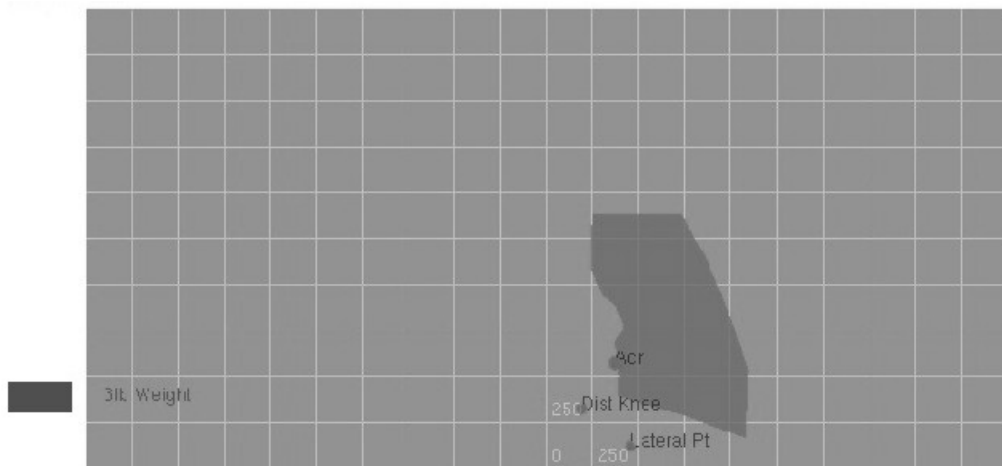


Figure 20 : Functional protocol - Reach envelopes

4.2.8 The document class

The document class is the heart of AnthroDB and does the task of supplying the member functions and/or variables to whichever class asks for it. The constructor of this class does different tasks such as one time initializations, resolution change on

the user's machine (described earlier), importing of the 3DS objects, declaration of the database object, declaration of the record set object etc.

The *OnNewDocument* is similar to a constructor but it is only called when the user clicks the open new document button. All the time and resource consuming tasks are done in the constructor so that they are called much before the *OnNewDocument* is called.

One of the most important things that *OnNewDocument* does is that it checks the total number of participants in the database by executing a query. This is useful for allocating memory to the different pointers.

The different functions in the document class perform different tasks such as retrieving the data, rendering the 3D scenes etc. Some of the important functions are described in the following sections.

Query Functions

These functions are used in querying the data from the database. The query functions are characterized by a prefix *Get* for eg. *GetStructuralDataHumanData*. The working of all the *Get* functions is similar to the one explained in this section. *GetStructuralDataHumanData* is used to retrieve the participants structural data, i.e. the data related to his body size. The extraction process begins by defining some local variables and constructing a SQL query. After the SQL query has been constructed, it is executed by calling the *Open* function of the record set. This statement is

included in a try-catch block in order to catch any exceptions because of missing participant entries in the database. If an exception is caught, a message box is displayed alerting the user that the participant doesn't exist in the database and the record set is closed.

The record set is similar to a linked list and hence it has to be traversed till an *EOF* is encountered. The linked list traversing is relatively easy and is accomplished by functions such as *MoveFirst*, *MoveNext*, *MovePrevious* and *MoveLast*. Looping through the linked list helps us extract the data and store it in respective variables. All the variables that store the data are structures which can store the X,Y,Z,L and A values, which correspond to X, Y, Z coordinates, length and angle respectively. The get functions are never called independently. Depending upon which type of data is queried, the *GetDataFromDB* makes calls to the appropriate functions.

Rendering Functions

These functions are responsible for drawing different scenes in OpenGL. The prefix *Render* tells us that it's an OpenGL rendering and not MFC drawing. All the MFC draw functions are characterized by prefix *Draw*. *RenderScene* is the mother function which makes calls to the other *Render* functions depending upon the protocol and data type. Let us take a closer look at *RenderScene*.

This function declaration begins by declaring some color surfaces which are used to render the scene in different colors. Since AnthroDB uses lighting, the lights are declared before any calls to the rendering functions are made. OpenGL is a state

machine. If a state is turned on it remains on until explicitly turned off. Hence when we declare the lights before calling the rendering functions, they are applied to all of them. *RenderScene* makes a call to *InitLights* which is the main function used for declaring and initializing the different light sources. *InitLights* is defined as follows

```
void CAnthroDBDoc::InitLights(void)
{
    GLfloat ambient[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat diffuse[] = { 0.8, 0.8, 0.8, 1.0 };
    GLfloat specular[] = { 0.8, 0.8, 0.8, 1.0 };
    GLfloat position[] = { 1000.0, 1000.0, 1000.0, 0.0 };
    GLfloat lmodel_ambient[] = { 0.4, 0.4, 0.4, 1.0 };
    GLfloat local_view[] = { 0.0 };

    glClearColor(1.0, 1.0, 1.0, 0.0);
    glShadeModel(GL_SMOOTH);

    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, position);
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
    glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);

    glEnable(GL_LIGHT0);
}
```

Different RGBA values are defined for ambient, diffuse and specular light sources. The light position is defined by the variable *position*, in this case (1000, 1000, 1000). *glClearColor* is a function call to define the window color. In this case all the rendering is done on a white background hence it is defined as (1, 1, 1, 0).

The RGBA values are normalized and hence take any floating point values between 0 and 1. *glLightfv*(*GLenum light*, *GLenum pname*, *GLfloat*) sets the light parameters. *Light* is the identifier of a light. The number of lights available depends on the implementation, but at least eight lights are supported. They are identified by symbolic names of the form *GL_LIGHTi* where $0 \leq i < \text{GL_MAX_LIGHTS}$. *pname* is a single-valued light-source parameter for light. Different values used in this case are described below

GL_AMBIENT

params contains four integer or floating-point values that specify the ambient RGBA intensity of the light. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial ambient light intensity is (0,0,0,1).

GL_DIFFUSE

params contains four integer or floating-point values that specify the diffuse RGBA intensity of the light. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial value for *GL_LIGHT0* is (1,1,1,1); for other lights, the initial value is (0,0,0,0).

GL_SPECULAR

params contains four integer or floating-point values that specify the specular RGBA intensity of the light. Integer values are mapped linearly such that the most positive

representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial value for `GL_LIGHT0` is (1,1,1,1); for other lights, the initial value is (0,0,0,0).

`GL_POSITION`

params contains four integer or floating-point values that specify the position of the light in homogeneous object coordinates. Both integer and floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The position is transformed by the model view matrix when *glLight* is called (just as if it were a point), and it is stored in eye coordinates. If the *w* component of the position is 0, the light is treated as a directional source. Diffuse and specular lighting calculations take the light's direction, but not its actual position, into account, and attenuation is disabled. Otherwise, diffuse and specular lighting calculations are based on the actual location of the light in eye coordinates, and attenuation is enabled. The initial position is (0,0,1,0); thus, the initial light source is directional, parallel to, and in the direction of the -z axis.

The function *glLightModelfv*(*GLenum pname*, *GLenum param*) sets the lighting model parameters. *pname* specifies the single valued lighting model parameter. *param* specifies the value that *pname* will be set to. The different model parameters used in this case are

`GL_LIGHT_MODEL_AMBIENT`

params contains four integer or floating-point values that specify the ambient RGBA intensity of the entire scene. Integer values are mapped linearly such that the most positive representable value maps to 1.0, and the most negative representable value maps to -1.0. Floating-point values are mapped directly. Neither integer nor floating-point values are clamped. The initial ambient scene intensity is (0.2,0.2,0.2,1.0).

GL_LIGHT_MODEL_LOCAL_VIEWER

params is a single integer or floating-point value that specifies how specular reflection angles are computed. If *params* is 0 (or 0.0), specular reflection angles take the view direction to be parallel to and in the direction of the -z axis, regardless of the location of the vertex in eye coordinates. Otherwise, specular reflections are computed from the origin of the eye coordinate system. The initial value is 0.

After the lights parameters are set, the light is enabled by calling the *glEnable* function. Once the light is enabled the color and the depth buffer are cleared to preset values by calling

```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

The *glClear* function sets the bit plane area of the window to values previously selected by *glClearColor*. The *gluLookAt* function creates a viewing matrix derived from an eye point, a reference point indicating the center of the scene, and an up vector. The matrix maps the reference point to the negative z-axis and the eye point to the origin, so that when you use a typical projection matrix, the center of the scene maps to the center of the view port. Similarly, the direction described by the up vector projected onto the viewing plane is mapped to the positive y-axis so that it

points upward in the view port. The up vector must not be parallel to the line of sight from the eye to the reference point. The matrix generated by *gluLookAt* post multiplies the current matrix. Hence the *gluLookAt*, the transforms and all the drawing functions are encapsulated in the *glPushMatrix* and *glPopMatrix* in order to preserve the model view matrix. All the objects are drawn at the origin and then transformed to their final position and orientation. *glTranslatef* and *glRotatef* applies translation and rotation to the respective objects.

After the rendering is complete, the *glFlush* function is called. The *glFlush* function forces execution of OpenGL functions in finite time. Different OpenGL implementations, buffer commands in several different locations, including network buffers and the graphics accelerator itself. The *glFlush* function empties all these buffers, causing all issued commands to be executed as quickly as they are accepted by the actual rendering engine. Though this execution may not be completed in any particular time period, it does complete in a finite amount of time.

Because any OpenGL program might be executed over a network, or on an accelerator that buffers commands, it is important to call *glFlush* in any programs requiring that all of their previously issued commands have been completed. For example, call *glFlush* before waiting for user input that depends on the generated image. The *glFlush* function can return at any time. It does not wait until the execution of all previously issued OpenGL functions is complete.

All the render functions begin with local declaration of some materials associated with the scene. When lighting is enabled, it is mandatory to enable and specify the material, or else the lighting effect cannot be seen. *glMaterialfv(GLenum face, GLenum pname, GLfloat param)*, specifies the material parameters for the lighting model. *face* specifies which face or faces are being updated. It must be GL_FRONT, GL_BACK or GL_FRONT_AND_BACK. *pname* specifies the single valued material parameter of the face or faces that are being updated. *param* specifies the value that the parameter is set to.

4.2.9 Dialogs

Dialogs form an integral part of AnthroDB. There are at least fifteen different types of dialogs in AnthroDB. Dialogs are added from the resource view and can be of two types modal and modeless dialogs. A modal dialog waits for the users input and hence halts the execution of the program until the user hits ok (or cancel) on the modal dialog. A modeless dialog is a dialog which initiates an independent thread and doesn't wait for an users input. An example of a modeless dialog is the progress dialog. The progress dialog tells the user that AnthroDB is currently working on something but it does not wait for the users input.

When the user opens a new document in AnthroDB, a series of dialogs appear and ask the user to make a selection. The dialogs are described below in the order, in which they appear

Population Characteristics Dialog

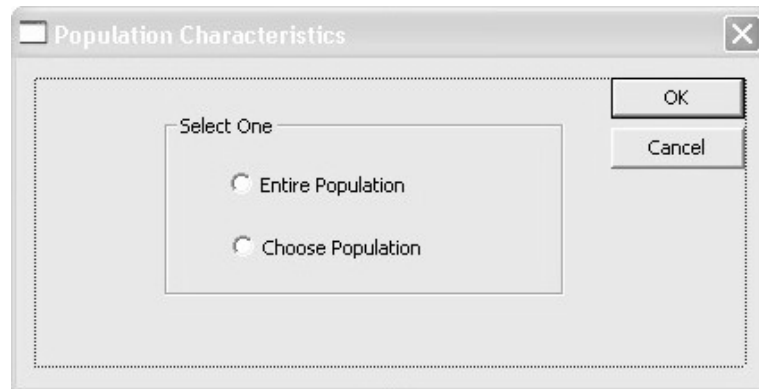


Figure 21 : Population characteristics dialog

When the design begins, the user can choose to design for the entire population or just a subset of that population depending upon what kind of a design he is interested in. Choose population lets the user select a specific subset of the population which satisfies a particular disability and demographic criteria. This dialog gives the users two options; select entire population and choose a population. Since it is done with the help of radio buttons, the user can only choose one of these.

Data Category Dialog

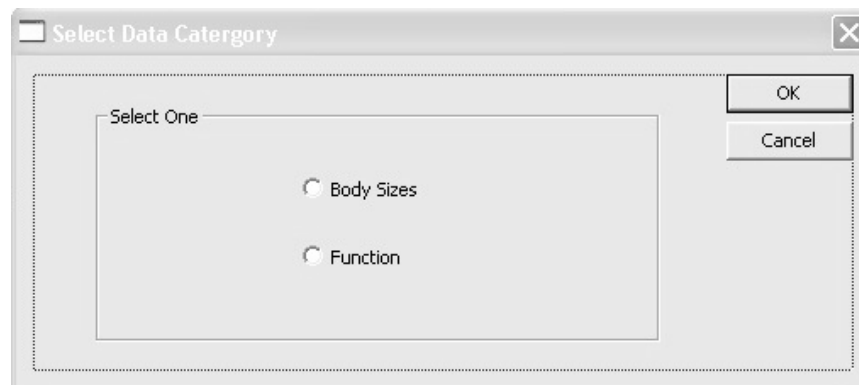


Figure 22 : Data category dialog

After the user has made the selection about the population, a data category dialog is displayed. There are two options, body sizes and function. Body sizes is the structural study which deals with a humans body dimensions and the function deals with a individuals ability to perform various tasks.

Population Dialog

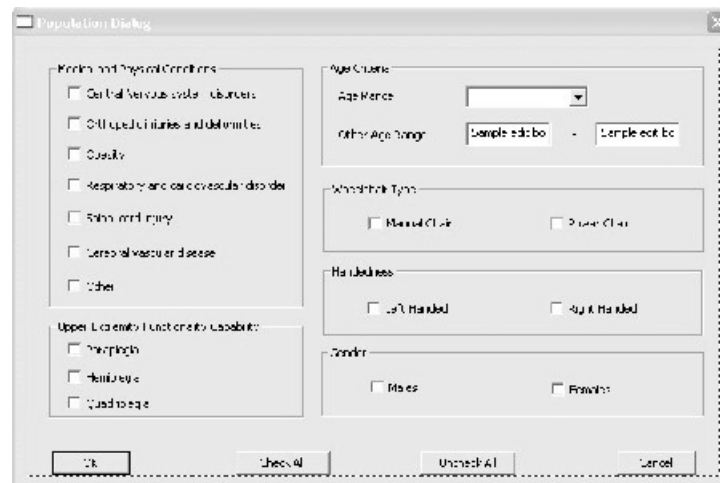


Figure 23 : Select population dialog

The population dialog allows the user to choose a particular population depending upon the medical and physical condition, wheelchair type, age, gender etc. This is useful when the user is going to do a categorized analysis on the participant data.

This dialog is used to select different reach parameters like reference point, weight condition etc. This is used in the individual functional reach data.

Picture Dialog

The picture dialog displays a blown up picture of the participant. When the user chooses to view a larger picture, a jpeg image is loaded and displayed in this dialog.

The calls to this are made from the *OnPaint* function.

```
m_Dialog_Picture.Load(m_Dialog_Picture_Name);  
// Get Picture Dimentions In Pixels  
m_Dialog_Picture.UpdateSizeOnDC(&dc);  
//Show the JPEG at actual size  
m_Dialog_Picture.Show(&dc, CPoint(0,0), CPoint(700,700), 0,0);
```

m_Dialog_Picture is a object of the *CPicture* class. *CPicture* class has been written in order to handle jpeg images. By default MFC supports bmps and hence an external plug-in is needed to display jpegs. The *load* function loads the jpeg image and the *show* member function displays it in the specified size.

Video Dialog

The video dialog is used to display the participant videos in the maneuverability protocol. The video is displayed in an ActiveX object *ActiveMovieControl*. The movie control is similar to a windows media player. When a user chooses to view the participant video, the dialog is initialized and in the *OnInitDialog* member function, the name of the file to be played and the different properties are specified.

```
Video.put_Enabled(true);  
Video.put_FileName(m_Dialog_File_Name);//m_Dialog_Lturn_Name);
```

```
Video.put_MovieWindowSize(0.9);  
Video.put_EnableSelectionControls(true);  
Video.put_AutoRewind(TRUE);  
Video.put_AutoStart(TRUE);
```

Active movie control makes use of the standard implementation of the movie control class. The video name is provided with the help of the *FileName* variable. The flags *AutoRewind* and *AutoStart*, rewind and start the video as soon as the dialog has been initialized.

Progress Dialog

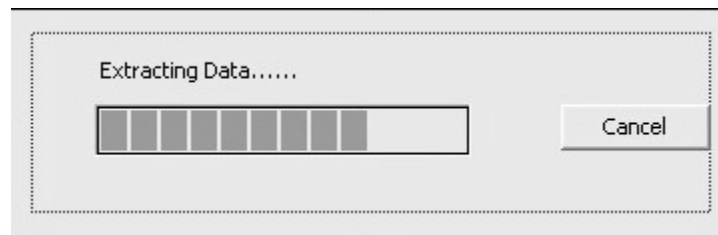


Figure 26 : Progress dialog

This dialog is used to display the progress of data extraction. When extracting functional data for a range of participants, it can often be time consuming. In order to tell the user that there is some activity going on, this progress dialog is displayed. Since this is a modeless dialog, it doesn't wait for the user input and goes away as soon as the data extraction is complete.

All the modal dialogs are displayed by calling the *DoModal* member function. The program halts for the users input and as soon as the user clicks ok, the data exchange

takes place. All the dialogs are inherited from the *CDialog* class and hence can use all the functionality of that class.

The four windows in the child frame make use of the *CFormView* class as their parent class. These windows are similar to dialogs except that they don't pop-up. They are embedded within the child frame. This is done by changing the window properties from pop-up to child type from the resource editor.

4.2.10 Menus and Toolbars

Menu and toolbar items are added from the resource editor. Each menu pick can have its own event handler and an accelerator key. Most of the dialogs can be accessed from the menus. However, depending upon the protocol and the data type, certain menu picks automatically become unavailable, if they are irrelevant. Menu and toolbar event handlers are stored in the view class.

The working of navigation menu is explained in this section which is similar to all the other menus. The navigation menu contains entries that manipulate the 3D scene (translations and rotations). There is also a navigation tool bar with corresponding entries as buttons instead of menu picks. The navigation toolbar is a dockable toolbar. i.e. it can be turned on and off as required and can be moved across the screen. The code associated with this is located in the *CMainFrame* class. The tool bar is created by making the following function calls.

```
if(!m_wndToolBar1.CreateEx(this, TBSTYLE_FLAT, WS_CHILD | WS_VISIBLE | CBRS_TOP  
/ CBRS_GRIPPER | CBRS_TOOLTIPS | CBRS_FLYBY | CBRS_SIZE_DYNAMIC) //
```

```

        !m_wndToolBar1.LoadToolBar(IDR_Navigation)){
        TRACE0("Failed to create toolbar\n");
        return -1;
    }

```

Docking is enabled by calling the *EnableDocking* function. *ShowControlBar* decides if the tool bar is by default on or off.

```

m_wndToolBar1.EnableDocking(CBRS_ALIGN_ANY);
EnableDocking(CBRS_ALIGN_ANY);
DockControlBar(&m_wndToolBar1);
ShowControlBar(&m_wndToolBar1,FALSE,FALSE);

```

When the user chooses to turn the tool bar on, the following function is called.

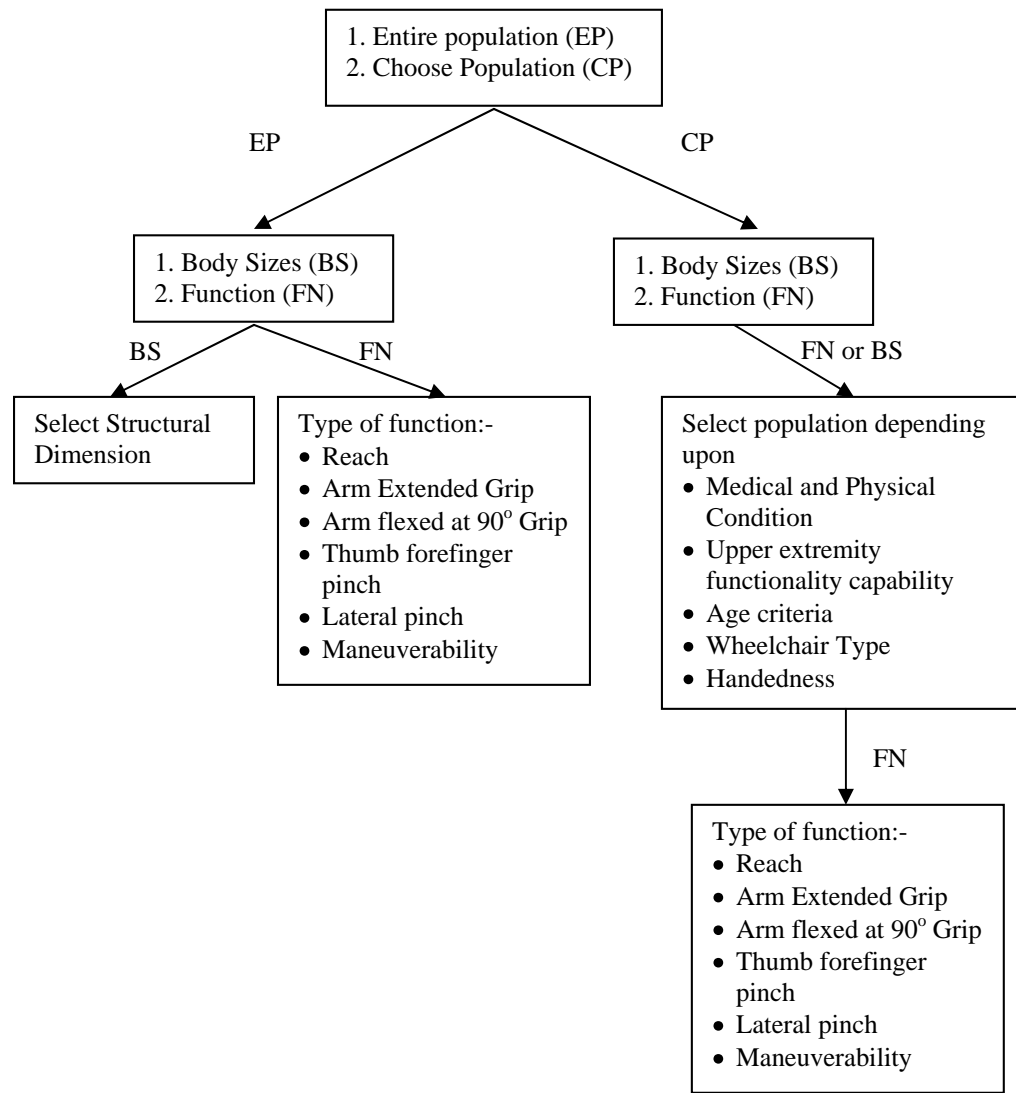
```

ShowControlBar(&m_wndToolBar1,TRUE,FALSE);

```

4.2.11 Menu Navigation

In order to prevent the user from going in unwanted directions and getting confused about what to click, menu navigation is used. Menu navigation is a simple series of menus which the user goes through before getting a graphical representation of the data. On each of the menus, the user has an ability to go back and change the previous selection. This is handled with the help of *GOTO* statements. Menu navigation can be described diagrammatically as follows.



4.2.12 Other functionality

AnthroDB has some additional functionality which is still in research phase. Most of this has been used to demonstrate the functionality and the possible areas of exploration in future. Let us look at it one by one.

Excel Export

It has been described earlier that the outliers or the participants excluded from the design are displayed in the flex grid in the top left form. It is very useful if the user is

able to export this data to an excel sheet so that he can use the statistical functions in MS Excel to carry out further analysis. For this reason, AnthroDB provides a function to export the data to an excel worksheet. The excel export takes place with the help of a DSN driver. The code for export is shown below.

```

CFileDialog Dialog FALSE, NULL, NULL, OFN_OVERWRITEPROMPT |
OFN_ALLOWMULTISELECT | OFN_HIDEREADONLY, "Excel Files (*.xls)/ *.xls/ All Files
(*.*)/*.*|", this);
CAnthroDBDoc* pDoc = GetDocument();
CDatabase database;
CString sDriver = "MICROSOFT EXCEL DRIVER (*.XLS)";
CString sSql;

TRY
{
    sSql.Format("DRIVER={%s};DSN=";FIRSTROWHASNAMES=1;
READONLY=FALSE;CREATE_DB=\ "%s\ ";DBQ=%s", sDriver,
sExcelFile, sExcelFile);

    if (database.OpenEx(sSql,CDatabase::noOdbcDialog)
    {
        sSql.Format("%s%s%s","CREATE TABLE demo (Participant_ID NUMBER,", pDoc->
m_Selected, StructStatDim," NUMBER)");
        database.ExecuteSQL(sSql);

        for(int i=1;i<pDoc->MAXI;i++)
        {
            sSql.Format("%s%s%s/%s/%0.2f%s","INSERT INTO demo (Participant_ID,",pDoc-
>m_Selected_StructStatDim,) VALUES (" ,pDoc->m_StatisticalData[i].A," ,pDoc-
>m_StatisticalData[i].L,")");
            database.ExecuteSQL(sSql);
        }
    }

    database.Close();
}
CATCH_ALL(e)
{
    TRACE1("Driver not installed: %s",sDriver);
}

```

```
    }  
    END_CATCH_ALL;
```

The *CFileDialog* provides standard file open/ save dialog. Bits such as overwrite prompt provide a warning message if one tries to replace the existing file. The export process begins with formatting a SQL string to open a connection to the excel file with the help of a DSN. A regular insert query is then constructed which will be used to insert the data into the excel file. The execute SQL command executes the query and the data is inserted into the excel file.

3DS Import

One of the things that the designer might be interested in, is using his designs and putting the participant in that environment. Most of these designs of environments are done using 3DS Max. The user can simply import the entire design by just a menu click. The 3DS file structure is open ended and the data is stored in it, in a typical format. The 3DS class simply parses a 3DS file and picks up important data points, lighting conditions, textures and renders it. At this moment only the coordinate information is extracted from the 3DS file so that OpenGL can be used to render it.

File Save and Open

When the user is working on a range of data, it is not always possible to finish the work at a single stretch. The user might want to just save all the data and come back later and work on it. This is done by using the default implementation of the serialize function of the document class. All the important values are stored in a binary file

and then simply read back when the user selects to open his previously saved projects. The file open/ save dialogs, storing and reading functions are provided by MFC by default.

5. WORKING

This section describes the working of the computer package and explains how to use the available resources efficiently.

Let us take an example that a designer is trying to design a men's apparel store. In order to be accessible to handicapped people, the aisles have to be designed such that a wheelchair user can easily navigate through it. The computer package that we have developed can be very useful in such a case. AnthroDB is constantly interfacing with a database of a large sample of wheelchair users at the back end. This data can be used to come up with a standard for designing environments for wheelchair users.

When the user starts AnthroDB, he/ she sees a blank screen with the parent window as shown in figure 27.

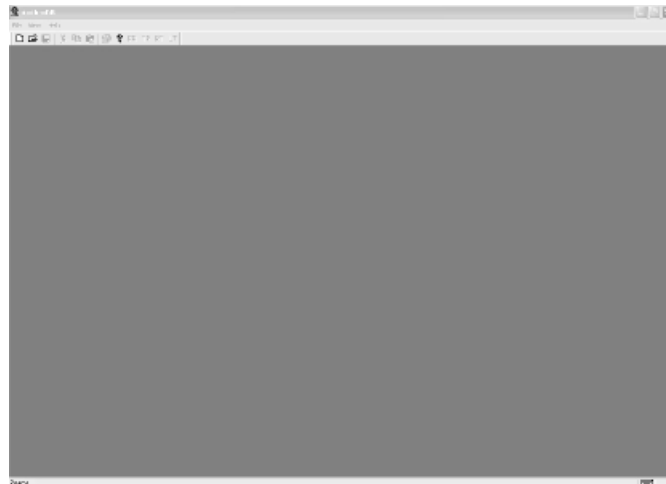


Figure 27 : AnthroDB parent window

done by setting the left and right bars at appropriate locations. Since there is only one participant within a range of 50% to 100%, the right bar can be set at the 50% interval.

As soon as the bar is placed at the 50% value, the statistical data shown in the top right corner of the window (figure 28 – 4) changes showing the user, new values. The excluded participant is shown in the flex grid (figure 28 - 1). The designer can now select this outlier and then try to analyze this large variation in the overall width. This can be done by clicking the button on the bottom of the window (figure 28 – 2). As soon as the button is pressed, a new child window is opened displaying information about the outlier’s body size.

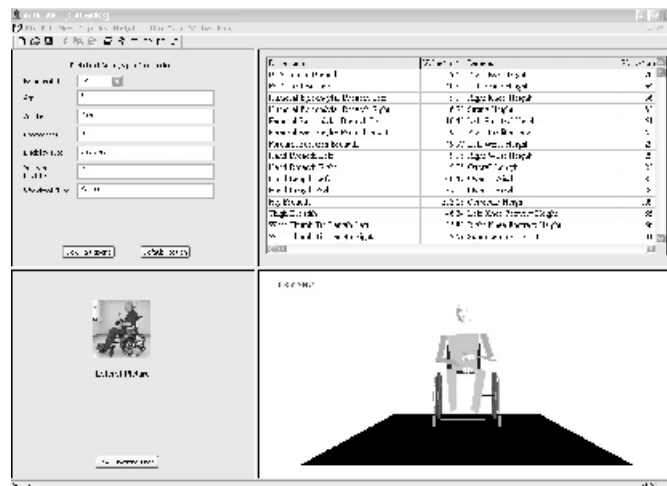


Figure 29 : AnthroDB showing structural data

The user can now study in detail about this participant or continue with his original design. Let us take another example that if the user is interested in seeing the reaching capabilities of the above participant. He/ she can switch the user mode from body sizes to functional reach data by making the appropriate selections from the menus. If the user chooses to view the 5 pound reach of this participant, he is provided with a three dimensional reach envelope and

a two dimensional reach graph. The figure 30 shows the reach data of the participant. Part 1 is the distance of the clipping plane from the different reference points. Part 2 is the three dimensional reach envelope and part 3 is the two dimensional representation of the 3D envelope.

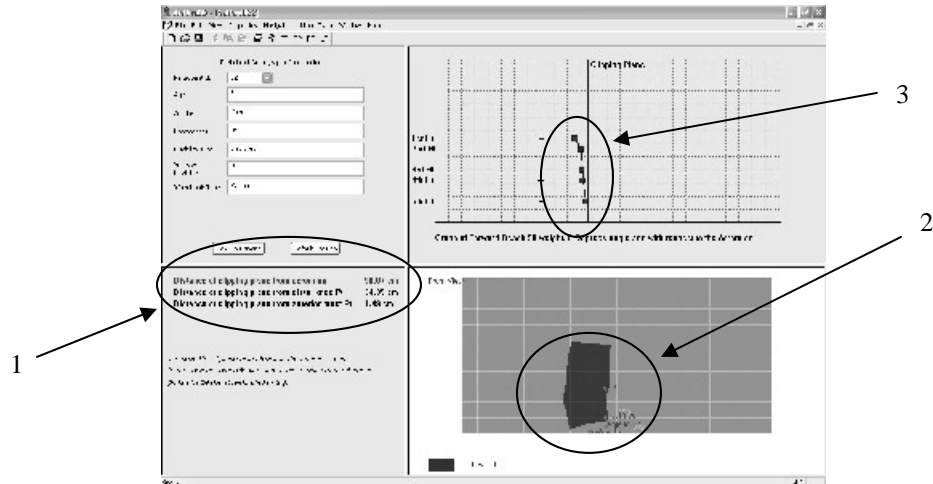


Figure 30 : AnthroDB showing reach data

If the user chooses to view all the reach envelopes at the same time, the different envelopes are shown in different shades of gray color shown in figure 31. This makes it easy to identify the different envelopes and also compute the common intersection points.

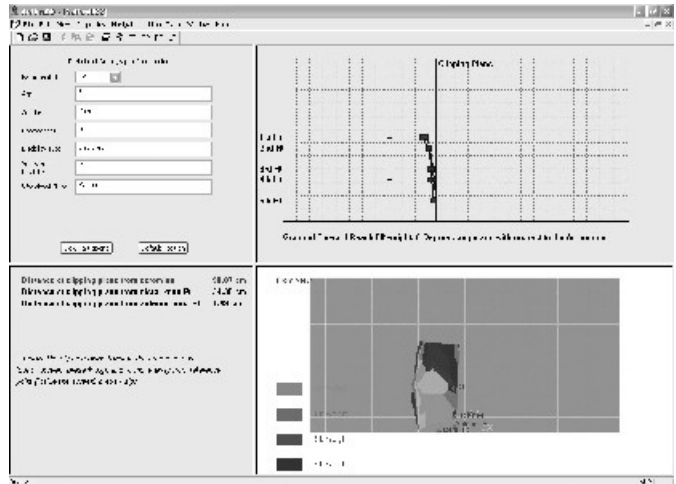


Figure 31 : Reach data with all the 4 reach envelopes

6. CONCLUSIONS

In this manuscript two most important protocols of AnthroDB have been described in detail (body sizes and reaching capabilities). Apart from these, the user can carry out different analyses on the grip and pinch strength, maneuverability study etc. AnthroDB has the ability to carry out analysis on an individual participant as well as statistical analyses on a range of participants. It incorporates unique features such as providing the user with videos and pictures of the different activities, importing 3D scenes, exporting data to the excel sheet etc. The most important feature of the package is that it caters to specific user requirements.

While performing the statistical analyses, a user can select a population based on different disability criteria, age, handedness etc. Designers can adjust the proportion of a population to accommodate for a specific dimension by interacting with a histogram plot of the frequency distribution for that dimension. The excluded participants are shown in the flex grid so that the designer can then look at them and decide why those participants fell in the lower or higher exclusion ranges. There are over 50 dimensions in which a designer can compare across the range of participants. These dimensions can be used to design most of the everyday objects. For example wrist breadth can be used to design watches; overall width can be used to design aisles in a grocery store. Overall height or eye height can be used to design seating in a theater, foot breadth and length can be used to design shoes etc.

The reach data gives important information about a user's reaching capabilities which can be used in taking critical decisions while designing environments such as placing door knobs or designing kitchen shelves etc. The user has an ability to place various clipping planes in front

of the participant and calculate the intersection points on that plane. These intersection points are the places where the participant can reach. The clipping planes can be easily moved forward and backwards with the help of key strokes in order to get a set of such points. The grid lines on the clipping plane help user to pick the dimensions of the intersection points. The two dimensional reach graphs provide more detailed information of a particular weight condition, a specific angle with respect to a reference point as opposed to the entire reach envelope in the 3D plot.

The basic idea of this thesis is to present different visualization techniques that can be applied to anthropometric data. It can easily be extended to any field. It is not necessary that this kind of an interface be used only while designing environments for people with disabilities. It gives designers, architects and engineers an ability to interact with the data in a very unique way. Pure numeric data can often be confusing. Audio and visual data by itself can be often incomplete. Pictures lack a so called depth perspective. The unique combination of photographs, videos, numbers and virtual reality makes AnthroDB a comprehensive and powerful tool.

7. FUTURE WORK

Even after making such a comprehensive package, some areas remain unexplored. It has been explained earlier that some of the features of AnthroDB have been prototyped in this version. One of them is the 3DS object import functionality.

Currently the 3DS class is only capable of importing the coordinate data from the 3DS file. Apart from the coordinate data, the 3DS file contains information about lighting, textures, reflections, shading etc used for producing different effects. Parsing this information and using it in the rendering functions of AnthroDB is an extremely complex task but something worth considering in the future.

The Excel export function is capable of exporting all the dimensional data to a worksheet. However it cannot export categorized information in this version of the code. In the future there will be a functionality to allow the export of selected data.

Some of the most important parts of the code that need immediate attention are visualization of the reach data for a range of participants and the 3D human model. The current human model is a primitive model made with polygon approximations. In order to produce a more realistic effect, a NURBS model which makes use of splines will be very helpful. The research project is also looking at a mechanism to capture real time data. For this purpose, the motion monitor, a device that tracks the location of sensors in 3D magnetic field can be very useful. This real time data can help us carry out kinematical

analyses. Other research assistants on the R1 project are already looking into incorporating this information in the code.

Last but not the least is software testing. This part is often forgotten and is extremely important. We would like to see how the software performs on different machines before it is released to the market. This includes tasks like hardware tweaking as well as thread checking.

APPENDIX A - Landmarks and Associated Considerations

(Feathers, 2003)

Landmark	Marked	Applications	Associated Anthropometric Dimensions		
			Height	Medial/Lateral	Anterior / Posterior
Abdominal Point, Anterior	No	D	Waste height to the floor.		Clearance for tables, trays, etc.
Acromion, Left and Right	Yes	D, P, M	Height of Shoulder Possible Shoulder JCR Acromial Height, sitting (NASA, 1978)	Length of Clavicle (with Sternal Notch) Possible Shoulder JCR	Possible Shoulder JCR
Ankle (JCR), Constructed Midpoint	No	D, M	Ankle JCR Height		
Anterior Superior Iliac Spine, Left and Right	Yes	M, P	Hip JCR	Hip JCR ASIS breadth	Hip JCR Pelvic Tilt
Cervicale	Yes	P, M	Cervical Height, Sitting Height, Lower Neck JCR	Lower Neck JCR	Lower Neck JCR
Deltoid Point, Left and Right	Yes	D	Height of Shoulder Clearances Deltoid Arc, (NASA, 1978)	Lateral measure of Shoulder Clearances.	Shoulder Clearances in Relative Position to the Wheelchair.
Digit I Distal, Left and Right	No	P	Hand Length and Reach Measures	Hand Length and Reach Measures	Hand Length and Reach Measures
Digit II Distal, Left and Right	No	P	Forearm Length, Hand Length and Reach Measures	Forearm Length, Hand Length and Reach Measures	Forearm Length, Hand Length and Reach Measures
Digit III Distal, Left and Right	No	P	Forearm Length, Hand Length and Reach Measures	Forearm Length, Hand Length and Reach Measures	Forearm Length, Hand Length and Reach Measures
Ectocanthus, Left and Right	No	D	Eye Height	Head Tilt (with contra-lateral side)	Head Tilt
Elbow (JCR), Constructed Midpoint	No	D, M	Elbow JCR Height		
Elbow Point, Inferior, Left and Right	No	D, M	Elbow Height	Lateral Position of the Elbow	Anterior Position of the Elbow
Elbow Point, Posterior, Left and Right	No	D, M	Posterior portion of the Elbow Height	Lateral Position of the Elbow	Anterior Position of the Elbow
Elbow Rest Plane	No	D, M	Elbow Rest Height	Elbow Rest Plane Position	Elbow Rest Plane Position

Elbow Rest Point Distal, Left and Right	No	D, M	Distal portion of the Elbow Rest Height		
Elbow Rest Point Proximal, Left and Right	No	D, M	Proximal portion of the Elbow Rest Height		
Femoral Epicondyle, Lateral, Left and Right	Yes	M, P	Knee JCR	Knee JCR Femoral Breadth (Knee Breadth) (NASA, 1978) Knee-knee breadth calculation Knee Width (Hobson and Molenbroek, 1990)	Knee JCR
Femoral Epicondyle, Medial, Left and Right	Yes	M, P	Knee JCR	Knee JCR Femoral Breadth (Knee Breadth) NASA, 1978 Biepicondylar breadth	Knee JCR
Floor Plane	No	D, M, P	Reference	Reference	Reference
Footrest Plane, Left and Right	No	D, M, P	Footrest Plane Height	Footrest Plane Position	Footrest Plane Position
Footrest Point, Distal, Left and Right	No	D, M, P	Footrest Plane Height	Footrest Plane Position	Footrest Plane Position
Footrest Point, Proximal, Left and Right	No	D, M, P	Footrest Plane Height	Footrest Plane Position	Footrest Plane Position
Forearm Point, Lateral, Left and Right	No	D, M, P	Forearm Point Lateral Height	Forearm Point Lateral Position	Forearm Point Lateral Position
H Point, Constructed	No	M, P	H Point Height	H Point Position	H Point Position
Heel Back, Left and Right	No	P	Floor to Foot Height	Distance between Heels.	Position of Foot in proximity to Wheelchair. Instep Length, (NASA, 1978; Functional Leg Length, NASA, 19780
Hip Area Lateral, Left and Right	No	D, M, P	Lateral Hip Area Height	Lateral Hip Area Position	Lateral Hip Area Position
Hub Center Point, Left and Right	No	D, M, P	Hub Center Point Height	Hub Center Point Position	Hub Center Point Position
Humeral Epicondyle, Lateral, Left and Right	Yes	M, P	Elbow JCR	Elbow JCR Lateral Elbow Clearance	Elbow JCR
Humeral Epicondyle, Medial, Left and Right	Yes	M, P	Elbow JCR	Elbow JCR	Elbow JCR
Infraorbitale, Left and Right	No	M, P	Frankfurt Plane	Frankfurt Plane	Frankfurt Plane

Knee (JCR), Constructed Midpoint	No	M, P	Knee JCR Height	Knee JCR Position	Knee JCR Position
Knee Point Anterior, Left and Right	No	M, P			Knee Clearance Buttock-knee length (NASA, 1978) Maximum Length of Upper Leg
Knee Point Superior, Left And Right	No	D, P	Maximum Height of Lower Leg Knee-Height sitting. (NASA, 1978)		Knee Clearance
Legrest Plane, Left and Right	No	D, M, P	Height of Legrest	Position of Legrest	Position of Legrest
Legrest Point, Distal, Left and Right	No	D, M, P	Distal Height of Legrest	Distal Position of Legrest	Distal Position of Legrest
Legrest Point, Proximal, Left and Right	No	D, M, P	Proximal Height of Legrest	Proximal Position of Legrest	Proximal Position of Legrest
Malleolus, Lateral, Left (Malleolus lateralis)	Yes	M, P	Ankle JCR Lateral Malleolus Height, NASA, 1978 Fibular Length	Ankle JCR Lateral Distance between Ankles	Ankle JCR
Malleolus, Lateral, Right (Malleolus lateralis)	Yes	M, P	Ankle JCR Lateral Malleolus Height (NASA, 1978) Fibular Length	Ankle JCR Lateral Distance between Ankles	Ankle JCR
Malleolus, Medial, Left (Malleolus medialis)	Yes	M, P	Ankle JCR Medial Malleolus Height (NASA, 1978) Tibial Length	Ankle JCR Minimum Distance between Ankles	Ankle JCR
Malleolus, Medial, Right (Malleolus medialis)	Yes	M, P	Ankle JCR Medial Malleolus Height (NASA, 1978) Tibial Length	Ankle JCR Minimum Distance between Ankles	Ankle JCR
MCP I, Left	No	M, P	MCP I Height	Hand Breadth Across Thumb	
MCP I, Right	No	M, P	MCP I Height	Hand Breadth Across Thumb	
MCP II, Left	No	M, P	MCP II Height	Hand Breadth	
MCP II, Right	No	M, P	MCP II Height	Hand Breadth	
MCP III, Left	No	P	MCP III Height		Palm Length
MCP III, Right	No	P	MCP III Height		Palm Length

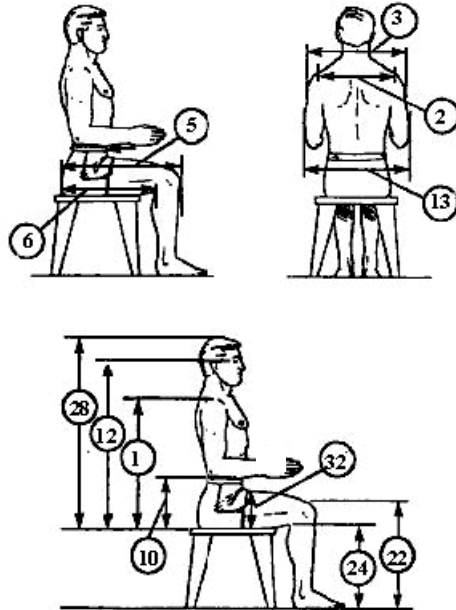
MCP V, Left	No	M, P	MCP V Height	Hand Breadth	
MCP V, Right	No	M, P	MCP V Height	Hand Breadth	
Occipital Protuberance (occiput)	No	P, M	Occipital Protuberance Height	Distance Between Midline and Occipital Protuberance	
Olecranon, Bottom, Left [Elbow Point, Inferior]	No	D	Length of Upper Arm Segment	Determination of Length of Upper Arm Segment.	Determination of Length of Upper Arm Segment. Elbow Clearances.
Landmark	Marked	Applications	Height	Medial/Lateral	Anterior/Posterior
Olecranon, Bottom, Right [Elbow Point, Inferior]	No	D	Length of Upper Arm Segment	Determination of Length of Upper Arm Segment.	Determination of Length of Upper Arm Segment. Elbow Clearances.
Olecranon, Rear, Left [Elbow Point, Posterior]	No	D, P	Proximal point for Length of Forearm	Elbow Clearances	Forearm position relative to the Wheelchair.
Olecranon, Rear, Right [Elbow Point, Posterior]	No	D, P	Proximal point for Length of Forearm	Elbow Clearances	Forearm position relative to the Wheelchair.
Popliteal, Left	Yes	D	Popliteal Height (NASA, 1978)	Popliteal Breadth	Maximum Seat Depth Upper Leg Length Estimate
Popliteal, Right	Yes	D	Popliteal Height (NASA, 1978)	Popliteal Breadth	Maximum Seat Depth Upper Leg Length Estimate
Radiale, left	Yes	M, P	Forearm Length	Forearm Length	Forearm Length
Radiale, right	Yes	M, P	Forearm Length	Forearm Length	Forearm Length
Sternal Notch (Suprasternale, Army, 1988)	Yes	D, P	Seated Height (one type of measure) Anterior Neck Length (NASA, 1978)	Variance from H-point (skew)	Upper Chest Thickness from Seat Back.
Stylian, Left	Yes	P, M	Forearm Length Calculation	Forearm Length Calculation	Forearm Length Calculation
Stylian, Right	Yes	P, M	Forearm Length Calculation	Forearm Length Calculation	Forearm Length Calculation
Thigh Point, Lateral, Left [Thigh Point, Lateral]	No	D		Maximum Distance for lateral leg clearance. Seat pan width.	
Thigh Point, Lateral, Right [Thigh Point, Lateral]	No	D		Maximum Distance for lateral leg clearance. Seat pan width.	

Toe Tip, Left (Acropodion)	No	P, D	Height of Toe Tip to Floor	Distance Between Toe Tips.	Distal Measure of Toe-Tip to Wheelchair Reference Point.
Toe Tip, Right (Acropodion)	No	P, D	Height of Toe Tip to Floor	Distance Between Toe Tips.	Distal Measure of Toe-Tip to Wheelchair Reference Point.
Top of Thigh, Left [Thigh Point, Superior]	Yes	D	Maximum Thigh Height Thigh Height from Floor		
Landmark	Marked	Applications	Height	Medial/Lateral	Anterior/Posterior
Top of Thigh, Right [Thigh Point, Superior]	Yes	D	Maximum Thigh Height Thigh Height from Floor		
Tragion, Left (tragus)	No	P, M	Frankfurt Plane	Frankfurt Plane	Frankfurt Plane
Tragion, Right (tragus)	No	P, M	Frankfurt Plane	Frankfurt Plane	Frankfurt Plane
Vertex	No	D, P	Sitting Height Head Clearance	Head Lateral Location	Head Tilt (back or forward)

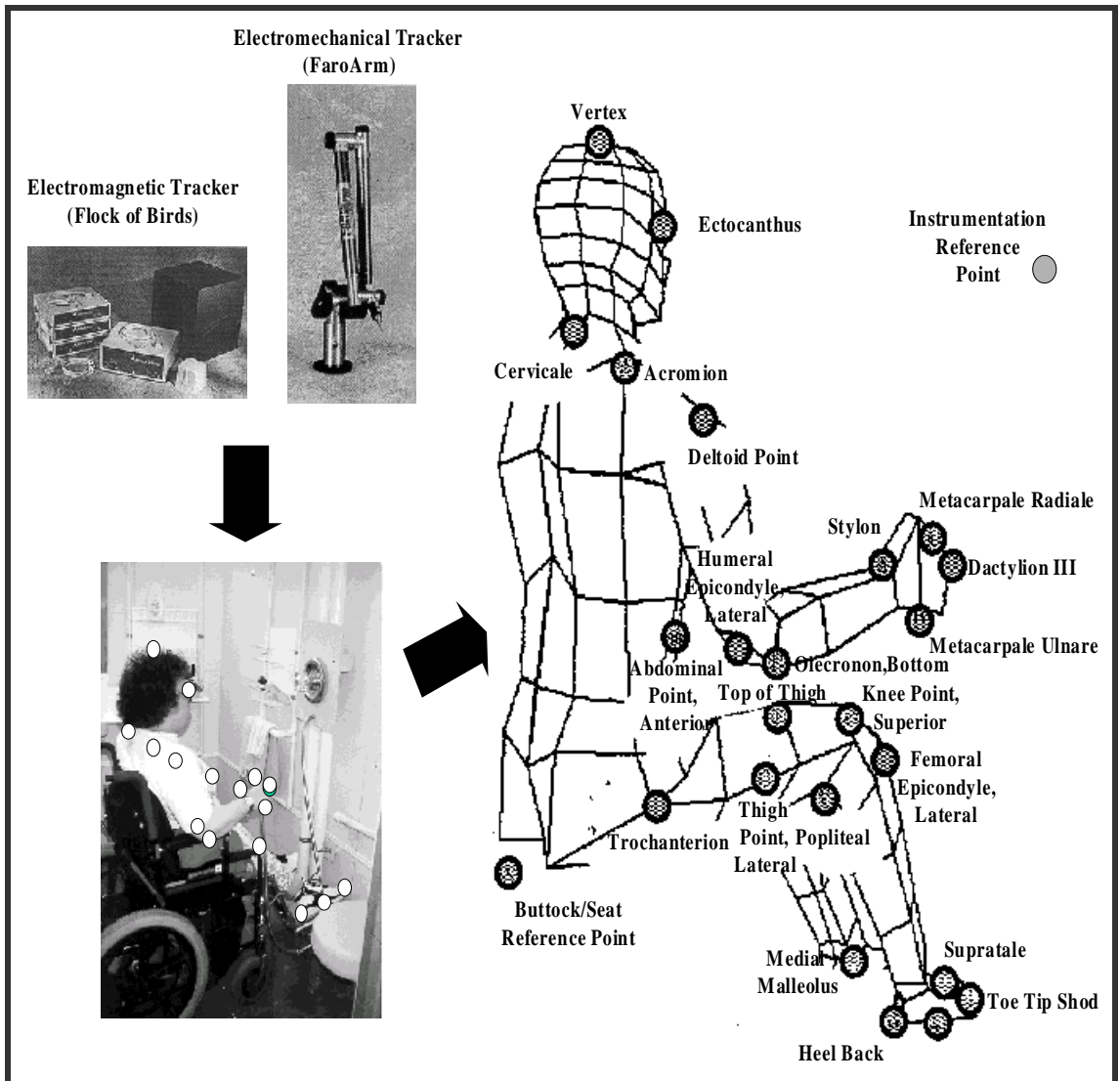
APPENDIX B - Anthropometric Measurements

All measurements are taken with the participant seated. (Feathers, 2003)

1. Body Dimensions		
		1 Acromial height
		2 Biacromial Breadth
		3 Bideltoid Breadth
		4 Biepicondylar Breadth, Elbow
		5 Buttock-Knee Length
		6 Buttock-Popliteal Length
		7 Chest Breadth
		8 Chest/Bustpoint Height
		9 Chest Depth
		10 Elbow Rest Height
		11 Elbow-Wrist Length
		12 Eye Height
		13 Forearm-Forearm Breadth
		14 Forearm-Hand Length
		15 Foot Length
		16 Foot Breadth
		17 Hand Breadth
		18 Hand Length
		19 Heel Breadth
		20 Hip Breadth
		21 Leg length (functional)
		22 Knee Height
		23 Knee-Knee Breadth
		24 Popliteal Height
		25 Shoulder-Elbow Length
		26 Shoulder-Thumbtip Length
		27 Shoulder-Wrist Length
		28 Sitting Height (seat to head)
		29 Span (Arm)
		30 Span (Elbow)
		31 Thigh Breadth
		32 Thigh Clearance
		33 Waist Depth
		34 Weight
		35 Wrist-Center of Grip Length
		36 Wrist Height
		37 Wrist-Index Finger
		38 Wrist Thumbtip Length
2. Reach/Grasp	1-6 Functional Reach (abdominal extension to functional pinch)	
	7-12 Functional Reach (acromion process to functional pinch)	
	13-18 Fingertip Reach (abdominal extension to tip of middle finger)	
	19-24 Fingertip Reach (acromion process to tip of middle finger)	
	25 Grip Breadth (tip of thumb to tip of middle finger)	
	26 Hand Spread (middle joint of thumb to middle joint middle finger)	
	27 Ulnar and Radial Deviation Range of Motion	
	28 Wrist Flexion and Extension Range of Motion	
3. Wheelchair Use	1 Acromion height (floor to acromion)	
	2 Eye Height (floor to eye)	
	3-8 Fingertip Reach (outer edge of wheelchair to tip of middle finger)	
	9-14 Functional Reach (outer edge of wheelchair to functional pinch)	
	15 Maneuverability	
	16 Wheelchair model number	
4. Upper Extremity Strength	1-4 Pinch Strength	
	5-6 Power Grip Strength	
	6-12 Upper Extremity Strength	



APPENDIX C – Approximate location of some key landmarks (Feathers, 2003)



BIBLIOGRAPHY

- Feathers, D. (2002) Anthropometrics and error. Unpublished Master's thesis. Department of Industrial Engineering, University at Buffalo, State University of New York.
- Feathers, D. (2003) Anthropometry Manual for Three-Dimensional Electromechanical Measurement. Technical Series No. 1. Center for Inclusive Design and Environmental Access- RERC UD at Buffalo, Buffalo, New York.
- Feathers, D., Paquet, V. and Drury, C. (2003). Measurement Consistency and Three Dimensional Electromagnetical Anthropometry. *International Journal of Industrial Ergonomics*.
- Gordon C, Bradtmiller B, Clauser C, Churchill T, McConville J, Tebbetts I, and Walker R (1989) 1987-1988 Anthropometric Survey of U.S. Army Personnel: Methods and Summary Statistics. Technical Report (TR-89/ 027) (AD A209600). Natick, MA: U.S. Army Natick Research, Development and Engineering Center.
- Horton, I. (1998). *Beginning Visual C++ 6*. Wrox Press. UK.
- Kroemer, K., Kroemer, H. and Kroemer-elbert, K. (1994). *Ergonomics*. Englewood Cliffs, NH: Prentice-Hall
- Paquet, V. and Feathers, D. (2003). *An Anthropometric Study of Manual and Powered Wheelchair Users*.
- Roebuck, J. (1995). *Anthropometric Methods: Designing to Fit the Human Body*. Santa Monica: Human Factors and Ergonomics Society.
- Wright, R and Sweet M. (1999). *OpenGL Super Bible, Second Edition*. Waite Group press.

INDEX

A

Acromion height.....	15
AnthroDB v, 21, 22, 23, 25, 28, 30, 31, 57, 65, 73, 77, 79, 80	
anthropometer.....	iii, 9
Anthropometry.....	8
Arm length.....	16

B

Blending.....	52
---------------	----

C

Child Frame.....	28
clipping plane.....	9, 39, 41, 44, 53, 55, 80, 83

D

Database.....	19
tables.....	19
actual data.....	20
demographics.....	19
DSN Driver.....	20
grip and pinch strength.....	19
pictures and videos.....	19
reach info.....	20
wheel chair data sheet.....	20
dockable toolbar.....	71
document class ...	4, 23, 25, 26, 33, 35, 54, 57, 58, 76
document/view architecture.....	23

F

Faro Arm.....	9, 11, 20
flex grid.....	34, 35, 36, 43, 73, 79, 82
Floor to center distance.....	15
Functional measurement.....	12

G

grip dynamometer.....	12
Grip span.....	14
grip span cone.....	14
Grip strength.....	12

K

keyboard events.....	54
----------------------	----

L

Lateral Pinch.....	13
--------------------	----

M

Main Frame.....	27
maneuverability.....	17
door use.....	19
K-turn.....	17
L Turn.....	18
level maneuverability.....	17
transfer maneuverability.....	17
modal dialog.....	65
modeless dialog.....	65, 70
mouse events.....	54
movie control.....	69, 70

N

navigation menu.....	71
----------------------	----

O

OnDraw.....	45
Overhead reach.....	16

P

pinch dynamometer.....	14
Pinch Strength.....	13
pixel format.....	48, 49, 50, 51

R

Reach.....	15
record set.....	30, 31, 32, 57, 58
rendering context.....	50, 51

S

Screen Resolution.....	23
splash screen.....	25, 26, 27

T

Thumb-Forefinger Pinch.....	13
-----------------------------	----

This document was created with Win2PDF available at <http://www.daneprairie.com>.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.