DATA-DRIVEN HUMAN BODY MORPHING

A Thesis

by

XIAO ZHANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2005

Major Subject: Visualization Sciences

DATA-DRIVEN HUMAN BODY MORPHING

A Thesis

by

XIAO ZHANG

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by

Co-Chairs of Committee,	Louis G. Tassinary
	Ergun Akleman
Committee Member,	Heather Bortfeld
Head of Department,	Phillip Tabb

August 2005

Major Subject: Visualization Sciences

ABSTRACT

Data-Driven Human Body Morphing. (August 2005)

Xiao Zhang, B.E., Tsinghua University

Co-Chairs of Advisory Committee: Dr. Louis G. Tassinary
Dr. Ergun Akleman

This thesis presents an efficient and biologically informed 3D human body morphing technique through data-driven alteration of standardized 3D models. The anthropometric data is derived from a large empirical database and processed using principal component analysis (PCA). Although techniques using PCA are relatively commonplace in computer graphics, they are mainly used for scientific visualizations and animation. Here we focus on uncovering the underlying mathematical structure of anthropometric data and using it to build an intuitive interface that allows the interactive manipulation of body shape within the normal range of human variation.

We achieve weight/gender based body morphing by using PCA. First we calculate the principal vector space of the original data. The data then are transformed into a new orthogonal multidimensional space. Next, we reduce the dimension of the data by only keeping the components of the most significant principal vectors. We then fit a curve through the original data points and are able to generate a new human body shape by inversely transforming the data from principal vector space back to the original measuring data space. Finally, we sort the original data by the body weight, calculating males and females separately. This enables us to use weight and gender as two intuitive controls for body morphing.

The Deformer program is implemented using the programming language C^{++} with OPENGL and FLTK API. 3D and human body models are created using Alias Maya^{Tm}.

ACKNOWLEDGMENTS

I would like to express my thanks to Dr. Louis G Tassinary, for his guidance and attention to details. I am also grateful to my thesis committee members Dr. Ergun Akleman and Dr. Heather Bortfeld for providing additional insight throughout this process. I would also like to thank Han Lei for his willingness to provide me with his 3D models for testing and Nie Chu for his insightful suggestions on my PCA research. I would also like to thank Peter Yunker for proofreading my thesis draft. My gratitude goes to all of the faculty and staff of the Viz Lab who have raised my level of, and enthusiasm for, knowledge, and have aided me in reaching my goals. I also wish to thank all of the friends that I have made during my time in the Viz Lab for their inspiration and enthusiasm. Most of all, I would like to thank my parents and my sister for their unconditional support, trust, encouragement and love through the years.

TABLE OF CONTENTS

Page

ABSTRACT iii

ACKNOWLEDGMENTS iv

TABLE OF CONTENTS v

LIST OF FIGURES vii

LIST OF TABLES viii

CHAPTER

I INTRODUCTION 1

I.1. Inspiration and Motivation 1
I.2. Objective 2

II RELATED WORK 3

II.1. Human Body Shape 3
II.2. Human Body Motion 5

III METHODOLOGY 6

III.1. Framework 6
III.2. Preprocessing 6
III.2.1. Database 6
III.2.2. Model Preparation 6
III.2.3. PCA of the Database 13
III.3. Body Morphing 22
III.3.1. Direct Control 22
III.3.2. Weight/Gender Control 23

IV IMPLEMENTATION AND RESULTS 26

IV.1. PCA Program 26
IV.2. Deformer Program 26
IV.2.1. Overview of User Interface 26
IV.3. Results 32

V CONCLUSION AND FUTURE WORK 37

REFERENCES 39

APPENDIX A 40

Page

APPENDIX B 49

VITA 135

LIST OF FIGURES

FIGURE Page

1 Framework of the Data-driven Body Morphing 7

2 Male Model Before and After Separated 9

3 Male Model with Minimum/Maximum Chest 10

4 Female Model Before and After Separated.. 11

5 Female Model with Minimum/Maximum Chest.. 12

6 Scree Plot of Eigenvalues 18

7 Point Cluster of the Principal Components vs. Their Body Weights 21

8 PCA Data Processing Pipeline.. 22

9 Deformer Application Window 27

10 Model Display in Wire-frame Mode.. 28

11 The “About” Popup Window.. 30

12 Deformer Main View Window 31

13 The Control Panels.. 33

14 Male Body in Direct Control Mode 34

15 Female Body in Direct Control Mode 35

16 Body Shapes in Gender/Weight Control Mode 36

LIST OF TABLES

TABLE Page

1 Covariance Matrix of the Original Dataset 14

2 Correlation Matrix of the Original Dataset 15

3 Eigenvectors and Eigenvalues of the Correlation Matrix 16

CHAPTER I

INTRODUCTION

I.1. Inspiration and Motivation

The human body has a wide range of shapes and sizes, and a better understanding and characterization of these variations can benefit research and applications in areas such as ergonomic design, computer animation, online games, avatar-based 3D virtual environments, computer vision, and movie special effects. Creating the 3D shape models for these applications is a difficult and time-consuming task, no matter what degree of realism is desired. The amount of work is further multiplied for each different character that is needed for a particular application. A common approach requires the user to first build a generalized 3D human model representing a normal body. The user is then provided with some control and manipulation methods. With these methods, the user is able to change the original body shape into any target shape that is demanded. We can subcategorize this general method into three basic types: the direct method, the procedural method and the target blending method. In the direct method, the user changes the original shape by directly adjusting the face/vertex positions of the 3D model, and this is usually done in modeling software. There are few limits on the target model, and user is able to adjust details on any part of the 3D model. This is a long and laborious process, however, that requires modeling experience and familiarity with the software. The procedural method usually involves some algorithmic or computational model as a way to generate the new target shape, such as genetic programming or a bone-muscle driven system. The user does not need to do low level modeling work, but the computation time and amount of control represent significance challenges for this approach. Additionally, this approach does not capture the detailed shape variations that may be desired for

This journal model is *IEEE Transactions on Visualization and Computer Graphics*.

realistic or artistic requirements. The third method, target shape blending, is a mix of the previous two methods. A series of predefined target shapes are provided to the user, who then changes in-between the original shape and different targets based on linear interpolation. A slider is usually employed as a simple user interface. Different weights are assigned to each target shape, so a wide variety of shapes can be generated. More predefined shapes allow the user to obtain more target shapes. This is a fast and robust method. The limitations, however, are that the user needs to justify and provide not only all the correct and necessary predefined shapes, but also all the corresponding weights. In most cases, the weights are manually adjusted by an artist, which requires a lot of experience and is learned through trial and error.

I.2. Objective

In the case of a 3D human body, the user would like to be able to change the body shape by adjusting some high level parameters. The change of one part of the body should automatically have a reasonable corresponding effect on other parts of the body. The effects should correctly match and reflect statistical data from the measurement of real human bodies. That is the main objective of this thesis. Our 3D body shape will not be as realistic and detailed as 3D scanning models, but will be essentially “hand modeled” with visually and biologically plausible proportions.

CHAPTER II

RELATED WORK

II.1. Human Body Shape

Automatic human body generation and evolution has been an interesting and active research topic in the field of computer graphics.

B. Allen, B. Curless, and Z. Popović at the University of Washington have been working on the Digital Humans Project for several years [2]. The goal of their project is to create the appearance of realistic human characters. This task can be broken down into many small problems: animation, body shape, surface appearance, etc. To date, most of their work has focused on the body shape, specifically, how to use data from 3D scanners to create models of human body shape.

Allen et al. propose an example-based method for calculating skeleton-driven body deformations [1]. Their example data is a range of scans of a human body in a variety of poses. Using markers captured during range scanning, they construct a kinematic skeleton and identify the pose of each scan. Then they construct a mutually consistent parameterization of all the scans using a subdivision surface template. The detail deformations are represented as displacements from this surface, and holes are filled smoothly within the displacement maps. Finally, they combine the range scans using *k*-nearest neighbor interpolation in pose space. In addition, they also wish to model how body shape varies between individuals. They used 250 laser range scans of volunteers in approximately the same pose and developed an algorithm to fit a common template to each range scan in order to create a common parameterization of the body surface. The common parameterization enables them to analyze the variation in body shape, using techniques such as principle component analysis. They further improved this idea by relating the variation of body shape to concrete parameters, such as body circumference, point-to-point measurements.

However, because their result is based on 3D scanning data of real human beings, the artist does not have many options for the design of the human body. The scans also require huge amounts of storage space. Although they use PCA to analyze the scanning data, the vector space is composed from the original vertices from the 3D model, so a second high level abstraction must be constructed in order to obtain more intuitive parameters for end users.

Matthew Lewis from Ohio State University did some experiments in commercial graphics packages by evolving a simple human body using genetic evolutionary design [7], [8]. Aesthetic selection is employed as a fitness function for producing successive populations of human models. Using this method, a wide range of body types can be easily generated beginning with only a small set of attributes and refined through simple, selection-driven interactions. Because the algorithm is based on evolutionary processes, a large computer graphics population first needs to be created in order to have a sufficient base for aesthetic selection. This requirement of a large population typically results in models of low-resolution (i.e. low poly-count) due to storage constraints. Also, the process may take many generations of evolution and a long time to get the desired body shape, making the method inefficient and inconvenient. Finally, because it is a purely aesthetic approach, there is no underlying relationship between the computer graphics body and a real human body.

Some commercial software packages also have a special functionality designed for modeling the human body shape in different resolutions. Poser^{TM} provides a collection of human bodies with high polygon counts as well as an interface for smoothly scaling individual body parts. ANTHROPOS^{TM} is a 3D Studio MAX^{TM} plug-in which allows body segment scaling and includes some high level parameters for adjusting attributes like race and weight. Avatar modeling software has specifically been created to allow users to create custom, low polygon human figures for use in online multi-user environments such as Blaxxun™ and Active Worlds™. Many avatar creation systems allow the replacement of individual body part segments with arbitrary geometric objects, individual part scaling, and selection from a discrete set of textures.

While most of the above allow for the individual sizing of body parts, games such as the many Quake™ and Unreal™ variants typically only allow players to select only from a set of models, with little support for avatar customization beyond texture selection. Ergonomic simulation products such as Jack™, BMD-HMS™, and Safework VirtualMan™ have all provided the ability to represent humans with highly realistic, percentile-based proportions. These ergonomics packages are generally focused on functional representation and animation, however, and do not usually provide a great deal of aesthetic flexibility.

II.2. Human Body Motion

Human body motion is also a very active area of research. Most of the existing systems for recognition, classification, synthesis, and editing of biological motion are based explicitly on empirical databases. A number of different techniques have been used to achieve motion that appears “natural” to a naive perceiver. One is Brand and Hertzmann’s “style machines.” This technique is based on a hidden Markov model, a probabilistic finite-state machine consisting of a set of discrete states, state-to state transition probabilities, and state-to-signal emission probabilities. Rose, Bodenheimer, and Cohen presented a model using radial basis functions and low-order polynomials that provide blending between actions and interpolation within stylistic state spaces. Fourier techniques are for periodic motions, such as locomotion patterns. A few of the above techniques use principal components analysis to reduce the dimensionality to a degree that stands in a reasonable relation to the size of the available dataset. Nikolaus F. Troje [9], [10], for example developed a framework that transforms biological motion into a representation allowing for analysis using linear methods from statistics and pattern recognition by using the PCA method.

CHAPTER III

METHODOLOGY

III.1. Framework

As shown in Fig. 1, our system contains two main steps: preprocessing and body morphing. Preprocessing includes creation of the base and target models, and the generation of biologically plausible and structured 3D human body models based on the empirical data contained in the database. The PCA analysis of the original database is also carried out in this phase. The preprocessing phase is a separable module, such that any the calculations performed here are independent from the body-morphing stage.

In the body-morphing phase, direct shape blending allows the user to change each body part separately, while more intuitive control is achieved by using the PCA-based weight/gender interpolation. We build the body-morphing stage into a pilot program with a simple and intuitive interface that allows user to control and manipulate the body-morphing interactively.

III.2. Preprocessing

III.2.1. Database

The database used is based on the 1988 Anthropometric Survey of the U.S. Army. This survey was conducted from August 1987 though July 1988 and carried out at 11 U.S. Army posts throughout the continental U.S. [4], [6]. The database contains 2230 female records and 2190 male records, totaling 4420 data records.

III.2.2. Model Preparation

The original model is built by following reference images of a male and female human body. The proportions of our 3D human models are kept consistent with the statistical patterns within the database. We chose eight main dimensions from the database to serve as our body "blueprint". These dimensions are: Arm Bicep Circumference, Calf Circumference, Chest

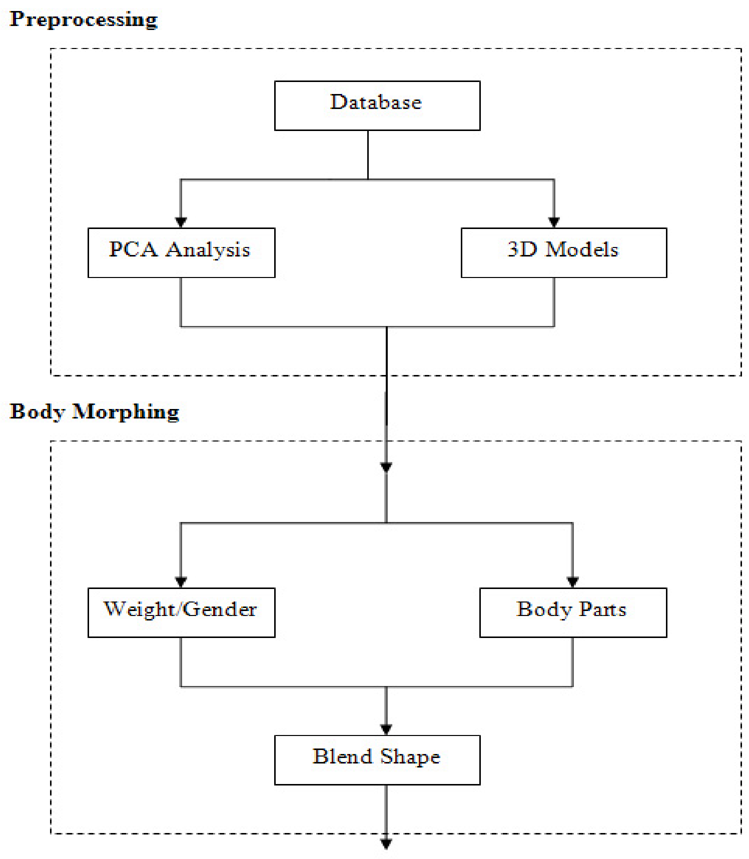
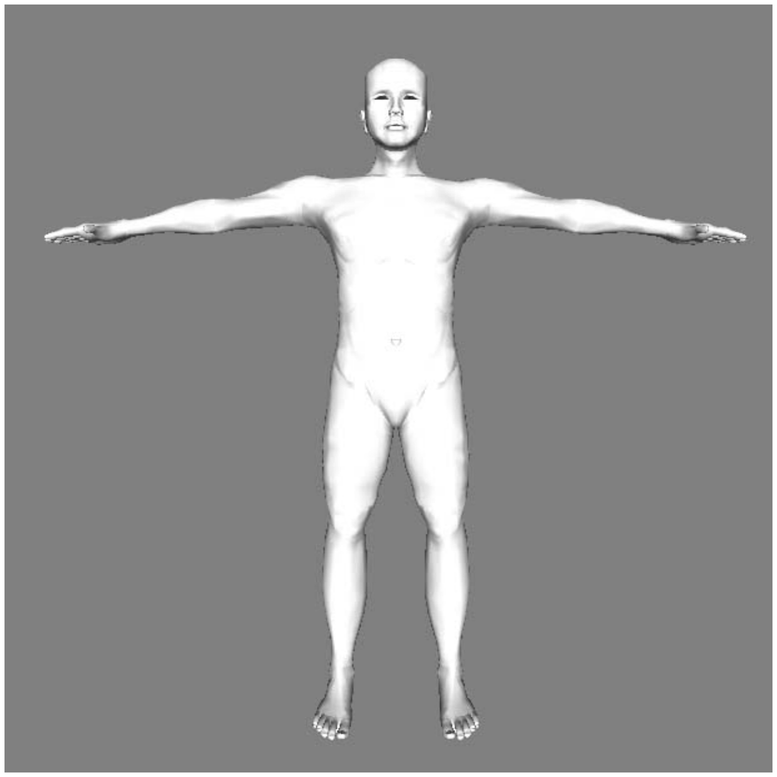


Fig. 1. Framework of the Data-driven Body Morphing.

Breadth, Forearm Circumference, Head Breadth, Hip Breadth, Thigh Circumference, and Waist Breadth. These eight variables were chosen because they represented the smallest number necessary to characterize the major anatomical divisions of the human body. The database comes with separate measurements for male and female groups. We use the average, maximum and minimum values of the above categories as our references to build the 3D models in MAYA^{TM}. There are 17 models for each gender, for a total of 34 pre-built 3D models. For each category, we have two models which correspond to the maximum and minimum value of the body part. We also have a normal body model which represents the average size for all eight categories and serve as the source model in the morphing stage. As shown through Figs. 2-5, each preformed 3D model has the same number of polygons (17709 triangle faces), allowing each new body shape to be smoothly interpolated between them. Because we require the output of our program to be a new model that could readily be imported into existing animation software environments, we split the 3D body model into individual pieces so that it can easily be attached to animation skeletons. The process is procedurally implemented as MEL script in MAYA^{TM}. The boundary polygon of each piece was carefully modeled to avoid any cracks or visible seams arising during the morphing process.



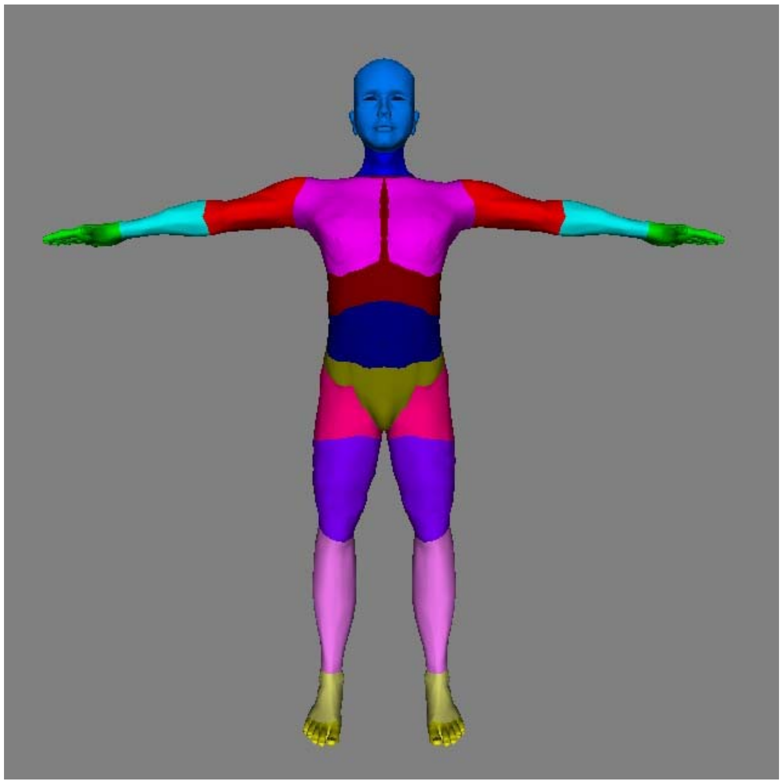
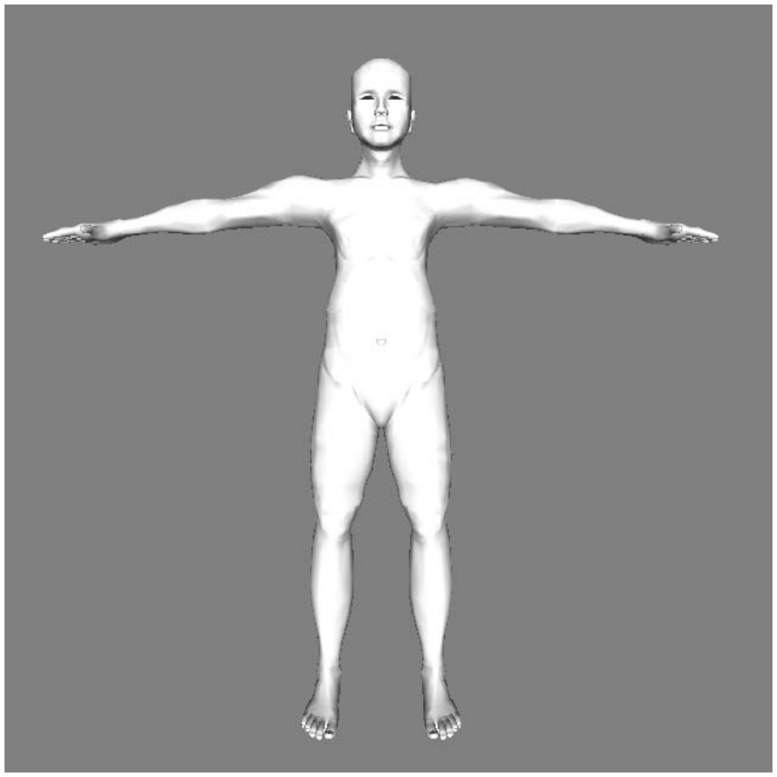


Fig. 2. Male Model Before and After Separated.



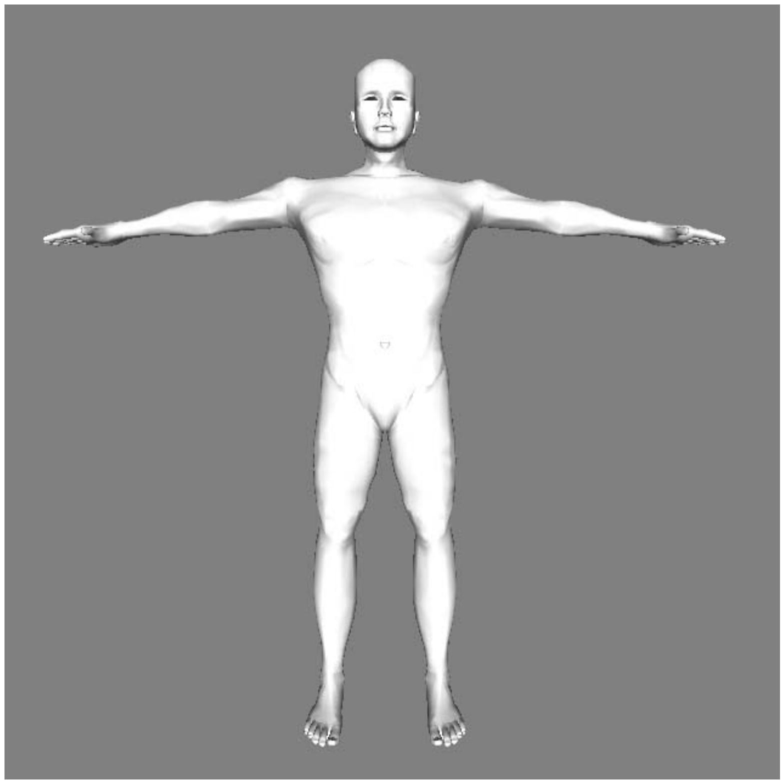
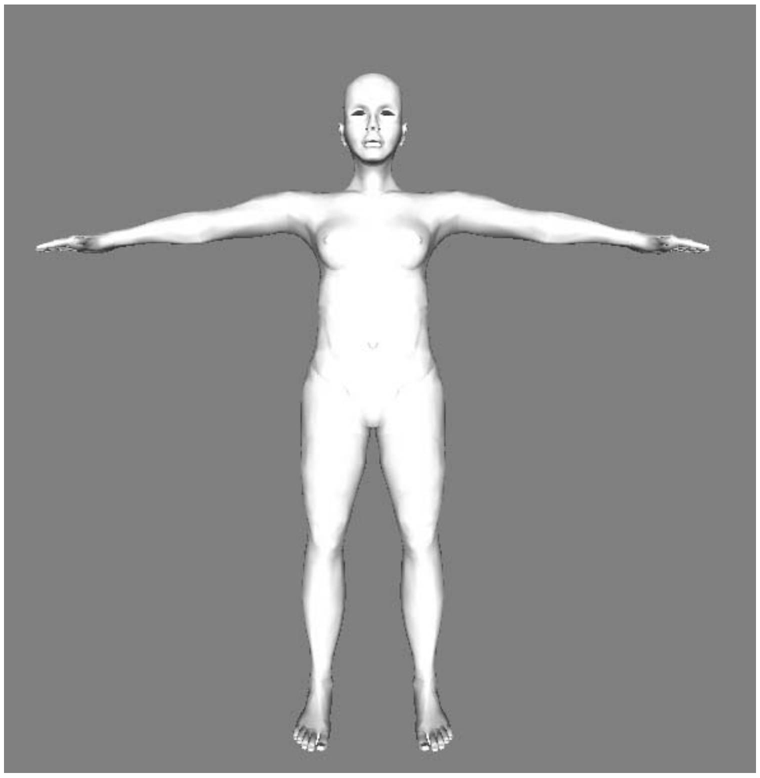


Fig. 3. Male Model with Minimum/Maximum Chest.



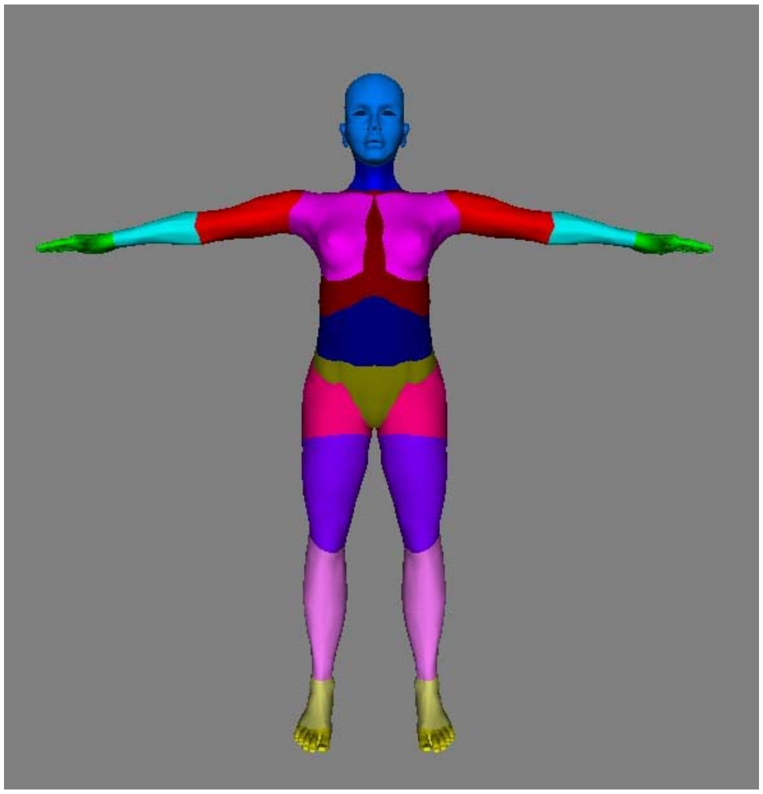
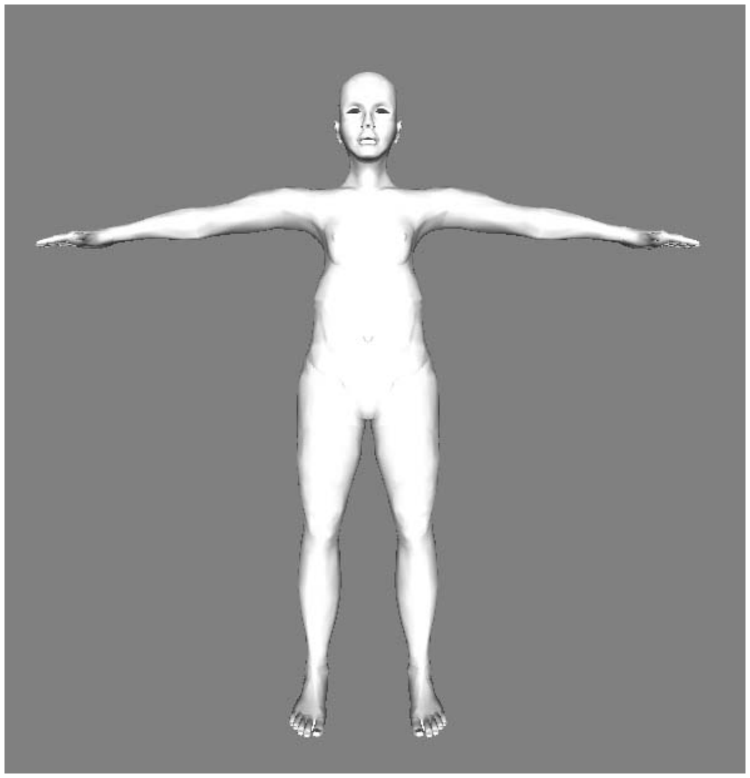


Fig. 4. Female Model Before and After Separated.



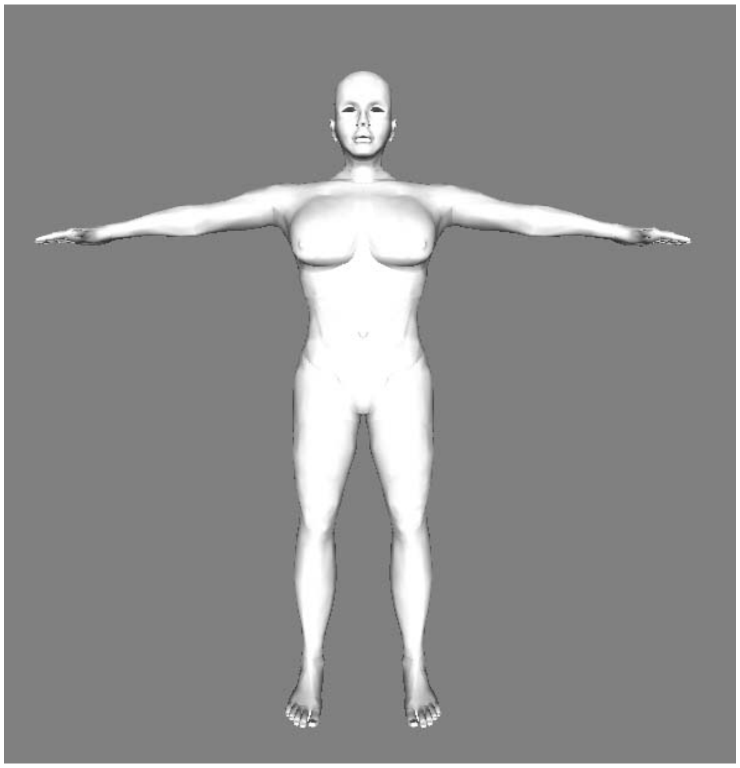


Fig. 5. Female Model with Minimum/Maximum Chest.

III.2.3. PCA of the Database

Principal Component Analysis (PCA) is a statistical technique that has found application in fields such as face recognition and image compression [5]. PCA is a common technique for finding patterns in data of high dimensionality, and expressing the data in such a way as to highlight their similarities and differences. Because patterns can be difficult to find in data of high dimensionality, where graphical representation is challenging, PCA is a powerful tool for analyzing data. Having discovered patterns in a large dataset, the data can be either expressed in terms of those patterns or compressed by reducing the number of dimensions without much loss of information. PCA is thus an ideal technique for helping us to understand and manipulate the human form explicitly.

Step 1: Collecting and cleaning the data

Because we use eight dimensions of the databases to build our model, our original dataset has eight dimensions. The data records form the male and female databases were checked for errors, and a single large eight dimensional dataset was constructed. The dataset contains all the male and female samples with the eight dimensional measurement data for each sample.

Step 2: Calculate the covariance and correlation matrix

Covariance is a statistical measure of the extent to which two variables co-vary. The covariance between two dimensions in our dataset can be calculated as:

$$\operatorname{cov}(X,Y)=\frac{\sum_{i=1}^{n}(X_i-\overline{X})(Y_i-\overline{Y})}{(n-1)}$$

where $\bar{x}$ is the mean of the x dimension of all the data points, and $\bar{y}$ is the mean of y dimension of all the data points.

If the value is negative, then as one dimension increases, the other decreases, and vise versa. If the covariance is zero, the two dimensions are independent of each other.

A useful way to express all possible covariance values between all the different dimensions is to calculate them all and construct a matrix. The definition for the covariance matrix for a set of data with n dimensions is:

$$C^{n\times n} = (c_{i,j}, c_{i,j} = \mathrm{cov}(Dim_i, Dim_j))$$

where $C^{n\times n}$ is a matrix with *n* rows and *n* columns, and Dim_i is the *i*th dimension. Essentially, this formula says for an *n*-dimensional data set, the matrix has *n* rows and *n* columns (it is square) and each entry in the matrix is the result of calculating the covariance between two separate dimensions; e.g., the entry on row3, column 2, is the covariance value calculated between the 2nd dimension and the 3rd dimension.

Down the main diagonal, the covariance value is between one of the dimensions and itself. These are the variances for that dimension. Also, because $\mathrm{cov}(X,Y) = \mathrm{cov}(Y,X)$, the matrix is symmetrical about the main diagonal. Since our data is eight dimensional, the covariance matrix will be 8×8 shown in Table 1:

$$C = \begin{pmatrix} 1376 & * & * & * & * & * & * & * \\ 746 & 743 & * & * & * & * & * & * \\ 937 & 579 & 995 & * & * & * & * & * \\ 1025 & 579 & 747 & 881 & * & * & * & * \\ 134 & 77 & 117 & 114 & 41 & * & * & * \\ 288 & 315 & 316 & 175 & 25 & 456 & * & * \\ 1083 & 944 & 863 & 717 & 87 & 760 & 2240 & * \\ 703 & 503 & 756 & 498 & 84 & 467 & 1026 & 935 \end{pmatrix}$$

Table. 1. Covariance Matrix of the Original Dataset.

The absolute magnitude of a covariance $\text{cov}(X_i, X_j)$ depends upon the standard deviations of the two components X_i and X_j. To obtain a more direct indication of how two components co-vary, we scale the covariance to obtain a correlation. Given any pair of components, X_i and X_j, we denote their correlation as either $cor(X_i, X_j)$ or $\rho_{i,j}$. The correlation is defined as:

$$\rho_{i,j} = \frac{\text{cov}(X_i, X_j)}{\sigma_i \sigma_j}$$

where σ_i and σ_j are the standard deviations of X_i and X_j. By construction, a correlation is always a number between –1 and 1. Correlation inherits the symmetry property of covariance, i.e., $\rho_{i,j} = \rho_{j,i}$, which indicates that a variable co-varies perfectly with itself. We can summarize all the correlations of our eight dimensions into a symmetric correlation matrix shown in Table 2:

$$C = \begin{pmatrix} 1.000 & * & * & * & * & * & * & * \\ 0.738 & 1.000 & * & * & * & * & * & * \\ 0.801 & 0.674 & 1.000 & * & * & * & * & * \\ 0.930 & 0.716 & 0.797 & 1.000 & * & * & * & * \\ 0.563 & 0.438 & 0.579 & 0.597 & 1.000 & * & * & * \\ 0.364 & 0.541 & 0.469 & 0.277 & 0.185 & 1.000 & * & * \\ 0.617 & 0.732 & 0.578 & 0.510 & 0.285 & 0.752 & 1.000 & * \\ 0.620 & 0.604 & 0.784 & 0.549 & 0.426 & 0.715 & 0.709 & 1.000 \end{pmatrix}$$

Table. 2. Correlation Matrix of the Original Dataset.

Step 3: Calculate the eigenvectors and eigenvalues of the correlation matrix

Because the correlation matrix is square, we can calculate the eigenvectors and eigenvalues for this matrix. These are rather important, as they provide us useful information about the data. The eigenvectors and eigenvalues, sorted by the eigenvalues, with the eigenvectors listed in row major order are shown in Table 3:

$$eigenvalues = \begin{pmatrix} 5.21 \\ 1.25 \\ 0.58 \\ 0.41 \\ 0.21 \\ 0.18 \\ 0.11 \\ 0.06 \end{pmatrix}$$

$$eigenvectors = \begin{pmatrix} 0.39 & 0.37 & 0.39 & 0.37 & 0.27 & 0.29 & 0.35 & 0.37 \\ 0.27 & -0.04 & 0.14 & 0.37 & 0.44 & -0.60 & -0.39 & -0.25 \\ -0.30 & -0.39 & 0.13 & -0.28 & 0.68 & 0.22 & -0.19 & 0.35 \\ 0.11 & -0.43 & 0.48 & 0.11 & -0.52 & -0.11 & -0.29 & 0.45 \\ 0.39 & -0.71 & -0.29 & 0.21 & 0.06 & 0.12 & 0.42 & -0.15 \\ -0.10 & 0.00 & 0.05 & -0.30 & 0.05 & -0.69 & 0.58 & 0.31 \\ 0.23 & 0.14 & -0.70 & 0.09 & 0.00 & -0.09 & -0.28 & 0.59 \\ 0.68 & 0.07 & 0.11 & -0.70 & 0.02 & 0.04 & -0.15 & -0.10 \end{pmatrix}$$

Table. 3. Eigenvectors and Eigenvalues of the Correlation Matrix.

It is important to note that these eigenvectors are unit eigenvectors; that is, their lengths are 1 and they are perpendicular to each other. But, more importantly, they provide us with information about the patterns in the data. The first eigenvector represents a line of best fit through variable space. It shows how the variables are related along that line. The second eigenvector gives us an additional, less important, pattern in the data. It shows that all the data points follow the first eigenvector line, but are off to the side of the main line by some amount. The rest of the eigenvectors do the same task as the second. By taking the eigenvectors of the correlation matrix, we have been able to extract lines that characterize the data. The rest of the steps involve transforming the data so that it is expressed in terms of the eigenvector lines.

Step 4: Choosing components and forming a feature vector

If we look at the eigenvectors and eigenvalues in Table 2, we notice that the eigenvalues are quite different. In fact, the first principle component of the data set is defined as the eigenvector with the highest eigenvalue. In our case, the eigenvectors with the largest and second largest eigenvalues account for the lion's share of the variance in the original dataset. In general, once eigenvectors are defined from the covariance matrix, the next step is to order them by eigenvalues, from highest to lowest. This places the components in order of significance. The significance of each eigenvalue can be shown more clearly with the scree plot [6](see Fig. 6).

We can decide to ignore the components of lesser significance. We do lose some information, but if the eigenvalues are small, we don't lose much. If we leave out some components, the final data set will have fewer dimensions than the original. To be precise, if originally there are n dimensions in our data, we calculate n eigenvectors with n corresponding eigenvalues. If we then choose to keep only the first p eigenvectors are kept, then the final data set has only p dimensions. Next we need to form a *feature vector*, which is defined as a matrix of vectors. A feature vector is constructed by taking the eigenvectors that we want to keep, and forming a matrix with these eigenvectors as the columns:

$$FeatureVector = \left(eig_1, eig_2, eig_3, \ldots eig_n\right)$$

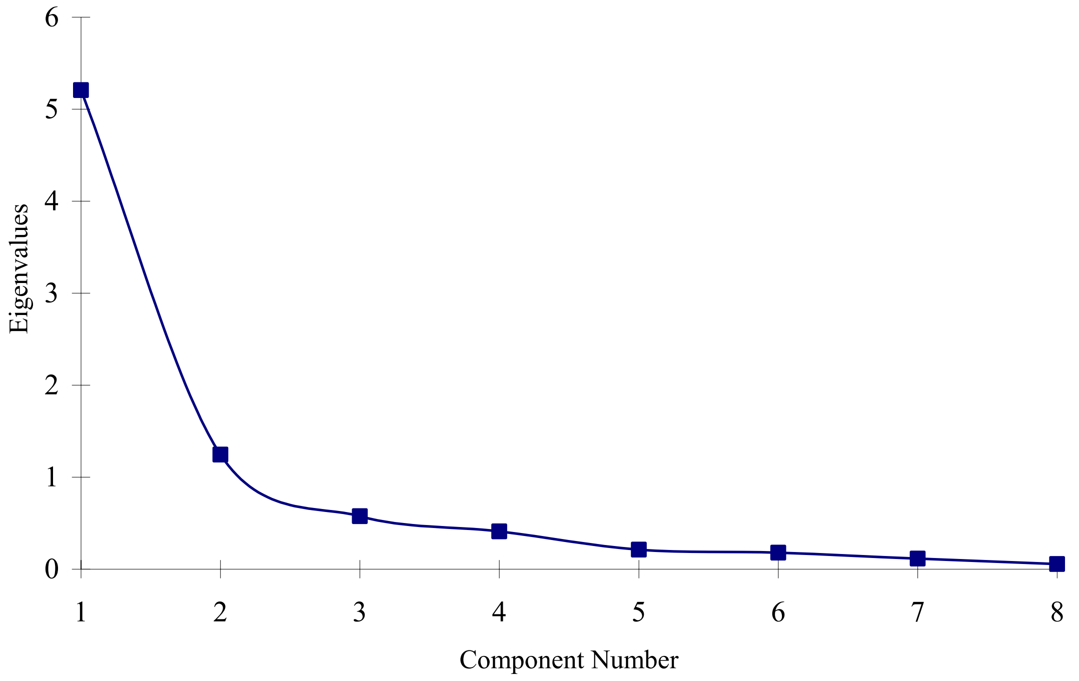


Fig. 6. Scree Plot of Eigenvalues.

For our case, we decided to form a feature vector set with all eight eigenvectors. However, only the first two principal vectors will play a key role in later steps:

$$\begin{pmatrix} 0.44 & 0.31 & 0.36 & 0.33 & 0.04 & 0.19 & 0.56 & 0.34 \\ 0.45 & 0.02 & 0.28 & 0.46 & 0.07 & -0.34 & -0.62 & -0.14 \end{pmatrix}$$

As shown in Fig. 6., their corresponding eigenvalues are much higher than the rest of eigenvalues:

Step 5: Deriving the new data set according to the body weight

The final step in PCA is to derive the new data set. Once we have chosen the components (eigenvectors) that we wish to keep and formed a feature vector, we then take the transpose of the vector and multiply it on the left of the original data set, transposed. Put formally:

$$M_2^{n\times 8} = V^{8\times 8}{}_1 \times M_1^{n\times 8},$$

where V_1 is the feature vector matrix transposed so that the eigenvectors are now in the rows, with the most significant eigenvector at the top, and M_1 is the mean-adjusted data transposed, i.e. the data items are in each column, with each row holding a separate dimension. M_2 is the final data set, with data items in columns and dimensions along rows, and n is the number of the data items for each gender. It will give us the original data solely in terms of the vectors we chose. Our original data set had eight dimensions, so our data was originally in terms of those eight dimensions. However, it is possible to express data in terms of any eight axes. If these axes are perpendicular, then the expression is at its maximum efficiency. That is why it is important that eigenvectors are always perpendicular to each other. By transforming the original data with the feature vector matrix, we have changed our data from being in terms of the original axes, to being in terms of our eight eigenvectors. When the new data set has reduced dimensionality, (i.e. some of the eigenvectors have been left out), the new data is only in terms of the vectors that are kept.

The new dataset is still an eight dimensional space because all of the eigenvectors have been retained, but now only the first two components of the new data serve as our principal components. Finally, in order to show relationship between the body weights of each gender and corresponding values of their first two principal components, the two principal components of our new male and female data records are sorted according to their body weights (see Fig. 7).

After this step, we derive four linear equations that will be used to calculate the new body shape in the body-morphing phase:

$$y = 0.5269x - 415.23 \quad ; \; y: \textit{first principal component} \quad x: \textit{male body weight}$$

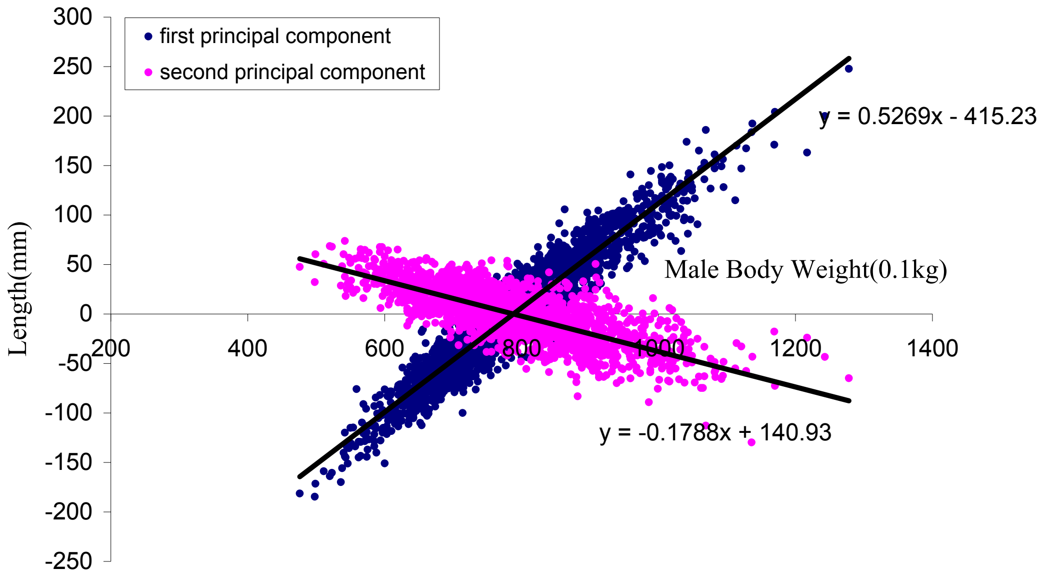
$$y = -0.1788x + 140.93 \quad ; \; y: \textit{second principal component} \quad x: \textit{male body weight}$$

$$y = 0.6079x - 377.25 \quad ; \; y: \textit{first principal component} \quad x: \textit{female body weight}$$

$$y = -0.2407x + 149.37 \quad ; \; y: \textit{second principal component} \quad x: \textit{female body weight}$$

Basically we have transformed our data so it is expressed in terms of its patterns, where the patterns are the lines that most closely describe the relationships among the data. This is helpful because we have now classified our data points as combinations of the contributions from each of those lines. Initially we had eight axes, but the values of each data point were uninformed with respect to how each point relates to the rest of the data. Now, the values of the data points tell us exactly where (i.e. above/below) a data point sits in relation to the trend lines. In the case of the transformation using all eigenvectors, we have simply altered the data so that it is in terms of those eigenvectors instead of the usual axes. In addition, we sorted the first two principal components according to the body weights so we can use body weight as a control parameter in the morphing stage, the control is made possible by fitting first order linear equations through the sampling points. Also, this procedure was done with male and female data records separately, so we can interpolate between different genders. The flow chart displayed in Fig. 8 shows the different stages in the PCA data preprocessing pipeline more clearly.

Male Body Weight vs. Principal Components



Female Body Weight vs. Principal Components

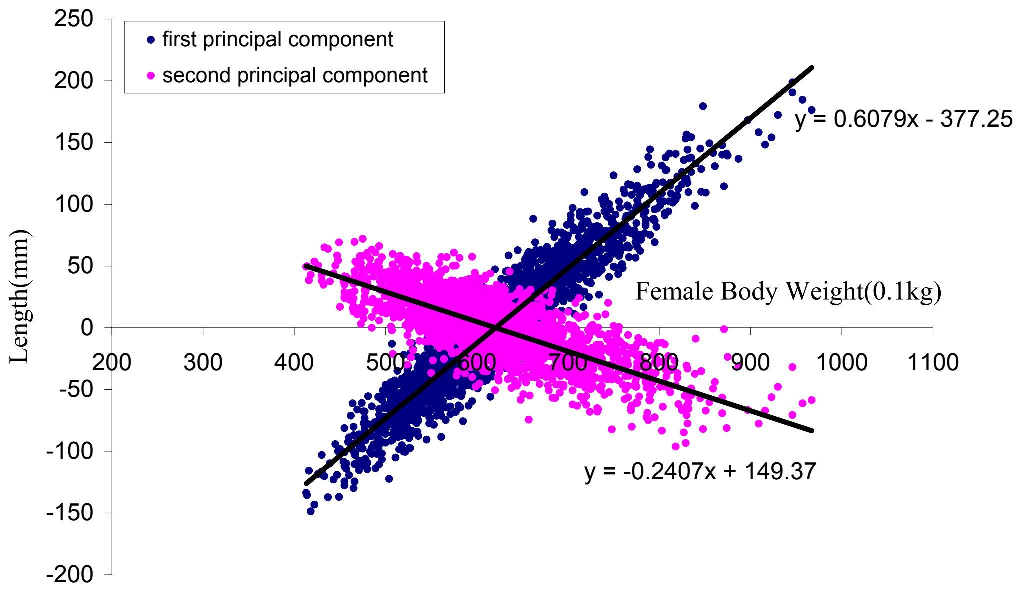


Fig. 7. Point Cluster of the Principal Component vs. Their Body Weights.

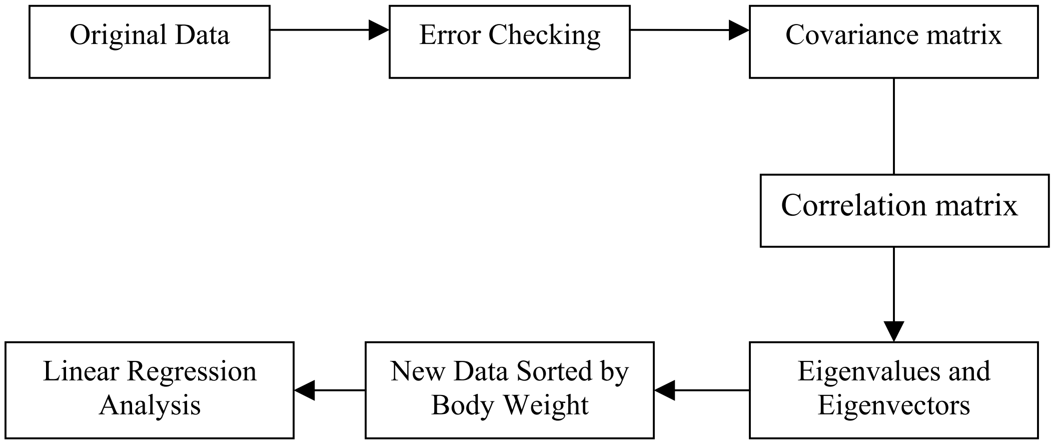

Original Data
Error Checking
Covariance matrix
Correlation matrix
Eigenvalues and Eigenvectors
New Data Sorted by Body Weight
Linear Regression Analysis

Fig. 8. PCA Data Processing Pipeline.

III.3. Body Morphing

In body morphing stage, we use two different ways to generate new body shapes from the original 3D models: direct control and weight/gender Control. They are related to each other by the underlying linear interpolation scheme; however, weight/gender control gives the user a higher-level control based on the results of the PCA.

III.3.1.Direct Control

In this control mode, we use simple linear interpolation on each body part directly to generate a new body shape. We assign different weights to each body part, and then the interpolation is done between the predefined minimum and maximum 3D body parts:

$$NewBodyShape = \sum_{i=1}^{8} \left[\min_i \times w_i + \max_i \times (1.0 - w_i) \right]$$

where the summation is over all of the eight dimensions we have used to model the 3D body, $\min_i$ is minimum of the current dimension, $\max_i$ maximum is the maximum of the current category, w_i is the weight we assigned for this body part category. In the case of the control of a single body part, the above equation is reduced to:

$$NewBodyPart_i = \min_i \times w_i + \max_i \times w_i$$

As mentioned in Chapter 1, this control method is easy but it is more arbitrary than deriving the new shape from the database.

III.3.2.Weight/Gender Control

In this control mode, we use the result from previous PCA section.

Step 1: Getting the principal component from the body weight

This is accomplished by fitting the body weight into the four linear equations we calculated, which map the body weight according to the first and second principal components for males and females.

Step 2: Getting the "original" data back

After we obtain two principal components for the new body shape, we want to generate the values for the eight body parts in terms of the original data space. To do so we need to transform our principal components back to the original data space. This is done by assigning 0 to the other six components prior to performing the transformation. Theoretically, if we have the "complete" and "correct" principal components of the body shape, we will recover the original data. This is because the contributions due to the smaller eigenvectors are negligible. By reducing the number of eigenvectors in the final transformation, however, the retrieved data has most likely lost some information. Recall that the final transform is:

$$M_2^{n\times 8} = V^{8\times 8}{}_1 \times M_1^{n\times 8},$$

This equation can be turned around to get the original data back:

$$M_1^{n\times 8} = (V_1^{8\times 8})^{-1} \times M_2^{n\times 8},$$

where $(V_1^{8\times 8})^{-1}$ is the inverse of $V_1^{8\times 8}$ (feature vector matrix). When we take all the eigenvectors in our feature vector, however, the inverse of our feature vector is actually equal to the transpose of our feature vector. This is true because the elements of the matrix are all the unit eigenvectors of our data set (an orthogonal matrix). Consequently, the new equation becomes:

$$M_1^{n\times 8} = (V_1^{8\times 8})^T \times M_2^{n\times 8},$$

To get the actual original data back, however, we need to add the mean of the original data that had been subtracted at the beginning of this process. For completeness, this procedure takes on the following form:

$$M_1^{n\times 8} = (V_1^{8\times 8})^T \times M_2^{n\times 8} + M_0^{n\times 8},$$

where $M_0^{n\times 8}$ is the mean value matrix of the original dataset.

Step 3: Interpolation

Once we return the data to the original data space, we can use them as the weights for the eight body parts, and interpolate the new body shape in the same way as in the direct control method.

$$NewBodyShape = \sum_{i=1}^{8} \left[\min{}_i \times w_i + \max{}_i \times (1.0 - w_i)\right]$$

where $NewBodyShape$ is the new 3D model, the summation is across the eight data dimensions, and $\min_i$ and $\max_i$ correspond to the minimum and maximum models for each data dimension.

Step 4: Gender Interpolation

Because we have different ranges of body weight for males and females, we break the entire body weight range into three separate ranges. Each range corresponds to the correct gender property:

$$Gender \begin{cases} female & 41.3kg \le bodyweight \le 47.6kg; \\ female/male & 47.6kg \le bodyweight \le 96.7kg; \\ male & 96.7kg \le bodyweight \le 127.8kg; \end{cases}$$

Females and males share the second body weight range. If the body weight falls into this range, we are able to generate two new body shapes with the same body weight, one for a male and one for a female. Since our male and female models contain the same amount of polygons, we can interpolate between them to get a new gender based body shape:

$$NewBodyShape = FemaleBodyShape \times g + MaleBodyShape \times (1.0 - g)$$

where *g* is the weight we assign for the interpolation.

According to our above gender range function, when the body weight is smaller than 47.6kg, g is automatically set to 1.0, which generates a 100% female body; when the body weight is between 47.6kg and 96.7kg, g can be user adjusted, which provides the user with a gender based control ; when the body weight is heavier than 96.7kg, g is automatically set to 0.0, which generates a 100% male body shape.

CHAPTER IV

IMPLEMENTATION AND RESULTS

IV.1. PCA Program

The PCA program is written in C++. It uses the GNU Scientific Library (GSL1.3), a collection of numerical routines for scientific computing, to calculate the eigenvectors and eigenvalues of the covariance matrix of the original dataset, and to calculate a new dataset using the principal vectors. The new dataset is then sorted by body weight, and the four linear equations are also fitted with GSL library's linear regression functionality.

IV.2. Deformer Program

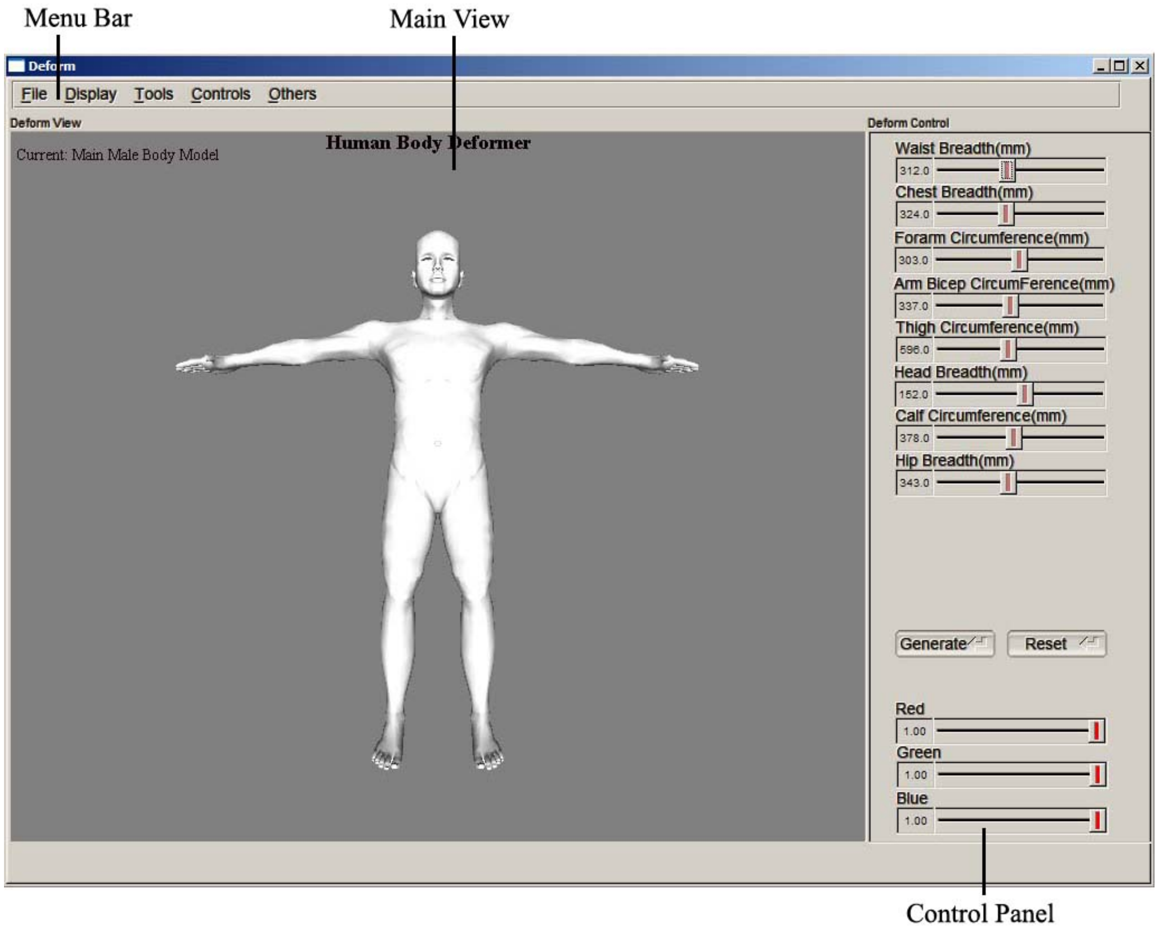
The Deformer program is written in C++, using the Fast Light Toolkit (FLTK), a cross-platform C++ GUI toolkit, for the interface programming. FLTK also includes an excellent UI builder called FLUID that can be used to create applications with simple UI in minutes. The Deformer's interface is directly built with FLUID with the help of extra custom C++ codes for user interaction. The display of the 3D models uses OpenGL. The pre-calculated principal vectors and linear equations are incorporated into the Deformer program to generate new 3D body models. The Deformer program can be compiled and run on Windows, Linux and MacOS X.

IV.2.1. Overview of user interface

The Deformer user interface refers to everything that the user sees and operates with Deformer. The menus, scene views, windows, and panels comprise the user interface. Through the Deformer interface one can access the features and operate the tools that allow the creation of three dimensional body shapes.

As shown in Fig. 9, the Deformer interface contains three main parts: the Menu Bar, the Main View, and the Control Panel.

The Menu Bar has five submenus: File, Display, Tools, Controls and Others. The File



Current: Main Male Body Model

Waist Breadth(mm) 312.0
Chest Breadth(mm) 324.0
Foram Circumference(mm) 303.0
Arm Bicep CircumFerence(mm) 337.0
Thigh Circumference(mm) 596.0
Head Breadth(mm) 152.0
Calf Circumference(mm) 378.0
Hip Breadth(mm) 343.0

Red 1.00
Green 1.00
Blue 1.00

Fig. 9. Deformer Application Window.

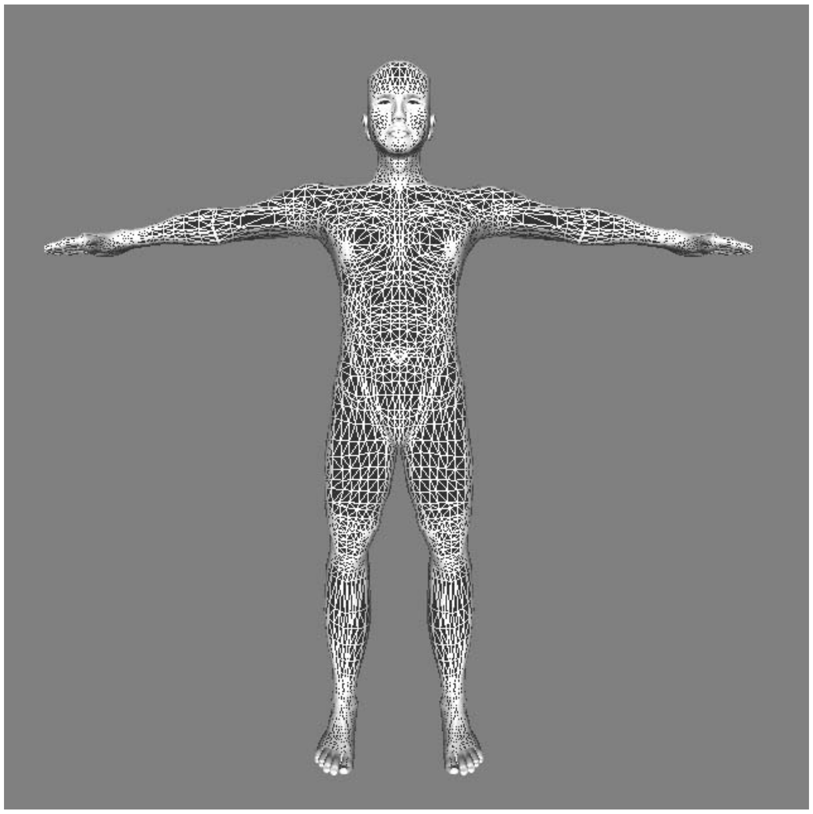


Fig. 10. Model Display in Wire-frame Mode.

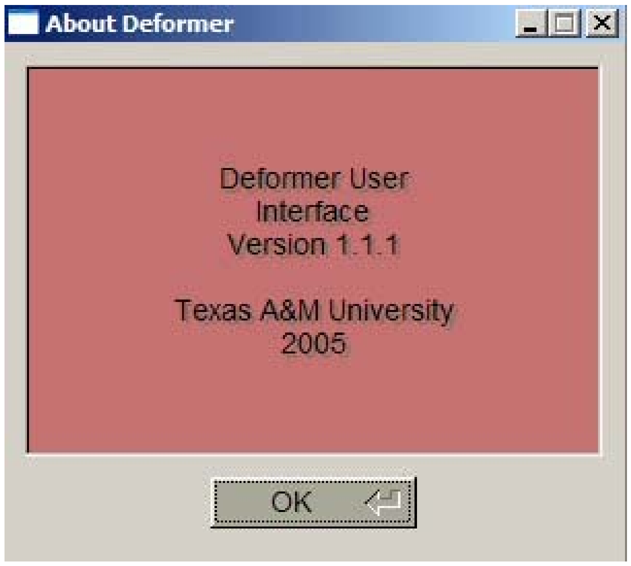
submenu has three menu items: Load Models, Load Obj File, and Save Obj File. Load Models automatically loads all the pre-generated 3D male and female models and initialize the control weights and control mode according to the male gender. Load Obj File and Save Obj File will load and save individual 3D model.

The Display sub menu has four menu items: Display Models, Smooth Shading, Wire Frame, and Show Normal. The Display Models option can switch the current model in the Main View to other pre-generated models. The Smooth Shading option uses averaged vertex normal or face normal to shade the model with OpenGL. The Wire Frame option displays the model in wire-frame mode in OpenGL, as shown in Fig. 10. The Show Normal option displays the face normal of the model. The default display mode uses vertex normal shading with wire-frame mode and face normal shading is turned off.

The Tools sub menu only contains one item: Reset. It has the same function as the Reset button in the Control Panel, which restores the original shape and setup of the current deformed model.

The Controls sub menu is the most important sub menu. It allows the user to switch to different control modes. These are: Normal Control Male, Normal Control Female, and Weight and Gender Control. The default control mode is Normal Control Male. Normal Control Male and Normal Control Female have the same control panel layout for direct control of the body parts. The Weight and Gender Control, however, has a different layout allowing for gender/weight based control of the entire body. The Others sub menu currently has only one menu item: About. It will pop up a small window that contains the basic information about the program, as shown in Fig. 11.

The Main View is the central window where the user can manipulate and display the 3D body model (see Fig. 12). The panel is labeled *Deform View* at the top to indicate that you are viewing the model from a perspective camera view. The user can zoom, pan and roll the model



Deformer User
Interface
Version 1.1.1

Texas A&M University
2005

Fig. 11. The “About” Popup Window.

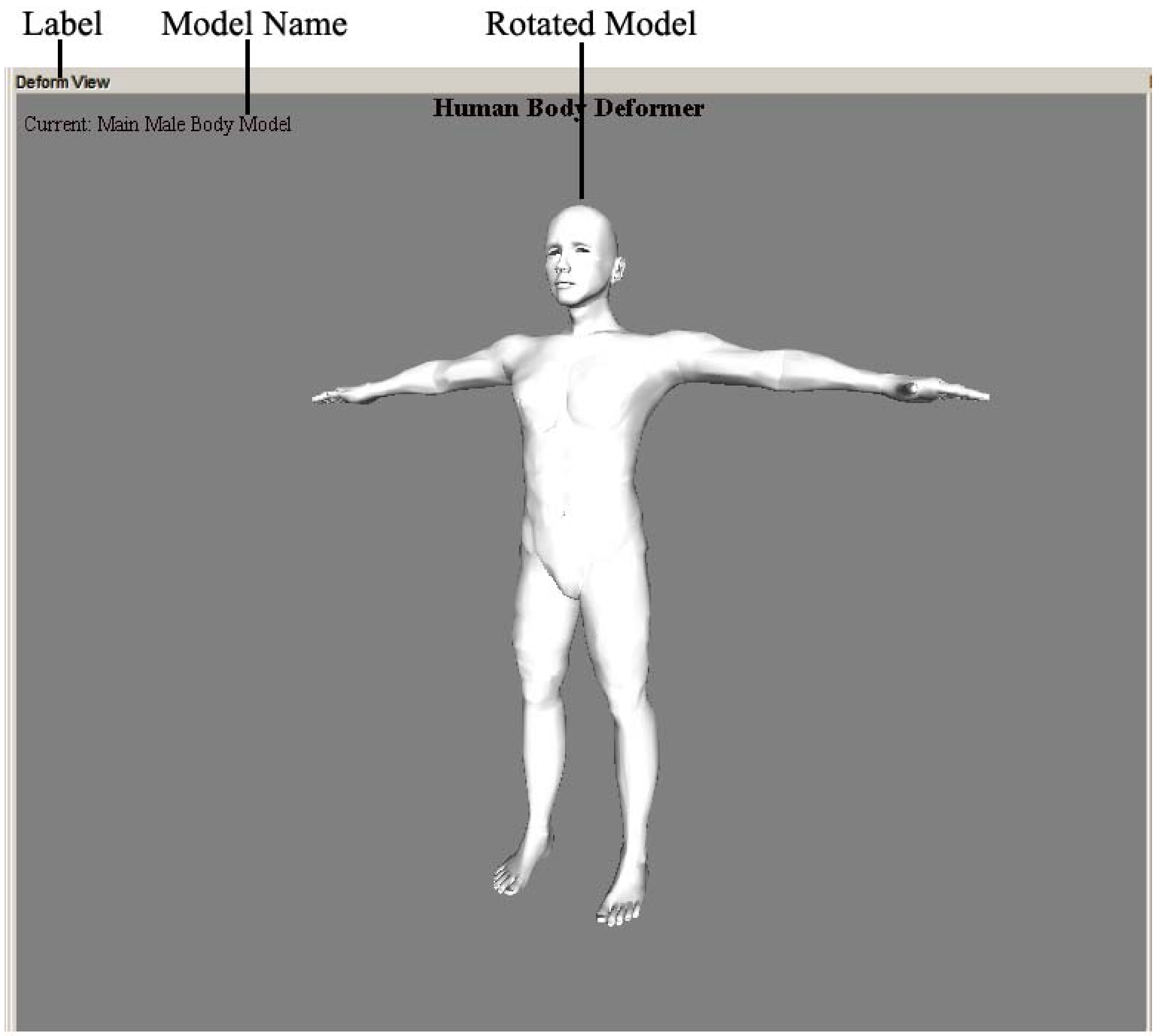


Fig. 12. Deformer Main View Window.

from a perspective camera view. The user can zoom, pan and roll the model with the left, middle and right mouse buttons. The current model name is displayed at the left top of the Main View window.

The Control Panel is where the shape of the current model can be changed (see Fig. 13). Depending on the control mode, which is found in the control submenu, the layout is different for the direct control and gender/weight based control. You can switch the control mode in Control submenu.

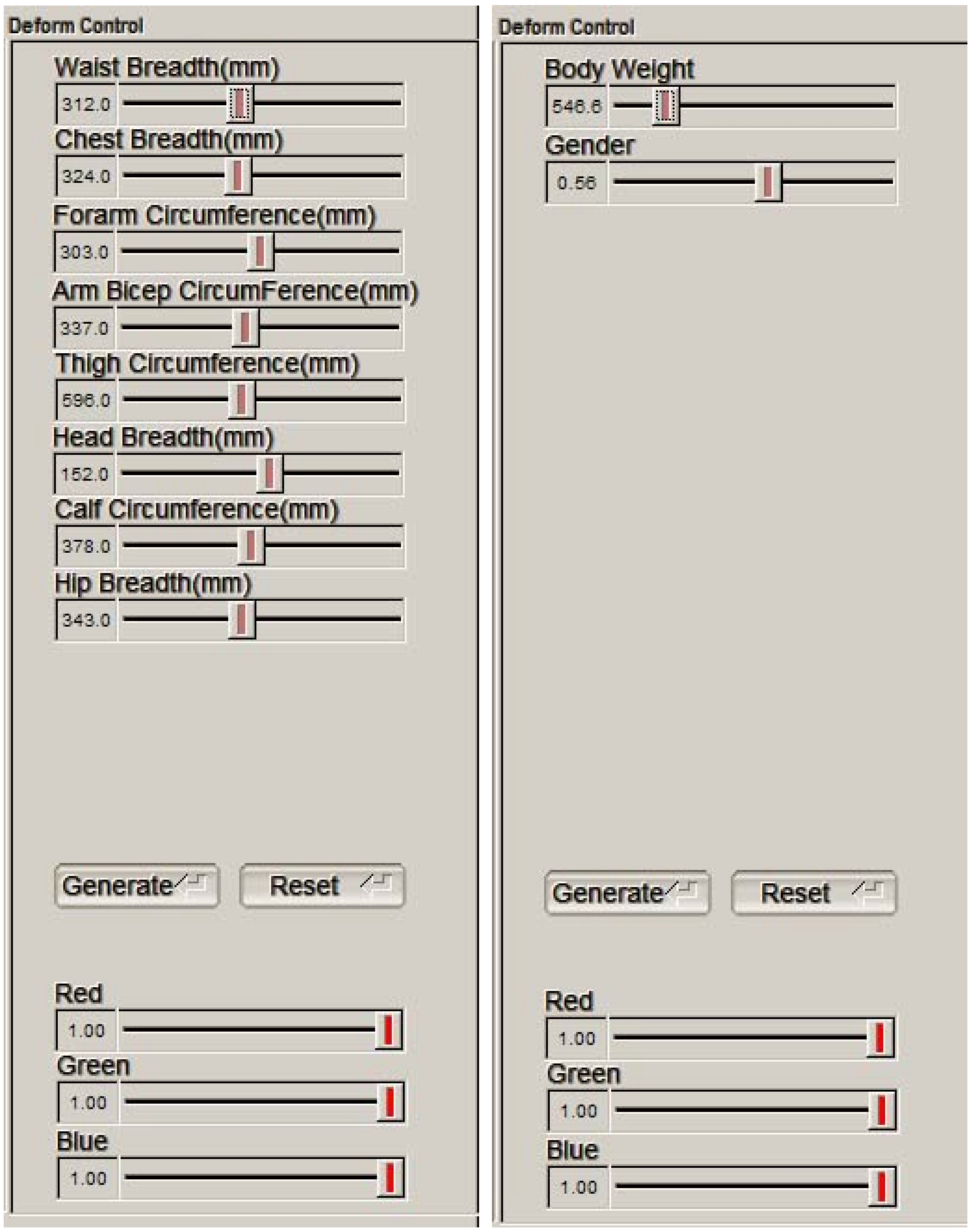
Moving the sliders displayed in the panel changes the interpolation weights. Each slider has a minimum and maximum value assigned to it. After the interpolation weights are set, pressing the Generate Button will generate the new body shape. The Reset button will reset the model to its initial shape. Adjusting the Red, Green and Blue sliders at the bottom of the control panel can change the color of the model.

IV.3. Results

The following images show results generated from the Deformer program. Because the user can control the model in two different modes, (direct control mode and gender/weight control mode) they will be shown separately.

As displayed in Fig. 14 and Fig. 15, the direct control mode allows great flexibility. It is difficult, however, to generate biologically plausible human bodies using this mode because the user must directly adjust the proportion of each body part to get the final result, and there is no guarantee that the final shape will be consistent with any of the statistical regularities expressed in the database.

In the Gender/Weight control mode, the user can easily get new body shapes for both males and females which follow the statistical patterns in the database. This control mode reduces our eight data dimensions to two much more user-friendly control parameters. One representative set of results is displayed in Fig. 16.



Waist Breadth(mm) 312.0
Chest Breadth(mm) 324.0
Foram Circumference(mm) 303.0
Arm Bicep CircumFerence(mm) 337.0
Thigh Circumference(mm) 596.0
Head Breadth(mm) 152.0
Calf Circumference(mm) 378.0
Hip Breadth(mm) 343.0
Red 1.00
Green 1.00
Blue 1.00

Body Weight 546.6
Gender 0.56
Red 1.00
Green 1.00
Blue 1.00

Fig. 13. The Control Panels.

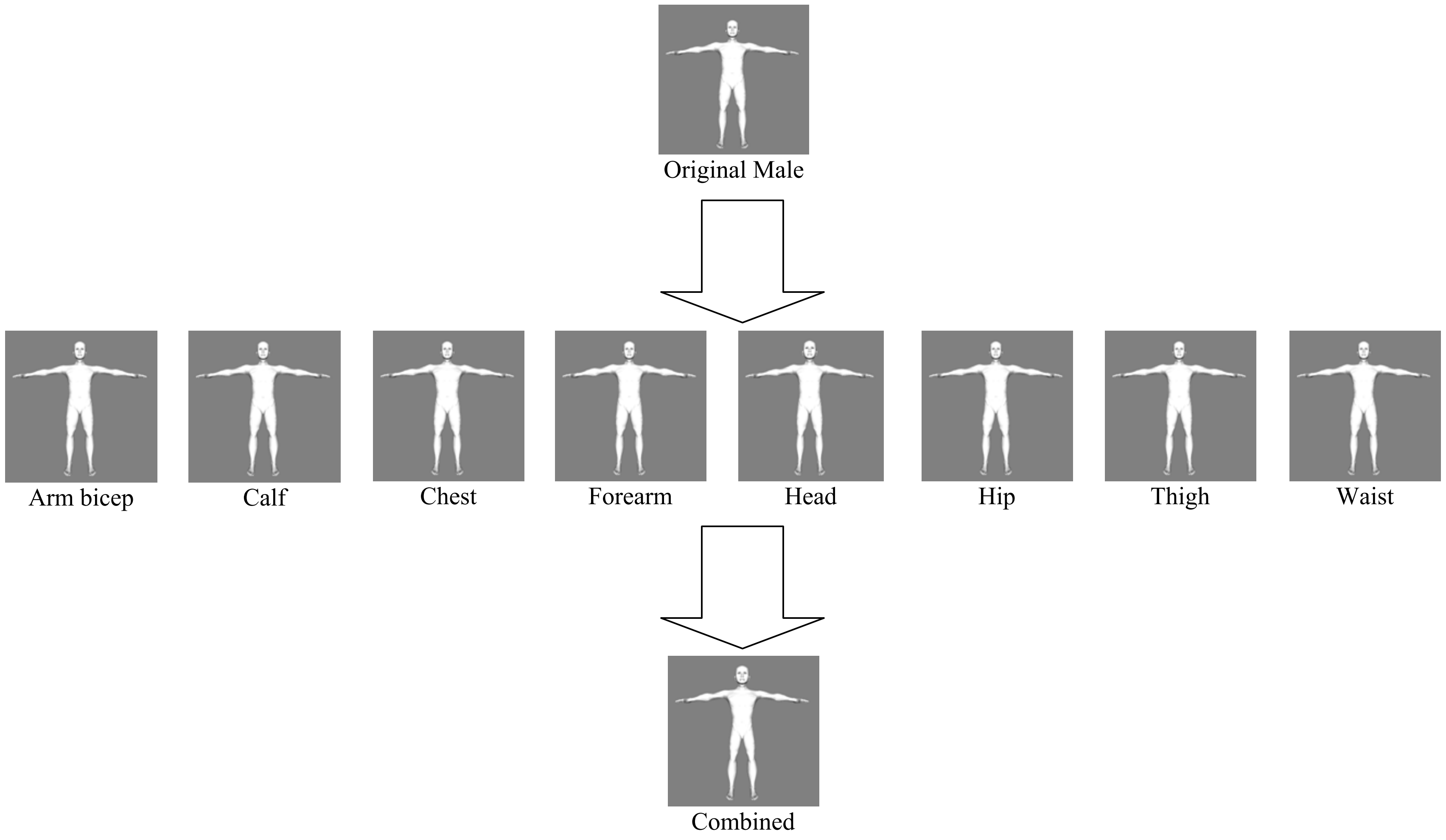


Fig. 14. Male Body in Direct Control Mode.

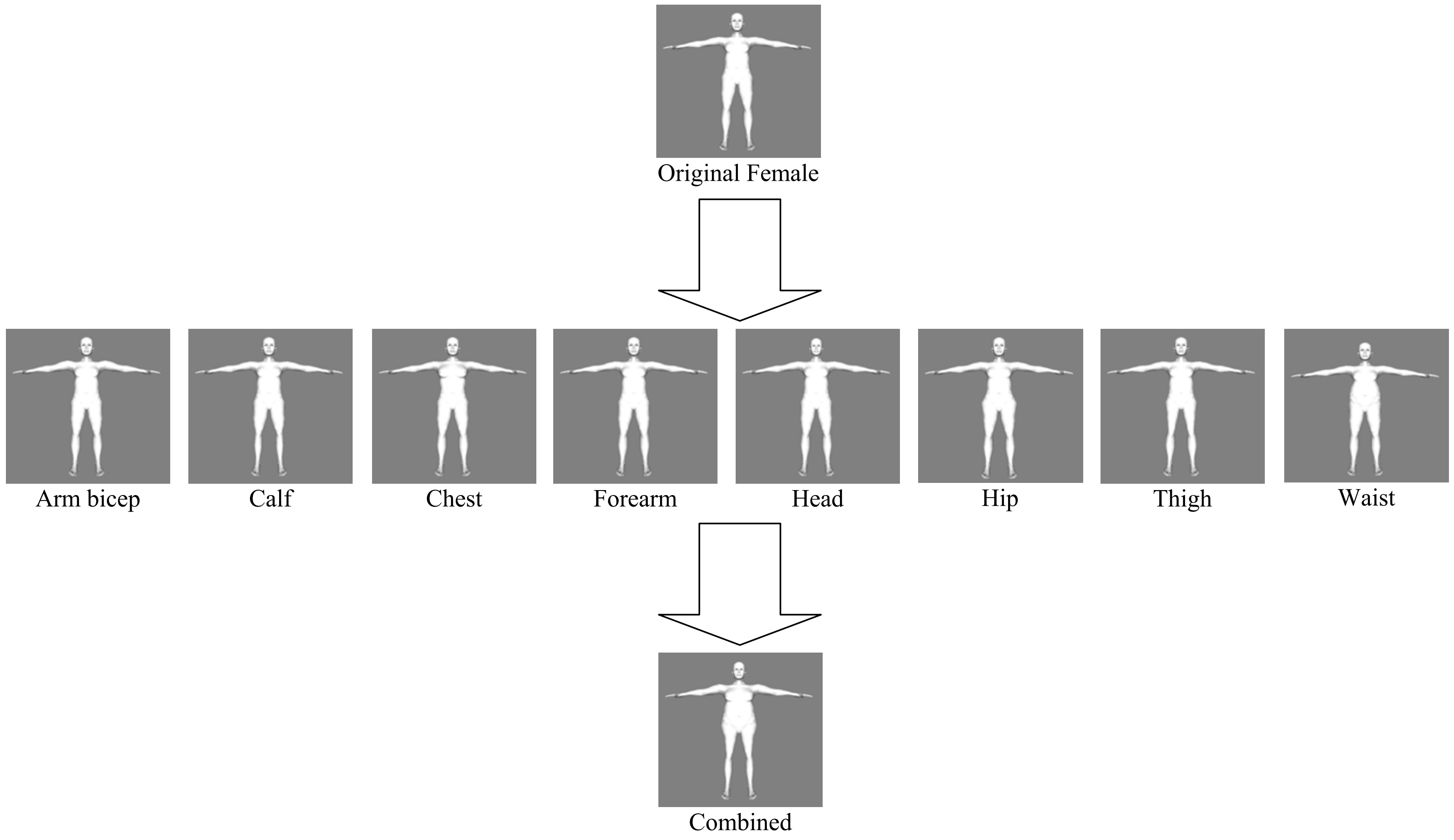


Fig. 15. Female Body in Direct Control Mode.

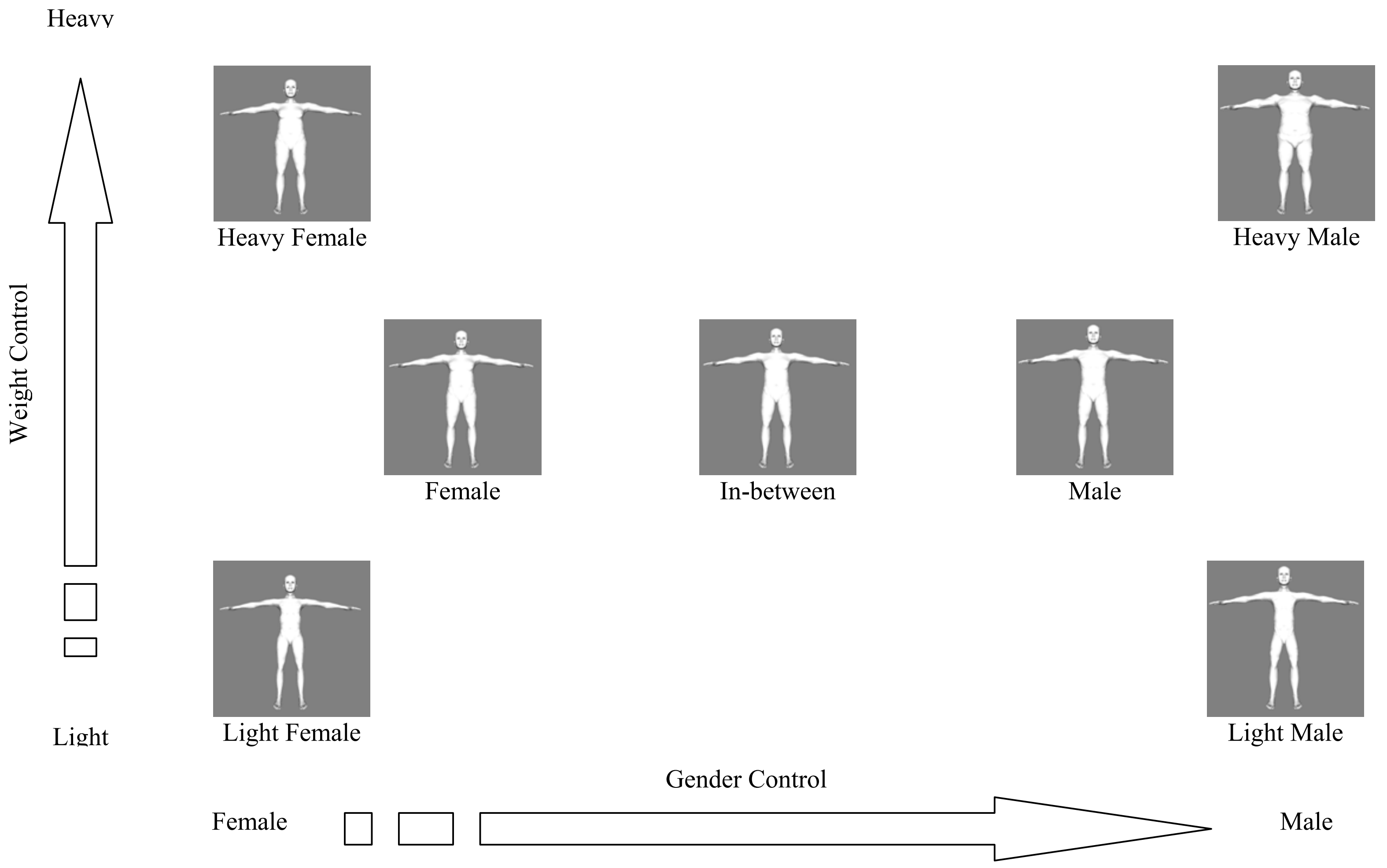


Fig. 16. Body Shapes in Gender/Weight Control Mode.

CHAPTER V

CONCLUSION AND FUTURE WORK

This thesis employed the PCA technique to analyze an anthropometric database, which contains measurement data from real people. The principal vectors derived from this analysis were used to map the data between the original database and the target model shape. This allowed us to create a gender/weight based high level of control on the top of the original control of the program based solely on a simplified linear relationship between dominant principal components and real body weight values. In addition, this analysis was developed into a standalone graphics application program. With this application, the user is able to:

1. Use linear interpolation techniques to interactively morph a series of existing 3D models based on the data from database.
2. Use Principal Component Analysis (PCA) to interactively manipulate the body shape within the normal range of human variation with a more intuitive interface based on gender and body weight.

Our application takes the 3D human body models as inputs. We provide users a base body shape and a series of target body shapes. The user can freely deform each body part separately, and we add all the relative changes to the base shape to get a final correct body shape. The output model of the program maintains all the necessary grouping information of the human body to allow it to be successfully exported to industry standard animation program.

Future research is needed to explore more accurate interpolation techniques than the simple linear interpolation scheme, especially on fitting a curve through the principal components points cluster. Future research is also needed to explore the utility of including more dimensions from the original database for our PCA, because the addition of more dimensions will likely result in more accurate body models. For example, the height of the body could be included. Furthermore, based on the result of this research, one might apply the same method to other bipedal character shapes, such as monsters or aliens, so that artists can generate different types of populations of species, and their shape features can still be manipulated by using the same high-level function. Real-time rendering and animation of the body is also a future research direction.

REFERENCES

[1] B. Allen, B. Curless, and Z. Popovic, “The space of all body shapes: reconstruction and parameterization from range scans”, *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 587-594, July 2003.

[2] B. Allen, B. Curless, and Z. Popovic, “Articulated body deformation from range scan data,” *ACM Transactions on Graphics*, vol. 21, no. 3, pp. 612-619, July 2002.

[3] R. B. Cattell, “The scree test for the number of factors,” *Multivariate Behavior Research*, vol.1, pp. 629-637, 1966.

[4] C. E. Clauser, I. O. Tebbetts, B. Bradtmiller, J. T. McConville, and C. C. Gordon. “Measurer’s Handbook: US Army Anthropometric Survey, 1987-1988.” Technical Report NATICK/TR-88/043, AD A202 721.

[5] R. C. Gonzalez and P.Wintz, *Digital Image Processing*, Reading, Mass: Addison-Wesley, 1987.

[6] C. C. Gordon, B. Bradtmiller, T. Churchill, C. E. Clauser, J. T. McConville, I. O. Tebbetts, and R. A. Walker. “1988 Anthropometric Survey of US army Personnel: Methods and Summary Statistics.” Technical Report NATICK/TR-89/044, AD A225 094.

[7] M. Lewis, “Evolving human figure geometry,” OSU-ACCAD-5/00- TR1, ACCAD, The Ohio State University, May 2002.

[8] M. Lewis, Richard Parent, “An implicit surface prototype for evolving human figure geometry,” OSU-ACCAD-11/00-TR2, ACCAD, The Ohio Sate University, November 2000.

[9] N. Troje, 2002, *Dynamic Perception,* Berlin: AKA Press, , pp. 115-120, 2002.

[10] N. Troje, “Decomposing biological motion: A framework for analysis and synthesis of human gait patterns,” *Journal of Vision 2002*, vol. 2, pp.371-387, 2002.

APPENDIX A

A.1 Correlation Matrix Calculation Source Code

```
#include <iostream.h>
#include <string.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include <gsl/gsl_statistics.h>
#include "Vector.h"
#include "Matrix.h"

void readdata(char *filename, FILE *input, unsigned int size, double *dim)
{
        char line[100];
        if((input = fopen(filename, "r")) == NULL)
        {
                cout << "Cannot open the file " << filename << endl;
                cout << "please check the file name again." << endl;
                exit(1);
        }
        unsigned int j = 0;
        while(!feof(input))
        {
                fgets(line, 100, input);
                if(*line != '#' && *line != '\n')       //pass the comment and blank lines
                {
                        sscanf(line, "%lf", &dim[j]);
                        j++;
                }

        }
        fclose(input);
}

int main(void)
{
        //reading the data for each category (alphabetic order) and calculate the covariance
        //term
        //for our database, female has 2230 samples, male has 2190 samples
        //so totally, for each dimension, the sample amount is: 2230 + 2190 = 4420
        double *dim1 = NULL;
        double *dim2 = NULL;
        FILE *input1 = NULL;
        FILE *input2 = NULL;
        unsigned int size = 4420;
        dim1 = new double[size];
        dim2 = new double[size];

        readdata("waist_brth_all.TXT", input1, size, dim1);
        readdata("chest_brth_all.TXT", input2, size, dim2);

        double mean1, mean2, std1, std2;
        mean1 = gsl_stats_mean (dim1, 1, size);
        mean2 = gsl_stats_mean (dim2, 1, size);
        std1 = gsl_stats_sd (dim1, 1, size);
        std2 = gsl_stats_sd (dim2, 1, size);

        for(unsigned int i = 0; i < size; i++)
        {
                dim1[i] -= mean1;
                dim2[i] -= mean2;
        }
        mean1 = gsl_stats_mean (dim1, 1, size);
        mean2 = gsl_stats_mean (dim2, 1, size);

        //calculate the covariance between dim1 and dim2
        double covariance, correlation;
        covariance = gsl_stats_covariance (dim1, 1, dim2, 1, size);
        correlation = covariance / (std1*std2);
                if(dim1 != NULL)
                delete dim1;
        if(dim2 != NULL)
                delete dim2;
        return 0;
}
```

A.2 Eigen System Solver Source Code

```
#include <iostream.h>
#include <string.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>

#include <gsl/gsl_statistics.h>
#include <gsl/gsl_math.h>
#include <gsl/gsl_eigen.h>

#include "Vector.h"
#include "Matrix.h"

int
main (void)
{
  ofstream fout;
  double data[] =
 {1.000000, 0.737978, 0.800701, 0.930497, 0.562943, 0.363683, 0.616672, 0.619527,
  0.737978, 1.000000, 0.673619, 0.715568, 0.437551, 0.540663, 0.731721, 0.604034,
  0.800701, 0.673619, 1.000000, 0.797360, 0.578927, 0.468673, 0.578380, 0.784363,
  0.930497, 0.715568, 0.797360, 1.000000, 0.597219, 0.276735, 0.510142, 0.548925,
  0.562943, 0.437551, 0.578927, 0.597219, 1.000000, 0.185234, 0.285491, 0.426335,
  0.363683, 0.540663, 0.468673, 0.276735, 0.185234, 1.000000, 0.752487, 0.715471,
  0.616672, 0.731721, 0.578380, 0.510142, 0.285491, 0.752487, 1.000000, 0.708804,
  0.619527, 0.604034, 0.784363, 0.548925, 0.426335, 0.715471, 0.708804, 1.000000};
  gsl_matrix_view m
    = gsl_matrix_view_array(data, 8, 8);

  gsl_vector *eval = gsl_vector_alloc (8);
  gsl_matrix *evec = gsl_matrix_alloc (8, 8);

  gsl_eigen_symmv_workspace * w =
    gsl_eigen_symmv_alloc (8);

  gsl_eigen_symmv (&m.matrix, eval, evec, w);

  gsl_eigen_symmv_free(w);

  gsl_eigen_symmv_sort (eval, evec,
                        GSL_EIGEN_SORT_VAL_DESC);

  fout.open("EigenOUT.txt");

  {
    int i, j;

    for (i = 0; i < 8; i++)
      {
        double eval_i
           = gsl_vector_get(eval, i);
                fout << "eigenvalue " << i << endl << eval_i << endl;
        gsl_vector_view evec_i
           = gsl_matrix_column(evec, i);

        printf("eigenvalue = %g\n", eval_i);
        printf("eigenvector = \n");
                fout << "eigenvector " << i << " " << endl;
                for(j=0; j < 8; j++)
                {
                        double v = gsl_vector_get (&evec_i.vector, j);
                        cout << v << " " ;
                        fout << v << " " ;
                }
                cout << endl;
                fout << endl << endl;
      }
  }
  return 0;
}
```

A.3 PCA Transformation Source Code

```
#include <iostream.h>
#include <string.h>
#include <fstream.h>
#include <stdio.h>
#include <stdlib.h>
#include "Vector.h"
```

```
#include "Matrix.h"

void readdata(FILE *input, unsigned int size, double *dim, double *data)
{
        char line[100];
        unsigned int i = 0;
        //arm_bicep_circ, calf_circ, chest_brth, forarm_circ, head_brth, hip_brth, thigh_circ,
        //waist_brth
        if((input = fopen("arm_bicep_circ_woman_sorted_adjusted.TXT", "r")) == NULL)
        {
                cout << "Cannot open the file " << endl;
                cout << "please check the file name again." << endl;
                exit(1);
        }
        unsigned int j = 0;
        while(!feof(input))
        {
                fgets(line, 100, input);
                if(*line != '#' && *line != '\n')       //pass the comment and blank lines
                {
                        sscanf(line, "%lf", &dim[j]);
                        j++;
                }
        }
        for(j = 0; j < size; j++)
        {
                data[i] = dim[j];
                i++;

        }
        fclose(input);

        if((input = fopen("calf_circ_woman_sorted_adjusted.TXT", "r")) == NULL)
        {
                cout << "Cannot open the file "  << endl;
                cout << "please check the file name again." << endl;
                exit(1);
        }
        j = 0;
        while(!feof(input))
        {
                fgets(line, 100, input);
                if(*line != '#' && *line != '\n')       //pass the comment and blank lines
                {
                        sscanf(line, "%lf", &dim[j]);
                        j++;
                }
        }
        for(j = 0; j < size; j++)
        {
                data[i] = dim[j];
                i++;
        }
        fclose(input);
```

```
if((input = fopen("chest_brth_woman_sorted_adjusted.TXT", "r")) == NULL)
{
        cout << "Cannot open the file "  << endl;
        cout << "please check the file name again." << endl;
        exit(1);
}
j = 0;
while(!feof(input))
{
        fgets(line, 100, input);
        if(*line != '#' && *line != '\n')       //pass the comment and blank lines
        {
                sscanf(line, "%lf", &dim[j]);
                j++;
        }
}
for(j = 0; j < size; j++)
{
        data[i] = dim[j];
        i++;
}
fclose(input);

if((input = fopen("forarm_circ_woman_sorted_adjusted.TXT", "r")) == NULL)
{
        cout << "Cannot open the file "  << endl;
        cout << "please check the file name again." << endl;
        exit(1);
}
j = 0;
while(!feof(input))
{
        fgets(line, 100, input);
        if(*line != '#' && *line != '\n')       //pass the comment and blank lines
        {
                sscanf(line, "%lf", &dim[j]);
                j++;
        }
}
for(j = 0; j < size; j++)
{
        data[i] = dim[j];
        i++;
}
fclose(input);

if((input = fopen("head_brth_woman_sorted_adjusted.TXT", "r")) == NULL)
{
        exit(1);
}
j = 0;
while(!feof(input))
{
        fgets(line, 100, input);
        if(*line != '#' && *line != '\n')       //pass the comment and blank lines
        {
                sscanf(line, "%lf", &dim[j]);
                j++;
        }
}
for(j = 0; j < size; j++)
{
        data[i] = dim[j];
        i++;
}
fclose(input);

if((input = fopen("hip_brth_woman_sorted_adjusted.TXT", "r")) == NULL)
{
        exit(1);
}
j = 0;
while(!feof(input))
{
        fgets(line, 100, input);
        if(*line != '#' && *line != '\n')       //pass the comment and blank lines
        {
                sscanf(line, "%lf", &dim[j]);
                j++;
        }
}
for(j = 0; j < size; j++)
{
        data[i] = dim[j];
        i++;
}
fclose(input);

if((input = fopen("thigh_circ_woman_sorted_adjusted.TXT", "r")) == NULL)
{
        cout << "Cannot open the file "  << endl;
        cout << "please check the file name again." << endl;
        exit(1);
}
j = 0;
while(!feof(input))
{
        fgets(line, 100, input);
        if(*line != '#' && *line != '\n')       //pass the comment and blank lines
        {
                sscanf(line, "%lf", &dim[j]);
                j++;
        }
}
for(j = 0; j < size; j++)
{
        data[i] = dim[j];
```

```
                i++;
        }
        fclose(input);

        if((input = fopen("waist_brth_woman_sorted_adjusted.TXT", "r")) == NULL)
        {
                cout << "Cannot open the file "  << endl;
                cout << "please check the file name again." << endl;
                exit(1);
        }
        j = 0;
        while(!feof(input))
        {
                fgets(line, 100, input);
                if(*line != '#' && *line != '\n')       //pass the comment and blank lines
                {
                        sscanf(line, "%lf", &dim[j]);
                        j++;
                }
        }
        for(j = 0; j < size; j++)
        {
                data[i] = dim[j];
                i++;

        }
        fclose(input);

}

void writedata(char *filename, FILE *output, Matrix &m)
{
        if((output = fopen(filename, "w")) == NULL)
        {
                cout << "Cannot open the file " << filename << endl;
                cout << "please check the file name again." << endl;
                exit(1);
        }
        fputc('#', output);
        fputs(filename, output);
        fputc('\n', output);

        int i = 0;
        int j = 0;
        for(j = 0; j < m.Ncols; j++)
        {
                for(i = 0; i < m.Nrows; i++)
                {
                        fprintf(output, "%lf    ", m.row[i][j]);

                }
                fputc('\n', output);
        }
        fclose(output);
}

int main(void)
{
        //reading the data for each category and calculate the covariance term
        //for our database, female has 2230 samples, male has 2190 samples
        //so totally, for each dimension, the sample amount is: 2230 + 2190 = 4420
        double *dim1 = NULL;
        FILE *input1 = NULL;
        FILE *output1 = NULL;
        unsigned int size = 2230;
        dim1 = new double[size];

        double M[] =
        {0.389586,0.373814,0.39121,0.372263,0.271651,0.287558,0.353002,0.368466,
         0.265639,-0.040108,0.14194,0.373591,0.439875,-0.599295,-0.390746,-0.250565,
        -0.295048,-0.387585,0.134994,-0.281084,0.676562,0.222938,-0.186087,0.351322,
         0.113059,-0.426968,0.476175,0.112673,-0.517943,-0.113688,-0.292977,0.445481,
         0.394034,-0.708308,-0.290964,0.214379,0.0624053,0.122529,0.415259,-0.145157,
        -0.0980046,-0.00331772,0.0466905,-0.297693,0.0478856,-0.686478,0.576253,0.306545,
         0.225942,0.140944,-0.695487,0.09188,-0.000100067,-0.0858078,-0.275517,0.594701,
         0.679637,0.0674847,0.106299,-0.697944,0.0229865,0.0435934,-0.147738,-0.104212};

        //unsigned int size_n = 17840; //2230 * 8;
        //unsigned int size_n = 17520; //2190 * 8;
        double data_adjusted_array[17840];
        readdata(input1, size, dim1, data_adjusted_array);

        Matrix m(8, 8, M);
        Matrix data_adjusted(8, 2230, data_adjusted_array);
        Matrix data_new(8, 2230, data_adjusted_array);

        data_new = m * data_adjusted;
        writedata("woman_new_data_corre.TXT", output1, data_new);

        if(dim1 != NULL)
                delete[] dim1;
        return 0;
}
```

A.3 Linear Regression Source Code

```
#include <cstdio>
#include <gsl/gsl_fit.h>

#define nman1  483

int main (void)
{
  int i;
  double x[nman1];
  double y1[nman1];
  double y2[nman1];
  double w[nman1];

  double c0, c1, cov00, cov01, cov11, chisq;

  FILE *fp;
  fp=fopen("man1.txt","r");

  for(i=0;i<nman1;i++)
  {
          fscanf(fp,"%lf,%lf,%lf",&x[i],&y1[i],&y2[i]);
          w[i]=1.0;
  }

  gsl_fit_wlinear (x, 1, w, 1, y1, 1, nman1,
                   &c0, &c1, &cov00, &cov01, &cov11,
                   &chisq);

  printf("# best fit: Y = %g + %g X\n", c0, c1);

  gsl_fit_wlinear (x, 1, w, 1, y2, 1, nman1,
                   &c0, &c1, &cov00, &cov01, &cov11,
                   &chisq);
  printf("# best fit: Y = %g + %g X\n", c0, c1);
  return 0;
}
```

APPENDIX B

B.1 Deformer User Interface Header File Source Code

```
// generated by Fast Light User Interface Designer (fluid) version 1.0103
#ifndef DeformUI_h
#define DeformUI_h
#include <FL/Fl.H>
#include <FL/Fl_Double_Window.H>
#include <FL/Fl_Group.H>
#include <FL/Fl_Menu_Bar.H>
#include <FL/Fl_Value_Slider.H>
#include <FL/Fl_Return_Button.H>
#include "DeformView.h"
#include <FL/Fl_Box.H>

class DeformUI {
public:
  DeformUI();
private:
  Fl_Double_Window *mainWindow;
public:
  Fl_Group *All_UI;
  Fl_Group *MainMenu;
  Fl_Menu_Bar *MainMenuBar;
  static Fl_Menu_Item menu_MainMenuBar[];
private:
  static Fl_Menu_Item *Menu_File;
  static Fl_Menu_Item *Menu_Load_Models;
  inline void cb_Menu_Load_Models_i(Fl_Menu_*, void*);
  static void cb_Menu_Load_Models(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Load_Male;
  inline void cb_Menu_Load_Male_i(Fl_Menu_*, void*);
  static void cb_Menu_Load_Male(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Load_Female;
  inline void cb_Menu_Load_Female_i(Fl_Menu_*, void*);
  static void cb_Menu_Load_Female(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_LoadObj;
  inline void cb_Menu_LoadObj_i(Fl_Menu_*, void*);
  static void cb_Menu_LoadObj(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_SaveObj;
  inline void cb_Menu_SaveObj_i(Fl_Menu_*, void*);
  static void cb_Menu_SaveObj(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Display;
  static Fl_Menu_Item *Menu_Models;
  static Fl_Menu_Item *Menu_MainModel;
  inline void cb_Menu_MainModel_i(Fl_Menu_*, void*);
  static void cb_Menu_MainModel(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_SourceModel;
  inline void cb_Menu_SourceModel_i(Fl_Menu_*, void*);
  static void cb_Menu_SourceModel(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_TargetModel_Waist;
  inline void cb_Menu_TargetModel_Waist_i(Fl_Menu_*, void*);
  static void cb_Menu_TargetModel_Waist(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_TargetModel_Chest;
  inline void cb_Menu_TargetModel_Chest_i(Fl_Menu_*, void*);
  static void cb_Menu_TargetModel_Chest(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_TargetModel_Forarm;
  inline void cb_Menu_TargetModel_Forarm_i(Fl_Menu_*, void*);
  static void cb_Menu_TargetModel_Forarm(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_TargetModel_Bicep;
  inline void cb_Menu_TargetModel_Bicep_i(Fl_Menu_*, void*);
  static void cb_Menu_TargetModel_Bicep(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_TargetModel_Thigh;
  inline void cb_Menu_TargetModel_Thigh_i(Fl_Menu_*, void*);
  static void cb_Menu_TargetModel_Thigh(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_TargetModel_Head;
  inline void cb_Menu_TargetModel_Head_i(Fl_Menu_*, void*);
  static void cb_Menu_TargetModel_Head(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_TargetModel_Calf;
  inline void cb_Menu_TargetModel_Calf_i(Fl_Menu_*, void*);
  static void cb_Menu_TargetModel_Calf(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_TargetModel_Hip;
  inline void cb_Menu_TargetModel_Hip_i(Fl_Menu_*, void*);
  static void cb_Menu_TargetModel_Hip(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Smooth;
  inline void cb_Menu_Smooth_i(Fl_Menu_*, void*);
  static void cb_Menu_Smooth(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Wireframe;
  inline void cb_Menu_Wireframe_i(Fl_Menu_*, void*);
  static void cb_Menu_Wireframe(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Normal;
  inline void cb_Menu_Normal_i(Fl_Menu_*, void*);
  static void cb_Menu_Normal(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Tools;
  static Fl_Menu_Item *Menu_Reset_Model;
  inline void cb_Menu_Reset_Model_i(Fl_Menu_*, void*);
  static void cb_Menu_Reset_Model(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Interpolation;
  static Fl_Menu_Item *Menu_Linear;
  inline void cb_Menu_Linear_i(Fl_Menu_*, void*);
  static void cb_Menu_Linear(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Cosine;
  inline void cb_Menu_Cosine_i(Fl_Menu_*, void*);
  static void cb_Menu_Cosine(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Cubic;
  inline void cb_Menu_Cubic_i(Fl_Menu_*, void*);
  static void cb_Menu_Cubic(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Controls;
  static Fl_Menu_Item *Menu_Normal_Control_Male;
  inline void cb_Menu_Normal_Control_Male_i(Fl_Menu_*, void*);
  static void cb_Menu_Normal_Control_Male(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Normal_Control_Female;
  inline void cb_Menu_Normal_Control_Female_i(Fl_Menu_*, void*);
  static void cb_Menu_Normal_Control_Female(Fl_Menu_*, void*);
```

```
  static Fl_Menu_Item *Menu_Weight_Gender_Control;
  inline void cb_Menu_Weight_Gender_Control_i(Fl_Menu_*, void*);
  static void cb_Menu_Weight_Gender_Control(Fl_Menu_*, void*);
  static Fl_Menu_Item *Menu_Help;
  static Fl_Menu_Item *Menu_Contents;
  static Fl_Menu_Item *Menu_Feedback;
  static Fl_Menu_Item *Menu_About;
  inline void cb_Menu_About_i(Fl_Menu_*, void*);
  static void cb_Menu_About(Fl_Menu_*, void*);
public:
  Fl_Group *MainControlPanel;
private:
  Fl_Value_Slider *Slider_Shape_Blend_Waist;
  inline void cb_Slider_Shape_Blend_Waist_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Shape_Blend_Waist(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Shape_Blend_Chest;
  inline void cb_Slider_Shape_Blend_Chest_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Shape_Blend_Chest(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Shape_Blend_Forarm;
  inline void cb_Slider_Shape_Blend_Forarm_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Shape_Blend_Forarm(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Shape_Blend_Bicep;
  inline void cb_Slider_Shape_Blend_Bicep_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Shape_Blend_Bicep(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Shape_Blend_Thigh;
  inline void cb_Slider_Shape_Blend_Thigh_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Shape_Blend_Thigh(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Shape_Blend_Head;
  inline void cb_Slider_Shape_Blend_Head_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Shape_Blend_Head(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Shape_Blend_Calf;
  inline void cb_Slider_Shape_Blend_Calf_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Shape_Blend_Calf(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Shape_Blend_Hip;
  inline void cb_Slider_Shape_Blend_Hip_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Shape_Blend_Hip(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Body_Weight;
  inline void cb_Slider_Body_Weight_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Body_Weight(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Gender;
  inline void cb_Slider_Gender_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Gender(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Red;
  inline void cb_Slider_Red_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Red(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Green;
  inline void cb_Slider_Green_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Green(Fl_Value_Slider*, void*);
  Fl_Value_Slider *Slider_Blue;
  inline void cb_Slider_Blue_i(Fl_Value_Slider*, void*);
  static void cb_Slider_Blue(Fl_Value_Slider*, void*);
  Fl_Return_Button *Button_Generate;
  inline void cb_Button_Generate_i(Fl_Return_Button*, void*);
  static void cb_Button_Generate(Fl_Return_Button*, void*);
  Fl_Return_Button *Button_Reset;
  inline void cb_Button_Reset_i(Fl_Return_Button*, void*);
  static void cb_Button_Reset(Fl_Return_Button*, void*);
public:
  Fl_Group *MainView;
  DeformView *DeformObj;
private:
  Fl_Double_Window *aboutWindow;
  inline void cb_OK_i(Fl_Return_Button*, void*);
  static void cb_OK(Fl_Return_Button*, void*);
public:
  void show(int argc, char **argv);
};
#endif
```

B.2 Deformer User Interface Source File Source Code

```
// generated by Fast Light User Interface Designer (fluid) version 1.0103

#include "DeformUI.h"
#include <stdio.h>
#include <stdlib.h>
#include <FL/FL_File_Chooser.H>

inline void DeformUI::cb_Menu_Load_Models_i(Fl_Menu_*, void*) {
  //load the generic model as a male model
DeformObj->mainModel_male_generic.Clean();
DeformObj->mainModel_male_generic.ReadObjFile("model/male/male_orig.obj");
DeformObj->mainModel_generic.Clean();
DeformObj->mainModel_generic.ReadObjFile("model/male/male_orig.obj");
DeformObj->mainModel_female_generic.Clean();
DeformObj->mainModel_female_generic.ReadObjFile("model/female/female_orig.obj");

//directly load all the male models
DeformObj->mainModel_male.Clean();
DeformObj->sourceModel_male.Clean();
DeformObj->targetModel_waist_max_male.Clean();
DeformObj->targetModel_chest_max_male.Clean();
DeformObj->targetModel_forarm_max_male.Clean();
DeformObj->targetModel_bicep_max_male.Clean();
DeformObj->targetModel_thigh_max_male.Clean();
DeformObj->targetModel_head_max_male.Clean();
DeformObj->targetModel_hip_max_male.Clean();
DeformObj->targetModel_calf_max_male.Clean();
DeformObj->targetModel_waist_min_male.Clean();
DeformObj->targetModel_chest_min_male.Clean();
DeformObj->targetModel_forarm_min_male.Clean();
DeformObj->targetModel_bicep_min_male.Clean();
DeformObj->targetModel_thigh_min_male.Clean();
DeformObj->targetModel_head_min_male.Clean();
DeformObj->targetModel_hip_min_male.Clean();
```

```
DeformObj->targetModel_calf_min_male.Clean();

DeformObj->mainModel_male.ReadObjFile("model/male/male_orig.obj");
DeformObj->sourceModel_male.ReadObjFile("model/male/male_orig.obj");
DeformObj->targetModel_waist_max_male.ReadObjFile("model/male/male_belly_max.obj");
DeformObj->targetModel_chest_max_male.ReadObjFile("model/male/male_chest_max.obj");
DeformObj->targetModel_forarm_max_male.ReadObjFile("model/male/male_forarm_max.obj");
DeformObj->targetModel_bicep_max_male.ReadObjFile("model/male/male_bicep_max.obj");
DeformObj->targetModel_thigh_max_male.ReadObjFile("model/male/male_thigh_max.obj");
DeformObj->targetModel_head_max_male.ReadObjFile("model/male/male_head_max.obj");
DeformObj->targetModel_calf_max_male.ReadObjFile("model/male/male_calf_max.obj");
DeformObj->targetModel_hip_max_male.ReadObjFile("model/male/male_hip_max.obj");
DeformObj->targetModel_waist_min_male.ReadObjFile("model/male/male_belly_min.obj");
DeformObj->targetModel_chest_min_male.ReadObjFile("model/male/male_chest_min.obj");
DeformObj->targetModel_forarm_min_male.ReadObjFile("model/male/male_forarm_min.obj");
DeformObj->targetModel_bicep_min_male.ReadObjFile("model/male/male_bicep_min.obj");
DeformObj->targetModel_thigh_min_male.ReadObjFile("model/male/male_thigh_min.obj");
DeformObj->targetModel_head_min_male.ReadObjFile("model/male/male_head_min.obj");
DeformObj->targetModel_calf_min_male.ReadObjFile("model/male/male_calf_min.obj");
DeformObj->targetModel_hip_min_male.ReadObjFile("model/male/male_hip_min.obj");

//directly load all the female models
DeformObj->mainModel_female.Clean();
DeformObj->sourceModel_female.Clean();
DeformObj->targetModel_waist_max_female.Clean();
DeformObj->targetModel_chest_max_female.Clean();
DeformObj->targetModel_forarm_max_female.Clean();
DeformObj->targetModel_bicep_max_female.Clean();
DeformObj->targetModel_thigh_max_female.Clean();
DeformObj->targetModel_head_max_female.Clean();
DeformObj->targetModel_hip_max_female.Clean();
DeformObj->targetModel_calf_max_female.Clean();
DeformObj->targetModel_waist_min_female.Clean();
DeformObj->targetModel_chest_min_female.Clean();
DeformObj->targetModel_forarm_min_female.Clean();
DeformObj->targetModel_bicep_min_female.Clean();
DeformObj->targetModel_thigh_min_female.Clean();
DeformObj->targetModel_head_min_female.Clean();
DeformObj->targetModel_hip_min_female.Clean();
DeformObj->targetModel_calf_min_female.Clean();

DeformObj->mainModel_female.ReadObjFile("model/female/female_orig.obj");
DeformObj->sourceModel_female.ReadObjFile("model/female/female_orig.obj");
DeformObj->targetModel_waist_max_female.ReadObjFile("model/female/female_belly_max.obj");
DeformObj->targetModel_chest_max_female.ReadObjFile("model/female/female_chest_max.obj");
DeformObj->targetModel_forarm_max_female.ReadObjFile("model/female/female_forarm_max.obj");
DeformObj->targetModel_bicep_max_female.ReadObjFile("model/female/female_bicep_max.obj");
DeformObj->targetModel_thigh_max_female.ReadObjFile("model/female/female_thigh_max.obj");
DeformObj->targetModel_head_max_female.ReadObjFile("model/female/female_head_max.obj");
DeformObj->targetModel_calf_max_female.ReadObjFile("model/female/female_calf_max.obj");
DeformObj->targetModel_hip_max_female.ReadObjFile("model/female/female_hip_max.obj");
DeformObj->targetModel_waist_min_female.ReadObjFile("model/female/female_belly_min.obj");
DeformObj->targetModel_chest_min_female.ReadObjFile("model/female/female_chest_min.obj");
DeformObj->targetModel_forarm_min_female.ReadObjFile("model/female/female_forarm_min.obj");
```

```
DeformObj->targetModel_bicep_min_female.ReadObjFile("model/female/female_bicep_min.obj");
DeformObj->targetModel_thigh_min_female.ReadObjFile("model/female/female_thigh_min.obj");
DeformObj->targetModel_head_min_female.ReadObjFile("model/female/female_head_min.obj");
DeformObj->targetModel_calf_min_female.ReadObjFile("model/female/female_calf_min.obj");
DeformObj->targetModel_hip_min_female.ReadObjFile("model/female/female_hip_min.obj");

/////////////////////////////////////////////////////////////////////////////////////////
//Set all the average values
DeformObj->waist_avg_male = 312.0;
DeformObj->chest_avg_male = 324.0;
DeformObj->forarm_avg_male = 303.0;
DeformObj->bicep_avg_male = 337.0;
DeformObj->thigh_avg_male = 596.0;
DeformObj->head_avg_male = 152.0;
DeformObj->hip_avg_male = 343.0;
DeformObj->calf_avg_male = 378.0;

DeformObj->waist_avg_female = 291.0;
DeformObj->chest_avg_female = 281.0;
DeformObj->forarm_avg_female = 254.0;
DeformObj->bicep_avg_female = 281.0;
DeformObj->thigh_avg_female = 581.0;
DeformObj->head_avg_female = 144.0;
DeformObj->hip_avg_female = 344.0;
DeformObj->calf_avg_female = 353.0;

//Reset all the slider to the normal male body type
Slider_Shape_Blend_Waist->value(312);
Slider_Shape_Blend_Chest->value(324);
Slider_Shape_Blend_Forarm->value(303);
Slider_Shape_Blend_Bicep->value(337);
Slider_Shape_Blend_Thigh->value(596);
Slider_Shape_Blend_Head->value(152);
Slider_Shape_Blend_Calf->value(378);
Slider_Shape_Blend_Hip->value(343);

//Set the range for the sliders (male)
Slider_Shape_Blend_Waist->minimum(235);
Slider_Shape_Blend_Waist->maximum(422);
Slider_Shape_Blend_Chest->minimum(257);
Slider_Shape_Blend_Chest->maximum(422);
Slider_Shape_Blend_Forarm->minimum(233);
Slider_Shape_Blend_Forarm->maximum(372);
Slider_Shape_Blend_Bicep->minimum(259);
Slider_Shape_Blend_Bicep->maximum(437);
Slider_Shape_Blend_Thigh->minimum(458);
Slider_Shape_Blend_Thigh->maximum(787);
Slider_Shape_Blend_Head->minimum(128);
Slider_Shape_Blend_Head->maximum(173);
Slider_Shape_Blend_Calf->minimum(300);
Slider_Shape_Blend_Calf->maximum(470);
Slider_Shape_Blend_Hip->minimum(282);
Slider_Shape_Blend_Hip->maximum(428);
```

```
////////////////////////////////////////////////////////////////////////////////////////
Button_Generate->activate();
//DeformObj->model_gender = 0;  //default control is normal female
DeformObj->model_loaded = true;
DeformObj->initialize();        //initialize the eigen lines for male and female
                                //, pca matrix and vector, read in the data from files
Slider_Shape_Blend_Waist->show();
Slider_Shape_Blend_Chest->show();
Slider_Shape_Blend_Forarm->show();
Slider_Shape_Blend_Bicep->show();
Slider_Shape_Blend_Thigh->show();
Slider_Shape_Blend_Head->show();
Slider_Shape_Blend_Calf->show();
Slider_Shape_Blend_Hip->show();
Slider_Body_Weight->hide();
Slider_Gender->deactivate();
Slider_Gender->hide();

DeformObj->control_type = 0;
DeformObj->redraw();
}
void DeformUI::cb_Menu_Load_Models(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_Load_Models_i(o,v);
}

inline void DeformUI::cb_Menu_Load_Male_i(Fl_Menu_*, void*) {

}
void DeformUI::cb_Menu_Load_Male(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_Load_Male_i(o,v);
}

inline void DeformUI::cb_Menu_Load_Female_i(Fl_Menu_*, void*) {

}
void DeformUI::cb_Menu_Load_Female(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_Load_Female_i(o,v);
}

inline void DeformUI::cb_Menu_LoadObj_i(Fl_Menu_*, void*) {
  char *newfile = fl_file_chooser("Save Obj File?", "OBJ Files (*.obj)", DeformObj-
>mainfilename);
if(newfile != NULL)
{
        //load source obj file
        strcpy(DeformObj->mainfilename, newfile);
        DeformObj->mainModel.Clean();
        DeformObj->mainModel.ReadObjFile(newfile);

        DeformObj->loadModelInfos();

}
DeformObj->redraw();
}
```

```
void DeformUI::cb_Menu_LoadObj(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_LoadObj_i(o,v);
}

inline void DeformUI::cb_Menu_SaveObj_i(Fl_Menu_*, void*) {
  char *newfile = fl_file_chooser("Save Obj File?", "OBJ Files (*.obj)", DeformObj-
>mainfilename);
if(newfile != NULL)
{
        //load source obj file
        strcpy(DeformObj->mainfilename, newfile);
        if(DeformObj->control_type == 0)
                DeformObj->mainModel_male.WriteObjFile(newfile);
        else if(DeformObj->control_type == 1)
                DeformObj->mainModel_female.WriteObjFile(newfile);
        else if(DeformObj->control_type == 2)
                DeformObj->mainModel_generic.WriteObjFile(newfile);
        DeformObj->saveModelInfos();
}
DeformObj->redraw();
}
void DeformUI::cb_Menu_SaveObj(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_SaveObj_i(o,v);
}

inline void DeformUI::cb_Menu_MainModel_i(Fl_Menu_*, void*) {
  DeformObj->model_id = 0;
}
void DeformUI::cb_Menu_MainModel(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_MainModel_i(o,v);
}

inline void DeformUI::cb_Menu_SourceModel_i(Fl_Menu_*, void*) {
  DeformObj->model_id = 1;
}
void DeformUI::cb_Menu_SourceModel(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_SourceModel_i(o,v);
}

inline void DeformUI::cb_Menu_TargetModel_Waist_i(Fl_Menu_*, void*) {
  DeformObj->model_id = 2;
}
void DeformUI::cb_Menu_TargetModel_Waist(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_TargetModel_Waist_i(o,v);
}

inline void DeformUI::cb_Menu_TargetModel_Chest_i(Fl_Menu_*, void*) {
  DeformObj->model_id = 3;
}
void DeformUI::cb_Menu_TargetModel_Chest(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_TargetModel_Chest_i(o,v);
}

inline void DeformUI::cb_Menu_TargetModel_Forarm_i(Fl_Menu_*, void*) {
  DeformObj->model_id = 4;
}
void DeformUI::cb_Menu_TargetModel_Forarm(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_TargetModel_Forarm_i(o,v);
}

inline void DeformUI::cb_Menu_TargetModel_Bicep_i(Fl_Menu_*, void*) {
  DeformObj->model_id = 5;
}
void DeformUI::cb_Menu_TargetModel_Bicep(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_TargetModel_Bicep_i(o,v);
}

inline void DeformUI::cb_Menu_TargetModel_Thigh_i(Fl_Menu_*, void*) {
  DeformObj->model_id = 6;
}
void DeformUI::cb_Menu_TargetModel_Thigh(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_TargetModel_Thigh_i(o,v);
}

inline void DeformUI::cb_Menu_TargetModel_Head_i(Fl_Menu_*, void*) {
  DeformObj->model_id = 7;
}
void DeformUI::cb_Menu_TargetModel_Head(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_TargetModel_Head_i(o,v);
}

inline void DeformUI::cb_Menu_TargetModel_Calf_i(Fl_Menu_*, void*) {
  DeformObj->model_id = 8;
}
void DeformUI::cb_Menu_TargetModel_Calf(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_TargetModel_Calf_i(o,v);
}

inline void DeformUI::cb_Menu_TargetModel_Hip_i(Fl_Menu_*, void*) {
  DeformObj->model_id = 9;
}
void DeformUI::cb_Menu_TargetModel_Hip(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_TargetModel_Hip_i(o,v);
}

inline void DeformUI::cb_Menu_Smooth_i(Fl_Menu_* o, void*) {
  int i;

for(i = 0; i < DeformObj->mainModel_generic.numOfObjects; i++)
{
        if(DeformObj->mainModel_generic.pObject[i]->g_smothness == 1)
        //if(((Fl_Menu_Item *)o)->value() = 1)
                DeformObj->mainModel_generic.pObject[i]->g_smothness = 2;
        else
                DeformObj->mainModel_generic.pObject[i]->g_smothness = 1;
}

for(i = 0; i < DeformObj->mainModel_male.numOfObjects; i++)
```

```
{
        if(DeformObj->mainModel_male.pObject[i]->g_smothness == 1)
        //if(((Fl_Menu_Item *)o)->value() = 1)
                DeformObj->mainModel_male.pObject[i]->g_smothness = 2;
        else
                DeformObj->mainModel_male.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->sourceModel_male.numOfObjects; i++)
{
        if(DeformObj->sourceModel_male.pObject[i]->g_smothness == 1)
                DeformObj->sourceModel_male.pObject[i]->g_smothness = 2;
        else
                DeformObj->sourceModel_male.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_waist_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_waist_max_male.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_waist_max_male.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_waist_max_male.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_chest_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_chest_max_male.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_chest_max_male.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_chest_max_male.pObject[i]->g_smothness = 1;
}

for(i = 0; i < DeformObj->targetModel_forarm_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_forarm_max_male.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_forarm_max_male.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_forarm_max_male.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_calf_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_calf_max_male.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_calf_max_male.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_calf_max_male.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_hip_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_hip_max_male.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_hip_max_male.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_hip_max_male.pObject[i]->g_smothness = 1;
}

for(i = 0; i < DeformObj->targetModel_bicep_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_bicep_max_male.pObject[i]->g_smothness == 1)
```

```
                DeformObj->targetModel_bicep_max_male.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_bicep_max_male.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_thigh_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_thigh_max_male.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_thigh_max_male.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_thigh_max_male.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_head_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_head_max_male.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_head_max_male.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_head_max_male.pObject[i]->g_smothness = 1;
}

for(i = 0; i < DeformObj->mainModel_female.numOfObjects; i++)
{
        if(DeformObj->mainModel_female.pObject[i]->g_smothness == 1)
        //if(((Fl_Menu_Item *)o)->value() = 1)
                DeformObj->mainModel_female.pObject[i]->g_smothness = 2;
        else
                DeformObj->mainModel_female.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->sourceModel_female.numOfObjects; i++)
{
        if(DeformObj->sourceModel_female.pObject[i]->g_smothness == 1)
                DeformObj->sourceModel_female.pObject[i]->g_smothness = 2;
        else
                DeformObj->sourceModel_female.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_waist_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_waist_max_female.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_waist_max_female.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_waist_max_female.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_chest_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_chest_max_female.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_chest_max_female.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_chest_max_female.pObject[i]->g_smothness = 1;
}

for(i = 0; i < DeformObj->targetModel_forarm_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_forarm_max_female.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_forarm_max_female.pObject[i]->g_smothness = 2;
        else
```

```
                DeformObj->targetModel_forarm_max_female.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_calf_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_calf_max_female.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_calf_max_female.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_calf_max_female.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_hip_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_hip_max_female.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_hip_max_female.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_hip_max_female.pObject[i]->g_smothness = 1;
}

for(i = 0; i < DeformObj->targetModel_bicep_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_bicep_max_female.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_bicep_max_female.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_bicep_max_female.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_thigh_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_thigh_max_female.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_thigh_max_female.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_thigh_max_female.pObject[i]->g_smothness = 1;
}
for(i = 0; i < DeformObj->targetModel_head_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_head_max_female.pObject[i]->g_smothness == 1)
                DeformObj->targetModel_head_max_female.pObject[i]->g_smothness = 2;
        else
                DeformObj->targetModel_head_max_female.pObject[i]->g_smothness = 1;
}

DeformObj->redraw();
}
void DeformUI::cb_Menu_Smooth(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_Smooth_i(o,v);
}

inline void DeformUI::cb_Menu_Wireframe_i(Fl_Menu_*, void*) {
  int i;

for(i = 0; i < DeformObj->mainModel_generic.numOfObjects; i++)
{
        if(DeformObj->mainModel_generic.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->mainModel_generic.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->mainModel_generic.pObject[i]->g_ViewMode = GL_FILL;
```

```
}

for(i = 0; i < DeformObj->mainModel_male.numOfObjects; i++)
{
        if(DeformObj->mainModel_male.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->mainModel_male.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->mainModel_male.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->sourceModel_male.numOfObjects; i++)
{
        if(DeformObj->sourceModel_male.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->sourceModel_male.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->sourceModel_male.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_waist_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_waist_max_male.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_waist_max_male.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_waist_max_male.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_chest_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_chest_max_male.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_chest_max_male.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_chest_max_male.pObject[i]->g_ViewMode = GL_FILL;
}

for(i = 0; i < DeformObj->targetModel_forarm_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_forarm_max_male.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_forarm_max_male.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_forarm_max_male.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_calf_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_calf_max_male.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_calf_max_male.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_calf_max_male.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_hip_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_hip_max_male.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_hip_max_male.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_hip_max_male.pObject[i]->g_ViewMode = GL_FILL;
}

for(i = 0; i < DeformObj->targetModel_bicep_max_male.numOfObjects; i++)
```

```
{
        if(DeformObj->targetModel_bicep_max_male.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_bicep_max_male.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_bicep_max_male.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_thigh_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_thigh_max_male.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_thigh_max_male.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_thigh_max_male.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_head_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_head_max_male.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_head_max_male.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_head_max_male.pObject[i]->g_ViewMode = GL_FILL;
}

///////////
for(i = 0; i < DeformObj->mainModel_female.numOfObjects; i++)
{
        if(DeformObj->mainModel_female.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->mainModel_female.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->mainModel_female.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->sourceModel_female.numOfObjects; i++)
{
        if(DeformObj->sourceModel_female.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->sourceModel_female.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->sourceModel_female.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_waist_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_waist_max_female.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_waist_max_female.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_waist_max_female.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_chest_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_chest_max_female.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_chest_max_female.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_chest_max_female.pObject[i]->g_ViewMode = GL_FILL;
}

for(i = 0; i < DeformObj->targetModel_forarm_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_forarm_max_female.pObject[i]->g_ViewMode == GL_FILL)
```

```
                DeformObj->targetModel_forarm_max_female.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_forarm_max_female.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_calf_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_calf_max_female.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_calf_max_female.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_calf_max_female.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_hip_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_hip_max_female.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_hip_max_female.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_hip_max_female.pObject[i]->g_ViewMode = GL_FILL;
}

for(i = 0; i < DeformObj->targetModel_bicep_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_bicep_max_female.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_bicep_max_female.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_bicep_max_female.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_thigh_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_thigh_max_female.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_thigh_max_female.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_thigh_max_female.pObject[i]->g_ViewMode = GL_FILL;
}
for(i = 0; i < DeformObj->targetModel_head_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_head_max_female.pObject[i]->g_ViewMode == GL_FILL)
                DeformObj->targetModel_head_max_female.pObject[i]->g_ViewMode = GL_LINE;
        else
                DeformObj->targetModel_head_max_female.pObject[i]->g_ViewMode = GL_FILL;
}

DeformObj->redraw();
}
void DeformUI::cb_Menu_Wireframe(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_Wireframe_i(o,v);
}

inline void DeformUI::cb_Menu_Normal_i(Fl_Menu_*, void*) {
  int i;

for(i = 0; i < DeformObj->mainModel_generic.numOfObjects; i++)
{
        if(DeformObj->mainModel_generic.pObject[i]->showNormal == 0)
                DeformObj->mainModel_generic.pObject[i]->showNormal = 1;
        else
                DeformObj->mainModel_generic.pObject[i]->showNormal = 0;
}

for(i = 0; i < DeformObj->mainModel_male.numOfObjects; i++)
{
        if(DeformObj->mainModel_male.pObject[i]->showNormal == 0)
                DeformObj->mainModel_male.pObject[i]->showNormal = 1;
        else
                DeformObj->mainModel_male.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->sourceModel_male.numOfObjects; i++)
{
        if(DeformObj->sourceModel_male.pObject[i]->showNormal == 0)
                DeformObj->sourceModel_male.pObject[i]->showNormal = 1;
        else
                DeformObj->sourceModel_male.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_waist_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_waist_max_male.pObject[i]->showNormal == 0)
                DeformObj->targetModel_waist_max_male.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_waist_max_male.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_chest_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_chest_max_male.pObject[i]->showNormal == 0)
                DeformObj->targetModel_chest_max_male.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_chest_max_male.pObject[i]->showNormal = 0;
}

for(i = 0; i < DeformObj->targetModel_forarm_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_forarm_max_male.pObject[i]->showNormal == 0)
                DeformObj->targetModel_forarm_max_male.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_forarm_max_male.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_calf_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_calf_max_male.pObject[i]->showNormal == 0)
                DeformObj->targetModel_calf_max_male.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_calf_max_male.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_hip_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_hip_max_male.pObject[i]->showNormal == 0)
                DeformObj->targetModel_hip_max_male.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_hip_max_male.pObject[i]->showNormal = 0;
}
```

```
for(i = 0; i < DeformObj->targetModel_bicep_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_bicep_max_male.pObject[i]->showNormal == 0)
                DeformObj->targetModel_bicep_max_male.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_bicep_max_male.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_thigh_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_thigh_max_male.pObject[i]->showNormal == 0)
                DeformObj->targetModel_thigh_max_male.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_thigh_max_male.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_head_max_male.numOfObjects; i++)
{
        if(DeformObj->targetModel_head_max_male.pObject[i]->showNormal == 0)
                DeformObj->targetModel_head_max_male.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_head_max_male.pObject[i]->showNormal = 0;
}

///////
for(i = 0; i < DeformObj->mainModel_female.numOfObjects; i++)
{
        if(DeformObj->mainModel_female.pObject[i]->showNormal == 0)
                DeformObj->mainModel_female.pObject[i]->showNormal = 1;
        else
                DeformObj->mainModel_female.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->sourceModel_female.numOfObjects; i++)
{
        if(DeformObj->sourceModel_female.pObject[i]->showNormal == 0)
                DeformObj->sourceModel_female.pObject[i]->showNormal = 1;
        else
                DeformObj->sourceModel_female.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_waist_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_waist_max_female.pObject[i]->showNormal == 0)
                DeformObj->targetModel_waist_max_female.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_waist_max_female.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_chest_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_chest_max_female.pObject[i]->showNormal == 0)
                DeformObj->targetModel_chest_max_female.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_chest_max_female.pObject[i]->showNormal = 0;
}

for(i = 0; i < DeformObj->targetModel_forarm_max_female.numOfObjects; i++)
```

```
{
        if(DeformObj->targetModel_forarm_max_female.pObject[i]->showNormal == 0)
                DeformObj->targetModel_forarm_max_female.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_forarm_max_female.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_calf_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_calf_max_female.pObject[i]->showNormal == 0)
                DeformObj->targetModel_calf_max_female.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_calf_max_female.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_hip_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_hip_max_female.pObject[i]->showNormal == 0)
                DeformObj->targetModel_hip_max_female.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_hip_max_female.pObject[i]->showNormal = 0;
}

for(i = 0; i < DeformObj->targetModel_bicep_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_bicep_max_female.pObject[i]->showNormal == 0)
                DeformObj->targetModel_bicep_max_female.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_bicep_max_female.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_thigh_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_thigh_max_female.pObject[i]->showNormal == 0)
                DeformObj->targetModel_thigh_max_female.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_thigh_max_female.pObject[i]->showNormal = 0;
}
for(i = 0; i < DeformObj->targetModel_head_max_female.numOfObjects; i++)
{
        if(DeformObj->targetModel_head_max_female.pObject[i]->showNormal == 0)
                DeformObj->targetModel_head_max_female.pObject[i]->showNormal = 1;
        else
                DeformObj->targetModel_head_max_female.pObject[i]->showNormal = 0;
}


DeformObj->redraw();
}
void DeformUI::cb_Menu_Normal(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_Normal_i(o,v);
}

inline void DeformUI::cb_Menu_Reset_Model_i(Fl_Menu_*, void*) {
}
void DeformUI::cb_Menu_Reset_Model(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_Reset_Model_i(o,v);
```

```
}

inline void DeformUI::cb_Menu_Linear_i(Fl_Menu_*, void*) {
  DeformObj->interp_type = 0;
}
void DeformUI::cb_Menu_Linear(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_Linear_i(o,v);
}

inline void DeformUI::cb_Menu_Cosine_i(Fl_Menu_*, void*) {
  DeformObj->interp_type = 1;
}
void DeformUI::cb_Menu_Cosine(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_Cosine_i(o,v);
}

inline void DeformUI::cb_Menu_Cubic_i(Fl_Menu_*, void*) {
  DeformObj->interp_type = 2;
}
void DeformUI::cb_Menu_Cubic(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_Cubic_i(o,v);
}

inline void DeformUI::cb_Menu_Normal_Control_Male_i(Fl_Menu_*, void*) {
  Slider_Shape_Blend_Waist->show();
Slider_Shape_Blend_Chest->show();
Slider_Shape_Blend_Forarm->show();
Slider_Shape_Blend_Bicep->show();
Slider_Shape_Blend_Thigh->show();
Slider_Shape_Blend_Head->show();
Slider_Shape_Blend_Calf->show();
Slider_Shape_Blend_Hip->show();
Slider_Body_Weight->hide();
Slider_Gender->hide();

Menu_SourceModel->activate();
Menu_TargetModel_Waist->activate();
Menu_TargetModel_Chest->activate();
Menu_TargetModel_Forarm->activate();
Menu_TargetModel_Bicep->activate();
Menu_TargetModel_Thigh->activate();
Menu_TargetModel_Head->activate();
Menu_TargetModel_Calf->activate();
Menu_TargetModel_Hip->activate();

DeformObj->control_type = 0;
//DeformObj->model_gender = 0;
DeformObj->model_id = 0;

//Set the range for the sliders (male)
Slider_Shape_Blend_Waist->minimum(235);
Slider_Shape_Blend_Waist->maximum(422);
Slider_Shape_Blend_Chest->minimum(257);
Slider_Shape_Blend_Chest->maximum(422);
```

```
Slider_Shape_Blend_Forarm->minimum(233);
Slider_Shape_Blend_Forarm->maximum(372);
Slider_Shape_Blend_Bicep->minimum(259);
Slider_Shape_Blend_Bicep->maximum(437);
Slider_Shape_Blend_Thigh->minimum(458);
Slider_Shape_Blend_Thigh->maximum(787);
Slider_Shape_Blend_Head->minimum(128);
Slider_Shape_Blend_Head->maximum(173);
Slider_Shape_Blend_Calf->minimum(300);
Slider_Shape_Blend_Calf->maximum(470);
Slider_Shape_Blend_Hip->minimum(282);
Slider_Shape_Blend_Hip->maximum(428);
//
Slider_Shape_Blend_Waist->value(DeformObj->waist_avg_male);
Slider_Shape_Blend_Chest->value(DeformObj->chest_avg_male);
Slider_Shape_Blend_Forarm->value(DeformObj->forarm_avg_male);
Slider_Shape_Blend_Bicep->value(DeformObj->bicep_avg_male);
Slider_Shape_Blend_Thigh->value(DeformObj->thigh_avg_male);
Slider_Shape_Blend_Head->value(DeformObj->head_avg_male);
Slider_Shape_Blend_Calf->value(DeformObj->calf_avg_male);
Slider_Shape_Blend_Hip->value(DeformObj->hip_avg_male);

DeformObj->weight_waist_max_male = 0.0;
DeformObj->weight_chest_max_male = 0.0;
DeformObj->weight_forarm_max_male = 0.0;
DeformObj->weight_bicep_max_male = 0.0;
DeformObj->weight_thigh_max_male = 0.0;
DeformObj->weight_head_max_male = 0.0;
DeformObj->weight_calf_max_male = 0.0;
DeformObj->weight_hip_max_male = 0.0;

DeformObj->weight_waist_min_male = 0.0;
DeformObj->weight_chest_min_male = 0.0;
DeformObj->weight_forarm_min_male = 0.0;
DeformObj->weight_bicep_min_male = 0.0;
DeformObj->weight_thigh_min_male = 0.0;
DeformObj->weight_head_min_male = 0.0;
DeformObj->weight_calf_min_male = 0.0;
DeformObj->weight_hip_min_male = 0.0;

DeformObj->blend_shape();
DeformObj->redraw();


mainWindow->redraw();
}
void DeformUI::cb_Menu_Normal_Control_Male(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))-
>cb_Menu_Normal_Control_Male_i(o,v);
}

inline void DeformUI::cb_Menu_Normal_Control_Female_i(Fl_Menu_*, void*) {
  Slider_Shape_Blend_Waist->show();
Slider_Shape_Blend_Chest->show();
```

```
Slider_Shape_Blend_Forarm->show();
Slider_Shape_Blend_Bicep->show();
Slider_Shape_Blend_Thigh->show();
Slider_Shape_Blend_Head->show();
Slider_Shape_Blend_Calf->show();
Slider_Shape_Blend_Hip->show();
Slider_Body_Weight->hide();
Slider_Gender->hide();

Menu_SourceModel->activate();
Menu_TargetModel_Waist->activate();
Menu_TargetModel_Chest->activate();
Menu_TargetModel_Forarm->activate();
Menu_TargetModel_Bicep->activate();
Menu_TargetModel_Thigh->activate();
Menu_TargetModel_Head->activate();
Menu_TargetModel_Calf->activate();
Menu_TargetModel_Hip->activate();

DeformObj->control_type = 1;
//DeformObj->model_gender = 1;
DeformObj->model_id = 0;

//Set the range for the sliders (female)
Slider_Shape_Blend_Waist->minimum(225);
Slider_Shape_Blend_Waist->maximum(390);
Slider_Shape_Blend_Chest->minimum(222);
Slider_Shape_Blend_Chest->maximum(375);
Slider_Shape_Blend_Forarm->minimum(210);
Slider_Shape_Blend_Forarm->maximum(325);
Slider_Shape_Blend_Bicep->minimum(213);
Slider_Shape_Blend_Bicep->maximum(371);
Slider_Shape_Blend_Thigh->minimum(454);
Slider_Shape_Blend_Thigh->maximum(748);
Slider_Shape_Blend_Head->minimum(126);
Slider_Shape_Blend_Head->maximum(167);
Slider_Shape_Blend_Calf->minimum(285);
Slider_Shape_Blend_Calf->maximum(459);
Slider_Shape_Blend_Hip->minimum(270);
Slider_Shape_Blend_Hip->maximum(420);


Slider_Shape_Blend_Waist->value(DeformObj->waist_avg_female);
Slider_Shape_Blend_Chest->value(DeformObj->chest_avg_female);
Slider_Shape_Blend_Forarm->value(DeformObj->forarm_avg_female);
Slider_Shape_Blend_Bicep->value(DeformObj->bicep_avg_female);
Slider_Shape_Blend_Thigh->value(DeformObj->thigh_avg_female);
Slider_Shape_Blend_Head->value(DeformObj->head_avg_female);
Slider_Shape_Blend_Calf->value(DeformObj->calf_avg_female);
Slider_Shape_Blend_Hip->value(DeformObj->hip_avg_female);

DeformObj->weight_waist_max_female = 0.0;
DeformObj->weight_chest_max_female = 0.0;
DeformObj->weight_forarm_max_female = 0.0;
```

```
DeformObj->weight_bicep_max_female = 0.0;
DeformObj->weight_thigh_max_female = 0.0;
DeformObj->weight_head_max_female = 0.0;
DeformObj->weight_calf_max_female = 0.0;
DeformObj->weight_hip_max_female = 0.0;

DeformObj->weight_waist_min_female = 0.0;
DeformObj->weight_chest_min_female = 0.0;
DeformObj->weight_forarm_min_female = 0.0;
DeformObj->weight_bicep_min_female = 0.0;
DeformObj->weight_thigh_min_female = 0.0;
DeformObj->weight_head_min_female = 0.0;
DeformObj->weight_calf_min_female = 0.0;
DeformObj->weight_hip_min_female = 0.0;

DeformObj->blend_shape();
DeformObj->redraw();

mainWindow->redraw();
}
void DeformUI::cb_Menu_Normal_Control_Female(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))-
>cb_Menu_Normal_Control_Female_i(o,v);
}

inline void DeformUI::cb_Menu_Weight_Gender_Control_i(Fl_Menu_*, void*) {
  Slider_Body_Weight->show();
Slider_Gender->show();
Slider_Shape_Blend_Waist->hide();
Slider_Shape_Blend_Chest->hide();
Slider_Shape_Blend_Forarm->hide();
Slider_Shape_Blend_Bicep->hide();
Slider_Shape_Blend_Thigh->hide();
Slider_Shape_Blend_Head->hide();
Slider_Shape_Blend_Calf->hide();
Slider_Shape_Blend_Hip->hide();

Menu_SourceModel->deactivate();
Menu_TargetModel_Waist->deactivate();
Menu_TargetModel_Chest->deactivate();
Menu_TargetModel_Forarm->deactivate();
Menu_TargetModel_Bicep->deactivate();
Menu_TargetModel_Thigh->deactivate();
Menu_TargetModel_Head->deactivate();
Menu_TargetModel_Calf->deactivate();
Menu_TargetModel_Hip->deactivate();

DeformObj->control_type = 2;
//DeformObj->model_gender = 2;
DeformObj->model_id = 0;

Slider_Body_Weight->value(DeformObj->value_body_weight);
Slider_Gender->value(DeformObj->value_gender);
Slider_Body_Weight->redraw();
```

```
Slider_Gender->redraw();
mainWindow->redraw();
}
void DeformUI::cb_Menu_Weight_Gender_Control(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))-
>cb_Menu_Weight_Gender_Control_i(o,v);
}

inline void DeformUI::cb_Menu_About_i(Fl_Menu_*, void*) {
  aboutWindow->show();
}
void DeformUI::cb_Menu_About(Fl_Menu_* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Menu_About_i(o,v);
}

Fl_Menu_Item DeformUI::menu_MainMenuBar[] = {
 {"&File", 0,  0, 0, 64, 3, 0, 14, 56},
 {"Load Models", 0,  (Fl_Callback*)DeformUI::cb_Menu_Load_Models, 0, 128, 0, 0, 14, 56},
 {"Load Male Models", 0,  (Fl_Callback*)DeformUI::cb_Menu_Load_Male, 0, 16, 0, 0, 14, 56},
 {"Load Female Models", 0,  (Fl_Callback*)DeformUI::cb_Menu_Load_Female, 0, 144, 0, 0, 14, 56},
 {"&Load Obj File", 0,  (Fl_Callback*)DeformUI::cb_Menu_LoadObj, 0, 0, 0, 0, 14, 56},
 {"&Save Obj File", 0,  (Fl_Callback*)DeformUI::cb_Menu_SaveObj, 0, 0, 0, 0, 14, 56},
 {0},
 {"&Display", 0,  0, 0, 64, 3, 0, 14, 56},
 {"&Show Models", 0,  0, 0, 64, 0, 0, 14, 56},
 {"MainModel", 0,  (Fl_Callback*)DeformUI::cb_Menu_MainModel, 0, 12, 0, 0, 14, 56},
 {"SourceModel", 0,  (Fl_Callback*)DeformUI::cb_Menu_SourceModel, 0, 8, 0, 0, 14, 56},
 {"TargetModel_Waist", 0,  (Fl_Callback*)DeformUI::cb_Menu_TargetModel_Waist, 0, 8, 0, 0, 14,
56},
 {"TargetModel_Chest", 0,  (Fl_Callback*)DeformUI::cb_Menu_TargetModel_Chest, 0, 8, 0, 0, 14,
56},
 {"TargetModel_Forarm", 0,  (Fl_Callback*)DeformUI::cb_Menu_TargetModel_Forarm, 0, 8, 0, 0, 14,
56},
 {"TargetModel_Bicep", 0,  (Fl_Callback*)DeformUI::cb_Menu_TargetModel_Bicep, 0, 8, 0, 0, 14,
56},
 {"TargetModel_Thigh", 0,  (Fl_Callback*)DeformUI::cb_Menu_TargetModel_Thigh, 0, 8, 0, 0, 14,
56},
 {"TargetModel_Head", 0,  (Fl_Callback*)DeformUI::cb_Menu_TargetModel_Head, 0, 8, 0, 0, 14, 56},
 {"TargetModel_Calf", 0,  (Fl_Callback*)DeformUI::cb_Menu_TargetModel_Calf, 0, 8, 0, 0, 14, 56},
 {"TargetModel_Hip", 0,  (Fl_Callback*)DeformUI::cb_Menu_TargetModel_Hip, 0, 8, 0, 0, 14, 56},
 {0},
 {"&Smooth Shading", 0,  (Fl_Callback*)DeformUI::cb_Menu_Smooth, 0, 2, 0, 0, 14, 56},
 {"&Wire Frame", 0,  (Fl_Callback*)DeformUI::cb_Menu_Wireframe, 0, 2, 0, 0, 14, 56},
 {"&Show Normal", 0,  (Fl_Callback*)DeformUI::cb_Menu_Normal, 0, 2, 0, 0, 14, 56},
 {0},
 {"&Tools", 0,  0, 0, 64, 3, 0, 14, 56},
 {"&Reset Model", 0,  (Fl_Callback*)DeformUI::cb_Menu_Reset_Model, 0, 0, 0, 0, 14, 56},
 {"&Interpolation", 0,  0, 0, 80, 0, 0, 14, 56},
 {"&Linear", 0,  (Fl_Callback*)DeformUI::cb_Menu_Linear, 0, 12, 0, 0, 14, 56},
 {"&Consine", 0,  (Fl_Callback*)DeformUI::cb_Menu_Cosine, 0, 8, 0, 0, 14, 56},
 {"&Cubic", 0,  (Fl_Callback*)DeformUI::cb_Menu_Cubic, 0, 8, 0, 0, 14, 56},
 {0},
 {0},
 {"&Controls", 0,  0, 0, 64, 3, 0, 14, 56},
 {"&Normal Control Male", 0,  (Fl_Callback*)DeformUI::cb_Menu_Normal_Control_Male, 0, 12, 0, 0,
14, 56},
 {"&Normal Control Female", 0,  (Fl_Callback*)DeformUI::cb_Menu_Normal_Control_Female, 0, 8, 0,
0, 14, 56},
 {"&Weight and Gender Control", 0,  (Fl_Callback*)DeformUI::cb_Menu_Weight_Gender_Control, 0, 8,
0, 0, 14, 56},
 {0},
 {"&Others", 0,  0, 0, 64, 3, 0, 14, 56},
 {"&Content and Index", 0,  0, 0, 16, 0, 0, 14, 56},
 {"&Send Feedback", 0,  0, 0, 16, 0, 0, 14, 56},
 {"&About...", 0,  (Fl_Callback*)DeformUI::cb_Menu_About, 0, 0, 0, 0, 14, 56},
 {0},
 {0}
};
Fl_Menu_Item* DeformUI::Menu_File = DeformUI::menu_MainMenuBar + 0;
Fl_Menu_Item* DeformUI::Menu_Load_Models = DeformUI::menu_MainMenuBar + 1;
Fl_Menu_Item* DeformUI::Menu_Load_Male = DeformUI::menu_MainMenuBar + 2;
Fl_Menu_Item* DeformUI::Menu_Load_Female = DeformUI::menu_MainMenuBar + 3;
Fl_Menu_Item* DeformUI::Menu_LoadObj = DeformUI::menu_MainMenuBar + 4;
Fl_Menu_Item* DeformUI::Menu_SaveObj = DeformUI::menu_MainMenuBar + 5;
Fl_Menu_Item* DeformUI::Menu_Display = DeformUI::menu_MainMenuBar + 7;
Fl_Menu_Item* DeformUI::Menu_Models = DeformUI::menu_MainMenuBar + 8;
Fl_Menu_Item* DeformUI::Menu_MainModel = DeformUI::menu_MainMenuBar + 9;
Fl_Menu_Item* DeformUI::Menu_SourceModel = DeformUI::menu_MainMenuBar + 10;
Fl_Menu_Item* DeformUI::Menu_TargetModel_Waist = DeformUI::menu_MainMenuBar + 11;
Fl_Menu_Item* DeformUI::Menu_TargetModel_Chest = DeformUI::menu_MainMenuBar + 12;
Fl_Menu_Item* DeformUI::Menu_TargetModel_Forarm = DeformUI::menu_MainMenuBar + 13;
Fl_Menu_Item* DeformUI::Menu_TargetModel_Bicep = DeformUI::menu_MainMenuBar + 14;
Fl_Menu_Item* DeformUI::Menu_TargetModel_Thigh = DeformUI::menu_MainMenuBar + 15;
Fl_Menu_Item* DeformUI::Menu_TargetModel_Head = DeformUI::menu_MainMenuBar + 16;
Fl_Menu_Item* DeformUI::Menu_TargetModel_Calf = DeformUI::menu_MainMenuBar + 17;
Fl_Menu_Item* DeformUI::Menu_TargetModel_Hip = DeformUI::menu_MainMenuBar + 18;
Fl_Menu_Item* DeformUI::Menu_Smooth = DeformUI::menu_MainMenuBar + 20;
Fl_Menu_Item* DeformUI::Menu_Wireframe = DeformUI::menu_MainMenuBar + 21;
Fl_Menu_Item* DeformUI::Menu_Normal = DeformUI::menu_MainMenuBar + 22;
Fl_Menu_Item* DeformUI::Menu_Tools = DeformUI::menu_MainMenuBar + 24;
Fl_Menu_Item* DeformUI::Menu_Reset_Model = DeformUI::menu_MainMenuBar + 25;
Fl_Menu_Item* DeformUI::Menu_Interpolation = DeformUI::menu_MainMenuBar + 26;
Fl_Menu_Item* DeformUI::Menu_Linear = DeformUI::menu_MainMenuBar + 27;
Fl_Menu_Item* DeformUI::Menu_Cosine = DeformUI::menu_MainMenuBar + 28;
Fl_Menu_Item* DeformUI::Menu_Cubic = DeformUI::menu_MainMenuBar + 29;
Fl_Menu_Item* DeformUI::Menu_Controls = DeformUI::menu_MainMenuBar + 32;
Fl_Menu_Item* DeformUI::Menu_Normal_Control_Male = DeformUI::menu_MainMenuBar + 33;
Fl_Menu_Item* DeformUI::Menu_Normal_Control_Female = DeformUI::menu_MainMenuBar + 34;
Fl_Menu_Item* DeformUI::Menu_Weight_Gender_Control = DeformUI::menu_MainMenuBar + 35;
Fl_Menu_Item* DeformUI::Menu_Help = DeformUI::menu_MainMenuBar + 37;
Fl_Menu_Item* DeformUI::Menu_Contents = DeformUI::menu_MainMenuBar + 38;
Fl_Menu_Item* DeformUI::Menu_Feedback = DeformUI::menu_MainMenuBar + 39;
Fl_Menu_Item* DeformUI::Menu_About = DeformUI::menu_MainMenuBar + 40;

inline void DeformUI::cb_Slider_Shape_Blend_Waist_i(Fl_Value_Slider* o, void*) {
  if(DeformObj->control_type == 0)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->waist_avg_male) >= 0.0)
        {
                DeformObj->weight_waist_max_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>waist_avg_male;
                DeformObj->weight_waist_max_male = DeformObj->weight_waist_max_male / (DeformObj-
>waist_max_male - DeformObj->waist_avg_male);
                DeformObj->weight_waist_min_male = 0.0;
        }
        else
        {
                DeformObj->weight_waist_min_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>waist_avg_male;
                DeformObj->weight_waist_min_male = DeformObj->weight_waist_min_male / (DeformObj-
>waist_min_male - DeformObj->waist_avg_male);
                DeformObj->weight_waist_max_male = 0.0;
        }
}
else if(DeformObj->control_type == 1)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->waist_avg_female) >= 0.0)
        {
                DeformObj->weight_waist_max_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>waist_avg_female;
                DeformObj->weight_waist_max_female = DeformObj->weight_waist_max_female /
(DeformObj->waist_max_female - DeformObj->waist_avg_female);
                DeformObj->weight_waist_min_female = 0.0;
        }
        else
        {
                DeformObj->weight_waist_min_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>waist_avg_female;
                DeformObj->weight_waist_min_female = DeformObj->weight_waist_min_female /
(DeformObj->waist_min_female - DeformObj->waist_avg_female);
                DeformObj->weight_waist_max_female = 0.0;
        }
}

}
void DeformUI::cb_Slider_Shape_Blend_Waist(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))-
>cb_Slider_Shape_Blend_Waist_i(o,v);
}

inline void DeformUI::cb_Slider_Shape_Blend_Chest_i(Fl_Value_Slider* o, void*) {
  if(DeformObj->control_type == 0)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->chest_avg_male) >= 0.0)
        {
                DeformObj->weight_chest_max_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>chest_avg_male;
                DeformObj->weight_chest_max_male = DeformObj->weight_chest_max_male / (DeformObj-
>chest_max_male - DeformObj->chest_avg_male);
                DeformObj->weight_chest_min_male = 0.0;
        }
        else
        {
                DeformObj->weight_chest_min_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>chest_avg_male;
                DeformObj->weight_chest_min_male = DeformObj->weight_chest_min_male / (DeformObj-
>chest_min_male - DeformObj->chest_avg_male);
                DeformObj->weight_chest_max_male = 0.0;
        }
}
else if(DeformObj->control_type == 1)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->chest_avg_female) >= 0.0)
        {
                DeformObj->weight_chest_max_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>chest_avg_female;
                DeformObj->weight_chest_max_female = DeformObj->weight_chest_max_female /
(DeformObj->chest_max_female - DeformObj->chest_avg_female);
                DeformObj->weight_chest_min_female = 0.0;
        }
        else
        {
                DeformObj->weight_chest_min_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>chest_avg_female;
                DeformObj->weight_chest_min_female = DeformObj->weight_chest_min_female /
(DeformObj->chest_min_female - DeformObj->chest_avg_female);
                DeformObj->weight_chest_max_female = 0.0;
        }
}
}
void DeformUI::cb_Slider_Shape_Blend_Chest(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))-
>cb_Slider_Shape_Blend_Chest_i(o,v);
}

inline void DeformUI::cb_Slider_Shape_Blend_Forarm_i(Fl_Value_Slider* o, void*) {
  if(DeformObj->control_type == 0)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->forarm_avg_male) >= 0.0)
        {
                DeformObj->weight_forarm_max_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>forarm_avg_male;
                DeformObj->weight_forarm_max_male = DeformObj->weight_forarm_max_male /
(DeformObj->forarm_max_male - DeformObj->forarm_avg_male);
                DeformObj->weight_forarm_min_male = 0.0;
        }
        else
        {
                DeformObj->weight_forarm_min_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>forarm_avg_male;
                DeformObj->weight_forarm_min_male = DeformObj->weight_forarm_min_male /
(DeformObj->forarm_min_male - DeformObj->forarm_avg_male);
                DeformObj->weight_forarm_max_male = 0.0;
        }
}
else if(DeformObj->control_type == 1)
```

```
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->forarm_avg_female) >= 0.0)
        {
                DeformObj->weight_forarm_max_female = ((Fl_Value_Slider *)o)->value() -
DeformObj->forarm_avg_female;
                DeformObj->weight_forarm_max_female = DeformObj->weight_forarm_max_female /
(DeformObj->forarm_max_female - DeformObj->forarm_avg_female);
                DeformObj->weight_forarm_min_female = 0.0;
        }
        else
        {
                DeformObj->weight_forarm_min_female = ((Fl_Value_Slider *)o)->value() -
DeformObj->forarm_avg_female;
                DeformObj->weight_forarm_min_female = DeformObj->weight_forarm_min_female /
(DeformObj->forarm_min_female - DeformObj->forarm_avg_female);
                DeformObj->weight_forarm_max_female = 0.0;
        }
}
}
void DeformUI::cb_Slider_Shape_Blend_Forarm(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))-
>cb_Slider_Shape_Blend_Forarm_i(o,v);
}

inline void DeformUI::cb_Slider_Shape_Blend_Bicep_i(Fl_Value_Slider* o, void*) {
  if(DeformObj->control_type == 0)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->bicep_avg_male) >= 0.0)
        {
                DeformObj->weight_bicep_max_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>bicep_avg_male;
                DeformObj->weight_bicep_max_male = DeformObj->weight_bicep_max_male / (DeformObj-
>bicep_max_male - DeformObj->bicep_avg_male);
                DeformObj->weight_bicep_min_male = 0.0;
        }
        else
        {
                DeformObj->weight_bicep_min_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>bicep_avg_male;
                DeformObj->weight_bicep_min_male = DeformObj->weight_bicep_min_male / (DeformObj-
>bicep_min_male - DeformObj->bicep_avg_male);
                DeformObj->weight_bicep_max_male = 0.0;
        }
}
else if(DeformObj->control_type == 1)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->bicep_avg_female) >= 0.0)
        {
                DeformObj->weight_bicep_max_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>bicep_avg_female;
                DeformObj->weight_bicep_max_female = DeformObj->weight_bicep_max_female /
(DeformObj->bicep_max_female - DeformObj->bicep_avg_female);
                DeformObj->weight_bicep_min_female = 0.0;
        }
        else
        {
                DeformObj->weight_bicep_min_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>bicep_avg_female;
                DeformObj->weight_bicep_min_female = DeformObj->weight_bicep_min_female /
(DeformObj->bicep_min_female - DeformObj->bicep_avg_female);
                DeformObj->weight_bicep_max_female = 0.0;
        }
}
}
void DeformUI::cb_Slider_Shape_Blend_Bicep(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))-
>cb_Slider_Shape_Blend_Bicep_i(o,v);
}

inline void DeformUI::cb_Slider_Shape_Blend_Thigh_i(Fl_Value_Slider* o, void*) {
  if(DeformObj->control_type == 0)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->thigh_avg_male) >= 0.0)
        {
                DeformObj->weight_thigh_max_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>thigh_avg_male;
                DeformObj->weight_thigh_max_male = DeformObj->weight_thigh_max_male / (DeformObj-
>thigh_max_male - DeformObj->thigh_avg_male);
                DeformObj->weight_thigh_min_male = 0.0;
        }
        else
        {
                DeformObj->weight_thigh_min_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>thigh_avg_male;
                DeformObj->weight_thigh_min_male = DeformObj->weight_thigh_min_male / (DeformObj-
>thigh_min_male - DeformObj->thigh_avg_male);
                DeformObj->weight_thigh_max_male = 0.0;
        }
}
else if(DeformObj->control_type == 1)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->thigh_avg_female) >= 0.0)
        {
                DeformObj->weight_thigh_max_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>thigh_avg_female;
                DeformObj->weight_thigh_max_female = DeformObj->weight_thigh_max_female /
(DeformObj->thigh_max_female - DeformObj->thigh_avg_female);
                DeformObj->weight_thigh_min_female = 0.0;
        }
        else
        {
                DeformObj->weight_thigh_min_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>thigh_avg_female;
                DeformObj->weight_thigh_min_female = DeformObj->weight_thigh_min_female /
(DeformObj->thigh_min_female - DeformObj->thigh_avg_female);
                DeformObj->weight_thigh_max_female = 0.0;
        }
}
```

```
}
void DeformUI::cb_Slider_Shape_Blend_Thigh(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))-
>cb_Slider_Shape_Blend_Thigh_i(o,v);
}

inline void DeformUI::cb_Slider_Shape_Blend_Head_i(Fl_Value_Slider* o, void*) {
  if(DeformObj->control_type == 0)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->head_avg_male) >= 0.0)
        {
                DeformObj->weight_head_max_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>head_avg_male;
                DeformObj->weight_head_max_male = DeformObj->weight_head_max_male / (DeformObj-
>head_max_male - DeformObj->head_avg_male);
                DeformObj->weight_head_min_male = 0.0;
        }
        else
        {
                DeformObj->weight_head_min_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>head_avg_male;
                DeformObj->weight_head_min_male = DeformObj->weight_head_min_male / (DeformObj-
>head_min_male - DeformObj->head_avg_male);
                DeformObj->weight_head_max_male = 0.0;
        }
}
else if(DeformObj->control_type == 1)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->head_avg_female) >= 0.0)
        {
                DeformObj->weight_head_max_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>head_avg_female;
                DeformObj->weight_head_max_female = DeformObj->weight_head_max_female /
(DeformObj->head_max_female - DeformObj->head_avg_female);
                DeformObj->weight_head_min_female = 0.0;
        }
        else
        {
                DeformObj->weight_head_min_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>head_avg_female;
                DeformObj->weight_head_min_female = DeformObj->weight_head_min_female /
(DeformObj->head_min_female - DeformObj->head_avg_female);
                DeformObj->weight_head_max_female = 0.0;
        }
}
}
void DeformUI::cb_Slider_Shape_Blend_Head(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Slider_Shape_Blend_Head_i(o,v);
}

inline void DeformUI::cb_Slider_Shape_Blend_Calf_i(Fl_Value_Slider* o, void*) {
  if(DeformObj->control_type == 0)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->calf_avg_male) >= 0.0)
        {
                DeformObj->weight_calf_max_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>calf_avg_male;
                DeformObj->weight_calf_max_male = DeformObj->weight_calf_max_male / (DeformObj-
>calf_max_male - DeformObj->calf_avg_male);
                DeformObj->weight_calf_min_male = 0.0;
        }
        else
        {
                DeformObj->weight_calf_min_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>calf_avg_male;
                DeformObj->weight_calf_min_male = DeformObj->weight_calf_min_male / (DeformObj-
>calf_min_male - DeformObj->calf_avg_male);
                DeformObj->weight_calf_max_male = 0.0;
        }
}
else if(DeformObj->control_type == 1)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->calf_avg_female) >= 0.0)
        {
                DeformObj->weight_calf_max_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>calf_avg_female;
                DeformObj->weight_calf_max_female = DeformObj->weight_calf_max_female /
(DeformObj->calf_max_female - DeformObj->calf_avg_female);
                DeformObj->weight_calf_min_female = 0.0;
        }
        else
        {
                DeformObj->weight_calf_min_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>calf_avg_female;
                DeformObj->weight_calf_min_female = DeformObj->weight_calf_min_female /
(DeformObj->calf_min_female - DeformObj->calf_avg_female);
                DeformObj->weight_calf_max_female = 0.0;
        }
}
}
void DeformUI::cb_Slider_Shape_Blend_Calf(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Slider_Shape_Blend_Calf_i(o,v);
}

inline void DeformUI::cb_Slider_Shape_Blend_Hip_i(Fl_Value_Slider* o, void*) {
  if(DeformObj->control_type == 0)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->hip_avg_male) >= 0.0)
        {
                DeformObj->weight_hip_max_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>hip_avg_male;
                DeformObj->weight_hip_max_male = DeformObj->weight_hip_max_male / (DeformObj-
>hip_max_male - DeformObj->hip_avg_male);
                DeformObj->weight_hip_min_male = 0.0;
        }
        else
        {
                DeformObj->weight_hip_min_male = ((Fl_Value_Slider *)o)->value() - DeformObj-
>hip_avg_male;
                DeformObj->weight_hip_min_male = DeformObj->weight_hip_min_male / (DeformObj-
>hip_min_male - DeformObj->hip_avg_male);
                DeformObj->weight_hip_max_male = 0.0;
        }
}
else if(DeformObj->control_type == 1)
{
        if( (((Fl_Value_Slider *)o)->value() - DeformObj->hip_avg_female) >= 0.0)
        {
                DeformObj->weight_hip_max_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>hip_avg_female;
                DeformObj->weight_hip_max_female = DeformObj->weight_hip_max_female / (DeformObj-
>hip_max_female - DeformObj->hip_avg_female);
                DeformObj->weight_hip_min_female = 0.0;
        }
        else
        {
                DeformObj->weight_hip_min_female = ((Fl_Value_Slider *)o)->value() - DeformObj-
>hip_avg_female;
                DeformObj->weight_hip_min_female = DeformObj->weight_hip_min_female / (DeformObj-
>hip_min_female - DeformObj->hip_avg_female);
                DeformObj->weight_hip_max_female = 0.0;
        }
}
}
void DeformUI::cb_Slider_Shape_Blend_Hip(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Slider_Shape_Blend_Hip_i(o,v);
}

inline void DeformUI::cb_Slider_Body_Weight_i(Fl_Value_Slider* o, void*) {
  if( ((Fl_Value_Slider *)o)->value() >= 476.0 && ((Fl_Value_Slider *)o)->value() <= 967.0 )
{
        Slider_Gender->activate();
        DeformObj->model_gender = 2;
        DeformObj->value_body_weight = ((Fl_Value_Slider *)o)->value();
}
else if ( ((Fl_Value_Slider *)o)->value() < 476.0 )
{
        Slider_Gender->value(1.0);
        DeformObj->model_gender = 1;
        DeformObj->value_gender = 1.0;
        Slider_Gender->deactivate();

        DeformObj->value_body_weight = ((Fl_Value_Slider *)o)->value();
}
else
{
        Slider_Gender->value(0.0);
        DeformObj->model_gender = 0;
        DeformObj->value_gender = 0.0;
        Slider_Gender->deactivate();
        DeformObj->value_body_weight = ((Fl_Value_Slider *)o)->value();
};
}
void DeformUI::cb_Slider_Body_Weight(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Slider_Body_Weight_i(o,v);
}

inline void DeformUI::cb_Slider_Gender_i(Fl_Value_Slider* o, void*) {
  DeformObj->value_gender = ((Fl_Value_Slider *)o)->value();
}
void DeformUI::cb_Slider_Gender(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Slider_Gender_i(o,v);
}

inline void DeformUI::cb_Slider_Red_i(Fl_Value_Slider* o, void*) {
  DeformObj->red = ((Fl_Value_Slider *)o)->value();
DeformObj->redraw();
}
void DeformUI::cb_Slider_Red(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Slider_Red_i(o,v);
}

inline void DeformUI::cb_Slider_Green_i(Fl_Value_Slider* o, void*) {
  DeformObj->green = ((Fl_Value_Slider *)o)->value();
DeformObj->redraw();
}
void DeformUI::cb_Slider_Green(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Slider_Green_i(o,v);
}

inline void DeformUI::cb_Slider_Blue_i(Fl_Value_Slider* o, void*) {
  DeformObj->blue = ((Fl_Value_Slider *)o)->value();
DeformObj->redraw();
}
void DeformUI::cb_Slider_Blue(Fl_Value_Slider* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Slider_Blue_i(o,v);
}

inline void DeformUI::cb_Button_Generate_i(Fl_Return_Button*, void*) {
  if(DeformObj->model_loaded)
{
        //cout << "control type:" << DeformObj->control_type << endl;
        if(DeformObj->control_type == 0 || DeformObj->control_type == 1)
        {
                DeformObj->blend_shape();
                DeformObj->redraw();
        }
        else if(DeformObj->control_type == 2)
        {
                //cout << " button click " << endl;
                DeformObj->calculate_generic_body_shapes();
                DeformObj->blend_shape();
                DeformObj->redraw();
        }
```

```
};
}
void DeformUI::cb_Button_Generate(Fl_Return_Button* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Button_Generate_i(o,v);
}

inline void DeformUI::cb_Button_Reset_i(Fl_Return_Button*, void*) {
  if(DeformObj->model_loaded)
{
        if(DeformObj->control_type == 0)
        {
                Slider_Shape_Blend_Waist->value(DeformObj->waist_avg_male);
                Slider_Shape_Blend_Chest->value(DeformObj->chest_avg_male);
                Slider_Shape_Blend_Forarm->value(DeformObj->forarm_avg_male);
                Slider_Shape_Blend_Bicep->value(DeformObj->bicep_avg_male);
                Slider_Shape_Blend_Thigh->value(DeformObj->thigh_avg_male);
                Slider_Shape_Blend_Head->value(DeformObj->head_avg_male);
                Slider_Shape_Blend_Calf->value(DeformObj->calf_avg_male);
                Slider_Shape_Blend_Hip->value(DeformObj->hip_avg_male);

                DeformObj->weight_waist_max_male = 0.0;
                DeformObj->weight_chest_max_male = 0.0;
                DeformObj->weight_forarm_max_male = 0.0;
                DeformObj->weight_bicep_max_male = 0.0;
                DeformObj->weight_thigh_max_male = 0.0;
                DeformObj->weight_head_max_male = 0.0;
                DeformObj->weight_calf_max_male = 0.0;
                DeformObj->weight_hip_max_male = 0.0;

                DeformObj->weight_waist_min_male = 0.0;
                DeformObj->weight_chest_min_male = 0.0;
                DeformObj->weight_forarm_min_male = 0.0;
                DeformObj->weight_bicep_min_male = 0.0;
                DeformObj->weight_thigh_min_male = 0.0;
                DeformObj->weight_head_min_male = 0.0;
                DeformObj->weight_calf_min_male = 0.0;
                DeformObj->weight_hip_min_male = 0.0;

                DeformObj->blend_shape();
                DeformObj->redraw();
        }
        else if(DeformObj->control_type == 1)
        {
                Slider_Shape_Blend_Waist->value(DeformObj->waist_avg_female);
                Slider_Shape_Blend_Chest->value(DeformObj->chest_avg_female);
                Slider_Shape_Blend_Forarm->value(DeformObj->forarm_avg_female);
                Slider_Shape_Blend_Bicep->value(DeformObj->bicep_avg_female);
                Slider_Shape_Blend_Thigh->value(DeformObj->thigh_avg_female);
                Slider_Shape_Blend_Head->value(DeformObj->head_avg_female);
                Slider_Shape_Blend_Calf->value(DeformObj->calf_avg_female);
                Slider_Shape_Blend_Hip->value(DeformObj->hip_avg_female);

                DeformObj->weight_waist_max_female = 0.0;
                DeformObj->weight_chest_max_female = 0.0;
                DeformObj->weight_forarm_max_female = 0.0;
                DeformObj->weight_bicep_max_female = 0.0;
                DeformObj->weight_thigh_max_female = 0.0;
                DeformObj->weight_head_max_female = 0.0;
                DeformObj->weight_calf_max_female = 0.0;
                DeformObj->weight_hip_max_female = 0.0;

                DeformObj->weight_waist_min_female = 0.0;
                DeformObj->weight_chest_min_female = 0.0;
                DeformObj->weight_forarm_min_female = 0.0;
                DeformObj->weight_bicep_min_female = 0.0;
                DeformObj->weight_thigh_min_female = 0.0;
                DeformObj->weight_head_min_female = 0.0;
                DeformObj->weight_calf_min_female = 0.0;
                DeformObj->weight_hip_min_female = 0.0;


                DeformObj->blend_shape();
                DeformObj->redraw();
        }
        else if(DeformObj->control_type == 2)
        {
                Slider_Body_Weight->value(413);
                Slider_Gender->value(1.0);
                Slider_Gender->deactivate();
                DeformObj->initialize();
                DeformObj->calculate_generic_body_shapes();
                DeformObj->blend_shape();
                DeformObj->redraw();
        }
}
}
void DeformUI::cb_Button_Reset(Fl_Return_Button* o, void* v) {
  ((DeformUI*)(o->parent()->parent()->parent()->user_data()))->cb_Button_Reset_i(o,v);
}

inline void DeformUI::cb_OK_i(Fl_Return_Button* o, void*) {
  ((Fl_Window*)(o->parent()))->hide();
}
void DeformUI::cb_OK(Fl_Return_Button* o, void* v) {
  ((DeformUI*)(o->parent()->user_data()))->cb_OK_i(o,v);
}

DeformUI::DeformUI() {
  Fl_Double_Window* w;
  { Fl_Double_Window* o = mainWindow = new Fl_Double_Window(1024, 721, "Deform");
    w = o;
    o->box(FL_UP_BOX);
    o->labelsize(12);
    o->user_data((void*)(this));
    { Fl_Group* o = All_UI = new Fl_Group(0, 0, 1065, 702);
      { Fl_Group* o = MainMenu = new Fl_Group(5, 3, 992, 25);
        o->box(FL_ENGRAVED_FRAME);
        { Fl_Menu_Bar* o = MainMenuBar = new Fl_Menu_Bar(5, 3, 992, 25);
```

```
          o->box(FL_NO_BOX);
          o->menu(menu_MainMenuBar);
        }
        o->end();
      }
      { Fl_Group* o = MainControlPanel = new Fl_Group(771, 48, 294, 637, "Deform Control");
        o->box(FL_DOWN_FRAME);
        o->labeltype(FL_ENGRAVED_LABEL);
        o->labelsize(11);
        o->align(FL_ALIGN_TOP_LEFT);
        { Fl_Value_Slider* o = Slider_Shape_Blend_Waist = new Fl_Value_Slider(795, 71, 190, 25,
"Waist Breadth(mm)");
          o->type(5);
          o->selection_color(9);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->minimum(235);
          o->maximum(422);
          o->step(0.1);
          o->value(312);
          o->callback((Fl_Callback*)cb_Slider_Shape_Blend_Waist);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Value_Slider* o = Slider_Shape_Blend_Chest = new Fl_Value_Slider(795, 110, 190, 25,
"Chest Breadth(mm)");
          o->type(5);
          o->selection_color(9);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->minimum(257);
          o->maximum(422);
          o->step(0.1);
          o->value(324);
          o->callback((Fl_Callback*)cb_Slider_Shape_Blend_Chest);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Value_Slider* o = Slider_Shape_Blend_Forarm = new Fl_Value_Slider(794, 151, 190, 25,
"Forarm Circumference(mm)");
          o->type(5);
          o->selection_color(9);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->minimum(233);
          o->maximum(372);
          o->step(0.1);
          o->value(303);
          o->callback((Fl_Callback*)cb_Slider_Shape_Blend_Forarm);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Value_Slider* o = Slider_Shape_Blend_Bicep = new Fl_Value_Slider(794, 192, 190, 25,
"Arm Bicep CircumFerence(mm)");
          o->type(5);
          o->selection_color(9);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->minimum(259);
          o->maximum(437);
          o->step(0.1);
          o->value(337);
          o->callback((Fl_Callback*)cb_Slider_Shape_Blend_Bicep);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Value_Slider* o = Slider_Shape_Blend_Thigh = new Fl_Value_Slider(795, 231, 190, 25,
"Thigh Circumference(mm)");
          o->type(5);
          o->selection_color(9);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->minimum(458);
          o->maximum(787);
          o->step(0.1);
          o->value(596);
          o->callback((Fl_Callback*)cb_Slider_Shape_Blend_Thigh);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Value_Slider* o = Slider_Shape_Blend_Head = new Fl_Value_Slider(794, 271, 190, 25,
"Head Breadth(mm)");
          o->type(5);
          o->selection_color(9);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->minimum(128);
          o->maximum(173);
          o->step(0.1);
          o->value(152);
          o->callback((Fl_Callback*)cb_Slider_Shape_Blend_Head);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Value_Slider* o = Slider_Shape_Blend_Calf = new Fl_Value_Slider(795, 310, 190, 25,
"Calf Circumference(mm)");
          o->type(5);
          o->selection_color(9);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->minimum(300);
          o->maximum(470);
          o->step(0.1);
          o->value(378);
          o->callback((Fl_Callback*)cb_Slider_Shape_Blend_Calf);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Value_Slider* o = Slider_Shape_Blend_Hip = new Fl_Value_Slider(795, 350, 190, 25,
"Hip Breadth(mm)");
          o->type(5);
          o->selection_color(9);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->minimum(282);
          o->maximum(428);
          o->step(0.1);
          o->value(343);
          o->callback((Fl_Callback*)cb_Slider_Shape_Blend_Hip);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Value_Slider* o = Slider_Body_Weight = new Fl_Value_Slider(795, 71, 190, 25, "Body
Weight");
```

```
          o->type(5);
          o->selection_color(9);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->minimum(413);
          o->maximum(1278);
          o->step(0.1);
          o->value(413);
          o->callback((Fl_Callback*)cb_Slider_Body_Weight);
          o->align(FL_ALIGN_TOP_LEFT);
          o->hide();
        }
        { Fl_Value_Slider* o = Slider_Gender = new Fl_Value_Slider(795, 112, 190, 25, "Gender");
          o->type(5);
          o->selection_color(9);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->callback((Fl_Callback*)cb_Slider_Gender);
          o->align(FL_ALIGN_TOP_LEFT);
          o->hide();
        }
        { Fl_Value_Slider* o = Slider_Red = new Fl_Value_Slider(795, 572, 190, 25, "Red");
          o->type(5);
          o->selection_color(1);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->value(1);
          o->callback((Fl_Callback*)cb_Slider_Red);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Value_Slider* o = Slider_Green = new Fl_Value_Slider(796, 611, 190, 25, "Green");
          o->type(5);
          o->selection_color(1);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->value(1);
          o->callback((Fl_Callback*)cb_Slider_Green);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Value_Slider* o = Slider_Blue = new Fl_Value_Slider(796, 652, 190, 25, "Blue");
          o->type(5);
          o->selection_color(1);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->value(1);
          o->callback((Fl_Callback*)cb_Slider_Blue);
          o->align(FL_ALIGN_TOP_LEFT);
        }
        { Fl_Return_Button* o = Button_Generate = new Fl_Return_Button(795, 494, 90, 25,
"Generate");
          o->tooltip("Generate New Body Shape");
          o->box(FL_PLASTIC_UP_BOX);
          o->selection_color(16);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->callback((Fl_Callback*)cb_Button_Generate);
          o->deactivate();
          w->hotspot(o);
        }
        { Fl_Return_Button* o = Button_Reset = new Fl_Return_Button(895, 494, 90, 25, "Reset");
```

```
          o->tooltip("Reset The Body Shape");
          o->box(FL_PLASTIC_UP_BOX);
          o->selection_color(16);
          o->labeltype(FL_EMBOSSED_LABEL);
          o->callback((Fl_Callback*)cb_Button_Reset);
        }
        o->end();
      }
      { Fl_Group* o = MainView = new Fl_Group(4, 48, 766, 635, "Deform View");
        o->box(FL_DOWN_BOX);
        o->labeltype(FL_ENGRAVED_LABEL);
        o->labelsize(11);
        o->align(FL_ALIGN_TOP_LEFT);
        { DeformView* o = DeformObj = new DeformView(4, 49, 764, 634, "This is the Deform GL
View");
          o->box(FL_DOWN_FRAME);
          o->color(49);
          o->selection_color(49);
          o->labeltype(FL_NORMAL_LABEL);
          o->labelfont(0);
          o->labelsize(14);
          o->labelcolor(56);
          o->align(FL_ALIGN_CENTER|FL_ALIGN_INSIDE);
          o->when(FL_WHEN_RELEASE);
        }
        o->end();
      }
      o->end();
    }
    o->end();
    o->resizable(o);
  }
  { Fl_Double_Window* o = aboutWindow = new Fl_Double_Window(300, 250, "About Deformer");
    w = o;
    o->user_data((void*)(this));
    { Fl_Return_Button* o = new Fl_Return_Button(100, 210, 100, 25, "OK");
      o->color(16);
      o->callback((Fl_Callback*)cb_OK);
    }
    { Fl_Box* o = new Fl_Box(10, 10, 280, 190, "Deformer User\nInterface\nVersion 1.1.1\n\nTexas
A&M University\n2005");
      o->box(FL_DOWN_BOX);
      o->color(9);
      o->labeltype(FL_SHADOW_LABEL);
    }
    o->end();
  }
}

void DeformUI::show(int argc, char **argv) {
  mainWindow->show(argc, argv);
}
```

B.3 Deformer Main View Header File Source Code

```
#ifndef DeformView_H
#define DeformView_H

//the order is important, if you put "Camera.h"
//in front of "DeformObj.h", the compilor will get crazy
//maybe Microsoft mean to do that:O)
//my guess is because the Vector and Quaternion header file
#include "Deformobj.h"
#include "Camera.h"
#include "Line.h"
#include "Vector.h"
#include "Matrix.h"
#include <vector>
#include <FL/Fl.H>
#include <FL/Fl_Gl_Window.H>

#include <FL/gl.h>
#ifdef __APPLE__
#include <OpenGL/gl.h>
#include <OpenGL/glu.h>
#else
#include <GL/glut.h>
#include <GL/glu.h>
#endif


enum gender{male, female, generic};

class DeformView : public Fl_Gl_Window
{
public:
        Matrix eigenVectorMatrixInv;
        Line eigenMaleOne, eigenMaleTwo, eigenFemaleOne, eigenFemaleTwo;
        double eigenWeightMaleOne, eigenWeightMaleTwo, eigenWeightFemaleOne,
        eigenWeightFemaleTwo;
        double size;
        Camera mainCam;
        float red, green, blue; //the color of the model
        bool model_loaded;
        Model3D sourceModel, targetModel_waist_max, targetModel_chest_max,
        targetModel_forarm_max, targetModel_bicep_max,  targetModel_thigh_max,
        targetModel_head_max, targetModel_hip_max, targetModel_calf_max,
        targetModel_waist_min, targetModel_chest_min, targetModel_forarm_min,
        targetModel_bicep_min,  targetModel_thigh_min,  targetModel_head_min,
        targetModel_hip_min, targetModel_calf_min, mainModel;

        Model3D sourceModel_male, mainModel_male,
        targetModel_waist_max_male, targetModel_chest_max_male, targetModel_forarm_max_male,
        targetModel_bicep_max_male, targetModel_thigh_max_male, targetModel_head_max_male,
        targetModel_hip_max_male, targetModel_calf_max_male,
        targetModel_waist_min_male, targetModel_chest_min_male, targetModel_forarm_min_male,
        targetModel_bicep_min_male, targetModel_thigh_min_male, targetModel_head_min_male,
        targetModel_hip_min_male, targetModel_calf_min_male,
        sourceModel_female, mainModel_female,
        targetModel_waist_max_female, targetModel_chest_max_female,
        targetModel_forarm_max_female,
        targetModel_bicep_max_female, targetModel_thigh_max_female, targetModel_head_max_female,
        targetModel_hip_max_female, targetModel_calf_max_female,
        targetModel_waist_min_female, targetModel_chest_min_female,
        targetModel_forarm_min_female,
        targetModel_bicep_min_female, targetModel_thigh_min_female, targetModel_head_min_female,
        targetModel_hip_min_female, targetModel_calf_min_female,

        mainModel_male_generic,
        mainModel_generic,
        mainModel_female_generic;

        GLuint mainModel_List;  // the opengl display list of the main model

        char sourcefilename[256];
        char target_waist_max_filename[256];
        char target_chest_max_filename[256];
        char target_forarm_max_filename[256];
        char target_bicep_max_filename[256];
        char target_thigh_max_filename[256];
        char target_head_max_filename[256];
        char target_hip_max_filename[256];
        char target_calf_max_filename[256];
        char target_waist_min_filename[256];
        char target_chest_min_filename[256];
        char target_forarm_min_filename[256];
        char target_bicep_min_filename[256];
        char target_thigh_min_filename[256];
        char target_head_min_filename[256];
        char target_hip_min_filename[256];
        char target_calf_min_filename[256];
        char mainfilename[256];
        ////////////////////////////////////


        DeformView(int x,int y,int w,int h,const char *l=0);

    /*The widget class draw() override.
     *
     *The draw() function initialize Gl for another round o f drawing
     * then calls specialized functions for drawing each of the
     * entities displayed in the cube view.
     *
     */
    void draw();
        int handle(int e);
        //the model which will be shown: 0.main 1.source 2.target
        int model_id;
        //the gender of the model
        int model_gender;
```

```
//interpolation method type: 0.linear 1.cosine 2.cubic
int interp_type;
//control type
int control_type;
//blending weight for each target model
double weight_waist_max_male;
double weight_chest_max_male;
double weight_forarm_max_male;
double weight_bicep_max_male;
double weight_thigh_max_male;
double weight_head_max_male;
double weight_hip_max_male;
double weight_calf_max_male;
double weight_waist_min_male;
double weight_chest_min_male;
double weight_forarm_min_male;
double weight_bicep_min_male;
double weight_thigh_min_male;
double weight_head_min_male;
double weight_hip_min_male;
double weight_calf_min_male;

double weight_waist_max_female;
double weight_chest_max_female;
double weight_forarm_max_female;
double weight_bicep_max_female;
double weight_thigh_max_female;
double weight_head_max_female;
double weight_hip_max_female;
double weight_calf_max_female;
double weight_waist_min_female;
double weight_chest_min_female;
double weight_forarm_min_female;
double weight_bicep_min_female;
double weight_thigh_min_female;
double weight_head_min_female;
double weight_hip_min_female;
double weight_calf_min_female;

//the body weight range for male and female
double body_weight_male_min;
double body_weight_male_max;
double body_weight_female_min;
double body_weight_female_max;

//blending weight for the gender
double value_body_weight;
double value_gender;

///////////////////////////////
double eigen_weight_male_one;
double eigen_weight_male_two;
double eigen_weight_female_one;
double eigen_weight_female_two;
double eigen_weight_array[8];
double orig_value_array_male[8];
double orig_value_array_female[8];
double orig_mean_array_male[8];
double orig_mean_array_female[8];
//the average value for each category
double waist_avg_male;
double chest_avg_male;
double forarm_avg_male;
double bicep_avg_male;
double thigh_avg_male;
double head_avg_male;
double hip_avg_male;
double calf_avg_male;

double waist_max_male;
double chest_max_male;
double forarm_max_male;
double bicep_max_male;
double thigh_max_male;
double head_max_male;
double hip_max_male;
double calf_max_male;

double waist_min_male;
double chest_min_male;
double forarm_min_male;
double bicep_min_male;
double thigh_min_male;
double head_min_male;
double hip_min_male;
double calf_min_male;

double waist_avg_female;
double chest_avg_female;
double forarm_avg_female;
double bicep_avg_female;
double thigh_avg_female;
double head_avg_female;
double hip_avg_female;
double calf_avg_female;

double waist_max_female;
double chest_max_female;
double forarm_max_female;
double bicep_max_female;
double thigh_max_female;
double head_max_female;
double hip_max_female;
double calf_max_female;

double waist_min_female;
double chest_min_female;
double forarm_min_female;
```

```
        double bicep_min_female;
        double thigh_min_female;
        double head_min_female;
        double hip_min_female;
        double calf_min_female;

        void calculate_generic_body_shapes();
        void blend_shape();
        void initialize();
        void saveModelInfos();
        void loadModelInfos();
 /* HAVE_GL */
private:
        // HANDLING MOUSE EVENTS
        int handle_mouse(int event, int button, int x, int y);
        // HANDLIGN KEYBORD EVENTS
        int handle_key(int event, int key);
        // Mouse position before pushing a button
        int old_x, old_y;

};
#endif
```

B.4 Deformer Main View Source File Source Code

```
#include "DeformView.h"
#include <math.h>

char * foo;

DeformView::DeformView(int x,int y,int w,int h,const char *l)
            : Fl_Gl_Window(x,y,w,h,l)
{
        mainCam.set(0.0, 20.0, 200.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);
        interp_type = 0;
        model_id = 0;
        red = green = blue = 1.0;
        weight_waist_max_male = 0.0;
        weight_chest_max_male = 0.0;
        weight_forarm_max_male = 0.0;
        weight_bicep_max_male = 0.0;
        weight_thigh_max_male = 0.0;
        weight_head_max_male = 0.0;
        weight_hip_max_male = 0.0;
        weight_calf_max_male = 0.0;
        weight_waist_min_male = 0.0;
        weight_chest_min_male = 0.0;
        weight_forarm_min_male = 0.0;
        weight_bicep_min_male = 0.0;
        weight_thigh_min_male = 0.0;
        weight_head_min_male = 0.0;
        weight_hip_min_male = 0.0;
        weight_calf_min_male = 0.0;

        weight_waist_max_female = 0.0;
        weight_chest_max_female = 0.0;
        weight_forarm_max_female = 0.0;
        weight_bicep_max_female = 0.0;
        weight_thigh_max_female = 0.0;
        weight_head_max_female = 0.0;
        weight_hip_max_female = 0.0;
        weight_calf_max_female = 0.0;
        weight_waist_min_female = 0.0;
        weight_chest_min_female = 0.0;
        weight_forarm_min_female = 0.0;
        weight_bicep_min_female = 0.0;
        weight_thigh_min_female = 0.0;
        weight_head_min_female = 0.0;
        weight_hip_min_female = 0.0;
        weight_calf_min_female = 0.0;
        //initialize the source and target file name string for fltk file chooser
        strcpy(sourcefilename, "*");
        strcpy(target_waist_max_filename, "*");
        strcpy(target_chest_max_filename, "*");
        strcpy(target_forarm_max_filename, "*");
        strcpy(target_bicep_max_filename, "*");
        strcpy(target_thigh_max_filename, "*");
        strcpy(target_head_max_filename, "*");
        strcpy(target_hip_max_filename, "*");
        strcpy(target_calf_max_filename, "*");
        strcpy(target_waist_min_filename, "*");
        strcpy(target_chest_min_filename, "*");
        strcpy(target_forarm_min_filename, "*");
        strcpy(target_bicep_min_filename, "*");
        strcpy(target_thigh_min_filename, "*");
        strcpy(target_head_min_filename, "*");
        strcpy(target_hip_min_filename, "*");
        strcpy(target_calf_min_filename, "*");
        strcpy(mainfilename, "*");
        model_loaded = false;
        model_gender = 3;       //set it neither male nor female at the beginning
}

void DeformView::initialize()
{
        int i, j;
        model_gender = 1;
        ////////////////////
        waist_max_male = 422;
        chest_max_male = 422;
        forarm_max_male = 372;
        bicep_max_male = 437;
        thigh_max_male = 787;
        head_max_male = 173;
        hip_max_male = 428;
```

```
    calf_max_male = 470;

    waist_min_male = 235;
    chest_min_male = 257;
    forarm_min_male = 233;
    bicep_min_male = 259;
    thigh_min_male = 458;
    head_min_male = 128;
    hip_min_male = 282;
    calf_min_male = 300;

    waist_max_female = 390;
    chest_max_female = 375;
    forarm_max_female = 325;
    bicep_max_female = 371;
    thigh_max_female = 748;
    head_max_female = 167;
    hip_max_female = 420;
    calf_max_female = 459;

    waist_min_female = 225;
    chest_min_female = 222;
    forarm_min_female = 210;
    bicep_min_female = 213;
    thigh_min_female = 454;
    head_min_female = 126;
    hip_min_female = 270;
    calf_min_female = 285;

    ////////////////////

    eigenMaleOne.a = 0.568;
    eigenMaleOne.b = -449.73;
    eigenMaleTwo.a = -0.1268;
    eigenMaleTwo.b = 100.69;

    eigenFemaleOne.a = 0.653;
    eigenFemaleOne.b = -406.66;
    eigenFemaleTwo.a = -0.1766;
    eigenFemaleTwo.b = 110.23;
    cout << "line intialized!" << endl;

    double M[] =
    {0.43996,0.313417,0.363306,0.327341,0.0448193,0.194733,0.558511,0.342341,
     0.44594,0.0197583,0.279198,0.455892,0.0662949,-0.335078,-0.616249,-0.136102,
     0.226877,0.282167,-0.451775,0.162712,-0.0315524,-0.225464,0.357672,-0.677181,
    -0.332663,0.870043,0.103074,-0.0591539,0.0157371,0.17388,-0.292771,-0.045164,
     0.334792,0.0744839,-0.713555,0.122946,-0.015645,0.37026,-0.274843,0.381071,
    -0.0424769,0.216877,-0.207121,-0.141602,0.0102814,-0.794098,0.100662,0.497372,
     0.575341,0.109918,0.145311,-0.779656,-0.0938827,0.0253626,-0.0931145,-0.0989709,
    -0.0229947,0.0109837,0.0464532,0.109571,-0.991559,-0.0201442,-0.0177883,0.0357216};
Matrix eigenVectorMatrix(8, 8, M);
    //since it's an orthogonal matrix, so the inverse is the same as transpose
    eigenVectorMatrixInv = eigenVectorMatrix.transpose();
```

```
eigenWeightMaleOne = 0.0;
eigenWeightMaleTwo = 0.0;
eigenWeightFemaleOne = 0.0;
eigenWeightFemaleTwo = 0.0;

body_weight_male_min = 476.0;
body_weight_male_max = 1278.0;
body_weight_female_min = 413.0;
body_weight_female_max = 967.0;

//////////////////////////
//arm_bicep_circ, calf_circ, chest_brth, forarm_circ, head_brth, hip_brth, thigh_circ,
//waist_brth
bicep_avg_male = 337.0;
calf_avg_male = 378.0;
chest_avg_male = 324.0;
forarm_avg_male = 303.0;
head_avg_male = 152.0;
hip_avg_male = 343.0;
thigh_avg_male = 596.0;
waist_avg_male = 312.0;
orig_mean_array_male[0] = 337.0;
orig_mean_array_male[1] = 378.0;
orig_mean_array_male[2] = 324.0;
orig_mean_array_male[3] = 303.0;
orig_mean_array_male[4] = 152.0;
orig_mean_array_male[5] = 343.0;
orig_mean_array_male[6] = 596.0;
orig_mean_array_male[7] = 312.0;

bicep_avg_female = 281.0;
calf_avg_female = 353.0;
chest_avg_female = 281.0;
forarm_avg_female = 254.0;
head_avg_female = 144.0;
hip_avg_female = 344.0;
thigh_avg_female = 581.0;
waist_avg_female = 291.0;
orig_mean_array_female[0] = 281.0;
orig_mean_array_female[1] = 353.0;
orig_mean_array_female[2] = 281.0;
orig_mean_array_female[3] = 254.0;
orig_mean_array_female[4] = 144.0;
orig_mean_array_female[5] = 344.0;
orig_mean_array_female[6] = 581.0;
orig_mean_array_female[7] = 291.0;


value_body_weight = 413;        //
value_gender = 1.0;     //fully female

eigen_weight_female_one = eigenFemaleOne.a * value_body_weight + eigenFemaleOne.b;
eigen_weight_female_two = eigenFemaleTwo.a * value_body_weight + eigenFemaleTwo.b;
```

```
eigen_weight_male_one = 0.0;
eigen_weight_male_two = 0.0;

eigen_weight_array[0] = eigen_weight_female_one;
eigen_weight_array[1] = eigen_weight_female_two;

for(i = 2; i < 8; i++)
        eigen_weight_array[i] = 0.0;

Vector eigen_vec(8, eigen_weight_array);
//get the orignal value for each body part by inverse transfer the eigen weight into the
//original data space
Vector orig_vec = eigenVectorMatrixInv * eigen_vec;

for(j = 0; j < 8; j++)
        orig_value_array_female[j] = orig_vec[j] + orig_mean_array_female[j];


if( (orig_value_array_female[0] - bicep_avg_female) >= 0.0)
{
        weight_bicep_max_female = orig_value_array_female[0] - bicep_avg_female;
        weight_bicep_max_female = weight_bicep_max_female /
                                  (bicep_max_female - bicep_avg_female);
        weight_bicep_min_female = 0.0;
}
else
{
        weight_bicep_min_female = orig_value_array_female[0] - bicep_avg_female;
        weight_bicep_min_female = weight_bicep_min_female /
                                 (bicep_min_female - bicep_avg_female);
        weight_bicep_max_female = 0.0;
}

if( (orig_value_array_female[1] - calf_avg_female) >= 0.0)
{
        weight_calf_max_female = orig_value_array_female[1] - calf_avg_female;
        weight_calf_max_female = weight_calf_max_female /
                                (calf_max_female - calf_avg_female);
        weight_calf_min_female = 0.0;
}
else
{
        weight_calf_min_female = orig_value_array_female[1] - calf_avg_female;
        weight_calf_min_female = weight_calf_min_female /
                                (calf_min_female - calf_avg_female);
        weight_calf_max_female = 0.0;
}

if( (orig_value_array_female[2] - chest_avg_female) >= 0.0)
{
        weight_chest_max_female = orig_value_array_female[2] - chest_avg_female;
        weight_chest_max_female = weight_chest_max_female /
                                (chest_max_female - chest_avg_female);
        weight_chest_min_female = 0.0;
}
else
{
        weight_chest_min_female = orig_value_array_female[2] - chest_avg_female;
        weight_chest_min_female = weight_chest_min_female /
                                  (chest_min_female - chest_avg_female);
        weight_chest_max_female = 0.0;
}

if( (orig_value_array_female[3] - forarm_avg_female) >= 0.0)
{
        weight_forarm_max_female = orig_value_array_female[3] - forarm_avg_female;
        weight_forarm_max_female = weight_forarm_max_female /
                                  (forarm_max_female - forarm_avg_female);
        weight_forarm_min_female = 0.0;
}
else
{
        weight_forarm_min_female = orig_value_array_female[3] - forarm_avg_female;
        weight_forarm_min_female = weight_forarm_min_female /
                                  (forarm_min_female - forarm_avg_female);
        weight_forarm_max_female = 0.0;
}

if( (orig_value_array_female[4] - head_avg_female) >= 0.0)
{
        weight_head_max_female = orig_value_array_female[4] - head_avg_female;
        weight_head_max_female = weight_head_max_female /
                                  (head_max_female - head_avg_female);
        weight_head_min_female = 0.0;
}
else
{
        weight_head_min_female = orig_value_array_female[4] - head_avg_female;
        weight_head_min_female = weight_head_min_female /
                                  (head_min_female - head_avg_female);
        weight_head_max_female = 0.0;
}

if( (orig_value_array_female[5] - hip_avg_female) >= 0.0)
{
        weight_hip_max_female = orig_value_array_female[5] - hip_avg_female;
        weight_hip_max_female = weight_hip_max_female /
                                  (hip_max_female - hip_avg_female);
        weight_hip_min_female = 0.0;
}
else
{
        weight_hip_min_female = orig_value_array_female[5] - hip_avg_female;
        weight_hip_min_female = weight_hip_min_female /
                                  (hip_min_female - hip_avg_female);
        weight_hip_max_female = 0.0;
}
```

```
if( (orig_value_array_female[6] - thigh_avg_female) >= 0.0)
{
        weight_thigh_max_female = orig_value_array_female[6] - thigh_avg_female;
        weight_thigh_max_female = weight_thigh_max_female /
                                (thigh_max_female - thigh_avg_female);
        weight_thigh_min_female = 0.0;
}
else
{
        weight_thigh_min_female = orig_value_array_female[6] - thigh_avg_female;
        weight_thigh_min_female = weight_thigh_min_female /
                                (thigh_min_female - thigh_avg_female);
        weight_thigh_max_female = 0.0;
}

if( (orig_value_array_female[7] - waist_avg_female) >= 0.0)
{
        weight_waist_max_female = orig_value_array_female[7] - waist_avg_female;
        weight_waist_max_female = weight_waist_max_female /
                                (waist_max_female - waist_avg_female);
        weight_waist_min_female = 0.0;
}
else
{
        weight_waist_min_female = orig_value_array_female[7] - waist_avg_female;
        weight_waist_min_female = weight_waist_min_female /
                                (waist_min_female - waist_avg_female);
        weight_waist_max_female = 0.0;
}

for(i = 0; i < mainModel_male.numOfObjects; i++)
{
        // Go through all of the faces (polygons) of the object and draw them
        for(j = 0; j < mainModel_male.pObject[i]->numOfFaces; j++)
        {
                // Go through each corner of the triangle
                for(int whichVertex = 0; whichVertex < 3; whichVertex++)
                {
                        // Get the vertex index for each point of the face
                        int vertIndex =
                        mainModel_male.pObject[i]->pFaces[j].vertIndex[whichVertex];

                        mainModel_generic.pObject[i]->pVerts[ vertIndex ] =
                        (targetModel_waist_max_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_waist_max_female
                        +
                        (targetModel_chest_max_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_chest_max_female
                        +
                        (targetModel_forarm_max_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_forarm_max_female
```

```
+
(targetModel_hip_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_hip_max_female
+
(targetModel_calf_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_calf_max_female
+
(targetModel_bicep_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_bicep_max_female
+
(targetModel_thigh_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_thigh_max_female
+
(targetModel_head_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_head_max_female
+
(targetModel_waist_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_waist_min_female
+
(targetModel_chest_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_chest_min_female
+
(targetModel_forarm_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_forarm_min_female
+
(targetModel_hip_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_hip_min_female
+
(targetModel_calf_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_calf_min_female
+
(targetModel_bicep_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_bicep_min_female
+
(targetModel_thigh_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_thigh_min_female
+
(targetModel_head_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_head_min_female
+ sourceModel_female.pObject[i]->pVerts[ vertIndex ];
```

```
                        mainModel_female_generic.pObject[i]->pVerts[ vertIndex ] =
                        mainModel_generic.pObject[i]->pVerts[ vertIndex ];
                        }
                }
        }
}

void DeformView::calculate_generic_body_shapes()
{
        int i, j;

        if(model_gender == 1)
        {
                cout << " calculate generic female body only" << endl;
                eigen_weight_female_one = eigenFemaleOne.a * value_body_weight
                                        + eigenFemaleOne.b;
                eigen_weight_female_two = eigenFemaleTwo.a * value_body_weight
                                        + eigenFemaleTwo.b;
                eigen_weight_male_one = 0.0;
                eigen_weight_male_two = 0.0;

                eigen_weight_array[0] = eigen_weight_female_one;
                eigen_weight_array[1] = eigen_weight_female_two;

                for(i = 2; i < 8; i++)
                        eigen_weight_array[i] = 0.0;

                Vector eigen_vec(8, eigen_weight_array);
                //get the orignal value for each body part by inverse transfer the eigen weight
                //into the original data space
                Vector orig_vec = eigenVectorMatrixInv * eigen_vec;

                for(j = 0; j < 8; j++)
                        orig_value_array_female[j] = orig_vec[j] + orig_mean_array_female[j];

                if( (orig_value_array_female[0] - bicep_avg_female) >= 0.0)
                {
                        weight_bicep_max_female = orig_value_array_female[0] - bicep_avg_female;
                        weight_bicep_max_female = weight_bicep_max_female /
                                                (bicep_max_female - bicep_avg_female);
                        weight_bicep_min_female = 0.0;
                }
                else
                {
                        weight_bicep_min_female = orig_value_array_female[0] - bicep_avg_female;
                        weight_bicep_min_female = weight_bicep_min_female /
                                                (bicep_min_female - bicep_avg_female);
                        weight_bicep_max_female = 0.0;
                }

                if( (orig_value_array_female[1] - calf_avg_female) >= 0.0)
                {
                        weight_calf_max_female = orig_value_array_female[1] - calf_avg_female;
                        weight_calf_max_female = weight_calf_max_female /
```

```
                                (calf_max_female - calf_avg_female);
        weight_calf_min_female = 0.0;
}
else
{
        weight_calf_min_female = orig_value_array_female[1] - calf_avg_female;
        weight_calf_min_female = weight_calf_min_female /
                                (calf_min_female - calf_avg_female);
        weight_calf_max_female = 0.0;
}

if( (orig_value_array_female[2] - chest_avg_female) >= 0.0)
{
        weight_chest_max_female = orig_value_array_female[2] - chest_avg_female;
        weight_chest_max_female = weight_chest_max_female /
                                (chest_max_female - chest_avg_female);
        weight_chest_min_female = 0.0;
}
else
{
        weight_chest_min_female = orig_value_array_female[2] - chest_avg_female;
        weight_chest_min_female = weight_chest_min_female /
                                (chest_min_female - chest_avg_female);
        weight_chest_max_female = 0.0;
}

if( (orig_value_array_female[3] - forarm_avg_female) >= 0.0)
{
        weight_forarm_max_female = orig_value_array_female[3] –
                                forarm_avg_female;
        weight_forarm_max_female = weight_forarm_max_female /
                                (forarm_max_female - forarm_avg_female);
        weight_forarm_min_female = 0.0;
}
else
{
        weight_forarm_min_female = orig_value_array_female[3] –
                                forarm_avg_female;
        weight_forarm_min_female = weight_forarm_min_female /
                                (forarm_min_female - forarm_avg_female);
        weight_forarm_max_female = 0.0;
}

if( (orig_value_array_female[4] - head_avg_female) >= 0.0)
{
        weight_head_max_female = orig_value_array_female[4] - head_avg_female;
        weight_head_max_female = weight_head_max_female /
                                (head_max_female - head_avg_female);
        weight_head_min_female = 0.0;
}
else
{
        weight_head_min_female = orig_value_array_female[4] - head_avg_female;
        weight_head_min_female = weight_head_min_female /
```

```
                                (head_min_female - head_avg_female);
        weight_head_max_female = 0.0;
}

if( (orig_value_array_female[5] - hip_avg_female) >= 0.0)
{
        weight_hip_max_female = orig_value_array_female[5] - hip_avg_female;
        weight_hip_max_female = weight_hip_max_female /
                                (hip_max_female - hip_avg_female);
        weight_hip_min_female = 0.0;
}
else
{
        weight_hip_min_female = orig_value_array_female[5] - hip_avg_female;
        weight_hip_min_female = weight_hip_min_female /
                                (hip_min_female - hip_avg_female);
        weight_hip_max_female = 0.0;
}

if( (orig_value_array_female[6] - thigh_avg_female) >= 0.0)
{
        weight_thigh_max_female = orig_value_array_female[6] - thigh_avg_female;
        weight_thigh_max_female = weight_thigh_max_female /
                                (thigh_max_female - thigh_avg_female);
        weight_thigh_min_female = 0.0;
}
else
{
        weight_thigh_min_female = orig_value_array_female[6] - thigh_avg_female;
        weight_thigh_min_female = weight_thigh_min_female /
                                (thigh_min_female - thigh_avg_female);
        weight_thigh_max_female = 0.0;
}

if( (orig_value_array_female[7] - waist_avg_female) >= 0.0)
{
        weight_waist_max_female = orig_value_array_female[7] - waist_avg_female;
        weight_waist_max_female = weight_waist_max_female /
                                (waist_max_female - waist_avg_female);
        weight_waist_min_female = 0.0;
}
else
{
        weight_waist_min_female = orig_value_array_female[7] - waist_avg_female;
        weight_waist_min_female = weight_waist_min_female /
                                (waist_min_female - waist_avg_female);
        weight_waist_max_female = 0.0;
}

for(i = 0; i < mainModel_male.numOfObjects; i++)
{
        // Go through all of the faces (polygons) of the object and draw them
        for(j = 0; j < mainModel_male.pObject[i]->numOfFaces; j++)
        {
```

```
// Go through each corner of the triangle
for(int whichVertex = 0; whichVertex < 3; whichVertex++)
{
        // Get the vertex index for each point of the face
int vertIndex =
mainModel_male.pObject[i]->pFaces[j].vertIndex[whichVertex];
mainModel_female_generic.pObject[i]->pVerts[ vertIndex ]
= (targetModel_waist_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_waist_max_female
+
(targetModel_chest_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_chest_max_female
+
(targetModel_forarm_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_forarm_max_female
+
(targetModel_hip_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_hip_max_female
+
(targetModel_calf_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_calf_max_female
+
(targetModel_bicep_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_bicep_max_female
+
(targetModel_thigh_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_thigh_max_female
+
(targetModel_head_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_head_max_female
+
(targetModel_waist_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_waist_min_female
+
(targetModel_chest_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_chest_min_female
+
(targetModel_forarm_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_forarm_min_female
+
(targetModel_hip_min_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_hip_min_female
```

```
                        +
                        (targetModel_calf_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_calf_min_female
                        +
                        (targetModel_bicep_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_bicep_min_female
                        +
                        (targetModel_thigh_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_thigh_min_female
                        +
                        (targetModel_head_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_head_min_female
                        + sourceModel_female.pObject[i]->pVerts[ vertIndex ];
                        }
                }
        }

}
else if(model_gender == 0)
{
        cout << " calculate generic male body only" << endl;
        eigen_weight_male_one = eigenMaleOne.a * value_body_weight + eigenMaleOne.b;
        eigen_weight_male_two = eigenMaleTwo.a * value_body_weight + eigenMaleTwo.b;
        eigen_weight_female_one = 0.0;
        eigen_weight_female_two = 0.0;

        eigen_weight_array[0] = eigen_weight_male_one;
        eigen_weight_array[1] = eigen_weight_male_two;

        for(i = 2; i < 8; i++)
                eigen_weight_array[i] = 0.0;

        Vector eigen_vec(8, eigen_weight_array);
        //get the orignal value for each body part by inverse transfer the eigen weight
        //into the original data space
        Vector orig_vec = eigenVectorMatrixInv * eigen_vec;

        for(j = 0; j < 8; j++)
                orig_value_array_male[j] = orig_vec[j] + orig_mean_array_male[j];

        if( (orig_value_array_male[0] - bicep_avg_male) >= 0.0)
        {
                weight_bicep_max_male = orig_value_array_male[0] - bicep_avg_male;
                weight_bicep_max_male = weight_bicep_max_male /
                                        (bicep_max_male - bicep_avg_male);
                weight_bicep_min_male = 0.0;
        }
        else
        {
                weight_bicep_min_male = orig_value_array_male[0] - bicep_avg_male;
                weight_bicep_min_male = weight_bicep_min_male /
                                        (bicep_min_male - bicep_avg_male);
                weight_bicep_max_male = 0.0;
        }

        if( (orig_value_array_male[1] - calf_avg_male) >= 0.0)
        {
                weight_calf_max_male = orig_value_array_male[1] - calf_avg_male;
                weight_calf_max_male = weight_calf_max_male /
                                        (calf_max_male - calf_avg_male);
                weight_calf_min_male = 0.0;
        }
        else
        {
                weight_calf_min_male = orig_value_array_male[1] - calf_avg_male;
                weight_calf_min_male = weight_calf_min_male /
                                        (calf_min_male - calf_avg_male);
                weight_calf_max_male = 0.0;
        }

        if( (orig_value_array_male[2] - chest_avg_male) >= 0.0)
        {
                weight_chest_max_male = orig_value_array_male[2] - chest_avg_male;
                weight_chest_max_male = weight_chest_max_male /
                                        (chest_max_male - chest_avg_male);
                weight_chest_min_male = 0.0;
        }
        else
        {
                weight_chest_min_male = orig_value_array_male[2] - chest_avg_male;
                weight_chest_min_male = weight_chest_min_male /
                                        (chest_min_male - chest_avg_male);
                weight_chest_max_male = 0.0;
        }

        if( (orig_value_array_male[3] - forarm_avg_male) >= 0.0)
        {
                weight_forarm_max_male = orig_value_array_male[3] - forarm_avg_male;
                weight_forarm_max_male = weight_forarm_max_male /
                                        (forarm_max_male - forarm_avg_male);
                weight_forarm_min_male = 0.0;
        }
        else
        {
                weight_forarm_min_male = orig_value_array_male[3] - forarm_avg_male;
                weight_forarm_min_male = weight_forarm_min_male /
                                        (forarm_min_male - forarm_avg_male);
                weight_forarm_max_male = 0.0;
        }

        if( (orig_value_array_male[4] - head_avg_male) >= 0.0)
        {
                weight_head_max_male = orig_value_array_male[4] - head_avg_male;
                weight_head_max_male = weight_head_max_male /
```

```
                                (head_max_male - head_avg_male);
        weight_head_min_male = 0.0;
}
else
{
        weight_head_min_male = orig_value_array_male[4] - head_avg_male;
        weight_head_min_male = weight_head_min_male /
                                (head_min_male - head_avg_male);
        weight_head_max_male = 0.0;
}

if( (orig_value_array_male[5] - hip_avg_male) >= 0.0)
{
        weight_hip_max_male = orig_value_array_male[5] - hip_avg_male;
        weight_hip_max_male = weight_hip_max_male /
                                (hip_max_male - hip_avg_male);
        weight_hip_min_male = 0.0;
}
else
{
        weight_hip_min_male = orig_value_array_male[5] - hip_avg_male;
        weight_hip_min_male = weight_hip_min_male /
                                (hip_min_male - hip_avg_male);
        weight_hip_max_male = 0.0;
}

if( (orig_value_array_male[6] - thigh_avg_male) >= 0.0)
{
        weight_thigh_max_male = orig_value_array_male[6] - thigh_avg_male;
        weight_thigh_max_male = weight_thigh_max_male /
                                (thigh_max_male - thigh_avg_male);
        weight_thigh_min_male = 0.0;
}
else
{
        weight_thigh_min_male = orig_value_array_male[6] - thigh_avg_male;
        weight_thigh_min_male = weight_thigh_min_male /
                                (thigh_min_male - thigh_avg_male);
        weight_thigh_max_male = 0.0;
}

if( (orig_value_array_male[7] - waist_avg_male) >= 0.0)
{
        weight_waist_max_male = orig_value_array_male[7] - waist_avg_male;
        weight_waist_max_male = weight_waist_max_male /
                                (waist_max_male - waist_avg_male);
        weight_waist_min_male = 0.0;
}
else
{
        weight_waist_min_male = orig_value_array_male[7] - waist_avg_male;
        weight_waist_min_male = weight_waist_min_male /
                                (waist_min_male - waist_avg_male);
        weight_waist_max_male = 0.0;
```

```
}

for(i = 0; i < mainModel_male.numOfObjects; i++)
{
        // Go through all of the faces (polygons) of the object and draw them
        for(j = 0; j < mainModel_male.pObject[i]->numOfFaces; j++)
        {
                // Go through each corner of the triangle
                for(int whichVertex = 0; whichVertex < 3; whichVertex++)
                {
                // Get the vertex index for each point of the face
                int vertIndex =
                mainModel_male.pObject[i]->pFaces[j].vertIndex[whichVertex];
                mainModel_male_generic.pObject[i]->pVerts[ vertIndex ] =
                (targetModel_waist_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_waist_max_male
                +
                (targetModel_chest_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_chest_max_male
                +
                (targetModel_forarm_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_forarm_max_male
                +
                (targetModel_hip_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_hip_max_male
                +
                (targetModel_calf_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_calf_max_male
                +
                (targetModel_bicep_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_bicep_max_male
                +
                (targetModel_thigh_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_thigh_max_male
                +
                (targetModel_head_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_head_max_male
                +
                (targetModel_waist_min_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_waist_min_male
                +
                (targetModel_chest_min_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_chest_min_male
                +
                        (targetModel_forarm_min_male.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                        * weight_forarm_min_male
                        +
                        (targetModel_hip_min_male.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                        * weight_hip_min_male
                        +
                        (targetModel_calf_min_male.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                        * weight_calf_min_male
                        +
                        (targetModel_bicep_min_male.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                        * weight_bicep_min_male
                        +
                        (targetModel_thigh_min_male.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                        * weight_thigh_min_male
                        +
                        (targetModel_head_min_male.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                        * weight_head_min_male
                        + sourceModel_male.pObject[i]->pVerts[ vertIndex ];
                        }
                }
        }

}
else
{
        cout << " calculate generic male and female body only" << endl;
        eigen_weight_female_one = eigenFemaleOne.a * value_body_weight
                                + eigenFemaleOne.b;
        eigen_weight_female_two = eigenFemaleTwo.a * value_body_weight
                                + eigenFemaleTwo.b;
        eigen_weight_male_one = 0.0;
        eigen_weight_male_two = 0.0;

        eigen_weight_array[0] = eigen_weight_female_one;
        eigen_weight_array[1] = eigen_weight_female_two;

        for(i = 2; i < 8; i++)
                eigen_weight_array[i] = 0.0;

        Vector eigen_vec_female(8, eigen_weight_array);
        //get the orignal value for each body part by inverse transfer the eigen weight
        //into the original data space
        Vector orig_vec_female = eigenVectorMatrixInv * eigen_vec_female;

        for(j = 0; j < 8; j++)
                orig_value_array_female[j] = orig_vec_female[j]
                                        + orig_mean_array_female[j];
```

```
if( (orig_value_array_female[0] - bicep_avg_female) >= 0.0)
{
        weight_bicep_max_female = orig_value_array_female[0] - bicep_avg_female;
        weight_bicep_max_female = weight_bicep_max_female /
                                 (bicep_max_female - bicep_avg_female);
        weight_bicep_min_female = 0.0;
}
else
{
        weight_bicep_min_female = orig_value_array_female[0] - bicep_avg_female;
        weight_bicep_min_female = weight_bicep_min_female /
                                 (bicep_min_female - bicep_avg_female);
        weight_bicep_max_female = 0.0;
}

if( (orig_value_array_female[1] - calf_avg_female) >= 0.0)
{
        weight_calf_max_female = orig_value_array_female[1] - calf_avg_female;
        weight_calf_max_female = weight_calf_max_female /
                                 (calf_max_female - calf_avg_female);
        weight_calf_min_female = 0.0;
}
else
{
        weight_calf_min_female = orig_value_array_female[1] - calf_avg_female;
        weight_calf_min_female = weight_calf_min_female /
                                 (calf_min_female - calf_avg_female);
        weight_calf_max_female = 0.0;
}

if( (orig_value_array_female[2] - chest_avg_female) >= 0.0)
{
        weight_chest_max_female = orig_value_array_female[2] - chest_avg_female;
        weight_chest_max_female = weight_chest_max_female /
                                 (chest_max_female - chest_avg_female);
        weight_chest_min_female = 0.0;
}
else
{
        weight_chest_min_female = orig_value_array_female[2] - chest_avg_female;
        weight_chest_min_female = weight_chest_min_female /
                                 (chest_min_female - chest_avg_female);
        weight_chest_max_female = 0.0;
}

if( (orig_value_array_female[3] - forarm_avg_female) >= 0.0)
{
        weight_forarm_max_female = orig_value_array_female[3] –
                                  forarm_avg_female;
        weight_forarm_max_female = weight_forarm_max_female /
                                 (forarm_max_female - forarm_avg_female);
        weight_forarm_min_female = 0.0;
}
else
```

```
{
        weight_forarm_min_female = orig_value_array_female[3] –
                                forarm_avg_female;
        weight_forarm_min_female = weight_forarm_min_female /
                                (forarm_min_female - forarm_avg_female);
        weight_forarm_max_female = 0.0;
}

if( (orig_value_array_female[4] - head_avg_female) >= 0.0)
{
        weight_head_max_female = orig_value_array_female[4] - head_avg_female;
        weight_head_max_female = weight_head_max_female /
                                (head_max_female - head_avg_female);
        weight_head_min_female = 0.0;
}
else
{
        weight_head_min_female = orig_value_array_female[4] - head_avg_female;
        weight_head_min_female = weight_head_min_female /
                                (head_min_female - head_avg_female);
        weight_head_max_female = 0.0;
}

if( (orig_value_array_female[5] - hip_avg_female) >= 0.0)
{
        weight_hip_max_female = orig_value_array_female[5] - hip_avg_female;
        weight_hip_max_female = weight_hip_max_female /
                                (hip_max_female - hip_avg_female);
        weight_hip_min_female = 0.0;
}
else
{
        weight_hip_min_female = orig_value_array_female[5] - hip_avg_female;
        weight_hip_min_female = weight_hip_min_female /
                                (hip_min_female - hip_avg_female);
        weight_hip_max_female = 0.0;
}

if( (orig_value_array_female[6] - thigh_avg_female) >= 0.0)
{
        weight_thigh_max_female = orig_value_array_female[6] - thigh_avg_female;
        weight_thigh_max_female = weight_thigh_max_female /
                                (thigh_max_female - thigh_avg_female);
        weight_thigh_min_female = 0.0;
}
else
{
        weight_thigh_min_female = orig_value_array_female[6] - thigh_avg_female;
        weight_thigh_min_female = weight_thigh_min_female /
                                (thigh_min_female - thigh_avg_female);
        weight_thigh_max_female = 0.0;
}

if( (orig_value_array_female[7] - waist_avg_female) >= 0.0)
```

```
{
        weight_waist_max_female = orig_value_array_female[7] - waist_avg_female;
        weight_waist_max_female = weight_waist_max_female /
                                (waist_max_female - waist_avg_female);
        weight_waist_min_female = 0.0;
}
else
{
        weight_waist_min_female = orig_value_array_female[7] - waist_avg_female;
        weight_waist_min_female = weight_waist_min_female /
                                (waist_min_female - waist_avg_female);
        weight_waist_max_female = 0.0;
}

for(i = 0; i < mainModel_male.numOfObjects; i++)
{
        // Go through all of the faces (polygons) of the object and draw them
        for(j = 0; j < mainModel_male.pObject[i]->numOfFaces; j++)
        {
                // Go through each corner of the triangle
                for(int whichVertex = 0; whichVertex < 3; whichVertex++)
                {
                // Get the vertex index for each point of the face
                int vertIndex =
                mainModel_male.pObject[i]->pFaces[j].vertIndex[whichVertex];
                mainModel_female_generic.pObject[i]->pVerts[ vertIndex ] =
                (targetModel_waist_max_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_waist_max_female
                +
                (targetModel_chest_max_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_chest_max_female
                +
                (targetModel_forarm_max_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_forarm_max_female
                +
                (targetModel_hip_max_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_hip_max_female
                +
                (targetModel_calf_max_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_calf_max_female
                +
                (targetModel_bicep_max_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_bicep_max_female
                +
                (targetModel_thigh_max_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_thigh_max_female
                +
                (targetModel_head_max_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_head_max_female
                +
                (targetModel_waist_min_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_waist_min_female
                +
                (targetModel_chest_min_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_chest_min_female
                +
                (targetModel_forarm_min_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_forarm_min_female
                +
                (targetModel_hip_min_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_hip_min_female
                +
                (targetModel_calf_min_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_calf_min_female
                +
                (targetModel_bicep_min_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_bicep_min_female
                +
                (targetModel_thigh_min_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_thigh_min_female
                +
                (targetModel_head_min_female.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                * weight_head_min_female
                + sourceModel_female.pObject[i]->pVerts[ vertIndex ];
                }
        }
}
///////////////////////////////////////////////////////////////////////////////
eigen_weight_male_one = eigenMaleOne.a * value_body_weight + eigenMaleOne.b;
eigen_weight_male_two = eigenMaleTwo.a * value_body_weight + eigenMaleTwo.b;
eigen_weight_female_one = 0.0;
eigen_weight_female_two = 0.0;

eigen_weight_array[0] = eigen_weight_male_one;
eigen_weight_array[1] = eigen_weight_male_two;

for(i = 2; i < 8; i++)
        eigen_weight_array[i] = 0.0;

Vector eigen_vec_male(8, eigen_weight_array);
//get the orignal value for each body part by inverse transfer the eigen weight
//into the original data space
```

```
Vector orig_vec_male = eigenVectorMatrixInv * eigen_vec_male;

for(j = 0; j < 8; j++)
        orig_value_array_male[j] = orig_vec_male[j] + orig_mean_array_male[j];

if( (orig_value_array_male[0] - bicep_avg_male) >= 0.0)
{
        weight_bicep_max_male = orig_value_array_male[0] - bicep_avg_male;
        weight_bicep_max_male = weight_bicep_max_male /
                                (bicep_max_male - bicep_avg_male);
        weight_bicep_min_male = 0.0;
}
else
{
        weight_bicep_min_male = orig_value_array_male[0] - bicep_avg_male;
        weight_bicep_min_male = weight_bicep_min_male /
                                (bicep_min_male - bicep_avg_male);
        weight_bicep_max_male = 0.0;
}

if( (orig_value_array_male[1] - calf_avg_male) >= 0.0)
{
        weight_calf_max_male = orig_value_array_male[1] - calf_avg_male;
        weight_calf_max_male = weight_calf_max_male /
                                (calf_max_male - calf_avg_male);
        weight_calf_min_male = 0.0;
}
else
{
        weight_calf_min_male = orig_value_array_male[1] - calf_avg_male;
        weight_calf_min_male = weight_calf_min_male /
                                (calf_min_male - calf_avg_male);
        weight_calf_max_male = 0.0;
}

if( (orig_value_array_male[2] - chest_avg_male) >= 0.0)
{
        weight_chest_max_male = orig_value_array_male[2] - chest_avg_male;
        weight_chest_max_male = weight_chest_max_male /
                                (chest_max_male - chest_avg_male);
        weight_chest_min_male = 0.0;
}
else
{
        weight_chest_min_male = orig_value_array_male[2] - chest_avg_male;
        weight_chest_min_male = weight_chest_min_male /
                                (chest_min_male - chest_avg_male);
        weight_chest_max_male = 0.0;
}

if( (orig_value_array_male[3] - forarm_avg_male) >= 0.0)
{
        weight_forarm_max_male = orig_value_array_male[3] - forarm_avg_male;
        weight_forarm_max_male = weight_forarm_max_male /
```

```
                                (forarm_max_male - forarm_avg_male);
        weight_forarm_min_male = 0.0;
}
else
{
        weight_forarm_min_male = orig_value_array_male[3] - forarm_avg_male;
        weight_forarm_min_male = weight_forarm_min_male /
                                (forarm_min_male - forarm_avg_male);
        weight_forarm_max_male = 0.0;
}

if( (orig_value_array_male[4] - head_avg_male) >= 0.0)
{
        weight_head_max_male = orig_value_array_male[4] - head_avg_male;
        weight_head_max_male = weight_head_max_male /
                                (head_max_male - head_avg_male);
        weight_head_min_male = 0.0;
}
else
{
        weight_head_min_male = orig_value_array_male[4] - head_avg_male;
        weight_head_min_male = weight_head_min_male /
                                (head_min_male - head_avg_male);
        weight_head_max_male = 0.0;
}

if( (orig_value_array_male[5] - hip_avg_male) >= 0.0)
{
        weight_hip_max_male = orig_value_array_male[5] - hip_avg_male;
        weight_hip_max_male = weight_hip_max_male /
                                (hip_max_male - hip_avg_male);
        weight_hip_min_male = 0.0;
}
else
{
        weight_hip_min_male = orig_value_array_male[5] - hip_avg_male;
        weight_hip_min_male = weight_hip_min_male /
                                (hip_min_male - hip_avg_male);
        weight_hip_max_male = 0.0;
}

if( (orig_value_array_male[6] - thigh_avg_male) >= 0.0)
{
        weight_thigh_max_male = orig_value_array_male[6] - thigh_avg_male;
        weight_thigh_max_male = weight_thigh_max_male /
                                (thigh_max_male - thigh_avg_male);
        weight_thigh_min_male = 0.0;
}
else
{
        weight_thigh_min_male = orig_value_array_male[6] - thigh_avg_male;
        weight_thigh_min_male = weight_thigh_min_male /
                                (thigh_min_male - thigh_avg_male);
        weight_thigh_max_male = 0.0;
```

```
}

if( (orig_value_array_male[7] - waist_avg_male) >= 0.0)
{
        weight_waist_max_male = orig_value_array_male[7] - waist_avg_male;
        weight_waist_max_male = weight_waist_max_male /
                                (waist_max_male - waist_avg_male);
        weight_waist_min_male = 0.0;
}
else
{
        weight_waist_min_male = orig_value_array_male[7] - waist_avg_male;
        weight_waist_min_male = weight_waist_min_male /
                                (waist_min_male - waist_avg_male);
        weight_waist_max_male = 0.0;
}

for(i = 0; i < mainModel_male.numOfObjects; i++)
{
        // Go through all of the faces (polygons) of the object and draw them
        for(j = 0; j < mainModel_male.pObject[i]->numOfFaces; j++)
        {
                // Go through each corner of the triangle
                for(int whichVertex = 0; whichVertex < 3; whichVertex++)
                {
                // Get the vertex index for each point of the face
                int vertIndex =
                mainModel_male.pObject[i]->pFaces[j].vertIndex[whichVertex];
                mainModel_male_generic.pObject[i]->pVerts[ vertIndex ] =
                (targetModel_waist_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_waist_max_male
                +
                (targetModel_chest_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_chest_max_male
                +
                (targetModel_forarm_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_forarm_max_male
                +
                (targetModel_hip_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_hip_max_male
                +
                (targetModel_calf_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_calf_max_male
                +
                (targetModel_bicep_max_male.pObject[i]->pVerts[ vertIndex ]
                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                * weight_bicep_max_male
                +
                (targetModel_thigh_max_male.pObject[i]->pVerts[ vertIndex ]
                                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                                * weight_thigh_max_male
                                +
                                (targetModel_head_max_male.pObject[i]->pVerts[ vertIndex ]
                                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                                * weight_head_max_male
                                +
                                (targetModel_waist_min_male.pObject[i]->pVerts[ vertIndex ]
                                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                                * weight_waist_min_male
                                +
                                (targetModel_chest_min_male.pObject[i]->pVerts[ vertIndex ]
                                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                                * weight_chest_min_male
                                +
                                (targetModel_forarm_min_male.pObject[i]->pVerts[ vertIndex ]
                                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                                * weight_forarm_min_male
                                +
                                (targetModel_hip_min_male.pObject[i]->pVerts[ vertIndex ]
                                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                                * weight_hip_min_male
                                +
                                (targetModel_calf_min_male.pObject[i]->pVerts[ vertIndex ]
                                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                                * weight_calf_min_male
                                +
                                (targetModel_bicep_min_male.pObject[i]->pVerts[ vertIndex ]
                                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                                * weight_bicep_min_male
                                +
                                (targetModel_thigh_min_male.pObject[i]->pVerts[ vertIndex ]
                                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                                * weight_thigh_min_male
                                +
                                (targetModel_head_min_male.pObject[i]->pVerts[ vertIndex ]
                                - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
                                * weight_head_min_male
                                + sourceModel_male.pObject[i]->pVerts[ vertIndex ];
                                }
                        }
                }
        }
}

void DeformView::blend_shape()
{
        //Vector3d mainXYZ, sourceXYZ;
        for(int i = 0; i < mainModel_male.numOfObjects; i++)
        {
                // Go through all of the faces (polygons) of the object and draw them
                for(int j = 0; j < mainModel_male.pObject[i]->numOfFaces; j++)
                {
                        // Go through each corner of the triangle
```

```
for(int whichVertex = 0; whichVertex < 3; whichVertex++)
{
        // Get the vertex index for each point of the face
        int vertIndex =
        mainModel_male.pObject[i]->pFaces[j].vertIndex[whichVertex];

        switch(interp_type)
        {
        case 0:
        // Linear Interpolation
        if(control_type == 0)
        mainModel_male.pObject[i]->pVerts[ vertIndex ] =
        (targetModel_waist_max_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
        * weight_waist_max_male
        +
        (targetModel_chest_max_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
        * weight_chest_max_male
        +
        (targetModel_forarm_max_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
        * weight_forarm_max_male
        +
        (targetModel_hip_max_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
        * weight_hip_max_male
        +
        (targetModel_calf_max_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
        * weight_calf_max_male
        +
        (targetModel_bicep_max_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
        * weight_bicep_max_male
        +
        (targetModel_thigh_max_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
        * weight_thigh_max_male
        +
        (targetModel_head_max_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
        * weight_head_max_male
        +
        (targetModel_waist_min_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
        * weight_waist_min_male
        +
        (targetModel_chest_min_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
        * weight_chest_min_male
        +
        (targetModel_forarm_min_male.pObject[i]->pVerts[ vertIndex ]
        - sourceModel_male.pObject[i]->pVerts[ vertIndex ])
```

```
* weight_forarm_min_male
+
(targetModel_hip_min_male.pObject[i]->pVerts[ vertIndex ]
- sourceModel_male.pObject[i]->pVerts[ vertIndex ])
* weight_hip_min_male
+
(targetModel_calf_min_male.pObject[i]->pVerts[ vertIndex ]
- sourceModel_male.pObject[i]->pVerts[ vertIndex ])
* weight_calf_min_male
+
(targetModel_bicep_min_male.pObject[i]->pVerts[ vertIndex ]
- sourceModel_male.pObject[i]->pVerts[ vertIndex ])
* weight_bicep_min_male
+
(targetModel_thigh_min_male.pObject[i]->pVerts[ vertIndex ]
- sourceModel_male.pObject[i]->pVerts[ vertIndex ])
* weight_thigh_min_male
+
(targetModel_head_min_male.pObject[i]->pVerts[ vertIndex ]
- sourceModel_male.pObject[i]->pVerts[ vertIndex ])
* weight_head_min_male
+ sourceModel_male.pObject[i]->pVerts[ vertIndex ];
else if(control_type == 1)
mainModel_female.pObject[i]->pVerts[ vertIndex ] =
(targetModel_waist_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_waist_max_female
+
(targetModel_chest_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_chest_max_female
+
(targetModel_forarm_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_forarm_max_female
+
(targetModel_hip_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_hip_max_female
+
(targetModel_calf_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_calf_max_female
+
(targetModel_bicep_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_bicep_max_female
+
(targetModel_thigh_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
* weight_thigh_max_female
+
(targetModel_head_max_female.pObject[i]->pVerts[ vertIndex ]
- sourceModel_female.pObject[i]->pVerts[ vertIndex ])
```

```
                        * weight_head_max_female
                        +
                        (targetModel_waist_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_waist_min_female
                        +
                        (targetModel_chest_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_chest_min_female
                        +
                        (targetModel_forarm_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_forarm_min_female
                        +
                        (targetModel_hip_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_hip_min_female
                        +
                        (targetModel_calf_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_calf_min_female
                        +
                        (targetModel_bicep_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_bicep_min_female
                        +
                        (targetModel_thigh_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_thigh_min_female
                        +
                        (targetModel_head_min_female.pObject[i]->pVerts[ vertIndex ]
                        - sourceModel_female.pObject[i]->pVerts[ vertIndex ])
                        * weight_head_min_female
                        + sourceModel_female.pObject[i]->pVerts[ vertIndex ];

                        else if(control_type == 2)
                        {
                        mainModel_generic.pObject[i]->pVerts[ vertIndex ] =
                        (mainModel_male_generic.pObject[i]->pVerts[ vertIndex ]
                        - mainModel_female_generic.pObject[i]->pVerts[ vertIndex ])
                        * (1.0 - value_gender)
                        + mainModel_female_generic.pObject[i]->pVerts[ vertIndex ];
                        }
                        break;
                        case 1:
                        break;
                        case 2:
                        break;
                        }

                }
        }

}
```

```
        if(control_type == 0)
                mainModel_male.ComputeNormals();
        else if(control_type == 1)
                mainModel_female.ComputeNormals();
        else if(control_type == 2)
                mainModel_generic.ComputeNormals();
        redraw();
}

void DeformView::saveModelInfos()
{
        ;
}

void DeformView::loadModelInfos()
{
        ;
}

void DeformView::draw()
{
    if (!valid())
        {
        glLoadIdentity();
        glViewport(0, 0, w(), h());
                //glOrtho(-10,10,-10,10,-20030,10000);
                glMatrixMode(GL_PROJECTION);
                glLoadIdentity();
                gluPerspective(60, (float)w()/h(), 1.0, 10000);
                glClearColor(0.5, 0.5, 0.5, 0.0);
                //glClearColor(1.0, 1.0, 1.0, 1.0);

                gl_font(FL_TIMES, 16 );

                glEnable(GL_LIGHT0);
        // Turn on a light with defaults set
                glEnable(GL_LIGHTING);
        // Turn on lighting
                glEnable(GL_COLOR_MATERIAL);
                glEnable(GL_DEPTH_TEST);

        }

        glMatrixMode(GL_MODELVIEW);
        glLoadIdentity();

    mainCam.setModelViewMatrix();

    glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
        glColor3f(red, green, blue);
        gl_font(FL_TIMES, 14);

        glPushMatrix();
        if(model_id == 0 && mainModel_male.numOfObjects > 0)
```

```
{
        if(control_type == 0)
        {
                mainModel_male.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Main Male Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }
        else if(control_type == 1)
        {
                mainModel_female.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Main Female Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }
        else if(control_type == 2)
        {
                mainModel_generic.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Main Generic Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }

        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glPopAttrib();

}
else if(model_id == 1 && sourceModel_male.numOfObjects > 0)
{
        if(control_type == 0)
        {
                sourceModel_male.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Original Male Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }
        else if(control_type == 1)
        {
                sourceModel_female.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Original Female Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }

        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glPopAttrib();
}
else if(model_id == 2 && targetModel_waist_max_male.numOfObjects > 0)
{
        //for now, i just display all the target models, future will provide choice to
        //display each model
        if(control_type == 0)
        {
                targetModel_waist_max_male.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Fat Belly Male Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }
        else if(control_type == 1)
        {
                targetModel_waist_max_female.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Fat Belly Female Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }

        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glPopAttrib();
}
else if(model_id == 3 && targetModel_chest_max_male.numOfObjects > 0)
{
        //for now, i just display all the target models, future will provide choice to
        //display each model
        if(control_type == 0)
        {
                targetModel_chest_max_male.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Big Chest Male Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }
        else if(control_type == 1)
        {
                targetModel_chest_max_female.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Big Chest Female Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }

        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glPopAttrib();
}
```

```
else if(model_id == 4 && targetModel_forarm_max_male.numOfObjects > 0)
{
        //for now, i just display all the target models, future will provide choice to
        //display each model
        if(control_type == 0)
        {
                targetModel_forarm_max_male.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Thin ForArm Male Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }
        else if(control_type == 1)
        {
                targetModel_forarm_max_female.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Thin ForArm Female Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }

        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glPopAttrib();

}
else if(model_id == 5 && targetModel_bicep_max_male.numOfObjects > 0)
{
		//for now, i just display all the target models, future will provide choice to
		//display each model
		if(control_type == 0)
		{
			targetModel_bicep_max_male.DisplayGL();
			glPopMatrix();
			//save states
			glPushAttrib(GL_ALL_ATTRIB_BITS);
			//set modelview matrix
			glPushMatrix();
			glLoadIdentity();
			//set projection matrix
			glMatrixMode(GL_PROJECTION);
			glPushMatrix();
			glLoadIdentity();
			glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

			gl_color(FL_WHITE);
			glDisable(GL_DEPTH_TEST);
			gl_draw("Current: Big Arm Male Body Model", 5.0f, 25.0f );
			glEnable(GL_DEPTH_TEST);
		}
		else if(control_type == 1)
		{
			targetModel_bicep_max_female.DisplayGL();
			glPopMatrix();
			//save states
			glPushAttrib(GL_ALL_ATTRIB_BITS);
			//set modelview matrix
			glPushMatrix();
			glLoadIdentity();
			//set projection matrix
			glMatrixMode(GL_PROJECTION);
			glPushMatrix();
			glLoadIdentity();
			glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

			gl_color(FL_WHITE);
			glDisable(GL_DEPTH_TEST);
			gl_draw("Current: Big Arm Female Body Model", 5.0f, 25.0f );
			glEnable(GL_DEPTH_TEST);
		}

		glPopMatrix();
		glMatrixMode(GL_MODELVIEW);
		glPopMatrix();
		glPopAttrib();
	}
	else if(model_id == 6 && targetModel_thigh_max_male.numOfObjects > 0)
	{
		//for now, i just display all the target models, future will provide choice to
		//display each model
		if(control_type == 0)
		{
```

```
                targetModel_thigh_max_male.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Big Leg Male Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }
        else if(control_type == 1)
        {
                targetModel_thigh_max_female.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Big Leg Female Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }

        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glPopAttrib();
}
else if(model_id == 7 && targetModel_head_max_male.numOfObjects > 0)
{
        //for now, i just display all the target models, future will provide choice to
        //display each model
        if(control_type == 0)
        {
                targetModel_head_max_male.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Huge Head Male Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }
        else if(control_type == 1)
        {
                targetModel_head_max_female.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Huge Head Female Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }

        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glPopAttrib();
}
else if(model_id == 8 && targetModel_calf_max_male.numOfObjects > 0)
{
        //for now, i just display all the target models, future will provide choice to
        //display each model
        if(control_type == 0)
        {
                targetModel_calf_max_male.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Calf Male Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }
        else if(control_type == 1)
        {
                targetModel_calf_max_female.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                gl_color(FL_WHITE);
                glDisable(GL_DEPTH_TEST);
                gl_draw("Current: Calf Female Body Model", 5.0f, 25.0f );
                glEnable(GL_DEPTH_TEST);
        }

        glPopMatrix();
        glMatrixMode(GL_MODELVIEW);
        glPopMatrix();
        glPopAttrib();
}
else if(model_id == 9 && targetModel_hip_max_male.numOfObjects > 0)
{
        //for now, i just display all the target models, future will provide choice to
        //display each model
        if(control_type == 0)
        {
                targetModel_hip_max_male.DisplayGL();
                glPopMatrix();
                //save states
                glPushAttrib(GL_ALL_ATTRIB_BITS);
                //set modelview matrix
                glPushMatrix();
                glLoadIdentity();
                //set projection matrix
                glMatrixMode(GL_PROJECTION);
                glPushMatrix();
                glLoadIdentity();
                glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);
```

```
                    gl_color(FL_WHITE);
                    glDisable(GL_DEPTH_TEST);
                    gl_draw("Current: Hip Male Body Model", 5.0f, 25.0f );
                    glEnable(GL_DEPTH_TEST);
            }
            else if(control_type == 1)
            {
                    targetModel_hip_max_female.DisplayGL();
                    glPopMatrix();
                    //save states
                    glPushAttrib(GL_ALL_ATTRIB_BITS);
                    //set modelview matrix
                    glPushMatrix();
                    glLoadIdentity();
                    //set projection matrix
                    glMatrixMode(GL_PROJECTION);
                    glPushMatrix();
                    glLoadIdentity();
                    glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

                    gl_color(FL_WHITE);
                    glDisable(GL_DEPTH_TEST);
                    gl_draw("Current: Hip Female Body Model", 5.0f, 25.0f );
                    glEnable(GL_DEPTH_TEST);
            }

            glPopMatrix();
            glMatrixMode(GL_MODELVIEW);
            glPopMatrix();
            glPopAttrib();
    }

glPopMatrix();

    //save states
    glPushAttrib(GL_ALL_ATTRIB_BITS);
    //set modelview matrix
    glPushMatrix();
    glLoadIdentity();
    //set projection matrix
    glMatrixMode(GL_PROJECTION);
    glPushMatrix();
    glLoadIdentity();
    glOrtho(0.0f, w(), h(), 0.0f, -1.0f, 1.0f);

    gl_font(FL_TIMES_BOLD, 18 );
    gl_color(FL_WHITE);
    glDisable(GL_DEPTH_TEST);
    gl_draw("Human Body Deformer", 282.0f, 15.0f);
    glEnable(GL_DEPTH_TEST);

    glPopMatrix();
    glMatrixMode(GL_MODELVIEW);
```

```
        glPopMatrix();
        glPopAttrib();

}

int DeformView::handle(int event)
{
  switch(event)
  {
          case FL_PUSH:
                /*
                ... mouse down event ...
                ... position in Fl::event_x() and Fl::event_y()
                */

          case FL_DRAG:
                /*
                ... mouse moved while down event ...
                */

          case FL_RELEASE:
                /*
                ... mouse up event ...
                */
                //cout << "FL mouse event" << endl;
                return handle_mouse(event,Fl::event_button(), Fl::event_x(),Fl::event_y());

          case FL_FOCUS :
                damage(1);
                redraw();
        return 1;



          case FL_UNFOCUS :
                /*
                  ... Return 1 if you want keyboard events, 0 otherwise
                */
                damage(1);
                redraw();
        return 1;
          case FL_KEYBOARD:
                /*
                  ... keypress, key is in Fl::event_key(), ascii in Fl::event_text()
                ... Return 1 if you understand/use the keyboard event, 0 otherwise...
                */
                //cout << "FL keyboard event" << endl;
                return handle_key(event,Fl::event_key());
          case FL_SHORTCUT:
                /*
                  ... shortcut, key is in Fl::event_key(), ascii in Fl::event_text()
                ... Return 1 if you understand/use the shortcut event, 0 otherwise...
                */
                return 1;
          default:
                // pass other events to the base class...
                return Fl_Gl_Window::handle(event);
  }
}

int DeformView::handle_mouse(int event, int button, int x, int y) {

    if (foo) delete [] foo;

    foo = new char[100];

    int ret = 0;

    switch ( button )
        {
            case 1: // LMB
                ret = 1;
                        // Based on the action, print the action and
            // coordinates where it occurred.
            if ( event == FL_PUSH )
                        {
                    sprintf(foo,"LMB PUSH ( %d , %d )",x,y);
                //cout << foo << endl;
                                old_x = x;
                                old_y = y;
                damage(1);
                                redraw();
            }
            else if (event == FL_DRAG )
                        {
                sprintf(foo,"LMB Drag ( %d , %d )",x,y);
                //cout << foo << endl;
                                mainCam.roll((old_x - x), (y - old_y));
                damage(1);
                                redraw();
            }
            else if ( event == FL_RELEASE )
                        {
                sprintf(foo,"LMB Release ( %d , %d )",x,y);
                //cout << foo << endl;
                                mainCam.update();
                damage(1);
                                redraw();
            }
            break;

        case 2: // MMB
            ret = 1;
                        // Based on the action, print the action and
            // coordinates where it occurred.
            if ( event == FL_PUSH )
                        {
                                sprintf(foo,"MMB Push ( %d , %d )",x,y);
                //cout << foo << endl;
                                old_x = x;
                                old_y = y;
                damage(1);
                                redraw();
            }
            else if (event == FL_DRAG )
                        {
                sprintf(foo,"MMB Drag ( %d , %d )",x,y);
                //cout << foo << endl;
                                double dolly_y = y - old_y;
                                double dolly_x = old_x - x;
                                if(fabs(dolly_y) >= fabs(dolly_x))
                                        mainCam.dolly(dolly_y);
                                else
                                        mainCam.dolly(dolly_x);
                                damage(1);
                                redraw();
            }
            else if ( event == FL_RELEASE )
                        {
                sprintf(foo,"MMB Release ( %d , %d )",x,y);
                //cout << foo << endl;
                                mainCam.update();
                damage(1);
                                redraw();
            }
            break;

        case 3: // RMB
            ret = 1;
                        // Based on the action, print the action and
            // coordinates where it occurred.
            if ( event == FL_PUSH )
                        {
                 sprintf(foo,"RMB Push ( %d , %d )",x,y);
                 //cout << foo << endl;
                                 old_x = x;
                                 old_y = y;
                 damage(1);
                                 redraw();
            }
            else if (event == FL_DRAG )
                        {
                 sprintf(foo,"RMB Drag ( %d , %d )",x,y);
                 //cout << foo << endl;
                                 mainCam.pan((old_x - x), (y - old_y));
                 damage(1);
                                 redraw();
            }
            else if ( event == FL_RELEASE )
                        {
                 sprintf(foo,"RMB Release ( %d , %d )",x,y);
                 //cout << foo << endl;
```

```
                                mainCam.update();
                damage(1);
                                redraw();
            }
            break;

    }

    return ret;
}

int DeformView::handle_key(int event, int key)
{
        int i;
        switch (key)
        {
                case '1':
                        for(i = 0; i < mainModel_male.numOfObjects; i++)
                        {
                                mainModel_male.pObject[i]->g_smothness = 1;
                        }
                        //cout << "key 1 pushed" << endl;
                        redraw();
                        return 1;
                case '2':
                        for(i = 0; i < mainModel_male.numOfObjects; i++)
                        {
                                mainModel_male.pObject[i]->g_smothness = 2;
                        }
                        //cout << "key 2 pushed" << endl;
                        redraw();
                        return 1;
                case '3':
                        for(i = 0; i < mainModel_male.numOfObjects; i++)
                        {
                                mainModel_male.pObject[i]->showNormal =
                                !mainModel_male.pObject[i]->showNormal;
                        }
                        //cout << "key 3 pushed" << endl;
                        redraw();
                        return 1;
                case '4':
                        for(i = 0; i < mainModel_male.numOfObjects; i++)
                        {
                                mainModel_male.pObject[i]->g_ViewMode = GL_LINE;
                        }
                        //cout << "key 4 pushed" << endl;
                        redraw();
                        return 1;
                case '5':
                        for(i = 0; i < mainModel_male.numOfObjects; i++)
                        {
                                mainModel_male.pObject[i]->g_ViewMode = GL_FILL;
                        }
                        //cout << "key 5 pushed" << endl;
                        redraw();
                        return 1;
                default:
                        damage(1);
            //printf("nothing to do\n");
            return 1;
     }
}
```

VITA

Xiao Zhang
Texas A&M University
A216 Langford Center
3137 TAMU
College Station, TX 77843-3137
xiao@viz.tamu.edu

Education

M.S. in Visualization Science	Texas A&M University, August 2005
B.E. in Engineering Physics	Tsinghua University, P.R.CHINA, May 1998