PARAMETRIC HUMAN BODY MODELING FOR VIRTUAL DRESSING

by

Başar Uğur

Bachelor of Science, in Computer Engineering, Boğaziçi University, 2005

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Master of Science

Graduate Program in Computer Engineering
Boğaziçi University

2008

PARAMETRIC HUMAN BODY MODELING FOR VIRTUAL DRESSING

APPROVED BY:

Dr. Ali Vahit Şahiner (Thesis Supervisor)	.....
Prof. Lale Akarun	.....
Assist. Prof. Burak Acar	.....

DATE OF APPROVAL: 18.06.2008

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my thesis supervisor Dr. Ali Vahit Şahiner, for his support throughout this study with not only the subject itself but also many other common interests of ours.

I would like to thank TÜBİTAK for their financial support during the first two years of my study.

I am also thankful for Ayşegül Başar, who was the initiator of the virtual dress project.

I can not forget the support of my family, with their existence and pure love. And also my band mates from Sakareller and colleagues who encouraged and enlightened me on or off the subject with their always-precious ideas.

ABSTRACT

PARAMETRIC HUMAN BODY MODELING FOR VIRTUAL DRESSING

This thesis presents a parameterized 3D human body modeling tool based on shape interpolation. In order to have local control on shape interpolation, human body shape descriptions are anthropometrically segmented. Smooth continuity at the boundaries of individual segments is achieved by weighted interpolation.

A 3D editing tool has been developed for segmentation. This tool employs winged edge data structure. This structure allows fast access to vertex neighborhoods. The interpolation weights for vertices are assigned during segmentation. We use decaying functions to assign weights on boundary regions, enabling smooth continuity.

Raytracing accelerated by octrees is used for synthesizing images of the resulting models. Anti-aliasing in these images is achieved by super-sampling with jittered patterns. The thesis also introduces a virtual dressing tool. This tool contains a 3D modeling interface and a dressing environment combined with a compositing subsystem, which creates dressed images of virtual replicas by utilizing layers of photographic images of garments and layers of synthesized images of body replicas.

ÖZET

SANAL GİYDİRME ORTAMI İÇİN PARAMETRİK İNSAN BEDENİ MODELLEME

Bu tez şekil interpolasyonuna dayalı üç boyutlu parametrik bir insan bedeni modelleme gereci ortaya koymaktadır. İnsan bedeni modellemede günümüze kadar uygulanmış yöntemlerin bir sınıflandırması verilmektedir. Şekil interpolasyonunda yerel kontrolu sağlamak için, insan bedeni şekil tanımları antropometrik olarak bölümlenmiştir. Bölümler arası sınır bölgelerde pürüzsüz süreklilik, ağırlıklı bir interpolasyonla sağlanmıştır.

Üç boyutlu model üzerinde bölümlerin imlenmesi için başka bir gereç geliştirilmiştir. Bu gereç her model için kanatlı kenar veri yapısı oluşturmaktadır. Bu yapı sayesinde nokta komşulukları bulunabilmektedir. Noktaların interpolasyon ağırlıkları da bu aşamada atanmaktadır. Sınır bölgelerde pürüzsüz sürekliliği sağlamak üzere, ağırlık atamaları için sönümlenen işlevler kullanılmaktadır.

Sonuç modellerden görüntü sentezlemek için, octree'lerle hızlandırılmış ışın izleme yöntemi uygulanmaktadır. Örtüşme-önleme için kararsız desenli ileri örnekleme kullanılmaktadır. Uygulama olarak bir sanal giydirme gereci geliştirilmiştir. Bu gereç, üç boyutlu bir modelleme arayüzü ve giysilerin fotoğrafik görüntüleri ile bedenlerin sentezlenen görüntülerini kullanan bileşim altyapısıyla sanal bedenlerin giyinmiş görüntülerini oluşturan bir giydirme ortamı içermektedir.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

2D	Two Dimensional
3D	Three Dimensional
PCA	Principal Component Analysis, a vector space transform often used to reduce multidimensional data sets to lower dimensions for analysis
FFD	Free-form Deformation
IDT	Intensity Difference Threshold
OpenGL	Open Graphics Library, a standard specification defining a cross-language cross-platform API for writing applications that produce 2D and 3D computer graphics
GUI	Graphical User Interface
CAESAR	Civilian American and European Surface Anthropometry Resource Project
JPEG	Compression method and type of files using that method, named after the *Joint Photographic Experts Group*, the name of the committee that created the standard
PNG	Portable Network Graphics, a bitmap image format that employs lossless data compression
TGA	Truevision (Advanced Raster) Graphics Adapter, a raster graphics file format
TIFF	Tagged Image File Format, a file format for storing images, including photographs and line art
VRML	Virtual Reality Modeling Language, a standard file format for representing 3D interactive vector graphics.
S, T	Morph targets
M_i	i^{th} submesh
$\mathbf{v}_i$	i^{th} vertex
u	Amount of interpolation / morphing

w	Weight of a vertex
$B(u)$	An in-between shape
R	Roughness measure
d	Depth of a vertex in a propagation
p	Depth of propagation
P_i	i^{th} point in algorithmic examples
b_i	A barycentric coordinate
I	Intensity of a light source
K	Effective constant for a light source
$\mathbf{N}$	Surface normal vector
$\mathbf{L}$	Light vector
σ	Variance in a Gaussian function
μ	A mapping between two meshes

1. INTRODUCTION

3D human body models are used in a wide spectrum of applications that require images of human replicas. Their use in film and entertainment industry and on internet is very common. The methods used for creating and interacting with body models have also evolved. From real-time modeling to volumetric rendering, a wide range of applications exist.

Body models are employed in visualization and animation environments, for giving an interactive experience to the users. They can be used for creating personalized replicas (or *human avatars*) in virtual worlds.

3DTV programs which involve moving bodies and talking heads such as sports competitions or news, employ body models. Virtual reality environments containing person - object interactions also utilize body models. Internet applications such as mass multiplayed games and news & advertisement videos contain personalized models. Many medical applications make use of detailed graphical actualization of human bodies.

Human body modeling is the act of creating a shape description of a specific human body. The shape description is usually a geometrical representation in the 3D computer environment. It is combined with skin coloring and texture mapping to give more realistic results. Skeleton based structures can further be added to the model for applying kinematics and giving motion.

Body can be modeled at different 3D description levels. *Surface modeling* is based on creating the body using layers of polygonal meshes. Only the outer skin, or several internal layers may be created for the purpose of the application. However *volumetric modeling* is based on creating the body using scanned volume slices. They are largely employed in medical applications.

In this thesis, surface modeling is used. Surface based models of human body can be constructed at different levels of detail. Posture modification is made possible by attaching a body skeleton during construction. Skin detail can be realized by material coloring, using body-wide texture mapping or modifying small-scale geometry. Figure 1.1 shows an example from the CAESAR data-set, where high level details of the body geometry is depicted.

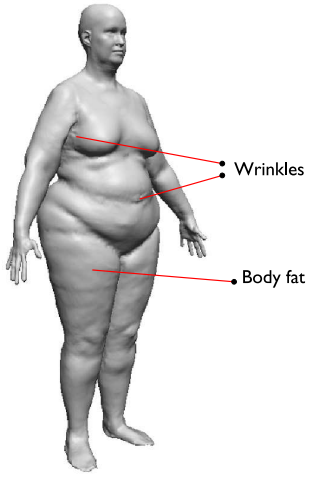


Figure 1.1. A 3D human body with unique properties such as fatty parts and different skin geometry.[1]

This thesis presents a parametric human body modeler based on shape interpolation. Shape interpolation is based on the idea of creating new shapes by blending existing shapes. It is simple and stable. The term 3D morphing is widely used for the specific case where two pre-constructed and properly mapped polygonal meshes (a *morph-source* and a *morph-target*) are used to create in-between shapes.

Morphing causes the source shape to be transformed globally as a whole. Segmentation of both the source and the target shapes are required if local control is desired. An anthropometry based segmentation for the human body shape is implemented in this thesis. The smooth continuity at the boundaries of individual segments is achieved by weighted interpolation.

Body segments are marked by a mesh editing tool. This tool keeps a winged edge data structure for each model, enabling fast determination of vertex neighborhoods.

Vertex neighborhoods are employed in vertex set propagations, defining the boundary regions of segments. Decreasing Gaussian functions are used for weight assignments during propagation, to obtain smooth shapes at boundary regions.

Interpolation based body modeler is presented in a virtual dressing room application with an easy to use graphical interface. 2D image of the resulting 3D model is synthesized by raytracing accelerated by octrees. Anti-aliasing problem is solved by super-sampling with jittered patterns.

Synthesized images are further personalized by utilizing facial photographs. Resulting images are made available in a dressing environment. Multilayer image compositing is employed using these images and photographic images of garments.

1.1. Thesis Organization

The organization of the thesis is as follows: Chapter 2 gives an overview of human body modeling methods applied thus far. Chapter 3 is a detailed discussion on 3D shape interpolation techniques. Chapter 4 gives the detailed explanation of the interpolation based body modeler. Image synthesis method is explained in Chapter 5. Chapter 6 contains the working structure of Virtual Dressing Room application. Chapter 7 contains the evaluation of the methods used. Thesis ends with conclusions and planned future work.

2. PREVIOUS WORK

In the field of human body modeling, we can classify the existing techniques by dividing into three categories: *Sculpting*, *Automatic* and *Parametric* techniques. Sculpting techniques depend entirely on a creative perception of the body. Automatic techniques are based on data obtained by different sources such as scanning, still images and image sequences. Parametric techniques use standard parameterizations of the human body and construct new bodies from existing ones using these parameters.

2.1. 3D Sculpting Techniques

3D sculpting is usually done by commercial modeling programs. This type of modeling requires an artistic conception of the human body and it is about creating the body gradually considering the detailed anatomical knowledge [4][5]. In Allen *et al.*'s work based on constructive techniques, body is separated into three main parts, in the order of forming: *The skeleton*, which consists of the bones and joints defining the body articulation. *The musculature*, containing all the muscles with either movable or non-movable parts. *The fat layer*, the fatty tissues under the skin, giving the body its final outer appearance. Although these layers are polygon meshes and not formed by volumetric data, the resulting models are realistic in the anatomical sense.

In these techniques, the user, usually a modeling artist, creates the model using the tools for geometry modification. Typically one starts with a simple geometrical object and modifies it gradually to come up with a human body part, as shown in Figure 2.1.

Nevertheless, sculpting method requires exhaustive user intervention for creating a reasonably *new* human body from scratch. Especially if building an anatomically correct and realistic human model is required, 3D sculpting is not adequate, since it is hard to model details such as face wrinkles and fatty body parts using this method.(see Figure 1.1)

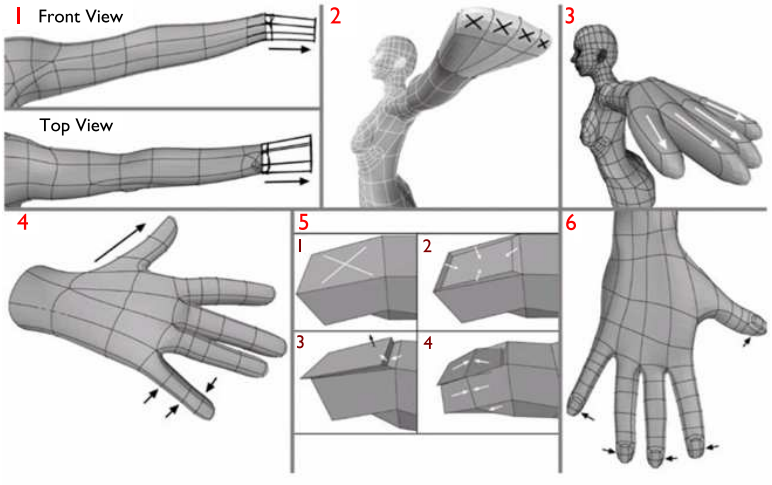


Figure 2.1. An example of 3D sculpting: Modeling the hands

2.2. Automatic Techniques

These approaches employ data extracted directly from real humans. Data extraction is done using 3D scanners[1][3], 2D photographs[2][6] and image sequences[7][8], especially when body postures are concerned. 3D scanning is extensively used in Civilian American and European Surface Anthropometry Resource Project (namely *CAESAR*) data set[9].

CAESAR is a partnership between government and associated industries to collect the most extensive sampling of consumer body measurements for comparison, modeling and such. The project collected data on 2,400 United States & Canadian and 2,000 European civilians and a database was developed. Each member of this database contains 40 traditional anthropometric measurements that were done with a tape measure and caliper. These values are important for sophisticated design and implementation patterns, as the whole set of measurements could be technically considered. For example, early anthropometric modeling systems[10] focus on driver-seat or cockpit design, and measurements are important for testing and determining the most suitable design for different bodies with different metrics.

The reconstruction techniques which exploit 2D photographs could be taken into account as more *personalized* implementations and creating -in a way- visually more

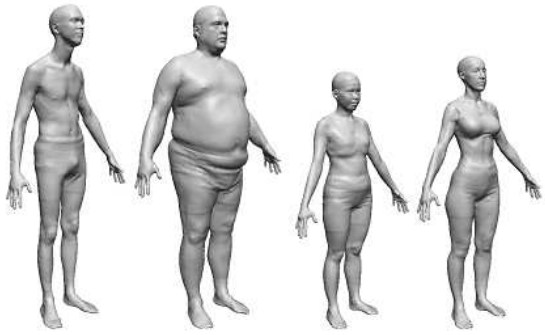


Figure 2.2. Samples from CAESAR data set[1]

acceptable models faster. But somehow they do not allow much user interaction with the resulting models as how different parameterizations of those would yield.

Figure 2.3. Image-based shape capture by Lee *et al.* [2]

2.3. Parametric Techniques

2.3.1. Reconstructive

Reconstruction from source data and the parametrization of the resulting models simplify the creation of new models. Furthermore, texture transferring and inter-model morphing implementations are an advantage for the realistic properties of the

output models, as depicted in the relevant studies. However, though the preprocessing part reduces the overall amount of work, these approaches generally require storing of some considerable number of the dataset geometry, and make use of morphing the coordinates in the relevant space in Principal Component Analysis (*PCA*) applied models[3][11]. For instance, anthropometric segment interpolation is not applied directly. Instead, additional work is needed to map segment interpolation controls and the resulting PCA weights. These techniques are well applied by Thalmann *et al.* [3][12]. Their implementation starts with the mapping of a template mesh to one specific body. PCA is then applied to the mapped models and certain features are determined by analysis. Features can be controlled in the parameter space by several weights. Supplying the anthropometric measurements for each body model -given in the dataset-, interpolation nodes could be determined. As a result, it becomes possible to form new models by entering a few sizing parameters.

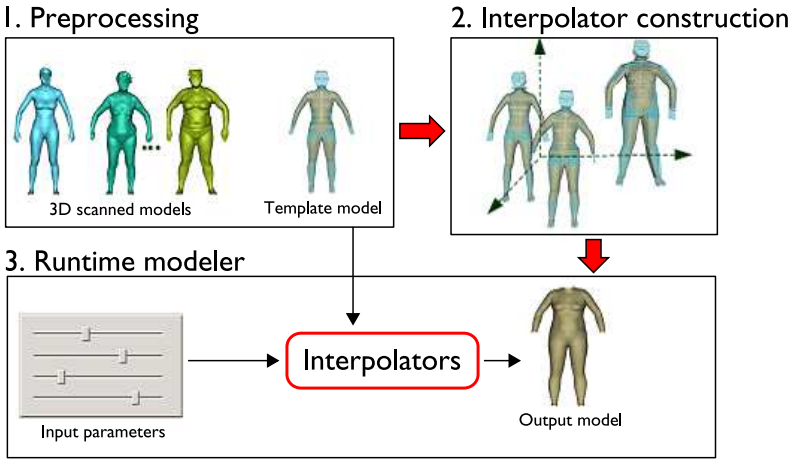


Figure 2.4. Data compression based method of Thalmann *et al.* [3]

2.3.2. Deformation Based

Deforming an object or transforming one object into another is a visually powerful design and animation technique. It is applied to human body modeling by an initial parameterization of the body. Here, the user has only one base model and creates new models by deforming the marked segments. Thalmann and Kasap[13] present a detailed segmentation based deformation. Their work additionally focuses on *regional*

deformation, a concept similar to *local morphing*.

Free-form Deformation

The main 3D object deformation method is *Free-form Deformation* or *FFD*. Inspired by Barr's work [14] and later developed by Sederberg and Parry [15], this method enables modifying any part of a geometry with an outlying set of control points. The interaction is very much like using the control points of a spline, but there are many variations on how to use the 3D control point set. A simple example can be seen in Figure 2.5 where the upper part of a sphere is controlled by a simple box. As the box is being deformed by a y-axis scaling, the points inside the box are deformed according to their relative coordinates.

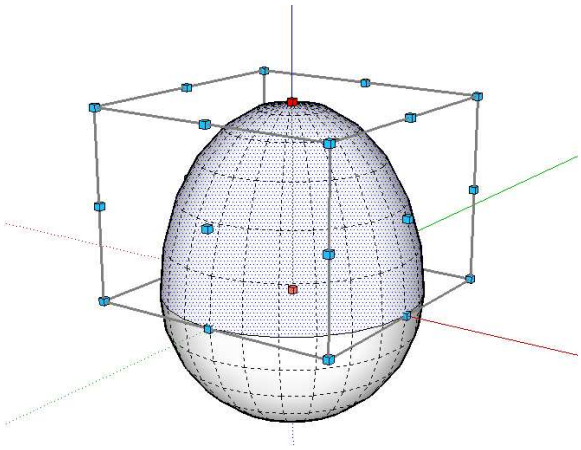


Figure 2.5. An example of free-form deformation

Deformation based methods are also used in sculpting, where the user deforms the submeshes of the segmented editable mesh. The parameterization may constrain the amount of deformation, defining minimum and maximum values. In these cases, deformation becomes similar to interpolation.

2.3.3. Interpolation Based

In this method there may be several target body models and new bodies are created by interpolating among them, either by parts or as a whole. Parameters may be directly connected to the geometry, such as the amount of morphing for a submesh

in a mesh segmentation. Or they may have indirect effects on the geometry, such as the effect of body fat amount on the shape of hip or effect of body weight on the whole geometry.

Deformation and interpolation based parametric techniques are tightly coupled with 3D shape interpolation, which is considered a separate field of study. Hence we examine and discuss the several aspects of the problem in the next chapter.

3. 3D SHAPE INTERPOLATION

Before moving on to the detailed explanation of our own method, we should also construct a background for the shape interpolation methods implemented thus far. Different terms are used for this field besides *shape interpolation*, such as *3D metamorphosis* or *mesh morphing* using morph targets. The main idea is creating new intermediate objects or obtain a smooth visual process, by using several target models.

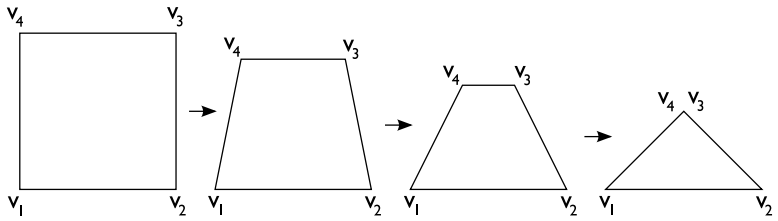


Figure 3.1. A shape interpolation example in 2D, a square is turned into a triangle.

The problem is generally separated into three phases. First, the determination of targets. How do we specify our reference points in the interpolating environment? Second, the correspondence (or *matching*) of the core geometric elements of these targets, usually vertices. Third, the interpolation method itself. That is, how the targets are transformed into each other and what further work is done on the resulting intermediate models.

3.1. Determination of Targets

We can also identify the problem as the "object space" determination problem. We divide the approaches in this area into three sections, as how the target models are obtained.

3.1.1. Manual methods

As previously mentioned, manual creation / 3D sculpting is a method of forming new bodies, which is also the case for any 3D object. Examined in many independent or commercial package based tutorials and books, and with high details, one can

produce satisfactory results for obtaining a 3D model as a starting point. From there on, concerning the morphing phase, several other target models can be obtained by preserving the mesh topology for both the whole model and certain areas. That is, keeping the same vertex-edge graph not only for the whole model, but also for the subsets of it; such as the same body areas for different featured models.

3.1.2. Data compression based methods

These methods are based on defining and recording certain features of scanned and matched (i.e. correspondence problem solved) 3D data, and applying data compression / pattern recognition techniques on those. The target models are obtained by locating certain featured models at some coordinate in the multi-dimensional space. Allen *et al.*'s work focuses on creating different-posed models [16][1]. Blanz and Vetter's face based personalization method[17] is also another inspiring work in this field. They use the laser scans of 200 heads of young adults, containing geometric and textural data. Using PCA again as a data compression method, they extract certain features such as gender, hooked nose and weight. Constructing a parametric face model that can easily generate any new face, the *morphing* problem of the example-based technique (*correspondence*, as they call it) becomes a mathematical optimization problem. Here, one should notice the differences of face and body modeling. Face, as a geometric object, has been more important an area of study about human recognition than body. Additionally, body features do not project onto an efficient space as faces do. One should give credit to the similar work done for bodies (as previously cited) but should also notice the difference in resulting quality.

3.1.3. Scan-and-match based methods

Using the state-of-the-art scanning technologies, many 3D datasets are formed for the use of different application areas. Human body scan is very common too; as previously mentioned, CAESAR is an important example. Scan-and-match method is based on defining a *template mesh* initially. The scanned models are then mapped onto this template mesh. The mapping algorithm consists of two parts: *Skeleton fitting*

is used for finding the pose of the scanned data. Template model's skeleton is fit to the pose of the body. *Skin refinement* is iteratively improving the fitting accuracy by minimizing the shape difference between the template and the scan model.

3.2. Correspondence between Targets

The problem is finding a mapping between the boundaries of the source and target shape. This is either achieved by constructing new models from the same base model without changing the mesh topology, or by finding a mapping between two arbitrary meshes.

Mapping methods are introduced early by Kent *et al.* [18] and later developed by Eck *et al.* [19]. Mapping is constructed on a unit sphere or disk -respectively in cited studies-, creating a correspondence between the Genus-0 polyhedra (a mesh without any holes), and finally merging their vertex-neighborhood graphs. General polyhedra are dissected into topological disks and each part of the mesh is mapped to a disk using a parameterization such as the original faces of the meshes [20] or harmonic embeddings [21]. Kanai *et al.*'s harmonic embeddings method is shown in Figure 3.2 as an example.

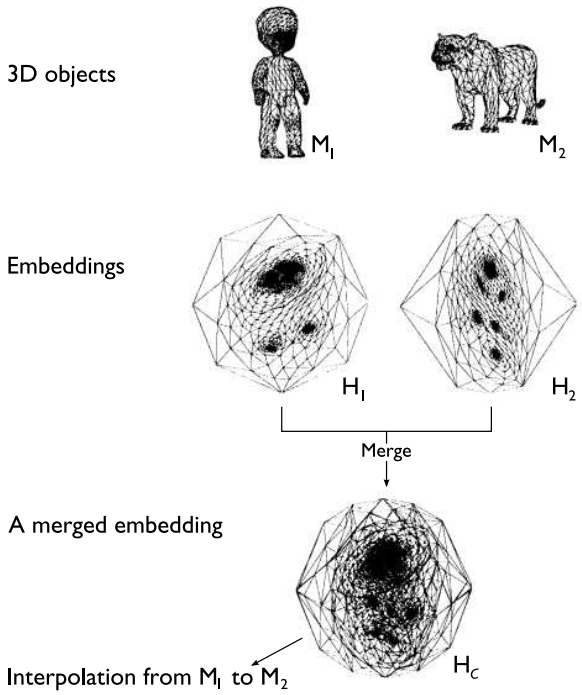


Figure 3.2. Harmonic mapping method by Kanai *et al.* [21]

3.3. Interpolation

Metamorphosis techniques are used in widespread applications and are commonly employed by the graphics industry. Image morphing has become a standard tool, because it is stable, easy to use and allows a lot of flexibility defining the actual morph sequence. Changing one 3D object into another is also a useful effect, but one with problems for which general-purpose solutions are still being developed. The main idea is to obtain smooth intermediate models or create a smooth interpolation among target models. (see Figure 3.1). Our main interest in this field is local morphing or segmented mesh morphing as we divide the body models into segments and try to have visually adequate outcoming metamorphosis.

Of our interest, one classification for morphing is *global* and *local*. This separation is different from that of curve control. In curve control points, global control means that moving one control point changes the entire curve; however distant sections may change only slightly. And local control means that moving one control point only changes the curve over a finite bounded region. In morphing terms, global morphing (or global surface interpolation) is the transformation of *all* the vertices contained in one mesh -usually, to the vertices of another-, by the specified interpolation function. Local morphing is the transformation of a *subset* of mesh vertices. Usually that subset is composed of neighboring vertices and edges, which form a meaningful *segment* of the the mesh.

An example can be seen in Figure 3.3. The left model is the base with normal parameters, with an overall dress sizing of 38. The middle one is obtained by a global morphing, controlled by the sizing parameter -which is 46 in this case. The right one is obtained by local morphing, where legs are made shorter, and hips are enlarged by local sizing parameters.

An early work by Barr [14] focuses on the spatial transformation in a hierarchical object space. He introduces globally and locally defined deformations as new hierarchical operations for use in solid modeling. These operations extend the conventional

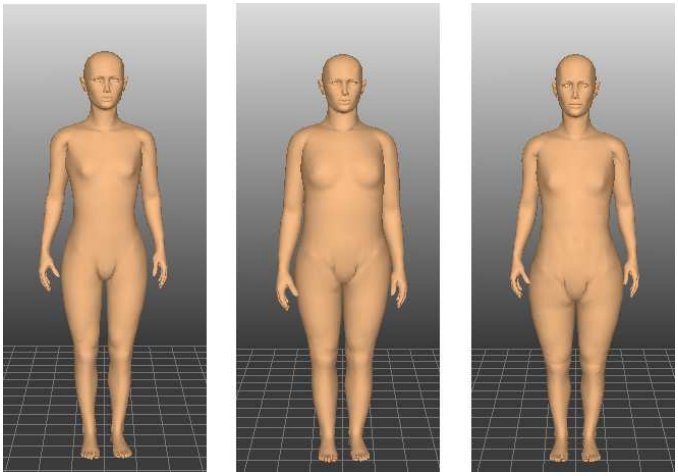


Figure 3.3. Different shaped bodies obtained by global and local interpolation.

operations of rotation, translation, Boolean union, intersection and difference. Deformations allow the user to treat a solid as if it were constructed from a special type of topological putty or clay, which may be bent, twisted, tapered, compressed, expanded, and otherwise transformed repeatedly into a final shape. They are highly intuitive and easily visualized operations which simulate some important manufacturing processes for fabricating objects, such as the bending of bar stock and sheet metal. (Figure 3.4)

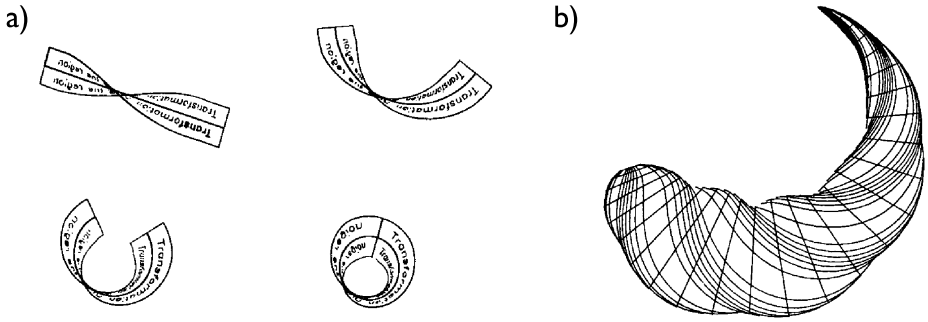


Figure 3.4. Spatial Transformation method: a) Moebius Band is produced with a twist and a bend, b) a bent, twisted, tapered solid 3D primitive [14]

Alexa's related work focuses on local controls and submesh morphing. In a 2001 work[22], he uses Laplacian coordinates to represent the subset of mesh vertices that is the region of interest. Given the correspondence with the target object, Laplacian representation enables independence from transformations of the shape and tolerate degeneracies in the mesh. However, it is *not insensitive* to scaling or rotating. Yet the *Transition State* concept is applied, which is similar to our implementation. That is, defining weights per vertex and then using these weights during morphing. Scaling and

rotation adjusted local morphing is achieved by the mesh fusion method of Kanai *et al.* Their work is also a mapping-based morphing technique. However they utilize the algorithm with harmonic mappings [21] and later develop the algorithm by introducing mesh fusion with Fusion Control Functions (*FCF*'s)[23]. This method parameterizes the smoothness at boundary regions and enables the control of the outcome polyhedron. Transformation adjustments are determined by a rigid transformation and a deformation step in these methods.

4. INTERPOLATION BASED BODY MODELER

In our body modeling tool, we have employed the interpolation based approach, calibrated target models and manually segmented these models utilizing an editing tool. We first explain our approach for constructing new models using the base measurement system. Then we introduce a body segmentation approach for creating target submeshes used in local morphing and explain our calibration methods for the measurement constraints. We finalize this part with the explanation of our local morphing method built upon the work done in the first two subsections.

4.1. Constructing Target Models

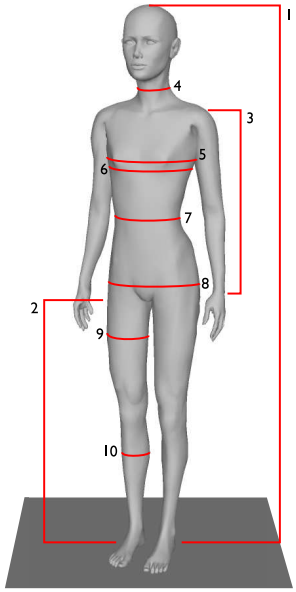
Human body is divided into parts to study in a detailed manner at every branch of science. Even at elementary school, the body is first taught to us with the rough division of "head - torso - arms & legs". Then as both industry and science branches into many different areas, body is examined by dividing into special parts according to the necessities of these particular areas. One of the many divisions is the standards based upon *anthropometry*.

Anthropometry began as the science of human body measurements at the beginning of the 20th century. The name was given by Alphonse Bertillon in 1883, who claimed that when these measurements were made and recorded systematically, every single individual would be found to be perfectly distinguishable from others. The very starting point of Bertillon, matches our final purpose in this work. Today, there exist international standards based upon anthropometry, such as ISO 7250 - *Basic human body measurements for technological design*, or EN 13402 - *European standard for labeling clothes sizes*. Mainly, according to these standards, body is measured by linear lengths, horizontal girths and body mass.

Our choice of anthropometric parameters is simply an extension of EN 13402 (see Table 4.1), depending on its simplicity for being definitive of human body geometry and

Table 4.1. Anthropometric measurements of the human body

	Body Measurement	**Definition**
1	Stature	Vertical distance between the top of the head and the bottom of the heel bone.
2	Leg Length	Vertical distance between the crotch level and the bottom of the heel bone
3	Arm Length	Distance from the armscye shoulder line intersection over the elbow to the far end of the wrist bone.
4	Neck Girth	Girth of the neck base
5	Chest / Bust Girth	Convex hull circumference at chest / bust height
6	Under-bust Girth	Horizontal girth at the height just below the breasts
7	Waist Girth	Horizontal girth at waist height.
8	Hip Girth	Horizontal girth at largest convex hull at hip area.
9	Upper Leg / Thigh Girth	Horizontal girth at the largest convex hull at the upper leg.
10	Lower Leg / Lower Limb Girth	Horizontal girth at the largest convex hull at the lower leg.



the final aim of that work being a dressing simulation. For a customizable body model, the selection of control parameters should be subject oriented. If the task is creating a cartoon environment, reality is not much of an aspect; therefore anthropometric standards may be discarded. In this work, the directly-usable lengths and girths - geometric values included in the standard- are taken into account. Body mass is not used due to its indirect effect to the overall geometry. However, there are some examples in literature, e.g. Thalmann *et al.*'s work enables usage of body fat as a sizing parameter -i.e. a vector in downsampled space- and can generate new models accordingly.

4.1.1. Skeletal lengths

Defining the vertical stretch of both the overall model and its subsets, there are three main skeletal lengths defining the body. These are body height, leg length and arm length. The measurement of these parts differs according to the application. Body height is intuitively measured as the distance between the top and the bottom points

in the erect pose of the model. Leg length is defined as the distance between the crotch and body bottom. And arm length is measured from the arm top which is located on the joint between the shoulder and the arm, down to the wrist.

Applying a certain length is achieved by a vertical scaling that preserves the other unchanged lengths. However, changing a body length displaces the vertices and this forms a problem on the boundary regions. Therefore, the solution must both satisfy the required length and segment connectivity.

Body height: New body top is calculated according to the new height. All vertices except those of *legs* and *arms* are vertically scaled. Leg and arm vertices are then translated according to their topmost vertices' positions. *Hand* vertices are translated according to the new *wrist* position.

Leg length: Scaling is applied to whole leg. As *body height* is not changed, body vertices -except *arms*- that are not the members of the leg are also scaled to preserve the height. Arm and hand vertices follow the same translation.

Arm length: Arm top remains constant, scaling is applied to the arm vertices. Hand vertices are translated accordingly.

4.1.2. Outer body girths

These lengths are defined during the construction of the body models. However, girths make up the core part of the body measurement standards and one should refer to the standard metrics during the construction. We determine the body hulls at the predefined girth positions and calculate the actual girths of the 3D model at these positions. Then we compare these values to the standard metrics and calibrate the model by scaling horizontally at these positions.

4.2. Calibration

To calibrate the target models' girths according to the standard measurements (see Table 4.2 and 4.3), we use the convex hull circumferences at girth positions. For determining the convex hulls, a plane at the specific height for a girth is intersected with the lines contained in the girth area. Intersection points form the set of points whose convex hull will be determined.

Table 4.2. Dress measurement standards for the male body (in *cm*)

Size	**44**	**46**	**48**	**50**	**52**	**54**	**56**
Arm Length	63	63	64	64	64	64	65
Neck Girth	37	38	39	40	41	42	43
Bust Girth	88	92	96	100	104	108	112
Waist Girth	78	82	86	90	94	98	104

Table 4.3. Dress measurement standards for the female body (in *cm*)

Size	**34**	**36**	**38**	**40**	**42**	**44**	**46**	**48**	**50**	**52**
Arm Length	59	59	60	60	61	61	61	61	62	62
Neck Girth	34	35	36	37	38	39	40	41	42	43
Bust Girth	80	84	88	92	96	100	104	110	116	122
Under-bust Girth	65	70	75	80	85	90	95	100	105	110
Waist Girth	62	66	70	74	78	82	86	92	98	104
Hip Girth	86	90	94	98	102	106	110	116	122	128

The Graham scan algorithm [24] is often cited ([25], [26]) as the first "computational geometry" algorithm. As the size of the geometric problem (namely, n = the number of points in the set) increases, it achieves the optimal asymptotic efficiency of $O(n \log n)$ time.

The algorithm starts by picking a point in the finite set **S** known to be a vertex of the convex hull. We do this in $O(n)$ time by selecting the rightmost point in the

set; that is, a point with the maximum x coordinate. Having selected this base point, the algorithm then sorts the other points P in $\mathbf{S}$ by "left"ness, the increasing counter-clockwise (ccw) angle the line segment P_0P makes with the x-axis. One example of sorting is shown in Figure 4.1. The dashed red lines show the convex hull and the dashed gray lines show the intermediate hulls during iteration.

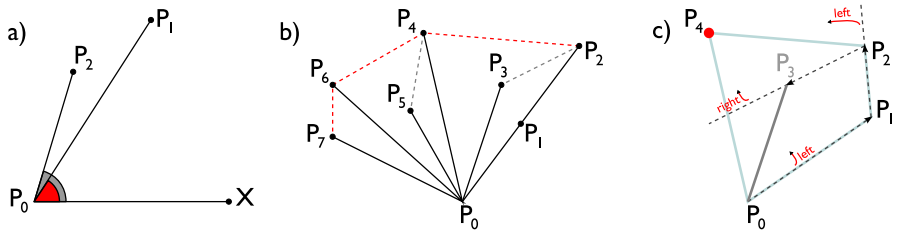


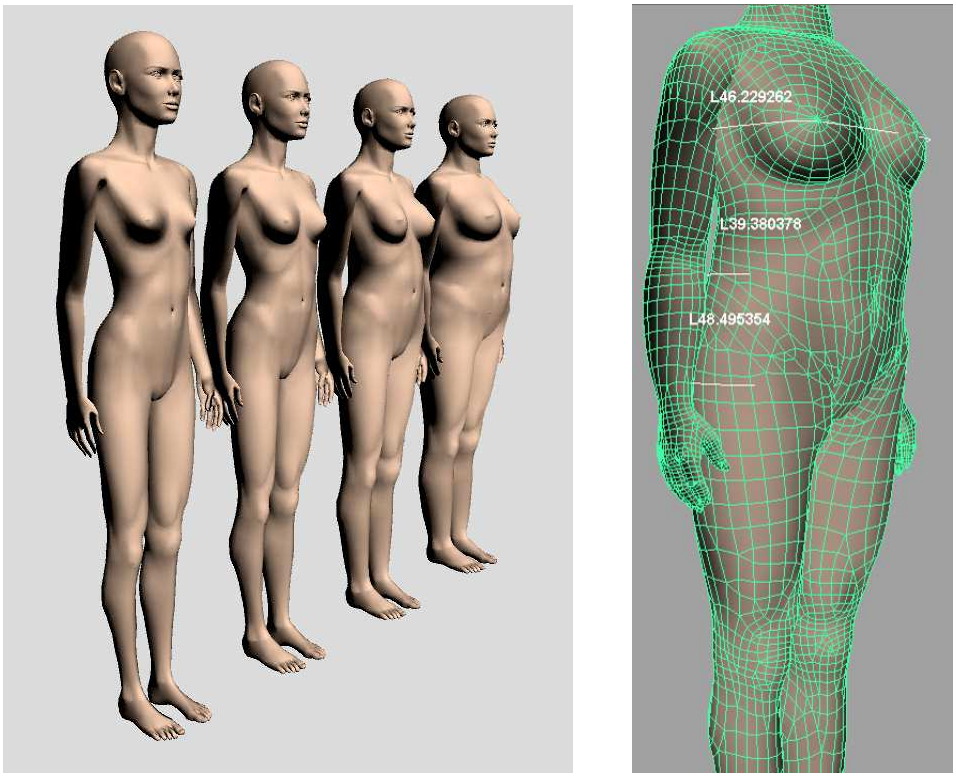
Figure 4.1. Graham-scan convex hull algorithm a) "left"ness check b) sorting (ccw) of a set of points c) example iteration step

Proceeding to find out the hull, rest of the algorithm is an inductive incremental procedure using a stack of points. At each stage, we save the vertex points for the convex hull of all points already processed. We start with P_0 and P_1 on the stack. Then at the k-th stage, we add the next point P_k, and compute how it alters the prior convex hull. If new inner points are formed, they are removed from the hull. One iteration step is depicted in Figure 4.1.c. Here, it is important to note that for each point of S there is one push and at most one pop operation, giving a maximum of $2n$ stack operations for the whole algorithm.

Finally we calculate the hull circumference by adding up the neighboring point distances. These circumferences are compared to the given standard measurements and the body model is horizontally scaled at the girth position according to the difference between the given and calculated measurements. Calibration process and resulting female models can be seen in Figure 4.3. Additionally, girths are made visible to help as *measurement guidelines* in the 3D environment (see Figure 4.4).

```
Input: a set of points S = {P = (P.x, P.y)}
Select the rightmost point P_0 in S.
Sort S angularly about P_0 as a center.
For ties, discard the closer points.
Let P[N] be the sorted array of points.
Push P[0]=P_0 and P[1] onto a stack W.
while i < N // number of points
{
    Let P_1 = the top point on W
    Let P_2 = the second top point on W
    if (P[i] is left of the line P_2 to P_1)
    {
        Push P[i] onto W
        i++
    }
    else
        Pop P_1 off the stack
}
Loop through W adding up the distance between consecutive elements
Output: the convex hull circumference
```

Figure 4.2. Pseudocode for convex hull circumference algorithm



(a) Female target models having dress size 34, 40, 46 and 52

(b) Calibration of target model sized 52

Figure 4.3. Target models and calibration

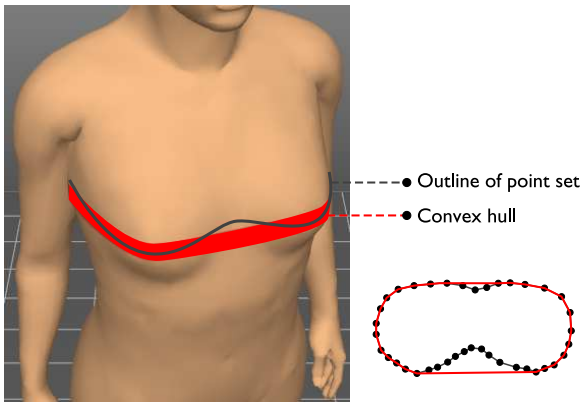


Figure 4.4. Visible girths in the 3D environment

4.3. Segmentation Tool

Having determined the measurements to use, our body segmentation approach becomes entirely based on the measurement areas. By measurement area, we mean a set of vertices covering a certain *length* for a region-of-interest; such as *neck* area for neck girth, *arm* area for arm length, or the *whole body* for body height. At this point, intersecting (not *containing*) or *neighboring* areas form a problem of how to morph into the corresponding ones in the target models. For instance, a discrete marking process followed by a discrete morphing results in jagged meshes.

For that purpose, we require smooth boundary regions. Making use of the winged edge polyhedron representation[27] and saving the geometry as a winged edge table, it becomes a simple task to produce a vertex propagation. Using neighboring relations saved in the data structure, it is possible to widen a set of selected vertices.

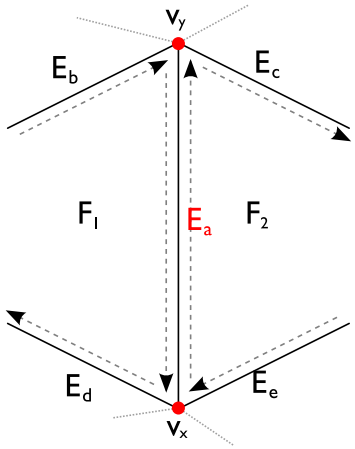


Figure 4.5. Winged edge polyhedron representation

To briefly explain the winged edge polyhedron representation, a simple figure (see 4.5) would be helpful. The representation is based on a table structure which contain information based only on the *edge*s of a geometry. We store this as an array of `Edge` class instances. The class contains the following information: An edge is defined by its start and end vertices, these are stored. It can be the edge of at most two faces in a mesh. These are called the *wing*s of the edge and they are stored as well. And finally, we store neighboring edges contained in the wings by a clockwise order; left and right wing's edges are stored as *predecessor* and *successor*.

Table 4.4. Edge-based data stored in winged edge structure.

Edge	Vertices		Faces		Left Traverse		Right Traverse	
E_a	Start	End	Left	Right	Pred	Succ	Pred	Succ
	$\mathbf{v}_x$	$\mathbf{v}_y$	F_1	F_2	E_b	E_d	E_e	E_c

We also keep two more tables for vertices and faces. These tables contain very simple information about these primitives. The vertex table has one entry for each vertex which contains an edge that is incident to this vertex. The face table has one entry for each face which contains an edge that is one of this face's boundary edges.

Table 4.5. Vertex and Face-related data stored in winged edge structure.

Vertex	Incident Edge
$\mathbf{v}_x$	a
$\mathbf{v}_y$	c

Face	Incident Edge
F_1	E_d
F_2	E_a

After storing the data structure in memory, the vertex and edge neighborhood queries can be answered easily. Assume we want to find a 1 neighborhood of a vertex $\mathbf{v}$. Starting from the *incident edge*, we traverse the neighboring edges by using the *predecessor* and *successor*s of the incident edge. If $\mathbf{v}$ is the start of its incident edge we add the end vertex to the stack, or vice versa. Again if $\mathbf{v}$ is the starter / ender, we move on to the *left traverse*'s successor / *right traverse*'s predecessor -which contains $\mathbf{v}$- and add the other vertex to the stack and so on. Iteration stops if the vertex to be pushed is already in the stack.

Assume a selected set of vertices $\mathbf{V}$, and we want to find a 1 neighborhood of this whole set. We start by assigning a membership value 1 to all members of the set. We find the 1 neighborhood stacks of each member, and merge them. This gives us a *propagation* of $\mathbf{V}$. Assume we add the propagation into $\mathbf{V}$ and operate on this new set, and so on. Consider each iteration stack as a new *propagation depth*. This helps us define the boundary regions of a selected submesh, using decaying membership values while propagating from the base set $\mathbf{V}$.

```
Input: Vertex set V, vertex v
Define stack W
E = Incident edge of v
REPEAT
{
  IF v is the start of I, v_pushed = the end v_e of E
  ELSE v_pushed = the start v_s of E

  IF v_pushed is in W terminate
  ELSE push v_pushed onto W

  IF v is the start of E, E = succ( left_trav(E) )
  ELSE E = pred( right_trav(E) )
}
```

Figure 4.6. Pseudocode for propagation algorithm

At this point, we use the depth of propagation and assign decreasing weights to vertices at a certain depth by Equation 4.1.

$$\forall\, \mathbf{v} \text{ at depth } d,\ w_{\mathbf{v}} = f(d, p) \tag{4.1}$$

Here, the function f determining the weight assigned to a vertex is defined by two parameters:

1. Depth of the vertex: Shortest *graph* distance to the selected set of base vertices.
2. Depth of the propagation: Maximum graph distance required.

It is assumed that the peak value of this formula is 1 when graph distance equals to 0. Several mathematical functions are suitable for that matter. The act of the function to morphing at boundary regions is similar to FCF's in Kanai *et al.*'s work.

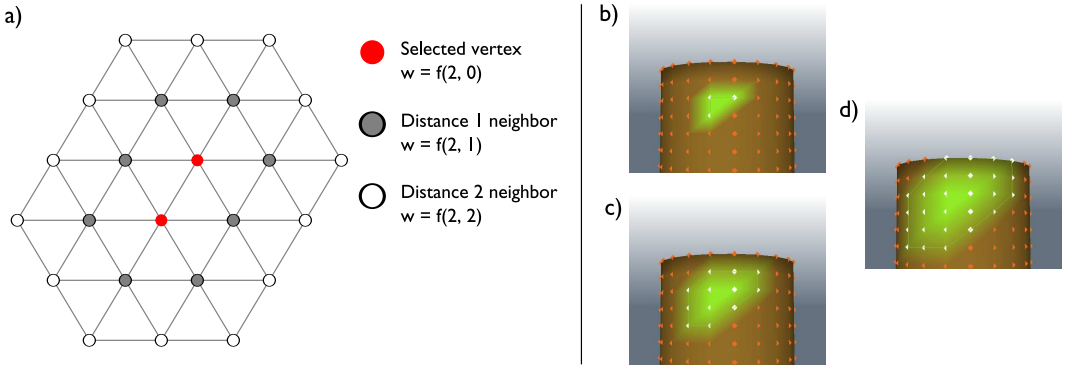


Figure 4.7. a) Weight functions according to vertex neighborhood. b), c) and d) a propagation on a 3D model.

Figure 4.8 shows the effect of different functions on morphing.

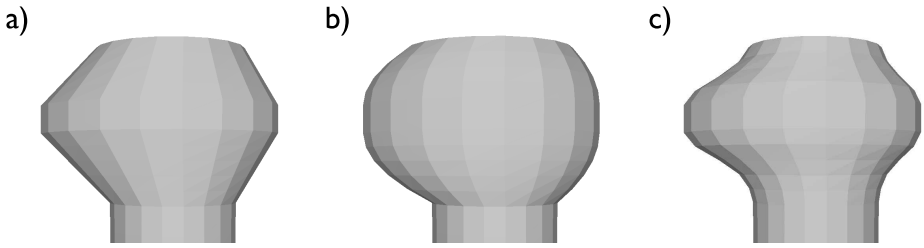


Figure 4.8. Effect of different weight functions on boundary regions a) Linear function $w = (p - d)/p$ b) Square function $w = (1 - d^2/p^2)$ c) Gaussian function $w = e^{-d^2/2\sigma^2}$ where $\sigma = p \cdot c$ and c is a constant

As we use middle density polyhedra in our work, human body is generally triangulated best as a cylinder-like object in such densities. In Lazarus and Verroust's work[28], morphing of cylinder-like objects is studied. Their work focuses on shape transformations which is mainly useful for animation purposes. However, the usage of axes and associated disks in that work is similar to our choice of body areas around measurement points.

We implemented a tool for marking these segments, and saving and loading them as VRML files. The tool takes 3D models as input and enables defining submeshes by picking the vertices. Vertices can be picked either in a per-vertex manner or by rectangular selection. After a *core* set of vertices are defined for a body segment, boundary regions can be defined by utilizing the vertex neighborhood and propagating

through the set of vertices.

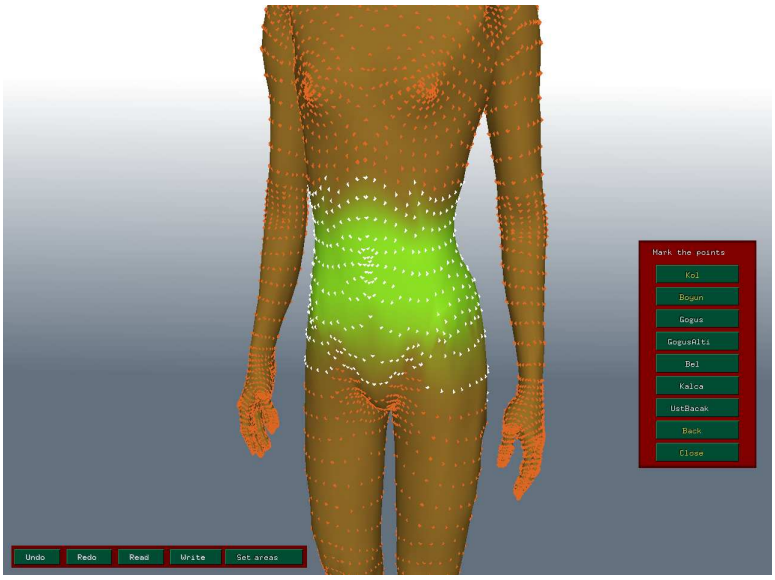


Figure 4.9. The Segmentation Tool enables marking of body areas and saving them as VRML files.

3D model marked with body segments are saved as VRML files, which is an open standard for 3D virtual environments. We save this segmentation related part in a comment section at the end of all geometry data inside the file. This way, the VRML file remains usable by other software. Starting with a keyword "*AREAS*", we write the number of elements in the first segment. Then for each member vertex, we write the index of the first vertex which is a member of the segment, followed by the amount of membership. We continue with the second segment data and so on. This structure is deployed into the interpolation environment and utilized accordingly.

```
 { ... geometry data ...}
# AREAS
# 181 360 0.063244 361 0.007369 362 0.00736345 ...
# 220 223 0.00743 224 0.0023362 225 0.0073696 ...
⋮
```

Segmentation data structure in VRML files.

Gaussian functions give the best visual results for marking boundary regions,

as body is modeled cylinder-like around morphing areas. Specifying our morphing weights, or *transition state*, for a particular body area; we examine the problem of morphing them in the next section.

4.4. Local Morphing

Given two sets of vectors S and T where each vector keeps the 3D coordinates of a source mesh vertex, an in-between shape $B(u)$ can be parametrically defined as a linear combination of s_i and t_j as follows:

$$\begin{aligned} S &= \{s_i \mid i = 1 \ldots n\} \\ T &= \{t_j \mid j = 1 \ldots m\} \quad m \geq n \qquad (4.2) \\ \mu &= \{(k,l) \mid s_k \in S, t_l \in T\} \qquad (4.3) \\ B(u) &= \{s_k + u \cdot (t_l - s_k) \mid s_k \in S, t_l \in T, (k,l) \in \mu\} \qquad (4.4) \end{aligned}$$

where μ is a mapping of models and u denotes the amount of morphing. However, with the pre-assigned weights for the area member vertices, the equation becomes

$$B(u) = s_k + u \cdot w_k \cdot (t_l - s_k) \qquad (4.5)$$

Yet this equation is the straightforward morphing equation for one particular area. For vertices that are shared by areas more than one, we must specify an equation of combining these weight values and using them on the morphed vertex. Several different equations are examined for this purpose. One should notice that these equations are crucial together with the choice of area selection. If all areas have vertices with members weighing at the peak value (1.0, for our implementation), boundary regions become undefined; unless some spatial parameters are considered, such as distance. This is not the case for our present work, we assign decaying weights to the outer area members, which defines our boundary regions.

At this point, vertex-neighborhood graph becomes important. In different trian-

gulations, same member vertices become nearer or further neighbors and this affects the assigned weights. This problem is depicted in Figure 4.10.

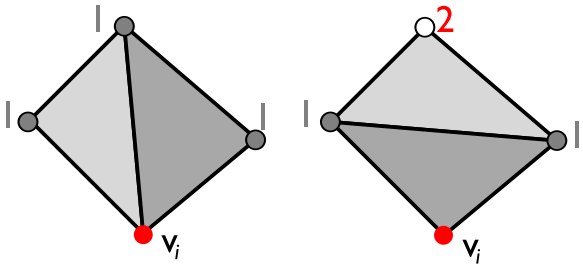


Figure 4.10. At left triangulation, topmost vertex is distance-1 neighbor of the bottom vertex; at right triangulation, it becomes distance-2 neighbor.

For obtaining smooth boundary regions, the morphing equation for shared vertices becomes important. One restriction is that we require all vertices to morph completely to the corresponding ones at the target mesh, when the morphing parameter u equals 1. In order to achieve a complete morphing at boundary regions with vertices that are *not* complete members of (i.e. having a weight value 1 for) *any* area, we use the weighted mean of the morphing functions. Given M_j, the subsets of target models with weight assignments, the morphing equation becomes:

$$\begin{aligned} M_j &= \{(w_{kj}, s_k) \mid s_k \in S\} \\ B(u)' &= s_k + (t_l - s_k)\frac{\sum u_j w_{kj}}{\sum w_{kj}} \mid s_k \in M_j,\ j = 1 \ldots n \end{aligned} \tag{4.6}$$

where u_j denotes the amount of morphing for the j^th subset.

This equation provides a complete morph for vertices that are not complete members of any area. Nevertheless, for a vertex that is the element of only *one* area, our requirement is still not satisfied. In that case, we discard the weight of the vertex and only use the morphing parameter. That is, the equation becomes the same as 4.4. This is a rare case though, as whole body is marked with anthropometric areas which are continuous by definition.

5. IMAGE SYNTHESIS

We create a sophisticated 2D image of the final models with raytracing. This part of work came out of the restriction that a study on 3D garment modeling could not be realized in the possible timeline and therefore we decided to keep the garments as image files that store necessary information. Otherwise, the study would lead to soft-body modeling combined with a 3D dressing simulation using the created human body. Nevertheless, in this section, we will basically go over the background of 3D to 2D image synthesis and justify our approach for that matter.

5.1. Rendering

Rendering is the process of synthesizing an image from a 3D environment. The environment is a description of three dimensional objects in a defined language or data structure. It may contain geometry, viewpoint, texture, lighting, and shading information. There are mainly two rendering techniques, *Scanline Rendering* and *Raytracing*.

Scanline rendering is the preferred method for generating most computer graphics in motion pictures. It is also the method used by video games and most scientific/engineering visualization software (usually via OpenGL). Currently, scanline algorithms have been widely and cheaply implemented in hardware.

For rendering photorealistic scenes, **raytracing** is the dominant method.

5.1.1. Raytracing

Raytracing is a technique for generating an image by tracing the path of light through pixels in an image plane. The technique is capable of producing a high degree of photorealism; usually higher than that of typical scanline rendering methods, but at a greater computational cost. However this computational cost can be reduced by acceleration techniques.

For each screen pixel, an imaginary ray is cast from the camera into the scene. Intersections between the ray and scene objects are compared and the closest one to the camera is used to color the pixel, making hidden surface removal implicit.

If the object is reflective or transparent/refractive, a second ray is cast (or bounced off the object) to find out what the object is reflecting or letting show through. We omit casting second rays as we have one object in the scene. Rays can also be cast towards light sources to determine shadows.

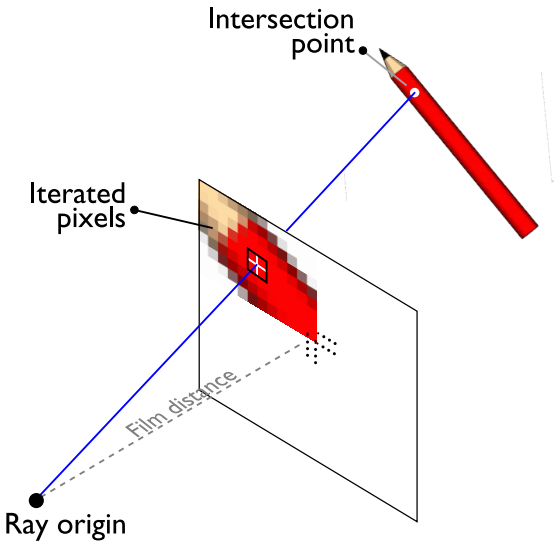


Figure 5.1. A visualization of the raytracing process

In our case, we have only one primitive, the whole body mesh. As the body mesh consists of triangles only, we reduce the problem to two main issues. First, finding the intersecting triangle. Second, finding the exact intersecting point inside that triangle.

For triangle intersection problem, we use 3D spatial subdivision as an acceleration technique. Spatial subdivision techniques offer an efficient means of identifying the objects which are near the path of a ray. We use octrees for this purpose. And for locating the exact coordinate of intersection, we use a barycentric coordinate based linear system.

5.2. Raytracing with Acceleration by Octrees

We create a 2D image of the resulting 3D model by raytracing accelerated by octree-based spatial subdivision. Rays are traced for only the first intersection, because the specular property of body skin is omitted. During raytracing, marked points on the 3D model are also marked in the 2D image for further use in the dressing environment (refer to chapter 6).

Octrees are hierarchical data structures used for efficiently indexing data associated with points in three-space and have been applied to problems such as hidden surface elimination and computation of 3D digital convex hulls. They are introduced into ray-tracing by Glassner [29]. They are constructed by recursively subdividing rectangular volumes into eight subordinate octants until the resulting *leaf volume*s, or *voxel*s, meet some criterion for simplicity. We use a *depth limit* for that purpose. We define the bounding box and then subdivide recursively until the depth limit is reached.

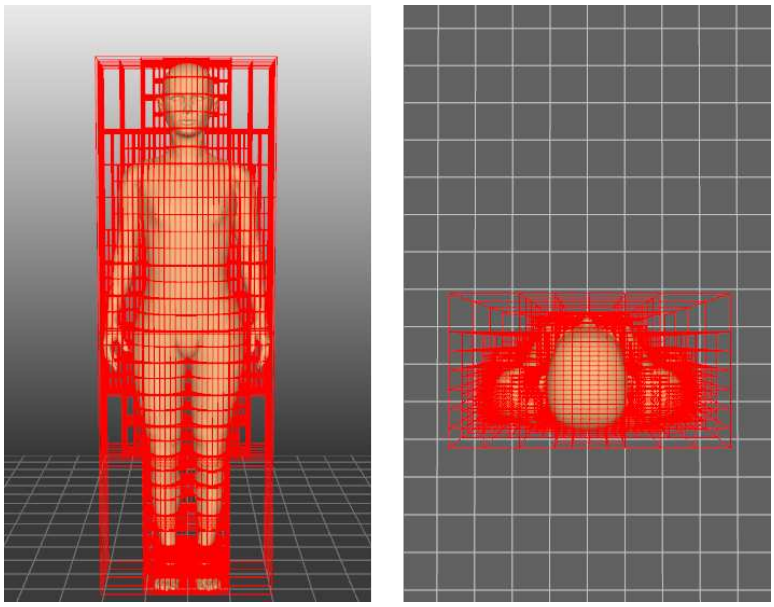


Figure 5.2. Octree built on polygon existence in 3D space

After we have the octree as a data structure in the memory, we start shooting rays as we move along the resulting image. The leaves of the octree which intersect with the shooting ray are pushed onto a *testing* stack. Then all the leaves in that stack are searched for the closest intersecting mesh. As seen in Figure 5.3, all the tested

leaves are candidates for containing the actual intersection. The acceleration works by finding the candidate leaves in *O(log n)* time instead of traversing the whole structure in *O(n)*.

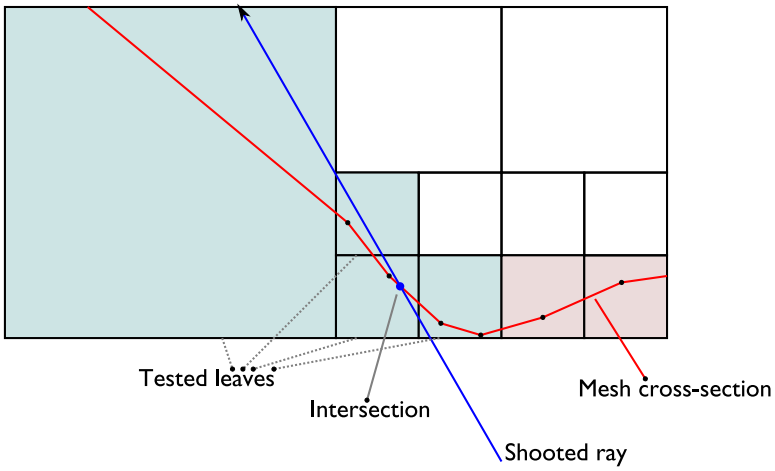


Figure 5.3. 2D representation of a ray shooting iteration using octrees

Having found the intersection, we calculate the light intensity at that point, mainly by the difference between the current light vector and the surface normal at the point of intersection. However, we do not have the surface normal directly at hand, therefore we use the barycentric coordinates of the point inside the containing polygon, to calculate the normal. Simply, the normal is a weighted sum of normals at the corners. We have the normal values in hand as we use them in drawing in the 3D OpenGL environment.

Barycentric coordinates provide a way to parameterize a triangle in terms of two variables, b_1 and b_2. To derive an algorithm for intersecting a ray with a triangle, we insert the parametric ray equation into the triangle equation.

$$o(r) + d(r)t = (1 - b_1 - b_2)\mathbf{p}_0 + b_1\mathbf{p}_1 + b_2\mathbf{p}_2 \tag{5.1}$$

where $\mathbf{p}_0$, $\mathbf{p}_1$ and $\mathbf{p}_2$ are the corners of the triangle.

Following the technique described by Möller and Trumbore [30], we use the shorthand notation $\mathbf{E}_1 = \mathbf{p}_1 - \mathbf{p}_0$, $\mathbf{E}_2 = \mathbf{p}_2 - \mathbf{p}_0$, and $\mathbf{T} = o(r) - \mathbf{p}_0$. To obtain both the barycentric coordinates of the intersection point and the distance along the ray, we solve the linear system:

$$\begin{bmatrix} d(r) & \mathbf{E}_1 & \mathbf{E}_2 \end{bmatrix} \begin{bmatrix} t \\ b_1 \\ b_2 \end{bmatrix} = \mathbf{T} \tag{5.2}$$

```
Input: a ray R
Find the voxels intersecting with R
Initiate t = ray position of intersection, with
a sufficiently large number L indicating "no intersection".
Loop through the voxels, starting from the closest voxel to ray origin.
{
  Loop through the triangles inside the voxel
  If there is an intersection < t, update t
}
If t < L, calculate illumination at that point.
```

Figure 5.4. Pseudocode for octree intersection algorithm

5.3. Illumination

After obtaining the point of intersection, we calculate the illumination at that point. Surface illumination calculation is done by using local diffuse illumination. In this method, surface normal, position of light and camera coordinates are taken into account. An ambient term is also added to simulate global diffuse illumination. Since specularity of the body skin is ignorable, specular reflection is discarded. Additionally,

in order to have a lighting similar to the 3D environment, more than one light source is used. Illumination is calculated by Equation 5.3.

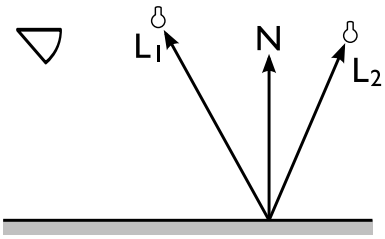


Figure 5.5. Rays used in illumination calculation

$$I = K_A I_A + \sum_{i}^{n} K_D(\mathbf{N} \bullet \mathbf{L}_i) I_i \tag{5.3}$$

In this equation, K_A and K_D denote the effective amount of ambient and diffuse light intensities on the scene. We take $K_A = 0.3$ and $K_D = 0.8$ in our current implementation.

Computing the surface normal $\mathbf{N}$ at the exact intersecting point is straightforward. We use the barycentric coordinates found by the equation 7.2, and put normal values instead of point coordinates.

$$\mathbf{N} = (1 - b_1 - b_2)\mathbf{n}_0 + b_1\mathbf{n}_1 + b_2\mathbf{n}_2 \tag{5.4}$$

5.4. Anti-aliasing

In signal processing, the result of under-sampling a high frequency function is *aliasing*, where low frequency errors that aren't present in the original function appear. Geometry is one of the biggest causes of aliasing in rendered images. When projected onto the image plane, an object's boundary introduces a step function, where the image function's value discontinuously jumps from one value to another.

In raytracing, using only the intersection points in the final image results in aliased edges. We implemented super-sampling using a *uniform* and a *stratified jittered* pattern for anti-aliased edges. Results of these two patterns are considerably different

at small scale windows. As seen in Appendix A, best results arise from super-sampling with jittered patterns. A stratified jittered pattern turns aliasing from the uniform pattern into high-frequency noise while still maintaining the benefits of stratification. The experiment also proves the claim that uniform super sampling may keep or even exacerbate aliasing.

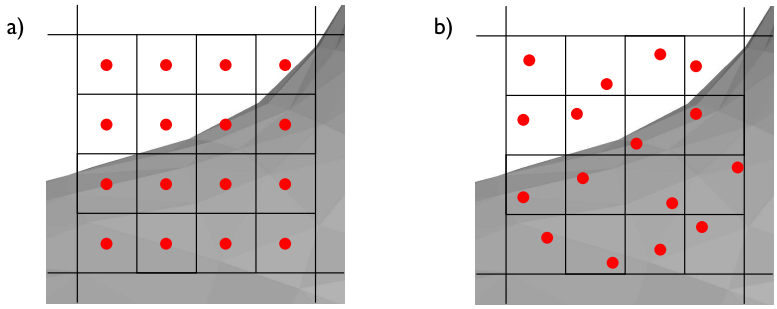


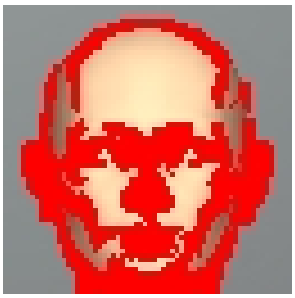
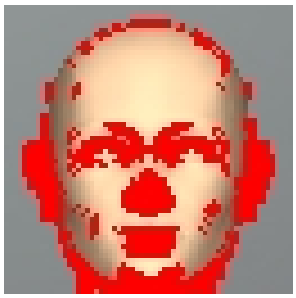
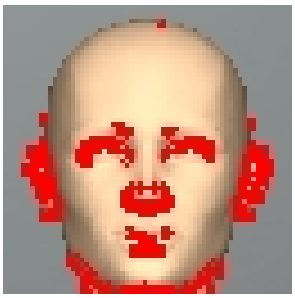
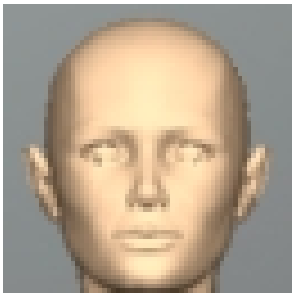
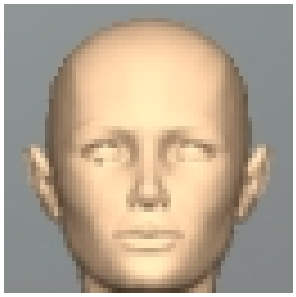
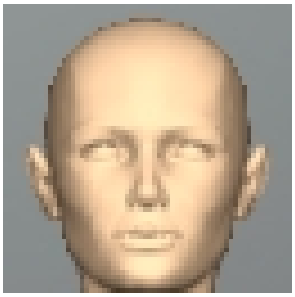
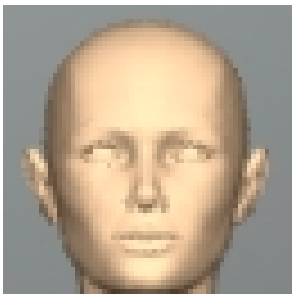
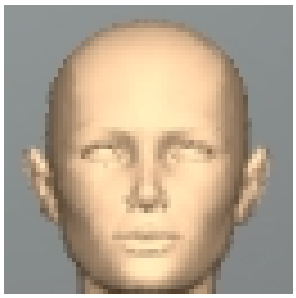
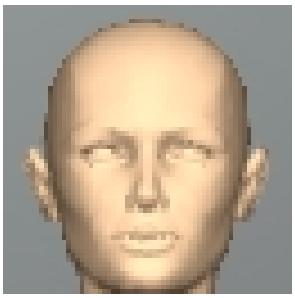
Figure 5.6. One pixel is divided into 16 subpixels. Rays are shot by a a) uniform pattern, using central coordinates b) stratified jittered pattern, using randomly chosen coordinates inside the subpixel squares.

In uniform super-sampling, a pixel is divided into the specified number of subpixels and rays are shot to these subpixels' central coordinates. The more the pixel is divided, the more information is collected from the geometry and projected onto the image. This idea is the same in jittered patterns, the only difference is that, for each subpixel a random coordinate inside it is used, not the central coordinate. The random pattern is expected to turn aliasing into a less objectionable distortion to human eye, which is the case for our virtual world with a mere human model.

After first pass of ray tracing, aliased edges of the image are found by defining a maximum intensity threshold for neighboring pixels. Pixels with neighbors having highly differentiated intensities are divided into specified number of subpixels, and a ray is shot through each. This time the overall color is determined as the average of all these subpixel colors. The number of subpixels is proportional to the render resolution. Specific numbers give sufficient results for specific resolutions. For our case, a three by three subdivision gave out satisfactory anti-aliasing results. The results of experimentations are discussed in the following section.

5.4.1. Experimentation

We made several experiments on anti-aliasing for the best visual results. First row shows the aliased pixels found according to the given *intensity difference threshold*s. Second row shows the images with anti-aliasing applied by 3x3 uniform patterns. Third row shows the images with anti-aliasing applied by 3x3 stratified jittered patterns. See how aliased pixels are found with higher thresholds and the patterns behave on the overall image.

Aliased pixels			
	IDT = 0.1	IDT = 0.15	IDT = 0.25
3x3 Uniform Pattern			
3x3 Stratified Jittered Pattern			

6. APPLICATION: VIRTUAL DRESSING ROOM

Virtual Dressing Room (Original - Turkish: ***Sanal Giyim***) is an application which combines our body modeling approach with an overall dressing environment personalized by making use of face photographs. Face photographs are processed to appear smoothly in the resulting synthetic body image. The whole procedure is explained in the following section.

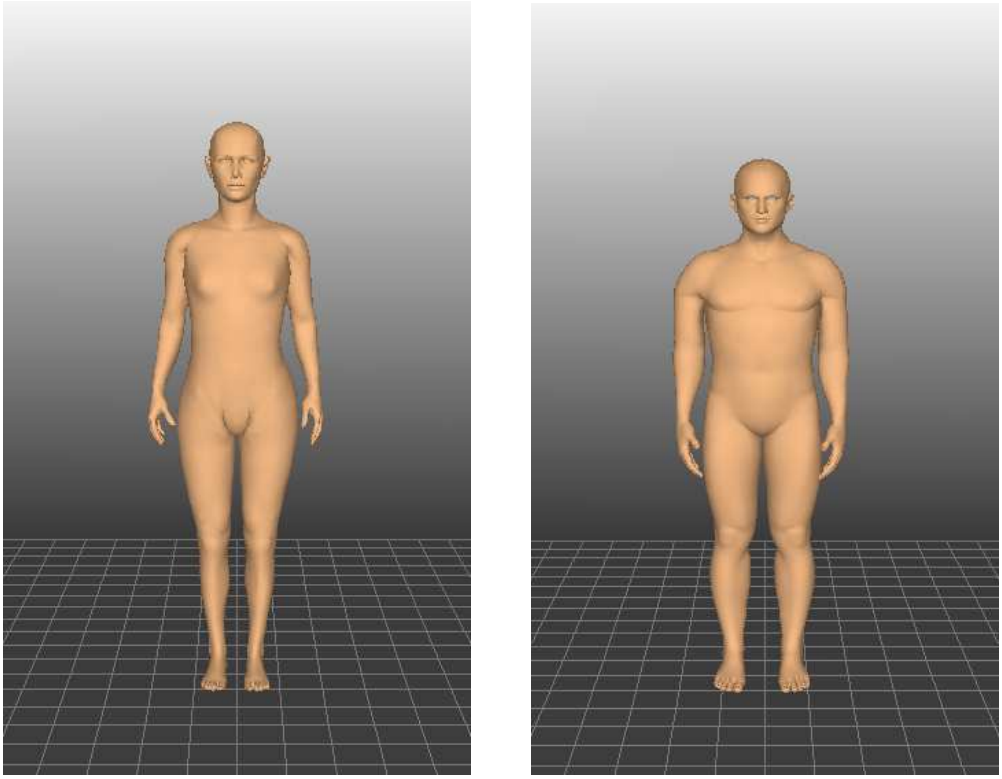
6.1. Overall Interface

We present a step-by-step, user-friendly interface for the program, so that a novice user can easily find its way through it, ending up with a personalized human model of his or hers, on which he or she may virtually try clothes.

In the first step, recently loaded files are presented to the user, in which we save several properties of our personalization process for a specific body. If the user does not a load a file, then gender selection is required. After these steps, the user is introduced to the 3D model of the selected gender interpolated to normal measures (Figure 6.1). Normal measures differ for male and female models. Following steps are the main personalization functions.

6.1.1. Body Personalization

This part of the program consists of a 3D OpenGL environment and common GUI elements such as sliders and textboxes. We mainly apply our body modeling approach at this part, presenting a real-time interactive modeling screen. The window enables the user to either enter his / her body measurements -which can be taken by simple measurement tools- or adjust a suitable geometry using body girth sliders.



(a) Female model (b) Male model

Figure 6.1. Initial models with normal measures in *Sanal Giyim*

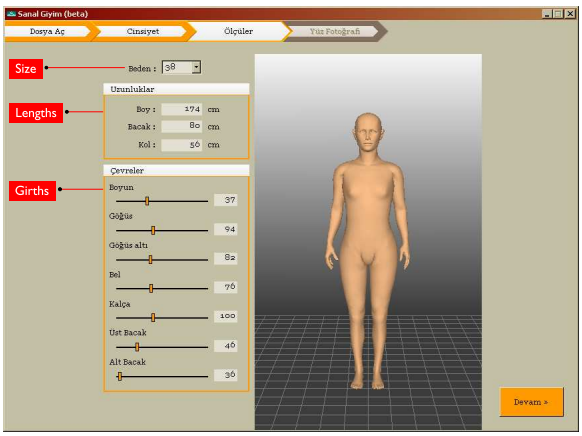


Figure 6.2. Sanal Giyim - Body Personalization

6.1.2. Face Personalization

After having a reasonable geometric replication of the body, we present the face photograph wizard. This part is also divided into steps, starting with the load of an existing frontal face photograph, which is a JPEG or BMP file.

Initially, neck points are marked on the face photograph. There are two purposes of this marking: One is to provide the skin - garment separation on the image, so that part of garment does not appear on the resulting image. The other is to resize the head image correctly so that it adjusts to the body render.

After the neck marking process we apply a background extraction so that the face and hair are separated from the background. The seed for the extraction is calculated as the average color of an $n \times n$ pixel neighborhood. Central pixel is chosen randomly with the only constraint that it is in the background, and n is specified by the resolution of the image. For a meaningful pixel neighborhood to represent the background color, n should be much smaller than the resolution parameters -i.e. width and height-, but large enough to tolerate noises. The color-spatial differentiation is depicted in Figure 6.3.

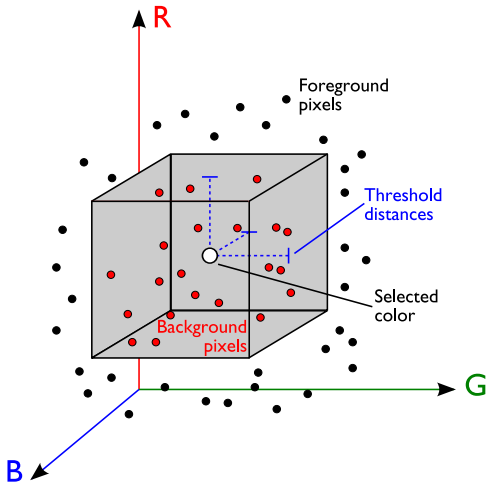


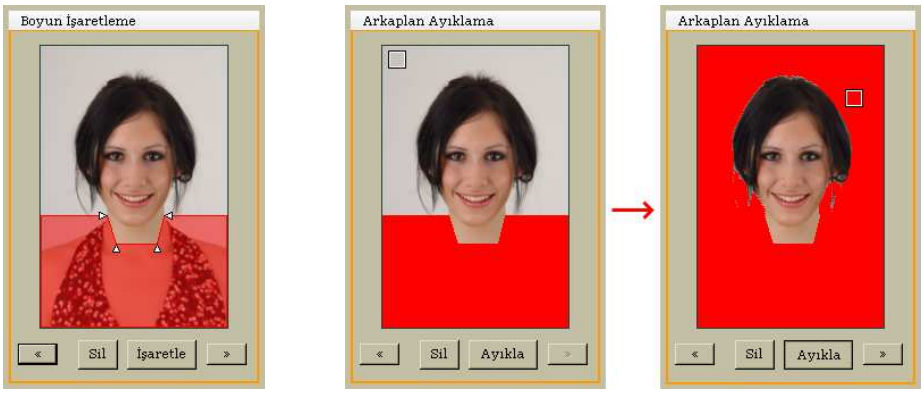
Figure 6.3. Color-spatial differentiation for background extraction

Another parameter for the extraction is the color-spatial distance threshold. We define a differentiation cube in this space, centered around the seed with threshold

distance as half-width. The pixels outside this cube are marked as *background* and the ones inside it are marked as *foreground*. We extract the background with a scanline boundary filling on the image, starting from the seed.

```
Input: Image pixels, Seed position (x_s,y_s) on image
Push (x_s,y_s) onto stack W
While Stack is not empty
{
  Pop Stack (retrieve coordinates x,y)
  Mark current run y by iterating on x until foreground is hit
  To left: Push unfilled pixels before above/below border pixels
           --> new above/below seeds
  To right: Push unfilled pixels after above/below border pixels
            --> new above/below seeds
}
```

Figure 6.4. Pseudocode for background extraction algorithm



(a) Neck Marking (b) Background Extraction

Figure 6.5. Two steps of the Photograph Wizard

Finally, we present a skin color selection interface, to make the user select a pixel neighborhood in the same manner that the background sample was selected in the background extraction step. When the user selects an area in the photograph to be the skin color sample, we form a preview render with that color. The user can change

the skin color and reproduce the render several times until the color matching of the render and the photograph is satisfactory. At this preview frame, we also present a simple GUI giving the ability to move the neck mark points *indirectly*, so that the face is *directly* and therefore correctly placed and rescaled on the body image.

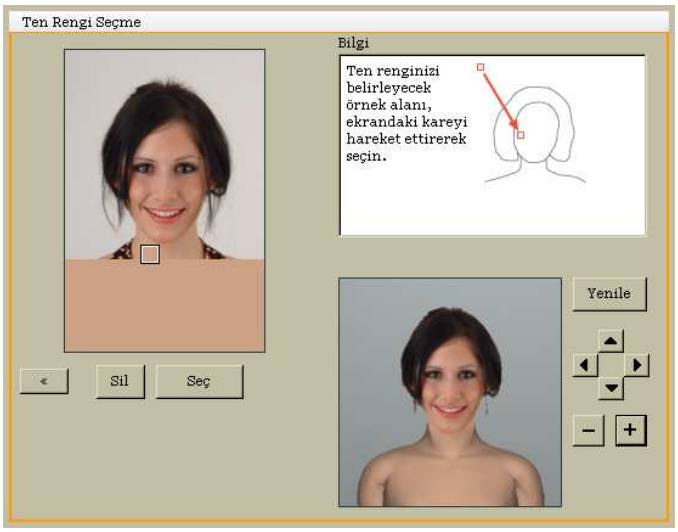


Figure 6.6. Sanal Giyim - Preview render and face positioning

6.1.2.1. Scaling. We apply a two-pass scaling method for resizing images. We employ the algorithm in adjusting the size of face and garment images. Scaling is applied to 32-bit pixels, consisting of a 24-bit RGB channel plus an 8-bit Alpha channel.

The procedure creates a new output buffer according to the required scale, either smaller or larger than the original image. The image is first scaled vertically and then horizontally, using the pre-determined filters. Best results are obtained with bilinear filters. A scaling process is shown in Figure 6.7.

When finally the user is satisfied with the preview, a full body render is performed in the virtual dressing room. And once the render is complete, the user's selection of available products from the website appears on the right pane. At this point, the selected size and color of an item in the list can be "tried on" by downloading the garment itself and the alpha channels.

Garment images are located on the server where website is located. Original

Figure 6.7. 2-Pass scaling of a dress image. Original image is on the left. Scaling is applied first vertically (middle), then horizontally (right).

images and their alpha channels reside in separate folders, sharing the same name defining the color and size of the garment. When that specific color and size is selected and a try-on request is triggered from the application, an FTP request is sent to the server for those files and download begins. When all three files are downloaded, they are loaded into the program according to the data saved at the comment section (i.e. *COM marker*) of JPEG files. We use libjpeg[31] for this purpose, which supports writing data to marker sections in JPEG files.

Figure 6.8. Sanal Giyim - *The Dressing Room*

We apply multilayered dressing by using a two step process:

- *Rendering arms to a second render layer*: That is simply because we present the side and rear views of the 3D body in the dressing room and enable dressing in these views, too. In these views, arms should be visible on top of the garment parts which are touching the torso.
- *Marking the arms as a second alpha layer in garments*: This completes the multilayered dressing together with the previous step. We mark out the arm section in the garment image and save as a second alpha layer. Inside the dressing room, after body arms are rendered to the second layer, we place garment's arms section on top of this second render layer. Note that another colored image of the garment is not required, just saving the arms section separately is a sufficient action.

A dressing process is depicted in Figure 6.9. On top left, the original render layer is shown. The other layers are built upon the render layer gradually. In painting order: The original rescaled garment layer, second render layer and second garment layer are drawn.

Figure 6.9. The multilayered dressing process

Besides body related multilayering, there is also a layering scheme for the garments. Required functionality is to preserve the real life dressing order. For example, wearing underwears first, then shirts / pants, and so on. We developed a flexible layering for that purpose, where new layers can easily be added to the system. The layers are drawn on top of each other, whereas at one layer, garments are drawn in the

wearing order specified by the user. Currently there are eight layers defined:

- Underwear
- Tank tops, Slips
- Shirts, Skirts, Pants
- Jackets, Cardigans
- Belts, Jewelry
- Coats
- Scarves, Shawls
- Shoes, Bags, Slippers

6.2. Garment Preparation

As mentioned in section 3.1, we parameterized our model using the EN 13402 standard, which is a dress measurement based human body parametrization. We implemented a small tool for both entering a specific garment's measurement information and also marking the garment for application use.

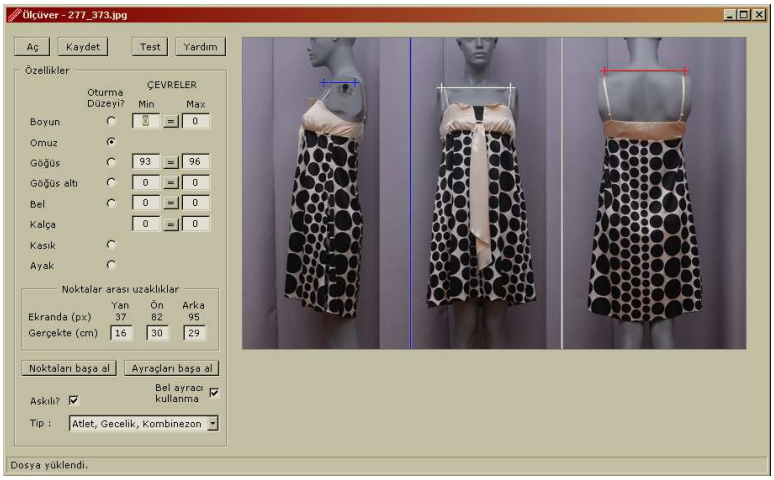


Figure 6.10. A sample screen from the Garment Measurement Tool.

The fitting of a garment image to a body render has three main aspects.

- *The real-world fitting constraints*: This is the algebraic actualization of garment stretching properties and body girths. If a garment's specified stretching part

stretches in a range of $s_{min} - s_{max}$, then the body on which it is tried should have a girth value higher than s_{min} and lower than s_{max} at that part.

- *Fitting levels*: We specified seven fitting levels for all kinds of garments. These levels are determined by the measurement areas and fit garments' vertically invariant points. This is, answering the question "*When dressed, which part of a garment stays invariant according to the body?*". A shirt fits to the shoulder but stretches down in the torso, or a bikini bottom fits to the crotch but can stretch up to different heights on the hip. The resulting fitting levels are namely neck, shoulders, bust, under-bust, waist, crotch and feet. These are represented as radio buttons in the GUI.
- *Unit transformation between the garment's image world to the body's 3D world*: For the coordinate transformation, a reference point must be supplied for both worlds to be merged in a new environment. First we specify a 3D-to-metric transformation by using the metrics of a standard size-34 female (or size-46 male) model and its corresponding 3D replica. The 3D model can only be determined *proportionally* given the standards of a reference model. Providing the proportions as a whole, one metric is sufficient for the transformation. Second, we specify a garment's actual metrics between two points at the fitting position by simply measuring. This *metric* indication enables the affine transformation of these two worlds in a new environment. We present the *pixel* distance to the user and enable the entering of *cm* data, so that we have the $pixel \leftrightarrow cm$ transformation.

Once the data are specified and previewed on the pre-rendered body images, they can be saved at the comment section of the JPEG files. By doing this, we both keep the garment files usable as images and do not require a separate place like file or database to store these values.

As we employ multilayer dressing, another issue is the determination of alpha channels. We do not employ the common image file formats with 8-bit alpha channels such as PNG, TGA or TIFF, but rather use a different approach, saving alpha channels as separate JPEG files, too. This method enables a flexible layering scheme and simplifies the garment preparation process.

7. EVALUATION

Our method is evaluated at three main phases. Initially, a smoothness measure is employed for the in-between shapes. It is followed by the computational cost of real-time morphing. Finally, for the image synthesis phase, the acceleration effect of octree-based spatial subdivision and the optimum anti-aliasing scheme are discussed.

7.1. Smoothness Measure for Morphed Objects

Laplacian coordinate representation is a stable scheme and is used for several purposes in graphics applications. Mesh smoothing and transformation invariant morphing are some examples. It is based on the center of mass of the neighboring vertices.

$$\bar{\mathbf{v}}_i = \frac{1}{n} \cdot \sum_{j}^{n} \mathbf{v}_j \tag{7.1}$$

A polygonal mesh can be smoothed by moving all of its vertices to their Laplacian

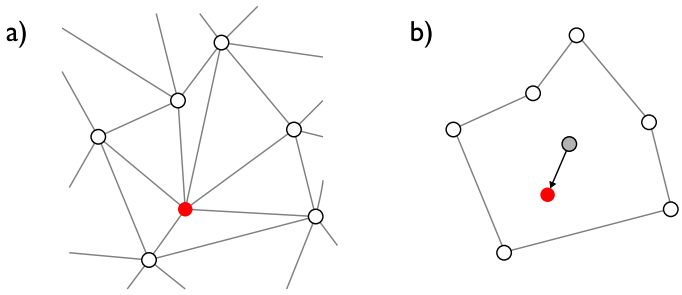


Figure 7.1. a) A vertex (red) and its neighborhood ring (white). b) In Laplacian coordinates, a vertex is represented by the difference to the centroid of its neighbors

coordinates. Therefore, *roughness* - R of a mesh or a vertex set is measured by the sum of distances between original vertices and their Laplacian coordinates. As the vertex count does not change, arithmetic mean - *average* can also be used.

$$R(\mathbf{V}) = \sum_{i}^{n} |\bar{\mathbf{v}}_i - \mathbf{v}_i| \tag{7.2}$$

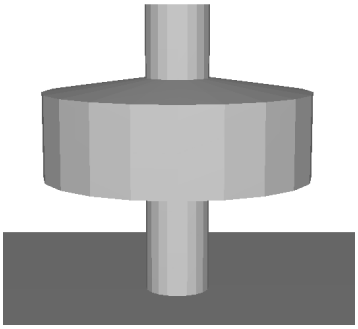
In Figure 7.2, a smoothness measurement process is shown on a cylindrical object. Initially, a deformed shape is obtained by extruding the middle segment of this object, as shown in 7.2(a). Afterwards, a smoothing process is applied to this deformed shape at each step. As seen in (7.2(f)), the final shape is visually acceptable and considerably smooth. This is the required effect for an outcoming human body shape. Therefore, Laplacian distances can be used for evaluating the smoothness of a morphed human body shape.

Figure 7.2(g) shows the average Laplacian distance values versus sample shapes in smoothing order. They are labeled "*Roughness*", being the inverse of smoothness as a measure. As mentioned earlier, we apply local morphing on marked segments of the human body. These segments have to be marked around the anthropometrically determined positions. Additionally, core vertex sets forming the segment should not interfere with each other, to achieve separate interpolation. However, the regions between neighboring segments should be smooth during interpolation. During the marking process, we define these *boundary* regions with vertex set propagations. Best propagation depth for marked segments are found by evaluating the smoothness measure. Smoothness values are recorded at each interpolation step, and these values are compared for different propagation depths.

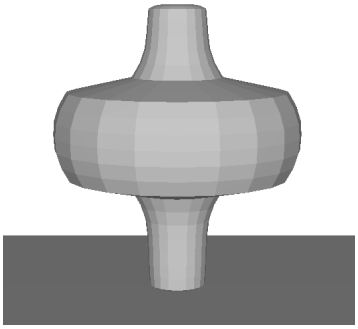
Figures 7.3, 7.4 and 7.5 show the roughness vs interpolation values, for specific segments and their propagations. Two neighboring segments are marked with increasing propagations of their core vertex sets. The effect of each propagation is represented with a separate color. Notice how roughness curves take smaller values and therefore smoothness increases for each case, as propagation depth grows.

7.2. Computational Cost of Morphing

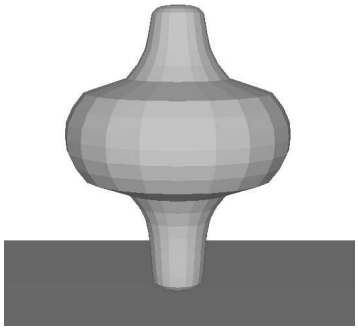
Using the straightforward weighted morphing formulation (4.5), it is obvious that the computational cost increases linearly with vertex or polygon count. This means that the algorithm runs in $O(n)$ complexity where n represents the vertex count. Furthermore, the morphing equation which uses the average mean of the multipliers



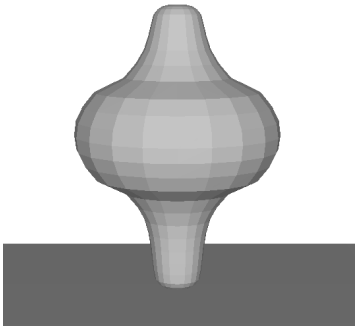
(a) $R = 0.0201896$



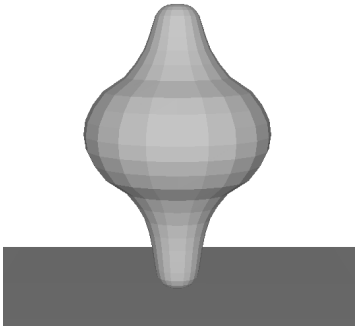
(b) $R = 0.0133217$



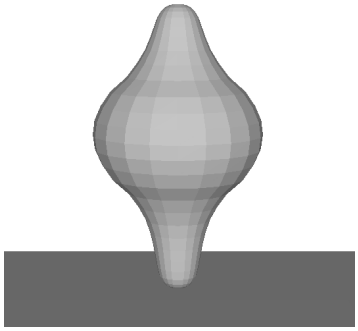
(c) $R = 0.00959175$



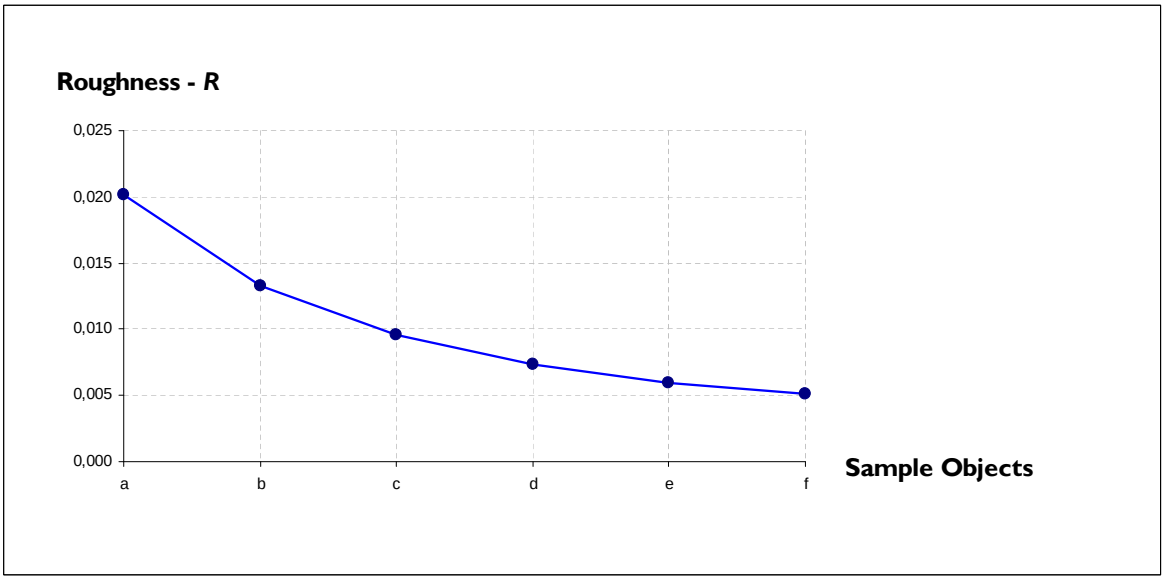
(d) $R = 0.00734621$



(e) $R = 0.00596873$

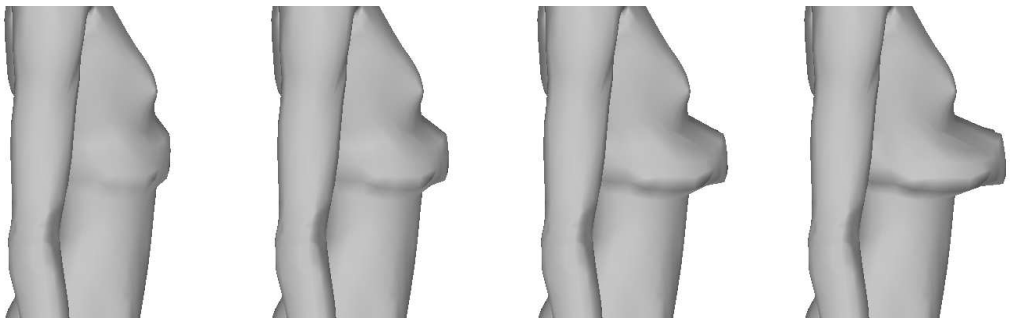


(f) $R = 0.00514077$



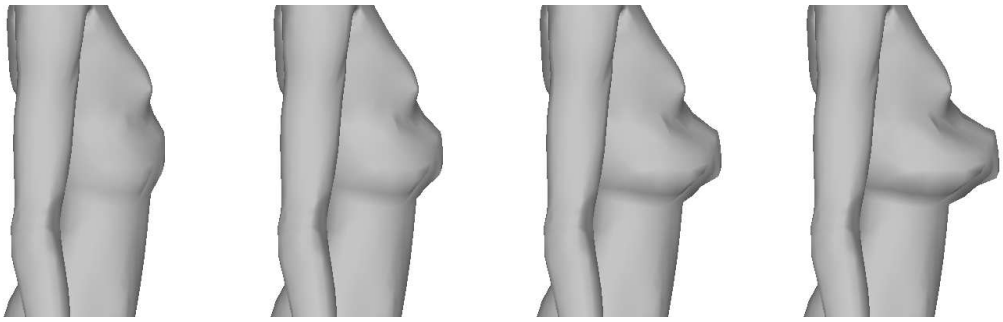
(g) Roughness of smoothed sample objects

Figure 7.2. Average Laplacian distances of a model (a-f) that is smoothed at each step, and the graph depicting the decrease in roughness, justifying the correctness of the measure



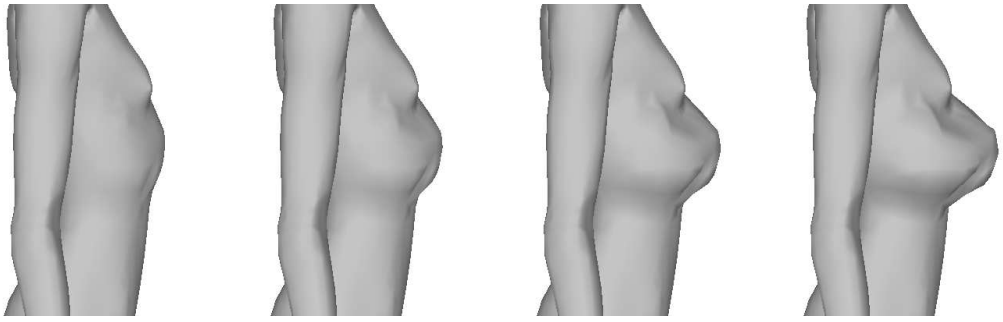
(a) $u = 0.25$ (b) $u = 0.5$ (c) $u = 0.75$ (d) $u = 1.0$

Interpolation of *Underbust* segment with propagation = 0



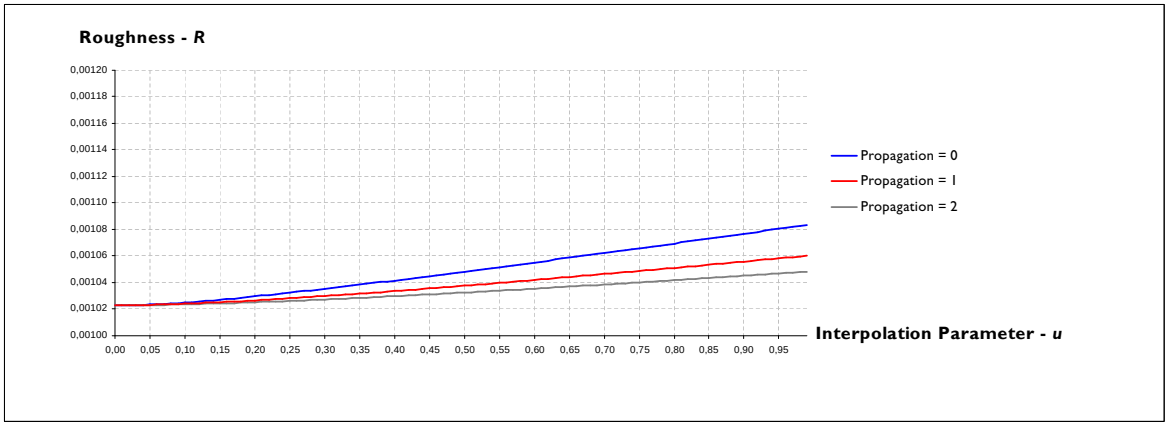
(e) $u = 0.25$ (f) $u = 0.5$ (g) $u = 0.75$ (h) $u = 1.0$

Interpolation of *Underbust* segment with propagation = 1



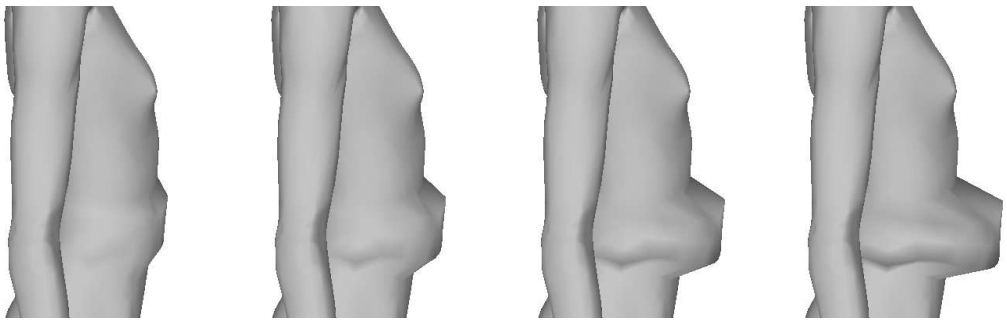
(i) $u = 0.25$ (j) $u = 0.5$ (k) $u = 0.75$ (l) $u = 1.0$

Interpolation of *Underbust* segment with propagation = 2



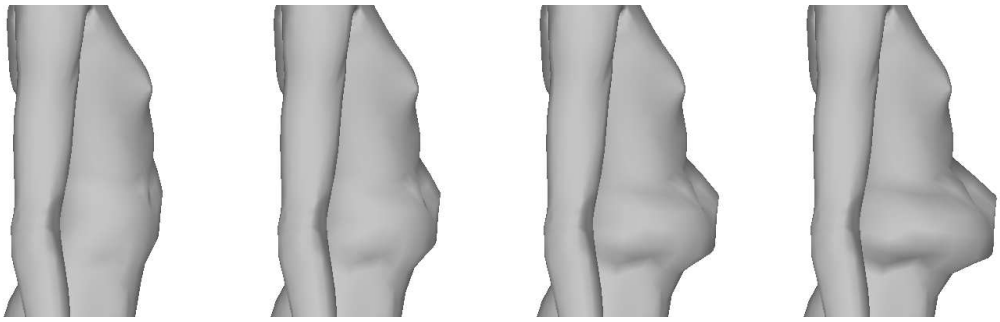
(m) Roughness vs Interpolation for *Underbust* segment with corresponding propagation depths

Figure 7.3. Smoothness measurement for Underbust segment interpolated separately



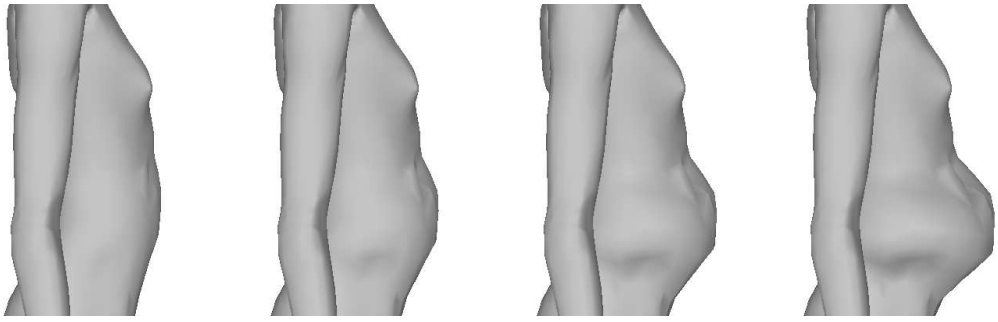
(a) $u = 0.25$ (b) $u = 0.5$ (c) $u = 0.75$ (d) $u = 1.0$

Interpolation of *Waist* segment with propagation = 0



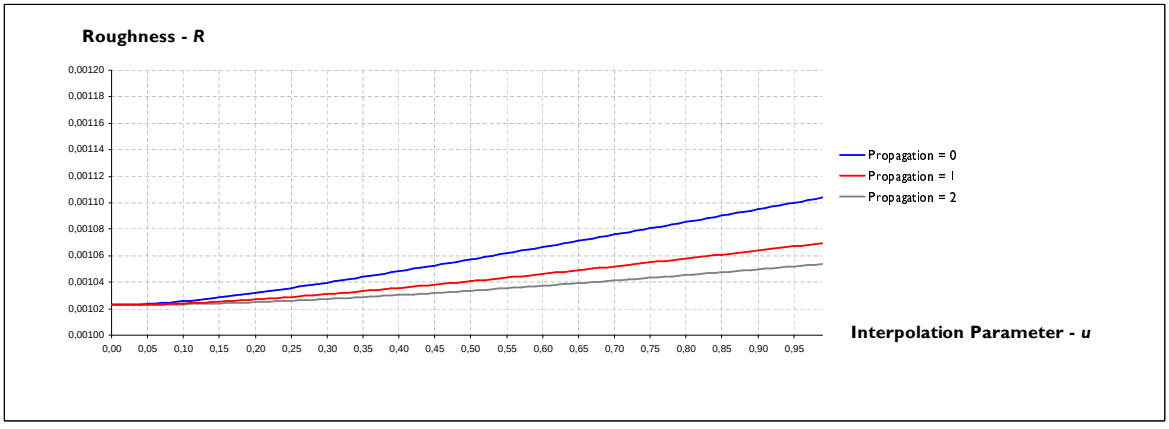
(e) $u = 0.25$ (f) $u = 0.5$ (g) $u = 0.75$ (h) $u = 1.0$

Interpolation of *Waist* segment with propagation = 1



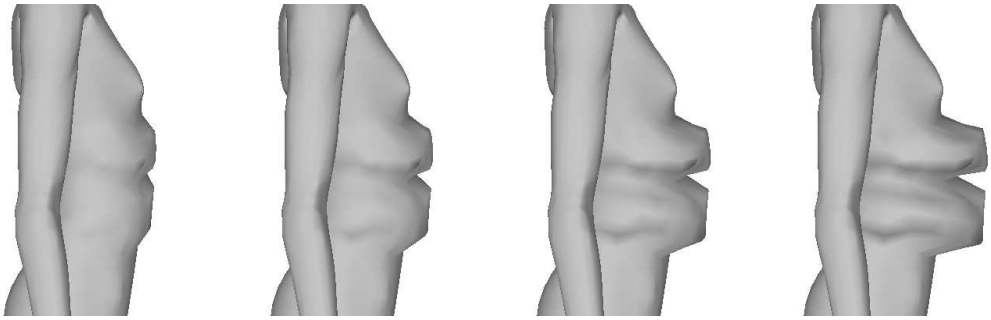
(i) $u = 0.25$ (j) $u = 0.5$ (k) $u = 0.75$ (l) $u = 1.0$

Interpolation of *Waist* segment with propagation = 2



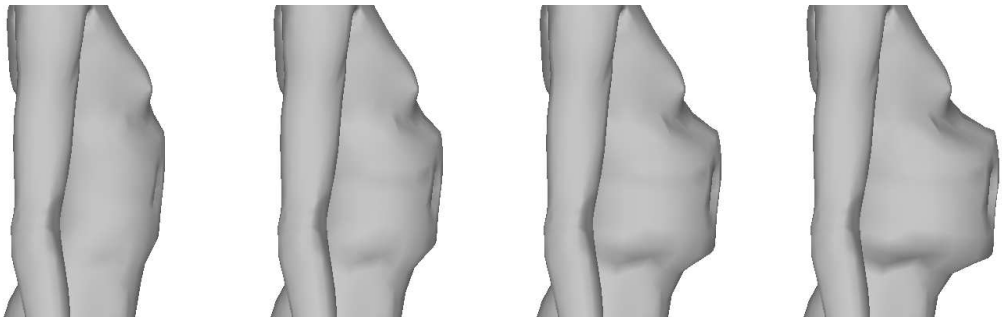
(m) Roughness vs Interpolation for *Waist* segment with corresponding propagation depths

Figure 7.4. Smoothness measurement for Waist segment interpolated separately



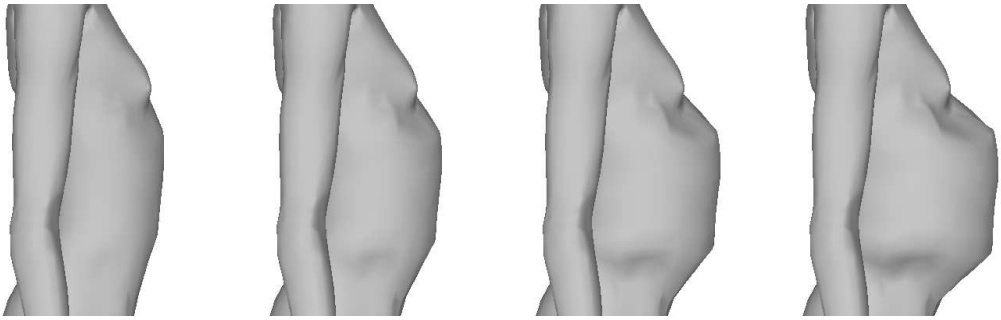
(a) $u = 0.25$ (b) $u = 0.5$ (c) $u = 0.75$ (d) $u = 1.0$

Interpolation of *Waist* and *Underbust* segment together with propagations = 0



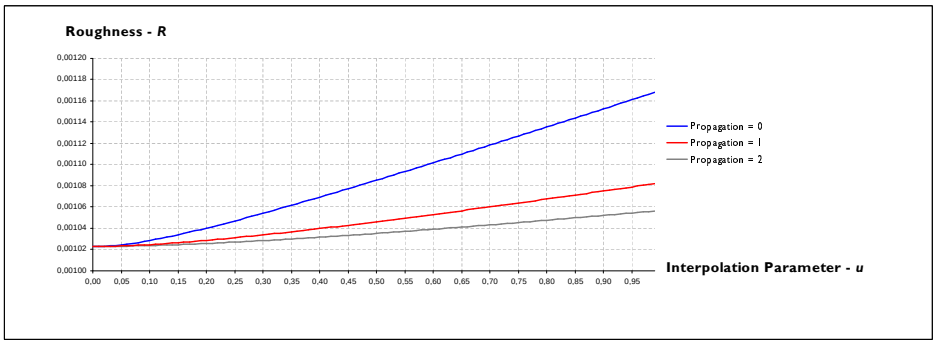
(e) $u = 0.25$ (f) $u = 0.5$ (g) $u = 0.75$ (h) $u = 1.0$

Interpolation of *Waist* and *Underbust* segment together with propagations = 1



(i) $u = 0.25$ (j) $u = 0.5$ (k) $u = 0.75$ (l) $u = 1.0$

Interpolation of *Waist* and *Underbust* segment together with propagations = 2



(m) Roughness vs Interpolation for *Waist* and *Underbust* segment together with corresponding propagation depths

Figure 7.5. Smoothness measurement for Waist and Underbust segment interpolated together

(4.6) adds only a constant computation per vertex, since the number of areas used are determined by a constant segmentation of the body.

Most of the computational time is accounted for vertex normal calculations in the 3D OpenGL environment. A vertex normal calculation contains the averaging of the face normals which contain that vertex as a corner. A face normal calculation is made by two subtractions and a cross-product. For a 15k polygon model, this calculation slows down the process considerably. Therefore, normal calculations are made each time a local shape morphing is finished.

7.3. Acceleration of Octrees

Several depth values for constructing the octrees are tried for accelerating the ray-tracing process. Small depths remain insufficient for acceleration and larger depths increase the number of tested volume leaves, making no improvement on computational time. Figure 7.6 shows the effect of several octree depths on total rendering time in seconds. As seen on the graph, depth 5 octree provides the shortest rendering time among others.

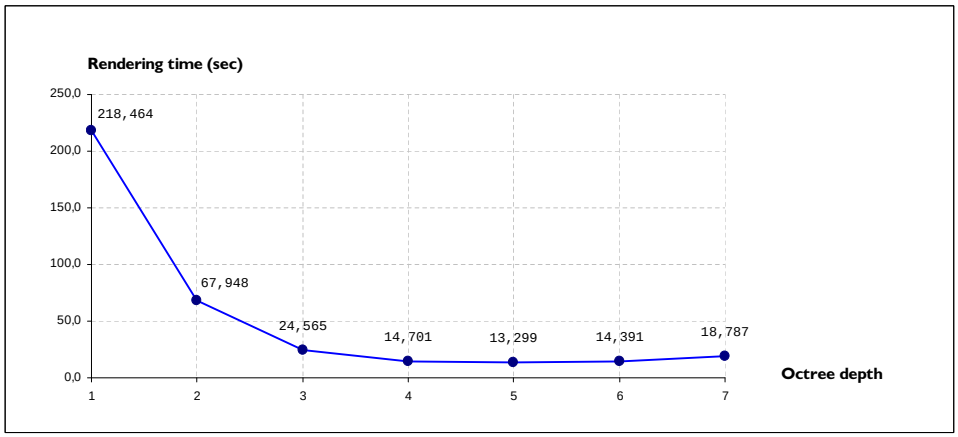


Figure 7.6. Rendering time versus Octree depth graph

7.4. Optimum anti-aliasing scheme

Aliased pixels are found by comparing the maximum intensity difference between the pixel and its 1-neighbors with a pre-determined threshold. This threshold determines the number of subdivided and re-processed pixels, affecting both the overall computational time and the quality of the resulting image.

Figure 7.7 shows the number of processed pixels *before* and *after* anti-aliasing pass determined by the given thresholds, on the same graph. The number of pixels remaining *aliased* after anti-aliasing are determined by a constant IDT, which is taken 0.2. This is for enabling comparison between different given thresholds and evaluating the improvement by the given threshold.

As seen on the graph, threshold values 0.1, 0.15 and 0.2 have nearly equal aliasing after processing on the first rendered image. Their processing time is quite different though, directly proportional to the number of pixels accounted for being *aliased*. Hence an IDT value around 0.2 is proved sufficient for determining the aliased pixels.

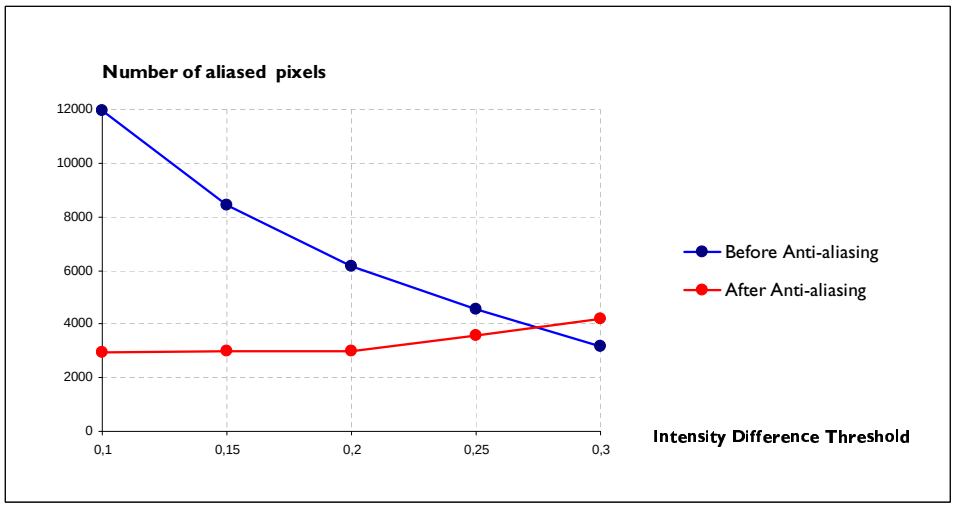


Figure 7.7. Number of aliased pixels before and after anti-aliasing versus IDT graph

8. CONCLUSIONS

We have developed en interpolation based 3D human body modeling environment and a virtual dressing application. Our evaluations have shown that anthropometric segmentation of the human body allows local control of the interpolation and enables better modeling.

Since human body has a smooth shape, created shapes by using local control should be smooth as well. For evaluating the smoothness of the in-between shapes, average distance between vertices and their Laplacian representations are used. Several measurements are taken with different vertex propagations of body segments. Measurements indicate that smoother in-between shapes are formed when vertex propagation is deeper. This is because the smooth characteristic of the Gaussian function that is used in assigning weights to the vertices affects the shape more when propagation is deeper.

Several octree depths are tried for raytracing and the optimum depth is found by comparing the rendering time. Tests show that there is an optimum depth for a specific raytracing setup with the same camera coordinates and mesh. Deepening the octree does not improve rendering time after optimum point. Additionally, tests with different IDT values for anti-aliasing show that there is an optimum threshold value for a specific raytracing setup. However, aliased pixels remain after subpixel rendering. For better anti-aliasing results and smoother images, computational cost increases.

Short term future work includes further acceleration of raytracing by experimenting with other spatial subdivision methods. Raytracing and anti-aliasing could also be implemented in an *adaptive* manner, which could save additional time. Long term future work includes creating skeleton-based animatable replicas by using the local control method. Facial replication can also be implemented in 3D, by analyzing photographs and further 3D deformation.

REFERENCES

1. Brett Allen, Brian Curless, and Zoran Popović. The space of human body shapes: reconstruction and parameterization from range scans. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 587–594, New York, NY, USA, 2003. ACM.

2. W-S. Lee, J. Gu, and N. Magnenat-Thalmann. Generating animatable 3D virtual humans from photographs. In M. Gross and F. R. A. Hopgood, editors, *Computer Graphics Forum (Eurographics 2000)*, volume 19(3), 2000.

3. Hyewon Seo and Nadia Magnenat-Thalmann. An automatic modeling of human bodies from sizing parameters. In *I3D '03: Proceedings of the 2003 symposium on Interactive 3D graphics*, pages 19–26, New York, NY, USA, 2003. ACM.

4. Ferdi Scheepers, Richard E. Parent, Wayne E. Carlson, and Stephen F. May. Anatomy-based modeling of the human musculature. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 163–172, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

5. Jianhua Shen and Daniel Thalmann. Interactive shape design using metaballs and splines. In *Implicit Surfaces'95*, pages 187–196, Grenoble, France, 1995.

6. Adrian Hilton, Daniel Beresford, Thomas Gentils, Raymond Smith, and Wei Sun. Virtual people: Capturing human models to populate virtual worlds. In *CA '99: Proceedings of the Computer Animation*, page 174, Washington, DC, USA, 1999. IEEE Computer Society.

7. Hee-Deok Yang and Seong-Whan Lee. Reconstructing 3d human body pose from stereo image sequences using hierarchical human body model learning. In *ICPR '06: Proceedings of the 18th International Conference on Pattern Recognition*, pages 1004–1007, Washington, DC, USA, 2006. IEEE Computer Society.

8. B. Rosenhahn and R. Klette. Geometric algebra for pose estimation and surface morphing in human motion estimation. pages 583–596, 2004.

9. K.M. Robinette, H.A.M. Daanen, and E. Paquet. The caesar project: a 3-d surface anthropometry survey. In *Second International Conference on 3-D Digital Imaging and Modeling*, pages 380–386, 1999.

10. Anthropometric modeling programs-a survey. *IEEE Comput. Graph. Appl.*, 2(9):17–25, 1982.

11. Zouhour Ben Azouz, Marc Rioux, Chang Shu, and Richard Lepage. Characterizing human shape variation using 3d anthropometric data. *Vis. Comput.*, 22(5):302–314, 2006.

12. Hyewon Seo, Frederic Cordier, and Nadia Magnenat-Thalmann. Synthesizing animatable body models with parameterized shape modifications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 120–125, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

13. Mustafa Kasap and Nadia Magnenat-Thalmann. Parameterized human body model for real-time applications. In *CW '07: Proceedings of the 2007 International Conference on Cyberworlds*, pages 160–167, Washington, DC, USA, 2007. IEEE Computer Society.

14. Alan H. Barr. Global and local deformations of solid primitives. *SIGGRAPH Comput. Graph.*, 18(3):21–30, 1984.

15. Thomas W. Sederberg and Scott R. Parry. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph.*, 20(4):151–160, 1986.

16. Brett Allen, Brian Curless, and Zoran Popović. Articulated body deformation from range scan data. *ACM Trans. Graph.*, 21(3):612–619, 2002.

17. Volker Blanz and Thomas Vetter. A morphable model for the synthesis of 3d faces. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 187–194, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

18. R.E. Parent J.R. Kent and W.E. Carlson. Establishing correspondences by topological merging: A new approach to 3d shape transformation. pages 271–278, San Francisco, Calif., June 1991. Graphics Interface, Morgan Kaufmann.

19. Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In *SIGGRAPH '95: Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 173–182, New York, NY, USA, 1995. ACM.

20. Douglas DeCarlo and Jean Gallier. Topological evolution of surfaces. In *GI '96: Proceedings of the conference on Graphics interface '96*, pages 194–203, Toronto, Ont., Canada, Canada, 1996. Canadian Information Processing Society.

21. Hiromasa Suzuki Takashi Kanai and Fumihiko Kimura. Three-dimensional geometric metamorphosis based on harmonic maps. *The Visual Computer*, 14(4):166–176, October 1998.

22. Marc Alexa. Local control for mesh morphing. In *SMI '01: Proceedings of the International Conference on Shape Modeling & Applications*, page 209, Washington, DC, USA, 2001. IEEE Computer Society.

23. Takashi Kanai, Hiromasa Suzuki, Jun Mitani, and Fumihiko Kimura. Interactive mesh fusion based on local 3d metamorphosis. In *Proceedings of the 1999 conference on Graphics interface '99*, pages 148–156, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

24. R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *IPL*, 1:132–133, 1972.

25. Franco P. Preparata and Michael Ian Shamos. *Computational Geometry - An Introduction*. Springer, 1985.

26. Joseph O'Rourke. *Computational Geometry in C*, chapter 3, "Convex Hulls in 2D". Cambridge University Press, New York, NY, USA, 1998.

27. Bruce G. Baumgart. Winged edge polyhedron representation. Technical report, Stanford, CA, USA, 1972.

28. Francis Lazarus and Anne Verroust. Metamorphosis of cylinder-like objects. *The Journal of Visualization and Computer Animation*, 8(3):131–146, 1997. ISSN 1049-8907.

29. Glassner A.S. Space subdivision for fast ray tracing. *Computer Graphics and Applications, IEEE*, 4(10):15–22, Oct. 1984.

30. Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of graphics tools*, 2(1):21–28, 1997.

31. Independent JPEG Group.