On the Human Form:
Efficient acquisition, modeling and manipulation of the human body

by

Otávio de Pinho Forin Braga

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

New York University

January, 2014

Davi Geiger

Dedication

To my wife, Gina, and to my beautiful baby daughter Elena, who never made it so clear that it was time to wrap it up and graduate.

Acknowledgments

First, I would like to thank my advisor, Davi Geiger, for his support throughout the years and for coming along with encouragement to work on whatever problems motivated me. Above all, it's been great to be able trust him as a friend, and to have his help in finding the comfort of home in an initially completely foreign environment.

I would also like to thank Chris Bregler, for valuable discussions, as well as for helping with necessary equipment whenever asked. I also thank him for providing me the data used in my research.

I'd like to thank the many members of the lab which I had the pleasure to meet throughout the years, in particular my office mate Ian Spiro and Jonathan Thompson, for volunteering to make the silly poses required to generate some of the data.

Lastly, I'd like to thank my wife, Gina, without whose love and unconditional support none of this would have been possible.

Abstract

This thesis concerns the acquisition, modeling and manipulation of the human form.

First, we acquire body models. We introduce an efficient *bootstraped* algorithm that we employed to register over 2,000 high resolution body scans of male and female adult subjects. Our algorithm outputs not only the traditional vertex correspondences, but also directly produces a high quality model which can be immediately deformed. We then employ the result to fit noisy depth maps coming from now commercially available 3D sensors such as Microsoft's Kinect and PrimeSense's Carmine.

We then switch focus to the topic of body manipulation. We first revisit the more traditional way of specifying bodies from a set of measurements, such as coming from clothing sizing charts, showing how the statistics of the population learned during the registration can aid us in accurately defining the body shape. We then introduce a new manipulation metaphor, where we navigate through the space of body shapes and poses by directly dragging the body mesh surface.

We conclude by describing a new real-time system for image-based body manipulation called *BodyJam*, that lets you change your outfit with a finger snap. *Body-*

ABSTRACT

Jam is inspired by a technique invented by the surrealists a century ago: "Exquisite Corpse", a method by which a collection of images (of body parts) is collectively assembled. BodyJam does it on a video display that mirrors the pose in real-time of a real-person standing in front of the camera/display mirror, and allows the user to change clothes and other appearance attributes. Using Microsoft's Kinect, poses are matched to a video database of different torsos and legs, and "pages" showing different clothes are turned by hand gestures.

Table of contents

TABLE OF CONTENTS

List of Figures

List of Figures

List of Figures

List of Tables

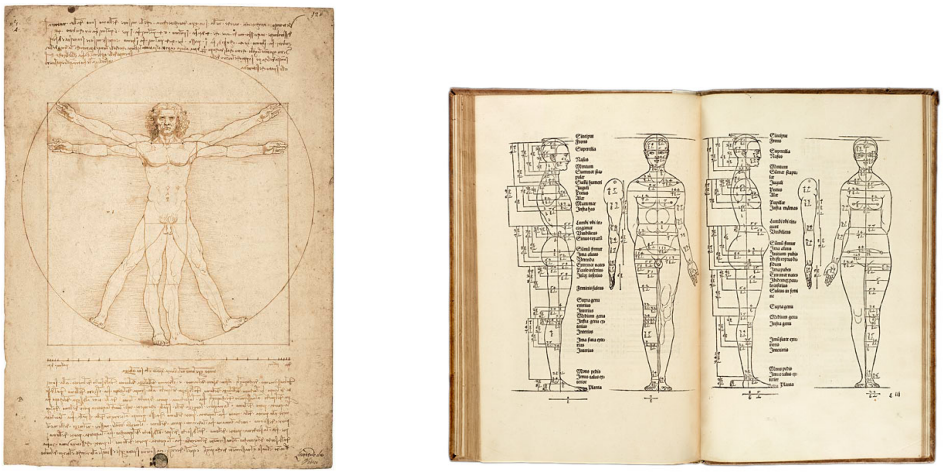
Notations

$\mathbf{x}^H$	$(\mathbf{x}, 1)^T$
$[\mathbf{u}]^\wedge, \widehat{\mathbf{u}}$	The skew-symmetric matrix in $\mathfrak{so}(3)$ associated with the vector $\mathbf{u}$
$\boldsymbol{\omega}^\vee, \breve{\boldsymbol{\omega}}$	The vector associated with the skew-symmetric matrix $\boldsymbol{\omega} \in \mathfrak{so}(3)$
n_V, V	Number of mesh vertices
n_T, T	Number of mesh triangles
n_E	Number of triangle edges
n_J	Number of skeleton joints
$\mathbf{e}_{k,j}$	Model edge of triangle k, $j = 1, 2$
$\overline{\mathbf{e}}_{k,j}$	Template edge of triangle k, $j = 1, 2$
$\boldsymbol{\beta}$	Person specific body shape parameters
$\boldsymbol{\omega}$	Body pose parameters
$p(k)$	Rigid body part of triangle k
$R_{p(k)}(\boldsymbol{\omega})$	Rotation of rigid body part $p(k)$ of triangle k
$S_k(\boldsymbol{\beta})$	Body shape variation matrix of triangle k
$Q_k(\boldsymbol{\omega})$	Pose dependent triangle deformation matrix
$X(\boldsymbol{\omega})$	Body vertex positions matrix
$X_i(\boldsymbol{\omega})$	i-th body vertex positions shape basis matrix
$\overline{E}'$	$n_E \times 3$ matrix of transformed template edges
$\overline{E_i}'$	Shape basis X_i transformed template edges matrix
$\overline{\mathbf{e}'}^i_{k,j}$	i-shape basis transformed template edges of triangle k, $j = 1, 2$

Introduction

The study of the human form played a central role during the Renaissance, where artists such as Leonardo da Vinci and Albrecht Dürer undertook detailed studies on the proportions of the human body, greatly influenced by the classical work of the Roman architect Vitruvius (Pollio n.d.). This influence is summarized in the famous illustration, *Vitruvian Man*, by da Vinci (figure 5.1a), which has now become part of popular culture.

Dürer, the famous German painter and printmaker, in particular, wrote an entire treatise on the human form, *Four Books on Human Proportion* (Dürer 1528), published posthumously through the efforts of his wife Agnes and close friend Willibald Pirckheimer. The first two books are mainly concerned about constructions on ideal human proportions, basely largely on Vitruvius and on his own observations. The third book is particularly interesting, where Dürer describes how the ideal proportions presented before in the first two books can be altered and changed, with the parts being lengthened and shortened to become different than the original (see figure 0.2). He details geometric constructions with which the figure can be systematically deformed. Interestingly, this is a departure from the search of ideal proportions, and beauty now comes from the whole of the



(a) (b)

Figure 0.1: Study of the human form during the Renaissance. (a) Vitruvian Man, by Leonardo da Vinci (c.1485-90) (b) Albrecht Dürer's *Four Books on Human Proportion* (1528). Illustration from the first book.

population, instead of an ideal intentioned by God. He once wrote:

> "If you wish to make a beautiful human figure, it is necessary that you probe the nature and proportions of many people: a head from one; a breast, arm, leg from another..."

Dürer then includes specific methods to modify the human face, describing how to alter the proportions of the features of the head, and warns on how pushing them to the extremes brings us into the grotesque (see figure 0.3a). At this point the connection with da Vinci is clear. Leonardo da Vinci was fascinated with the grotesque, and was constantly sketching deformed, ugly faces, in contrast to his more "divine" paintings celebrated in the 20th century (figure 0.3b).

On a modern setting, anthropometry, or measurements of the human body, is a science by itself, which applications to clothing design, industrial design and

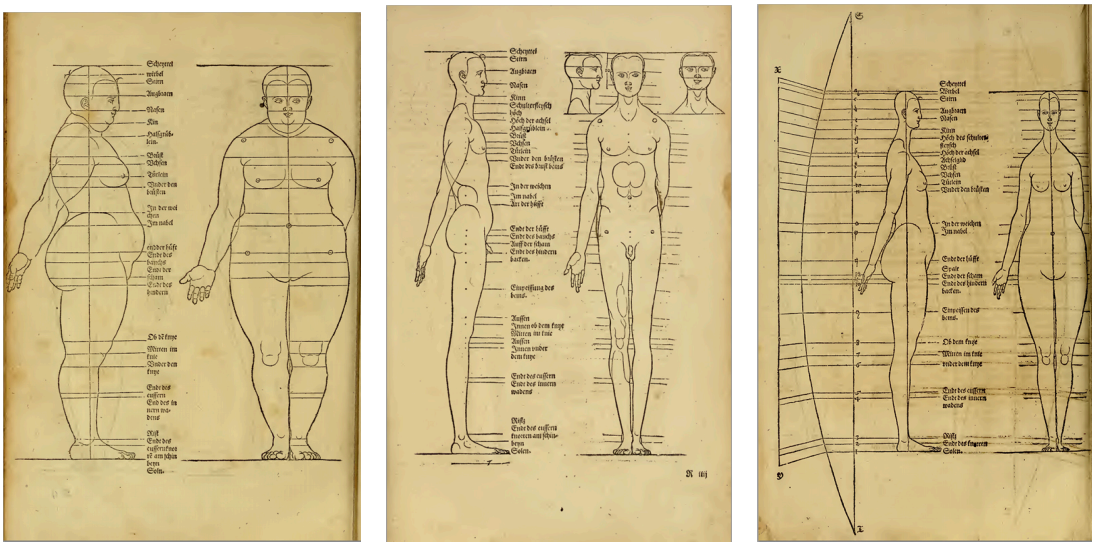


Figure 0.2: Albrecht Dürer's *Four Books on Human Proportion*. Illustrations from the third book, on deformations of the human form.

ergonomics. With the rise of mass production in the 20th century, it became of paramount importance to understand the statistics of human body variation in order to optimize the production. In clothing design, for instance, ready-to-wear clothing sizes are designed to cover the variation of body types of the brand's average customers, while the typically more expensive made-to-measure and bespoke garments are designed in accordance with the measurements of a specific person.

Recently, however, with the advance of manufacturing technologies, such as 3D printers and fabric cutting technology, we are entering a new area of *mass personalization*, were it is becoming increasingly cost effective to create products customized to a specific person. Therefore, efficient algorithms for measuring and manipulating the human form are growing in importance. Moreover, relatively inexpensive 3D sensors such as Microsoft's Kinect, and hand held devices equipped

(a) (b)

Figure 0.3: Deforming the human face into the grotesque. (a) Dürer's head transformations. Illustration from the third book of *Four Books on Human Proportion*. (b) *Study of five grotesque heads*, by Leonardo da Vinci (Milan, c. 1494). Royal Collection, Windsor Castle, London, UK.

with multiple cameras are starting to populate the consumer market, making them a more than ever valuable source of data to understand the body.

On this thesis we focus on the efficient acquisition, modeling and manipulation of the human form, in a way revisiting Dürer's ideas from the perspective of the 21st century. Our specific contributions are:

- An efficient *bootstrapped registration* algorithm used to register over 2,400 high resolution body scans from the CAESAR body scanning dataset. Our algorithm differs from traditional registration algorithms by directly learning inside the body model space in additional to only computing the vertex corre-

spondences, using the learned deformation of the body as a strong regularizer during the optimization.

- We then present a fast algorithm to fit the learned body model to noisy depth maps, improving on comparable published methods (Weiss et al. 2011) from about an hour of optimization to just a few seconds.

- Slightly changing gears, we introduce a new, direct body manipulation technique, where body shapes and poses are specified by directly dragging the vertices of the body mesh. We show how this technique can be employed to directly manipulate body images and point clouds of human bodies

- Finally, still on the topic of body manipulation, we conclude by showing a real-time video-base body manipulation system used to puppeteer video recordings of another person

Chapter 1

Human Body Model

We begin by reviewing the human body model used throughout this work, as well as describing important practical observations and calculations that will be crucial to efficiently manipulate the model.

1.1 Related Work

Human Body Models On the faster-but-less-realistic side of the spectrum, human characters are modeled with pure geometric priors, such as in linear blend skinning (LBS) and related methods (Kavan et al. 2008), which lack in realism but can be computed very efficiently for real-time applications. Skin weights are either painted manually by artists, or automatically computed by methods such as (Baran et al. 2007).

Pose space deformation (Lewis et al. 2000) tightens the modeling and animation evaluation loop, where vertex positions driven by generic parameters are interpo-

lated using radial basis functions. Similarly, shape-by-example (Sloan et al. 2001) interpolates a set of example meshes by computing cardinal basis functions. Even though these methods are not necessarily specific to human bodies, we cite them as representatives of methods used in systems that model the human body with a human artist in the loop.

Pushing the number of examples to the extreme, on the other side of the spectrum is the class of methods that try to automatically learn the space of human bodies from a greater set of examples. Allen et al. 2003 learns a morphable model from examples obtained from high resolution body scans, and (Anguelov, Srinivasan, Koller, et al. 2005) builds on this model to also allow for pose deformation, with further extensions in (Hasler et al. 2009). We draw from this class of models for the current work.

1.2 SCAPE

In this work we use the SCAPE (Anguelov, Srinivasan, Koller, et al. 2005) model to represent human bodies, which we review in this section. Please refer to the original paper for the details.

On the SCAPE model, a point in the space of human bodies is represented by a set of body shape ($\boldsymbol{\beta}$) and pose ($\boldsymbol{\omega}$) parameters. The model itself is a triangle mesh, with vertex positions in a given configuration specified differentially by setting, in the least squares sense, the two edges of each triangle $k = 1, \ldots, n_T$ to

$$\mathbf{e}_{k,j} = R_{p(k)}(\boldsymbol{\omega}) S_k(\boldsymbol{\beta}) Q_k(\boldsymbol{\omega}) \bar{\mathbf{e}}_{k,j}, \qquad j = 1, 2, \tag{1.1}$$

where the components are:

Current Model and Template Mesh Edges: $\mathbf{e}_{k,j} = \mathbf{x}_{k,j} - \mathbf{x}_{k,0}$ is the edge in the current configuration $(\boldsymbol{\beta}, \boldsymbol{\omega})$ and $\overline{\mathbf{e}}_{k,j}$ is the corresponding edge in a template mesh used as reference to train the model.

Rigid Body Parts Rotations: $R_{p(k)} \in SO(3)$ is the rotation of the body part $p(k)$ to which triangle k belongs. One can think of it as purely geometric prior, defined directly by the pose.

Body Shape Variation Matrices: S_k is a 3×3 matrix used to account for body shape variation between individuals, learned from scans of different people in roughly the same pose. It is computed as linear subspace using PCA for all the sample mesh deformations and, in terms of the shape parameters $\boldsymbol{\beta}$, is given by $S_k(\boldsymbol{\beta}) = S_k^0 + \sum_i \beta_i S_k^i$. Stacking up vertically the matrices for all triangles reshaped into column vectors, we can write

$$\mathbf{S}(\boldsymbol{\beta}) = \mathbf{S}^0 + \sum_i \beta_i \mathbf{S}^i. \tag{1.2}$$

In the next chapter we present the details of the technique we developed in order to learn the shape deformation matrices.

Pose Dependent Triangle Deformation Matrices: The remaining 3×3 matrix, Q_k, accounts for triangle shape variations between poses. It is learned for each triangle from a set of registered scans of one single individual in various poses. For a given pose $\boldsymbol{\omega}$, Q_k is computed as an linear function of the exponential coordinates of the rotations of the joints adjacent to the body part to which the triangle belongs. For instance, for a triangle on the biceps, Q_k would be computed as a linear function of the exponential coordinates of the elbow and shoulder rotations.

The linear function is learned for each triangle from the training data. Intuitively, after the rotation of the body part is applied to a triangle one can think of the pose dependent deformation matrices Q_k as learned to act on top of the underlying deformation R_k trying to "fix" the triangle to match the examples and account for what remains in the deformation. Despite its simplicity, this linear function works surprisingly well in practice.

With all the components described, note that 1.1 defines the final vertex positions in the least squares sense, and one must solve for the 3 components of the vertex positions. The positions are defined up to a translation, so we can, say, fix one vertex position, and solve the system for the remaining vertices.

After this quick review of the SCAPE model, we add a couple of our own remarks that will be important for the current work when manipulating the model.

1.3 Real-time Model Evaluation

We first make the observation that the most costly part of evaluating the model is solving the sparse least squares system defined by 1.1. However, note that only the right hand side of equation 1.1 changes with the model parameters, so the system can be decomposed offline and solved in real-time with only forward and back substitution.

More precisely, if we stack vertically the vertex coordinates as rows of a $n_V \times 3$ matrix X, we can rewrite the linear system defined by 1.1 as

$$MX = \overline{E}'(\boldsymbol{\omega}, \boldsymbol{\beta}), \tag{1.3}$$

where M is the $n_E \times n_V$ sparse matrix transforming the mesh vertices into edges, where n_E is the number of edges, and $\overline{E}'$ is the $n_E \times 3$ matrix of *transformed template edges* defined by the right hand side of 1.1, whose solution is given by the normal equation

$$X = (M^T M)^{-1} M^T \overline{E}'(\boldsymbol{\omega}, \boldsymbol{\beta}). \tag{1.4}$$

In our implementation, we prefactor the system above using sparse Cholesky factorization (Yanqing Chen et al. 2008), so that, whenever the model parameters change, we only have to compute the right hand side of 1.3 and solve the system by forward and back substitution, which makes the model practical for interactive applications.

1.4 Hierarchical Rotations

Slightly different from the original SCAPE model, we represent the rotation of each body part with a skeleton joint hierarchy, instead of directly parameterizing the rotations $R_{p(k)}$ in 1.1 of each body part in the root coordinate system. This is useful in the optimization since the relative joint rotations are encoded directly in the parameterization, instead of having to be imposed as extra regularization constraints. We automatically insert a skeleton on the template mesh using (Baran et al. 2007).

To fix the notation, the rotation of each body part $R_{p(k)}$ then is written in

terms of the chain of joints influencing $p(k)$ as a forward kinematics equation

$$R_{p(k)} = R^G_{k_n} = R^l_{k_0} R^0_{k_0 k_1} \dots R^l_{k_i} R^0_{k_i k_{i+1}} \dots R^l_{k_n}, \tag{1.5}$$

where $R^0_{k_i k_{i+1}}$ is the relative rotation between joints k_i and a child joint k_{i+1} in the rest pose, and $R^l_{k_i}$ is the local rotation of joint k_i. $R^G_{k_n}$ is used to denote the accumulated global rotation of joint k_n.

Going back to 1.5, in coordinates it can thus be written as

$$R_{p(k)} = e^{\widehat{\omega}_{k_0}} R^0_{k_0 k_1} \dots e^{\widehat{\omega}_{k_i}} R^0_{k_i k_{i+1}} \dots e^{\widehat{\omega}_{k_n}} \tag{1.6}$$

and $\boldsymbol{\omega} = (\omega_i)_{i=1,\dots,n_j}$ then becomes the model pose parameters.

1.5 Efficient Optimizations with Respect to the Body Model Parameters

During optimization, having analytical expressions for the Jacobians of the objective function in hands generally results in algorithms with orders of magnitude faster than relying on numerical estimations of the Jacobians with finite differences. This is particularly true when function evaluations are expensive, which is the case when working with objective functions depending on the SCAPE model. Each evaluation of the model involves costly substeps, including solving a sparse linear system 1.3 after going through all the prior steps necessary to update its right hand side we have described. In this section we derive the analytical expressions for the Jacobians of the model vertex positions with respect to the shape and pose parameters.

We also describe an optimization strategy on pose space, which will be a recurring problem in the following chapters, so we factor out the main results in this section.

1.5.1 Shape Parameters

By equation 1.3, the model vertex positions depend linearly on the transformed template edges, which, in turn, depend linearly on the shape parameters. Therefore, for a fixed pose, the Jacobian matrix with respect to the shape parameters is constant. Combining equations 1.1, 1.2 and 1.3, we can express the final vertex positions during shape manipulation as a linear combination of basis X_i:

$$X = X_0 + \sum_i \beta_i X_i(\boldsymbol{\omega}), \tag{1.7}$$

and the Jacobian matrix with the respect to the pose parameters can be computed by concatenating the columns X_i:

$$\partial_{\boldsymbol{\beta}} X = (X_1, \ldots, X_i, \ldots, X_n). \tag{1.8}$$

The basis X_i are obtained by solving, for each shape parameter β_i,

$$X_i = (M^T M)^{-1} M^T \overline{E_i}'(\boldsymbol{\omega}), \tag{1.9}$$

with $\overline{E_i}'$ defined in a similar way to $\overline{E}'$ in 1.3, this time with the *transformed template edges* computed with respect to a single component of the shape deformation matrix. The transformed edges forming the rows of $\overline{E_i}'$ are given by

$$\overline{\mathbf{e}'}^i_{k,j} = R_{p(k)}(\boldsymbol{\omega}) S^i_k Q_k(\boldsymbol{\omega}) \overline{\mathbf{e}}_{k,j}, \qquad j = 1, 2, \tag{1.10}$$

where index i corresponds to the index of the shape component, the index k, for the triangle, and j, for the edge on triangle k.

Combining 1.8 and 1.9, we can summarize the calculations for the Jacobian as

$$\partial_{\boldsymbol{\beta}} X = (M^T M)^{-1} M^T (\overline{E_1}', \ldots, \overline{E_i}', \ldots, \overline{E_n}')(\boldsymbol{\omega}), \tag{1.11}$$

which can again be solved with the sparse Cholesky factorization computed offline, with multiple simultaneous right hand sides given by the transformed template edges $\overline{E_i}'$. Moreover, since X_i depends only on the pose parameters, the basis have to be computed only once for a fixed pose.

1.5.2 Pose Parameters

Optimizations with respect to the pose parameters become more complicated due the nonlinearity in the rotation optimization. We want an efficient formulation, like in the previous section, that makes use of the prefactorization of the system 1.4. Unlike traditional optimizations on the Euclidean space, which would compute the updates on each step in the global parameter space, we optimize here the incremental steps directly on the rotation group manifold (or, more precisely, the Cartesian product manifold $SO(3) \times \cdots \times SO(3) = SO(3)^{n_j}$, for n_j joints).

Specifically, at each step we solve for incremental rotations, ΔR_i, for the rotation of joint i, and the updates are given by the right translations $R_i^l \leftarrow R_i^l \Delta R_i$. In coordinates, the updates can be written as

$$\widehat{\omega_i} \leftarrow \widehat{\omega_i}' = \log(R_i^l \exp(\widehat{\Delta\omega_i})) = \log(\exp(\widehat{\omega_i}) \exp(\widehat{\Delta\omega_i})), \tag{1.12}$$

and we seek the local parameters $\Delta\omega_i$'s that minimize our energy in each step. The optimization, thus, corresponds to Gauss-Newton steps (or, Levenberg-Marquardt steps) on the manifold.

When optimizing specifically on the pose parameters of the SCAPE model, we first use the observation from (Anguelov, Srinivasan, Koller, et al. 2005) that most of the variation in the model shape is due to the geometric prior R_k, so we can keep Q_k fixed at each iteration. The transformed template edges on 1.10 simplify at each step to

$$\overline{\mathbf{e}'}_{k,j} = R_{p(k)}(\boldsymbol{\omega}')S_k Q_k(\boldsymbol{\omega})\overline{\mathbf{e}}_{k,j} = R'_{p(k)}\tilde{\mathbf{e}}_{k,j} = R'^{G}_{k_n}\tilde{\mathbf{e}}_{k,j}, \tag{1.13}$$

where $\tilde{\mathbf{e}}_{k,j}$ now denotes the *locally deformed template edge* $\overline{\mathbf{e}}_{k,j}$, before the body part rotation (and, therefore, constant during an iteration step).

Incorporating the forward kinematics equation 1.5 and the incremental rotation in the equation above we arrive at

$$\overline{\mathbf{e}'}_{k,j} = R^l_{k_0} R^0_{k_0 k_1} R^l_{k_1} \dots R^l_{k_i} R^0_{k_i k_{i+1}} \dots R^l_{k_n} \tilde{\mathbf{e}}_{k,j}. \tag{1.14}$$

Computing the directional (Lie) derivatives in the direction of our rotation

increments for each joint, with some algebraic manipulation we have

$$\begin{aligned}
\partial_{\Delta\omega_i}\overline{\mathbf{e}'}_{k,j}\big|_{\Delta\boldsymbol{\omega}=0} &= R^l_{k_0}R^0_{k_0k_1}R^l_{k_1}\dots R^l_{k_i}\widehat{\Delta\omega_{k_i}}R^0_{k_ik_{i+1}}\dots R^l_{k_n}\tilde{\mathbf{e}}_{k,j} \\
&= R^G_{k_i}\widehat{\Delta\omega_{k_i}}\left(R^G_{k_i}\right)^{-1}R^G_{k_i}R^0_{k_ik_{i+1}}\dots R^l_{k_n}\tilde{\mathbf{e}}_{k,j} \\
&= R^G_{k_i}\widehat{\Delta\omega_{k_i}}\left(R^G_{k_i}\right)^{-1}R^G_{k_n}\tilde{\mathbf{e}}_{k,j} \\
&= \left[R^G_{k_i}\Delta\omega_{k_i}\right]^\wedge\tilde{\mathbf{e}}^G_{k,j} \\
&= -\left[\tilde{\mathbf{e}}^G_{k,j}\right]^\wedge R^G_{k_i}\Delta\omega_{k_i} \\
&= \left(-\left[\tilde{\mathbf{e}}^G_{k,j}\right]^\wedge R^G_{k_i}\right)\Delta\omega_{k_i}.
\end{aligned} \tag{1.15}$$

Now, note that $\tilde{\mathbf{e}}^G_{k,j} = R_{p(k)}S_k(\boldsymbol{\beta})Q_k(\boldsymbol{\omega})\overline{\mathbf{e}}_{k,j}$ is exactly the *transformed template edges* computed at the current pose, which we have already computed in order to evaluate the model at the current pose. The term inside the parenthesis is thus precisely the Jacobian expressed in the canonical local parameterization we are looking for, which resembles the manipulator Jacobian for inverse kinematics.

With the Jacobian of the transformed template edges with respect to the differential pose parameters in hands, we only have to include the dependency from the edges of the actual vertex positions in order to obtain the Jacobian of the vertex positions with respect to the pose parameters. From 1.4,

$$\begin{aligned}
\partial_{\Delta\omega}X &= \partial_{\Delta\omega}\left[(M^TM)^{-1}M^T\overline{E}'(\boldsymbol{\omega},\boldsymbol{\beta})\right] \\
&= (M^TM)^{-1}M^T\partial_{\Delta\omega}\overline{E}'(\boldsymbol{\omega},\boldsymbol{\beta}),
\end{aligned} \tag{1.16}$$

where $\partial_{\Delta\omega}\overline{E}'(\boldsymbol{\omega},\boldsymbol{\beta})$ is given by 1.15, and $\partial_{\Delta\omega}X$ can be efficiently computed again with the Cholesky factorization we have computed offline.

With the Jacobian matrix, we can perform the local optimization in the differential rotations parameters $\Delta\omega$, say, using Levenberg-Marquardt, and update the local joint rotations with $R_k^l \leftarrow R_k^l \exp(\widehat{\Delta\omega_k})$ on each accepted iteration step.

1.6 Biased Levenberg-Marquardt Pose Optimization

We conclude this chapter by presenting a technique for biasing iterative optimizations with respect to the model pose parameters towards solutions that pass closer to the sample poses. At each step, we work by locally modifying the metric to introduce the bias, and our algorithm fits nicely withing a typical Levenberg-Marquardt (LVMA) or Gauss-Newton steps optimization framework.

This will be useful when optimizing with respect to the pose parameters, where not only we want solutions that minimize the objective function, but also that somehow lie close to the region of the domain where the model is well defined.

Related Work. The technique described in this section can be compared to (Grochow et al. 2004), where the authors apply dimensionality reduction techniques to construct a gaussian process latent variable model to learn a lower dimensional representation from a large set of motion capture data. Our technique differs in an important way though. There, the authors learn the GPLVM from long sequences of complete motions as training data, modeling specific motion styles (running, throwing a pitch, etc…), while here our data contains only a very sparse set of isolated poses, not representing any motion in particular, chosen only to cover the

deformations of the body parts.

In order to fix the notation, we first we review the standard LVMA algorithm, then proceed to describe the details of our biased pose optimization. Refer to (Nocedal et al. 2006; Press et al. 2007; K. Madsen April 2004) for more details.

1.6.1 The Levenberg-Marquardt Algorithm

Say we want to minimize a least squares energy

$$\min E(\boldsymbol{\omega}) = ||\mathbf{F}(\boldsymbol{\omega}) - \mathbf{y}||^2 = ||\mathbf{r}(\boldsymbol{\omega})||^2, \tag{1.17}$$

where $\mathbf{F}(\boldsymbol{\omega})$ is some function of the model pose parameters, such as, for example, some subset of skeleton joint position in an inverse kinematics problem (and where $\mathbf{y}$ would be the target positions).

In a typical LVMA iteration, we linearize $\mathbf{F}$ around the current solution $\boldsymbol{\omega}$:

$$\mathbf{F}(\boldsymbol{\omega} + \Delta\boldsymbol{\omega}) \approx \mathbf{F}(\boldsymbol{\omega}) + J(\boldsymbol{\omega})\Delta\boldsymbol{\omega} \tag{1.18}$$

and the step of each iteration is the solution to the linear system

$$(J^T J + \lambda D^T D)\Delta\boldsymbol{\omega} = J^T(\mathbf{y} - \mathbf{F}(\boldsymbol{\omega})). \tag{1.19}$$

where D is a diagonal matrix. In Levenberg's original algorithm (Levenberg 1944), D is set to the identity matrix. λ is the damping parameter, which weights how much the step direction deviates from pure Gauss-Newton steps towards the gradient descent direction. When λ is small, we have $J^T J\Delta\boldsymbol{\omega} \approx J^T(\mathbf{y} - \mathbf{F}(\boldsymbol{\omega}))$, which corresponds to Gauss-Newton steps. On the other hand, when λ is large, we have

$\Delta\boldsymbol{\omega} \propto J^T(\mathbf{y} - \mathbf{F}(\boldsymbol{\omega}))$, which is exactly the gradient descent direction. Around minima, where gradient descent performs poorly, we use the Hessian approximation $J^T J$ to locally approximate $\mathbf{F}$ by a quadratic function, and the Gauss-Newton step corresponds to moving towards the minima of the approximation. On the other hand, far from the minima, where the quadratic approximation is poor, we move with gradient descent steps. The contribution from Marquardt (Marquardt 1963) was to notice that, even far from the minima, we can use information from the Hessian approximation in order to make use of estimated local curvature. Marquardt replaced the identity matrix of Levenberg's algorithm by the diagonal of the Hessian $D^T D = \text{diag}(J^T J)$, which scales the gradient descent direction to move further in the directions where the gradient is smaller in order to circumvent the classic zigzagging on narrow valleys problem of gradient descent.

It is useful to rewrite the iteration above as the solution to the least squares system

$$\begin{pmatrix} J \\ \sqrt{\lambda} D \end{pmatrix} \Delta\boldsymbol{\omega} = \begin{pmatrix} \mathbf{y} - \mathbf{F}(\boldsymbol{\omega}) \\ \mathbf{0} \end{pmatrix}. \tag{1.20}$$

Or, equivalently, another way of looking at the damping term is that at each step of the optimization we are actually solving the linear least squares problem

$$\min_{\Delta\boldsymbol{\omega}} ||\mathbf{F}(\boldsymbol{\omega}) + J(\boldsymbol{\omega})\Delta\boldsymbol{\omega} - \mathbf{y}||^2 + \lambda ||D\Delta\boldsymbol{\omega}||^2, \tag{1.21}$$

and the damping shows up as a Tikhonov regularization term. The damping parameter λ is adjusted at each iteration to control the trust region of where the

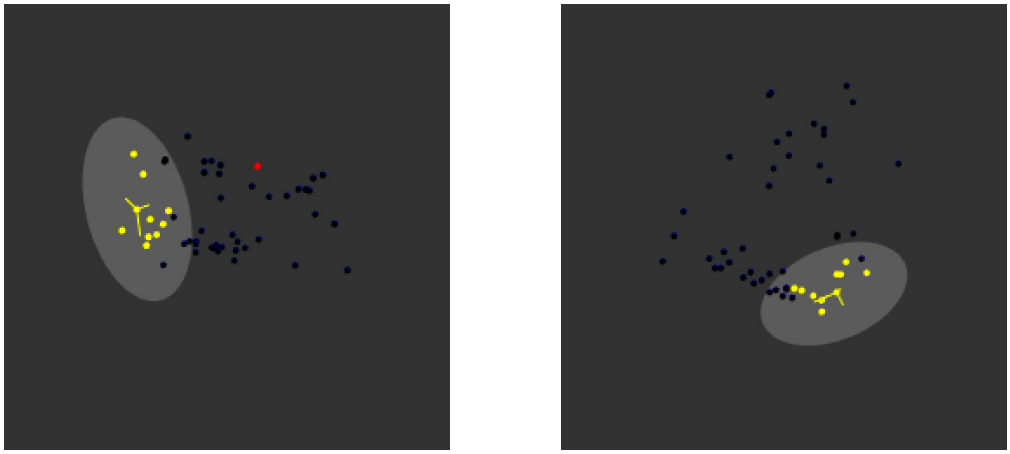


Figure 1.1: Local metrics for the right shoulder and elbow during an inverse kinematics manipulation. The points in yellow correspond to the k nearest neighbors, and the ellipses illustrate the local covariance matrix. The 2D projection of the 3D coordinates is chosen along the 2 principal directions of the whole sample set.

linear approximation is good. Large λ imposes a smaller radius, and small λ, less weight of the regularization, allowing a larger radius for $\Delta\boldsymbol{\omega}$.

1.6.2 Local Joint Rotation Metric

Now, we show how to bias the iteration with a metric induced by the sample poses. To simplify the discussion, we consider only one isolated joint first, and the extension to multiple joints is straightforward. Say that the sample data contains poses $R_1, \ldots, R_n$, $R_i \in SO(3)$, as sample rotations of the joint. During the optimization, if the current solution is $\boldsymbol{\omega}$, with the orientation of the joint in question being R_0, we first find the k nearest neighbors to R_0 using the Riemannian

metric

$$d(R_0, R_i) = \frac{1}{\sqrt{2}} || \log (R_0^T R_i) ||_F = ||\Delta \boldsymbol{\omega}_i||, \tag{1.22}$$

where $\Delta \boldsymbol{\omega}_i = \log(R_0^T R_i)^\vee$ corresponds to the exponential coordinates of the rotation along the geodesic from R_0 to R_i, given by $g(t) = R_0 \exp(t\widehat{\Delta \boldsymbol{\omega}_i})$. Without loss of generality, we will assume the k nearest neighbors are $\Delta\omega_1, \ldots, \Delta\omega_k$.

Next, we compute a local metric based on the Mahalanobis metric induced by the neighboring rotations. We start by computing the covariance matrix of the sample local rotations

$$\Sigma_j = \frac{1}{k} \sum_{i=1}^{k} \Delta \boldsymbol{\omega_i} \Delta \boldsymbol{\omega_i}^T, \tag{1.23}$$

and then the precision matrix $\Sigma_j^{-1} = \Sigma_j$, which will actually induce the metric. Figure 1.1 illustrates the local metric for the right shoulder and elbow during an inverse kinematics manipulation.

1.6.3 Biased Iterations

Now we show how to bias the LVMA algorithm iterations based on the local joint rotation metrics.

We first compute the local metric for each joint, Σ_i^{-1}, and then normalize the metrics for the different joints as follows. We start with the eigendecomposition of the precision matrix

$$\Sigma_i^{-1} = U \Lambda U^T, \tag{1.24}$$

where U is the 3×3 matrix of eigenvectors in each column, and $\Lambda = \text{diag}(\lambda_1, \lambda_2, \lambda_3)$, $\lambda_1 > \lambda_2 > \lambda_3 > 0$, is the diagonal matrix of eigenvalues. We scale the principal axis so that the largest eigenvalue is 1,

$$\overline{\Lambda} = \begin{pmatrix} 1 & & \\ & \lambda_2/\lambda_1 & \\ & & \lambda_3/\lambda_1 \end{pmatrix}, \tag{1.25}$$

and reassemble in the new scaled precision matrix $\overline{\Sigma_i}^{-1} = U\overline{\Lambda}U^T$. This normalization is used to ensure that all the joints have roughly the same scale in the bias, and we mainly influence the step direction within each joint.

In order to integrate this local metric into the optimization framework, we then compute the Cholesky decomposition of the scaled precision matrix

$$\overline{\Sigma_i}^{-1} = L_i^T L_i, \tag{1.26}$$

and we formulate a new least squares system (as in the corresponding linear system 1.33)

$$\begin{pmatrix} & & J & & \\ \sqrt{\alpha}L_1 & & & & \\ & \ddots & & & \\ & & \sqrt{\alpha}L_i & & \\ & & & \ddots & \\ & & & & \sqrt{\alpha}L_J \\ & & \sqrt{\lambda}I & & \end{pmatrix} \Delta\boldsymbol{\omega} = \begin{pmatrix} \mathbf{y} - \mathbf{F}(\boldsymbol{\omega}) \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}. \tag{1.27}$$

Abbreviating the diagonal of L_i entries into a block diagonal matrix B, we have

$$\begin{pmatrix} J \\ \sqrt{\alpha}B(\boldsymbol{\omega}) \\ \sqrt{\lambda}D \end{pmatrix} \Delta\boldsymbol{\omega} = \begin{pmatrix} \mathbf{y} - \mathbf{F}(\boldsymbol{\omega}) \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}. \tag{1.28}$$

For comparison with 1.21, the solution of the least squares linear system is the minimum of

$$\min_{\Delta\boldsymbol{\omega}} ||\mathbf{F}(\boldsymbol{\omega}) + J(\boldsymbol{\omega})\Delta\boldsymbol{\omega} - \mathbf{y}||^2 + \alpha||B\Delta\boldsymbol{\omega}||^2 + \lambda||D\Delta\boldsymbol{\omega}||^2, \tag{1.29}$$

where the new Tikhonov regularization term is a block diagonal matrix with the local metrics

$$\Gamma = B^T B = \begin{pmatrix} L_1^T L_1 & & & & \\ & \ddots & & & \\ & & L_i^T L_i & & \\ & & & \ddots & \\ & & & & L_J^T L_J \end{pmatrix} = \begin{pmatrix} \overline{\Sigma_1^{-1}} & & & & \\ & \ddots & & & \\ & & \overline{\Sigma_i^{-1}} & & \\ & & & \ddots & \\ & & & & \overline{\Sigma_J^{-1}} \end{pmatrix}. \tag{1.30}$$

In a typical LVMA implementation, we have to provide methods that evaluate the function and Jacobian at a given parameter. In order to implement the biased scheme above, note that we only have to augment the function evaluation to, instead of returning $\mathbf{y} - \mathbf{F}(\boldsymbol{\omega})$, appending the zeros to the end of the vector corresponding to the bias terms

$$\mathbf{r}'(\boldsymbol{\omega}) = \begin{pmatrix} \mathbf{y} - \mathbf{F}(\boldsymbol{\omega}) \\ \mathbf{0} \end{pmatrix} \tag{1.31}$$

As for the Jacobian, appending the block diagonal matrix

$$J' = \begin{pmatrix} J \\ \sqrt{\alpha}B \end{pmatrix}, \tag{1.32}$$

the optimization thus becomes a standard LVMA formulation

$$\begin{pmatrix} J' \\ \sqrt{\lambda}D \end{pmatrix} \Delta\boldsymbol{\omega} = \begin{pmatrix} \mathbf{r}'(\boldsymbol{\omega}) \\ \mathbf{0} \end{pmatrix} \tag{1.33}$$

(compare to equation 1.33).

1.7 Conclusion

This chapter was a quick summary of the body model we are going to use throughout the following chapters. Besides reviewing prior work, we have presented non-trivial observations that in our experience revealed to be crucial to efficiently manipulate the model. To the best of our knowledge, they are entirely missing in the literature. We also presented an optimization technique in pose space that bias optimization trajectories using local curvature information obtained from the learning data. With this machinery in hands, we can proceed to apply the model to our problems without having to completely rely on black-box optimization solvers.

Chapter 2

Body Registration

2.1 Introduction

In this chapter we present an efficient algorithm used to register the body model of a single person deformable to multiple poses with a high resolution body scan of another person.

A major difference and advantage of our algorithm over methods that produce generic mappings between surfaces (such as Allen et al. 2003; Anguelov, Srinivasan, Pang, et al. 2004; Kim et al. 2011) is that ours produces not only vertex-to-vertex correspondences, but, by learning the correspondences while constrained to the human bodies manifold, we directly produce a full deformable model of the target subject to which we are registering. The result can then be immediately used to place the subject in multiple poses. Not only we avoid the extra step of learning the body model after the registration, but also, and more importantly, we incorporate knowledge of the deformation of the human body to help in establishing

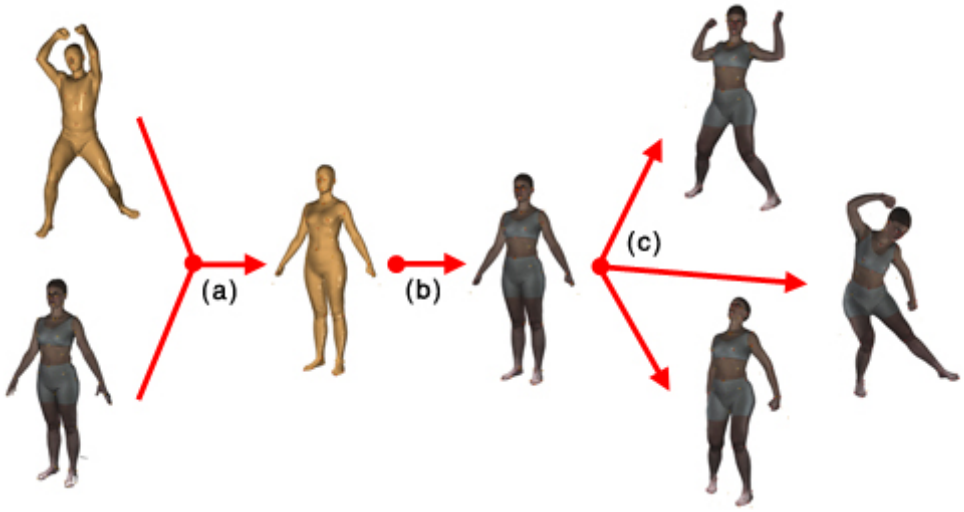


Figure 2.1: Overview of the registration process. (a) The deformable model learned for a single person is registered with a high resolution scan. (b) We then synthesize a texture and normal map for the model learned for that specific person. (c) The textured model can then be deformed into various poses.

the correspondences. Another practical benefit is that we can reuse during the registration all the machinery already implemented to manipulate the body model. On the downside, the registration technique is specific to human body meshes, but we feel that this is a class important enough to deserve dedicated attention.

We applied our algorithm to register over 2000 high resolution body scans from the CAESAR body scanning database, and used the results to learn a general model of the male and female population.

2.2 The CAESAR Body Scanning Database

The Civilian American and European Surface Anthropometry Resource (CAESAR) database is the result of a survey carried out by the U.S. Air Force on a

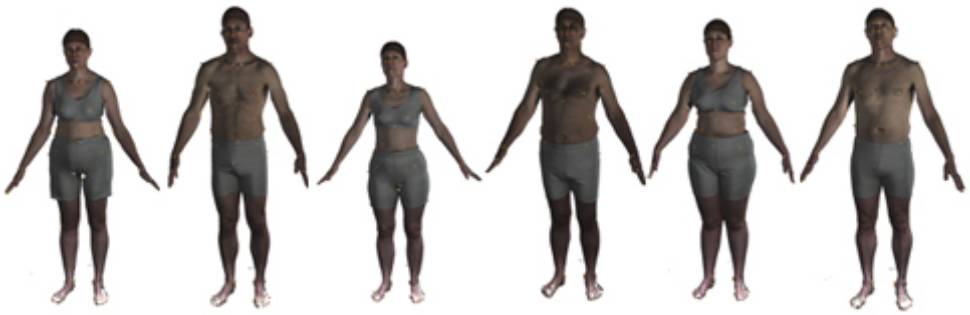


Figure 2.2: High resolution body scans from the CAESAR database.

population sampled from the United States, Canada, The Netherlands and Italy, the last two picked for being the tallest and shortest population in the NATO (North Atlantic Treaty Organization), respectively. It consists of about 2000 body scans of male and female adults, each mesh with approximately 200k vertices and 415k faces. The scans are provided as reconstructed meshes, with colors and a confidence measure as vertex attributes. Figure 2.2 shows six scans from the database.

Each subject was scanned in 3 poses, one standing and two seated poses, although we only used the standing pose for the registration. Moreover, each subject is annotated with 74 markers placed in various body landmarks, and a set of body measurements, such as chest/waist/hips circumference, height, arm length, etc….

In addition to the markers originally present in the database, we added two more markers to improve the registration accuracy: one in the navel and one on the tip of the nose. We also disregarded the markers placed on the ribs [1], since they didn't provide consistent landmarks. Figure 2.4 shows all the markers on a subject.

1. Tenth rib, left and right, in the dataset nomenclature.

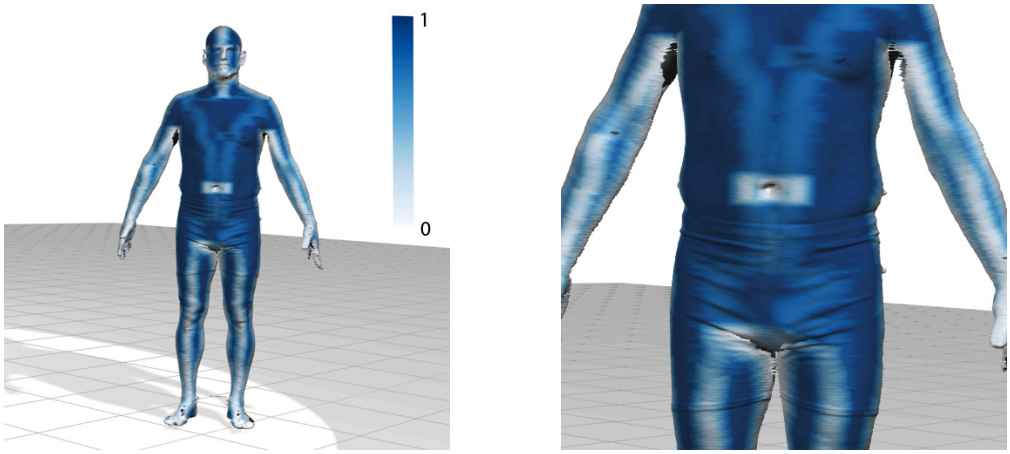


Figure 2.3: Vertex confidences of a CAESAR scan.

2.3 Bootstrapped Registration Algorithm

2.3.1 Overview

Given a high resolution scan of one subject, our goal is to retrieve the pose parameters $\boldsymbol{\omega}$, and the shape deformation matrix of each triangle of the template mesh $\mathbf{S} = \{S_1, \ldots, S_T\}$. Each matrix S_i is a general 3×3 matrix that captures the body shape deformation specific to each person. We alternate the optimization of the pose and shape parameters, using the markers for the pose optimization, and also point correspondences between the template and scan mesh for the shape optimization.

For each vertex $\mathbf{x}_i$, $i = 1, \ldots, V$ of the template mesh, we denote the corresponding closest vertex on the body scan mesh by $\mathbf{y}_i$. The model vertex positions are actually a function of the pose parameters $\boldsymbol{\omega}$, which we denote by $\mathbf{x}_i(\boldsymbol{\omega})$ when we want to make this dependency explicit.

Figure 2.4: Markers on a subject body scan in the standing pose from the CAESAR database.

The set of markers on the template mesh is denoted by $\mathbf{x}_{M_i}, i = 1, \ldots, M$, and the corresponding markers on the scan by $\mathbf{y}_{M_i}$. Note that each marker corresponds to a vertex on the template or target scan mesh, the former manually picked for each CAESAR landmark before running the registrations, and the later found by computing the closest vertex to each landmark position originally present in the CAESAR dataset. Figure 2.5 shows the markers placed on the template model at the rest pose.

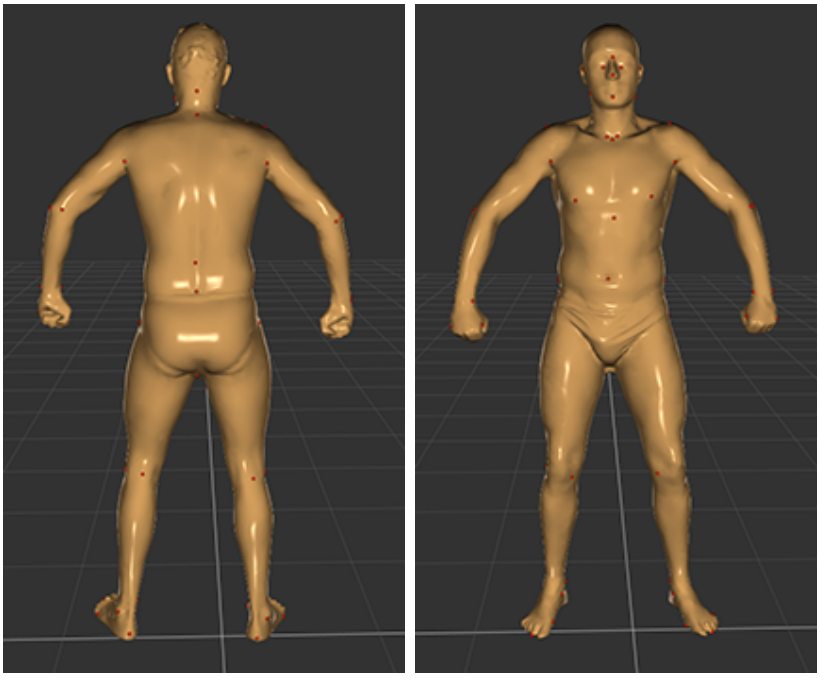


Figure 2.5: Landmarks on the template body model

Algorithm 1 Registration Algorithm

1: **(Initialization)** Initialize the shape matrices S_i, $i = 1, \ldots, T$ and global rigid transformation with the optimal similarity transformation between the markers
2: **while** $E(\boldsymbol{\omega}, \mathbf{S}) > \epsilon$ **do**
3: Solve for the new pose $\boldsymbol{\omega}$
4: Solve for the vertex transformations T_i, $i = 1, \ldots, V$
5: Solve for the triangle transformations S_i, $i = 1, \ldots, T$
6: Factor out the rigid components of the triangle transformations
7: **end while**

2.3.2 Pose Optimization

For the pose optimization, since we had enough precisely placed markers, we found it was enough to minimize the energy defined only by the marker errors

$$E(\boldsymbol{\omega}) = \sum_{i=1}^{M} ||\mathbf{y}_{M_i} - \mathbf{x}_{M_i}(\boldsymbol{\omega})||^2. \tag{2.1}$$

We solve this optimization with Levenberg-Marquardt on the rotation group manifold using the Jacobian matrix we derived analytically in section 1.5.2. More precisely, since $\mathbf{x}_{M_i}$ forms a subset of the model vertices, and the target marker positions $\mathbf{y}_{M_i}$ are constants, the Jacobian of the energy above can be trivially obtained from slices of the Jacobian matrix of the vertex positions with respect to the pose parameters we derived in equation 1.16.

At each step, we solve the local optimization in terms of the differential rotations parameters $\Delta\boldsymbol{\omega}$, and update the local joint rotations with $R_k^l \leftarrow R_k^l \exp(\widehat{\Delta\boldsymbol{\omega}}_k)$ on each accepted step.

2.3.3 Vertex Transformation Optimization

Given the vertex and marker correspondences, we first solve for an affine transformation of each vertex. We want to solve for the 3×4 vertex transformation matrices $\mathbf{T}_i$, $i = 1, \ldots, V$. Letting $\mathbf{x}^H = (\mathbf{x}, 1)^T$ denote the homogeneous vector

associated with the vector $\mathbf{x}$, we minimize the energy

$$E(\mathbf{T}) = w_D \sum_{i=1}^{V} w_i ||\mathbf{y}_i - \mathbf{T}_i \mathbf{x_i}^H||^2 + w_M \sum_{i=1}^{M} ||\mathbf{y}_{M_i} - \mathbf{T}_{M_i} \mathbf{x}_{M_i}^H||^2 + w_S \sum_{(i,j)\in E} ||\mathbf{T}_i - \mathbf{T}_j||_F^2. \tag{2.2}$$

The first term corresponds to the error of each vertex, where the vertex weights $w_i \in [0, 1]$ are set to the vertex confidence provided by the CAESAR dataset. The second term corresponds to the marker error, and the third is a smoothness term between the transformation of adjacent vertices, where $||\cdot||_F$ is the Frobenius norm. $w_{\{D,M,S\}}$ are smoothness weights of each energy term. The transformation matrices are concatenated in a big column vector, and the energy is minimized by solving the corresponding sparse least squares linear system.

We run the overall optimization in 3 stages, according to the following schedule for the weights:

Stage	w_D	w_M	w_S
1	0	10	1
2	1	1	1
3	10	1	1

That is, on the first stage, we take only the markers into account, then give equal weight to the markers and vertex proximity correspondences, then give a larger weight to the vertex correspondences.

This step is similar to (Allen et al. 2003), where the authors resort to a multiresolution strategy in order to solve the optimization, since the energy above by

itself requires the transformations to slowly propagate throughout the whole mesh, and can easily be trapped in local minima.

In our strategy, however, the underlying body deformation model acts as a much stronger geometric prior than the simple smoothness term in the energy above, which not only speeds up the convergence, but also makes the optimization robust to local minima. We solve the optimization directly, without any multiresolution strategies. Moreover, we are ultimately interested in recovering the shape and pose parameters of the model, so the vertex transformation optimization is only an intermediate step for the whole optimization.

2.3.4 Shape Matrix Optimization

After optimizing for the transformation matrix of each *vertex* $\mathbf{T}_i$, we minimize the shape transformation matrix of each *triangle* S_i, $i = 1, \ldots, T$. Let $\tilde{\mathbf{x}_i} = \mathbf{T}_i \mathbf{x}_i^H$ denote the transformed vertex positions, and $\tilde{\mathbf{e}}_{k,j} = \tilde{\mathbf{x}}_{k,j} - \tilde{\mathbf{x}}_{k,0}$, a transformed edge.

Similarly to the previous section, we combine a data and a smoothness term in the energy we want to minimize

$$E(\mathbf{S}) = \sum_{i=1}^{T} \sum_{j=1}^{2} ||\mathbf{S}_i Q_i(\boldsymbol{\omega}) \overline{\mathbf{e}}_{i,j} - R_{p(i)}^T \tilde{\mathbf{e}}_{i,j}||^2 + \alpha \sum_{\text{adj}(i,j)} ||\mathbf{S}_i - \mathbf{S}_j||_F^2, \qquad (2.3)$$

where $\overline{\mathbf{e}}_{i,j}$, $j = 1, 2$ denotes the two template edges of triangle i, and the sum on the second term now is on adjacent triangles instead of vertices. Again, this is a sparse least squares problem, and we solve as in the previous section by prefactoring the sparse system during initialization with Cholesky decomposition and solving the system at each iteration with a different right hand side.

The shape matrices $\mathbf{S}_i$ can then be directly incorporated into the SCAPE model to define one subject in multiple poses.

2.3.5 Factoring Out Rigid Transformations

At this stage, the shape matrices of triangles $\mathbf{S}_i$ in the same body part typically share a rotation component which can be factored out and incorporated into the pose. Therefore, after optimizing for the matrices $\mathbf{S}_i$, we perform an extra step in order to extract such rotation component as follows:

Polar Decomposition. First, we compute the polar decomposition of each matrix $\mathbf{S}_i = \mathbf{U}_i\mathbf{P}_i$, where $\mathbf{U}_i$ is orthogonal and $\mathbf{P}_i$ is positive semi-definite (Shoemake 1994). Then, we want to somehow compute the "average" rotation of each body part, given the orthogonal matrices $\mathbf{U}_i$, so that this common factor can be incorporated as pose instead of shape deformation.

Fréchet Mean on $SO(3)$. One way to define such a mean is to define a metric on $SO(3)$, and compute the Fréchet mean of the given matrices. The Fréchet mean is the point (assuming one exists) that minimizes the sum of distances to the other points:

$$R = \underset{R \in SO(3)}{\operatorname{argmin}} \sum_i d(R, R_i). \tag{2.4}$$

For instance, the arithmetic mean of a set of points minimizes the *absolute difference* distance to the other points $d_E(x, y) = |x - y|$, and the geometric mean minimizes the *hyperbolic distance* $d_H(x, y) = |\log(x) - \log(y)|$.

In $SO(3)$, one natural metric is the Euclidean distance of the ambient space

$$d_E(R_1, R_2) = ||R_1 - R_2||_F. \tag{2.5}$$

Another metric, which better reflects the Riemannian structure of the manifold is defined in terms of the shortest curve that connects the two rotation matrices

$$d_R(R_1, R_2) = \frac{1}{\sqrt{2}}||\log(R_1^T R_2)||_F = ||\log(R_1^T R_2)^\vee||. \tag{2.6}$$

For our purposes, we decided to use the former, Euclidean distance, since it becomes faster and straightforward to minimize 2.4, and we didn't observe significant gains when using the Riemannian distance. With the Euclidean distance 2.5 the solution to 2.4 is simply the polar factor of the arithmetic mean $\overline{R} = \sum R_i/N$ (Moakher 2002).

With the mean rotation matrix of each body part in hands, we factor it out from the shape matrices $\mathbf{S}_i$, and incorporate the rotation matrices in the pose.

2.3.6 Initialization

With the pose fixed, we initialize all matrices $\mathbf{S}_i$ with the scaling of the best similarity transformation $(s, \mathbf{R}, \mathbf{t})$ that brings the markers on the template mesh into alignment with the markers on the subject's scan (figure). The rigid transformation components $(\mathbf{R}, \mathbf{t})$ are used to set the global rigid transformation of the model.

Optimal Similarity Transformation. The optimal similarity transformation $T = (s, \mathbf{R}, \mathbf{t})$ between corresponding point sets $\mathbf{x}_i \leftrightarrow \mathbf{y}_i$ can be computed as follows. Let $\overline{\mathbf{x}}_i = \sum \mathbf{x}_i/N$ and $\overline{\mathbf{y}}_i = \sum \mathbf{y}_i/N$ denote the centroids of the source and target point sets, respectively.

1. Compute the **rotation matrix** from the SVD of the covariance matrix

$$C = \sum (\mathbf{x}_i - \overline{\mathbf{x}}_i)^T (\mathbf{y}_i - \overline{\mathbf{y}}_i)$$
$$C = U\Sigma V^T$$
$$\mathbf{R} = UDV^T, \text{ where } D = \text{diag}(1, \ldots, \det(UV^T))$$

2. Compute the **scale**

$$s = \sqrt{\frac{\sum ||\mathbf{x}_i - \overline{\mathbf{x}}_i||^2}{\sum ||\mathbf{y}_i - \overline{\mathbf{y}}_i||^2}} \tag{2.7}$$

3. Then compute the **translation vector**

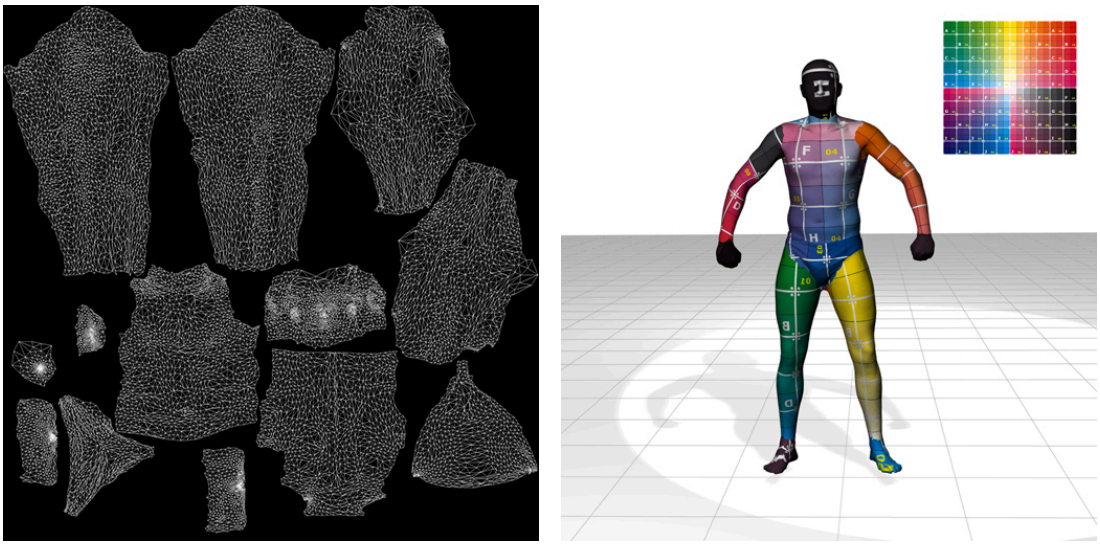
$$\mathbf{t} = \overline{\mathbf{y}}_i - s\mathbf{R}\overline{\mathbf{x}_i}; \tag{2.8}$$

We then begin the iterations of our registration algorithm, starting with the pose optimization step described in section 2.3.2.

2.4 Appearance Synthesis

Normal Mapping: After the registration, much of the geometric detail is inevitably lost since we are using a considerably lower resolution model than the original scans. To circumvent this issue, we "fake" surface geometry details from the high resolution scans by sampling a normal map for our model.

First, we compute a parameterization of the template mesh $F : \mathcal{D} \subset [0,1] \times [0,1] \rightarrow \mathcal{M} \subset \mathbb{R}^3$ (figure 2.6b). Each pixel on the normal map image we are generating corresponds to a point $(u, v) \in [0,1] \times [0,1] \subset \mathbb{R}^2$ in parameter space, which



(a) Mesh parameterization (b) Mesh parameterization with debug grid

Figure 2.6

either corresponds to a point $F(u, v) \in \mathbb{R}^3$ on the mesh surface, or corresponds to no point at all. For the pixels with a correspondence, we compute its normal by first finding the closest point to $F(u, v)$ on the high resolution scan, and then linearly interpolating the normals of the matched triangle vertices.

Note that with this strategy some of the points will have matches without a reliable normal, i.e., when the model and scan normals do not point to the same direction, when the matched points are too far apart, or the interpolated confidence at the point matched on the scan is too low, which is commonly observed at the armpits, between the legs and close to the ears. To fix the normals at the pixels with bad matches, we interpolate the normals with a membrane energy by solving the Laplace equation $\Delta n = 0$ at the problematic image regions, with the known reliable normals as boundary condition. Figure 2.8b shows one example of a normal

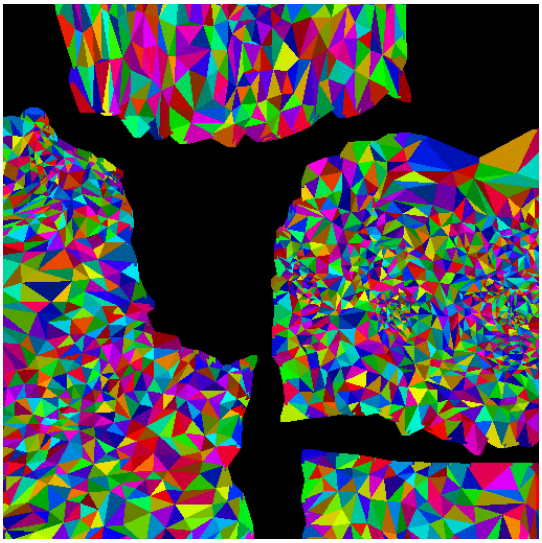
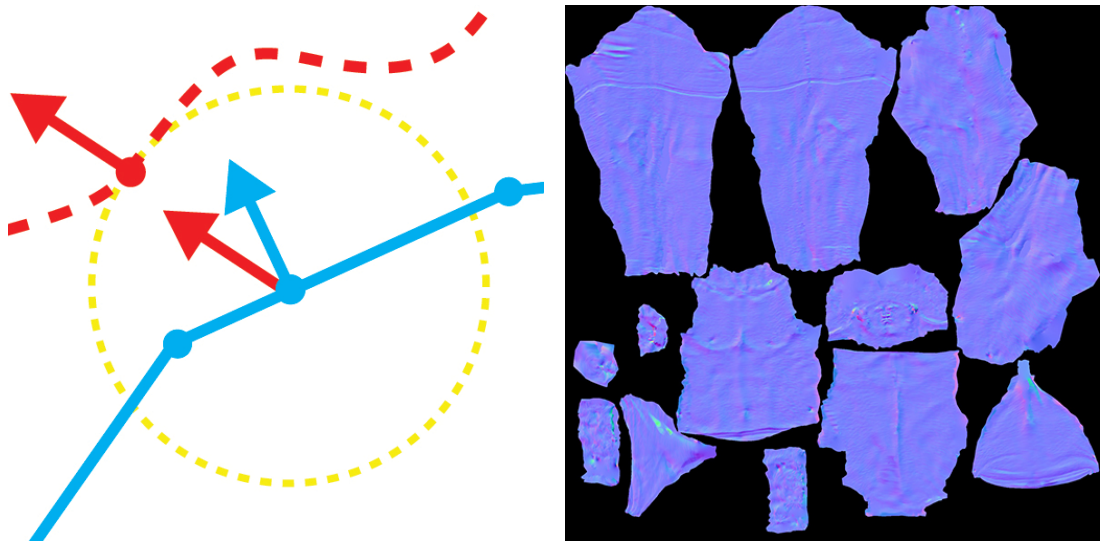


Figure 2.7: Zoom in of a color coded triangle IDs map used during the texture synthesis for quickly looking up the triangle corresponding to a pixel in (u, v) parameter space.

map after sampling and interpolation.

To quickly find the point on the model surface $F(u, v)$ corresponding to a pixel at parameter (u, v) we use the GPU to generate a triangle ids map (figure 2.7). For each pixel, we can then lookup the triangle corresponding to that pixel in constant time. In case there is a matching triangle, we then compute the pixel's barycentric coordinates in (u, v) space inside that triangle, and then find the corresponding point on the template mesh by linearly interpolating the vertex positions of the same triangle with the barycentric coordinates we computed in parameter space.

Figure 2.9a shows the model before and after applying the normal map for comparison. Also note that, since the normal map is represented in tangent space, we can directly use it when deforming the model into other poses (see figure 2.9b). Despite the much lower resolution of the template model, most geometric details



(a) (b) Normal map

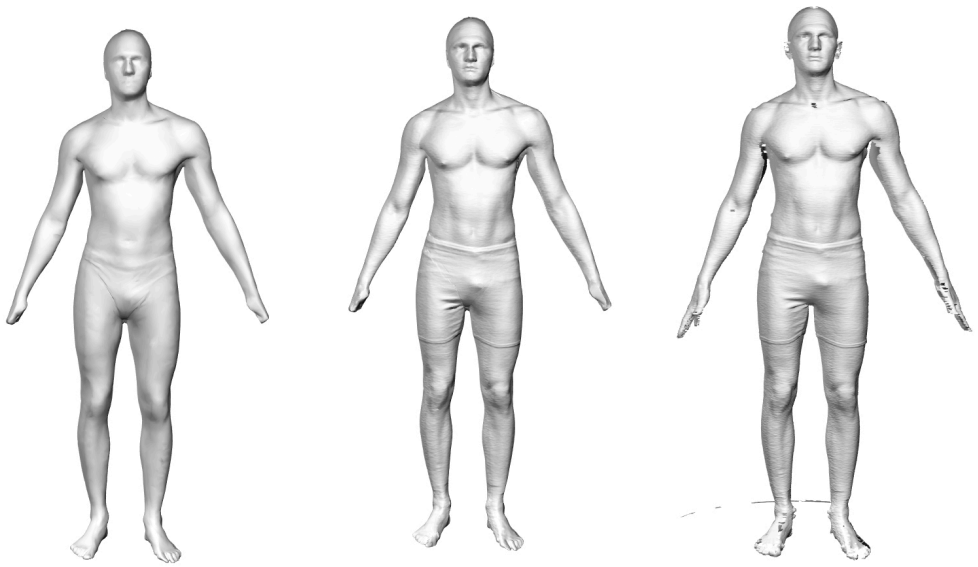
Figure 2.8: (a) Sampling the normal map for a lower resolution mesh (blue) from righer resolution scan (red). (b) A normal map (encoded in tangent space).

are captured in the normal mapped model.

Texture Synthesis: The CAESAR scanning data also includes a color attribute for each vertex, and we would like to use this information in order to generate color attributes for our registered model as well. We proceed by synthesizing a texture to be used as diffuse map for our model after the registration is completed.

As in the synthesis of the normal map, for each pixel in the texture image, we compute the corresponding point on the low resolution mesh from the parameterization, then compute the color from the interpolated color of the closest point on the high resolution scan. Figure 2.10 shows a texture atlas synthesized this way.

Note that, ideally, one would factor out the illumination and albedo from the



(a)

(b)

Figure 2.9: Faking lost geometric detail with normal mapping. (a) Model before and after normal mapping, and the high resolution scan for comparison. (b) The model deformed into other poses, before and after using the normal map.

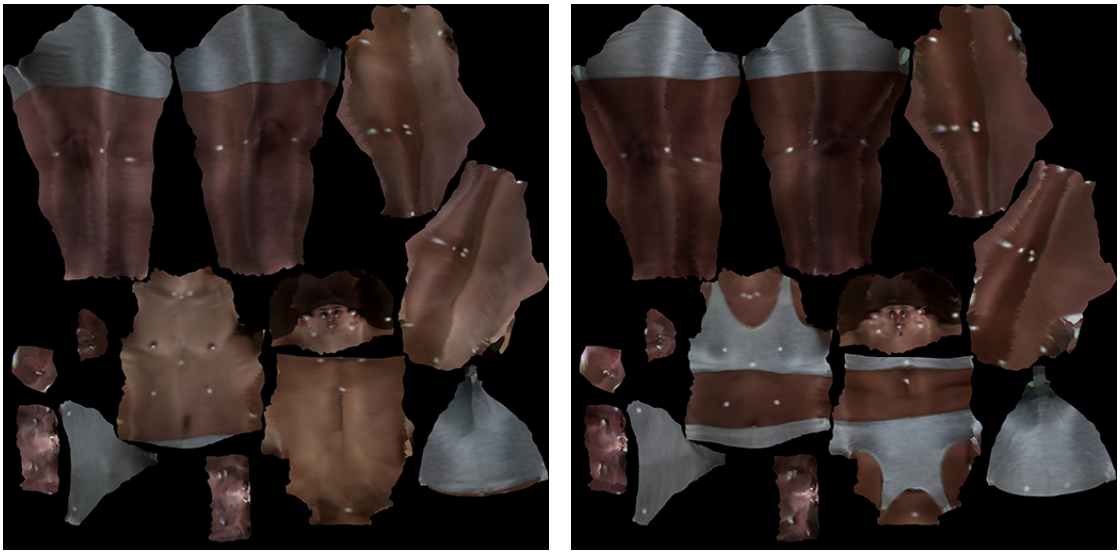


Figure 2.10: Textures synthesized for the model after the registration by sampling the high resolution scan.

diffuse map, but we leave it as future work, which can be easily incorporated.

2.5 Implementation and Evaluation

Numerical Solvers: For the nonlinear least squares solvers we used our own implementation of the Levenberg-Marquardt algorithm, with the update strategy for the damping parameter outlined in (K. Madsen April 2004). For the sparse linear systems, we used the CHOLMOD sparse Cholesky solver (Yanqing Chen et al. 2008) with the wrapper provided by the Eigen library (eigen.tuxfamily.org n.d.).

Proximity Queries: In order to accelerate the model to scan vertex proximity queries we employed axis aligned bounding boxes trees (AABBTree) on the model and the scans.

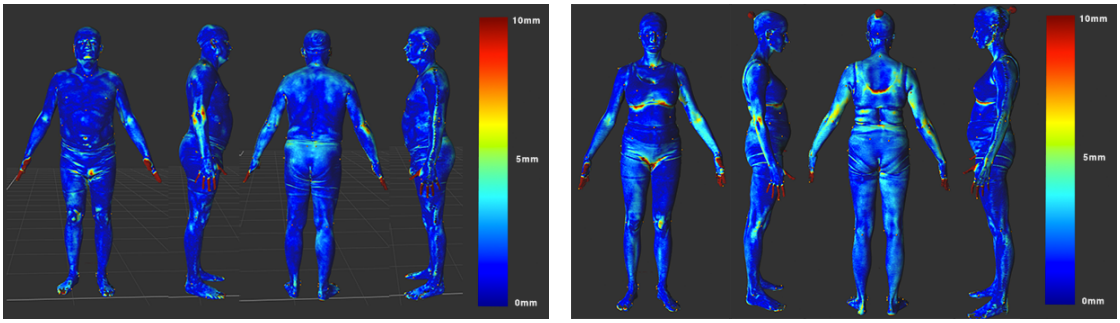


Figure 2.11: Error distribution of the high resolution scans with respect to the registered model on a male and female subject.

Evaluation: We registered over 2,000 scans from the CAESAR database, using the model learned for a single person using the already registered scans from the original SCAPE dataset, kindly provided by Dragomir Anguelov. The SCAPE meshes contain 12,500 vertices and 24,992 triangles, versus around 200,000 vertices and 415,000 triangles on the high resolution scans. Figure 2.12 show 6 of the resulting registered meshes on male and female subjects. To measure the accuracy of the registration we computed the root-mean-square error (RMSE) using the distance from all vertices of the high resolution scans to the closest point in the model. Figure 2.11 shows the error distribution on two subjects. Despite the overall very good quality of the registration results, note that the hands were not completely captured since our single person model had his hands closed, while the scans had them open. Table 2.1 shows the RMSE for the subjects depicted in figure 2.12, as well as the overall error for all scans. Each registration took about 30 seconds on a MacBook Pro with a 2.2GHz Intel Core i7 processor, 8GB of RAM and an AMD Radeon HD 6750M graphics card with 1GM of memory. Table 2.1 also details the running times for individual scans.

CAESAR Standing Scan	**Time (s)**	**RMSE (mm)**
CSR0002A	32	5.6
CSR0016A	33	5.1
CSR0025A	32	5.2
CSR0064A	33	5.7
CSR0065A	32	5.4
Overall	**32s/scan**	**5.2**

Table 2.1: Registration root-mean-squared error and running time on a subset of the CAESAR standing scans, as well as the overall performance on all the scans from the database.

With the shape deformation matrices of each registration in hands, we were then able to deform the resulting models into various poses, shown in figures 2.13 and 2.14. We also show the final fully textured model being animated in figure 2.15.

Finally, each registration outputs the $3x3$ shape deformation matrix for each triangle, so by unrolling the matrices into a 9-dimensional column vector, and stacking the vectors for all the T triangles, we have a $9T$ shape descriptor vector for that person. On its simplest form, we can perform principal component analysis on the person-specific descriptor (Allen et al. 2003; Anguelov, Srinivasan, Pang, et al. 2004) to get a lower dimensional model for the space of human bodies. We learned a model for the male and one for the female population this way.

2.6 Conclusion

In this chapter we presented an efficient algorithm employed to register over 2000 meshes of the CAESAR body scanning dataset. Our registration algorithm not only computes vertex correspondences, but also produces a full SCAPE model

specific to the subject that can be then manipulated into various poses. We have also improved the visual quality of the registration results by synthesizing diffuse and normal maps by sampling the high resolution scan.

Figure 2.12: High resolution scan (left) and registration result (right).

Figure 2.13: Registered male models deformed to other poses.

Figure 2.14: Registered female models deformed to other poses.

Figure 2.15: Pose deformation after registration and texture synthesis. The left-most mesh is the high resolution scan, followed by the model resulting from our registration algorithm after texturing, and then final model of the subject deformed into various poses.

Chapter 3

Fast Body Model Fitting to Noisy Depth Map

3.1 Introduction

The availability of relatively inexpensive depth sensors is making depth maps a primary source of data for inverse problems and mixed and augmented reality applications, and algorithms to efficiently work with such noisy depth maps are gaining increasing attention in academia and industry. In this chapter we focus on understanding the human body shape.

Unlike in chapter 2, where we were registering with a complete high resolution 3D scan of another person, here we focus on registering the model against noisy depth images obtained from a single point of view. Specifically, the main contribution of this chapter is to describe in detail an efficient algorithm that is able, in couple of seconds, to find the closest point on the manifold of human bodies

(a) Point Cloud (b) Initialization with IK (c) After Pose Fitting

(d) Base Shape (e) After Shape Fitting (f) Recovered Model

Figure 3.1: Overview of the optimization pipeline. Going from the top left to the bottom right, the figures show (a) the user point cloud, (b) the initialization with inverse kinematics (and the corresponding model and matched skeleton), (c) the result after the pose refinement, (d) the starting (template) model mesh by itself after the pose optimization, the result of the shape parameters optimization (e), and, finally, the final fitted model mesh by itself (f). This whole process takes only a couple of seconds, and the animation of the fitting process can be seen in the accompanying videos.

learned from the high resolution 3D body scans to the depth data coming from inexpensive noisy depth sensors such as Microsoft's Kinect. It is a significant jump in efficiency with respect to published comparable methods, opening new possibilities for applications.

3.2 Related Work

Body Scanning From Depth Maps. Related to our method, (Weiss et al. 2011) presents a technique to scan bodies from RGBD images coming from the Kinect sensor. For initialization, they assume a rough initial pose from the user, who is also required to input its height. In contrast, our method exploits the readily available estimated user skeleton (OpenNI n.d.; Shotton et al. 2011) to initialize the optimization, which has the benefit of greatly relaxing the assumptions in the initialization, while, at the same time, placing less restrictions on the user's movement. Moreover, the authors report the optimization taking about 65 minutes per body for 4 frames. We recognize it is hard to make fair comparisons here, but, in contrast, by exploiting the structure of the objective function being minimized, each frame in our optimization is solved in just a few seconds.

On a similar line, (Tong et al. 2012) presents a system to scan human bodies using multiple Kinects while the subject stands nearly static on a turntable. The various point clouds are registered to reconstruct a 3D mesh of the user. In (Wang et al. 2012) the authors describe a technique to estimate a 3D body shape with an approximate cylindrical model to the point cloud, while the user rotates in a constrained pose. Finally, in (Cui et al. 2012), multiple scans from a single Kinect are registered to reconstruct a 3D model of the user from a single pose without the use of body templates.

Lastly, we should also mention commercial body scanning solutions, ranging from expensive 3D scanners (e.g. (Cyberware n.d.)) to recent Kinect-based alternatives such as (Bodymetrics n.d.) and (Styku n.d.).

Even though our method can be clearly used to build a 3D body scanning system, and it is certainly a motivation for us, our main concern and contribution in this chapter is to describe a fast technique to register a learned, expressive parametric human body model to the point cloud coming from a single depth camera. We believe that this is useful not only as a component of a 3D body scanning system, but also to provide a richer semantic description of users in a 3D scene.

ICP and Nonrigid Registration. During our optimization, we employ the best practices of the iterative closest point algorithm (ICP), originally proposed for rigid registration (Y. Chen et al. 1991; Besl et al. 1992). We won't attempt to cover the vast ICP literature here, and we refer the reader to (Rusinkiewicz et al. 2001) instead, which compares the best practices for rigid registration. In (Brown et al. 2007), the ICP algorithm is extended to cope with moderately non-rigid transformations between multiple scans.

Also related to our method are nonrigid registration techniques that seek to reconstruct full meshes from partial views of deformable scans. We can make the broad distinction between methods that make use of a template mesh to aid the registration (Allen et al. 2003; Anguelov, Srinivasan, Koller, et al. 2005; De Aguiar et al. 2008), often aided by markers or user selected correspondences, and methods that work directly on the data without the aid of a template (Chang et al. 2009; Cui et al. 2012). Recently, (H. Li et al. 2012) presented a technique that is able to create temporally coherent watertight surfaces from high resolution partial scans obtained from state of the art multi-view 3D acquisition systems.

We should make the remark that our main goal is not to reconstruct a mesh representing the user, but to gain semantic information about the user's body. The template is used not simply as an aid to the registration, but it is our representation of the space of human bodies. For instance, even after very precise non-rigid registration, there is no semantic knowledge of what each mesh vertex represents on the body, which would still be necessary on a later "body semantic interpretation" stage.

Pose Recognition and Human Motion Tracking. Finally, our algorithm relies on skeleton pose recognition or tracking for initialization. We won't attempt to be exhaustive in the vast markerless human motion tracking literature, since we are not concerned here in tracking per se, but it is used as an input for our initialization, so the best algorithm in hands can be simply plugged in to initialize our optimization. Nevertheless, we refer the reader to the survey (Forsyth et al. 2006), and to (Shotton et al. 2011) for the skeleton tracking algorithm for range images in Microsoft's Kinect.

3.3 Efficiently Fitting the Body Model to Noisy Depth Map

Given a frame from a noisy depth map, the problem we want to solve is to find the point on the space of bodies that best approximates the data. With an established correspondence between 3D data and mesh points, $\mathbf{x}_i^D \leftrightarrow \mathbf{x}_i^M$, we seek

to minimize the sum of squared distances

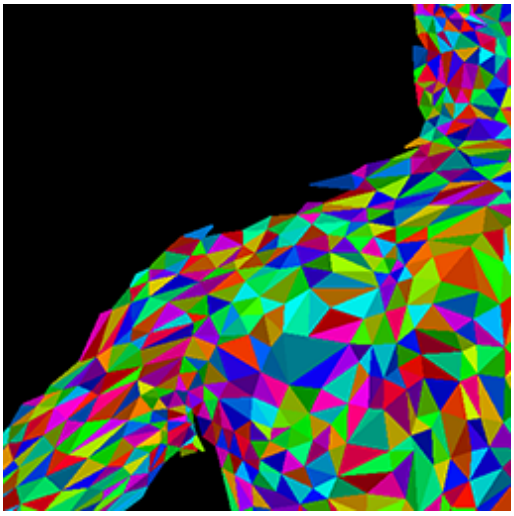
$$E(\boldsymbol{\omega}, \boldsymbol{\beta}) = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{x}_i^D - \mathbf{x}_i^M(\boldsymbol{\omega}, \boldsymbol{\beta})||^2 = ||X^D - X^M(\boldsymbol{\omega}, \boldsymbol{\beta})||^2 = ||F(\boldsymbol{\omega}, \boldsymbol{\beta})||^2 \quad (3.1)$$

with respect to the model pose ($\boldsymbol{\omega}$) and shape ($\boldsymbol{\beta}$) parameters. The shape parameters are the PCA coefficients learned from the registered scans of the CAESAR dataset. It is a nonlinear least squares problem, similar to problems we have already formulated in the previous chapters, and, again, we must resort to iterative methods, solving it using the Levenberg-Marquardt algorithm.

Note that even though the objective function has a similar form to the energy we minimized during the registration to the high resolution scans in chapter 2, the nature of the data requires a different ad-hoc optimization strategy. Our optimization can be described in a typical ICP-style *sample selection-matching-parameter optimization* framework, and we describe the steps next.

3.3.1 Initialization

An important point in our algorithm, which avoids many difficulties present in nonrigid registration techniques, is that we exploit the readily available markerless motion-captured skeleton computed from the depth map (OpenNI n.d.; Shotton et al. 2011). We initialize the optimization with inverse kinematics solved directly on the model skeleton, making each model joint correspond to a tracked joint, and optimize the squared error of all matched joints simultaneously with Levenberg-Marquardt iterations. Due to the high level of noise in the skeleton estimation, in order to improve robustness in the initialization of the root joint orientation,



(a) Triangle IDs Map

(b) User Mask

Figure 3.2: (a): Triangle IDs map. The mesh is projected on the depth image plane, with the depth camera projection matrix, and an image with the ID of the triangle that projects into each pixel is quickly computed with the GPU. The map is then used to quickly establish correspondences between 3D data points inside the user mask and mesh triangles. (b): Point cloud and user mask created with background subtraction on the depth map.

before solving the IK, we first estimate its root pose by directly solving for the rigid alignment between the model skeleton and tracked skeleton torso joints (shoulders, neck and hips). After a rough initialization has been computed with inverse kinematics, the pose and shape parameters are refined using the optimization we detail in the following sections.

3.3.2 Point Cloud to Mesh Correspondence

The first step is to establish a set of correspondences between data points and points on the model mesh for the current parameters.

We exploit the fact that the point cloud comes from a range image in or-

der to quickly establish correspondences by projection, as in (Neugebauer 1997; Rusinkiewicz et al. 2001) for rigid registration. Moreover, we use the GPU to render the mesh using the calibrated depth camera as the projection matrix, and output the triangle ID per pixel as the fragment "color". The resulting image will have, in each pixel, the triangle that renders into that pixel. Figure 3.2a shows a model instance with the corresponding triangle IDs map. With this triangle IDs map in hands, each pixel in the depth camera image plane will either lie on the intersection between the user data mask and the mesh projection, on the user mask and not on the mesh projection, or on the mesh projection but not on the user data mask.

In the first case, on the overlap between the two masks, the correspondence is immediate. The data point corresponding to the pixel (computed by inverse projection) is matched to the triangle projecting to the pixel. On the other hand, if a projected data point doesn't correspond to a projected triangle, the correspondence is found from the closest triangle on the triangle IDs map *in image space*. Conversely, a pixel on the triangle IDs map not corresponding to any projected data points is matched to a data point by finding the closest point on the user mask (again, in image space). These correspondences can be efficiently computed with the distance transform to the projected mesh and user data mask (Felzenszwalb et al. 2012). Note that the distance transform to the data mask is constant during the optimization, and can be computed only once, while the mesh's mask has to be updated whenever the body parameters change.

The user data mask is created by segmenting the user with background sub-

traction on the depth map, which is simple, fast, and works well for our purposes. Figure 3.2b shows an example of a point cloud and the corresponding user mask.

We found that it was necessary to include correspondences in both directions, from model to data and vice-versa, since having source point samples to the proximity queries coming only from one of the sets made the algorithm much more susceptible to local minima.

Moreover, we only match a random subsample of the mesh and data points ($\sim$ 3000 pairs, corresponding typically to 10% of the available pairs), which significantly speeds up the optimization with no perceived loss in accuracy.

3.3.3 Point to Triangle Distance and Associated Jacobian Matrices

With the correspondences between data points and triangles in hands, the point on the mesh corresponding to each data point is the closest point on the triangle. When the model parameters change, the triangle vertices change positions, causing the corresponding point to also move. The main goal of this section is to describe how this closest point changes in terms of the Jacobians of the vertex positions with respect to the model parameters, which we need to minimize 3.1[1]. The actual Jacobians of the vertex positions were already presented in section 1.5.2.

Given a point Q and a triangle T with vertices P_1, P_2, $P_3 \in \mathbb{R}^3$, defining a plane $\pi = (P_1, N)$ passing through P_1 with normal $N = (P_2 - P_1) \times (P_3 - P_1)$, the closest

1. This discussion was missing entirely on the registration algorithm in chapter 2 since there it was enough to consider only the correspondences from mesh vertices to the scanning data, which remains fixed. Here it is not enough, and we have to consider bidirectional correspondences, from model points to data points, and vice versa.

point to Q on π is given by

$$P = Q + \frac{(P_1 - Q)^T N}{N^T N} N = Q + ((P_1 - Q)^T \mathbf{n})\mathbf{n}, \tag{3.2}$$

with $\mathbf{n} = N/|N|$. Differentiating with respect to any of the model parameters

$$\begin{aligned}
\partial P &= \partial \left[\frac{(P_1 - Q)^T N}{N^T N} N \right] = \partial \left[((P_1 - Q)^T \mathbf{n})\mathbf{n} \right] \\
&= \mathbf{n} \partial \left[(P_1 - Q)^T \mathbf{n} \right] + \left[(P_1 - Q)^T \mathbf{n} \right] \partial \mathbf{n} \\
&= \mathbf{n}\mathbf{n}^T \partial P_1 + \left[\mathbf{n}(P_1 - Q)^T + ((P_1 - Q)^T \mathbf{n}) I \right] \partial \mathbf{n}.
\end{aligned}$$

By direct calculation, the derivative of the normalized plane normals are

$$\partial \mathbf{n} = (I + \mathbf{n}\mathbf{n}^T) \frac{\partial N}{|N|}.$$

And, finally, for the plane normals, we have

$$\begin{aligned}
\partial N &= \partial [(P_2 - P_1) \times (P_3 - P_1)] \\
&= \partial (P_2 - P_1) \times (P_3 - P_1) + (P_2 - P_1) \times \partial (P_3 - P_1) \\
&= [P_1 - P_3]^\wedge (\partial P_2 - \partial P_1) + [P_2 - P_1]^\wedge (\partial P_3 - \partial P_1).
\end{aligned}$$

Therefore, we have derived the Jacobian of the closest point on the plane of triangle T in terms of its three vertices P_1, P_2 and P_3, which are functions of the model parameters. Summarizing the calculations above, we have

$$\partial P = A_1 \partial P_1 + A_2 \partial P_2 + A_3 \partial P_3, \tag{3.3}$$

where the 3×3 matrices A_i are determined by collecting the terms in the derivation, and ∂P_i are $3 \times p$ matrices, with one column for each of the p body model parameters in question.

Note that the closest point to the data point on the triangle T may lie outside the triangle. The closest point will then be either on one of the triangle edges, or be one of the triangle vertices. On the first case, we can then compute the Jacobian of the point on the edge with respect to the two edge vertices in a similar way as we did for the triangle plane. On the second case, the Jacobian in simply the Jacobian of the closest triangle vertex (∂P_1, ∂P_2 or ∂P_3).

3.3.4 Parameter Optimization

We minimize the energy 3.1 by alternating minimization in shape and pose parameters, and we describe each separately.

3.3.4.1 Shape Parameters

Again, as in section 4.4.1 for direct shape manipulation, the linear dependency of the model vertex positions from the shape parameters turns 3.1 into a linear least squares problem. We remind the reader that, with respect to the shape parameters, the model vertex positions can be expressed as

$$X = X_0 + \sum_i \beta_i X_i(\boldsymbol{\omega}) \tag{3.4}$$

and the shape basis vectors X_i need to be update whenever the pose parameters change.

A key observation from this section is that when solving for the shape parameters we are actually solving two linear least squares problems, one for the vertex positions, and one for the shape parameters. However, the first is by far the

most expensive to compute, since it is on the size of the number of mesh vertices (thousands), while the second is only on the order of the shape parameters (30 parameters in our case). By using our observation from section 1.3 that the linear system corresponding to the normal equations of the vertex positions (1.7) can be pre-factored offline, we are left with only the smaller linear system in shape parameters during the optimization, while the sparse linear systems necessary to compute the shape basis in 3.4 are efficiently solved with the same Cholesky factorization used to update the model vertices with forward and back substitution.

3.3.4.2 Pose Parameters

The optimization with respect to the pose parameters is again more complicated due to the nonlinearity introduced by the rotations. The actual optimization is once more completely analogous to the pose optimizations we encountered during the registration algorithm in chapter 2 (section 2.3.2), and during direct pose manipulation in chapter 4 (section 4.4.2), which employed the pose manipulation Jacobian matrix derived in section 1.5.2. Here, however, we don't have markers or user constraints to guide the optimization, and the errors are thus measured with direct closest point correspondences between model and data. The main consequence for the optimization is that we have to solve a much larger linear system in each iteration of the Levenberg-Marquardt algorithm. We also allow during the pose optimization an overall translation of the root joint.

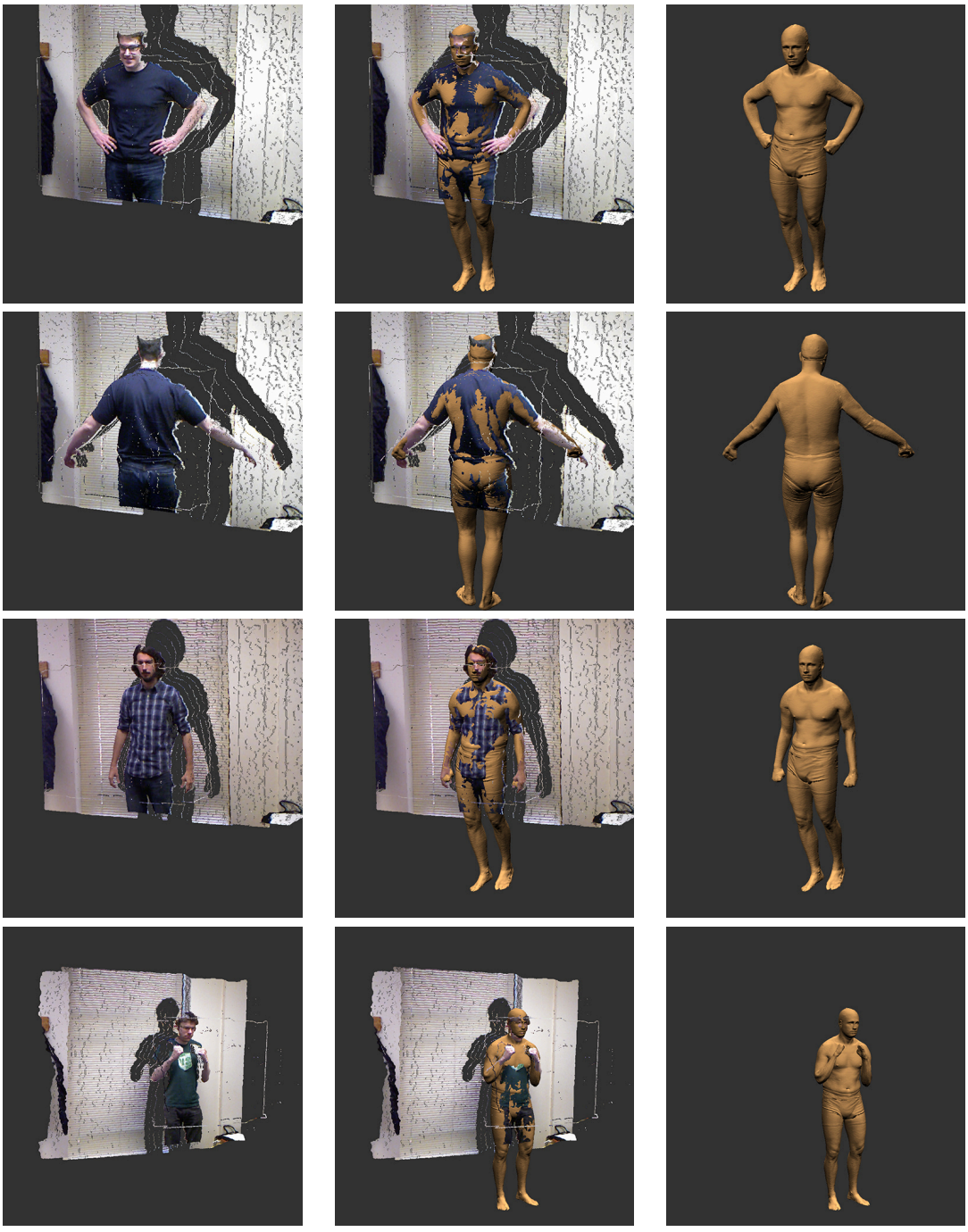


Figure 3.3: Fitting results for different people.

3.4 Results

Our data was collected with a Microsoft Kinect sensor, with pose annotated by the skeleton computed with OpenNI. During the recording, we also extracted with background subtraction the user mask we used later during the fitting.

Figure 3.3 shows qualitative results of our method on 3 different subjects. It takes about 5s to optimize the shape and pose parameters. The initialization with inverse kinematics by itself takes on average 20ms, the pose optimization, 2.9s, and the shape optimization about 1.6s. As a comparison, (Weiss et al. 2011) reports 65 minutes to solve for 4 frames, so we have achieved a significant gain in performance. The timings were computed on a MacBook Pro with Retina display, with a quad-core Intel Core i7 processor, with 16GB of memory, a NVIDIA GeForce GT 750M graphics card with 2GB of memory, running OS X 10.9

One limitation of our method is that it heavily relies on the initialization. If the algorithm is initialized with a bad skeleton it becomes very hard to recover, which comes at no surprise due to the ICP style nature of the algorithm.

3.5 Conclusion

We have described an efficient algorithm that is able to fit the rich class of SCAPE body models, learned from high resolution 3D scans of human bodies, to noisy depth maps obtained from relatively inexpensive depth sensors.

The efficiency was mainly achieved by (1) using the now readily available marker less tracked skeleton for initialization, and (2) by exploring the structure of the

bodies manifold during the optimization. By looking closely into the equations defining the SCAPE model, we were able to derive the analytical formulas for the Jacobians of the energies being minimized, as well as observe that key offline prefactorizations greatly improve the speed in the model and Jacobians evaluations.

In this work we have focused on fitting the body model to isolated frames. From here, we would like to explore temporal coherence to efficiently fit the model to video sequences and real time data.

Chapter 4

Human Body Manipulation

4.1 Introduction

Modeling and manipulating human bodies is important not only for computer animation, games and special effects, but also when designing around the human form, such as in apparel or product design, where it is important for designers, who are typically not animation experts, to be able to easily manipulate body shapes and poses with minimal training. We focus in this chapter on interactive manipulation of the body model shape and pose.

As far as shape manipulation is concerned, we identify two modalities: (1) Exact, quantitative specification, were the user directly sets the body parameters. The requirements could come from a clothing sizing chart, for instance, in order to create a typical body with a given set of measurements, say, to design a garment on a CAD, or to virtually preview a product online. (2) Alternatively, body shape can be interactively specified just by visual feedback, where one may, say, want to

Figure 4.1: Direct manipulation of human body shape. The leftmost image shows the starting shape, while the remaining three were obtained by directly dragging the vertices in red to the positions in blue. Our system shows the result of the manipulation in real-time to the user, who can, for instance, make the shoulders wider simply by dragging them out, make the individual shorter by dragging its feet up, and the belly bigger by dragging it out in a profile view, as shown in the rightmost image.

make a given body a little taller, or thinner, or fatter, or to make the shoulders wider by visual inspection without worrying about the exact measurements.

Body pose, on the other hand, is typically specified either by a combination of IK/FK manipulators controlled by animators, were the kinematic chains underlying the model are directly manipulated or driven from motion capture systems. In inverse kinematics, joint orientations along the chains are solved in order to satisfy a given set of end effectors constraints. The pose configuration is then used to deform the model surface, such as a skinned mesh. We propose here to solve the inverse problem one step further. Without exposing the joint hierarchy, we allow users to directly manipulate the surface by dragging vertices, and then solving for

the joint positions which produce the desired deformations.

We begin this chapter by revisiting the widespread linear model for shape parameter specification, evaluating its accuracy against actual body measurements, and then introducing an optimization strategy we employed in order to fine tune the model to precise measurements when better accuracy is required. We also show how to automatically infer all the measurements when only a subset of the measurements is present as a requirement.

We then move to our main contribution in this chapter: a direct human body manipulation technique, where the user specifies the body shape and pose by directly dragging mesh vertices to the desired positions, and a point on the bodies manifold satisfying the user constraints is automatically found. We conclude by showing how our direct manipulation metaphor can be applied to directly manipulate images and 3D point clouds of bodies at interactive rates.

4.2 Related Work

Pose Manipulation. Traditionally, character poses are manipulated by a combination of forward and inverse kinematics handles, or controlled directly from motion capture systems (Vicon n.d.; Shotton et al. 2011). It is beyond our scope to try to cover the extensive IK manipulation literature in robotics and computer graphics, but we refer the reader to (Murray et al. 1994) as a reference. As far as pose manipulation is concerned, our formulation is closely tied to IK manipulation, but the underlying joint rotation hierarchy is not exposed to the user, which manipulates the mesh surface directly. Also related, (Grochow et al. 2004) learns a

Gaussian Process Latent Variable Model for human poses from examples in order to prefer poses lying close to the training data during IK manipulation.

Shape Manipulation. On the other hand, body shape parameters are traditionally controlled by GUI sliders that either directly control the model parameters (e.g., PCA coefficients), or loosely describe the model parameters as functions of semantic attributes such as height, weight, etc…(Allen et al. 2003; Hasler et al. 2009), which are then also controlled by sliders. A contribution of our work is to automatically infer the model parameters based on the user direct interaction with the mesh.

Closer to us is (Sumner et al. 2005), who describes a method to navigate on a space defined by generic example meshes provided for training. More recently, (Jacobson et al. 2012) describes a method to automatically infer the pose manipulation degrees of freedom of a skinned mesh by constraining the space of solutions with an as-rigid-as-possible energy.

Body Image Warping. We applied our direct body manipulation metaphor to induce an image and space deformation, and let users directly manipulate images and meshes of themselves. Related in spirit to our goals in this application, (Zhou et al. 2010) presented a user-assisted fitting of a body model to change the shape of human bodies in images, and (Jain et al. 2010) developed the original idea applied to videos. Instead of indirectly controlling the body shape with sliders, a major difference in our work is that we let the user directly manipulate the image.

Moreover, with the richer input provided by the Kinect sensor, we apply our technique to the direct human-body aware manipulation of human body images

and point clouds. We fully automatically fit the body model to the data, with an algorithm several orders of magnitude faster than corresponding published methods (Weiss et al. 2011) that we detail in the next chapter.

Spatial and Image Deformation. Touching our application to image and point cloud warping, we relate to work on spatial deformation that traces back at least to free-form deformation (Sederberg et al. 1986). We don't try to be exhaustive, but we feel that it is worth mentioning the work of (Schaefer et al. 2006), where image displacements are interpolated with moving least squares, and (Botsch et al. 2005), which uses radial basis functions to define shape deformations in a similar way that we do here to deform point clouds.

4.3 Shape Manipulation by Direct Specification

4.3.1 Mapping Measurements to PCA Coefficients

The scans on the CAESAR database are annotated with body measurements, such as height, waist, hip and chest circumference. For a user manipulating the shape of our body model, however, the PCA shape coefficients do not have an intuitive meaning. Following (Allen et al. 2003), we learn a linear mapping from a subset of the measurements accompanying the scans to the PCA coefficients by fitting with linear least squares the measurements of each scan to the learned PCA coefficients of each subject.

The linear mapping permits the user to directly pick the semantic body attributes (height, waist, etc…), and the corresponding PCA coefficients are auto-

matically computed.

4.3.2 Incomplete Measurements Inference

By itself, one drawback of the method described in the previous section is that we have to set every measurement in order to specify a body. For instance, if we only want to make the body taller, while keeping the circumferences measurements proportional, we would also have to specify them if we don't want to also make the subject look disproportionately thinner. Or, for clothing sizing charts, which typically specify only a subset of the measurements, depending on the type of garment or the brand, we have to automatically infer reasonable values for the parameters which are not specified in the chart.

To address this issue, we also learned a normal distribution for the body measurements from the annotations of the scans. We learned a separate distribution for men and women. If $\mathbf{m} \sim \mathcal{N}(\boldsymbol{\mu}_m, \boldsymbol{\Sigma}_m)$ is a random variable denoting the body measurements, let's say that at one given point $\mathbf{m}_\mathrm{s}$ and $\mathbf{m}_\mathrm{f}$ denote the variables that were specified by the user, and the ones that are free to vary, respectively. We estimate the values for the free variables from the mode of the conditional distribution $P(\mathbf{m}_\mathrm{f}|\mathbf{m}_\mathrm{s})$, which, in our case, is again a Gaussian and simplifies the calculations.

For completeness, we include the full expression for the conditional mode. If Π is a permutation matrix that puts the free variables first, followed by the specified ones, the mean and covariance matrix of the permutation is $\boldsymbol{\mu}_\pi = \Pi\boldsymbol{\mu}_m$ and $\boldsymbol{\Sigma}_\pi =$

Algorithm 2 Sampling a Multivariate Gaussian Distribution $\mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$

1: Generate a n-dimensional vector $\mathbf{z}$ of standard unidimensional gaussian samples obtained with the Box-Muller transform:

$$z_i = \sqrt{-2 \ln u_{i_1}} \cos(2\pi u_{i_2}),$$

where u_{i_1} and u_{i_2} are uniform samples in the interval $[0, 1)$.
2: Decompose the covariance matrix $\boldsymbol{\Sigma} = LL^T$ with Cholesky decomposition
3: Compute the sample by applying the affine transformation $L\mathbf{z} + \boldsymbol{\mu}$

$\Pi\boldsymbol{\Sigma}\Pi^T$, which we express in block form by

$$\boldsymbol{\mu}_\pi = (\boldsymbol{\mu}_\mathrm{f}, \boldsymbol{\mu}_\mathrm{s})$$
$$\boldsymbol{\Sigma}_\pi = \begin{pmatrix} \boldsymbol{\Sigma}_\mathrm{ff} & \boldsymbol{\Sigma}_\mathrm{fs} \\ \boldsymbol{\Sigma}_\mathrm{sf} & \boldsymbol{\Sigma}_\mathrm{ss} \end{pmatrix}, \text{where } \boldsymbol{\Sigma}_\mathrm{sf} = \boldsymbol{\Sigma}_\mathrm{fs}^T.$$

The conditional mode is then given by the conditional mean

$$\mathbf{m}_f = \boldsymbol{\mu}_{f|s} = \boldsymbol{\mu}_f + \boldsymbol{\Sigma}_\mathrm{fs}\boldsymbol{\Sigma}_\mathrm{ss}^{-1}(\boldsymbol{m}_s - \boldsymbol{\mu}_s). \tag{4.1}$$

This way, the user can specify only a subset of the measurements $\mathbf{m}_s$, and we automatically estimate the free measurements $\mathbf{m}_f$. The full set of measurements can then be transformed into PCA coefficients with the mapping described in the previous section.

4.3.3 Measurements Fine Tuning

Since (Allen et al. 2003), it has become common practice to specify the PCA coefficients through a linear mapping like the one described in section 4.3.1. However, the mapping provides only a coarse approximation to the desired body specification, and we address its accuracy in this section.

Figure 4.2: Meshes from the male bodies sample used to evaluate the measurements errors.

We first checked the accuracy of the linear mapping by drawing a sample of bodies from the same multivariate Gaussian distribution of the measurements described in the last section (see algorithm 2), and comparing the specified values with the actual measurements taken on the meshes. Figure 4.2 shows a subset of the sample. For each sample, we first transformed the measurements into PCA coefficients with the linear mapping, which were then used to create body mesh instances where we could actually take the measurements. Figure 4.3 shows the curves used to take the measurements, traced in accordance with the procedure used to take the measurements in the CAESAR dataset. Figure 4.4 shows the errors of the measurements on the bodies obtained with the linear mapping. In clothing size charts, for instance, note that the magnitudes of the errors are enough for someone to go up or down a size. Therefore, the linear mapping by itself is not accurate enough.

To fix this problem we fine tune the measurements with a nonlinear optimization we describe next. Each measurement is taken by either computing the length of a curve defined intrinsically on the body surface, such as for chest/waist/hips circumferences, or as the projected distance between two points on the surface, such as for measuring the overall or shoulder heights. We seek to minimize

$$E(\boldsymbol{\beta}) = E_C(\boldsymbol{\beta}) + E_R(\boldsymbol{\beta}) \tag{4.2}$$

in terms of the shape parameters $\boldsymbol{\beta}$. The energy is a combination of a curve length errors term E_C and a projected distance errors term E_R.

Intrinsic Curve Length Error. Let $\mathbf{C}_i$ be the i-th measurement curve, linear by parts defined by a sequence of points $\mathbf{C}_{ij}$ on the surface. Each point, in turn, is defined from its barycentric coordinates in a triangle of the mesh, so that the curves change with the mesh:

$$\mathbf{C}_{ij}(\boldsymbol{\beta}) = \alpha_{ij}^1 \mathbf{v}_{ij}^1(\boldsymbol{\beta}) + \alpha_{ij}^2 \mathbf{v}_{ij}^2(\boldsymbol{\beta}) + \alpha_{ij}^3 \mathbf{v}_{ij}^3(\boldsymbol{\beta}), \tag{4.3}$$

where $\alpha_{ij}^1 + \alpha_{ij}^2 + \alpha_{ij}^3 = 1$ and $\alpha_{ij}^1, \alpha_{ij}^2, \alpha_{ij}^3 \geqslant 0$. $\mathbf{v}_{ij}^{(k)}(\boldsymbol{\beta})$ are mesh vertices, and $\boldsymbol{\beta}$ indicates the body shape parameters. The curves themselves were obtained either as cross sections of the mesh, or as geodesics computed with the heat equation method of (Crane et al. 2013).

If we denote the length of a curve $\mathbf{C}_i$ by $|\mathbf{C}_i|$, given target measurements L_i for each curve we want to minimize

$$E_C(\boldsymbol{\beta}) = \sum_i (|\mathbf{C}_i|(\boldsymbol{\beta}) - L_i)^2 = ||\mathbf{F}(\boldsymbol{\beta})||^2. \tag{4.4}$$

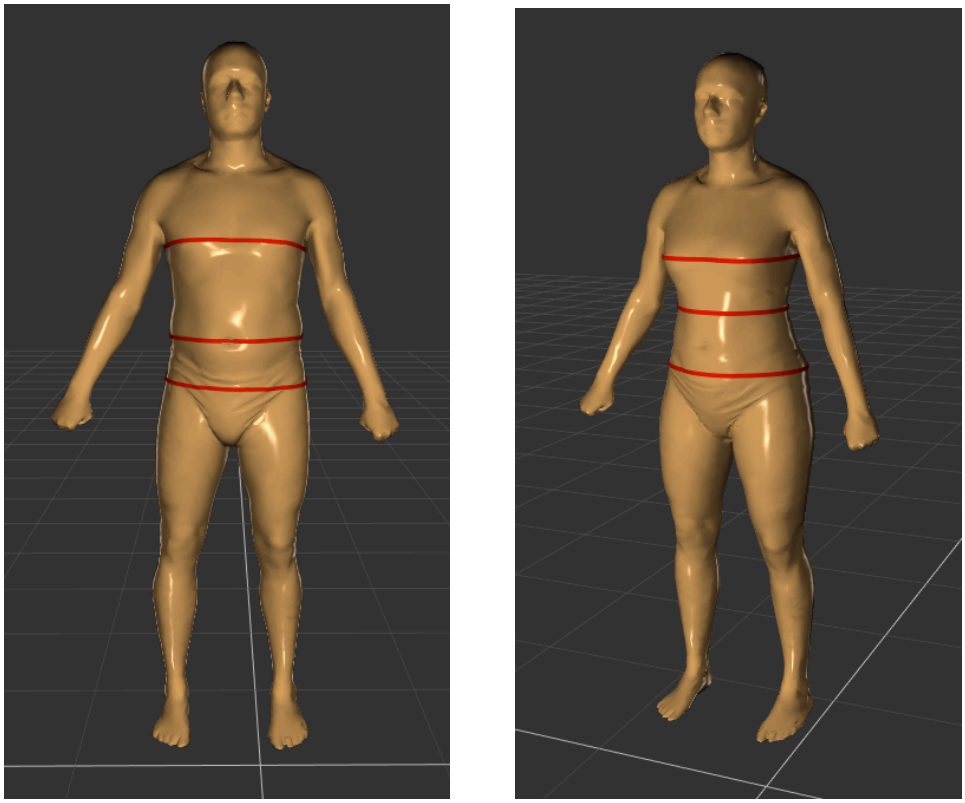


Figure 4.3: Measurement curves used to take the chest, waist and hips circumferences measurements in the male and female models.

For the optimization it is useful to derive here the Jacobian matrix of the length of a curve defined on the surface. Since

$$|\mathbf{C}_i|(\boldsymbol{\beta}) = \sum_j || \underbrace{\mathbf{C}_{i(j+1)}(\boldsymbol{\beta}) - \mathbf{C}_{ij}(\boldsymbol{\beta})}_{\mathbf{s}_{ij}(\boldsymbol{\beta})} || = \sum_j ||\mathbf{s}_{ij}(\boldsymbol{\beta})||, \tag{4.5}$$

we have that

$$\partial|\mathbf{C}_i| = \sum_j \partial||\mathbf{s}_{ij}|| = \sum_j \partial\sqrt{\mathbf{s}_{ij}^T\mathbf{s}_{ij}} = \sum_j \frac{1}{||\mathbf{s}_{ij}||}\mathbf{s}_{ij}^T\partial\mathbf{s}_{ij}, \tag{4.6}$$

where

$$\partial\mathbf{s}_{ij} = \partial\mathbf{C}_{i(j+1)} - \partial\mathbf{C}_{ij}. \tag{4.7}$$

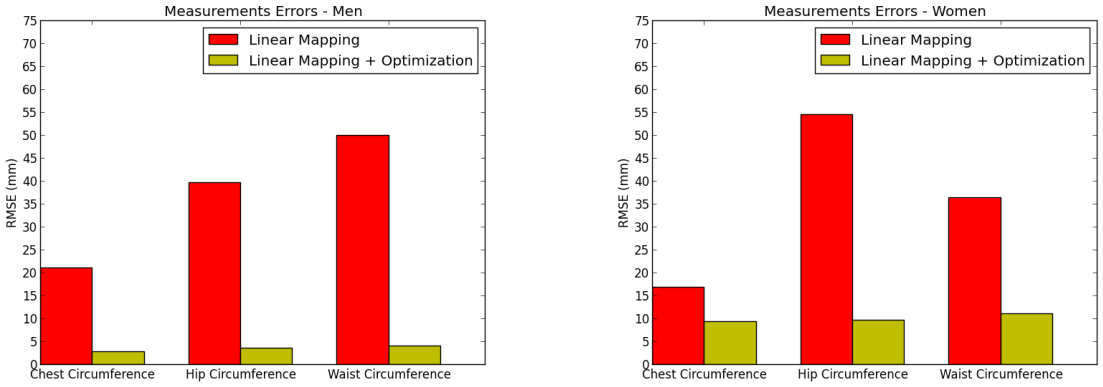


Figure 4.4: Measurement errors with only the linear mapping, and after fine tuning the measurements.

Since $\mathbf{C}_{ij}$ are points on the mesh specified in barycentric coordinates, from equation 4.3 we have that

$$\partial \mathbf{C}_{ij} = \alpha_{ij}^1 \partial \mathbf{v}_{ij}^{(1)} + \alpha_{ij}^2 \partial \mathbf{v}_{ij}^{(2)} + \alpha_{ij}^3 \partial \mathbf{v}_{ij}^{(3)} \tag{4.8}$$

and the remaining derivatives of the vertex positions with respect to the shape parameters on the right hand side are constants that where already derived before in equation 1.16. They remain constant throughout the optimization since the body pose doesn't change.

Projected Distance Error. A projected distance measurement is defined by a pair of points on the mesh $(\mathbf{P}_{i_1}(\boldsymbol{\beta}), \mathbf{P}_{i_2}(\boldsymbol{\beta}))$, again, given in barycentric coordinates in triangles of the mesh, and a projection direction $\mathbf{u}_i$. The height, for instance, is defined by the bottom and topmost vertices of the mesh in the standing pose, in the direction of the first principal direction extracted from the PCA of the mesh vertices. Given target lengths H_i for each projection, we want to minimize

$$E_R(\boldsymbol{\beta}) = \sum_i ((\mathbf{P}_{i_2}(\boldsymbol{\beta}) - \mathbf{P}_{i_1}(\boldsymbol{\beta})) \cdot \mathbf{u}_i - H_i)^2, \tag{4.9}$$

where we assume the point pairs are ordered such that the dot product is positive. The derivatives are trivially computed using 4.8.

We position the body in the standing pose (figure 4.3), and solve the nonlinear least squares problem 4.4 with the Levenberg-Marquardt algorithm, initialized with the PCA coefficients computed from the linear mapping. Figure 4.4 shows the measurements errors after the optimization for comparison with the original error we started with from the linear mapping alone.

4.4 Direct Body Mesh Manipulation

In contrast to the previous section, where the user specified a body by setting directly a set of body parameters, we present in this section an alternative manipulation metaphor where the body is specified by direct manipulation of the geometry. The user specifies the position for a set of vertices $\mathbf{x}_{c_i},\ i = 1, \ldots, C$ from the mesh, dragging to the desired positions $\mathbf{y}_{c_i}$, and our goal is to find the point on the space of human bodies that most closely approximates the user constraints.

On a given moment, the user can be either in a shape or pose manipulation mode, and we seek to minimize the user constraints

$$E(\boldsymbol{\alpha}) = \sum_{i=1}^{C} ||\mathbf{y}_{c_i} - \mathbf{x}_{c_i}(\boldsymbol{\alpha})||^2 + \gamma ||S(\boldsymbol{\alpha} - \boldsymbol{\alpha}_0)||^2, \tag{4.10}$$

where $\boldsymbol{\alpha}$ is either the shape ($\boldsymbol{\beta}$) or pose ($\boldsymbol{\omega}$) parameters. The regularization term is used to measure how close to the initial configuration the pose or shape stays after each drag of the cursor. S is a *stiffness matrix*, a diagonal matrix that sets the stiffness of each parameter. When in shape manipulation mode ($\boldsymbol{\alpha} := \boldsymbol{\beta}$), we

set it to the identity matrix, and, for pose manipulation, ($\boldsymbol{\alpha} := \boldsymbol{\omega}$) the individual diagonal entries are used to set the stiffness of each joint.

We now describe each optimization separately.

4.4.1 Body Shape Manipulation

When optimizing the shape after each drag of a control vertex, from the linearity of the model with respect to the shape parameters it follows that minimizing the energy 4.10 becomes a linear least squares problem. Using the results from section 1.5.1, the new shape parameter is the solution to the linear system

$$\begin{pmatrix} \partial_{\boldsymbol{\beta}} X^c \\ \sqrt{\gamma} I \end{pmatrix} \boldsymbol{\beta} = \begin{pmatrix} Y^c - X_0^c \\ \sqrt{\gamma} \boldsymbol{\beta}_0 \end{pmatrix}, \tag{4.11}$$

where $\partial_{\boldsymbol{\beta}} X^c$ is a $3C \times \dim(\boldsymbol{\beta})$ matrix obtained from slices of the Jacobian matrix of the model vertex positions with respect to the shape parameters (equation 1.11) corresponding to the constrained vertices. This matrix remains constant as long as the pose parameters don't change. $Y^c \in \mathbb{R}^{3C}$ is a column vector obtained by stacking the target constraints positions $\mathbf{y}_{c_i}$, and, similarly, X_0^c are slices from the mean shape vertices of the model (equation 1.7).

4.4.2 Body Pose Direct Manipulation

The nonlinear minimization of energy 4.10 with respect to the pose parameters $\boldsymbol{\omega}$ is completely analogous to the pose optimization step during our registration algorithm (section 2.3.2), so we refer the reader to the previous chapter. This time,

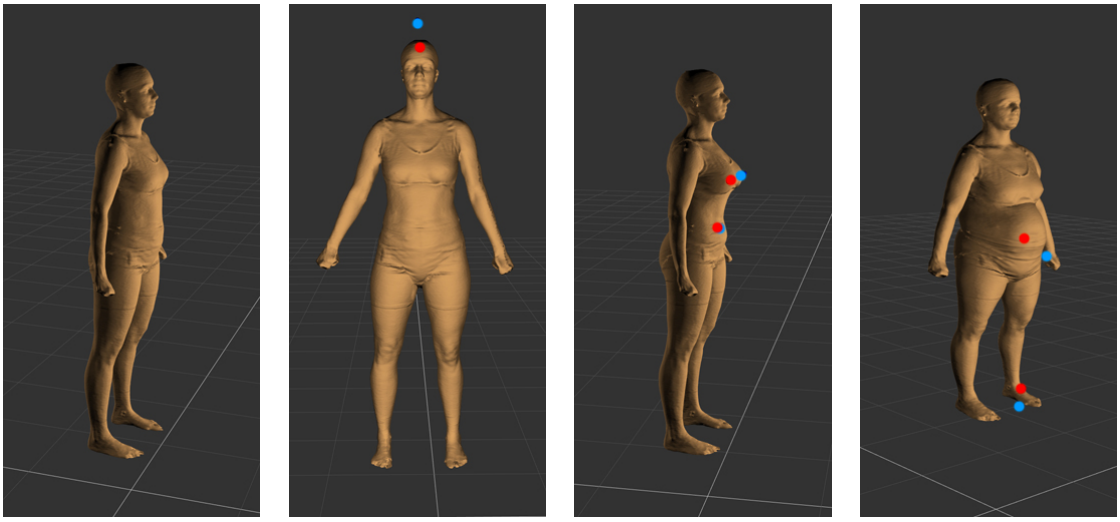


Figure 4.5: Direct manipulation of female body shape. The leftmost image shows the starting shape, while the remaining three were obtained by dragging the vertices in red to the positions in blue.

instead of the constraints imposed by the markers on the high resolution scan, we have position constraints imposed by the user.

The only significant difference is that since here we want responsiveness during the manipulation, we are less strict with the convergence criteria, allowing a smaller number of iterations (five) in order to avoid unexpected delays coming from the optimization algorithm.

The regularization term in 4.10 comes in acting as a bias on each step of the Levenberg-Marquardt iterations, giving a separate rotation stiffness for each joint.

4.5 Results and Evaluation

Using the registration results from the previous chapter on the CAESAR body scans, we created an overall body shape model for the male and female population.

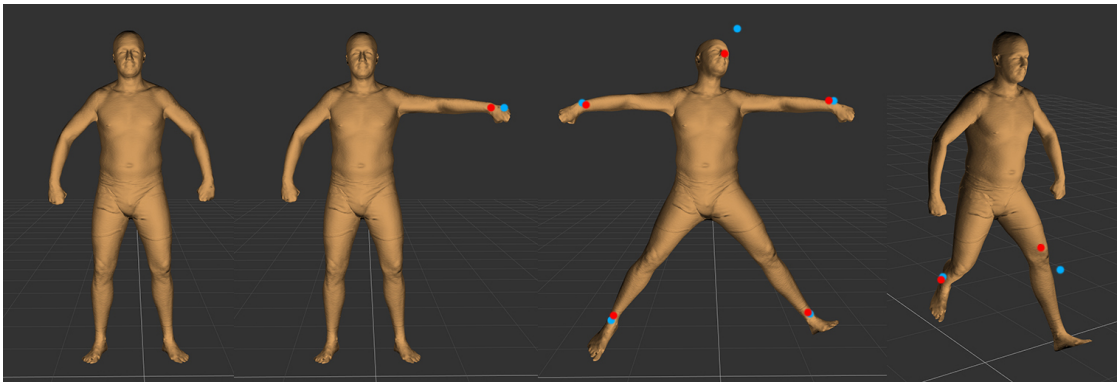


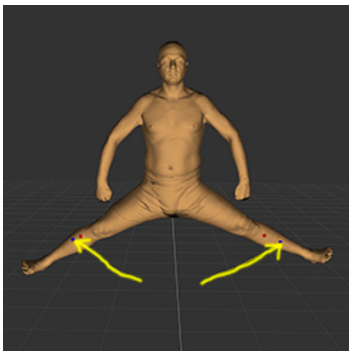
Figure 4.6: Direct manipulation of human body pose. The leftmost image shows the rest pose, while the remaining three were obtained by directly dragging the vertices in red to the positions in blue.

The registration of each body produces a 3×3 shape deformation matrix per triangle, so we reshape the matrix into a 9 dimensional column vector, and vertically stack all the vectors from all triangles of the mesh, resulting in a $9T$ dimensional vector defining the body shape of a subject. We compute the principal components for the whole training set, producing two models, one for the male, and another for the female population.

With the body models in hands, we can then proceed to directly manipulate them. Figure 4.1 shows the shape manipulation of the body shape on the male model, and figure 4.5, on the female model. Figure 4.6 shows the pose manipulation of the male model. The supplementary videos show the interactions in more detail.

Timing Benchmarks. In order to benchmark the performance of our manipulation technique, we recorded a set of pose manipulation sequences, writing down the target vertex positions while the user dragged the vertices to specify a pose, starting from the rest pose. We recorded the manipulated vertex indices, as well as

(a) Arm Raise



(b) Leg Split

(c) Walk

(d) Look Up

(e) Kick

(f) Boxing

Figure 4.7: Pose manipulation sequences for the timing benchmark.

the target cursor position after each mouse drag event. The 6 pose manipulation sequences are illustrated in figure 4.7 with the corresponding labels.

As a performance baseline, we compared the timings with traditional inverse kinematics. For each manipulation sequence, we computed the joint closest to the dragged vertex, an run our IK solver to find the joint rotations. Our IK solver works in exponential coordinates for the rotations, and we again solve the optimization with Levenberg-Marquardt. We recorded the time taken by each optimization for comparison, including the model update after the joint rotations are solved.

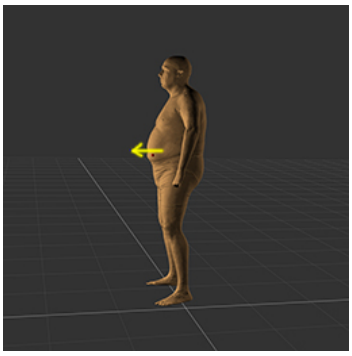
Sequence	Average Time(ms)	Best Time	Worst Time	IK (Avg)
Arm Raise	125	102	153	27
Boxing	131	100	206	29
Kick	154	92	378	31
Leg Split	126	100	161	31
Look Up	171	92	251	29
Walk	129	106	137	31
Overall	**134**	**92**	**378**	**30**

Table 4.1: Pose Manipulation Benchmark (timings in milliseconds).

Table 4.1 summarizes the timing results for each optimization between user events, i.e., the time spent searching for the optimal pose after each mouse drag. Note that currently we can only guarantee interactive rates, but further optimizations would be necessary to achieve real-time performance. The timings were measured in a MacBook Pro with Retina display, with a quad-core Intel Core i7 processor, with 16GB of memory, a NVIDIA GeForce GT 750M graphics card with 2GB of memory, running OS X 10.9.

We then ran a similar benchmark to measure the performance of pose manipulation. Figure 4.8 shows the shape manipulation sequences, and table 4.2, the timing results. Note that the timings do not include the initialization, which involves computing the shape basis (equation 1.7) and can be done only once since the pose parameters do not change. We trigger the initialization as soon as the user activates the tool. The initialization by itself took 120ms in our benchmark.

Finally, note that whenever the model parameters change, the model needs to be re-evaluated, which includes recalculating the vertex positions and updating the normals and tangents. During the benchmark, we also measured the time taken by the model update by itself, and in our current implementation it takes on average

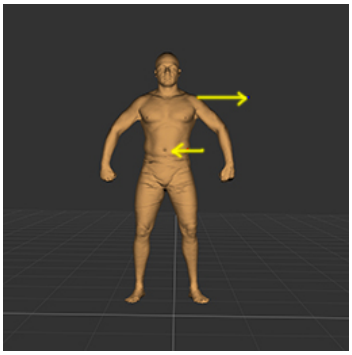


(a) Fatter

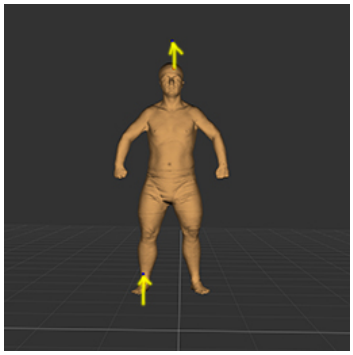
(b) Thinner

(c) Taller

(d) Shorter



(e) Stronger



(f) Short Legs

Figure 4.8: Shape manipulation sequences for the timing benchmark.

5ms per update.

4.6 Application to Body Warping

We conclude this chapter with an application of our direct body manipulation metaphor. We describe in this section how it can be used to indirectly warp images and point clouds of bodies, giving the user the impression that he is manipulating the image or point cloud directly.

Sequence	Average Time(ms)	Best Time	Worst Time
Fatter	48	38	53
Thinner	49	45	53
Taller	48	40	53
Shorter	49	41	57
Stronger	49	40	53
Short Legs	48	40	98
Overall	**49**	**38**	**98**

Table 4.2: Shape Manipulation Benchmark (timings in milliseconds). Times do not include the initialization, which took 120ms for this benchmark, and can be done only once before all manipulations take place.

This application is close in spirit to (Zhou et al. 2010) and (Jain et al. 2010), who describe a similar technique for images and videos. However, the main point here is that the manipulation is achieved by direct interaction with the object we are deforming, instead of indirectly controlled by sliders roughly linked to semantic attributes by a linear mapping to PCA coefficients (Allen et al. 2003). Here, one can make the shoulders wider, or make the hips thinner, for instance, by directly dragging on the image/point cloud. Moreover, to our knowledge, the extension to perform the body warping in a point cloud is also novel, which opens new possibilities for levels of realism in the deformation.

We perform our deformation on snapshots of RGB+Depth images of people, acquired with the Microsoft Kinect sensor, after fitting the model to the data with the algorithm we described in the last chapter.

4.6.1 Radial Basis Function Spatial Warping

As the body parameters are continuously modified, the mesh displacement naturally induces a vector field on $\mathbb{R}^3$, and on an image plane of the camera who is looking at the scene. We use this displacement to guide the warping.

We uniformly sample points $\mathbf{x}_i$ on the body mesh at the rest pose, who are transformed into $\mathbf{x}'_i$ at the current body parameters. For simplicity, in what follows $\mathbf{x}_i$ will be used to denote both planar displacements on the image plane or spatial displacements, since the method is completely analogous for the 2D and 3D cases, and when we say space one can think of either $\mathbb{R}^2$ or $\mathbb{R}^3$. We interpolate the displacements across the embedded space using radial basis functions, which induces, thus, a deformation on the image or point cloud.

More precisely, we want to interpolate the displacements $\delta_i = \mathbf{x}'_i - \mathbf{x}_i$ across the space. Defining

$$\delta(\mathbf{x}) = \sum_{k=1}^{K} \mathbf{w}_k \phi(||\mathbf{x} - \mathbf{c}_k||) + \mathbf{a}^T \begin{bmatrix} \mathbf{x} \\ 1 \end{bmatrix} \tag{4.12}$$

for the (x, y, z) coordinates displacements, the $\mathbf{c}_k$ corresponds to the centers of the radial basis ϕ, and $\mathbf{a}^T$ is a linear polynomial term used to give an overall approximation to the interpolation. We put a kernel centered on each body sample $\mathbf{x}_i$, and use polyharmonic splines of degree 3, $\phi(x) = x^3$, for reasons no other than that it was the lowest degree that produced good results, while avoiding the tweaking of radius parameters of Gaussian kernels, for instance.

Equation 4.12 must be solved for the weights w_i and linear coefficients $\mathbf{a}$, given the displacement constraints. This corresponds to solving a linear least squares

problem of the form

$$\begin{bmatrix} \Phi & X_h \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{a} \end{bmatrix} = \Delta. \tag{4.13}$$

Abusing the notation a little and setting $\Phi(\mathbf{x}) = [\ldots, \phi(||\mathbf{x} - \mathbf{x}_k||), \ldots]^T, k = 1, \ldots, K$, each row i of Φ, X_h and Δ in in 4.13 is defined by

$$\begin{bmatrix} \Phi(\mathbf{x}_i) & \mathbf{x}_i & 1 \end{bmatrix} \begin{bmatrix} \mathbf{w} \\ \mathbf{a} \end{bmatrix} = \delta_i = \mathbf{x}'_i - \mathbf{x}_i \tag{4.14}$$

for the x, y and z components. Again, note that only the right hand side changes as the body parameters change, so we can prefactor the normal equations of the least squares system, and quickly solve for the RBF weights and linear coefficients whenever the body parameters change. Equation 4.12 can then be evaluated at the locations we want to displace, which we evaluate directly on a vertex shader.

4.6.2 Direct Body Image Manipulation

First, on the 2D image setting, displacements are evaluated with the radial basis function 4.12 at pixel locations, with the weights and linear coefficients obtained by solving for the projection of the samples from the body mesh.

Figure 4.9 shows the result of "bulking up" a body by directly dragging the shoulders to make them wider, and making the arms shorter.

4.6.3 Direct Body Point Cloud Manipulation

A contribution of this chapter is the observation that we can also apply the spatial warping induced by the body deformation described in this section directly

Figure 4.9: "Bulking up" with our method. The first figure shows the original image, the middle figure, the result of our body fitting algorithm outlined in the appendix, and, on the right, the result of dragging the red to the blue dots to make the shoulders wider and the arms shorter (Note that we are aware of the border effects, we simply didn't place many points on the boundary to keep it completely still here, which is secondary to the main point we are trying to illustrate.)

on the point cloud in 3D. After solving for the weights and linear coefficients in equation 4.12, we apply the 3D displacements to the points on the cloud corresponding to the body in the depth image, allowing the introduction of effects not possible in the image setting, such as pushing out the belly to make the subject all around fatter, as shown in figure 4.10.

The displacements $\delta(\mathbf{x})$ are evaluated only on the points inside the body image mask, noting that, even though the points are defined in 3D, they still correspond to a 2D grid of depth image locations, which keeps the number of evaluations on the same order besides the additional dimension. The evaluation cost increases only because of the the RBF weights of the new dimension.

4.7 Conclusion

In this chapter we have visited the topic of body manipulation by user interaction. We first revisited the traditional method of specifying body shapes by directly

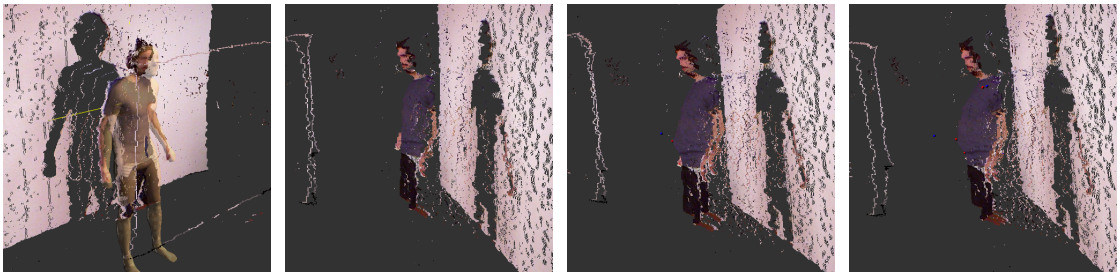


Figure 4.10: Point cloud deformation induced by direct manipulation. The first figure shows the undeformed point cloud and the model fit to the Kinect data. Then we show the belly progressively being pulled out to make the person look fatter.

setting the model parameters from linear mappings, improving on accuracy and describing a simple but practical way of inferring missing parameters from a body specification.

Our main contribution was to present a new body manipulation metaphor by directly interacting with the mesh. With our technique, a user can specify a body model shape and pose by directly dragging vertices on the body surface, and a "reasonable" body mesh is automatically computed.

As an application, we finished by showing how our direct manipulation metaphor can be used to manipulate color and RGBD images of people. We believe that our direct manipulation techniques could be a useful tool for the purpose of image annotation for consumption of machine learning algorithms in computer vision.

Chapter 5

Body Swapping and BodyJam

5.1 Introduction

The "fashion crisis": we have all suffered from it at one time or another, in the morning or before a party or a talk. We don't know what to wear, so we go through our closet and try out endless combinations of clothes. The consequences of this dreaded issue include arriving late to meetings, dates, and other events, and ending up in a bad outfit despite it all. Now there is help on the way. We introduce a new system called *BodyJam*, that lets you change your outfit with a finger snap. It is inspired by a long history of games based on a technique invented by the surrealists a century ago: 'Exquisite corpse," a method by which a collection of images (of body parts) is collectively assembled (Reverdy 1918) (see figure 5.1). Pages were folded into thirds, the top showing the head of a person or animal, the middle, the torso, and the bottom, the legs. Players 'mixed and matched" the sections by turning the pages. Over the years the technique has been transformed

(a) (b)

Figure 5.1: Exquisite Corpse (a) Nude. Cadavre Exquis with Yves Tanguy, Joan Miró, Max Morise, Man Ray, 1927. (b) By Andre Breton, Jacqueline Lamba and Yves Tanguy, 1938.

into children books, fashion retail websites, iPad apps, and art installations.

BodyJam does the same on a video display that mirrors the pose in real-time of a real-person standing in front of the camera/display mirror, and allows the user to change clothes and other appearance attributes. Using Microsoft's Kinect (Shotton et al. 2011), poses are matched to a video database of different torsos and legs. "Pages" are turned by gestures interpreted through the Kinect tracking. The current version of *BodyJam* is played by a single player, in choosing her/his appearance of the day with quick hand gestures. It differs from all previous "exquisite corpse" incarnations that the body pose mirrors the player in real-time. Any possible mix-and-matched outfit moves with and conforms to the player's pose and motions.

Our application strongly relies on the ability to produce photo-realistic ani-

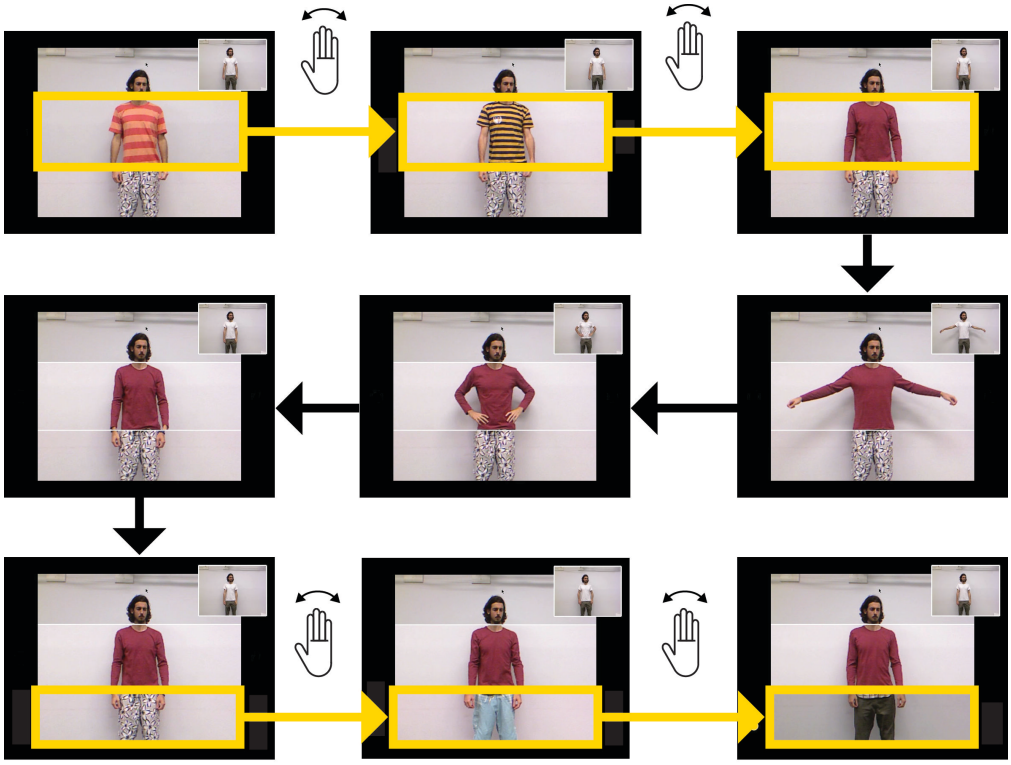


Figure 5.2: BodyJam: Overview of the application. Users can flip through different combinations of outfits by using hand gestures, while their poses are mimicked by the current outfit.

mations of people, one of the greatest challenges in current computer graphics research. An even harder problem is to do this in real-time such that it mirrors a user's pose and motion. Previously, many retargeting systems have been proposed that focus exclusively on facial remapping. When it comes to the domain of full body transfer in real-time, only sub-problems have been addressed so far. Most full body rendering solution are based on 3D mesh models, and texture mapping based cloth, or simulation based cloth (Stoll et al. 2010). The best photo-realistic

techniques so far have been based on image- or video-based rendering techniques (Schödl et al. 2002; Flagg et al. 2009). None of these techniques were linked to real-time marker-less input.

We propose a new system that takes advantage of recent progress in marker-less skeletal tracking techniques using Microsoft's Kinect. Unlike many example based rendering systems that need marker based data, our system uses Kinect based marker less annotation for the input video that is driving the animation *and* maker-less annotation for the video based render database.

Marker-less capture and rendering systems were limited so far only to faces. The face domain is much more constrained and has much less variations. Our current work is to the best of our knowledge the first system that allows this for full-body input motion, and full-body video based rendering techniques in real-time. Other potential applications include real-time puppeteering for new forms of theater and entertainment, real-time privacy filters, controlled appearance mapping for social and psychological studies, and many other domains.

Our system is divided as follows: First, we record the performance of one person (the model) wearing a piece of clothing, and create a database of the appearance of this piece of clothing from multiple poses. Later, we allow the original performance stored in the database to be controlled in real time by another person (the controller) using the Kinect sensor. The controller may wish not only to see the garment from different angles, but also how it would behave if he was wearing it and moving his or her arms or body, in real time. This ability to puppeteer the garment is a significant step towards the controller seeing him or her inside

the garment in real time and with a dynamic performance, i.e., towards a virtual mirror visual experience. This process is outlined in figure 5.3.

Furthermore, for the *BodyJam*, as outlined in figure 5.2, several image databases are combined in a clothing library, and then concurrently running databases can be used for the different body parts.

5.2 Related Work

As mentioned, *BodyJam* is inspired by the trail of games that started with century old parlor games called "Exquisite corpse" (Reverdy 1918) to most recent incarnations that extend it to video, face tracking, and hair/eye/mouth replacement, like Reface (Levin et al. 2007) by Golan Levin and Zach Lieberman. While Reface focuses exclusively on the face, it controls the appearance changes by gestures, in Reface's case, by eye-blinks detected in real-time with computer vision.

Computer graphics based full-body incarnations of this game have recently attracted interest by many fashion retail websites, starting with early versions by Glamour magazine (Glamour 2002), to H&M's current retail website's "Virtual Dressing Room" (H&M 2007), and new startups like Embodee's online try-on^{SM} experience (embodee 2011). All these applets are based on pre rendered still images, and pre-defined models. A very interesting extensions has been recently featured on JC Penny's website in collaboration with Seventeen Magazine's website using augmented reality and the web-browser's camera and a Flash plugin (Seventeen 20107). It has rudimentary face and body tracking, but it does not use video for the cloth appearance change. Instead, a pre-rendered still image is used.

Besides related application domains, *BodyJam*'s technology is also related to many face and body retargeting and re-writing systems. As we previously mentioned, most related approaches are focusing on the human face only. It is beyond the scope of this chapter to have an adequate summary of all facial retargeting systems. Video Rewrite was one of the first systems that used marker less annotation of large amounts of video data (Bregler et al. 1997), but only used audio as driving input. There have been several other systems that are controlled by audio, including (Brand 1999; Ezzat et al. 2002). Using the face as input to control a different output face has been reported by (Vlasic et al. 2005) using vision and subspace techniques. Most recently, (Weise et al. 2011) showed a Kinect-based real-time facial retargeting system. Many other approaches are based on either marker-based facial capture or vision-based tracking of facial features (Kemelmacher-Shlizerman et al. 2010) ((Deng et al. 2007) contains a survey of several different approaches).

Our technique is also related to the large body of literature that proposes full-body motion-capture based re-animation. Most techniques are inspired by graph-based structures derived from large motion-capture data. (Such systems were originally presented in the SIGGRAPH 2002 "motion-capture-soup" session (Kovar et al. 2002; Arikan et al. 2002; Pullen et al. 2002; Y. Li et al. 2002; Lee et al. 2002).) The input modality that drives these motion-capture based animation techniques are either 2D floor path directories or rough motion sketches, including key-frame animations. No video was used in these techniques. Again, it is beyond the scope of this chapter to give justice to the large amount of papers that were derived from these initial ideas.

Most closely related to our new techniques are more general video based techniques, like Video Sprites (Schödl et al. 2002), Human Video Textures (Flagg et al. 2009) and the video manipulation techniques of (Goldman et al. 2008). In those cases either matting-based extraction is used without explicit skeletal annotation, or a marker-based system in parallel to HD video acquisition is used, but no real-time video input is used to drive the animations. 3D extensions of video based acquisition techniques are recently reported by (De Aguiar et al. 2008; Huang et al. 2009; Xu et al. 2011), to name a few. Most recently, new dynamic simulation based cloth modeling have been incorporated into these 3D video based capture techniques (Stoll et al. 2010).

In addition to the novel application, our technique differs in two ways to previous approaches: 1) We circumvent the explicit 3D representation with an appearance based video database that samples the possible pose-space. 2) We utilize a fast real-time matching technique without the use of graph-based search (we also tried graph-based techniques, but they did not achieve better performance then our faster method using flat representations).

5.3 Overview

This chapter is organized as follows: First, we describe *Body Swapping*, the core technique on top of which we build our system, explaining how we create an image database from the model's performance, an then how the database can be used later for another person to control the model's performance in real time. Building of top if this new technique, we then describe our main application, *BodyJam*, detailing

Figure 5.3: Body Swapping: Overview of the technique.

how multiple image databases put together in a clothing library are used in our system for users to flip through different combinations of clothes. We conclude by showing results of our system in use.

5.4 Body Swapping: Image-based Puppeteering

The following two sections detail how we first create an image based database for the task of clothing visualization, and clothing-remapping in real-time.

5.4.1 Creating the Image Database

To build the image database for a piece of clothing, we dress a model with it and record a performance of him or her moving around, annotated by his/her 3D

Figure 5.4: Image database. This figure shows a few sample entries from an image database, where each entry holds an image from the video and the corresponding skeleton.

skeleton. We use the Kinect sensor to capture the performance, with the skeleton being computed by the OpenNI Framework (OpenNI n.d.). For each frame of the performance, captured at a constant frame rate, we create a database entry

containing the video frame image and the corresponding skeleton for the model's pose.

To establish a notation, an image database $D = \{E_f\}$ is a set of pairs $E_f = (S_f, I_f)$, composed of a 3D skeleton S_f and an image I_f extracted from the video of the performance. The video frame number is f. The skeleton $S_f = (J_j;\ j = 1, \ldots n)$, in turn, is composed of the 3D joint positions J_j. Figure 5.4 illustrates a database by showing a few frames from the video of a performance, along with the corresponding annotations.

5.4.2 Accessing the Image Database for Real Time Video Puppeteering

With a database D created from a previous recording in hands, we allow at a later time a second person (the controller) to control in real time the performance of the model stored in the database. This is accomplished by using the controller's skeleton $S(t)$, tracked in real time, at each moment t to query the database for the frame that best matches his or her current pose. First, we look for the entry $E_{f*} = (S_{f*}, I_{f*})$ containing the best matching skeleton

$$f^* = \underset{f}{\operatorname{argmin}}\, d(S_f, S(t)), \tag{5.1}$$

where d is a skeleton distance function we describe next, and then display image I_{f*} on the screen back to the controller, giving him the impression that the model is mimicking his performance.

Next, we address the details of the process outlined above:

- *Distance Function:* For the skeleton distance function d used to search the image database, we use a weighted sum of squared distance between the 3D joints of the two skeletons. Moreover, in order to make the control insensitive to translations, the skeletons are first centered around their torso's joint (note that the model is instructed to roughly move in place when recording the performance for the image database).

  More precisely, say that we want to compute the distance between skeletons $S = (J_i;\ i = 1, \ldots, n)$ and $S' = (J'_i;\ i = 1, \ldots, n)$. Their joints are first centered around the respective torsos J_T and J'_T, obtaining new, translated skeletons

$$\tilde{S} = (\tilde{J}_i;\ i = 1, \ldots, n) = (J_i - J_T;\ i = 1, \ldots, n),$$

$$\tilde{S}' = (\tilde{J}'_i;\ i = 1, \ldots, n) = (J'_i - J'_T;\ i = 1, \ldots, n).$$

  The distance is then computed as:

$$d(S, S') = d(\tilde{S}, \tilde{S}') = \sum_{i=1}^{n} w_i ||\tilde{J}_i - \tilde{J}'_i||^2, \tag{5.2}$$

  where the weights w_i are used to both improve the playback smoothness (e.g., joints on the torso should usually have higher weights than the limbs), as well as to eliminate some of the joints from the controlling altogether. For instance, if we are only interested in moving the upper body, the weights of the leg joints are set to zero.

  We also experimented with incorporating the joint velocities in addition to the positions, which is a simple matter of extending the database with more

Figure 5.5: Pose mimicking by the same person. Here we show a few frames from the real time control by the same person at a later time. The smaller rectangle shows the frame from the controller's real time performance, and the large rectangle, the corresponding frame fetched from the database shown back to him.

annotations. The velocities helped us resolve conflicts between nearby poses (say, an arm moving up vs moving down).

We should mention that we also tried to use the distance between the joint orientations quaternions, which are insensitive to the skeleton's bone sizes. However, simple joint positions yielded more robust results in general.

- *Nearest Neighbor Search*: For each query, we have to search through the whole database for the entry which holds the skeleton closest to the controller's current pose. We use straight linear search for that. We decided not to test more sophisticated nearest neighbor search algorithms, such as space partitioning approaches, since simple linear search already gave us reasonable real time performance. Clearly, on very large databases a better search

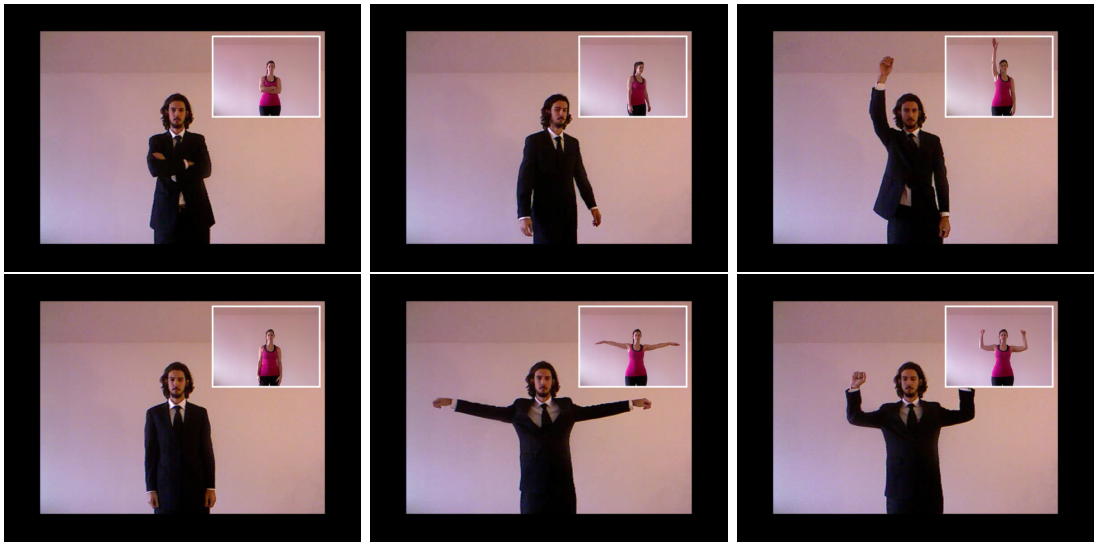


Figure 5.6: Pose mimicking by another person. Here we show a few frames from the real time control by a second person.

algorithm has to be employed.

- *"Hysteresis" Thresholding for Smoothness*: In order to remove jittering in the real time playback and try to keep the video smooth, we try to use nearby video frames in consecutive queries for as long as the skeleton distance stays within a threshold. That is, say that the query at one moment t returned the database entry $E_{f^*(t)} = (S_{f^*(t)}, I_{f^*(t)})$. For the next query, at time $t+1$, instead of searching the whole database, as in equation 5.1, we only use as candidates entries inside a window of width w around frame $f^*(t)$:

$$f^*(t+1) = \underset{\substack{f \\ |f-f^*(t)|\leqslant w}}{\operatorname{argmin}} \; d(S_f, S(t+1)), \tag{5.3}$$

where $S(t+1)$ is the controller's skeleton, and $f^*(t+1)$ will be the frame number displayed next. Typically, we used $w = 4$ in our experiments.

When the distance of the local optimum computed with equation 5.3 becomes too large, we allow a long transition by resorting back to searching the closest matching skeleton over the whole database using equation 5.1 instead.

The main reason why this removes jittering is that, when the original model moved around, he or she may have passed multiple times through nearby poses, which becomes a source of jittering in the real time playback. By using adjacent frames we try to force the system into using smoother video sequences present in the original recording.

- *Image Buffering*: We fetch the images from disk to main memory on demand when performing database queries. As an option to limit memory consumption in the case of large databases, we assign a memory budget on how many images we allow to be in memory at one given time. Then, we employ a LRU cache replacement policy by, when necessary, swapping first back to disk the frames with the oldest access time.

  Moreover, we use a simple predictive caching scheme, pre-loading to main memory a window of frames around the frame returned by a query.

- *Frame Discarding*: The system is organized around two main threads: the image database matching thread, which produces the best matching frame based on the controller's real time skeleton, and the rendering thread, which displays the matched frames on the screen. The matching thread adds frames to a queue, annotated with a timestamp of the query, and the rendering thread consumes frames from the queue. In order to avoid occasional long

lags between the controller's movement and the video that is displayed back to him, maintaining the feel of real-time control, the rendering thread discards the frames that are too old when dequeueing a new frame for display.

5.5 Skin Color Swapping

In order for the user to better identify him/herself with the body that is being shown on the screen, we also transfer his skin color to the model that was originally used to create the database. Our approach is to modify the images from the databases at runtime using a statistical model of the color distribution on both skin regions. We need not only a transformation that gives convincing results, but also that runs fast enough to be computed in real-time immediately after a new user steps in without disrupting the experience.

We will develop a transformation that alters an image from the database (a target image), based on a single image of the controller's face (the source image).

Skin Color Transfer We first transform both images from RGB space into $l\alpha\beta$ space (Ruderman et al. 1998). The details of the transformation from RGB to $l\alpha\beta$ space can be found in (Reinhard et al. 2001). In the discussion that follows, we assume the color is represented in $l\alpha\beta$ space. We model the skin color distribution of the target image $\mathbf{c}_t$ as a Gaussian

$$\mathbf{c}_t \sim \mathcal{N}(\mathbf{c}_t; \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t) \tag{5.4}$$

and the color distribution in the source image $\mathbf{c}_s$ as a mixture of Gaussians

$$\mathbf{c}_s \sim \sum_{i=1}^{n} \pi_i \mathcal{N}(\mathbf{c}_s; \boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i) \tag{5.5}$$

using the EM algorithm (Dempster et al. 1977). We use only two components for the Gaussian mixture of the face (i.e., $n = 2$), which appears to be enough in practice to model in one component the actual skin pixels and, in the other, pixels not corresponding to skin, such as eyes and hair pixels. We then use the Gaussian distribution responsible for the greater number of pixels to model the user's skin color distribution. Denote this component by $\mathcal{N}(\mathbf{c}_s; \boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s)$.

Having $\mathcal{N}(\mathbf{c}_s; \boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s)$ and $\mathcal{N}(\mathbf{c}_t; \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ describing the skin color distributions in the source and target images, respectively, we then transform each pixel in the skin region of the target image by warping the distribution $\mathcal{N}(\boldsymbol{c}_t; \boldsymbol{\mu}_t, \boldsymbol{\Sigma}_t)$ into $\mathcal{N}(\boldsymbol{c}_s; \boldsymbol{\mu}_s, \boldsymbol{\Sigma}_s)$. More precisely, let V_t be the 3×3 matrix of eigenvectors of Σ_t, with one eigenvector per column, and D_t a diagonal matrix holding the corresponding eigenvalues in the main diagonal. Define V_s and D_s the same way for Σ_s.

Each pixel $\boldsymbol{c}_t$ in the target image is then transformed by

$$\boldsymbol{c}'_t = D_s^{\frac{1}{2}} V_s D_t^{-\frac{1}{2}} V_t^T (\boldsymbol{c}_t - \boldsymbol{\mu}_t) + \boldsymbol{\mu}_s, \tag{5.6}$$

and then converted back to RGB space.

Figure 5.7 shows examples of our skin color transfer applied to transfer various skin tones to an image from one of our clothing databases.

Defining the Skin Masks To capture the skin region in the source image, whenever a new user jumps in, we run the Viola-Jones face detector (Viola et al.

Figure 5.7: Skin color transfer. The top left image shows the original image, and the others show the result of our skin color transfer transformation.

2001) to mask out the controller's face in the source image. The target skin region, on the other hand, can be computed offline, since it corresponds to images in the clothing database. We mask the skin regions by rotoscoping the database videos in Adobe After Effects using the Roto Brush tool. The result is that we simply have to store a skin area mask for each frame in the database, along with the Gaussian describing the skin color distribution of the reference model that was used to record the garment (also computed offline).

At runtime, our skin color transformation can be applied very fast, since all we have to do is estimate the Gaussian mixture inside the user face region in a single frame, and then use the resulting model to compute transformation (5.6). To further speed things up, transformation (5.6) and the $l\alpha\beta \leftrightarrow$ RGB color space conversions are computed in a fragment program in the GPU whenever we show

an image from the database.

Finally, as a comment, note that transformation (5.6) could have been directly applied in RGB space, but, in practice, this resulted in much more desaturated images and we opted for the $l\alpha\beta$ space instead.

5.6 BodyJam: Mixing and Matching Clothing Parts

We now describe our main application, *BodyJam*. Our system works as follows: The user moves in front of Microsoft's Kinect sensor, and a screen shows him or her dressed in different clothes. By using hand gestures, he/she can independently flip through the clothes dressing the upper and lower body. With this interface, the user is able to choose between different styles, patterns, colors, as well as to evaluate which garments go well together. Moreover, by making use of our *Body Swapping* technique for real time video puppeteering, users not only can see themselves in different clothes, but also control in real time the animation of the body.

As a general comment on perception that guided our design, our motivation in creating this system was to provide a nice experience for someone choosing what to dress from a library of garments. We considered various techniques to align and blend the controller's head with the body from the image database. However, not surprisingly, humans are extremely perceptive of any visual discrepancies that may arise in the result (a manifestation of the concept of Uncanny Valley introduced

by Mashiro Mori (Mori 1970)), so we chose to go in the opposite direction instead, making the separation of body parts even more explicit by drawing a horizontal line separating the different body segments. We believe the game-spirited approach of this interface leads users to be more forgiving to the discrepancies, compared to not-yet-mature real time techniques that would try to seamlessly mix heads and bodies.

The remainder of this section describes the details of the system.

5.6.1 Library of Clothing Databases

In order to allow users to browse different clothes, we record the performances of the reference model wearing various styles and colors of clothes. Each performance gives rise to a separate image database capturing the clothes' appearances from multiple poses. All the databases are then collected in a *clothing library*. Note that some databases were marked as being suitable exclusively for the upper body, others, just for the lower body, while some can be used for both. Figure 5.8 illustrates a database library by showing a selection of its databases.

5.6.2 Controlling the Three Separate Body Parts

Overview of the UI: The screen is divided in three separate stacked layers, as shown in figure 5.10: The upper layer shows the real time video of the controller's head, the middle layer shows the piece of clothing currently selected for the upper body (shirts, jackets, etc..), and, the bottom layer, the piece of clothing currently selected for the lower body (pants, skirts, etc…).

Figure 5.8: Image database library. This figure shows three entries from an image database library. Each row is a different clothing database, and, the columns, entries within the same database covering various poses.

We keep a library of pre-recorded clothes databases, and, at each moment, have one active for the upper body, and one for the lower body. The middle and bottom layers display the video outputs of these two concurrently running databases, driven by the user's real time skeleton. Each of these two videos, in addition to the cropped real time video of the user's head, is then texture mapped to its corresponding rectangle that is shown back to the user. Figure 5.5 shows a few frames from a real time performance.

Note that we use the whole skeleton even when driving the lower body database. This has the nice effect of making the arms and hands "cross the boundaries" and show up in the lower layer consistently with the upper body. Even though the alignment is not perfect most of the times, we believe that just seeing the arms crossing the video boundaries adds a nice visual effect to our system.

5.6.3 Aligning the Body Parts

In order to create the final composition of the three stacked layers, the real time video of the controller, the upper, and the lower body videos generated by the upper and lower body image databases have to be cropped, scaled and aligned:

- *Cropping*: We crop the video frames retrieved from a database feeding the upper body video between the neck and waist line. For that, we use the projection on Kinect's image plane of the 3D skeleton annotations contained in the result of a database query. When used for the lower body, we crop the frames below the waist line. The real time video of the controller, in turn, is cropped above the neck, this time using the real time skeleton tracked with the Kinect.

- *Alignment*: The three images are then aligned based on the projected skeletons. The real time head position is aligned with the neck position contained in the entry from the upper body database, and the lower body waist, in turn, is aligned with the upper body's.

- *Scaling*: Moreover, in addition to the alignment, the three videos have also to be properly scaled in order to generate a convincing final composition. Again, we use the projected joints for that. We scale the lower body in relation to the upper body based on the ratio of the projected torsos of each. The head is scaled in relation to the torso based on the distance from the neck to the head. For precision, we also allow an additional manual scale factor for small adjustments in cases where the size of the head is a little off.

5.6.4 Changing Clothes

We implemented three distinct ways of flipping through the clothes, *gesture driven switch*, *timed random switch* and a *hand tracking interface*:

- *Gesture Driven Switch*: When using hand gestures for control, at each moment we are either changing the clothes of the upper or the lower body, indicated to the user by two small yellow circles aligned with the active layer (see figure 5.11). A "push" gesture (moving the hand forward, towards the camera, and back) alternates between the two, and a hand waving gesture changes the piece of clothing of the active body part, accomplished in practice by switching the image database associated with it.

  The purpose of this mode is to offer a more playful interface, where users are "surprised" by the clothes chosen for them. This is useful, for instance, to promote a fixed set of looks with a very minimal interface, while still letting the user have some control of when to change clothes.

- *Timed Random Switch*: As an alternative, we also have a timed switch between clothes that randomly alternates between the databases available in the clothing library.

  This interface has the same purpose as above, except that it demands less effort from the user, which simply has to stand there, moving around, and let the system show him/her wearing the different combinations.

- *Hand Tracking Interface*: In a more realistic setting, users should be able to pick clothes from a catalog. For that, we also implemented a "hand

cursor" interface, where thumbnails of the available clothes are overlaid on the screen, and, by tracking the user's hand, he/she is able to pick different outfits by placing the cursor on top of the thumbnail of his choice of garment (figure 5.9).

5.7 Results and Applications

5.7.1 Body Swapping

Figure 5.4 illustrates an image database with a reference model dressed in a suit. The whole performance, lasting around 45 seconds, was then cropped in time to its usable part. Note that we manually delayed the video for a couple of frames when creating the database in order to compensate for the delay in the skeleton computation. The resulting image database, with about 1200 entries, was then used for the real time control.

Figure 5.5 shows a few frames from the real time control being used at a later time. In this example, the controller is the same person as the model, but has taken the suit off. Figure 5.6 shows a completely different person controlling the performance of the model in real time. Note that, despite the significant differences in body types between the controller and the reference model (her, being around 1.65m tall, while the reference model is 1.91m tall), the poses are very well matched and the video played back to her is reasonably smooth. The video included as supplementary material contains the real time performances.

Figure 5.9: Hand tracking interface.

Figure 5.10: Real time animation with the clothes fixed. Note that the shirt and board shorts were taken from two separate image databases. The small rectangle shows the video of the controller, while the larger one, the content of the screen shown back to him.

5.7.2 Floating Garment

Figure 5.12 shows another application of our technique. This time, using an HD camera, we recorded the performance in front of a green screen, with the model also wearing a green suit under a dark red sweater (figure 5.12a). Next, the footage was keyed out, leaving only the garment "floating" in the air by itself (figure 5.12b). Finally, after manually synchronizing the HD camera and Kinect footages, we used the resulting image database to manipulate the floating sweater in real time, as show in subfigure 5.12c. Again, the video included as supplementary material

contains a clip of the performance.

5.7.3 BodyJam

Figure 5.8 shows entries from a few sample of databases composing a library of clothes. To build such a library, we first recorded with the Kinect sensor performances of the reference model wearing various outfits, each lasting around 1 minute, which were then cropped in time to its usable part. Finally, the resulting 8 image databases were put together in a clothing library, each being available for use both as upper or lower body clothes during the real time performance. The entire recording can be seen fast-forwarded in the video included as supplementary material.

Figure 5.10 shows the real time control of one picked outfit composed of clothes from two separate databases. We can see that the body is mimicking the controller's pose, shown with the real time video inside the small rectangle. In figure 5.11, the user is seen first picking different shirts, driven by the waving gesture, while keeping the shorts fixed, and then switches to start picking shorts and pants while keeping the chosen red shirt fixed. The video included as supplementary material contains the whole performance of the user flipping through different outfits through hand gestures, as well as a clip of the clothes randomly changing. The whole process can be seen in a smaller scale in figure 5.2.

Figure 5.11: On the first row, we are changing the clothes of the upper body, keeping the clothes of the lower body fixed. While, on the second, we change the clothing of the lower body. The yellow circles indicate to the user which body segment is currently active for changing clothes.

5.8 Conclusions

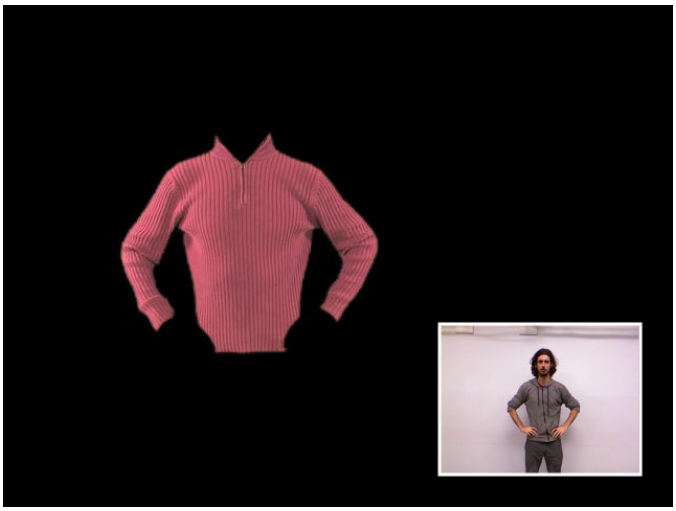
We presented a new system called *BodyJam*, that lets you change your outfit with a finger snap. *BodyJam* was inspired by a technique invented by the surrealists a century ago: "Exquisite corpse," a method by which a collection of images (of body parts) is collectively assembled.

BodyJam applies the "Exquisite corpse" on a video display that mirrors the pose in real-time of a real-person standing in front of the camera/display mirror, and allows the user to change clothes and other appearance attributes. Using Microsoft's Kinect, poses are matched to a video database of different torsos and legs, and "pages" are turned by hand gestures.

A limitation of our current system is that it is currently not able to handle

(a) Performance with the Green Suit

(b) Keyed Out Garment Footage



(c) Real time control of the floating garment

Figure 5.12: Floating Garment. This figure shows another application of our technique, where the controller drives the pose of a floating garment created by chroma keying.

poses radically different from the ones present in the database. We would like to address this issue in the future by employing image warping techniques, where the warping of the closest matched image could be guided by the controller's pose, retargeted to the skeleton of the model stored in the database. Currently, it is not a major limitation due to the scope of our application (i.e., people undergo a very limited set of poses when trying out clothes, and we are able to easily cover

that space with a large enough database), but we would like to further explore this direction in order to broaden the scope of our technique.

It would also be interesting to see the effects of warping the model's body in the clothes database in order to match the controller's body measurements, employing the techniques described in the last two chapters in a full, end-to-end system.

Chapter 6

Conclusions

In this thesis we have touched the topics of human body acquisition, modeling and manipulation. In chapter 1 we reviewed the exisiting SCAPE body model, adding important observations for its efficient use and optimization. In chapter 2, we introduced a new practical registration algorithm used to register thousands of high resolution body scans, which generated a model that was then fitted to noisy depth maps in a fast algorithm introduced in chapter 3. We then moved to the topic of body manipulation, where our main contribution was a new body manipulation technique based on direct interaction with the body mesh, showing how it can be applied to directly deform images and point clouds of bodies. Finally, we conclude with a real-time system for video-based body manipulation inspired by a technique invented by the surrealists.

Appendix A

Mathematical Background

A.1 Lie Algebra of the Rotation Group

Since we make heavy use of the Lie algebra of the rotation group, we make a quick review here and refer the reader to Murray et al. 1994, Hall 2003 and Varadarajan 1984 for more details on the subject.

The Lie algebra of $SO(3)$, denoted by $\mathfrak{so}(3)$, is the vector space of 3×3 skew symmetric matrices, which, in turn, is isomorphic to $\mathbb{R}^3$ with the mapping

$$\omega = (\omega_x, \omega_y, \omega_y) \mapsto \widehat{\omega} = [\omega]^\wedge = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{A.1}$$

Without the hat we'll be referring to the vector in $\mathbb{R}^3$, and, with the hat, to the skew-symmetric matrix. ω corresponds to the rotation axis and its length, the rotation angle.

A.1.1 The Exponential Map

The connection between a Lie Algebra and the corresponding matrix group is given by the exponential map

$$\exp(\mathbf{A}) = \sum_{i=0}^{\infty} \frac{\mathbf{A}^i}{i!}, \tag{A.2}$$

which, for the rotation group in particular, can be computed in closed form with the Rodrigues' rotation formula

$$\exp(\widehat{\omega}) = \mathbf{I} + \sin|\omega| \frac{\widehat{\omega}}{|\omega|} + (1 - \cos|\omega|) \frac{\widehat{\omega}^2}{|\omega|^2}, \tag{A.3}$$

which can be easily derived from the Taylor series expansions of $\sin|\omega|$ and $\cos|\omega|$ after noticing that the odd and even terms of the series A.2 simplify to to $\widehat{\omega}^{2i+1} = (-1)^i|\omega|^{2i}\widehat{\omega}$ and $\widehat{\omega}^{2i} = (-1)^{i-1}|\omega|^{2(i-1)}\widehat{\omega}^2$, respectively.

The following observation gives a more kinematic interpretation to the rotation group and its Lie Algebra: If a point $\mathbf{P}$ is rotating around an axis ω, its instantaneous velocity is given by $\omega \times \mathbf{P} = \widehat{\omega}\mathbf{P}$. Since $\exp(t\omega)\mathbf{P}_0$ is the solution to the ordinary differential equation

$$\dot{\mathbf{P}} = \widehat{\omega}\mathbf{P}, \tag{A.4}$$

we can see that $\mathbf{R} = \exp(t\omega)$ represents a rotation by an angle $t|\omega|$ around the axis ω.

A.1.2 The Logarithmic Map

The inverse mapping, the matrix logarithm, is defined for where the series

$$\log(\mathbf{A}) = \sum_{i=1}^{\infty} (-1)^{i+1} \frac{(\mathbf{A} - \mathbf{I})^i}{i} \tag{A.5}$$

converges. For $\mathbf{R} \in SO(3)$ in particular, the mapping is defined everywhere as long as special care is taken at the singularities. From the Rodrigues' formula, we have that $\mathrm{Tr}(\mathbf{R}) = 1 + 2\cos\theta$, where $\theta = |\omega|$ is the rotation angle. The rotation angle, therefore, is given by

$$\theta = \arccos\left(\frac{\mathrm{Tr}(\mathbf{R}) - 1}{2}\right). \tag{A.6}$$

Moreover, also from the Rodrigues' formula, we have that $\mathbf{R} - \mathbf{R}^T = \frac{2\sin\theta}{\theta}\widehat{\omega}$, since the symmetric terms cancel out and, therefore,

$$\log(\mathbf{R}) = \widehat{\omega} = \begin{cases} 0 & \text{if } \theta = 0 \\ \frac{\theta}{2\sin\theta}(\mathbf{R} - \mathbf{R}^T) & \text{otherwise} \end{cases} \tag{A.7}$$

Using the canonical basis of $\mathbb{R}^3$, the parameterization induced by the exponential map

$$\exp(\widehat{\omega}) = \exp(\omega_x\widehat{\mathbf{e}}_x + \omega_y\widehat{\mathbf{e}}_y + \omega_z\widehat{\mathbf{e}}_z), \qquad |\omega| < \pi \tag{A.8}$$

can be used to bijectively parameterize *almost* the whole group, *almost* because of the the singularity that occurs at the radius π of sphere boundary, where the antipodal points ω and $-\omega$ represent the same rotation by 180 degrees around the axis ω.

Bibliography

Allen, Brett, Brian Curless, and Zoran Popović (July 2003). "The space of human body shapes: reconstruction and parameterization from range scans". In: *ACM Trans. Graph.* 22.3, pp. 587–594. ISSN: 0730-0301.

Anguelov, Dragomir, Praveen Srinivasan, Hoi-cheung Pang, and Daphne Koller (2004). "The correlated correspondence algorithm for unsupervised registration of nonrigid surfaces". In: *In TR-SAIL-2004-100*, pp. 33–40.

Anguelov, Dragomir, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis (2005). "SCAPE: shape completion and animation of people". In: *ACM Trans. Graph* 24, pp. 408–416.

Arikan, O. and D. Forsyth (2002). "Interactive motion generation from examples". In: *ACM Transactions on Graphics* 21.3, pp. 483–490.

Baran, Ilya and Jovan Popović (July 2007). "Automatic rigging and animation of 3D characters". In: *ACM Trans. Graph.* 26.3. ISSN: 0730-0301.

Besl, Paul J. and Neil D. McKay (Feb. 1992). "A Method for Registration of 3-D Shapes". In: *IEEE Trans. Pattern Anal. Mach. Intell.* 14.2, pp. 239–256. ISSN: 0162-8828. DOI: 10.1109/34.121791.

Bodymetrics (n.d.). *http://www.bodymetrics.com/*.

BIBLIOGRAPHY

Botsch, Mario and Leif Kobbelt (2005). "Real-time shape editing using radial basis functions". In: *Computer Graphics Forum*, pp. 611–621.

Brand, M. (1999). "Voice puppetry". In: *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., pp. 21–28.

Bregler, C., M. Covell, and M. Slaney (1997). "Video rewrite: Driving visual speech with audio". In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., pp. 353–360.

Brown, Benedict and Szymon Rusinkiewicz (Aug. 2007). "Global Non-Rigid Alignment of 3-D Scans". In: *ACM Transactions on Graphics (Proc. SIGGRAPH)* 26.3.

Chang, W. and M. Zwicker (2009). "Range scan registration using reduced deformable models". In: *EG*.

Chen, Yanqing, Timothy A. Davis, William W. Hager, and Sivasankaran Rajamanickam (Oct. 2008). "Algorithm 887: CHOLMOD, Supernodal Sparse Cholesky Factorization and Update/Downdate". In: *ACM Trans. Math. Softw.* 35.3, 22:1–22:14. ISSN: 0098-3500.

Chen, Y. and G. Medioni (1991). "Object modeling by registration of multiple range images". In: *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, 2724–2729 vol.3. DOI: 10.1109/ROBOT.1991.132043.

Crane, Keenan, Clarisse Weischedel, and Max Wardetzky (2013). "Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow". In: *ACM Trans. Graph.*

BIBLIOGRAPHY

Cyberware (n.d.). *http://www.cyberware.com/*.

De Aguiar, E., C. Stoll, C. Theobalt, N. Ahmed, H.P. Seidel, and S. Thrun (2008). "Performance capture from sparse multi-view video". In: *ACM Transactions on Graphics (TOG)*. Vol. 27. ACM, p. 98.

Deng, Z. and J. Noh (2007). "Computer facial animation: A survey". In: *Data-Driven 3D Facial Animation*, pp. 1–28.

Dürer, Albrecht (1528). *Four Books on Human Proportion (Hierinn sind begriffen vier Bücher von menschlicher Proportion)*. Nuremberg: Agnes Dürer.

eigen.tuxfamily.org (n.d.). *Eigen Library*.

Dempster, A. P., N. M. Laird, and D. B. Rubin (1977). "Maximum likelihood from incomplete data via the EM algorithm". In: *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B* 39.1, pp. 1–38.

embodee (2011). *Embodee's online try-on^{SM} experience*. `http://hurley.embodee.com/try-on/`.

Reverdy, Pierre (1918). *Exquisite Corpse*. `http://en.wikipedia.org/wiki/Exquisite_corpse`.

Ezzat, Tony, Gadi Geiger, and Tomaso Poggio (2002). "Trainable videorealistic speech animation". In: *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '02. San Antonio, Texas: ACM, pp. 388–398. ISBN: 1-58113-521-1. DOI: `http://doi.acm.org/10.1145/566570.566594`.

Flagg, M., A. Nakazawa, Q. Zhang, S.B. Kang, Y.K. Ryu, I. Essa, and J.M. Rehg (2009). "Human video textures". In: *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. ACM, pp. 199–206.

BIBLIOGRAPHY

Forsyth, David A., Okan Arikan, and Deva Ramanan (2006). "D.: Computational Studies of Human Motion: Part 1, Tracking and Motion Synthesis". In: *Foundations and Trends in Computer Graphics and Vision*. Now Publishers Inc, p. 2006.

Glamour (2002). *Glamours Virtual Dressing Room Draws Readers Advertisers*. http://www.dmnews.com/glamours-virtual-dressing-room-draws-readers-advertisers/article/81906/.

Goldman, Dan B., Chris Gonterman, Brian Curless, David Salesin, and Steven M. Seitz (2008). "Video object annotation, navigation, and composition". In: *UIST*, pp. 3–12.

Grochow, Keith, Steven L. Martin, Aaron Hertzmann, and Zoran Popović (2004). "Style-based Inverse Kinematics". In: *ACM SIGGRAPH 2004 Papers*. SIGGRAPH '04. Los Angeles, California: ACM, pp. 522–531. DOI: 10.1145/1186562.1015755.

Hall, B. (2003). *Lie Groups, Lie Algebras, and Representations: An Elementary Introduction*. Graduate Texts in Mathematics. Springer. ISBN: 9780387401225.

Weise, Thibaut, Sofien Bouaziz, Hao Li, and Mark Pauly (July 2011). "Realtime Performance-Based Facial Animation". In: *ACM Transactions on Graphics (Proceedings SIGGRAPH 2011)* 30.4.

Hasler, N., C. Stoll, M. Sunkel, B. Rosenhahn, H. -p. Seidel, and Mpi Informatik (2009). "A statistical model of human pose and body shape. Computer Graphics Forum28". In:

H&M (2007). *H & M's Virtual Dressing Room*. http://www.hm.com/us/dressingroom.

BIBLIOGRAPHY

Huang, P., A. Hilton, and J. Starck (2009). "Human motion synthesis from 3D video". In: *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, pp. 1478–1485.

K. Madsen H. B. Nielsen, O. Tingleff (April 2004). *Methods for Non-linear Least Squares Problems*. Second. Technical University of Denmark.

Jacobson, Alec, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine (2012). "Fast Automatic Skinning Transformations". In: *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH)* 30.4, 77:1–77:10.

Jain, Arjun, Thorsten Thormählen, Hans-Peter Seidel, and Christian Theobalt (2010). "MovieReshape: Tracking and Reshaping of Humans in Videos". In: *ACM Trans. Graph. (Proc. SIGGRAPH Asia 2010)* 29.5.

Kavan, Ladislav, Steven Collins, Jiří Žára, and Carol O'Sullivan (Nov. 2008). "Geometric skinning with approximate dual quaternion blending". In: *ACM Trans. Graph.* 27.4, 105:1–105:23. ISSN: 0730-0301.

Kemelmacher-Shlizerman, Ira, Aditya Sankar, Eli Shechtman, and Steven M. Seitz (2010). "Being John Malkovich". In: *ECCV (1)*, pp. 341–353.

Kim, Vladimir G., Yaron Lipman, and Thomas Funkhouser (2011). "Blended intrinsic maps". In: *ACM SIGGRAPH 2011 papers*. SIGGRAPH '11. Vancouver, British Columbia, Canada: ACM, 79:1–79:12. ISBN: 978-1-4503-0943-1. DOI: 10.1145/1964921.1964974.

Shotton, Jamie, Andrew Fitzgibbon, Mat Cook, Toby Sharp, Mark Finocchio, Richard Moore, Alex Kipman, and Andrew Blake (2011). "Real-Time Human Pose Recognition in Parts from a Single Depth Image". In:

Cui, Yan, Will Chang, Tobias Nöll, and Didier Stricker (2012). "Kinectavatar: Fully Automatic Body Capture Using a Single Kinect". In: *ACCV Workshop on Color*

Depth Fusion in Computer Vision 2012 . Workshop on Color Depth Fusion in Computer Vision (ACCV-12), November 5-9, Daejeon, Korea, Republic of. KAIST, KROS. ACCV.

Kovar, L., M. Gleicher, and F. Pighin (2002). "Motion graphs". In: *ACM Transactions on Graphics (TOG)* 21.3, pp. 473–482.

Lee, J., J. Chai, P.S.A. Reitsma, J.K. Hodgins, and N.S. Pollard (2002). "Interactive control of avatars animated with human motion data". In: *ACM Transactions on Graphics* 21.3, pp. 491–500.

Levenberg, Kenneth (1944). "A Method for the Solution of Certain Non-Linear Problems in Least Squares". In: *The Quarterly of Applied Mathematics* 2, pp. 164–168.

Lewis, J. P., Matt Cordner, and Nickson Fong (2000). "Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation". In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. SIGGRAPH '00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., pp. 165–172. ISBN: 1-58113-208-5.

Li, Y., T. Wang, and H.Y. Shum (2002). "Motion texture: a two-level statistical model for character motion synthesis". In: *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*. ACM, pp. 465–472.

Li, Hao, Linjie Luo, Daniel Vlasic, Pieter Peers, Jovan Popović, Mark Pauly, and Szymon Rusinkiewicz (Jan. 2012). "Temporally Coherent Completion of Dynamic Shapes". In: *ACM Transactions on Graphics* 31.1.

Marquardt, Donald W. (1963). "An algorithm for least-squares estimation of nonlinear parameters". In: *SIAM Journal on Applied Mathematics* 11.2, pp. 431–441. DOI: 10.1137/0111030.

BIBLIOGRAPHY

Moakher, Maher (Jan. 2002). "Means and Averaging in the Group of Rotations". In: *SIAM J. Matrix Anal. Appl.* 24.1, pp. 1–16. ISSN: 0895-4798. DOI: 10.1137/S0895479801383877.

Mori, Masahiro (1970). "Bukimi no tani. The uncanny valley (in Japanese)". In: *Energy* (7), 33–35.

Murray, Richard M., S. Shankar Sastry, and Li Zexiang (1994). *A Mathematical Introduction to Robotic Manipulation*. 1st. Boca Raton, FL, USA: CRC Press, Inc. ISBN: 0849379814.

Neugebauer, P. J. (1997). "Geometrical cloning of 3D objects via simultaneous registration of multiple range images". In: *Proceedings of the 1997 International Conference on Shape Modeling and Applications (SMA '97)*. SMA '97. Washington, DC, USA: IEEE Computer Society, pp. 130–. ISBN: 0-8186-7867-4.

Nocedal, Jorge and Stephen J. Wright (2006). *Numerical optimization*. 2. ed. Springer series in operations research and financial engineering. New York, NY: Springer. XXII, 664. ISBN: 978-0-387-30303-1.

OpenNI (n.d.). *www.openni.org*.

Press, William H., Saul A. Teukolsky, William T. Vetterling, and Brian P. Flannery (2007). *Numerical Recipes 3rd Edition: The Art of Scientific Computing*. 3rd ed. New York, NY, USA: Cambridge University Press. ISBN: 0521880688, 9780521880688.

Pullen, Katherine and Christoph Bregler (2002). "Motion capture assisted animation: texturing and synthesis". In: *ACM Transactions on Graphics (SIGGRAPH 2002)* 21.3, pp. 501–508.

Levin, Golan and Zachary Lieberman (2007). *Reface [Portrait Sequencer]*. Bitforms gallery, NYC, http://www.flong.com/projects/reface/.

BIBLIOGRAPHY

Reinhard, Erik, Michael Ashikhmin, Bruce Gooch, and Peter Shirley (Sept. 2001). "Color Transfer between Images". In: *IEEE Comput. Graph. Appl.* 21.5, pp. 34–41. ISSN: 0272-1716.

Ruderman, D. L., T. W. Cronin, and C. C. Chiao (1998). "Statistics of cone responses to natural images: implications for visual coding". In: *Journal of the Optical Society of America A* 15.8, pp. 2036–2045.

Rusinkiewicz, Szymon and Marc Levoy (June 2001). "Efficient Variants of the ICP Algorithm". In: *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*.

Schaefer, Scott, Travis McPhail, and Joe Warren (July 2006). "Image deformation using moving least squares". In: *ACM Trans. Graph.* 25.3, pp. 533–540. ISSN: 0730-0301.

Schödl, A. and I.A. Essa (2002). "Controlled animation of video sprites". In: *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*. ACM, pp. 121–127.

Sederberg, Thomas W. and Scott R. Parry (Aug. 1986). "Free-form deformation of solid geometric models". In: *SIGGRAPH Comput. Graph.* 20.4, pp. 151–160. ISSN: 0097-8930.

Seventeen (20107). *JC Penny Augmented Reality 'Virtual dressing room'*. `ttp://www.seventeen.com/fashion/virtual-dressing-room`.

Shoemake, Ken (1994). "Graphics gems IV". In: *Graphics gems IV*. Ed. by Paul S. Heckbert. San Diego, CA, USA: Academic Press Professional, Inc. ISBN: 0-12-336155-9.

BIBLIOGRAPHY

Sloan, Peter-Pike J., Charles F. Rose III, and Michael F. Cohen (2001). "Shape by example". In: *Proceedings of the 2001 symposium on Interactive 3D graphics*. I3D '01. New York, NY, USA: ACM, pp. 135–143. ISBN: 1-58113-292-1.

Stoll, C., J. Gall, E. de Aguiar, S. Thrun, and C. Theobalt (2010). "Video-based reconstruction of animatable human characters". In: *ACM Transactions on Graphics (TOG)*. Vol. 29. 6. ACM, p. 139.

Styku (n.d.). *http://www.styku.com/*.

Sumner, Robert W., Matthias Zwicker, Craig Gotsman, and Jovan Popović (July 2005). "Mesh-based inverse kinematics". In: *ACM Trans. Graph.* 24.3, pp. 488–495. ISSN: 0730-0301.

Tong, Jing, Jin Zhou, Ligang Liu, Zhigeng Pan, and Hao Yan (2012). "Scanning 3d full human bodies using kinects". In: *Visualization and Computer Graphics, IEEE Transactions on* 18.4, pp. 643–650.

Felzenszwalb, Pedro F. and Daniel P. Huttenlocher (2012). "Distance Transforms of Sampled Functions". In: *Theory of Computing* 8.19, pp. 415–428.

Varadarajan, V.S. (1984). *Lie Groups, Lie Algebras, and Their Representation*. Graduate Texts in Mathematics. Springer. ISBN: 9780387909691.

Vicon (n.d.). *http://www.vicon.com*.

Viola, P. and M. Jones (2001). "Rapid object detection using a boosted cascade of simple features". In: *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. Vol. 1, pages. DOI: 10.1109/CVPR.2001.990517.

Pollio, Marcus Vitruvius (n.d.). *De Architectura*.

BIBLIOGRAPHY

Vlasic, D., M. Brand, H. Pfister, and J. Popović (2005). "Face transfer with multilinear models". In: *ACM Transactions on Graphics (TOG)* 24.3, pp. 426–433.

Wang, Ruizhe, Jongmoo Choi, and Gérard G. Medioni (2012). "Accurate Full Body Scanning from a Single Fixed 3D Camera". In: *3DIMPVT*, pp. 432–439.

Weiss, Alexander, David Hirshberg, and Michael J. Black (2011). "Home 3D body scans from noisy image and range data". In: *Proceedings of the 2011 International Conference on Computer Vision*. ICCV '11. Washington, DC, USA: IEEE Computer Society, pp. 1951–1958. ISBN: 978-1-4577-1101-5.

Xu, Feng, Yebin Liu, Carsten Stoll, James Tompkin, Gaurav Bharaj, Qionghai Dai, Hans-Peter Seidel, Jan Kautz, and Christian Theobalt (Aug. 2011). "Video-based characters: creating new human performances from a multi-view video database". In: *ACM Trans. Graph.* 30 (4), 32:1–32:10. ISSN: 0730-0301. DOI: http://doi.acm.org/10.1145/2010324.1964927.

Zhou, Shizhe, Hongbo Fu, Ligang Liu, Daniel Cohen-Or, and Xiaoguang Han (July 2010). "Parametric reshaping of human bodies in images". In: *ACM Trans. Graph.* 29 (4), 126:1–126:10. ISSN: 0730-0301.