Surface motion capture animation

Adnane Boukhayma

UNIVERSITÉ DE GRENOBLE

THÈSE

Pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ DE GRENOBLE

Spécialité : **Mathématiques, Sciences et Technologies de l'Information**

Arrêté ministériel : 7 août 2006

Présentée par

Adnane Boukhayma

Thèse dirigée par **Edmond Boyer**

préparée au sein **Inria Grenoble**
et de **l'école doctorale MSTII : Mathématiques, Sciences et Technologies de l'Information, Informatique**

Animation de Capture de Mouvement de Surface

Surface Motion Capture Animation

Thèse soutenue publiquement le **6 décembre 2017**,
devant le jury composé de :

Pr. Adrian Hilton
University of Surrey, Guilford, UK, Rapporteur

Dr. Gerard Pons Moll
Max Planck Institute of Intelligent Systems, Tübingen, Germany, Rapporteur

Pr. Céline Loscos (Président du jury)
Université de Reims, France, Examinateur

Dr. Jean-Sébastien Franco
Grenoble INP, France, Examinateur

Dr. Edmond Boyer
Inria Grenoble Rhône-Alpes, France, Directeur de thèse

Abstract

As a new alternative to standard motion capture, 4D surface capture is motivated by the increasing demand from media production for highly realistic 3D content. Such data provides real full shape, appearance and kinematic information of the dynamic object of interest. We address in this work some of the tasks related to the acquisition and the exploitation of 4D data, as obtained through multi-view videos, with an emphasis on corpus of moving subjects. Some of these problems have already received a great deal of interest from the graphics and vision communities, but a number of challenges remain open in this respect. We address namely example based animation synthesis, appearance modelling, semantic motion transfer and variation synthesis.

We first propose a method to generate animations using video-based mesh sequences of elementary movements of a shape. New motions that satisfy high-level user-specified constraints are built by recombining and interpolating the frames in the observed mesh sequences. Our method brings local improvement to the synthesis process through optimized interpolated transitions, and global improvement with an optimal organizing structure that we call the essential graph.

We then address the problem of building efficient appearance representations of shapes observed from multiple viewpoints and in several movements. We propose a per subject representation that identifies the underlying manifold structure of the appearance information relative to a shape. The resulting representation encodes shape appearance variabilities due to viewpoint and illumination, with Eigen textures, and due to local inaccuracies in the geometric model, with Eigen warps. In addition to providing compact representations, such decompositions also allow for appearance interpolation and appearance completion.

We additionally address the problem of transferring motion between captured 4D models. Given 4D training sets for two subjects for which a sparse set of semantically corresponding key-poses are known, our method is able to transfer a newly captured motion from one subject to the other. The method contributes a new transfer model based on non-linear pose and displacement interpolation that builds on Gaussian process regression.

Finally, we propose a data based solution for generating variations of captured 4D models, for automatic 4D dataset augmentation and realism improvement. Given a few 4D models representing movements of the same type, our method builds a probabilistic low dimensional embedding of shape poses using Gaussian Process Dynamical Models, and novel variants of motions are obtained by sampling trajectories from this manifold using Monte Carlo Markov Chain. We can synthesise an unlimited number of variations of any of the input movements, and also any blended version of them. The output variations are statistically similar to the input movements but yet

slightly different in poses and timings.

Keywords. Video-based animation • Multiview reconstruction • 4D modelling • Motion transfer • Essential graph • Eigen appearance maps • Gaussian process • Motion variation •

Résumé

En tant qu'une nouvelle alternative à la MoCap standard, la capture de surface 4D est motivée par la demande croissante des produits médiatiques de contenu 3D très réaliste. Ce type de données fournit une information complète et réelle de la forme, l'apparence et la cinématique de l'objet dynamique d'intérêt. On aborde dans cet ouvrage certaines taches liées à l'acquisition et l'exploitation de données 4D, obtenues à travers les vidéos multi-vues, avec un intérêt particulier aux formes humaines en mouvement. Parmi ces problèmes, certains ont déjà reçu beaucoup d'intérêt de la part des communautés de vision par ordinateur et d'infographie, mais plusieurs challenges restent ouverts à cet égard. En particulier, nous abordons la synthèse d'animation basée sur des exemples, la modélisation d'apparence, le transfert sémantique de mouvement et la géneration de variation de mouvement.

On introduit premièrement une méthode pour générer des animations en utilisant des séquences de maillages de mouvements élémentaires d'une forme donnée. De nouveaux mouvements satisfaisant des contraintes haut-niveau sont construites en combinant et interpolant les trames des données observées. Notre méthode apporte une amélioration local au processus de la synthèse grâce à l'optimisation des transitions interpolées, ainsi qu'une amélioration globale avec une structure organisatrice optimale qu'on appelle le graph essentiel.

On aborde en suite la construction de représentations efficaces de l'apparence de formes en mouvement observées à travers des vidéos multi-vue. On propose une représentation par sujet qui identifie la structure sous-jacente de l'information d'apparence relative à une forme particulière. La représentation propre résultante encode les variabilités d'apparence dues au point de vue et au mouvement avec les textures propres, et celles dues aux imprécisions locales dans le modèle géométrique avec les déformations propres. Outre fournir des représentation compactes, ces décompositions permettent aussi l'interpolation et la complétion des apparences.

On s'intéresse aussi au problème de transfert de mouvement entre deux modèles 4D capturés. Étant donné des ensembles d'apprentissages pour deux sujets avec des correspondances sémantiques éparses entre des poses clés, notre méthode est capable de transférer un nouveau mouvement capturé d'un sujet vers l'autre. Nous contribuons principalement un nouveau modèle de transfert basé sur l'interpolation non-linéaire de pose et de déplacement qui utilise les processus Gaussiens de régression.

Finalement, on propose une solution pour générer des variations de modèles 4D capturés, pour l'augmentation automatique et l'amélioration du réalisme des bases de données 4D. Étant donné quelques modèles 4D représentant des mouvements du même type, notre méthode construit un plongement probabiliste à basse dimension des poses en utilisant les modèles

de processus Gaussiens dynamiques, et des variantes nouvelles de mouvement sont obtenues en échantillonnant des trajectoires dans l'espace sous-jacent grâce à une chaine de Markov de Monte Carlo. On peut synthétiser un nombre illimité de variations des mouvements en entrée ainsi que toute version interpolée de ces derniers. On obtient en sortie des séquences statistiquement similaires aux entrées avec de légères différences dans les poses et leur timings.

Contents

List of Figures

Chapter 1

Introduction

Contents

1.1 Context

Realistic human motion and animation generation is motivated by the increasing demand from media production, namely the video game industry, the film industry and more recently virtual reality applications. Many researchers and industrials consider physical modelling for this task, but these solutions are computationally expensive, suffer from models' limitations, and come without effective guarantees on the realism of the synthesis. Animation through artistic Key-framing is unfortunately time-wise and money-wise expensive: Surveys estimated that a single character second costs between 100 and 250 \$, and that it takes an artist an entire day to complete roughly 2 to 3 seconds of finished character animation only. While manual animations are usually tedious and require artistic expertise, motion capture data allows to animate characters with real motion information extracted from real performances. In addition to significantly reduce the animation cost, the benefit of motion capture data also lies in the inherent realism of the recorded motions compared to synthetic data. Hence, a large body of work considers such input data to animate virtual characters.

After more than a decade of predominance of standard motion capture, we have witnessed in the recent years the emergence of 4D(3D+t) surface capture either through high quality multi-view setups Collet et al.

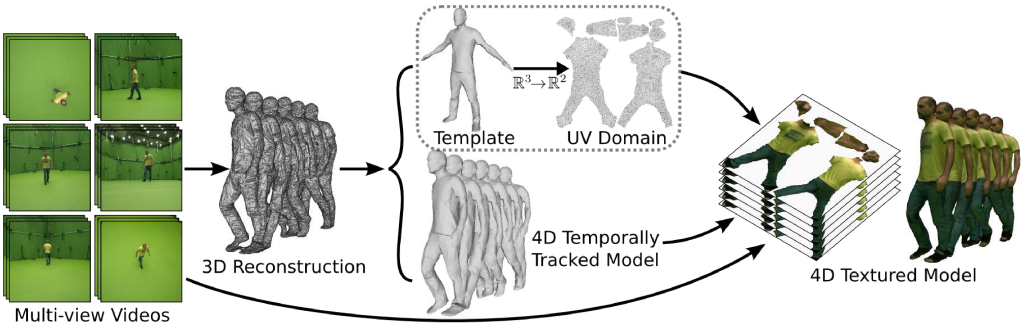


Figure 1.1 – Data acquisition pipeline.

[2015b], Dou et al. [2016], Lucas et al. or in a more democratized form with low-cost sensors Newcombe et al. [2015], Innmann et al. [2016], Wang et al. [2016]. Whilst traditional motion capture consists of sparse kinematic information, surface capture provides real full shape, appearance and motion information of the dynamic object of interest, in the form of temporally consistent mesh sequences with their appearance maps, and they are also referred to as 4D models.

Figure 1.1 illustrates the steps we use for creating 4D textured models. Our dynamic subject of interest is recorded from different view-points, the image silhouettes from these videos allow us to perform time-dependant surface reconstruction Franco and Boyer [2009]. Conformal mapping Lévy et al. [2002] is used to obtain a texture atlas for a template geometry from a set of mesh cuts. The same template is used to perform 3D tracking Allain et al. [2015] over the reconstructed sequence, thus rendering its geometry consistent in topology and connectivity information over time. The temporally tracked sequence geometries are then used to back-project the appearance information from the input views and into atlas-consistent texture maps Tsiminaki et al. [2014] thus allowing us to render textured 4D sequences.

The tasks related to the acquisition and exploitation of 4D data, such as tracking Allain et al. [2015], Mustafa et al. [2016], Budd et al. [2013], appearance modelling Boukhayma et al. [2016], Volino et al. [2014] and animation Prada et al. [2016], Casas et al. [2014a], Boukhayma and Boyer [2015] have received a great deal of interest from the vision and graphics communities recently, and there are still many problems to be solved in this respect. Notably, a number of challenges remain open when dealing with corpus of moving subjects. We mainly address in this thesis animation synthesis using 4D data in Chapters 2 and 3, building compact appearance representations for 4D models in Chapter 4, semantic motion

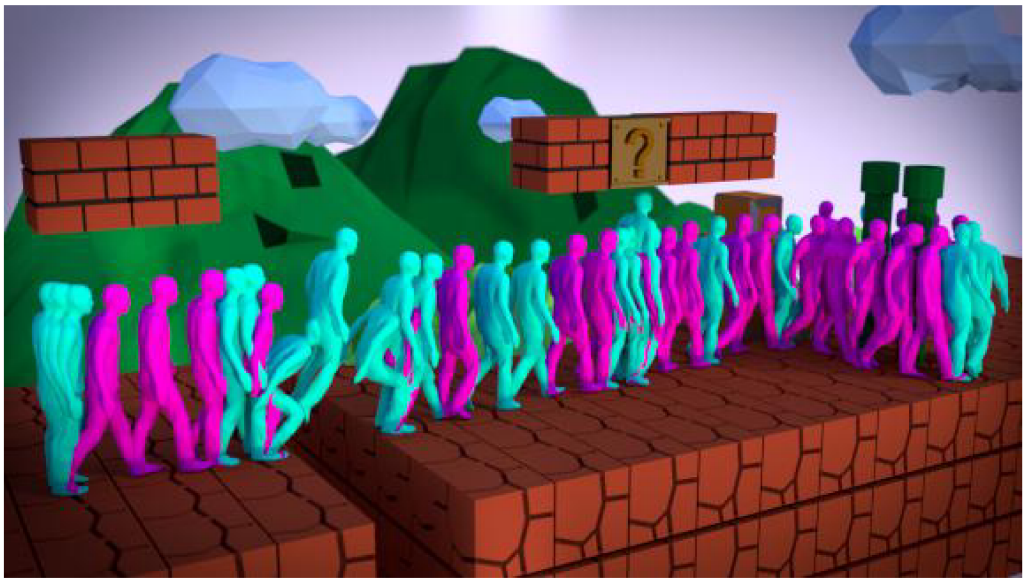


Figure 1.2 – 4D Animation from user constraints. Green frames are recorded and red frames are synthetic.

transfer between 4D models in chapter 5, and finally generating variations of 4D models in chapter 6.

1.2 Summary and Contributions

We first propose a solution for building animations using 4D models. We focus on the optimality of this synthesis with respect to a numerical realism criterion since our data is more intricate and sensitive to editing compared to, for instance, skeletal motion capture. We limit the synthesis to simple data reuse which consists of concatenating original motion segments and synthetic transitions between them as seen in Figure 1.2. In chapter 2, we contribute a method to synthesise visually plausible transition segments that improves local optimality of the synthesis. In chapter 3, we contribute a data structure to organize and control the animation process through user-guidance that improves global optimality of the synthesis.

Second, we propose to build a compact view-independent appearance representation and estimation algorithm, to encode the appearance variability of a dynamic subject in chapter 4. We contribute a new strategy that first coarsely registers geometry thus removing the main non-linearity of the problem, then copes with remaining fine-scale 3D misalignments

due to geometric modelling inaccuracies through 2D realignments in the texture domain.

Next, we propose to automatically transfer 4D captured performances in chapter 5, to the benefit of enabling the generation of uncaptured animations. Given two training sets of motions for two subjects and a sparse set of semantic annotations, we contribute an algorithm that builds on Gaussian Process Regression to transfer motions from one subject to the other without the need for subject shape correspondences.

Subsequently, we address the problem of generating variations of captured 4D models automatically in chapter 6. Given a few 4D models representing movements of the same type for a specific subject, we advocate the use of Gaussian Process Dynamical Models directly on mesh data to build a low dimensional space of motion and sample new motion variants from it. We also propose a piecewise linear parametrization scheme that allows synthesise of an unlimited number of variations of any of the input movements, and also any blended version of them. Finally, conclusions and future work are presented in Chapter 7.

Datasets

For qualitative and qualitative evaluations through out this work, we use the following datasets:

In chapters 2 and 3, we contribute a new dataset TOMAS and CATY Boukhayma and Boyer [2015] of temporally coherent mesh motion sequences, with 5000 vertices and 10000 faces per mesh, and a frame rate of 50fps. It contains 2633 frames of an actor performing basic locomotion activities, and 1910 frames of an actress performing random dancing. We also use datasets of temporally consistent mesh sequences DAN Casas et al. [2014b], JP Starck and Hilton [2007], Budd et al. [2013], CAPOEIRA Stoll et al. [2010] and ADOBE Vlasic et al. [2008a].

In chapter 4, we contribute another new dataset TOMAS and CATY Boukhayma et al. [2016] of temporally coherent mesh motion sequences, with 5000 vertices and 10000 faces per mesh and a frame rate of 50fps, with back-ground segmented images for each of the 68 capture views: 2048×2048 px, and texture maps: 4096×4096 px. It contains 207 frames of an actor walking, running, turning left and right, and 290 frames of an actress jumping far, close, high and low.

In chapter 5, we use datastes TOMAS and DAN, and in chapter 6 we use dataset DAN.

Publications

Controllable Variation Synthesis for Surface Motion Capture
3DV 2017 - International Conference on 3D Vision.
Adnane Boukhayma, Edmond Boyer.

Surface Motion Capture Transfer with Gaussian Process Regression
CVPR 2017 - Computer Vision and Pattern Recognition.
Adnane Boukhayma, Jean-Sbastien Franco, Edmond Boyer.

Eigen Appearance Maps of Dynamic Shapes
ECCV 2016 - European Conference on Computer Vision.
Adnane Boukhayma, Vagia Tsiminaki, Jean-Sbastien Franco, Edmond Boyer.

Video based Animation Synthesis with the Essential Graph
3DV 2015 - International Conference on 3D Vision.
Adnane Boukhayma, Edmond Boyer.

Surface Motion Capture Animation Synthesis (Submitted)
TVCG - Transactions on Visualization and Computer Graphics.
Adnane Boukhayma, Edmond Boyer.

Chapter 2

Shape pose and motion blending

Contents

2.1 Introduction

We propose in this work an example-based animation synthesis solution for textured mesh data. Given the input textured mesh sequences, we want to generate continuous motion streams combining these motion examples. Merely replaying the input sequences is hardly enough for this purpose, hence we need to generate transitions between them to insure

spatial and temporal continuity of the resulting animation. The output animation is thus a mesh sequence obtained by concatenating original mesh sequence segments and synthetic mesh sequence segments. The synthetic mesh segments provide smooth transitions between the original mesh segments both temporally and spatially. In this chapter, we explain how these motion transitions are generated, and introduce a quantitative measure of the realism of these synthetic transitions.

2.2 Related Work

We are given two mesh sequences with the same topology and connectivity information, a source mesh sequence and a target mesh sequence, and we want to generate a smooth transition from a given frame in the source sequence to another given frame in the target sequence. Note that the source and target sequences might be actually the same sequence.

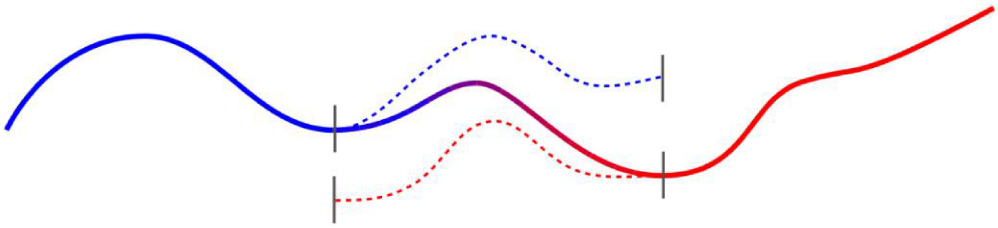
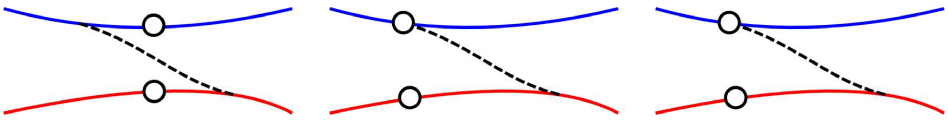


Figure 2.1 – Interpolated transition with gradual blending Rose et al. [1998]

One naive solution would be to simply concatenate the part of the source segment before the transition frame to the part of the destination segment after the transition frame as it was done in Huang et al. [2009]. This results inevitably in abrupt and discontinuous transitions. In fact, the authors in Huang et al. [2009] were forced to resort to this strategy because they lacked temporal coherency in their mesh sequence data then, i.e. unified mesh topology and connectivity information. In contrast under this later assumption, which is the case for us, mesh sequence data is akin to a homogeneous time series, and the editing, warping and blending techniques for smooth transitions that we present subsequently become applicable.

As it was proposed for Motion Capture data, source and destination segments can be edited so as to meet at a common transition frame using motion signal warping methods Witkin and Popovic [1995], but gradual



(a) Center-aligned (b) Left-aligned (c) Start-end

Figure 2.2 – Motion transition schemes.

blending remains a more straightforward solution that is widely used in the literature, either in a start-end Kovar et al. [2002a], center-aligned Arikan and Forsyth [2002] or left aligned Lee et al. [2002] Safonova and Hodgins [2007] transition form (figure 2.2). Gradual blending consists of gradually shifting from source motion to destination motion by interpolating them with a parameter that increases smoothly from 0 to 1, as shown in figure 2.1 Rose et al. [1998]. Traditionally for this type of transitions, the root displacements and joint orientations are interpolated, and the local skeleton coordinate frames need to be recurrently aligned in order to prevent the root trajectory from collapsing as it is done with registration curves Kovar and Gleicher [2003]. However, motion blending might generate unpleasant results if incorrectly applied especially with linear interpolation, and unless costly space-time constraints and inverse kinematics are used as in Rose et al. [1996]. Luckily, these limitations can be substantially reduced with motion dynamic time warping, which is usually constrained by contact information as it is done in Menardais et al. [2004]. Furthermore, the work of Wang and Bodenheimer [2008] thouroughly investigates and demonstrates the influence of transition duration, source segment length and destination segment length on the quality of interpolated transitions for Motion Capture.

Hence, we propose a novel solution for interpolated transitions that combines these previous methods and generalizes them for mesh data. But in order to properly introduce this dynamic transition method, we first need to define a static mesh pose distance, and a static mesh interpolation method.

2.3 Shape pose distance

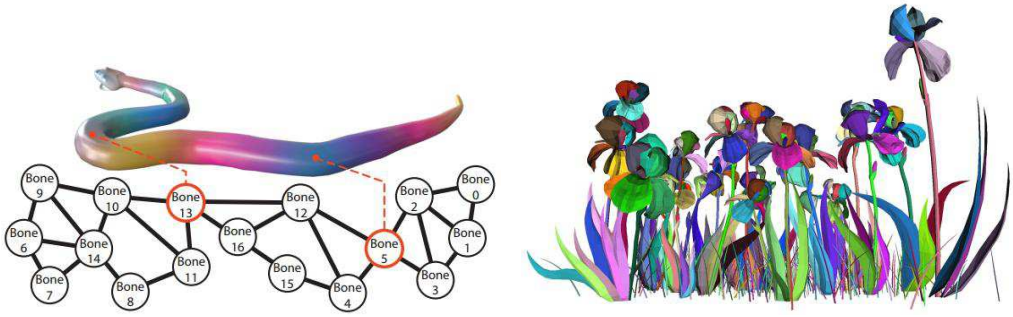
2.3.1 Skeleton-based representation

Standard Motion Capture data benefits from a well-defined pose parametrization. Each pose is entirely determined by the set of joint orientations of the skeleton hierarchy, which implies that poses, in the

form of stacked joint orientations, lie in the Cartesian product of special orthogonal groups of rotations $(R_1, ..., R_{n_{joints}}) \in SO(3)^{n_{joints}}$ where n_{joints} is the total number of joints in the skeleton hierarchy. Since the special orthogonal group is a Lie group, and Lie groups have a Riemannian manifold structure, poses lie within the product Riemannian manifold, and the corresponding geodesic metric between two poses i and j can be used as a pose similarity measure:

$$d_{skel}(i,j) = \sum_{n \in Joints} \frac{1}{\sqrt{2}} \| \log \left({R_n^i}^{-1} R_n^j \right) \|_F \tag{2.1}$$

Geodesic interpolation for each joint n can be used to interpolate poses accordingly: $R_n^i \exp(\lambda \log({R_n^i}^{-1} R_n^j))$ where λ is the interpolation parameter between poses i and j and varies between 0 and 1. This parametrization is used in the tasks of pose similarity measure and interpolation within many animation synthesis pipelines for Motion Capture data.



(a) Zheng [2013], the snake Briceño et al. [2003] is colormapped according to its linear blend skinning bone distribution Le and Deng [2012], beneath is its corresponding bone graph.

(b) James et al. [2007], the scene object is decomposed into colored groups arranged in a tree-structured kinematic hierarchy.

Figure 2.3 – Examples of mesh data animation with skeletal parametrization.

Only few works consider yet mesh sequences as input data for animation synthesis. Some of these works still chose to solve this problem in the skeleton domain. For instance (see figure 2.3), Zheng [2013] performs a skinning decomposition with rigid bones on the mesh data based on a linear blend skinning model Le and Deng [2012] prior to animation synthesis. The work of James et al. [2007] also uses a skeletal parametrization for input ensemble mesh data in the form of separate groups each arranged in tree-structured kinematic hierarchy forming the global object.

While methods exist that provide articulated motion information, e.g. Vlasic et al. [2008a] Gall et al. [2009], this strategy suffers anyway from two drawbacks: First the identification of such information is difficult to perform robustly with generic surfaces that do not always evolve according to the assumed articulated motion model, as with clothes for instance. Second, the use of an intermediate skeletal model makes it difficult to guarantee realism for transitions that finally concern surfaces around skeletons. Similar to this line of work Huang et al. [2009] Casas et al. [2012], we follow another strategy that avoids the intermediate motion model, removing therefore the issue of mesh/skeleton conversion, and that directly considers surfaces or meshes.

2.3.2 Mesh-based representation

We define a mesh-based static pose distance function between two given shape poses. This distance will be used further at various stages of our approach, for instance when estimating the cost of a dynamic transition that involves several intermediate poses. we assume in the following that mesh topology is consistent between source and target shape poses. When such shape property is not consistent, other solutions might be considered such as volumetric shape histograms Huang et al. [2007a] combined with exhaustive rotation search as in Huang et al. [2009], or rotation-invariant spherical harmonic descriptor of Kazhdan et al. [2003] as used in Prada et al. [2016] for shape pose similarity measurement.

Euclidean distance

One possible solution could be a variant of the metric defined in Kovar et al. [2002a], only it was genuinely used in the latter work to align point clouds driven by the skeleton formed over a window of frames, while we use it here to align pairs of 3D meshes.

Given two poses i and j, we first align them by finding the 2D rigid transformation T_{θ,x_0,z_0} that minimizes the weighted sum of squared distances between their respective mesh vertices. This rigid transformation is composed of a translation of (x_0, z_0) in the horizontal plane (O, x, z) (*i.e.* the plane parallel to the motion) and of a rotation of θ around the vertical axis (O, y). Weights are used to, for instance, give more importance to torso vertices with humans. Once aligned, one could take the residual of the alignment cost $d_{i,j}$ as the distance between the poses:

$$d_{Eucl}(i,j) = \min_{\theta,x_0,z_0} \sum_k w_k \|v_k^i - T_{\theta,x_0,z_0} v_k^j\|^2 \tag{2.2}$$

where index k is over the number of mesh vertices and $v_k = (x_k, z_k)^T$. In practice, this optimization has the following closed form solution:

$$\theta = \tan^{-1}\left(\frac{\sum_k w_k(x_k^i z_k^j - x_k^j z_k^i - \bar{x}^i \bar{z}^j + \bar{x}^j \bar{z}^i)}{\sum_k w_k(x_k^i x_k^j + z_k^i z_k^j - \bar{x}^i \bar{x}^j - \bar{z}^i \bar{z}^j)}\right) \tag{2.3}$$

$$x_0 = \bar{x}^i - \bar{x}^j \cos(\theta) - \bar{z}^j \sin(\theta) \tag{2.4}$$

$$z_0 = \bar{z}^i + \bar{x}^j \sin(\theta) - \bar{z}^j \cos(\theta) \tag{2.5}$$

where $\bar{x} = \frac{1}{\sum_k w_k} \sum_k w_k x_k$ and $\bar{z} = \frac{1}{\sum_k w_k} \sum_k w_k z_k$.

However, this residual does not account for the mesh geometry Sorkine et al. [2004]. We hence use a parametrization baed on deformation gradients instead Sumner and Popović [2004a], Yu et al. [2004a].

Non-Euclidean distance

For every pair of corresponding triangles $(v_1^i,\ v_2^i,\ v_3^i)$ and $(v_1^j,\ v_2^j,\ v_3^j)$ in pair of meshes (i,j), we compute the following unique transform matrix (see figure 2.8):

$$T = (v_1^i - v_3^i,\ v_2^i - v_3^i,\ n^i)^{-1} . (v_1^j - v_3^j,\ v_2^j - v_3^j,\ n^j) \tag{2.6}$$

where n^i and n^j are the triangle unit normals, and $v_k = (x_k, y_k, z_k)^T$. The collection of these affine transformation matrices for all mesh triangles is referred to as mesh deformation gradients, and it was first introduced in the work of Barr [1984] under the name of local deformations.

Among affine transformations, i.e. rotation, shearing and scaling, rotations are distorted by linear matrix interpolation. And thus extracting them properly is crucial for deformation parametrization. Rotations have the property of special orthogonality, which translates into $R^T R = I$, and $det(R) = +1$ to avoid reflections. For square non-singular matrices, three orthogonal decomposition algorithms have been proposed: Singular Value decomposition, QR decomposition and Polar decomposition.

SVD decomposition $T = U\Sigma V^T$ gives three factors, where U and V are orthogonal and Σ is diagonal and positive. But the orthogonal components produced by this decomposition are not unique. As a matter of fact, there is an infinite number of possible decomposition components

even for a pure rotation matrix, and the smallest perturbation in the input can result in different orthogonal component for this factorization regardless of the stability of the singular values.

QR decomposition $T = QR$ yields an orthogonal component Q and lower triangular one R. These components can be determined uniquely and are stable under small perturbations. However, the resulting orthogonal matrix is not independent of the coordinate basis. As a matter if fact, given matrix T in a different basis $T' = BTB^{-1}$, the basis independent decomposition would be $Q' = BQB^{-1}$ and $R' = BRB^{-1}$. But this latter is not lower triangular, because matrix similarity doesn't preserve this property.

Polar decomposition $T = RS$ gives an orthogonal component R and a symmetric positive definite component S. These components are unique and independent of the coordinate basis. This decomposition can be performed iteratively by initializing the orthogonal component with the transformation matrix and averaging it by its inverse transpose until it ceases to change:

$$R_0 = T,\ R_{i+1} = \frac{1}{2}\left(R_i + R_i^T\right) \tag{2.7}$$

Another way to perform polar decomposition is through Eigen-decomposition of T^TT. Thas is, if the polar decomposition of T is writen in the following from:

$$T = RS, \tag{2.8}$$

then, since R is orthogonal

$$T^TT = S^TS, \tag{2.9}$$

hence by performing SVD decomposition on symmetric matrix T^TT we obtain the following expression:

$$S^TS = U\Sigma U^T \tag{2.10}$$

where U is orthonormal and Σ is diagonal and its diagonal entries are positive. We can then finally obtain an expression of S and R from the SVD decomposition of T^TT as follows:

$$S = U\Sigma^{\frac{1}{2}}U^T \tag{2.11}$$

$$R = TS^{-1} \tag{2.12}$$

The resulting orthogonal component R can then be divided by $det(R)$ to insure that it belongs the special orthogonal group of rotations $SO(3)$, and the symmetric component S can be updated accordingly. Rigid triangle based rigid transformation T is finally uniquely decomposed into a rotation component R and shearing and anisotropic scaling component S.

Combining the Riemannian metric of the Lie group of rotations $SO(3)$ and Frobenius norm for symmetric 3×3 matrices, we define our pose similarity metric as the combination of the amounts of local deformations occurring at each triangle of the mesh to deform pose i into j or vice versa:

$$d(i,j) = \sum_{m \in Triangles} \frac{1}{\sqrt{2}} \| \log(R_m) \|_F + w_m \| S_m - I_3 \|_F \tag{2.13}$$

Through weighting factors w_m, we give more importance to the rotation term. In practice, we use the same value $w_m = 0.01$ for all mesh triangles.

Noticing that the shearing/scaling component S of the polar decomposition $T = RS$ is positive semi-definite, and that this type of matrices forms a Riemannian manifold as-well, we can use its Riemannian metric and create a variant of the pose distance defined in equation 2.13:

$$d'(i,j) = \sum_{m \in Triangles} \frac{1}{\sqrt{2}} \left(\| \log(R_m) \|_F + w_m \| \log(S_m) \|_F \right) \tag{2.14}$$

Practically, both metrics behaved similarly and we couldn't notice any significant difference between them as far as their influence in building our animation pipeline goes. It is also important to note that this pose representation is not rotation invariant. Hence, we need to align meshes globally using equation 2.3 before proceeding to pose distance computations.

Evaluation

We attempt to statistically evaluate our mesh-based pose metric using *Adobe* dataset Vlasic et al. [2008b]. This dataset (see figure 2.4) contains various motions performed by 3 different actors in the form of mesh sequence data. Each actor's set of mesh sequences are temporally consistent in topology. The 3D models represent detailed meshes of loosely dressed humans which makes this set of data relevant to our context. These sequences are additionally joined with ground truth pose information in the form of a template skeleton joint Euler angle values for each frame.

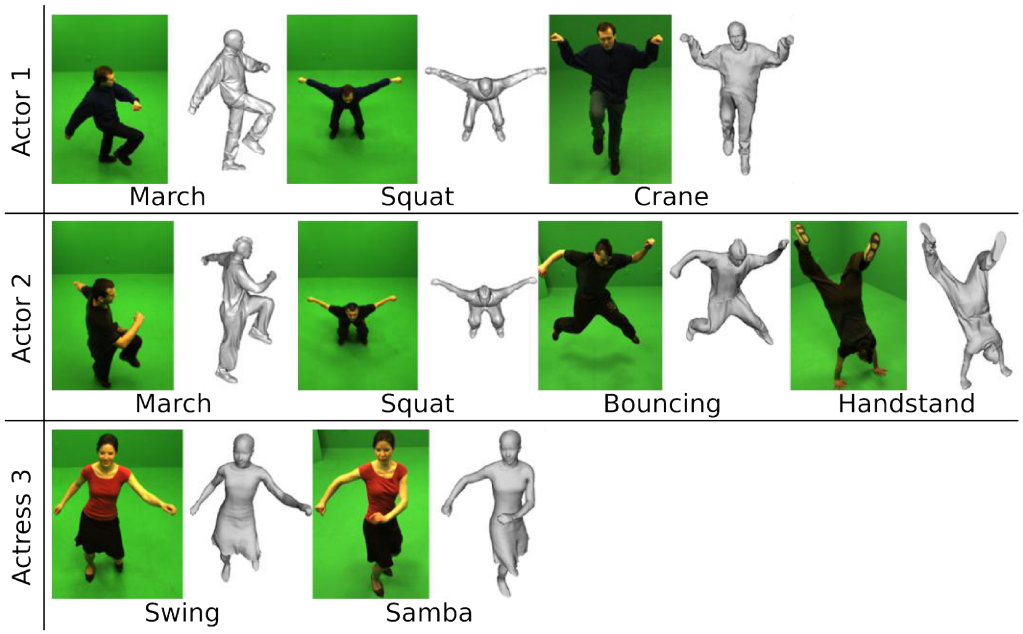


Figure 2.4 – *Adobe* Dataset

We consider three types of pose distances. The first one is derived from the Euclidean distance between Euler Angles Huynh [2009]. It is computed using the skeleton pose information available with the dataset, and will hence be considered as the ground truth distance in our following experiments. For a pair of frames i and j, represented each by Euler angles of their joints $\{\alpha_n, \beta_n, \gamma_n\}_{n \in Joints}$, we define this pose distance as follows:

$$d_{Euler}(i,j) = \sqrt{\mathtt{d}\left(\alpha_n^i, \alpha_n^j\right)^2 + \mathtt{d}\left(\beta_n^i, \beta_n^j\right)^2 + \mathtt{d}\left(\gamma_n^i, \gamma_n^j\right)^2} \tag{2.15}$$

where $\mathtt{d}(.,.)$ is the normalized difference between two angles such that $0 \leq \mathtt{d}(.,.) \leq \pi$:

$$\mathtt{d}(\alpha^i, \alpha^j) = \min\left(2\pi - |\alpha^i - \alpha^j|, |\alpha^i - \alpha^j|\right) \tag{2.16}$$

The second pose distance is our mesh-based pose metric defined in equation 2.13. Finally the third pose distance is the naive Euclidean one defined in equation 2.2 and which also uses the mesh data.

For each of the three distance definitions, we compute the pose distance values for every pair of frames in the dataset and normalize these values. We then consider each of these three sets of distance values as statistical samples from three separate random variables denoted as follows:

$$d_{Euler} := \{d_{Euleri}\}, \quad d := \{d_i\}, \quad d_{Eucl} := \{d_{Eucli}\} \tag{2.17}$$

Where index i spans over the number of all possible pairs of frames, which is $P = N(N-1)/2$ for a dataset of N frames.

In order to statistically validate our pose distance function, we measure the correlation of its random variable with the ground truth pose distance variable first using Pearson correlation coefficient defined as follows:

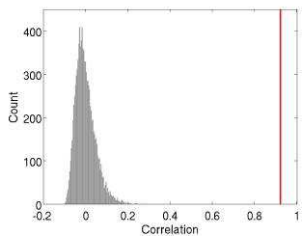
$$\nabla r(d, d') = \frac{\sum_i (d_i - \bar{d})(d'_i - \bar{d'})}{\sqrt{\sum_i (d_i - \bar{d})^2} \sqrt{\sum_i (d'_i - \bar{d'})^2}} \tag{2.18}$$

Where $\bar{d} = \frac{1}{P} \sum_i d_i$ and P is the total number of samples. This coefficient measure the linear correlation between two variables, and gives a values between 1 and -1, where 1 is total positive correlation, -1 is total negative correlation and 0 is no correlation at all.

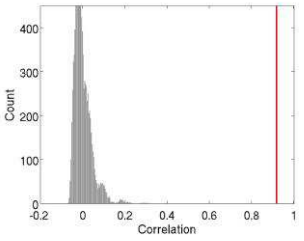
However, the significance of this coefficient alone can be limited as far as assessing the relationship between two variables is concerned. To deal with this limitation, we perform a variant of a statistical test called the Mantel test Mantel [1967]. It consists of randomly permuting samples within one of the distance variable sets, d' for instance, and computing new correlation values between the permuted distance d' and the reference distance d. The p-value of this test is the count of cases where the permuted distance d' is more correlated to the the reference distance d than its original non permuted version, devided by the total number of permutations. This p-value is derived from the reasoning that in case of absence of correlation between d' and d, permuting samples in d' should produce more or less correlated variables to d equally likely.

Figure 2.5 shows results of performing this statistical test with 10000 permutations to evaluate the correlation between our distance d and the ground truth d_{Euler} in the first row, and between the naive Euclidean distance d_{Eucl} and the ground truth d_{Euler} in the second row. The first, second and third columns show results using data from Actors 1, 2 and 3. The red line show the correlation value between our distance and the ground truth, and the the blue line shows the correlation between the Euclidean distance and the ground truth. The grey histograms represent the distribution of the test statistic, namely the correlation between the permuted distances and the ground-truth distance.

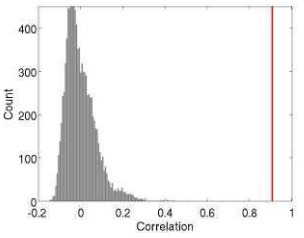
We can first easily see that the correlation value between our distance and the ground truth (red line) is always bigger than the correlation between the Euclidean distance and the ground truth (blue line). The



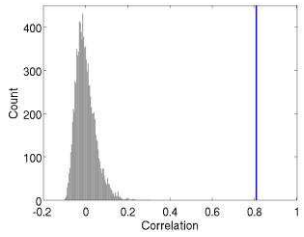
(a) Correlation between the ground truth and our pose distance, Actor 1



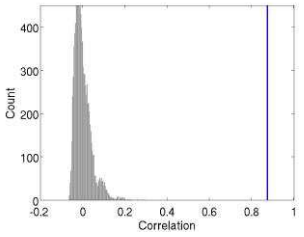
(b) Correlation between the ground truth and our pose distance, Actor 2



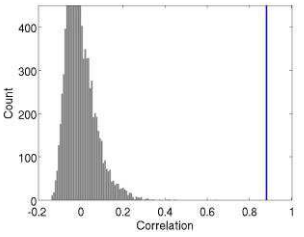
(c) Correlation between the ground truth and our pose distance, Actor 3



(d) Correlation between the ground truth and the Euclidean pose distance, Actor 1



(e) Correlation between the ground truth and the Euclidean pose distance, Actor 2

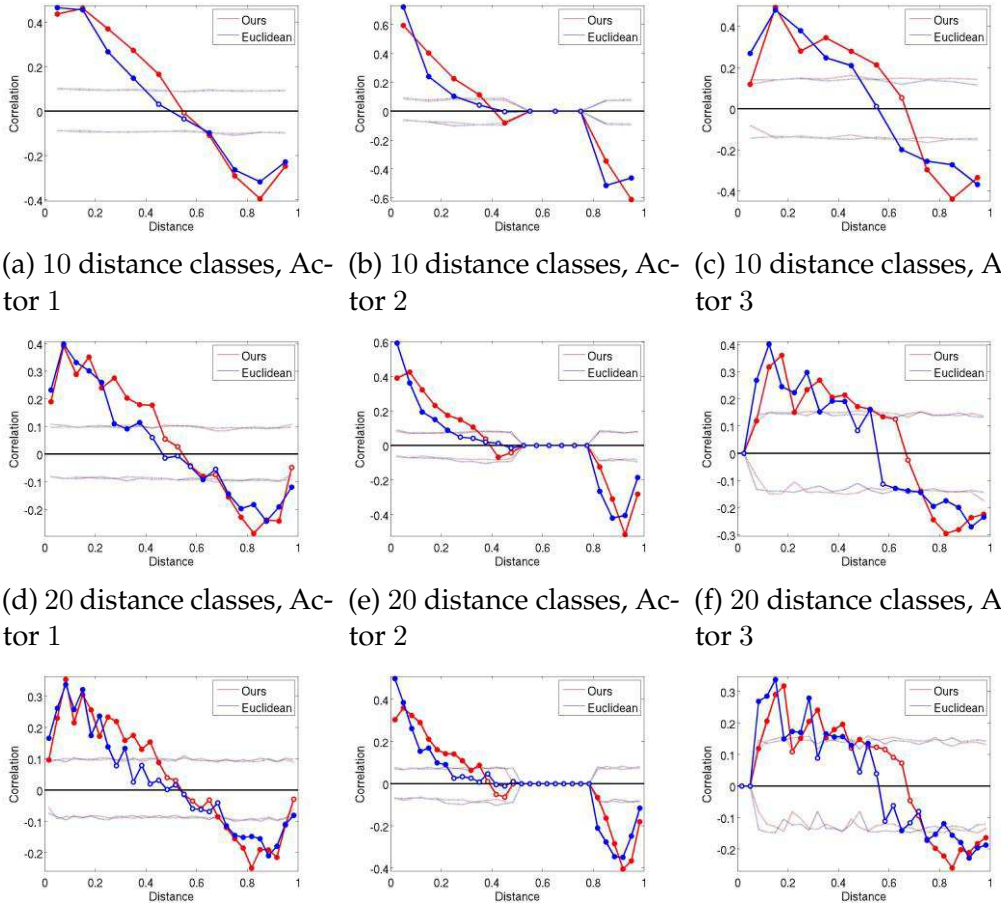


(f) Correlation between the ground truth and the Euclidean pose distance, Actor 3

Figure 2.5 – Mantel test: Distributions of the Pearson correlation statistic of permutation tests comparing the ground truth to ours and the Euclidean pose distance. Horizontal lines show the original value of correlation, and the histograms show the count of correlation values after permutations.

original correlation coefficient being always greater and clearly further from the permutation histograms, we can also easily confirm the test hypothesis of correlation between our distance and the ground truth, and also between the Euclidean distance and the ground truth. For both distances, ours and the Euclidean, no permuted version of the distance achieves a better correlation to the ground truth, which gives us 0 p-values for both. That is a bit unfortunate us we could have used the p-values as an additional measure to compare the distances' correlation to the ground truth if they were not both null.

One of the drawbacks of this statistical test is that the dependence measure between the distance variables is averaged over all distances, and so the test cannot discover changes in the pattern of correlation at different scales of distances. The Mantel correlogram helps to overcome this limitation by allowing the visualization and the assessment of the



(a) 10 distance classes, Actor 1 (b) 10 distance classes, Actor 2 (c) 10 distance classes, Actor 3

(d) 20 distance classes, Actor 1 (e) 20 distance classes, Actor 2 (f) 20 distance classes, Actor 3

(g) 30 distance classes, Actor 1 (h) 30 distance classes, Actor 2 (i) 30 distance classes, Actor 3

Figure 2.6 – Mantel Correlogram: Pearson correlation coefficient of classes of range of distances, between the ground truth and our pose distance (Red), and the ground truth and the Euclidean distance (Blue).

spatial dependency of distances more locally. The distance value sets are separated into classes of different distance ranges, and a Mantel test is performed on each of these classes separately. Figure 2.6 shows the comparison of these local correlation measures between the ground truth and our pose distance (Red) and the ground truth and the Euclidean pose distance (Blue). We also plot the lower and upper confidence limits bounding the 95% confidence interval about the null correlation hypothesis for both comparisons in the same figure. Reiterating the experiment for different total numbers of distance classes allows the assessment of the correla-

tion at different distance scales. We can see that the local behaviour of correlation scores is similar to the results that we obtained with global correlation, as shown in figure 2.5 and 2.7, and which favours the correlation of our distance to the ground-truth, compared to the Euclidean distance. We can also notice on the plot that both metrics perform well for small distance values, where both are highly correlated to the ground-truth as expected. This means that for relatively small mesh deformations, even the Euclidean distance can be a reliable estimate of shape pose distance. Unfortunately for higher distance values, both distances show low and even negative correlations to the ground truth in some cases.

An other limitation of the Pearson coefficient is that it only accounts for linear correlation between variables. We hence introduce distance correlation Székely et al. [2007] as an additional measure of statistical correlation that takes into account non-linear dependencies. For instance, if we consider the vector $x = [\![-10, 10]\!]^T$ and $y = x \circ x$ where $\circ$ is the Hadamart product, also known as the element wise product, i.e. $y_i = {x_i}^2$, the correlation between x and y in terms of Pearson coefficient approximates 0, while the distance correlation is nearly 0.5. It was proven that while Pearson correlation coefficient can approach zero easily for dependant random variables, distance correlation equals zero if and only if the variables are statistically independent.

To define distance correlation, we need to define distance covariance first. For two variables d and d' with P samples each, we compute the $P \times P$ distance matrices $(a_{i,j})$ et $(b_{i,j})$ containing pairwise Euclidean distances:

$$a_{i,j} = |d_i - d_j|, \quad b_{i,j} = |d'_i - d'_j|, \quad 1 \leq i, j \leq P \tag{2.19}$$

We then construct the $P \times P$ matrices of doubly centered distances $(A_{i,j})$ and $(B_{i,j})$ defined as follows:

$$A_{i,j} = a_{i,j} - \bar{a}_{i.} - \bar{a}_{.j} + \bar{a}_{..}, \quad B_{i,j} = b_{i,j} - \bar{b}_{i.} - \bar{b}_{.j} + \bar{b}_{..} \tag{2.20}$$

where $\bar{a}_{i.}$ is the i-th row mean of $(a_{i,j})$, $\bar{a}_{.j}$ is the j-th colomn mean of $(a_{i,j})$, and $\bar{a}_{..}$ is the global mean of matrix $(a_{i,j})$. The squared distance covariance is defined as the global mean of the Hadamart product of matrices $(A_{i,j})$ and $(B_{i,j})$:

$$\mathcal{V}^2(d, d') = \frac{1}{P^2} \sum_{i=1}^{P} \sum_{j=1}^{P} A_{i,j} B_{i,j} \tag{2.21}$$

Finally, the distance correlation between two variables d and d' is obtained by dividing their distance covariance by the product of their distance variances:

$$\mathcal{R}(d, d') = \frac{\mathcal{V}(d, d')}{\sqrt{\mathcal{V}(d, d)\mathcal{V}(d', d')}} \tag{2.22}$$

	Actor 1		Actor 2		Actress 3	
	Eucl.	Ours	Eucl.	Ours	Eucl.	Ours
Pearson Corr.	0.8078	0.9222	0.8749	0.9189	0.8804	0.9075
Distance Corr.	0.8031	0.9222	0.8686	0.8977	0.8582	0.8989

Figure 2.7 – Correlation between the ground truth and both our pose distance and the Euclidean pose distance.

Figure 2.7 shows the comparison results of the correlation between the ground truth d_{Euler} and both our pose distance d and the Euclidean pose distance d_{Eucl}. Overall, our pose distance is more correlated to the ground truth both in terms of Pearson correlation defined in equation 2.18 and distance correlation defined in equation 2.22.

2.4 Shape pose interpolation

2.4.1 Linear interpolation

The most straightforward way of achieving shape pose interpolation is to consider a purely Euclidean representation of meshes in $\mathbb{R}^{3\times N}$ where N is the total number of mesh vertices, and applying linear interpolation on vertex coordinates. After rigidly aligning pose mesh source i and pose mesh target j using equation 2.3, each vertex $\tilde{v}$ in the interpolated pose mesh is obtained as the following linear combination of the corresponding vertices v^i and v^j in source and target poses respectively:

$$\tilde{v}_k = (1 - \lambda)v_k^i + \lambda v_k^j \tag{2.23}$$

Where index k spans over the number of mesh vertices, and $\lambda \in [0, 1]$ is the interpolation parameter.

As shown in the example provided in figure 2.8, it is well known that linear mesh interpolation introduces artefacts such as surface distortion

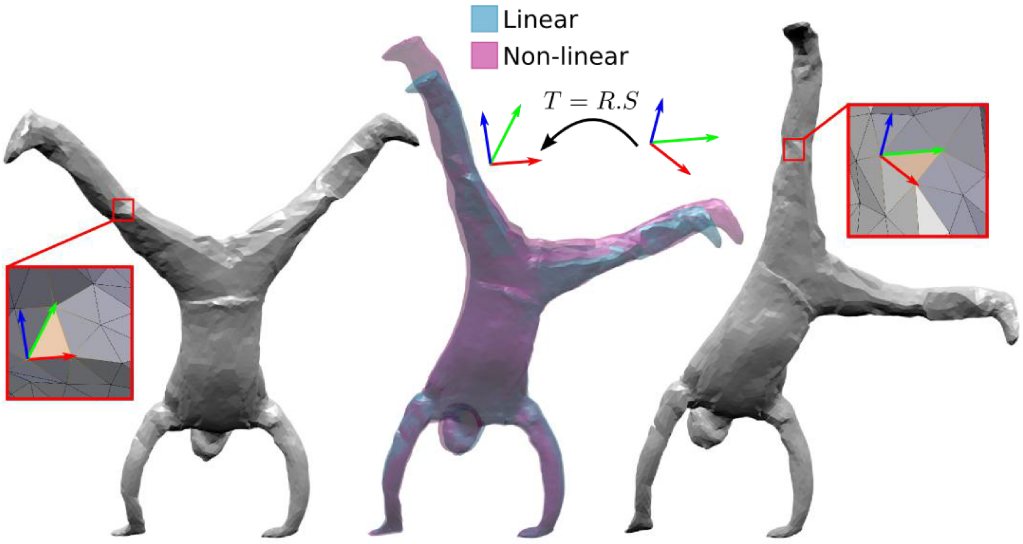


Figure 2.8 – Linear versus Non-linear mesh interpolation.

and shrinkage. To reduce these effects, we use a variant of Poisson shape interpolation Xu et al. [2005]. Our experiments demonstrated that this strategy performs better compared to linear vertex interpolation or as rigid as possible approaches Sorkine and Alexa [2007].

2.4.2 Differential operators for meshes

Let us recall that we consider a mesh with N vertices $\{v_1, ..., v_N\}$ and M triangles $\{t_1, ..., t_M\}$. The mesh can be considered as a vector field defined on a domain with a mesh structure that we refer to as the domain mesh. We express this vector field as a piecewise linear coordinate function $v(.)$ defined by barycentric interpolation of vertex coordinates $v_k = [x_k,\ y_k,\ z_k]^T$:

$$v(x) = \sum_{k=1}^{N} \phi_k(x) \cdot v_k \tag{2.24}$$

where parameter x is over the domain mesh and $\phi_k(.)$ is the piecewise linear hat function associated with the domain mesh vertices such that $\phi_i(v_k) = \delta_{i,k}$. This expression allows us in the following to formulate the application of differential operators on the surface mesh. The works of Tong et al. [2003] and Polthier and Preuß [2003] proposed well-defined discrete differential operators of scalar and vector fields on such irregular domains.

Gradient

We start first with the gradient of the mesh function $v(.)$ that writes:

$$\nabla v(x) = \sum_{k=1}^{N} \nabla \phi_k(x) v_k^T \tag{2.25}$$

Since function $v(.)$ is triangle-wise linear in the case of a triangular mesh, its gradient is intuitively constant within each mesh triangle. Thus for a triangle m in the mesh, we note its 3×3 mesh gradient matrix G_m such that

$$G_m = \nabla v|_m = \sum_{k=1}^{N} \nabla \phi_k|_m v_k^T \tag{2.26}$$

In practice, the gradient of the hat basis function associated to a vertex v_k is null when evaluated outside of the triangles incident to this vertex. Hence, only vertices forming triangle m contribute to the summation in equation 2.26 above, and if we note v_1, v_2 and v_3 the mesh vertices making up triangle m, $t_m = (v_1,\ v_2,\ v_3)$, equation 2.26 simplifies

$$G_m = (\nabla \phi_1|_m,\ \nabla \phi_2|_m,\ \nabla \phi_3|_m) \begin{pmatrix} v_1^T \\ v_2^T \\ v_3^T \end{pmatrix} \tag{2.27}$$

Next, we need to compute basis function gradients $\nabla \phi_k$. In triangle $t_m = (v_1,\ v_2,\ v_3)$, the gradient $\nabla \phi_1|_m$ is perpendicular to the opposite edge $(v_2,\ v_3)$, which is expressed as a null dot product:

$$\nabla \phi_1|_m{}^T (v_2 - v_3) = 0 \tag{2.28}$$

Let n be the unit vector normal to triangle m:

$$n = \frac{(v_1 - v_3) \times (v_2 - v_3)}{\| (v_1 - v_3) \times (v_2 - v_3) \|} \tag{2.29}$$

Gradient $\nabla \phi_1|_m$ also lies within triangle m, and is thus perpendicular to n, which writes:

$$\nabla \phi_1|_m{}^T n = 0 \tag{2.30}$$

Furthermore, the length of gradient $\nabla \phi_1|_m$ must be equal to the inverse of h, the distance between vertex v_1 and the opposite edge $(v_2,\ v_3)$ (see figure 2.9). Hence we can write:

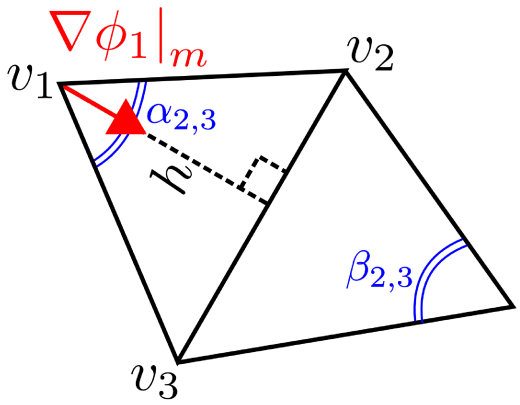


Figure 2.9 – Gradient of basis hat function in a mesh triangle

$$\begin{aligned}
\nabla\phi_1|_m{}^T (v_1 - v_2) &= \|\nabla\phi_1|_m\| \| (v_1 - v_2) \| \cos\left(\angle\left(\nabla\phi_1|_m, (v_1 - v_2)\right)\right) \\
&= \frac{1}{h} \cdot \| (v_1 - v_2) \| \cdot \frac{h}{\| (v_1 - v_2) \|} \\
&= 1
\end{aligned} \tag{2.31}$$

Identically we find $\nabla\phi_1|_m{}^T (v_1 - v_3) = 1$, and the three constraints on gradient $\nabla\phi_1|_m$ expressed in equations 2.28, 2.30 and 2.31, and generalized analogously for $\nabla\phi_2|_m$ and $\nabla\phi_3|_m$ lead to the following system:

$$\begin{pmatrix} (v_1 - v_3)^T \\ (v_2 - v_3)^T \\ n^T \end{pmatrix} (\nabla\phi_1|_m,\ \nabla\phi_2|_m,\ \nabla\phi_3|_m) = \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.32}$$

which implies:

$$(\nabla\phi_1|_m,\ \nabla\phi_2|_m,\ \nabla\phi_3|_m) = \begin{pmatrix} (v_1 - v_3)^T \\ (v_2 - v_3)^T \\ n^T \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.33}$$

By replacing the basis hat function gradients in equation 2.27 with their expression in equation 2.33, we obtain the final expression of mesh function gradient in triangle m:

$$G_m = \begin{pmatrix} (v_1 - v_3)^T \\ (v_2 - v_3)^T \\ n^T \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix} \begin{pmatrix} v_1^T \\ v_2^T \\ v_3^T \end{pmatrix} \tag{2.34}$$

A global gradient operator G can be yielded in the form of a $3M \times N$ matrix that multiplies the stacked transposed mesh vertex coordinates in a $N \times 3$ matrix, to give a $3M \times 3$ matrix of stacked triangle gradients:

$$\begin{pmatrix} G_1 \\ \vdots \\ G_M \end{pmatrix} = G \begin{pmatrix} v_1^T \\ \vdots \\ v_N^T \end{pmatrix}. \tag{2.35}$$

More explicitly, if we note $G = (g_{i,j})_{1 \leq i \leq 3M, 1 \leq j \leq N}$ then coefficients $g_{i,j}$ are defined as follows: let $m = \lceil \frac{i}{3} \rceil$ where $\lceil . \rceil$ symbolizes the ceiling function, $l = i - 3 \times (m-1)$, and triangle m be made of vertices v_{k_1}, v_{k_2} and v_{k_3} in this order: $t_m = (v_{k_1},\ v_{k_2},\ v_{k_3})$. If we note $A_m = (a_{q,r})_{1 \leq q,r \leq 3}$ the 3×3 matrix such that:

$$A_m = \begin{pmatrix} (v_{k_1} - v_{k_3})^T \\ (v_{k_2} - v_{k_3})^T \\ n^T \end{pmatrix}^{-1} \begin{pmatrix} 1 & 0 & -1 \\ 0 & 1 & -1 \\ 0 & 0 & 0 \end{pmatrix} \tag{2.36}$$

then:

$$g_{i,j} = \delta_{j,k_1} a_{l,1} + \delta_{j,k_2} a_{l,2} + \delta_{j,k_3} a_{l,3} \tag{2.37}$$

Figure 2.10 illustrates the relationship between gradient operator G and local triangle based gradients G_m. In simpler terms, in the m^{th} block of three lines in matrix G, all coefficients equal zero, except for the three 3-sized columns with indices k_1, k_2 and k_3. In fact, these three columns contain the values of the first, second and third columns of matrix A_m in this order, shown in figure 2.10 in red, green and blue.

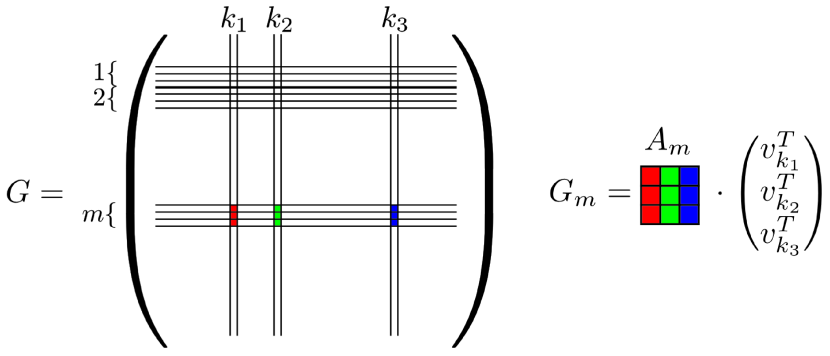


Figure 2.10 – Link between global mesh gradient operator G and mesh function gradient G_m at triangle $t_m = (v_{k_1},\ v_{k_2},\ v_{k_3})$

Laplacian

Once we computed the gradient of the mesh function ∇v, the divergence $\vec{\nabla}\cdot$ of this gradient gives the Laplacian Tong et al. [2003] of the mesh function Δv expressed at each vertex v_k as follows:

$$\begin{aligned}\Delta v(v_k) &= \left(\vec{\nabla}\cdot\nabla v\right)(v_k) \\ &= \sum_{m\in\mathcal{T}_k} area(t_m)\nabla\phi_k|_m{}^T G_m\end{aligned} \tag{2.38}$$

where $\mathcal{T}_k$ is the set of indices of triangles incident to vertex v_k in the mesh, and $area(t_m)$ is the area of triangle $t_m = (v_{k_1},\ v_{k_2},\ v_{k_3})$ and is equal to half the norm of the cross product of two vectors making up the triangle and originating from the same vertex:

$$area(t_m) = \frac{1}{2}\| (v_{k_1} - v_{k_3}) \times (v_{k_1} - v_{k_3}) \| \tag{2.39}$$

Note that the formulation in equation 2.38 is also akin to the standard discrete Laplace-Beltrami operator for triangular meshes Meyer et al. [2002]:

$$\Delta v_k = \sum_{i\in\mathcal{N}_k} \frac{1}{2}\left(cot(\alpha_{k,i}) + cot(\beta_{k,i})\right)(v_k - v_i)^T \tag{2.40}$$

where $\mathcal{N}_k$ is the set of indices of vertex v_k 1-ring neighbours, and $\alpha_{k,i}$ and $\beta_{k,i}$ are the angles opposite to edge $(v_k,\ v_i)$ (see figure 2.9). The result of equation 2.38 can also be formulated in a matrix product form. For this purpose we need to define D as a diagonal weighting mass matrix of size $3M \times 3M$ containing the mesh triangle areas in this form:

$$D = diag\begin{pmatrix} area(t_m) & 0 & 0 \\ 0 & area(t_m) & 0 \\ 0 & 0 & area(t_m) \end{pmatrix}_{1\le m\le M} \tag{2.41}$$

Then a global Laplace operator $L = G^T DG$ can be yielded in the form of a $N \times N$ matrix that multiplies the stacked transposed mesh vertex coordinates in a $N \times 3$ matrix, to give a $N \times 3$ matrix of stacked Laplacian coordinates:

$$\begin{pmatrix} \Delta v_1 \\ \vdots \\ \Delta v_N \end{pmatrix} = \underbrace{G^T D G}_{\Delta} \begin{pmatrix} v_1^T \\ \vdots \\ v_N^T \end{pmatrix} = \underbrace{G^T D}_{\vec{\nabla}\cdot} \begin{pmatrix} G_1 \\ \vdots \\ G_M \end{pmatrix} \tag{2.42}$$

In this equation, $G^T D$ is the matrix formulation of the divergence operator introduced in equation 2.38 and $G^T D G$ is the divergence of the gradient, i.e. the discrete Laplace operator. From this equation, we see that, in matrix formulation, we are able to express the mesh Laplacian in two ways: Either by applying the discrete Laplace operator $G^T D G$ directly on the mesh vertices, i.e. the coordinates vector field, or by applying the discrete divergence operator $G^T D$ to the mesh gradients. This equality evokes in its formulation the second order differential Poisson equation.

The Poisson equation seeks the reconstruction of an unknown vector or scalar function f whose gradient matches a vector field w. It is coupled with a condition on f, in the form of desirable value f^* over a certain boundary $\delta\Omega$, referred to as Dirichlet boundary condition, and it is expressed as follows:

$$\Delta f = \vec{\nabla} \cdot w, \qquad f|_{\delta\Omega} = f^*|_{\delta\Omega} \tag{2.43}$$

This equation is equivalent to the following least-square minimization from the viewpoint of a variational method:

$$\underset{f}{\operatorname{argmin}} \int_{\Omega} (\nabla f - w)^2 \, d\Omega \tag{2.44}$$

Our non-linear mesh interpolation technique uses the concept of 3D gradient-based mesh editing which is inspired from 2D gradient-based image editing Yu et al. [2004a]. It basically consists of manipulating the mesh gradient field and deriving the surface matching the deformed gradient by solving a linear Poisson system.

2.4.3 Non-linear interpolation

We are given a source pose i and a target pose j that we want to interpolate. For each triangle m in the mesh, we start by extracting the deformation gradient T_m that transforms the local coordinate frame associated to triangle m and its unit normal in mesh i to the same coordinate frame at that triangle in mesh j using equation 2.6. We then decompose the local transformation into a rotation component R_m and scaling/shearing component S_m using polar decomposition as it is explained previously in

section 2.3: $T_m = R_m S_m$. Let us recall that in that very section, deformation gradients are used to estimate pose distances. Then for an interpolation parameter λ, such that $\lambda \in [0, 1]$, we find the interpolated deformation gradient $\tilde{T}$ that equals identity matrix I_3 when λ is equal to 0, and reaches the total transformation $R_m S_m$ when λ is 1. This non-linear interpolation scheme combines the interpolation of the rotation component and the scaling/shearing component. The rotation component is geodesically interpolated between identity I_3 and rotation matrix R_m in the special orthogonal group $SO(3)$. As for the interpolation of the scaling/shearing component, it can be performed linearly in the Euclidean space of real matrices within the set of symmetric matrices giving thus the following interpolated transformation:

$$\tilde{T}(\lambda) = \exp\left(\lambda \log(R_m)\right)\left((1-\lambda)I_3 + \lambda S_m\right) \tag{2.45}$$

This formulation is compatible with the pose distance defined in equation 2.13, which uses Frobenius norm for the shearing/scaling component of the transformation. Another possible solution for the interpolation of the shearing/scaling component consists of using a geodesic in the Riemannian manifold of positive semi-definite matrices, thus giving the following interpolated transformation:

$$\tilde{T}'(\lambda) = \exp\left(\lambda \log(R_m)\right) \exp\left(\lambda \log(S_m)\right) \tag{2.46}$$

This second formulation is compatible with the pose distance defined in equation 2.14, and which uses the Riemannian metric of the manifold of positive semi-definite matrices. In practice, and since the rotation component is predominant in this process, we couldn't notice any significant difference between these two formulations of local mesh deformations interpolation judging by the final mesh interpolation results.

Once we obtained the interpolated deformation gradients $\tilde{T}_m$, we deform each triangle based gradient G_m in the source mesh i by this interpolated local transformation $\tilde{T}_m$ to obtain the new gradients $\tilde{G}_m$ in this mesh:

$$\tilde{G}_m = \tilde{T}_m G_m \tag{2.47}$$

These new deformed gradients are used to reconstruct the interpolated mesh made with vertices $\{\tilde{v}_1, ..., \tilde{v}_N\}$ and triangles $\{t_1, ..., t_M\}$. This resulting surface consists of the vertex sites matching the deformed gradients manipulated in equation 2.47, and is found by solving the following linear sparse Poisson system:

$$\underbrace{G^T D G}_{\Delta} \begin{pmatrix} \tilde{v}_1^T \\ \vdots \\ \tilde{v}_N^T \end{pmatrix} = \underbrace{G^T D}_{\vec{\nabla}\cdot} \begin{pmatrix} \tilde{G}_1 \\ \vdots \\ \tilde{G}_m \end{pmatrix} \tag{2.48}$$

Note that in this equation all components G, D and $\tilde{G}_m$ are computed for source pose mesh i. The mesh function $v(.)$ and its derivatives change from a pose to another as the differential properties of the mesh change under non-rigid deformations. Hence G, D and G_m are different when computed for target pose mesh j. Evaluating these component at mesh i and solving equation 2.48 comes to deforming mesh i towards mesh j with interpolation parameter λ. This results in a first solution $\{\tilde{v}_1^i, ..., \tilde{v}_N^i\}$. One might also apply deformations $\tilde{T}_m^{-1}$ on mesh j gradients, evaluate the Laplace and divergence operators on this mesh, and then solve the system in equation 2.48, which comes to deforming mesh j towards mesh i with interpolation parameter $1 - \lambda$. We refer to this second solution as $\{\tilde{v}_1^j, ..., \tilde{v}_N^j\}$. In our experiments, the two solutions are usually not similar but quite close to each other. Hence to insure the uniqueness of the resulting mesh of our pose interpolation method, we consider the final interpolated mesh vertices $\{\tilde{v}_1, ..., \tilde{v}_N\}$ to be the linear interpolation of these two solutions:

$$\tilde{v}_k = (1 - \lambda)\tilde{v}_k^i + \lambda \tilde{v}_k^j \tag{2.49}$$

where index k is over the mesh vertices. Figure 2.8 shows an example of two pose meshes interpolation using our non-linear interpolation scheme and the linear one. We can see that our interpolation preserves better the mesh intrinsic properties such as the length of feet in this specific case.

2.5 Shape Motion transition

2.5.1 Gradual frame blending

We use gradual frame blending to generate motion transitions as it is done for Motion Capture data Kovar et al. [2002a]. Given a frame i in a source motion sequence and a frame j in a destination motion sequence (see figure 2.11a), we consider a window of successive frames of size l^i in the source sequence starting at frame i: $[\![i,\ i + l^i - 1]\!]$, and a similar window of successive frames of size l^j in the destination motion sequence ending at frame j: $[\![j - l^j + 1,\ j]\!]$. We refer to these segments as the source

and destination segments respectively. Assuming that these segments have the same length, i.e $l^i = l^j$, we can generate a smooth transition from frame i to frame j by gradually interpolating each pair of corresponding frames in source and destination segments directly without any further temporal adjustment, the pairs of frames to be interpolated being the set: $\{(i, j-l^j+1),\ (i+1, j-l^j+2),\ ...,\ (i+l^i-1, j)\}$. In this way, the resulting motion segment contains l^i frames, and the k^{th} frame in this segment is the interpolation of frame $i+k-1$ in source segment with frame $j-l^j+k$ in destination segment with interpolation parameter $(k-1)/(l^i-1)$, that varies from 0 to 1 when index index k goes from 1 to l^i increasingly.

This type of direct interpolated transition often results in transition segments that are visually uncompelling due to two major aspects:

- The first one is static geometry distortions caused by deformations that are not isometric or/and do not preserve the mesh volume. These deformations are encountered when interpolating pairs of meshes corresponding to poses that are far from each other. The resulting distortions, for instance body parts shrinkage for human models as shown in figure 2.8, can be substantially reduced through non-linear mesh interpolation techniques, such as the one we present in section 2.4. Furthermore, the data organizing structure layer presented in chapter 3 governs the selection of the necessary transitions within the dataset, and ensures that interpolations only happen between relatively close poses for the most part. However, regardless of all these measures, these geometric flaws are still present in interpolated motions. The frequency of their occurrence is directly proportional to the variance of the poses in the dataset, as more variance in the poses implies a higher probability of interpolating poses far from each other. This frequency is also inversely proportional to the size of the dataset, as less transitions are needed usually for animation synthesis with larger datasets for the same transition selection criteria.
- The second cause of transition segments' lack of visual compelliness is when dynamic motion properties are not well synchronized or do not share the same temporal scale which requires temporal expansion or compression and alignment prior to blending. For instance, blending a running segment with a walking segment recorded at the same frame rate requires a global fitting of the time scale and a local adjustment of the contact states with the ground, in order to avoid unrealistic phenomena such as abrupt change in motion speed or foot-skate. This latter refers commonly to the feet of the character sliding on the floor in computer animation jargon, and is usually reduced, for Motion capture data, by using contact information annotations to guide temporal alignment while blending motions as a pre-process Menardais et al. [2004], while the

remaining is cleaned-up as a post-process using inverse-kinematics Kovar et al. [2002b].

Unfortunately, as we are daily exposed to real scenes of various objects in motion, the human eye is extremely adequate and very well trained to spot the slightest manifestations of visual flaws and unsmoothness in synthetic motion data, even for inexperienced users. Besides the challenges of 3D shape synthesis, the dynamic aspect of motion data is very intricate to edit and synthesise as irregularities are easily noticeable. In parallel, the realism in animation industry products in continuously increasing thus raising the bar of expectations of users.

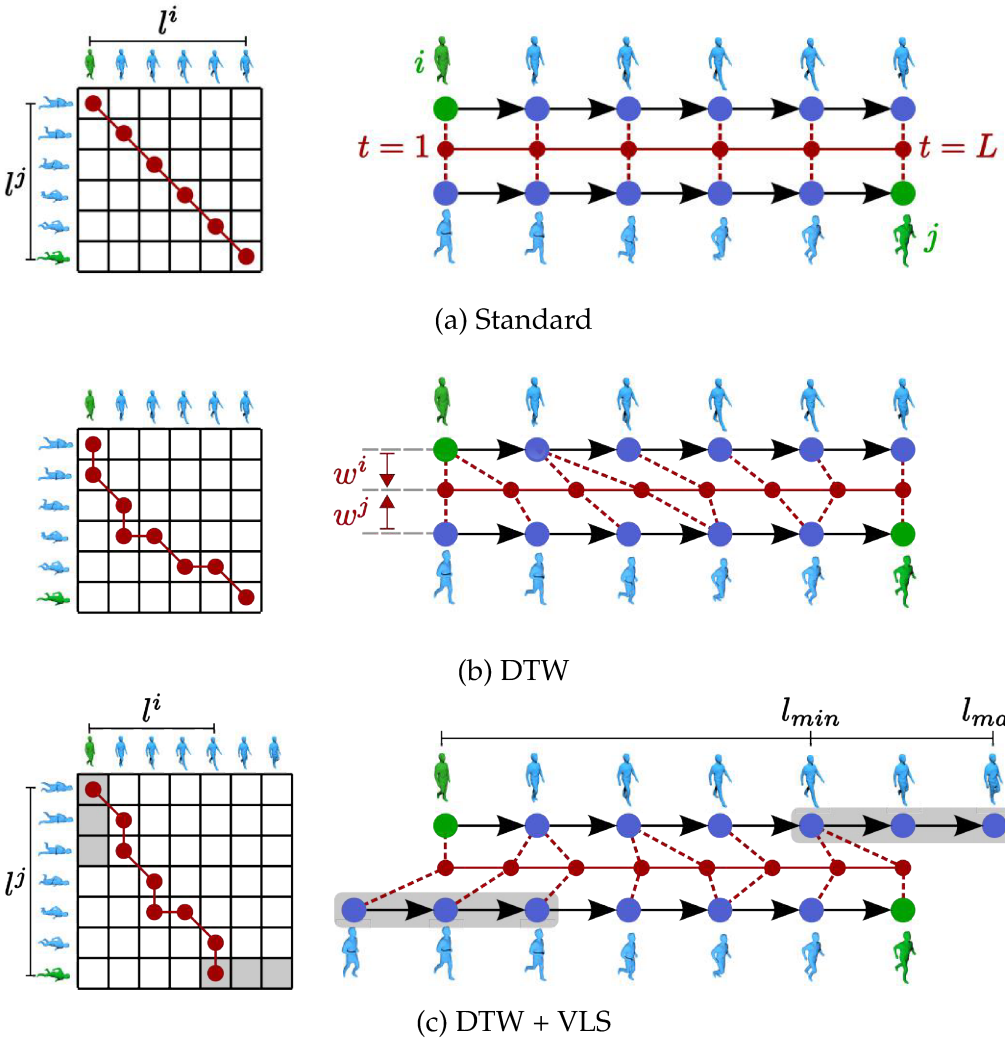
Hence, we propose to improve in this work on the standard gradual frame blending framework though the optimization of the parameters involved in the blending process. Going back to the formulation introduced in the beginning of this section, there are four elements that can be introduced or tuned to improve the interpolated transition from frame i to frame j using source and destination segments:

- First, temporal warps that we note w^i and w^j can be used to better align source segment and destination segment respectively.
- Second and last, source and target segment length l^i and l^j can be subject to optimization them selves.

Since no annotation such as contact information is available for our motion data, the only mean to evaluate the quality of synthesised transitions with a numerical criterion is through surface deformation cost, which is the cumulated distances between the poses to be interpolated in order to generate the desired transition, mesh based pose distance being defined in equation 2.13.

2.5.2 Dynamic time warping

We start with the time wrapping task. Given two fixed source and destination segment length l^i and l^j, our aim is to find two temporal warps w^i and w^j that when applied to the source segment $[\![i,\ i + l^i - 1]\!]$ and target segment $[\![j - l^j + 1,\ j]\!]$ in this order, the normalized cumulated surface deformation cost between corresponding frames in the warped segments is minimized. We solve this minimization with a variant of standard dynamic time warping algorithms that uses dynamic programming Müller [2007]. Dynamic time warping for signal time alignment was first introduced by Sakoe and Chiba [1987] and it was used in conjunction with dynamic programming techniques for the recognition of isolated words. It had been widely used since then mainly for recognition tasks.



(a) Standard

(b) DTW

(c) DTW + VLS

Figure 2.11 – gradual frame blending based transitions

As illustrated in figure 2.11b, we first construct an l^i by l^j distance matrix where lines stand for source segment's successive frames and columns for destination segment frames, such that the element at line k and column l equals $d(i+k-1,\ j-l^j+k)$, where $k \in [\![1, l^i]\!]$ and $l \in [\![1, l^j]\!]$. This two dimensional grid represents the solution space, where each node, i.e pair of line and column, corresponds to the association of the line's frame in the source segment to column's frame in the destination segment. The alignment solution is a path in this grid that joins case $(1, 1)$ in the top left corner, which stands for the frame association $(i,\ j - l^j + 1)$, to case (l^i, l^j) in the bottom right corner, which stands for frame association

$(i + l^i - 1,\ j)$. We note L the length of this path, which also represents the transition duration in time units. This path defines the warps applied to source and destination segments. If we view warps w^i and w^j abusively for the sake of simplicity as functions, their domains being sets $[\![i,\ i+l^i-1]\!]$ and $[\![j - l^j + 1,\ j]\!]$ respectively, and define $w^{i^{-1}}$ and $w^{j^{-1}}$ as their inverse functions, then the path defines the warps' image, which is the set $[\![1,\ L]\!]$. The t^{th} node in the path being at location (k, l) on the grid, with $t \in [\![1, L]\!]$, determines the inverse image of t by the warps, which writes $w^{i^{-1}}(t) = k$ and $w^{j^{-1}}(t) = l$, thus defining warps w^i and w^j at k and l respectively as-well: $w^i(k) = t$ and $w^j(l) = t$. We note additionally that the warps and the path are dual representations of the same operations on the segments to be aligned. As a matter of fact, when the path is diagonal between two successive nodes, this stands for a substitution operation in speech recognition terms Bruderlin and Williams [1995], and corresponds to the following case: $w^{i^{-1}}(t + 1) = w^{i^{-1}}(t) + 1$ and $w^{j^{-1}}(t + 1) = w^{j^{-1}}(t) + 1$. When the path is horizontal, it's a deletion operation and it corresponds to the case: $w^{i^{-1}}(t + 1) = w^{i^{-1}}(t)$ and $w^{j^{-1}}(t + 1) = w^{j^{-1}}(t) + 1$. Finally when the path is vertical, it's an insertion operation and it corresponds to the following case: $w^{i^{-1}}(t + 1) = w^{i^{-1}}(t)$ and $w^{j^{-1}}(t + 1) = w^{j^{-1}}(t)$.

We seek to minimize the normalized total cost of the dynamic time warping path through the distance matrix, which can be expressed as the summation of costs of all the nodes undertaken by the path through the distance matrix divided by the path length L. The cost of the t^{th} node of the path at location (k, l) on the grid being $d(i + k - 1, j - l^j + l) = d(w^{i^{-1}}(t), w^{j^{-1}}(t))$. The energy to be minimized writes then as:

$$E = \frac{1}{L} \sum_{t \in [\![1,L]\!]} d(w^{i^{-1}}(t), w^{j^{-1}}(t)) \tag{2.50}$$

As we do not only want to align signals, but we intend to actually warp the motion segments with these temporal warps, a set of conditions are imposed on the dynamic time warping path to insure the validity of the interpolated transition. This resulting motion segment should preserve the significant time related features of motion signals, which primarily include continuity and monotonicity. Continuity dictates that for each node t in the path, the next node $t + 1$ must be one of its direct neighbours on the grid. For the warps, this means that the absolute difference between the inverse image of t and $t+1$ is less than 1: $|w^{i^{-1}}(t+1) - w^{i^{-1}}(t)| \leq 1$ and $|w^{j^{-1}}(t + 1) - w^{j^{-1}}(t)| \leq 1$. Monotonicity imposes that the path does not reverse direction, which implies that for a node t, the next node $t + 1$ must

be one its right, bottom or right-bottom neighbours. This condition also means that the inverse warps are non-decreasing: $w^{i^{-1}}(t+1) \geq w^{i^{-1}}(t)$ and $w^{j^{-1}}(t+1) \geq w^{j^{-1}}(t)$. Another trivial constraint is the boundary condition, which states that the path starts at the top left corner at case $(1,1)$ and ends at the bottom right one at case (l^i, l^j). This translates to initial and final conditions on the inverse warps: the inverse image of 1 with source warp is i and with destination warp is $j - l^j + 1$, and the inverse image of L with source warp is $j - l^j + 1$ and with destination warp is j, which writes: $w^{i^{-1}}(1) = i$, $w^{j^{-1}}(1) = j + l^j - 1$, $w^{i^{-1}}(L) = i + l^i - 1$ and $w^{j^{-1}}(L) = j$. The last condition that we impose on the path is the slope constraint. To prevent the resulting motion from being degenerate, the path can not move in one direction more than p times in a row either horizontally or vertically. p consecutive nodes in the path in a row in the same direction must be followed by a diagonal move. In terms of warps, and for a node t in the path, this condition can be presented as follows: if $w^{i^{-1}}(t+k) = w^{i^{-1}}(t)$ and $w^{j^{-1}}(t+k) = w^{j^{-1}}(t) + k$, $\forall k \in [\![1, p-1]\!]$ then $w^{i^{-1}}(t+p) = w^{i^{-1}}(t) + 1$ and $w^{j^{-1}}(t+p) = w^{j^{-1}}(t) + p$. And identically if $w^{j^{-1}}(t+k) = w^{j^{-1}}(t)$ and $w^{i^{-1}}(t+k) = w^{i^{-1}}(t) + k$, $\forall k \in [\![1, p-1]\!]$ then $w^{j^{-1}}(t+p) = w^{j^{-1}}(t) + 1$ and $w^{i^{-1}}(t+p) = w^{i^{-1}}(t) + p$.

Taking these conditions into consideration, the minimization of the energy in equation 2.50 is obtained through a dynamic programming algorithm with a complexity of $\mathcal{O}(l^i \times l^j)$ which guaranties a globally optimal solution by visiting each node in the l^i by l^j grid once, with a constant amount of work per node. We recursively define a cost function $g(.,.)$ on the grid starting from case $(1,1)$ where we initialize it as follows:

$$g(1,1) = d(i, j - l^j - 1) \tag{2.51}$$

Then for each other node in the grid, the function value is obtained by exploring its possible predecessors in the path through the permissible routes allowed by the conditions elaborated previously. The permissible paths under different slope constraints which determine the potential predecessors are illustrated in figure 2.12. For each possible predecessor of the current node, a cost of this precession is computed as the summation of cost function g at that predecessor, plus the cumulative cost of the route joining the current node to this predecessor in terms of distance $d(.,.)$. The final predecessor of the current node is chosen as the one with the smallest precession cost among the candidates, and this very least precession cost is assigned to function $g(.,.)$ at the current node.

Following the scheme in figure 2.12, the potential predecessors of node (k,l) in the case of unconstrained path slopes, i.e $p = \infty$, are its left, top

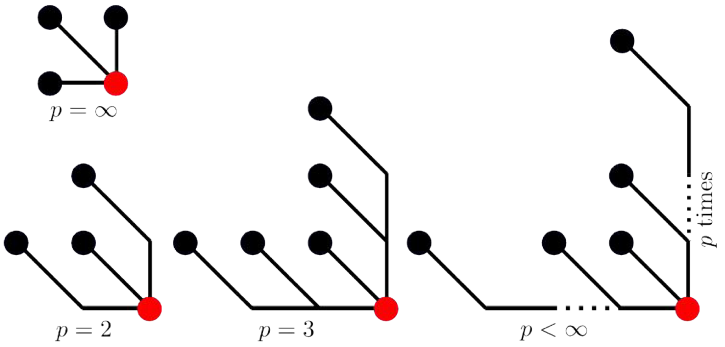


Figure 2.12 – Permissible routes through the grid for various slope constraints p on the dynamic time warping path. The Red node is the current node of the path and the black nodes are its possible predecessors

and top left direct neighbours: $(k-1, l)$, $(k, l-1)$ and $(k-1, l-1)$. And thus the predecessor of (k, l) is the one with the minimal cost among the three:

$$g(k,l) = \min \begin{cases} g(k, l-1) + d(i+k-1, j-l^j+l) \\ g(k-1, l) + d(i+k-1, j-l^j+l) \\ g(k-1, l-1) + d(i+k-1, j-l^j+l) \end{cases} \tag{2.52}$$

In case of a finite slope constraint $p < \infty$, the potential predecessors of node (k, l) are its top left direct neighbour $(k-1, l-1)$, then the top left node in the diagonal at each step after heading straight for $p-1$ steps (see figure 2.12), either vertically: $(k-2, l-1)$, $(k-3, l-1)$, ..., $(k-p, l-1)$, or horizontally: $(k-1, l-2)$, $(k-1, l-3)$, ..., $(k-1, l-p)$. The predecessor of (k, l) is the one among these $1 + 2 \times (p-1)$ nodes with the minimal path cost:

$$g(k,l) = \min \begin{cases} g(k-1,l-1) & +d(i+k-1, j-l^j+l) \\ g(k-2,l-1) & +d(i+k-1, j-l^j+l) \\ & +d(i+k-2, j-l^j+l) \\ \quad \vdots & \\ g(k-p,l-1) & +d(i+k-1, j-l^j+l) \\ & \quad \vdots \\ & +d(i+k-p, j-l^j+l) \\ g(k-1,l-2) & +d(i+k-1, j-l^j+l) \\ & +d(i+k-1, j-l^j+l-1) \\ \quad \vdots & \\ g(k-1,l-p) & +d(i+k-1, j-l^j+l) \\ & \quad \vdots \\ & +d(i+k-1, j-l^j+l-(p-1)) \end{cases} \tag{2.53}$$

In practice, we use a slope constraint of $p = 3$ in our experiments, and in this specific case equation2.53 writes:

$$g(k,l) = \min \begin{cases} g(k-1,l-1) & +d(i+k-1, j-l^j+l) \\ g(k-2,l-1) & +d(i+k-1, j-l^j+l) \\ & +d(i+k-2, j-l^j+l) \\ g(k-3,l-1) & +d(i+k-1, j-l^j+l) \\ & +d(i+k-2, j-l^j+l) \\ & +d(i+k-3, j-l^j+l) \\ g(k-1,l-2) & +d(i+k-1, j-l^j+l) \\ & +d(i+k-1, j-l^j+l-1) \\ g(k-1,l-3) & +d(i+k-1, j-l^j+l) \\ & +d(i+k-1, j-l^j+l-1) \\ & +d(i+k-1, j-l^j+l-2) \end{cases} \tag{2.54}$$

After browsing the grid by filling the values of function $g(.,.)$ and setting a predecessor for each node, starting from the top left corner node $(1,1)$ and reaching node (l^i, l^j) in the bottom right corner, the optimal solution is recovered by back tracking through the grid, starting form

this last node, thus obtaining the dynamic time warping path. Figure 2.13 shows how the optimal path in yellow obtained with a dynamic time warping optimization inside a distance matrix between two motion segments evolves when the slope constraint parameter p varies from 1 to 5. The first case on the left represents the linear time warping path. Notice how in this case, the path is far from the purple region with low surface deformation cost. The more we allow the path to contain consecutive vertical or horizontal nodes in a row ($p = 2, 3, ..., 5$), the closer it gets to the dark purple area thus resulting in a transition with lower total surface deformation cost. On the other hand, a big value of p might result in a temporally degenerate motion transition.

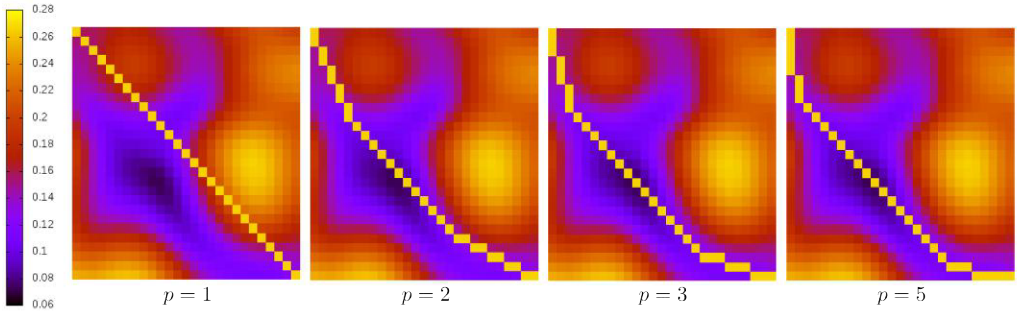


Figure 2.13 – Examples of a dynamic time warping path with various slope constraints

2.5.3 Varying blended segment lengths

Another set of parameters that can be optimized in the interpolated transition scheme is the source and destination segments lengths. Regarding these parameters, the work of Wang and Bodenheimer [2008] examines interpolated transitions for Motion Capture data and shows the impact of a suitable transition duration on the realism of the synthesised motion. In light of these results, that we assume can be generalized to mesh data as-well, we allow segments lengths l^i and l^j to vary between two boundaries l_{min} and l_{max} when generating a transition from frame i to frame j using gradual frame blending of source segment $[\![i, i + l^i - 1]\!]$ and destination segment $[\![j - l^j + 1, j]\!]$. the work of Wang and Bodenheimer [2008] argues that a transition duration for Motion Capture data should be comprised between a third of a second and 2 seconds. Assuming again that this observation can be applied to mesh data, we determine our boundaries l_{min} and l_{max} accordingly and with respect to the motion

signal frame rate. The datasets that we constructed is this work, namely TOMAS and CATY datasets, were recorded at 50 frames fer second. For these datasets, $l_{min} = 15$ and $l_{max} = 100$. Most of the other motion datasets available in the literature and that we used in our results are available at a frame rate of 24 frames per second, which implies segment boundaries of $l_{min} = 8$ and $l_{max} = 48$ for ADOBE, CAPOEIRA, DAN and JP datasets that we sued in our results.

2.5.4 Optimal motion transition

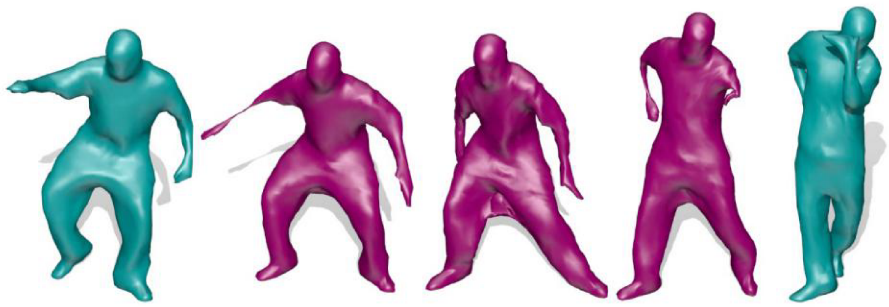
In addition to performing dynamic time warping for a better alignment, we seek the lengths l^i and l^j that minimize the normalized total surface deformation cost, assuming that this leads to a visually improved transition. This minimal cost defines the surface deformation cost $D(i, j)$ of going from frame i to frame j, which can be expressed as follows:

$$D(i,j) = \min_{l^i,l^j,w^i,w^j,L} \sum_{t\in[\![1,L[\![} d(w^{i^{-1}}(t), w^{j^{-1}}(t)) \tag{2.55}$$

In practice this minimization is solved by performing a dynamic time warping optimization for every pair of segment lengths (l^i, l^j) in $[\![l_{min}, l_{max}]\!] \times [\![l_{min}, l_{max}]\!]$. Additionally, transition duration L is uniquely defined by construction for a given set of segment lengths and temporal warps (l^i, l^j, w^i, w^j), and defines what we refer to as the duration cost of a transition $L_{i,j} = L$ (2.56) . Thus equation 2.55 rewrites:

$$D(i,j) = \min_{l^i,l^j\in[\![l_{min},l_{max}]\!]} \underbrace{\left(\min_{w^i,w^j} \sum_{t\in[\![1,L[\![} d(w^{i^{-1}}(t), w^{j^{-1}}(t)) \right)}_{Dynamic\ time\ warping} \tag{2.57}$$

In figure 2.14, we generate an interpolated transition from the green frame on the left side to the green frame on the right side, using the same source and destination motion sequences from JP dataset. On top, we show the result of our optimized interpolated transition that uses dynamic time warping along with variable length blended segments, with a slope constraint of $p = 3$. In the bottom, the result obtained with a standard interpolated transition. Both methods use the same non-linear mesh interpolation technique detailed in section 2.4. Notice how our interpolated transition avoids the surface distortions present



(a) Standard interpolated transition

(b) Optimized interpolated transition

Figure 2.14 – Comparison between a standard interpolated transition and our optimized interpolated transition. Green frames are original and red frames are interpolated.

in the standard transition in the dancer's arms due to the bad temporal alignment of the interpolated frames in red.

In conclusion, we present this transition optimization that combines variable length blended segment and dynamic time warping as a new contribution in our work that can be applied to both Surface Capture data and Motion Capture data, and regardless of the higher motion sequences organization layer, which can be a standard motion graph Kovar et al. [2002a], a parametric motion graph Heck and Gleicher [2007a], or the essential graph the we will introduce in this work for instance. We also note that while we use it here for motion transition generation, this temporal alignment technique can be used for aligning logically similar motions. This task is traditionally performed prior to motion parametrization either for Motion Capture data Kovar and Gleicher [2004a] or Surface Capture data more recently Casas et al. [2011a].

2.5.5 Evaluation

In this section, we evaluate our optimized interpolated transitions numerically by considering the normalized surface deformation costs and the durations costs of the transitions generated between all possible frame pairs in a motion dataset. The normalized surface deformation cost is the total cumulated cost of the dynamic time warping path between source and target motion segments of the transition, divided by the length of that path. The duration cost of a transition is the length of the dynamic time warping path.

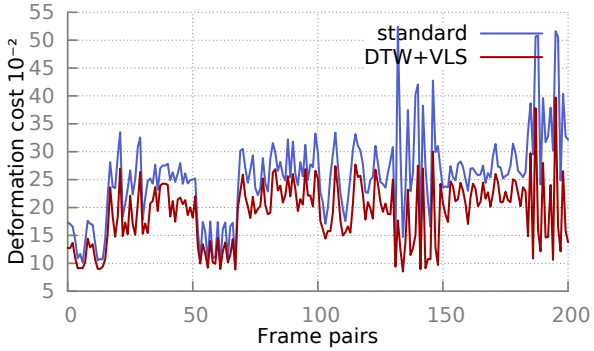
We also show qualitative comparisons between standard interpolated transition and our optimized transitions using various datasets in this video. As it can be seen in these results, our method allows us to avoid surface shrinkage, foot skate and other unnatural looking motion artefacts thanks to the improved alignment of poses in the interpolated motion segments prior to interpolation. Note that all these examples use non-linear mesh interpolation in the synthesis.

In Figure 2.15, we first show in Table 2.15a the average surface deformation cost over all pairs of frames in TOMAS dataset, using a standard interpolated transition scheme, transitions that use dynamic time warping only, then transitions that use a combination of dynamic time warping and varying length source and target segments. When dynamic time warping is involved, we evaluate the statistics of deformation cost for various slope constraints, allowing between 2 and 5 consecutive horizontal or vertical nodes in a row in the dynamic warping path. We can see that dynamic time warping improves transition deformation cost in average and this improvement increases when source and target segment lengths are optimized as-well. We notice also that relaxing the slope constraint results in lower normalized deformations costs. However, allowing a big number of vertical or horizontal nodes in a row in a dynamic time warping path might result in temporally degenerate transitions. Hence, we opt for an empirical limit of 3 allowed straight nodes in dynamic time warping for motion generation.

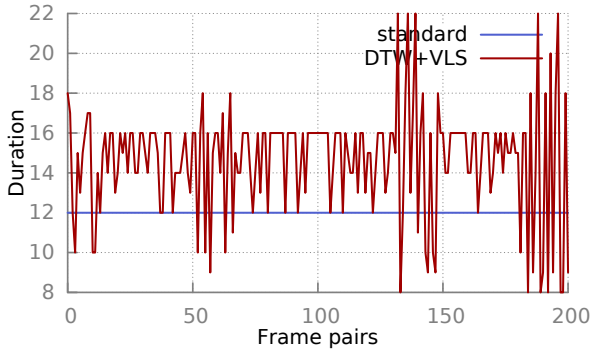
We plot the normalized deformation costs of transitions between a subset of all frame pairs from the dataset in Figure 2.15b using both standard transitions and our optimized transitions with a slope value of 3. We see that our method is clearly less costly. In figure 2.15c, we plot the corresponding durations of these transitions. Notice that the standard duration length is constant while our transition durations changes due to the use of dynamic time warping and also varying the interpolated segments' lengths.

			Mean	Std. dev.
Standard			0.2490	0.0758
DTW	Slope	2	0.2446	0.0769
		3	0.2427	0.0772
		4	0.2415	0.0774
		5	0.2409	0.0774
DTW + VLS	Slope	2	0.1942	0.0607
		3	0.1860	0.0574
		4	0.1817	0.0557
		5	0.1789	0.0547

(a) Average transition normalized surface deformation cost



(b) Transition normalized surface deformation costs

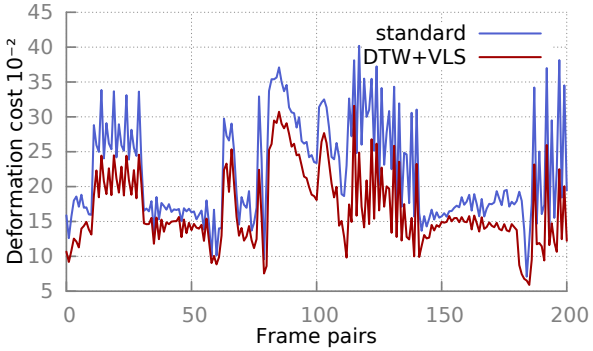


(c) Transition duration costs

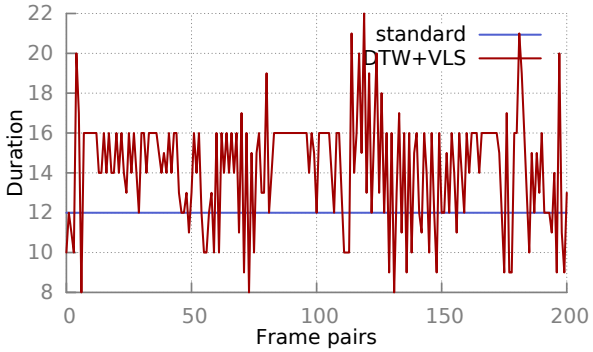
Figure 2.15 – Comparison of Standard transitions, transitions with dynamic time warping (DTW), and transitions with dynamic time warping and varying length segments (DTW+VLS) for TOMAS dataset.

			Mean	Std. dev.
Standard			0.2231	0.0706
DTW	Slope	2	0.2184	0.0713
		3	0.2161	0.0717
		4	0.2147	0.0719
		5	0.2138	0.0719
DTW + VLS	Slope	2	0.1731	0.0550
		3	0.1662	0.0519
		4	0.1629	0.0502
		5	0.1609	0.0490

(a) Average transition normalized surface deformation cost



(b) Transition normalized surface deformation costs

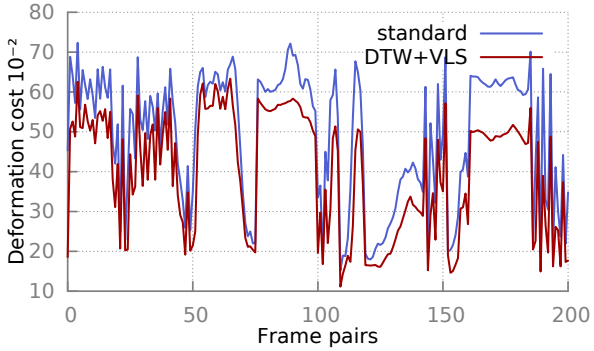


(c) Transition duration costs

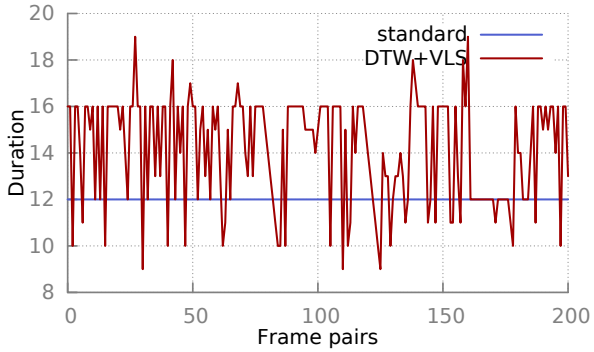
Figure 2.16 – Comparison of Standard transitions, transitions with dynamic time warping (DTW), and transitions with dynamic time warping and varying length segments (DTW+VLS) for DAN dataset.

			Mean	Std. dev.
Standard			0.4855	0.1626
DTW	Slope	2	0.4827	0.1630
		3	0.4817	0.1631
		4	0.4813	0.1632
		5	0.4812	0.1632
DTW + VLS	Slope	2	0.4043	0.1560
		3	0.3941	0.1543
		4	0.3893	0.1532
		5	0.3865	0.1525

(a) Average transition normalized surface deformation cost



(b) Transition normalized surface deformation costs



(c) Transition duration costs

Figure 2.17 – Comparison of Standard transitions, transitions with dynamic time warping (DTW), and transitions with dynamic time warping and varying length segments (DTW+VLS) for JP dataset.

Figures 2.16 and 2.17 show similar results for datasets DAN and JP respectively.

2.6 Conclusion

We presented in this chapter tools for evaluating distances between static shape poses and interpolating them. These tools are based on differential mesh processing and allow for a better assessment and manipulation of shape pose compared to straightforward vertex-based operations. We then used this static pose distance to build a tool that improves on standard gradual frame blending via a better temporal alignment of interpolated source and target segments. This is achieved by using dynamic time warping while allowing the length of the interpolated segments to vary within fixed boundaries. The compelling perceptual results obtained by our transition strategy stem from the numerical improvement it attains compared to standard interpolations. However, transitions between segments with substantially different shape poses and dynamics will still look unrealistic and contain mesh distortions even with optimized temporal alignments and non-linear mesh interpolation. Hence, the role of the animation organizing structure introduced in chapter 3 is to avoid these costly unrealistic transitions while generating animations.

Chapter 3

Shape animation synthesis

Contents

3.1 Introduction

Our example based animation scheme consists of reusing recorded 4D model data to generate spatially and temporally continuous motion streams only by concatenanting input segments and generating smooth transitions between them. In this respect, the problem that we aim to solve in this chapter is the following: Given a set of 4D models, how can we make the best use of this data to generate real looking motion animation

from user specified constraints ? We propose in this chapter an answer to this question in three parts:

- The first module is the user control level: We require an intuitive formulation of user defined constraints. These constraints must subsequently be transformed into a quantitative cost function and numerical synthesis conditions. This is detailed in section 3.5.
- The second module is the organizing structure level: We need a data structure that organizes the input sequences and encodes selected transitions between them. In order to select these transitions, we also need to define a numerical criterion to quantify the realism of a transition. In order to meet the requirement of the user perceptual acuity, the realism criterion must be reliable, and the data structure that we choose needs to deliver optimal results in terms of the latter criterion. Additionally, since we have access to a limited amount of data, one more challenge to the data structure is to make exhaustive use of the input sequences. Following seminal related work, we define a graph-based high level abstraction of the data to control the synthesis process, and a realism criterion that mixes surface deformation and duration costs as detailed in section 3.3.
- The last module is the motion data processing and synthesis level: We need to transform the output of the organizing structure level into a spatially and temporally continuous stream of meshes. This is achieved through the generation of realistic synthetic motion transitions, and seamless concatenation of real and synthetic motion segments. Since this work is applied to sensitive data with complex dynamics, this component relies on mesh processing and temporal alignment techniques explained in chapter 2, and other motion segment blending and alignment algorithms detailed in section 3.4.

The rest of the chapter develops the solutions we propose for each of these motion synthesis modules.

3.2 Related work

Motion graphs Kovar et al. [2002a] are an ubiquitous tool for automatic motion synthesis from examples of Motion Capture data in the literature. In this strategy, motion data sequences are represented by a graph where nodes stand for sequence frames and edges represent transitions between them (see figure 3.4). Thus for starters, every node is naturally connected to the next one in the same sequence. The method tries then to add new edges in the graph. For every pair of sequences in the dataset, a frame to

frame pose or/and velocity dissimilarity matrix is computed. The new transition points between this pair of sequences correspond then to the local minima in this matrix. Thresholding can eventually be performed on the resulting selected transitions to reduce their final number. Finally, Graph pruning is performed on the final structure to remove dead ends.

Motion graphs were extended to mesh data animation as-well. The work of Huang et al. [2009] was one of the first to use motion graphs on mesh data. As far as the organizing data structure is concerned, this work contributed a novel approach for optimal path search to avoid infinite numbers of paths caused by graph cycles. However, due to the lack of temporal coherency of data, no interpolation based transitions were used. Recently, the work of Prada et al. [2016] also used motion graphs on unstructured mesh sequences, i.e. sequences with inconsistent mesh topology. They contributed a solution for appearance interpolation in addition to geometry synthesis. Interpolation based transitions were performed through template based tracking applied locally to pre-selected similar source and target mesh transition segments only.

Although simple and intuitive, the way transitions are added in a standard motion graph follows a heuristic that does not guarantee the optimality of motion synthesis with graph walks. We try hence to improve this aspect with a novel structure that we introduce in this chapter, the essential graph, on which we will elaborate in section 3.3.1 . This optimality requirement is more vital in our case because of the data we use. Unlike skeletal motion capture data, interpolating and editing mesh data is non-trivial and hence motion transitions should be selected even more carefully for this data.

Other variants of motion graphs were proposed in the literature to solve the problem of automatic animation synthesis from examples for motion capture data. For instance, the work of Arikan and Forsyth [2002] also creates a graph where nodes represent sequences, and transition edges are subsequently added between sequence frame pairs if the dissimilarity cost between them is lower than a user specified threshold. A hierarchical randomized search is used then to generate motions. The authors of Lee et al. [2002] build first a directed graph in a similar manner to motion graphs. They then map the frame pose and velocity metric to a transition probability and and thus consider the resulting digraph as a Markov Process. They finally build a higher statistical model over this later for more intuitive control.

Other works proposed improvements that build on motion graphs, such as Interpolated Motion Graphs Safonova and Hodgins [2007], Well-Connected Motion Graphs Zhao and Safonova [2008], or Optimization

Based Motion Graphs Ren et al. [2010]. The structure proposed by Safonova and Hodgins [2007] consists of interpolating time-scaled paths within a standard motion graph. Zhao and Safonova [2008] increases the connectivity in a motion graph by using multi-target blending to generate intermediate interpolated motions between some of the dataset sequences and builds subsequenlty a more connected motion graph that incorporates these new motions. Ren et al. [2010] constructs an optimization-based graph combining constrained optimization and motion graphs.

We are well aware of the existence of improvements and extensions done to motion graphs, as the ones mentioned above. However, we note here that we only compare our method to a standard motion graph in the evaluation section, both quantitatively and qualitatively with visual results, for the following reasons:

- There is an enormous body of work on motion graph extensions in computer animation.
- Most of these improvements start from a motion graph as an initial stage and build on it by enhancing the connectivity. Our method serves the same purpose as a motion graph in these works, and can be in theory subject to the same improvements in a follow up step.
- Most of these methods, like interpolated motion graphs, imply even more and longer interpolations in the input data. In some cases, data resulting from interpolated inputs is also re-incorporated in the input set, and can be subject to additional rounds of new interpolations. In the case of surface capture data, we try to avoid long and costly interpolations for mesh data as much as we can since it is much more delicate to edit than skeletal data.
- Most of these methods use contact information that comes as annotation with standard motion capture data. This information is not available for mesh data.

Some statistical approaches were proposed to solve this problem also. In a two-level model, Tanco and Hilton [2000] apply dimension reduction and clustering to the input dataset and construct a Markov chain with these clusters as states in the first level. The second level consists of a hidden Markov model that relates the states of the Markov chain to the motion examples. Given a source and destination key frames defined by the user, the corresponding sequence of clusters in found in the first level, and the most likely succession of motion segments is found in the second. But unlike skeletal motion capture, and besides their very high dimensionality, 3D mesh motion datasets available currently are mostly limited, i.e. temporally short and poor in content variability. Thus, a similar statistical approach would not be very adequate to this specific

type of data.

Using skeletal data, Min and Chai [2012] proposes a graph where nodes represent motion primitives and edges transitions between them. Each node stores a statistical model learned from similar motion segments associated to the same primitive, and a transition distribution function is associated to every directed edge. Unfortunately, mesh motion datasets available currently do not offer multiple examples of the same motion type, which makes such statistical generative models inapplicable in this case. Additionally, unlike this method, we also need to solve for random acyclic motion, such as an unplanned dancing sequence (see section 3.5.2), which seems hard to decompose into distinctive simple motion primitives.

In a graph-less approach, the work of Ikemoto et al. [2007] clusters motion segments and pre-computes a table defining the intermediary segments needed to blend all pairs of cluster medoids. In order to generate a transition from a source segment to a target segment, these two segments are thus blended together with their respective cluster medoid intermediaries. Although this work gives in theory the possibility to find transitions between all pairs of frames in the dataset, it requires a transition naturalness scoring, motion segment clustering and mutli-way frame blending which would be difficult to achieve with 3D mesh sequences and without contact information.

More recently, the authors of Holden et al. [2016] introduced a deep learning framework for animation synthesis . First, and as detailed in this work Holden et al. [2015], a convolutional auto-encoder network is used to learn a low dimensional embedding of normalized motion capture data. Forward neural networks are used then to learn mappings between high level user constraints and motion segments in the low dimensional motion space. This pioneer work required a dataset twice larger than the size of the CMU motion capture database containing around six million frames for a single character sampled at 120 frames per second. Unfortunately, no such amount of surface capture data is available.

In a work similar to [Kovar and Gleicher, 2004a] with skeletal data, [Casas et al., 2011a] achieves motion synthesis by high-level parametrization of mesh data. These approaches consist in first temporally aligning logically similar motion clips, guided usually by additional annotation such as foot contact information. These aligned motions are then blended, and blending weights are mapped to high level parameters, thus creating parametric motion spaces. In the work of Kovar and Gleicher [2004a] fir instance, weights are inverse-proportional to the distance between the sample and the desired final positions of a leg in a kicking motions, of the hand in reaching motions. In the work of Casas et al. [2012] for in-

stance, a walking motion space is parametrized with speed, and jumping motion spaces are parametrized either by jump height or length. These parametrized spaces allow for the creation of a supervised variant of motion graphs called Parametric motion graphs, which are similar to move-trees Menache [2000] used for interactive control in video games. the graphs consist of connecting several parametric motion spaces either through pre-computed transitions, as in Heck and Gleicher [2007a] for skeletal data, or by computing optimal transitions at runtime, as in Casas et al. [2012] for 3D meshes. We note that The idea behind parametric motion graphs is similar to that of Fat graphs Shin and Oh [2006]. These methods allow motion synthesis with precise control but only for specific input data and require a high level of supervision. Our objective is different and improves over Huang et al. [2009] with globally optimal motion sequences generated from a general set of input sequences, and not necessarily cyclic or parametrizable motions.

3.3 Organizing graph structure

This section details the steps involved in the construction of the essential graph for mesh based motion segments. The input data is a set of temporally aligned sequences of 3D triangulated meshes, i.e. meshes present the same topology and vertex connectivity over all sequences. These sequences represent a model, typically but not necessarily a human, undergoing arbitrary movements with no prior restrictions on the nature of the movements. It should be noticed here that while we consider meshes as input, the graph construction is independent of the pose model used and could be applied with other parametrizations of shape and motion including articulated models.

3.3.1 Essential graph

We use a directed weighted graph structure to organize our input mesh sequences with the same semantics as motion graphs. That is, nodes represent frames and edges stand for possible transitions between them. Our goal is to synthesise optimal continuous motions from a set of example sequences with respect to a certain realism criterion. Hence, edge weights in the graph, that we note $E_{i,j}$ for an edge linking node i to j, represent the realism of transition segment (i, j) evaluated with the same criterion.

In a traditional motion graph, edges between sequences are obtained by selecting minima in transition cost matrices between pairs of sequences. This strategy is intuitive but yet not globally optimal, as transitions between sequences other than minima might give better total cost to a graph walk if the minimized criterion considers also duration for instance, which is an important constraint for motion synthesis. We propose therefore a global and principled strategy that consists of extracting the best paths between any pairs of poses and to keep only edges in the graph that contribute to these paths. This corresponds to extracting the essential sub-graph from the complete digraph induced from the input sequences. Unlike previous methods, ours ensures the existence of at least one transition between any two nodes in the graph, which potentially yields a better use of the original data with less dead ends trimming.

The three steps of construction of the essential graph are as follows:

- As shown in figure 3.3b, the initial instance of the graph is drawn from the input motions. Hence, edges in the graph only connect successive frames in the original sequences.
- In the second stage, we connect all nodes in the graph with directed weighted edges, thus forming a complete digraph, As shown in figure 3.5a. In our implementation, function $E_{.,.}$ satisfies positivity and positive definiteness, but it is lacking symmetry and triangle inequality to define a proper metric. Hence, the resulting graph structure is analogous to a quasi-semi-metric space, (V, E), where V is the set of nodes and $E : V \times V \rightarrow \mathbb{R}^+$ is the asymmetric function that defines edge weight values.
- Finally, for every pair of nodes in the graph, we use Dijkstra algorithm to find the path with the least cost joining the source node to the destination node. The cost of a path $p = [n_1, n_2, ..., n_N]$ that goes through nodes $n_1, n_2, ..., n_N$ is defined as the sum of the costs of all edges forming this path sequentially:

$$J(p) = J([n_1, n_2, ..., n_N]) = \sum_{i \in [\![1, N-1]\!]} E_{n_i, n_{i+1}}. \tag{3.1}$$

Once the optimal paths joining all pairs of nodes are found, all edges that do not belong to any of these paths are pruned. This gives an optimally connected graph which contains only the necessary edges to connect every pair of nodes with the best possible path cost in terms of realism (figure 3.5c). This resulting structure is also referred to as the union of shortest path trees rooted at every graph node.

Figure 3.1 shows an essential graph built using TOMAS dataset.

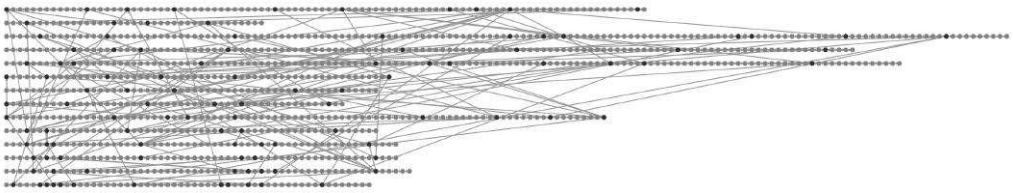


Figure 3.1 – Essential graph built with TOMAS dataset.

3.3.2 Graph edge weights

Weights attributed to edges in our graph should ideally capture the realism of the transitions they represent. This realism is quantified with a a criterion that derives from both physical and temporal considerations. It takes into account a surface interpolation cost, that measures the amount of surface deformation along a motion segment, as well as a duration cost that accounts for the number of frames of a segment. The first cost tries to preserve shape consistency along a motion segment while the second tries to minimize the number of poses within a segment.

Assuming that a realistic transition should involve as little surface deformation as possible, we use the optimal transition cost $D(.,.)$ defined in equation 2.57 as a metric that measures the surface deformation cost between any pair of nodes in the graph. As explained in section 2.5.4 , this metric evaluates the minimal normalized cumulative deformation cost through a dynamic time warping path, allowing for the best interpolated transition between a source and a destination frame segments. It involves several intermediate poses for that purpose, which makes it different from the static pose $d(.,.)$ similarity defined in equation 2.13. Naturally on another hand, edges connecting successive frames in the original data, present in the initial instance of the graph (figure 3.3b), have a null surface deformation cost. This cost, in addition to the transition duration, allows us to quantitatively evaluate the realism of a transition, as it was done in previous works Casas et al. [2012].

It is important to consider a temporal component in the realism criterion. As a matter of fact, the duration of a generated motion segment is one of its crucial characteristics and it is usually subjected to limitations deriving from the synthesis scenario conditions. Hence it must be subject to optimization in the process of motion synthesis and should be consequently represented in edge weights. Given a query pair of source and destination frames, and at the expense of some amount of surface deformation cost, we might obtain a graph walk that is much shorter than the one with the least surface deformation cost, and the difference

in surface deformation cost between these two solutions might not even be perceptually noticeable. We use the optimal transition duration $L_{.,.}$ defined in equation 2.56 as a metric that measures the duration cost between any pair of nodes in the graph. As explained in section 2.5.4, this metric evaluates the length of the Dynamic time warping path, allowing for the best interpolated transition between a source and a destination frame segments. Naturally on another hand again, edges connecting successive frames in the original data, present in the initial instance of the graph (figure 3.3b), have a one time unit duration cost, assuming uniform input data sampling.

if *i and j belong to the same sequence and $i < j$* **then**

$$E_{i,j} = \min[\alpha(j-i), D_{i,j} + \alpha\, L_{i,j}], \tag{3.2}$$

else

$$\underbrace{E_{i,j}}_{\text{realism}} = \underbrace{D_{i,j}}_{\text{surface deformation}} + \alpha \underbrace{L_{i,j}}_{\text{duration}} \tag{3.3}$$

end

Algorithm 1: Transition realism definition.

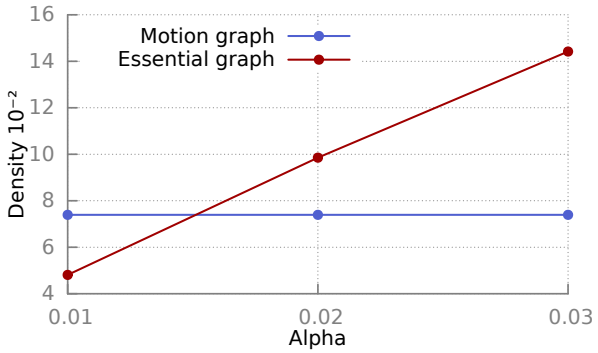
In conclusion, for each pair of nodes i and j in the graph, we define the weight E_{ij} of the directed edge (i, j) as the weighted sum of the optimal surface deformation cost $D_{i,j} = D(i, j)$ and the associated duration cost $L_{i,j}$: $E_{ij} = D_{i,j} + \alpha L_{i,j}$. In the situation where frames i and j belong to the same sequence, with $i < j$, it is unnecessary to consider an interpolated transition if its cost is greater than $\alpha(j - i)$, that is the cost of going from i to j through the original motion sequence. Algorithm 1 summarizes the definition for the edge weight $E_{i,j}$ between poses i and j.

The weight α represents the ratio of tolerance between surface deformation and transition duration. This user defined coefficient allows for some flexibility on the admissible surface deformation during a transition with respect to time duration. It controls the permissibility of adding new edges in the essential graph and hence the density of this later.

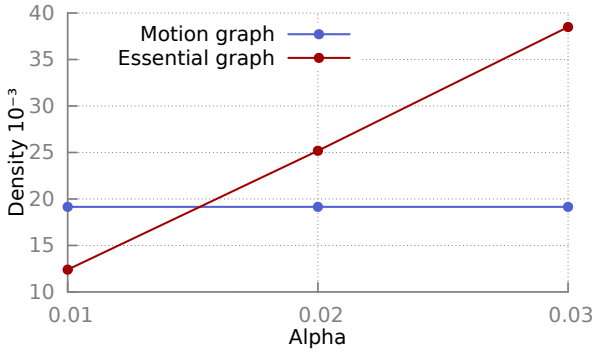
We define the density of a directed graph as the ratio of the number of its edges to the maximum number of edges:

$$\text{Density} = \frac{|E|}{|V|\,(|V|-1)} \tag{3.4}$$

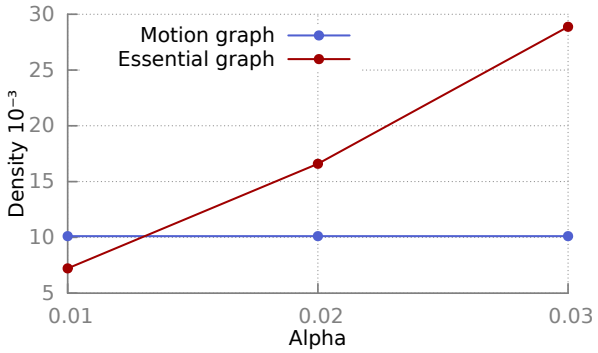
where operator $|.|$ is the cardinality of a set and V and E represent the sets of graph nodes and edges respectively.



(a) TOMAS dataset



(b) DAN dataset



(c) JP dataset

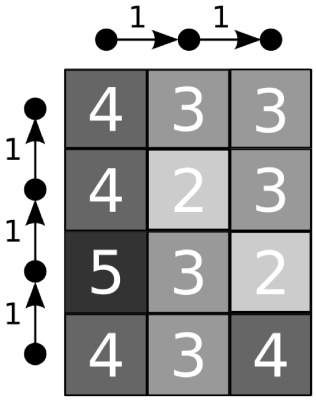
Figure 3.2 – Graph density

As one can see in figure 5.3, and using datasets TOMAS(3.2a) , JP(3.2c) and DAN(3.2b), we constructed motion graphs and essential graphs with various values of α. We evaluated the resulting graph structures and noticed that the density of the essential graphs increases almost linearly with the value of α. In practice for our synthesis experiments, we use a

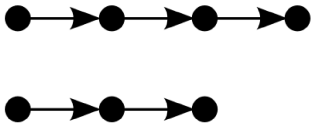
value of $\alpha = 0.01$, which seems to result in a good compromise between surface deformation and duration costs in terms of perceptual results.

3.3.3 Illustrative example

We construct the following simple synthetic example for a comparative visualization of data organizing structures. We are given two motion sequences represented by node sequences as shown in figure 3.3b. We are also given transition costs between every pair of nodes in this dataset as shown in figure 3.3a. We set the transition cost between two consecutive nodes in the same frame to the value of 1, and transition costs between frames belonging to different sequences are given in the transition cost matrix (figure 3.3a.).



(a) Transition costs

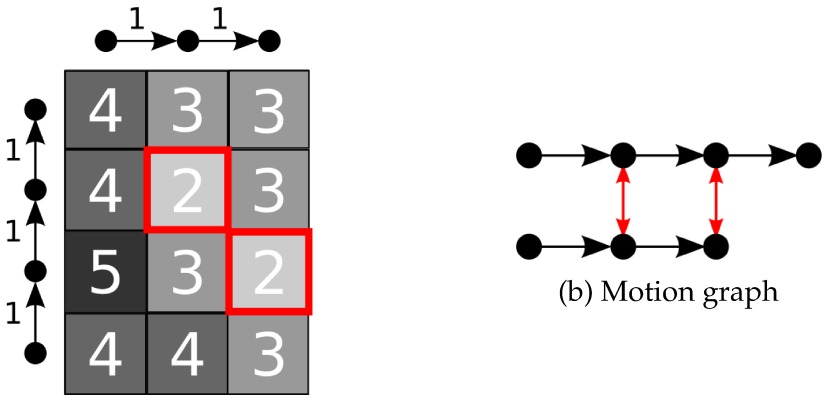


(b) Initial graph

Figure 3.3 – Graph representation of input sequences.

Figure 3.4 shows how a motion graph is constructed from the input sequences. Local minima in the dissimilarity matrix correspond to new added transitions (figure 3.4a). Namely, we add two bidirectional transitions: between the second node of the first sequence and the second node of the second sequence; between the third frame of the first sequence and the third frame of the second sequence (figure 3.4b).

To build the essential graph, first all nodes are connected thus creating a complete digraph (figure 3.5a). Then for every node in the graph, we find the shortest path tree rooted at this node by computing the shortest paths between this node and the rest of the graph nodes, in the previous complete digraph. Figure 3.5b shows the shortest path tree rooted at the first node of the first sequence. We repeat this process for the rest of the graph nodes, thus obtaining a shortest path tree rooted at each

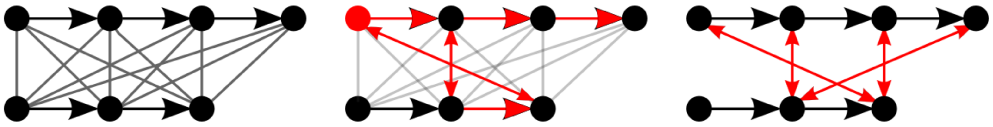


(a) Local minima

(b) Motion graph

Figure 3.4 – Building a motion graph.

node. Finally, the essential graph is made of the union of all these trees (figure 3.5c). By the end of the process, we add the following bidirectional transitions: between the first node of the first sequence and the third node of the second sequence; between the second node of the first sequence and the second node of the second sequence; between the third frame of the first sequence and the third frame of the second sequence; between the fourth node of the first sequence and the second node of the second sequence.



(a) Complete digraph (b) Shortest path tree (c) Essential graph

Figure 3.5 – Building an essential graph.

We subsequently set edge weights for both the motion graph and the essential graph to transition costs defined in figure 3.3a. This setting relates to our edge weights defined in section 3.3.2 , by setting the α parameter to 1, and considering dissimilarity matrix in figure 3.3a as the surface deformation cost of transitions referred to as $D(.,.)$ in section 3.3.2. Thereafter, we want to generate optimal graph walks between pairs of source and destination nodes.

Figures 3.6 and 3.7 show the ability of the essential graph to generate more optimal shortest paths than the motion graph. We first find the optimal transition from the first node in the first sequence to the third node in the second sequence using the essential graph (figure 3.6b) and

(a) Motion graph (b) Essential graph

Figure 3.6 – Shortest path from node $(1, 1)$ to node $(2, 3)$

(a) Motion graph (b) Essential graph

Figure 3.7 – Shortest path from node $(2, 1)$ to node $(1, 4)$

the motion graph (figure 3.6a). We can see that the essential graph outperforms the motion graph by giving a path with a cost of 3 while the smallest transition cost that can be obtained with the motion graph is 4. In the same way, we show the optimal transition from the first node in the second sequence to the fourth node in the first sequence using the essential graph (figure 3.7b) and the motion graph (figure 3.7a). We can see again that the essential graph outperforms the motion graph by giving a transition with a smaller cost of 4 compared to the cost of 5 given by the optimal transition obtained from the motion graph.

3.3.4 Evaluation

We evaluate in this section our organizing data structure, namely the essential graph, by comparing it to a standard motion graph both quantitatively and qualitatively. For this comparison, we use datasets TOMAS, DAN and JP, which contain 3D temporally coherent mesh sequences of real human performance capture.

Given a dataset of mesh sequences, we want to generate motions joining all frames minimizing a joint cost of surface interpolation and duration (equation 3.3). To this end, we construct an essential graph and a motion graph, and compute the costs of optimal paths joining every pair of frames in the dataset. In order to compare graph structures independently of the transition approach, we use the same transition cost for both methods. To construct a motion graph, we compute transition cost matrices between all pairs of sequences and select the local minima within them as transition points. In the original paper implementation, it is also recommended to

only accept local minima below an empirically determined threshold to reduce the computational burden. We deliberately ignore this measure to obtain a highly connected motion graph to better challenge our method. Note that we do not compare to existing extensions of the motion graph since our primary objective is to evaluate the initial structure that is used to select optimal transitions and we believe that many of these extensions could anyway apply to the essential graph as well. We reiterate these experiments for the three datasets and for different values of the α parameter.

Figures 3.8a, 3.9a and 3.10a, show the mean and the variance of the shortest path costs between all frames in TOMAS, DAN and JP datasets respectively for both the essential graph and the motion graph, and for different values of α. We also plot the optimal transition cost between a subsample of the dataset frames for cases $\alpha = 0.2$ and $\alpha = 0.05$ for datasets TOMAS (figures 3.8b and 3.8c), DAN (figures 3.9b and 3.9c) and JP (figures 3.10b and 3.10c).

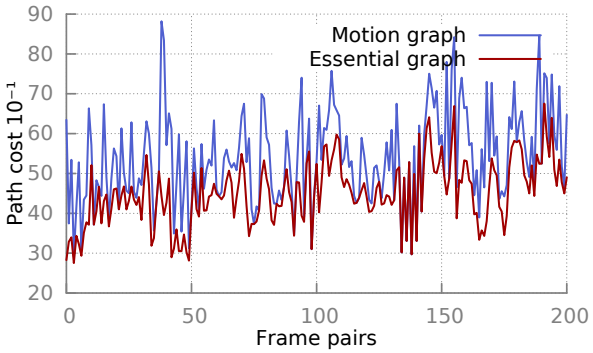
As summarized in figures 3.8, 3.9 and 3.10, the essential graph gives shortest paths with smaller costs than the essential graph in average for all our datasets. This is also visually confirmed in the plots of the costs of the shortest paths joining a subsample of the dataset frame pairs, where the motion graph curve is at it best always above the essential graph curve.

One could suspect that our graph representation outperforms a standard motion graph numerically merely because of a higher graph density for instance, as a graph with more edges allows for more flexibility in path optimization. The results in figure 5.3 show that this hypothesis is not true: For a value of $\alpha = 0.01$ for instance, essential graphs are less dense than motion graphs for the three datasets TOMAS, DAN and JP. Nevertheless, Figures 3.8a,3.9a and 3.10a still show a lower average shortest path cost for the essential graphs compared to motion graphs for the three datasets in this case. Hence, with a more sparsely connected graph, our method still performs numerically better than standard motion graphs. This proves that the reason behind the good performance of our method is not the density of our graph but rather its better connectivity.

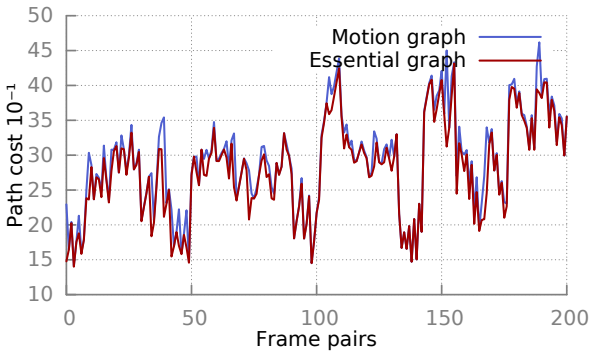
From figures 3.8b and 3.8c for TOMAS, 3.9b and 3.9c for DAN and 3.10b and 3.10c for JP datasets, we also notice that for smaller values of parameter α, the essential graph converges in its shortest path costs towards the motion graph, as path cost plots tend to look more similar to each other. As a matter of fact, building the essential graph with $\alpha = 0$ is equivalent to discarding the duration cost in the numerical realism criterion, which means solving the optimal transition selection problem in the case where initial edges between original sequence nodes are costless.

α	Motion Graph		Essential Graph	
	Mean	Std. dev.	Mean	Std. dev
0.01	2.0696	0.6040	2.0366	0.5985
0.02	2.3072	0.6203	2.2619	0.6168
0.03	2.5232	0.6375	2.4583	0.6331
0.05	2.9201	0.6800	2.8008	0.6694
0.1	3.8128	0.8158	3.4756	0.7406
0.2	5.4628	1.2008	4.5102	0.7917

(a) Average shortest path costs



(b) Shortest path costs, $\alpha = 0.2$



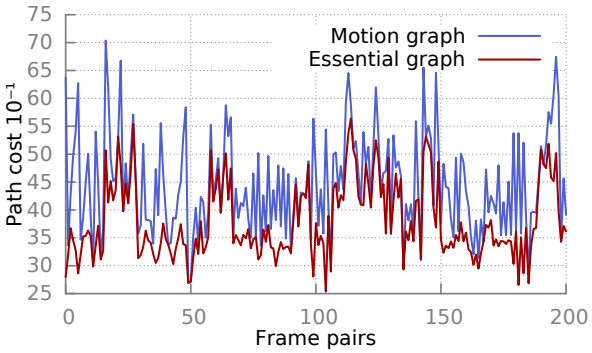
(c) Shortest path costs, $\alpha = 0.05$

Figure 3.8 – Comparison of Motions graphs and Essential graphs using TOMAS dataset.

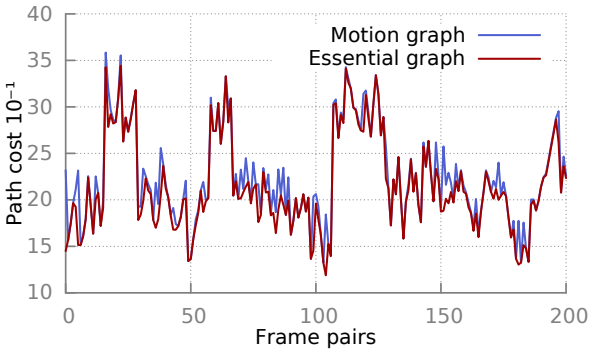
In this specific case, we know that the motion graph covers the optimal solution by definition, and hence it coincides with the essential graph.

α	Motion Graph		Essential Graph	
	Mean	Std. dev.	Mean	Std. dev
0.01	1.5589	0.4757	1.5509	0.4753
0.02	1.7595	0.4782	1.7392	0.4785
0.03	1.9391	0.4844	1.9006	0.4858
0.05	2.2695	0.5069	2.1830	0.5077
0.1	3.0397	0.6043	2.7847	0.5759
0.2	4.5011	0.8927	3.8083	0.6837

(a) Average shortest path costs



(b) Shortest path costs, $\alpha = 0.2$



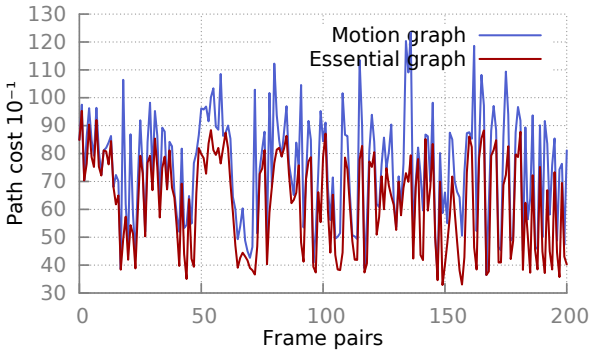
(c) Shortest path costs, $\alpha = 0.05$

Figure 3.9 – Comparison of Motions graphs and Essential graphs using DAN dataset.

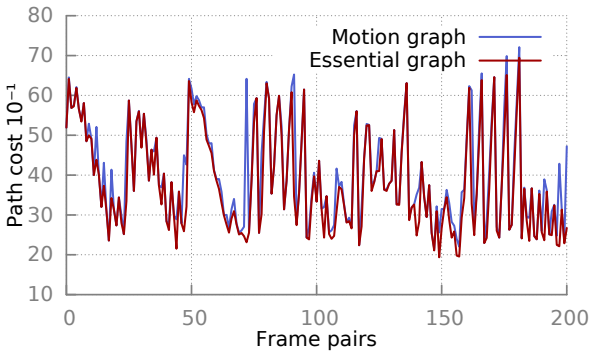
The performance of our method comes however with a significant computational complexity. We Denote in the following by V the set of graph nodes, i.e. motion dataset frames, and by E the set of graph

α	Motion Graph		Essential Graph	
	Mean	Std. dev.	Mean	Std. dev
0.01	2.2705	0.3990	2.2272	0.4002
0.02	2.7922	0.6676	2.7184	0.6619
0.03	3.2526	0.9098	3.1367	0.8999
0.05	4.0316	1.2635	3.8090	1.2454
0.1	5.4115	1.6108	4.8709	1.5476
0.2	7.5315	2.0157	6.2073	1.7041

(a) Average Shortest path costs



(b) Shortest path costs, $\alpha = 0.2$



(c) Shortest path costs, $\alpha = 0.05$

Figure 3.10 – Comparison of Motions graphs and Essential graphs using JP dataset.

edges. The complexity of a simple implementation of Dijkstra algorithm is $\mathcal{O}(|E| + |V|^2)$ which equals $\mathcal{O}(|V|^2)$ since $|E| = \mathcal{O}(|V|^2)$, where operator

$|.|$ symbolizes the cardinality of a set. In practice, we only need to find the shortest path between all pairs of node in the complete digraph in order to find the essential graph. Hence, the final complexity of building the essential graph naively amounts to $\mathcal{O}(|V|^4)$. However, we are not highly concerned with this cost since this step is performed off-line in our animation framework, and also mesh motion datasets available so far are relatively small in size.

Finally, for qualitative evaluation, we show some visual results where motion graphs fail to find graph walks as short and costless as essential graphs (see video).

3.4 Motion synthesis

Motion synthesis consists of converting a walk in the essential graph into a temporally and spatially continuous motion stream in the form of a 3D mesh sequence. These sequences are made of a mixture of original motion parts from the input dataset and synthetic ones.

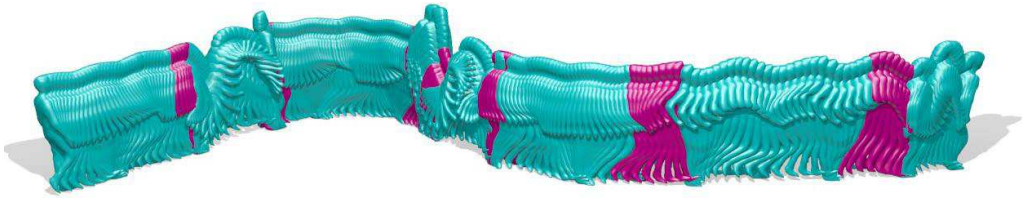


Figure 3.11 – A walk in the essential graph built with DAN dataset. Green frames are original, Red frames are interpolated.

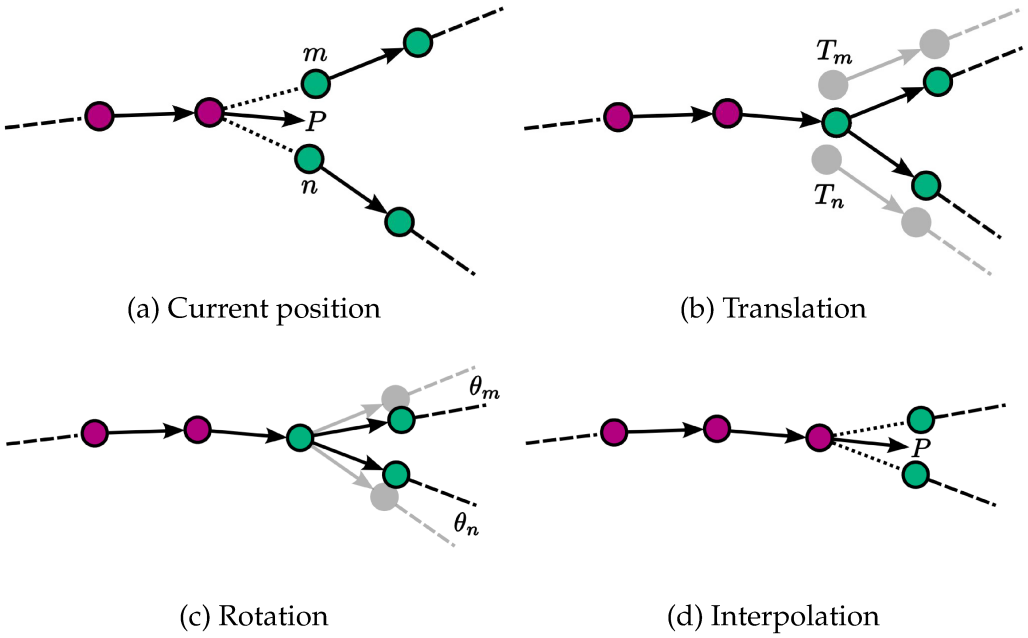
A Graph walk path is represented by a node sequence, where every pair of successive nodes must belong to an edge in the essential subgraph. We browse the path and sequentially identify two types of motion segments: Any succession of nodes belonging to the same input sequence in their original order forms an original motion segment, represented in green in figure 3.11. When a non original transition appears in the graph path, it signals the end of the current original motion segment. The next segment consists then of the interpolated transition frames, and we refer to it as a synthetic motion segment, represented is red in figure 3.11.

3.4.1 Transition synthesis

A synthetic motion segment represents a transition from a frame (i.e. node) i to a frame j in two sequences of the dataset. Such segment is

generated using the optimal parameters of the interpolated transition introduced in section 2.5.4. Let us recall that these optimizations were aimed to obtain the most naturally looking possible direct transition between a pair of frames. We retrieve the optimal parameters relative to the transition, which are source segment length l^i and temporal warp w^i, destination segment length l^j and temporal warp w^j and transition length L. Subsequently, we blend source segment $[\![i, i+l^i-1]\!]$ warped by w^i with destination segment $[\![j-l^j+1, j]\!]$ warped by w^j with a parameter that evolves gradually from 0 to 1 to ensure the smoothness of the generated segment.

Figure 3.12 – Transition from *Walk* to *Run* in TOMAS dataset.



(a) Current position

(b) Translation

(c) Rotation

(d) Interpolation

Figure 3.13 – Recurring 2D alignment for motion segment blending

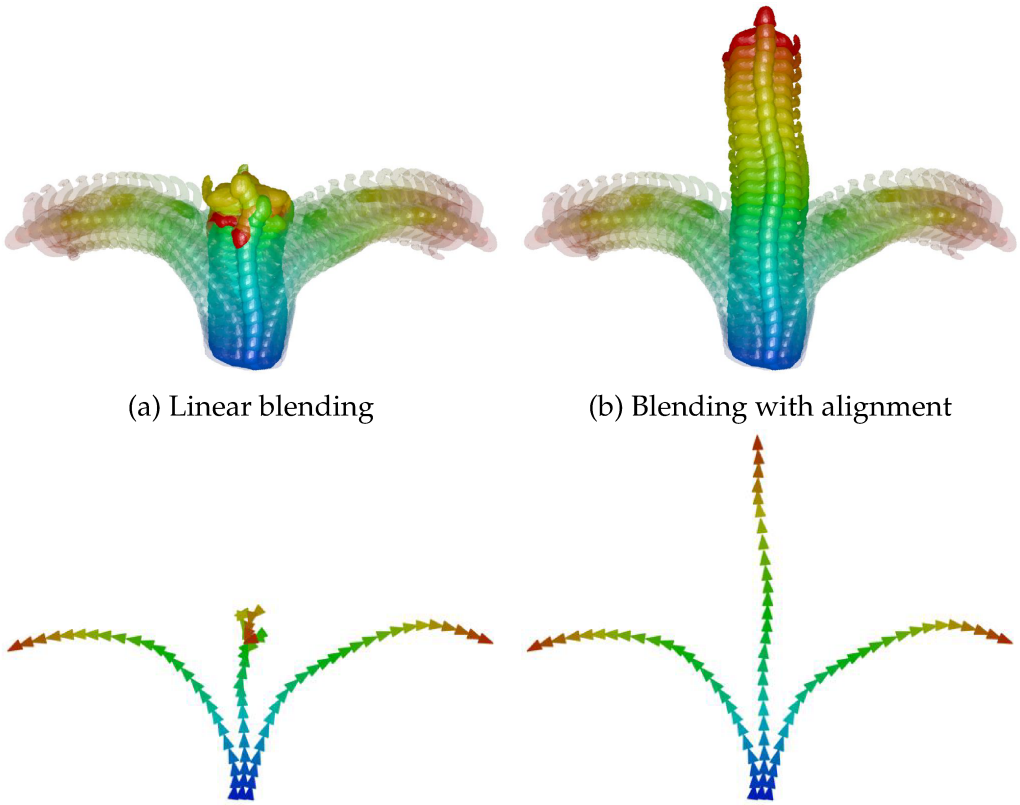
The process of motion segment blending that we use is described in Algorithm 2. The function detailed in this algorithm, that we call Blend(),

Data: Source segment $[\mathcal{M}_i, \mathcal{M}_{i+l^i-1}]$, Destination segment $[\mathcal{M}_{j-l^j+1}, \mathcal{M}_j]$, Source warp w^i, Destination warp w^j, Transition duration L.
Result: Output segment S_{out}
Current mesh center of mass $P \coloneqq \bar{\mathcal{M}}_i$;
for $t \in [\![1, L]\!]$ **do**
  Current source frame $n \coloneqq {w^i}^{-1}(t)$;
  Current destination frame $m \coloneqq {w^j}^{-1}(t)$;
  $\lambda \coloneqq (t-1)/(L-1)$;
  $\theta \coloneqq \text{Align}(\mathcal{M}_n, \mathcal{M}_m)$;
  $\theta_n \coloneqq -\lambda\theta$;
  $\theta_m \coloneqq (1-\lambda)\theta$;
  $T_n \coloneqq P - \bar{\mathcal{M}}_n$;
  $T_m \coloneqq P - \bar{\mathcal{M}}_m$;
  $\text{Move}([\mathcal{M}_n, \mathcal{M}_{i+l^i-1}], \theta_n, \bar{\mathcal{M}}_n, T_n)$;
  $\text{Move}([\mathcal{M}_m, \mathcal{M}_j], \theta_m, \bar{\mathcal{M}}_m, T_m)$;
  $\mathcal{M}_t \coloneqq \text{Interpolate}(\mathcal{M}_n, \mathcal{M}_m, \lambda)$;
  $P \coloneqq P + (1-\lambda)(\bar{\mathcal{M}}_{n+1} - \bar{\mathcal{M}}_n) + \lambda(\bar{\mathcal{M}}_{m+1} - \bar{\mathcal{M}}_m)$;
  $S_{out} \coloneqq [S_{out}, \mathcal{M}_t]$;
end

Algorithm 2: function Blend()

takes as input source and destination mesh segments with their respective alignment temporal warps, and outputs the interpolated transition mesh segment. In this algorithm, function Move(S, θ, C, T) moves sequence of meshes S by a rotation around the perpendicular axis to the motion plane, where θ is the rotation angle and C the rotation center. It also applies a translation T to the mesh sequence. Function Align$(\mathcal{M}_1, \mathcal{M}_2)$ finds the angle by which mesh $\mathcal{M}_2$ needs to be rotated around the perpendicular axis to the motion plane to align with mesh $\mathcal{M}_1$ using equation 2.3. Function Interpolate$(\mathcal{M}_1, \mathcal{M}_2, \lambda)$ interpolates meshes $\mathcal{M}_1$ and $\mathcal{M}_2$ non-linearly with interpolation parameter λ as explained in section 2.4.3.

At each step of the process, illustrated also in figure 3.13, the current source and destination frame indices n and m are obtained through inverse source and destination time warps w^i and w^j. Next, we move the current remaining of source segment $[\mathcal{M}_n, \mathcal{M}_{i+l^i-1}]$ and destination segment $[\mathcal{M}_m, \mathcal{M}_j]$ with planar translations T_n and T_m towards P the current position of mesh center of mass (figure 3.13b). We also align them with respect to the their first frames $\mathcal{M}_n$ and $\mathcal{M}_m$ with rotations θ_n and θ_m respectively around the vertical axis (figure 3.13c). After that, current



(a) Linear blending (b) Blending with alignment

(c) Linear blending (d) Blending with alignment

Figure 3.14 – Blending sequences *Left* and *Right* from TOMAS dataset

meshes $\mathcal{M}_n$ and $\mathcal{M}_m$ are interpolated non-linearly, and the current source and destination displacements of center of mass are also interpolated, thus giving the next mesh center of mass position (figure 3.13d).

In practice, our motion segment blending algorithm differs from standard linear blending with its recurring 2D rigid alignment step of current source and destination segments, which is somehow similar to coordinate alignment in the work of Kovar and Gleicher [2003] on registration curves for motion capture data. Figure 3.14 illustrates the impact of this process compared to linear blending. We blend sequences *Right* and *Left* from TOMAS dataset with a constant interpolation parameter $\lambda = 0.5$. Figures 3.14a and 3.14c show the resulting mesh sequence and its 2D center of mass trajectory projected on the motion plane for our algorithm 2. Figures 3.14b and 3.14d show the same results using linear blending. Our algorithm prevents the trajectory of center of mass of the synthesized sequence from collapsing or reversing path. This allows us to synthesise

natural looking motion transitions without major visual flaws and unnatural character global trajectories, regardless of the nature of the motions to be blended.

3.4.2 Motion segment concatenation

Finally, Algorithm 3 details how motion sequences are generated from a general walk in the essential subgraph. The final sequence is broadly a concatenation of original and synthetic motion segments. In order to insure spatial continuity of the resulting motion stream, we additionally perform 2D rigid alignment at the junction between two consecutive motion segments, in the form of translations in the motion plane and rotations around the perpendicular axis to it.

Data: Graph walk path $p \coloneqq [n_1, \dots, n_N]$
Result: Output mesh sequence S_{out}
current rotation angle θ_c, current rotation center C_c, current translation T_c **for** $i \in [\![1, N-1]\!]$ **do**
  if $[\![n_i, n_{i+1}]\!]$ *is an original segment* **then**
    Move($\left[\mathcal{M}_{n_i}, \mathcal{M}_{n_{i+1}}\right], \theta_c, C_c, T_c$);
    $S_{out} \coloneqq \left[S_{out}, \left[\mathcal{M}_{n_i}, \mathcal{M}_{n_{i+1}}\right[\right]$;
  end
  else
    Move($[\mathcal{M}_{n_i}, \mathcal{M}_{n_i + l^{n_i} - 1}], \theta_c, C_c, T_c$);
    $S \coloneqq$ Blend($[\mathcal{M}_{n_i}, \mathcal{M}_{n_i + l^{n_i} - 1}], \left[\mathcal{M}_{n_{i+1} - l^{n_{i+1}} + 1}, \mathcal{M}_{n_{i+1}}\right]$);
    $S_{out} \coloneqq [S_{out}, S(1 : \text{end} - 1)]$;
    $\theta_c \coloneqq$ Align($S(\text{end}), \mathcal{M}_{n_{i+1}}$);
    $C_c \coloneqq \bar{\mathcal{M}}_{n_{i+1}}$;
    $T_c \coloneqq \bar{S}(\text{end}) - \bar{\mathcal{M}}_{n_{i+1}}$;
  end
end
Move($\mathcal{M}_{n_N}, \theta_c, C_c, T_c$);
$S_{out} \coloneqq [S_{out}, \mathcal{M}_{n_N}]$;

Algorithm 3: Algorithm to generate a motion sequence from a graph walk

3.5 High-level constraints

As generating random graph walks is not of much interest in terms of applications, we consider in this part constrained navigation in the

essential graph with various user-specified constraints such as spatial, temporal and behavioural constraints. We cast motion extraction as a graph search problem, that consists of finding the walk $p = [n_1, n_2, ..., n_N]$ that minimizes a total error cost $J_c(p)$ defined as follows:

$$J_c(p) = J_c([n_1, n_2, ..., n_N]) = \sum_{i \in [\![1,N]\!]} j_c([n_1, ..., n_{i-1}], n_i) \tag{3.5}$$

Where $j_c([n_1, ..., n_{i-1}], n_i)$ is a scalar function evaluating the additional error of appending node n_i to the graph walk $[n_1, ..., n_{i-1}]$ with respect to the user specification. Since a graph node might be visited more than once in this search, the user also needs to specify a halting condition to prevent the search from recursing infinitely.

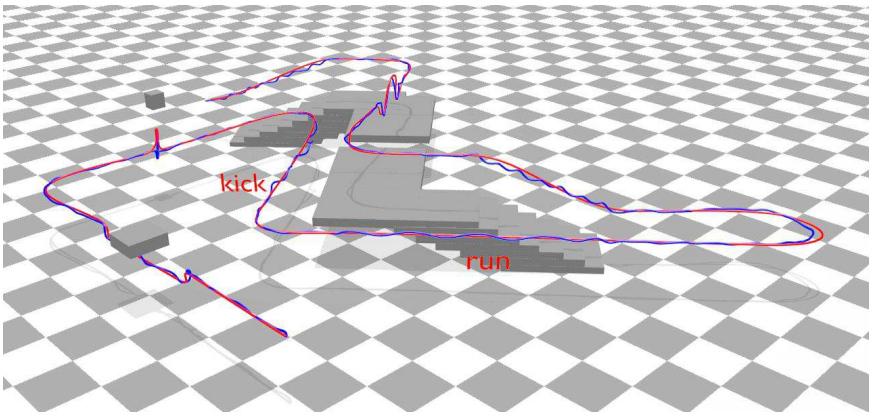
In order to demonstrate the interest of essential graphs for animation purposes, we implement two scenarios with different types of constraints. In both scenarios, the graph search was solved using Depth-First algorithm with Branch-and-Bound to reduce the search space. When both the start and end nodes are known as in the dance sequence editing scenario in section 3.5.2, we use bi-directional searh for a faster graph search.

3.5.1 Behavioral 3D path synthesis

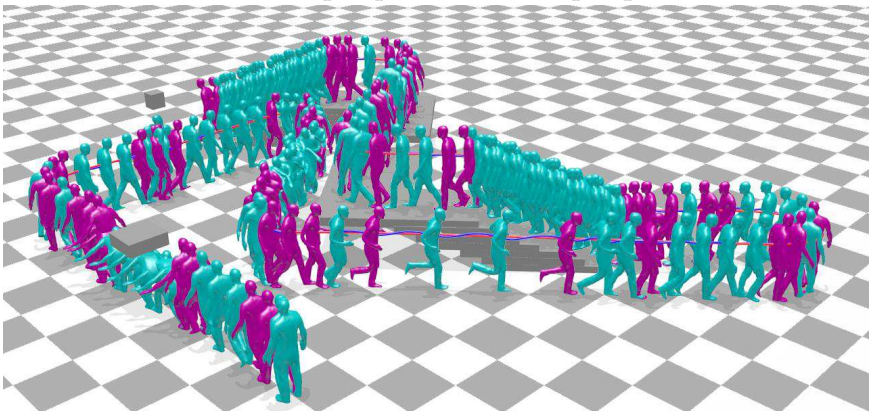
Given a set of motion sequences of the same character, the user provides a 3D curve that the characters center of mass should follow as closely as possible. Additionally, some parts of this path may impose specific types of motion taken from the dataset. Other than the character's mesh center of mass, the targeted element could be any vertex of the mesh, or the center of mass of any subgroup of vertices of the mesh such as a specific body part. This kind of constraints better adapts to datasets with a variety of motions involving locomotion activities, and for which there are significant displacements of the character in space. This is the case of our dataset TOMAS.

In this example scenario, and for a currently added node to the current generated path, cost function jc is obtained by first evaluating the length of the currently generated path after appending the current node to it; We subsequently find the 3D point in the input curve with the same arc length to the origin; The value of jc is then the distance between this point and the center of mass of the mesh added to the current generated graph walk. The halting condition is when the length of the travelled path exceeds the length of the query 3D curve.

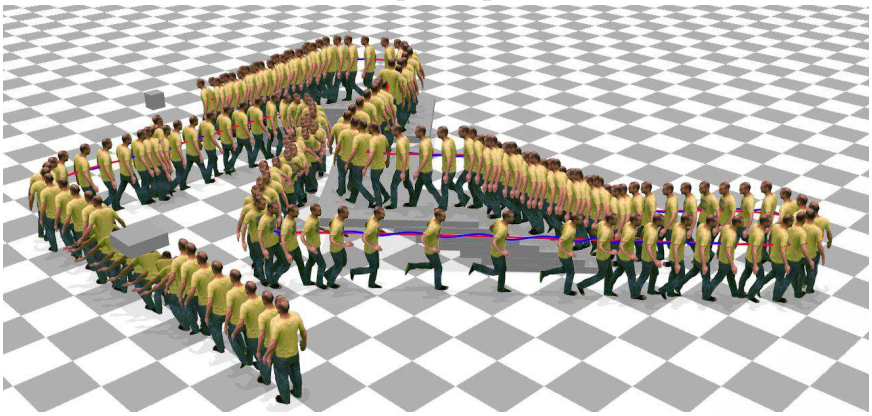
We test this scenario using TOMAS dataset. Figure 3.15 shows an example where the character has to follow a 3D curve provided by the



(a) Red: Input path, Blue: Output path

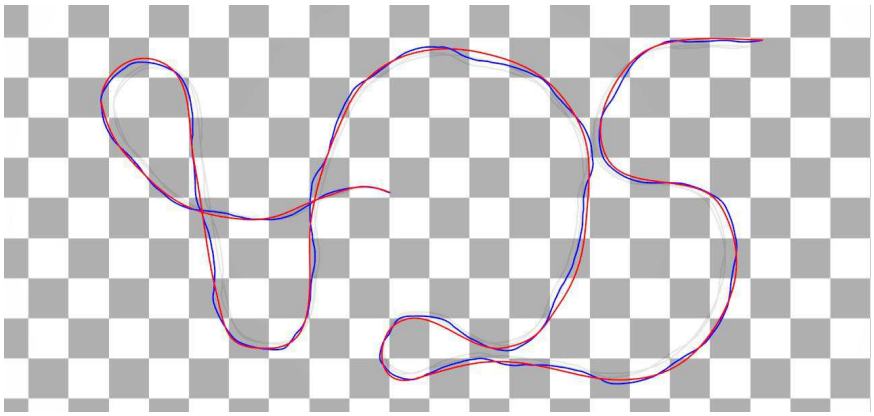


(b) Output sequence

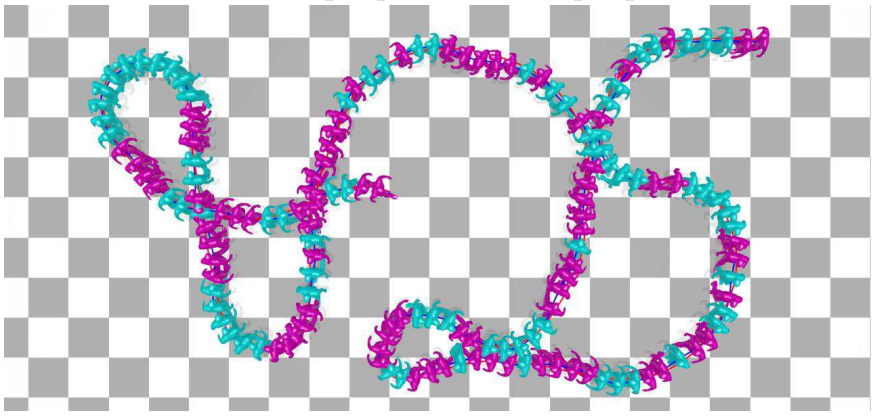


(c) Textured output sequence

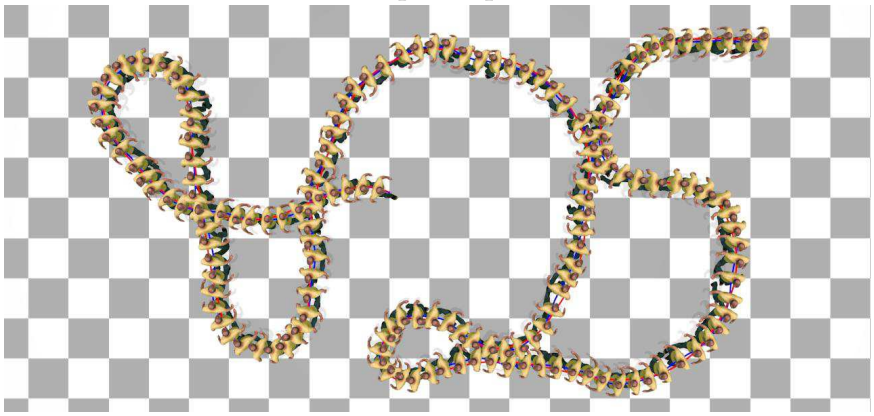
Figure 3.15 – 3D path synthesis. Green frames are original, Red frames are interpolated.



(a) Red: Input path, Blue: Output path



(b) Output sequence



(c) Textured output sequence

Figure 3.16 – 2D path synthesis. Green frames are original, Red frames are interpolated.

user. This user guidance allows the character to duck under an obstacle, jump over a gap, reach a target with its head and climb stairs up and down. For some segments of the curve, the user also requires that the character performs kicking and running motions exclusively. In this example, green output frames are original ones, and red frames are interpolated ones. We also show the output sequence with textured meshes.

Figure 3.16 shows an other example where we only require the projection of the center of mass of the character to follow a 2D path in the motion plane. In this example, only purely locomotion sequences of TOMAS dataset were involved in the graph search, i.e. sequences *Walk*, *Right*, *Right Sharp*, *Left* and *Left Sharp*. We pick a smaller value for the α parameter when building the essential graph for this case in order to densify the final graph. More edges in the graph means more possible transitions and hence a better ability to fit the 2D path accurately. However, this comes at the expense of realism as the surface deformation cost has less priority in the transition selection process. In this example also, green output frames are original ones, and red frames are interpolated ones. We also show the output sequence with textured meshes.

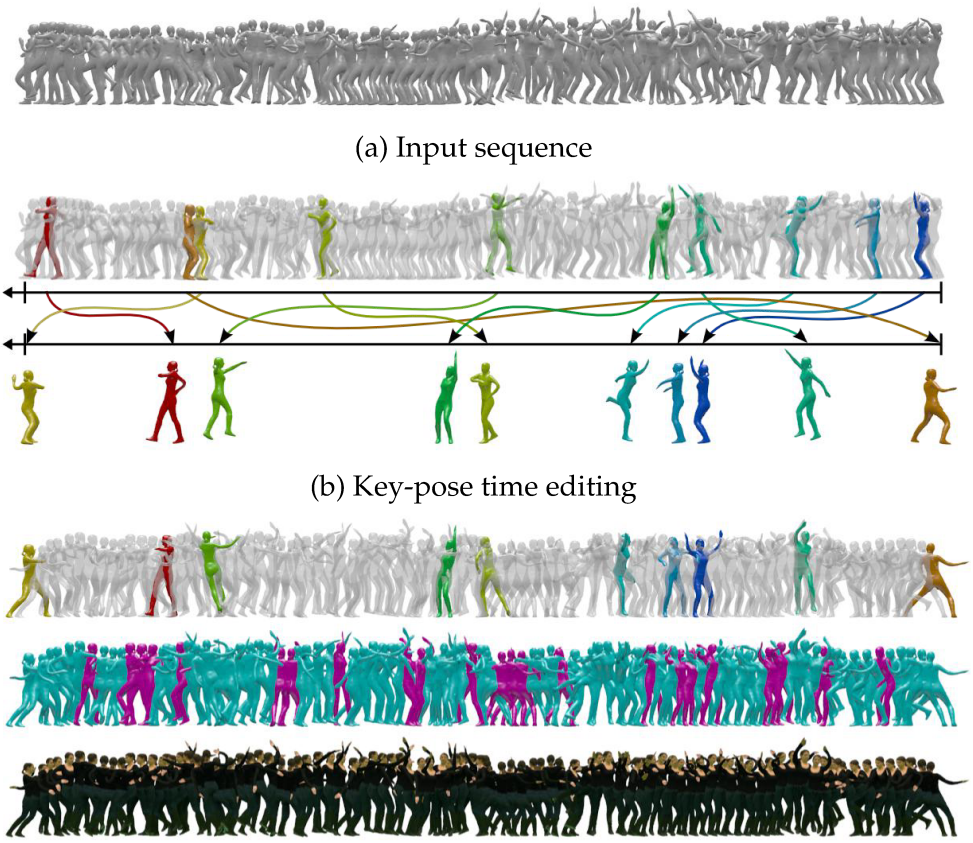
3.5.2 Pose/Time constraint

In this example scenario, and given an input motion sequence, the user selects a set of key-poses and provides new times of occurrence for each one of them. The graph search generates the motion sequence that respects as closely as possible these pose/time restrictions. This type of constraint is more adapted to sequences of random unplanned activities, such as dancing in our dataset *Caty*.

This time, the search is solved for every two successive query poses separately, and cost function j_c measures the duration cost of adding a node to the current path. The halting condition is when the duration of the travelled path between start and end key-poses exceeds the desired duration.

In Figure 3.17, we show an example where we apply this scenario to synthesize a new variant of the dance sequence in CATY dataset, where certain key-poses appear at different timings from their original occurrence times in the input sequence. In the same figure, green output frames are original ones, and red frames are interpolated ones. We also show the output mesh sequence with textures.

Figure 3.18 shows the numerical results of this experiment. In the top row, we show the original occurrence times of the selected key-poses. In the second row we show the desired times of occurrence of the same



(a) Input sequence

(b) Key-pose time editing

(c) Output sequence

Figure 3.17 – Pose/time editing. Green frames are original, Red frames are interpolated.

Input	1480	464	21	119	275	1212	541	864	1762	1459
Output	0	250	460	510	610	890	970	1420	1520	1800
Result	0	258	460	514	617	886	970	1433	1530	1797

Figure 3.18 – CATY dance sequence editing results.

key-poses in the output sequence. Finally in the last row, we show the real times of occurrence of these key-frames in the output sequence that we obtained with the minimal cost walk. Notice that for the most part, the obtained times of occurrence are very close and sometimes equal to the desired times. The timing errors are not propagated as graph search is done sequentially for every successive pair of start and end poses, and the

timing error in the previous solved pair is taken into account in the next pair search. As we show also in this video, the resulting dance sequence nearly matches the input sequence in realism and the visual results are quite convincing.

3.6 Conclusion

We contributed in this chapter a new organizing structure for animation synthesis, the essential graph, that proves to be more optimal than standard motion graphs, a widely used alternative in the literature. However, this improvement comes with additional computational complexity that becomes intractable for large motion datasets. Since, we work with mesh data which is more intricate and harder to interpolate compared to skeletal data, our primary concern was the optimality of the selected transitions in the animation graph with respect to the realism criterion combining surface deformation and duration costs.

Chapter 4

Shape appearance representation

Contents

4.1 Introduction

In this chapter, we propose a view-independent appearance representation and estimation algorithm, to encode the appearance variability of a dynamic subject, observed over one or several temporal sequences. Compactly representing image data from all frames and viewpoints of the subject can be seen as a non-linear dimensionality reduction problem in image space, where the main non-linearities are due to the underlying scene geometry. Our strategy is to remove these non-linearities with

state-of-the-art geometric and image-space alignment techniques, so as to reduce the problem to a single texture space, where the remaining image variabilities can be straightforwardly identified with linear PCA and thus encoded as Eigen texture combinations. To this goal, we identify two geometric alignment steps, as shown in figure 4.1:

- First, we coarsely register geometric shape models of all time frames to a single shape template, for which we pre-computed a single reference surface-to-texture unwrapping.
- Second, to cope with remaining fine-scale misalignments due to registration errors (figure 4.2), we estimate realignment warps in the texture domain.

Because they encode low-magnitude residual geometric variations, the realignment warps are also advantageously decomposed using linear PCA, yielding Eigen warps. The full appearance information of all subject sequences can then be compactly stored as linear combinations of Eigen textures and Eigen warps. Our strategy can be seen as a generalization of the popular work of Nishino *et. al.* Nishino et al. [2001], which introduces Eigen textures to encode appearance variations of a static object under varying viewing conditions, to the case of fully dynamic subjects with several viewpoints and motions.

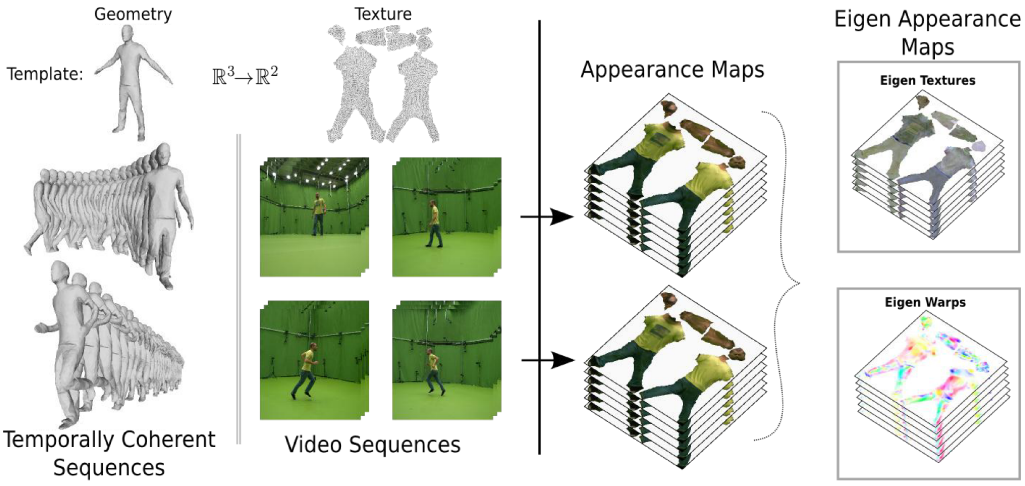


Figure 4.1 – Appearance representation of a dynamic subject from multiple views. Left: 3D Coarse geometric registration. Right: 2D realignment in texture domain.

The pipeline is shown to yield effective estimation performance. In addition, the learned texture and warp manifolds allow for efficient generalizations, such as texture interpolations to generate new unobserved

content from blended input sequences, or completions to cope with missing observations due to *e.g.* occlusions.

4.2 Related work

Obtaining appearance of 3D models from images was first tackled from static images for inanimate objects, e.g. Debevec et al. [1996], Nishino et al. [2001], a case largely explored since e.g. Lempitsky and Ivanov [2007], Waechter et al. [2014]. The task also gained interest for the case of subjects in motion, e.g. for human faces Blanz and Vetter [1999]. With the advent of full body capture and 3D interaction systems Carranza et al. [2003], Collet et al. [2015a] the task of recovering appearance has become a key issue, as the appearance vastly enhances the quality of restitution of acquired 3D models.

A central aspect of the problem is how to represent appearance, while achieving a proper trade-off between storage size and quality. 3D capture traditionally generates full 3D reconstructions, albeit of inconsistent topology across time. In this context the natural solution is to build a representation per time frame which uses or maps to that instant's 3D model. Such per instant representations come in two main forms. View-dependent texturing stores and resamples from each initial video frame Zitnick et al. [2004], eventually with additional alignments to avoid ghosting effects Eisemann et al. [2008]. This strategy creates high quality restitutions managing visibility issues on the fly, but is memory costly as it requires storing all images from all viewpoints. On the other hand, one can compute a single appearance texture map from the input views in an offline process Collet et al. [2015a], reducing storage but potentially introducing sampling artefacts. These involve evaluating camera visibility and surface viewing angles to patch and blend the view contributions in a single common mapping space. To overcome the resolution and sampling limitations, 3D superresolution techniques have been devised that leverage the viewpoint multiplicity to build such maps with enhanced density and quality Tung [2008], Tsiminaki et al. [2014], Goldlücke et al. [2014].

In recent years, a leap has been made in the representation of 3D surfaces captured, as they can now be estimated as a deformed surface of time-coherent topology de Aguiar et al. [2008], Cagniart et al. [2010]. This in turns allows any surface unwrapping and mapping to be consistently propagated in time, however in practice existing methods have only started leveraging this aspect. Tsiminaki et al. [2014] examines small temporal segments for single texture resolution enhancement. Volino et al.

[2014] uses a view-based multi-layer texture map representation to favour view-dependant dynamic appearance, using some adjacent neighbouring frames. Collet et al. [2015a] use tracked surfaces over small segments to improve compression rates of mesh and texture sequences. Methods are intrinsically limited in considering longer segments because significant temporal variability then appears due to light change and movement.

While global geometry consistency has been studied Boukhayma and Boyer [2015], Casas et al. [2012], most such works were primarily aimed at animation synthesis using mesh data. the work of Casas et al. [2014b] is the first to solve for appearance synthesis for parametric motion graphs as-well. They propose an image-based pair wise alignment between pre-selected frames within parametric motion spaces, and this approach is extended in Casas et al. [2015] through alignment pre-computation and storing for efficient real-time rendering. However, none of these works propose a global appearance model for sequences. In contrast, we propose an analysis and representation spanning full sequences and multiples sequences of a subject.

For this purpose, we build an Eigen texture and appearance representation that extends concepts initially explored for faces and static objects Turk and Pentland [1991b], Blanz and Vetter [1999], Cootes et al. [2001], Nishino et al. [2001]. Eigenfaces Turk and Pentland [1991b] were initially used to represent the face variability of a population for recognition purposes. The concept was broadened to built a 3D generative model of human faces both in the geometry and texture domains, using the fact that the appearance and geometry of faces are well suited to learning their variability as linear subspaces Blanz and Vetter [1999]. Cootes et al. [2001] perform the linear PCA analysis of appearance and geometry landmarks jointly in their active appearance model. Nishino et al. [2001] instead use such linear subspaces to encode the appearance variability of static objects under light and viewpoint changes at the polygon level. We use linear subspaces for full body appearance and over multiple sequences. Because the linear assumption doesn't hold for whole body pose variation, we use state of the art tracking techniques Allain et al. [2015] to remove the non-linear pose component by aligning a single subject-specific template to all the subject's sequence. This in turn allows to model the appearance in a single mapping space associated to the subject template, where small geometric variations and appearances changes can then be linearly modelled.

4.3 Texture maps variation

After a first examination of temporal sequences of texture maps of the same shape undergoing geometrical change, we notice that there is a great deal of information redundancy that could be factored to optimize storing and reusing such data. But we also notice a substantial amount of temporal variation in the appearance. With further inspection, we can describe the temporal change occurring in the texture map as an underlying reference texture, with inherent properties, undergoing temporal change in its pixel intensities, while being subject to a non-rigid displacement field. Hence, we break down texture variation into two components that we assume to be independent:

- Time varying surface-borne spatial displacement field.
- Pixel intensity variation over time.

Figure 4.2 – Captured (Green) versus Tracked (Red) geometries, sequence *Run*, TOMAS dataset.

The spatial displacement can be seen as a motion field that is tangential to the geometry, leading to non-rigid appearance drift. We argue that it accounts for the following elements introduced by our 4D textured model reconstruction pipeline:

- Geometric measurement errors introduced by calibration and static surface reconstruction imprecisions.
- Our template-based temporal surface tracking favours geometric over photo-metric consistency. Consequently, as we can see in figure 4.2, the resulting tracked geometry fails sometimes to capture small non-rigid details such as garment movements, facial expressions and hair change for human subjects.

Some of the factors behind temporal variation of pixel values are:

- Illumination changes that are due to change in the global position, orientation and pose of the subject.

- Viewpoint change, as our capture system combines visual information from 68 different views for instance.
- Blur in highly dynamic motions.

Provided that we could estimate the misalignment between texture maps accurately, we can then represent the aligned texture maps in a linear compact subspace that factors all types of intensity variation mentioned above. For more dimensional reduction, the displacement fields can also be represented in a compact subspace of their own. Such globally compact and simple representation can allow for a wide range of applications, some of which we present in section 4.6.

4.4 Eigen appearance maps

Our approach considers as input the texture maps of a subject over different motion sequences and as generated by a multi-view acquisition system. The approach is in principle agnostic in its construction to the particular technique used to generate these textures but still benefits from an optimal exploitation of the original image information. For the reduction of these subject specific textures, elaborate non-linear schemes from scratch might be used but would be burdened to rediscover non-linear variations resulting from the underlying geometry of the scene, for which good estimates are already available using state of the art techniques. Instead, we leverage these variations as compensation estimates in our method to eliminate the main non-linear components: the viewpoint to texture-space mapping on one hand, which is governed by the 3D tracking and initial per-frame texture-space estimation; the common texture-space mapping to reduce all textures to one consistent appearance layout on the other hand, which is compensated by the geometric warps. In practice, we proceed as follows:

- Texture deformation fields that map input textures to, and from, their aligned versions are estimated using optical flows. Given the deformation fields, Poisson reconstruction is used to warp textures.
- PCA is applied to the aligned maps and to the texture warps to generate the Eigen textures and the Eigen warps that encode the appearance variations due to, respectively, viewpoint, illumination, and geometric inaccuracies in the reference model.

Hence, The main modes of variation of aligned textures and deformation fields, namely Eigen textures and Eigen warps respectively, span the appearance space in our representation. The main steps of this method are depicted in Figure 4.3 and detailed in the following sections.

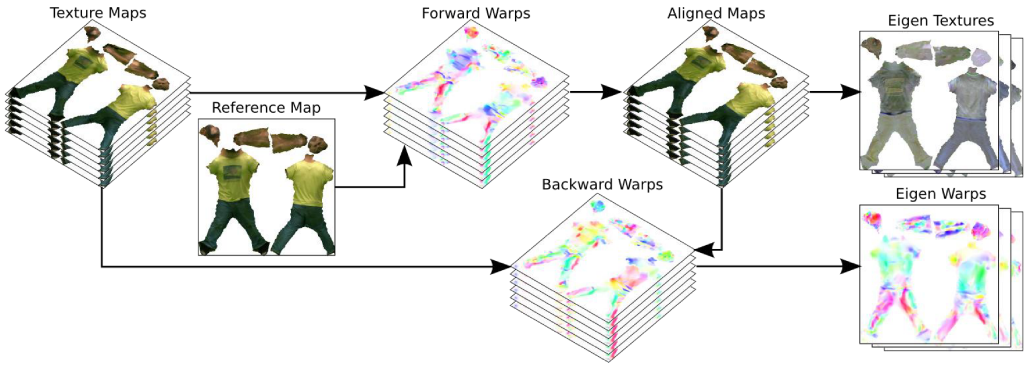


Figure 4.3 – Method pipeline from input textures (left) to eigen maps (right).

Note that due to texture space discretization, the warps between textures are not one-to-one and, in practice, two separate sets of warps are estimated. Forward warps map the original texture maps to the reference map. Backward warps map the aligned texture maps back to the corresponding input textures (see Figure 4.3).

4.4.1 Texture maps alignment

Appearance variations that are due to viewpoint and illumination changes are captured through PCA under linearity assumption for these variations. To this purpose, textures are first aligned in order to reduce geometric errors resulting from calibration, reconstruction and tracking imprecisions. Such alignment is performed using optical flow, as described below, and with respect to a reference map taken from the input textures. An exhaustive search of the best reference map with the least total alignment error over all input textures is prohibitive since it requires N^2 alignments given N input textures. We follow instead a medoid shift strategy over the alignment errors.

The alignment algorithm (see Algorithm 4) first initializes the reference map as one texture from the input set. All texture maps are then aligned to this reference map, and the alignment error is computed as the cumulative sum of squared pixel differences between the reference and the aligned texture maps. The medoid over the aligned texture maps, with respect to alignment error, then identifies the new reference map. These two steps, alignment and medoid shift, are iterated until the total alignment error

stops decreasing.

Data: Texture maps $\{I_k\}_{k\in[\![1..F]\!]}$
Result: Reference map A_{ref}, aligned textures A_k
A_{ref}, e_0 initializations;
while $e_i < e_{i-1}$ **do**
  Compute alignment warps: $\{w_k\}_{k\in[\![1..N]\!]}$ s.t. $A_{ref} \approx I_k(x + w_k)$;
  Align texture maps: $A_k = I_k(x + w_k)$;
  Update alignment error: $e_i = \sum_k \|A_k - A_{ref}\|^2$;
  Set A_{ref} as the texture that gives the medoid of the aligned textures:
    $A_{ref} = I_{k_0}$ s.t. $k_0 = arg\min_k \sum_l \|A_k - A_l\|^2$;
end

Algorithm 4: Texture alignment with iterative reference map selection.

Correspondence with optical flow

The warps $\{w_k\}$ in the alignment algorithm, both forward and backward in practice, are estimated as dense pixel correspondences with an optical flow method Snchez Prez et al. [2013]. Forward warp w aligns texture map I to the reference map A_{ref} such that $A_{ref} \approx I(x + w)$, thus giving the aligned texture map $A = I(x + w)$. Backward warp w maps the aligned texture map A back to the corresponding original input texture I such that $I \approx A(x + w)$. We mention here that the optical flow assumptions: brightness consistency, spatial coherency and temporal persistence, are not necessarily verified by the input textures. In particular, the brightness consistency does not hold if we assume appearance variations with respect to viewpoint and illumination changes. To cope with this in the flow estimation, we use histogram equalization as a preprocessing step, which presents the benefit of enhancing contrast and edges within images. Additionally, local changes in intensities are reduced using bilateral filtering, which smooths low spatial-frequency details while preserving edges. Figure 4.4 shows the result of performing histogram equalization followed by bilateral filtering on each color channel of the first texture frame from sequence *Walk* in TOMAS dataset. For the bilateral filter, we set the diameter of each pixel neighbourhood that is used during filtering to 80, the filter sigma in the color space to 15 and the filter sigma in the coordinate space to 80.

(a) Original texture map (b) Histogram equalization (c) Bilateral filtering

Figure 4.4 – Texture pre-alignment processing.

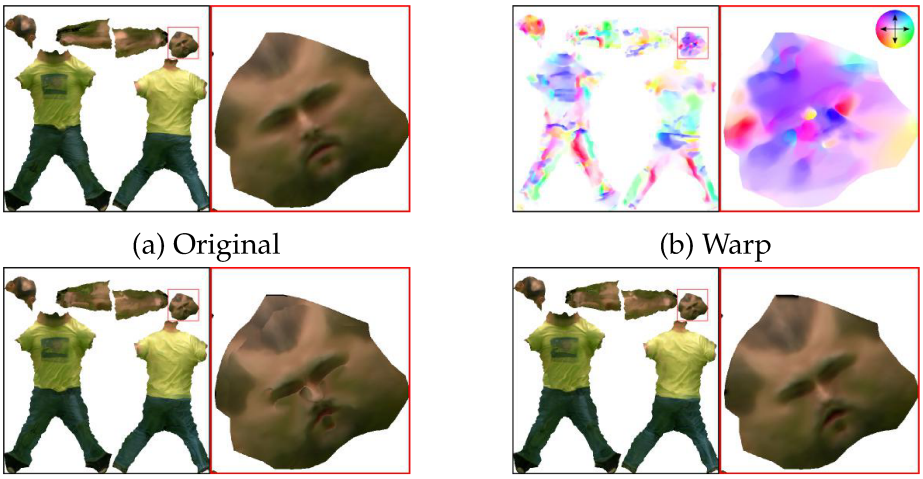
Texture warping

Optical flows give dense correspondences $\{w\}$ between the reference map and the input textures. To estimate the aligned textures $\{A\}$, we cast the problem as an optimization that seeks the texture map which, once moved according to w, best aligns with the considered input texture both in the color and gradient domains. Our experiments show that solving over both color and gradient domains significantly improves results as it tends to better preserve edges than with colors only. This is also demonstrated in works that use the Poisson equation for image and video editing and synthesis. In works aiming at composition or cloning using images or video frames Pérez et al. [2003] Jia et al. [2006] Wang et al. [2004a] Chen et al. [2013], the input image gradients remain spatially static. When the target application is image interpolation Linz et al. [2010] Mahajan et al. [2009], it is necessary to move or/and interpolate the input gradients prior to reconstruction. All of these previous cases amount to solving a 2D or 3D poisson system with well defined boundary conditions. In our case, we want to warp images instead of interpolating them. Consequently, we also need to move the input gradients, but we don't have boundary conditions as in the interpolation case. To overcome this issue, we reconstruct the warped images from the warped gradients, while using the warped intensities as soft constraints.

We are given an input texture map I, a dense flow w from A_{ref} to I, and the gradient image ∇I. The aligned texture A of I with respect to A_{ref} is then the map that minimizes the following term:

$$E(A) = \Sigma_x \|\nabla^2 A(x) - \overrightarrow{\nabla}.\nabla I(x+w)\|^2 + \lambda \|A(x) - I(x+w)\|^2, \quad (4.1)$$

where ∇^2 is the Laplacian operator, $\overrightarrow{\nabla}.$ the divergence operator, and

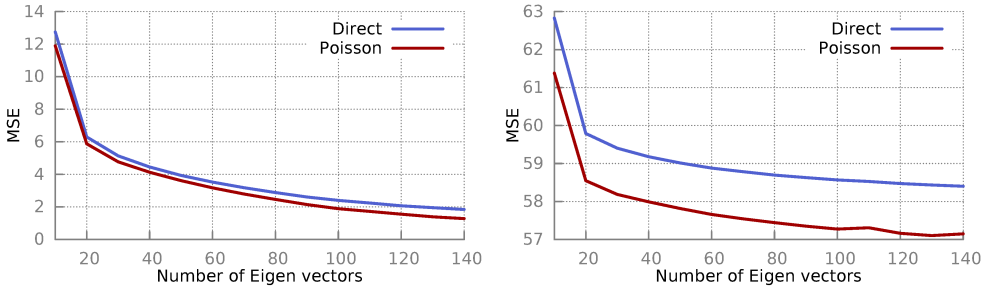


(a) Original (b) Warp

(c) Direct warping (d) Poisson warping

Figure 4.5 – Poisson versus direct texture warping.

x denotes pixel locations in texture maps. The weight λ balances the influence of color and gradient information. In our experiments, we found that the value 0.02 gives the best results with our datasets.



(a) Texture domain (b) Image domain

Figure 4.6 – MSE Reconstruction error for TOMAS using Poisson and direct warping

Using a vector image representation, the above energy can be minimized by solving, in the least-squares sense, the overdetermined $2N \times N$ system below, where N is the active region size of texture maps:

$$\begin{pmatrix} L \\ \Lambda \end{pmatrix} A = \begin{pmatrix} \overrightarrow{\nabla}.\nabla I(x+w) \\ \Lambda I(x+w) \end{pmatrix}, \tag{4.2}$$

where L is the linear Laplacian operator and $\Lambda = diag_N(\lambda)$. A solution for A is easily found by solving the associated normal equations:

$$\left(L^T L + \Lambda^2\right) A = L^T \overrightarrow{\nabla}.\nabla I(x+w) + \Lambda^2 I(x+w). \tag{4.3}$$

Since the Laplacian matrix doesn't have a full rank, soft constraints are needed to improve the system's conditioning and stabilize the factorization of the normal equations.

Poisson warping comes at no run-time computational cost. That is, since the whole dataset shares the same mesh topology and UV domain proxy, the linear Laplacian operator is the same for all examples and thus the factorization of the left hand side of the normal equations is computed only once and used for any element of the dataset.

Figure 4.5 shows an example where a texture map is warped, given a warp field, using both direct pixel remapping and Poisson warping. The latter strategy achieves a more visually compelling alignment that preserves the original edges, unlike direct warping which generates many intensity discontinuities in extreme cases.

In order to visualize the numerical impact of using Poisson texture warping on our pipeline, we plot the mean square reconstruction error for TOMAS dataset in both texture and image domains in Figure 4.6. The details of this evaluation mode are explained later on in section 4.5.1. Texture domain error is obtained by comparing the reconstructed textures to the textures estimated from the original images. Image domain error is obtained by projecting the reconstructed textures into the input views and comparing the resulting images to the original input views. We plot the errors as a function of the number of Eigen appearance maps used to build the subspace where textures are projected and reconstructed. Every 3 Eigen maps account for 2 Eigen textures and 1 Eigen warp. As we can see in the figure, Poisson warping reduces the error introduced by the reconstruction pipeline with respect to direct texture warping in both texture and image domains. We note that texture warping, as illustrated in figure 4.3, occurs twice in the process, first to align texture maps to the reference map and second to de-align them using backward warps, hence the importance of optimizing the warping algorithm.

We believe the overall numerical improvement achieved with Poisson warping comes from three partial improvements:

- The reconstruction error of the process of merely aligning textures than warping them back to their original state is improved.
- Aligned images with poisson warping are smoother, which gives better reconstruction errors in the Eigen texture space.
- The backward warps mapping these smoother aligned textures to the original textures are also smoother, which gives better reconstruction

errors in the Eigen warp space.

4.4.2 Eigen textures and Eigen warps

Once the aligned textures and the warps are estimated, we can proceed with the statistical analysis of appearances. Given the true geometry of shapes and their motions, texture map pixels could be considered as shape appearance samples over time and PCA applied directly to the textures would then capture appearance variability. In practice, incorrect geometry causes distortions in the texture space and textures must be first aligned before any statistical analysis. In turn, de-alignment must be also estimated to map the aligned textures back to their associated input textures (see Figure 4.3). And these backward warps must be part of the appearance model to enable appearance reconstruction. In the following, warps denote the backward warps. Also, we consider vector representations of the aligned texture maps and of the warps. These representations include only pixels that fall inside active regions within texture maps. We perform Principal Component Analysis on the textures and on the warp data separately to find the orthonormal bases that encode the main modes of variation in the texture space and in the warp space independently. We refer to vectors spanning the texture space as Eigen textures, and to vectors spanning the warp space as Eigen warps.

Let us consider first texture maps. Assume N is the dimension of the vectorized representation of active texture elements, and F the total number of frames available for the subject under consideration. To give orders of magnitude for our datasets, $N = 22438995$ and $F = 207$ for the TOMAS dataset, and $N = 25966476$ and $F = 290$ for the CATY dataset that will be presented in the next section. We start by computing the mean image $\bar{A}$, which is a vector of size N, and the centered data matrix M from aligned texture maps $\{A_i\}_{i\in[1..F]}$, of size $N \times F$:

$$\bar{A} = \frac{1}{F}\sum_k A_k \ , \ M = \begin{bmatrix} | & & | \\ A_1 - \bar{A} & \dots & A_F - \bar{A} \\ | & & | \end{bmatrix}. \tag{4.4}$$

Traditionally, the PCA basis for this data is formed by the Eigen vectors of the covariance matrix MM^T, of size $N \times N$, but finding such vectors can easily become prohibitive as a consequence of the texture dimensions. However, it appears that the non zero eigen values of MM^T are equal to the non zero Eigen values of M^TM, of size $(F \times F)$ this time, and that they are at most: $\min(F, N) - 1$. Based on this observation, and

since $F << N$ in our experiments, we solve the characteristic equation $\det(MM^T - \alpha I_N) = 0$ by performing Singular Value Decomposition on the matrix $M^T M$, as explained in Turk and Pentland [1991a]:

$$M^T M = D\Sigma D^T \ , \ D = \begin{bmatrix} | & & | \\ V_1 & \dots & V_F \\ | & & | \end{bmatrix} \tag{4.5}$$

where matrix D of size $F \times F$ contains the $(F-1)$ orthonormal Eigen vectors of $M^T M$, denoted by $\{V_i\}$, which are vectors of size F. Square diagonal matrix $\Sigma = diag(\alpha_i)_{1 \le i \le F}$ contains the Eigen values $\{\alpha_i\}_{1 \le i \le F-1}$. We can then write:

$$M^T M V_i = \alpha_i V_i \ , \ i \in [1..F-1], \tag{4.6}$$

and hence:

$$MM^T \underbrace{MV_i}_{T_i} = \alpha_i \underbrace{MV_i}_{T_i} \ , \ i \in [1..F-1], \tag{4.7}$$

where $\{T_i\}$ are the Eigen vectors of MM^T of size N, and therefore form the orthonormal basis of the aligned texture space after normalization, namely the Eigen textures.

We process the backward de-alignment warps in a similar way to the aligned texture maps. We start by computing the mean warp $\bar{w}$ and the centered data matrix M from de-alignment warps $\{w_i\}_{i \in [1..F]}$:

$$\bar{w} = \frac{1}{F} \sum_k w_k \ , \ M = \begin{bmatrix} | & & | \\ w_1 - \bar{w} & \dots & w_F - \bar{w} \\ | & & | \end{bmatrix}. \tag{4.8}$$

To avoid prohibitive computational cost, we find the non-zero Eigen values and Eigen vectors of matrix $M^T M$ of size $(F \times F)$ instead of matrix MM^T of size $N \times N$ ($F << N$) using Singular Value Decomposition:

$$M^T M = D\Sigma D^T \ , \ D = \begin{bmatrix} | & & | \\ V_1 & \dots & V_F \\ | & & | \end{bmatrix} \tag{4.9}$$

where matrix D contains the $(F-1)$ orthonormal Eigen vectors $\{V_i\}$ of $M^T M$, and $\Sigma = diag(\beta_i)_{1 \le i \le F}$ contains the Eigen values $\{\beta_i\}_{1 \le i \le F-1}$. The Eigen vectors of MM^T, denoted by $\{W_i\}$, can be subsequently found as follows:

$$M^T M V_i = \beta_i V_i \ , \ i \in [1..F-1], \tag{4.10}$$

and hence:

$$M M^T \underbrace{M V_i}_{W_i} = \beta_i \underbrace{M V_i}_{W_i} \ , \ i \in [1..F-1], \tag{4.11}$$

These vectors ($\{W_i\}$) of size N form therefore the orthonormal basis of the warp space after normalization, and are referred to as the Eigen warps.

4.4.3 Texture map instantiation

Given the Eigen textures and the Eigen warps, and as shown in Figure 4.7, a texture map can be generated by first creating an aligned texture map A by linearly combining Eigen textures $\{T_i\}$ and adding the mean texture $\bar{A}$ to this combination. Second, a backward warp w is created by linearly combining Eigen warps $\{W_i\}$ and adding the mean warp $\bar{w}$ to the combination. Finally aligned texture A is de-aligned with warp w using the algorithm detailed in section 4.4.1.

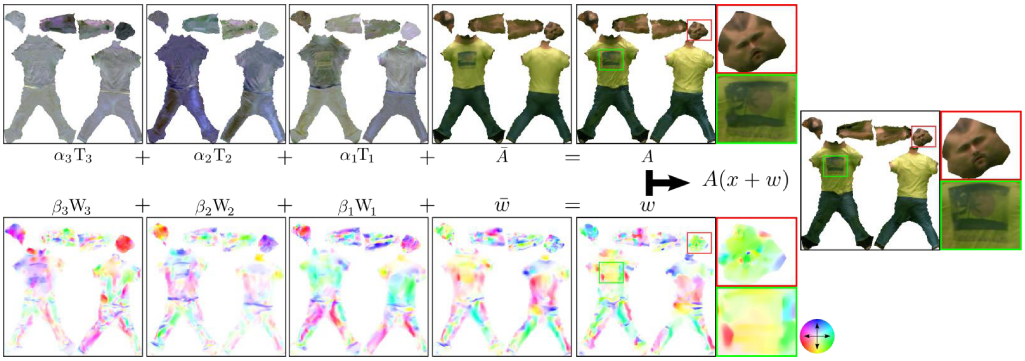


Figure 4.7 – Texture map generation by linear combination.

4.5 Model performance evaluation

To validate the estimation quality of our method, we apply our estimation pipeline to several datasets, project and warp input data using the built Eigen spaces, then evaluate the reconstruction error. To distinguish the different error sources, we evaluate this error both in texture space before projection, and in image domain by projecting into the input views. We compare the reconstructed textures to the textures estimated

from the original images in our pipeline using Tsiminaki et al. [2014], that we consider as ground-truth textures. We also project the reconstructed textures into the input views and compare the resulting images to the original input views, that we consider as ground-truth images. For the image error measurement, we use the 3D model that was fitted to the sequence, as tracked to fit the test frames selected, and render the model as textured with our reconstructed appearance map, using a standard graphics pipeline. In both cases, we experiment with both the structural similarity index (SSIM) Wang et al. [2004b] and the mean square error (MSE) as metrics to compare to the original data. All of our error estimates are computed in the active regions of the texture and image domains. In the texture domain, that means that we only consider the set of texels actually mapped to the 3D model. In the Image domain, it means that only actual silhouette are taken into consideration. Besides, In the case of image domain evaluations, we plot the average error among all viewpoints.

We study in particular the compactness and generalization abilities of our method, by examining the error response as a function of the number of Eigen components kept after constructing the linear subspaces, and the number of training images selected. For all these evaluations, we also provide the results for a naive PCA strategy, where only a set of Eigen appearance maps are built in texture space and used to project and reconstruct textures, to show the performance contribution of including the Eigen warps.

For validation, we used two multi-sequence datasets:

- The TOMAS dataset which consists of 4 different sequences left, right, run and walk with 207 total number of frames and 68 input views each captured at resolution 2048x2048 pixels per frame.
- The CATY dataset: low, close, high and far jumping sequences with 290 total number of frames and 68 input views each captured at resolution 2048x2048 pixels per frame.

The dimension of the vectorized texture maps is computed as the product of the number of occupied pixels multiplied by the number of color channels (RGB in our case). TOMAS dataset is of order 22×10^6 (N = 22.438.995) and in CATY dataset of order 25×10^6 (N = 25 966 476).

4.5.1 Estimation Quality and Compactness

We study the quality and compactness of the estimated representation by plotting the errors of reconstructed texture and image estimates of our method against naive PCA, for the two multi-sequence datasets (Figures 4.9 and 4.10). Note that all texture domain variability could be trivially

represented by retaining as many Eigen textures as there are input images, thus we particularly examine how the quality degrades with the fraction of Eigen components kept. We note that the ability of the model to represent textures with few parameters is highly useful for applications such as data compression or broadcasting.

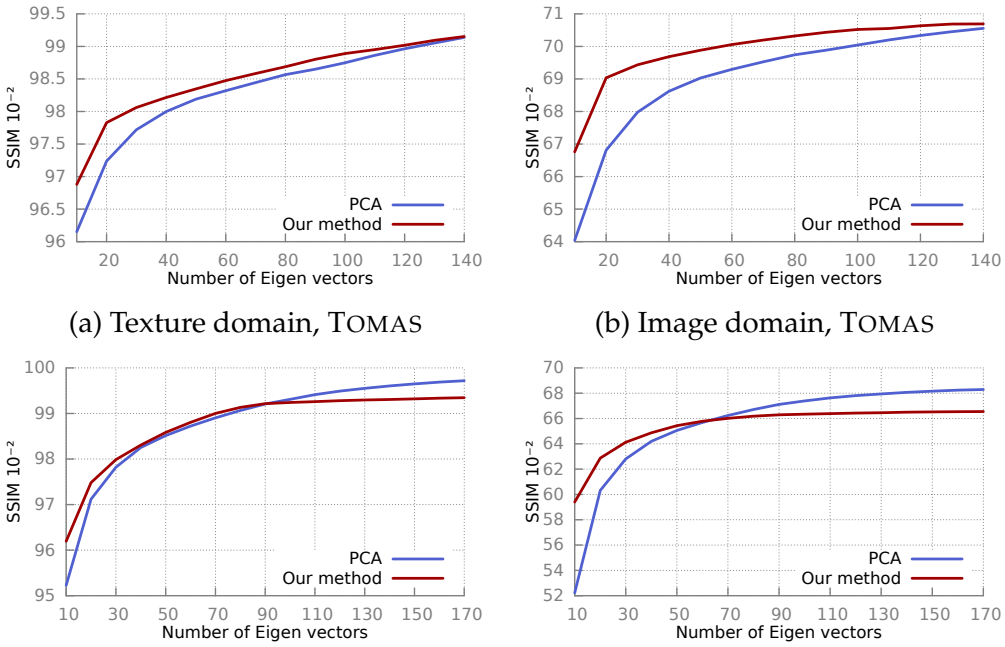
(a) 10 Eigen maps (b) 40 Eigen maps (c) 80 Eigen maps (d) Original texture

Figure 4.8 – Texture reconstruction with our method using various numbers of Eigen maps.

For a fair comparison, we plot the results of reconstructing textures and images with our method using a total number of Eigen components that amounts to the same number of Eigen vectors used for the naice PCA on texture maps. Our Eigen maps include both Eigen textures and Eigen warps with constant proportions in the total number of Eigen components. In our experiments, we found that using two thirds of the Eigen maps for textures and the rest for warps results in nearly optimal performance for our method.

Figure 4.9 shows the structural similarity reconstruction errors in both image and texture domains for CATY and TOMAS datasets. For both datasets, virtually no error (0.98 SSIM) is introduced by our method in the texture domain with as low as 50 components, a substantially low fraction compared to the number of input frames (207 for TOMAS and 290 for CATY). This illustrates the validity of the linear variability hypothesis in texture domain. The error is quite higher in the image domain (bounded by 0.7 in SSIM) for both our method and naive PCA, because measurements are then subject to fixed upstream errors due to geometric alignments, projections and image discretizations.

Nevertheless, visually indistinguishable results are achieved with 50



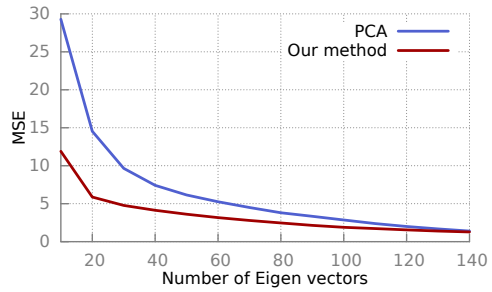
(a) Texture domain, TOMAS (b) Image domain, TOMAS

(c) Texture domain, CATY (d) Image domain, CATY

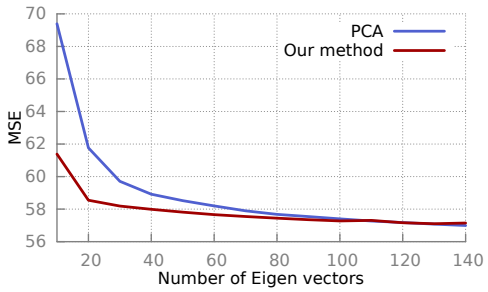
Figure 4.9 – SSIM Reconstruction Error for TOMAS and CATY.

Eigen components (images and warps), with a significant compactness gain. This can be confirmed with the reconstruction results in figure 4.8. In this figure, we show the evolution of a mesh appearance from the *Run* sequence of TOMAS dataset projected than reconstructed through our model, compared to the original texture (Figure 4.8d). Notice how appearance features and wrinkles are moved towards their true positions and get refined along with the increasing reconstruction Eigen subspace dimension (Figures 4.8a, 4.8b and 4.8c).

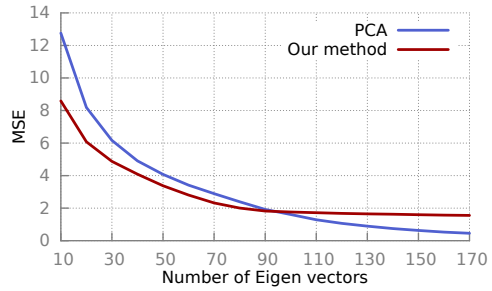
Our method outperforms naive PCA in image and texture domains on both datasets, achieving higher quality with a lower number of Eigen components, and only marginally lower quality as the number of components grows, where the method would be anyway less useful. Higher number of Eigen components marginally favours naive PCA, because naive PCA converges to input textures when increasing the Eigen textures retained by construction. Because of the non bijective behaviour of texture alignment and dealignment in our pipeline, our method hits on the other hand a quality plateau due to small constant errors introduced by texture forward and backward warping. The PCA curve is hence bound to surpass ours at a given number of Eigen components.



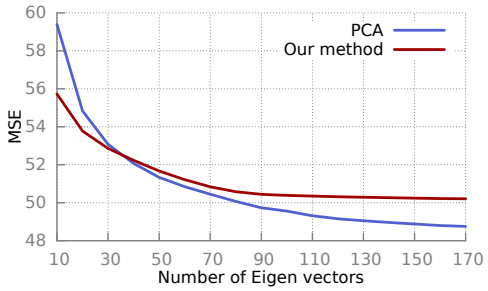
(a) Texture domain, TOMAS



(b) Image domain, TOMAS



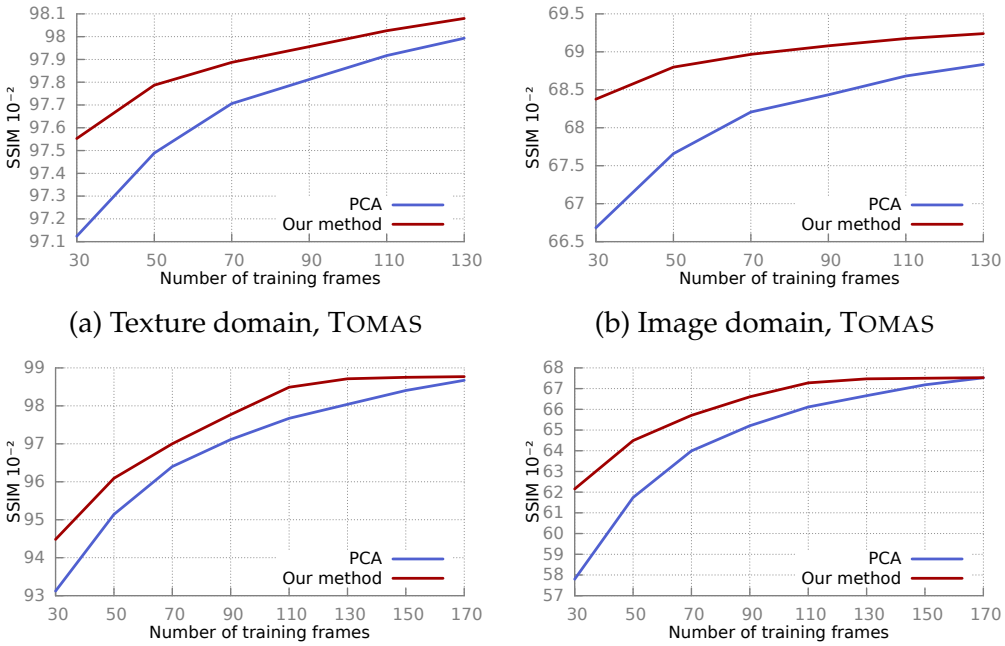
(c) Texture domain, CATY



(d) Image domain, CATY

Figure 4.10 – MSE Reconstruction Error for TOMAS and CATY.

Figure 4.10 shows the mean square reconstruction errors in both image and texture domains for CATY and TOMAS datasets. We notice that the MSE shows similar behaviour to the SSIM measure in both texture and image domains and for both datasets. The only main difference is the point where the naive PCA surpasses ours method. This happens earlier at a smaller subspace dimension for the MSE curves compared to the SSIM ones, which means that this metric favours our method less than the SSIM. We suspect this is due to the fact that, unlike the SSIM, the MSE does not take into account pixel neighbourhood information and hence is very sensitive to the errors introduced with texture warping. However, structural similarity Wang et al. [2004b] was proved to be a reliable measure of visual comeliness and was previously used in similar evaluations involving 3D shape texture map quality assessment Volino et al. [2014]. We hence believe that it is more fit to evaluate the quality and the compactness of our model.



(a) Texture domain, TOMAS (b) Image domain, TOMAS

(c) Texture domain, CATY (d) Image domain, CATY

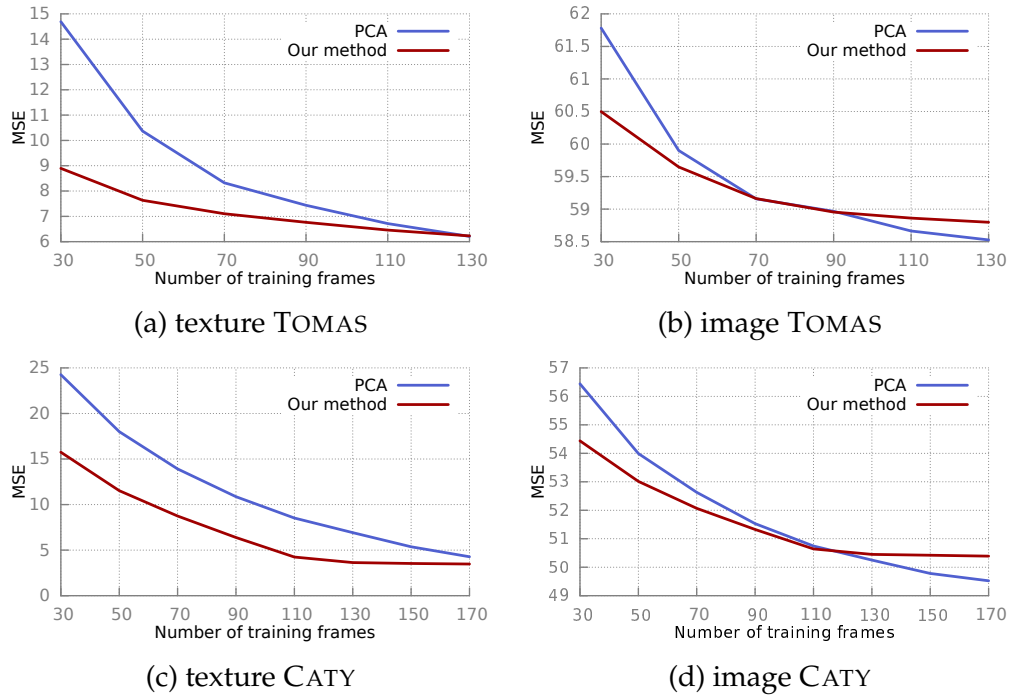
Figure 4.11 – SSIM Generalization Error for TOMAS and CATY

4.5.2 Generalization ability

In the previous section, we examined the performance of the method by constructing an Eigen space with all input frames. We here evaluate the ability of the model to generalize, *i.e.* how well the method reconstructs textures from input frames under a reduced number of examples that don't span the whole input set. We note that the ability of the model to describe instances outside its training set can be exploited in various applications such as data completion (see section 4.6.2).

For this purpose, we perform an experiment using a varying size training set, and a test set from frames not in the training set. We use a training set comprised of randomly selected frames spanning 0% to 60% of the total number of frames, among all sequences and frames of all datasets, and plot the error of projecting the complement frames on the corresponding Eigen space (Figures 4.11 and 4.12).

Figure 4.11 shows the structural similarity generalization errors in both image and texture domains for CATY and TOMAS datasets. These results shows that our representation produces a better generalization than naive PCA, *i.e.* less training frames need to be used to reconstruct a texture and reprojections of equivalent quality. For TOMAS dataset, one can



(a) texture TOMAS (b) image TOMAS

(c) texture CATY (d) image CATY

Figure 4.12 – MSE Generalization Error for TOMAS and CATY

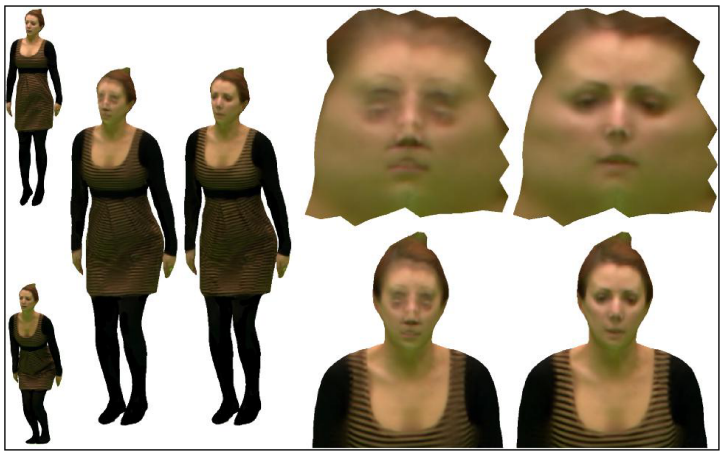
observe than less than half training images are needed to achieve similar performance in texture space, and a quarter less with CATY dataset.

Figure 4.12 shows the mean square generalization errors in both image and texture domains for CATY and TOMAS datasets. The MSE errors seem to follow the same pattern as the SSIM errors, except for reconstructions in the image domain, where the naive PCA catches up with the performance of our method at a smaller number of training frames. We believe this can be due to the same shortcomings of the MSE error measurement mentioned in the previous section.

4.6 Applications

We investigate below two applications of the appearance representation we propose. First, the interpolation between frames at different time instants and second, the completion of appearance maps at frames where some appearance information is lacking due to occlusions or missing observations during the acquisition. Results are shown in the following section and this video.

4.6.1 Interpolation



(a) Interpolating frames 46 from sequence *High* and 11 from sequence *Far*

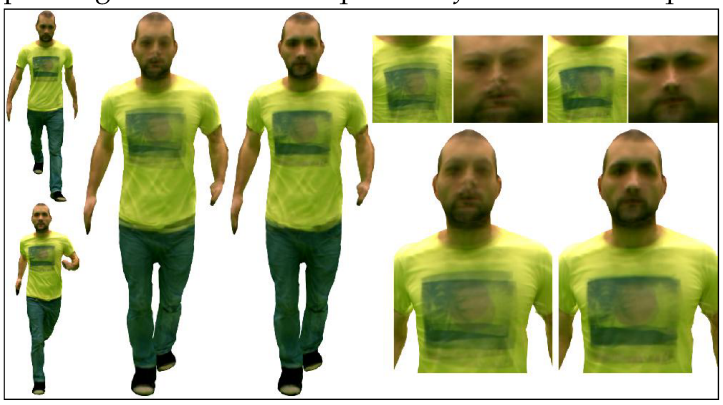
(b) Interpolating frames 31 from sequence *High* and 18 from sequence *High*

Figure 4.13 – Interpolation examples from CATY dataset using linear interpolation (left) and our pipeline (right). From left to right: Input frames, Interpolated models, and a close-up on the texture maps (top) and the rendered images (bottom).

In our framework, appearance interpolation benefits from the pre-computed warps and the low dimensionality of our representation to efficiently synthesize compelling new appearances with reduced ghosting-artefacts. It also easily enables extension of appearance interpolation from pairwise to multiple frames. Assume that shapes between two given frames are interpolated using a standard non-linear shape interpolation, for instance the method we propose in section 2.4.3. Consider then the associated aligned textures and associated warps at the given frames. We perform a linear interpolation in the Eigen texture and Eigen warp spaces respectively by blending the projection coefficients of the input

(a) Interpolating frames 25 from sequence *Run* and 31 from sequence *Walk*

(b) Interpolating frames 4 from sequence *Left* and 4 from sequence *Right*



(c) Interpolating frames 31 from sequence *Walk* and 22 from sequence *Run*

Figure 4.14 – Interpolation examples from TOMAS dataset using linear interpolation (left) and our pipeline (right). From left to right: Input frames, Interpolated models, and a close-up on the texture maps (top) and the rendered images (bottom).

appearance maps. Poisson warping, as introduced in section 4.4.1 is used to build de-aligned interpolated texture with the interpolated backward warp. Figures 4.13 and 4.14 compare interpolation using our pipeline to a standard linear interpolation for 4 examples from the CATY and

TOMAS datasets. Note that our method is also linear but benefits from the alignment performed in the texture space to reduce interpolation artefacts, as well as from the simplified computational aspects since interpolation applies to projection coefficients only.

4.6.2 Completion

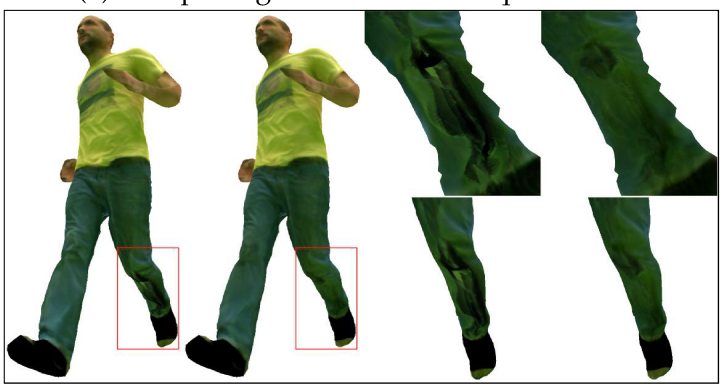
As mentioned earlier, appearance maps can be incomplete due to acquisition issues. For instance, as shown in Figure 4.15, during the running sequence the actor TOMAS bends his knees in such a way that the upper parts of his left and right shins become momentarily hidden to the acquisition system. This results in missing information for those body parts in the texture maps and over a few frames. Such an issue can be solved with our texture representation by omitting the incomplete frames when building our appearance representations, and then projecting these incomplete appearance maps in the Eigen spaces and reconstructing them using the projection coefficients and Poisson texture warping. Figure 4.15 shows two examples of this principle with occluded regions. The top figure shows frame 15 and the bottom one shows frame 24 from the *Run* sequence of *Tomas* dataset. We show side to side the original input model with missing texture and its completed version. In both cases, we show the rendered 3D textured model (left), and a close up on the region of interest of the model (bottom right) and the 2D texture map (top right). Note however, that while effectively filling gaps in the appearance map, this completion might yet loose appearance details in regions of the incomplete map where information is not duplicated in the training set.

4.7 Conclusion

We presented in this chapter a novel framework to efficiently represent the appearance of a subject observed from multiple viewpoints and in different motions. We propose a straightforward representation that decomposes this dynamic scene appearance modelling problem into Eigen textures and Eigen warps. These Eigen vectors encode, respectively, the appearance variations due to viewpoint and illumination changes, and due to geometric modelling imprecisions. The framework was evaluated on 2 datasets and with respect to:

- The ability to accurately reproduce appearances with compact representations.
- The ability to resolve appearance interpolation and completion tasks.

(a) Completing frame 15 from sequence *Run*



(b) Completing frame 24 from sequence *Run*

Figure 4.15 – Completion examples from TOMAS dataset. From left to right: Input and completed models, close-up on input and completed texture maps (top) and rendered images (bottom).

In both cases, the interest of a global appearance model for a given subject was demonstrated. Among the limitations, the representation performances are dependent on the underlying geometries. Future strategies that combine both shape and appearance information would thus be of particular interest. The method also depends on the robustness of the alignment and de-alignment of textures. Dense texture map correspondence through optical flows is proposed as a baseline solution, but other alternatives might be explored. Finally, the proposed model could also be extended to global representations over populations of subjects as a future step.

Although our representation builds on PCA, the problem of dynamic scene appearance modelling is fundamentally one of non-linear dimen-

sionality reduction of images from all viewpoints and time frames. The informed reduction to a linear PCA that we propose is one of the most straightforward approaches one can imagine to the problem and could be a baseline for future, more complex approaches. As a matter of fact, more elaborate reductions are always possible, but note that using e.g. agnostic non-linear schemes from scratch would be burdened to rediscover non-linear variations resulting from the underlying geometry of the scene, for which good estimates are already available using state of the art surface tracking techniques. We simply leverage these as compensation estimates in our method to eliminate the main non-linear components: the viewpoint to texture-space mapping on one hand, which is governed by the 3D tracking and initial per-frame texture-space estimation; the common texture-space mapping to reduce all textures to one consistent appearance layout on the other hand, which is compensated by the geometric warps.

Chapter 5

Shape Motion transfer

Contents

5.1 Introduction

In this chapter, we address the task of transferring captured motions from one subject to another, to the benefit of enabling the generation

of uncaptured animations, removing the burden of exhaustive motion acquisition for each subject, and broadening creative possibilities with captured 4D models. Given two training sets of motions for two subjects and a sparse set of annotated semantic correspondences between the two sets, a regression model can be built, by which a new captured motion associated to one of the subjects can be transferred to the other. While the transfer for sparse surface parametrizations such as motion capture data have received some attention Rhodin et al. [2014], Vögele et al. [2012], Yamane et al. [2010], Feng et al. [2008], only a handful of works exists that deal with the particular problem of surface to surface transfer relevant to our shape capture situation Sumner and Popović [2004b], Baran et al. [2009]. The most successful approach to date Baran et al. [2009] obtains inspiring results with a linear representation of pose mapping.

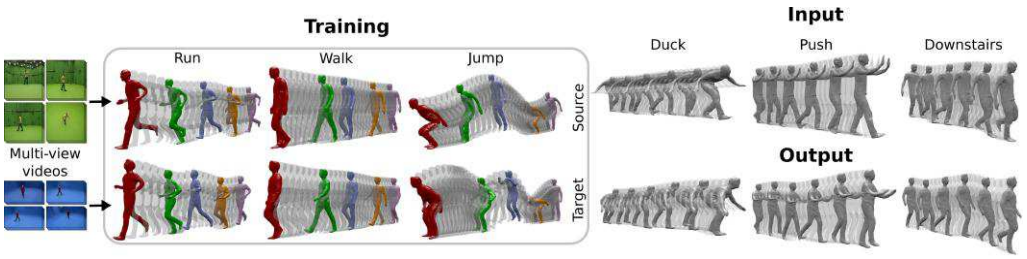


Figure 5.1 – Motion capture transfer. Left: Multi-View acquisition of motion sequences for 2 subjects; Middle: Training of the motion mapping between subjects given shape sequences with sparse frame correspondences (in color); Right: Motion animations of subject 2 (bottom) given new input motions for subject 1 (top).

We propose a more general model, which significantly broadens transferring capabilities in a multi-view capture context, thanks to the following two important contributions:

- First, we propose a novel motion mapping model, based on non-linear Gaussian Process regression and on body part segmentation, which significantly improves the accuracy and generalization abilities of mappings for a given training set. Unlike existing works, this regression model also captures the global rigid component of motion, allowing more realistic transfer of subject displacements.
- Second, while previous works only use a sparse set of matching key-frames in the training set, we provide a full temporal matching densification approach based on probabilistic dynamic time-warping. Starting from the initially provided matching pairs, or key-frames, between two

paired training sequences, it extrapolates correspondences densely between the two sequences. This allows the regression to benefit from an expanded and complete set of frame pairs, which in turn increases realism by better preserving learned individual dynamic patterns in new motions.

In order to conduct evaluation, we consider two annotated datasets acquired with different platforms. These datasets contain motions for two subjects, any subset of which can be used for training and the other for testing. Testing data annotations are used as baseline to measure the accuracy of the transfer and generalization capability of the method. We also evaluate the benefit of densification versus only using the initially annotated sparse key-frame pairs. Under this protocol, we are able to verify quantitatively and qualitatively that our method outperforms all existing strategies and yields more visually pleasing results in typical vision-based surface capture scenarios, in particular achieving higher fine-scale motion fidelity of the transfer.

5.2 Related Work

Existing works in the field of captured surface motion transfer can be roughly categorized with respect to the surface motion parametrization they consider and to the motion transfer model they apply. Articulated motions can be naturally parametrized using skeletons and several works resort to such models to transfer motions between subjects as in Feng et al. [2008], Vögele et al. [2012], Yamane et al. [2010]. In order to be applicable to surface based information, e.g. 3D meshes in our case, this strategy requires mesh rigging and skinning or tracked control points. Obtaining this information reliably in generic situations is often difficult, for instance with loose clothing, and we follow a different strategy that does not rely on intermediate representations.

Regarding surface to surface motion transfer, and given vertex correspondences between source and target meshes, the seminal work of Sumner and Popović [2004b] proposed an elegant solution to transfer deformations between triangulated meshes based on deformation gradients. Such a solution was applied to synthetic and real models of humans, animals as well as recently to faces Thies et al. [2016]. Ben-Chen et al. [2009] generalized such deformation transfer to multi-component and non manifold meshes using harmonic maps. While successful, this strategy however requires correspondences between surface meshes which can easily be involved and complicated. In addition, a local transfer strategy

can be disadvantageous with shapes that differ in the way they deform locally when undergoing similar global motions. We therefore favour a more global approach.

Some works attempted surface to surface motion transfer without correspondences. The work of Rhodin et al. [2014] uses a linear pose mapping from sparse point clouds to surface data for character control. Closer to our concern, Baran et al. [2009] proposed a method that performs semantic deformation transfer between surfaces. This method does not require local correspondences and accomplishes instead motion transfers through poses within spaces spanned by key-poses (key-frames), for which global correspondences are given. This approach was later extended to multi-component objects Zhou et al. [2010]. We follow a similar strategy, albeit significantly improving the pose space representation, which accounts for all training poses in our case, instead of sparse key-poses only. Ours differs also in the transfer function that accounts for global rigid displacements and is based on non-linear regressions instead of direct pose mapping, with the benefit of allowing the transfer of more complex motion patterns.

Motion transfer models have been largely studied in the literature in the case of motion capture and therefore sparse data. They range from direct transfer to linear and non-linear interpolation models. Among the latter, Gaussian Processes have proven to be effective for performing various tasks with motion capture data, such as non-trivial transfers to non-humanoid characters Yamane et al. [2010] or, interestingly with complex real subjects, motion style and variability modelling Wang et al. [2007, 2008a]. Our framework thus builds on Gaussian Process Regression and extends it to surface mesh data.

We use a body part based representation for pose regression. Many works use similar representations for mesh related tasks like registration Zuffi and Black [2015] or animation Tejera and Hilton [2013]. From an anatomical standpoint, this representation makes sense since body parts move independently, which makes it hard to align and learn their motions with a global model. To confirm these claims, we evaluate our regression with both a whole body and a part based representation, and we show in section 5.7 that the latter yields better results.

Furthermore, we propose a user friendly body partition scheme that doesn't require accurate body segmentation. The user merely provides a roughly selected area of uncertainty delimiting the body parts (see figure 6.2). In the learning stage, each body part is augmented by this overlap region with it's neighbour. In the inference stage, optimal boundaries within these overlap regions are automatically selected to stitch the body

parts back together seamlessly using Poisson mesh merging, inspired by this work Huang et al. [2007b]. Introduced first for images Pérez et al. [2003], the Poisson editing method was later incremented by optimal boundary selection Jia et al. [2006] for better results. These concepts were then generalized to videos Wang et al. [2004a], Chen et al. [2013]. In a similar fashion, Poisson editing for static meshes Yu et al. [2004a] was enhanced with optimal merger boundaries Huang et al. [2007b], and we generalize this concept in our work to dynamic mesh sequences.

5.3 Semantic Motion Transfer

Our approach considers as input 3D shape sequences of two moving subjects as acquired with multi-view acquisition systems (see figure 5.1). It handles the possibility of different acquisition systems for source and target, *e.g.* different resolution and frame rate. Shapes are represented by 3D meshes which are globally consistent for a given subject, *i.e.* all the subject's poses are represented with the same mesh. This is not true between subjects that can even present different topologies. A first set of various motions is used to train the transfer function between subjects. We assume that sparse frame correspondences between sequences (as illustrated with colors in figure 5.1) are given to bootstrap the motion transfer. They represent semantic correspondences Baran et al. [2009] between subject's poses as seen or desired by the user and therefore need not be accurate. The transfer mapping estimation follows then two steps:

- Correspondence Densification: from the sparse semantic correspondences, or key-poses, a dense correspondence map between the two subject's poses is obtained using dynamic time warping applied to motion sequences in the training set.
- Transfer Model: a non-linear mapping between the subjects' pose spaces is learned using Gaussian Process Regression applied on the full set of pose correspondences.

Given then a newly acquired motion sequence for one of the subjects, its transferred version onto the other subject is obtained by mapping each pose of the source sequence to a corresponding pose in the space of the target subject using the transfer model. In practice, shapes are represented as body parts over which regressions are performed. The benefit is to significantly improve the transfer model accuracy since the limbs of observed subjects can move independently, making global regression over the full body less precise. Another feature of the approach is to compensate the increased complexity with body part regressions using PCA dimension

reduction applied to body parts. We introduce the different components of the approach below.

5.4 Shape Pose Representation

A shape is represented by a 3D mesh $\mathcal{M}$, whose pose is encoded in its vertex positions. Different subjects are represented by different meshes. A shape pose is in practice characterized by elements such as its global rigid transformations and its body part poses. While not fully independent, only weak dependencies of body limb movement with body motions are observed and the body-part model produces better results in our experiments. Consequently, we choose to learn transfer functions for all elements using independent regressions. In addition, we further reduce body part pose representations using PCA.

5.4.1 Rigid Transformation

Let $\{\mathcal{M}_i\}_{1\leq i\leq n}$ be the n poses of a shape $\mathcal{M}$. We first rigidly align all the consistent meshes $\{\mathcal{M}_i\}_{1\leq i\leq n}$ of a subject using standard Procrustes analysis applied to the mesh vertex coordinates. We assume, without loss of generality, that all motions starts at the same space location and with the same initial orientation. Hence, a shape pose is represented by its aligned mesh $\bar{\mathcal{M}}_i$ and its globally rigid displacement from the previous pose in the motion sequence in which it appears. This elementary rigid motion is composed of a translation δT_i and rotation δR_i that will further be part of the transfer analysis.

In order to learn a mapping between source and target subject displacements, we need a linear parametrization of this data. Rotation matrices δR, which lie in the Lie group $SO(3)$, can be expressed as the exponential of skew symmetric matrices $\widehat{H}$, belonging to the Lie algebra $so(3)$:

$$\delta R = \mathrm{e}^{\widehat{H}},\ \widehat{H} = \begin{pmatrix} 0 & -h_3 & h_2 \\ h_3 & 0 & -h_1 \\ -h_2 & h_1 & 0 \end{pmatrix},\ H = \begin{pmatrix} h1 \\ h2 \\ h3 \end{pmatrix} \tag{5.1}$$

where vector's H norm and direction give the angle-axis representation of rotation δR. The global subject's rotations between successive frames being relatively small $\|H\| \approx 0$, they can be approximated accordingly:

$$\delta R = \sum_{k=0}^{\infty} \frac{\widehat{H}^k}{k!} \approx I + \widehat{H} \tag{5.2}$$

Thereafter, we hence represent elementary rigid subject displacements in the following 6-dimensional linear parametrization:

$$(\delta T, \delta R) \mapsto (t_x, t_y, t_z, h_1, h_2, h_3)^T \tag{5.3}$$

5.4.2 Body Parts

As mentioned earlier, body parts can move independently, for instance arms can move differently over various instances of a walking movement while legs present similar motions. As a result, learning independent transfer functions for each body part can increase accuracy compared to a global strategy, which is confirmed by our experiments (see section 5.7). In order to ease the decomposition in practice while keeping robustness, we adopt a strategy similar to Tejera and Hilton [2013] with a coarse but overlapping mesh segmentation. To this purpose, as illustrated in figure 6.2, for each subject the user provides closed and non-intersecting curves that delimit each overlapping region between contiguous body parts on $\mathcal{M}$. Throughout the regression process, each body part will be augmented with the overlap regions it shares with its neighboring parts. During the motion transfer, merging will be achieved to ensure seamless body part stitching as described later in section 5.6.1.

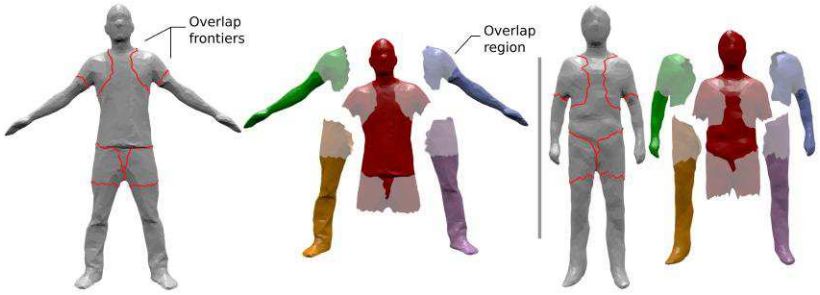


Figure 5.2 – Body parts: shaded colors represent overlapping regions between parts delimited by user given curves.

Each aligned mesh $\bar{\mathcal{M}}_i$ is therefore decomposed into N body part sub-meshes $\{\mathcal{P}_i^k\}_{1 \leq k \leq N}$. In our experiments, we use $N = 5$ parts in a tree structured hierarchy including a torso as the root, and a pair of arms and legs as children nodes.

5.4.3 Dimension Reduction

In order to reduce the computational cost, dimension reduction is applied on each body part using Principal Component Analysis. All

body part sub-meshes for a given subject are first rigidly aligned and PCA applied on the vertex locations. For a given pose i, each body part $\mathcal{P}_i^k$ is now on expressed as a vector of eigen decomposition coefficients $\mathbf{x} = (x_1, \ldots, x_m)^T$, where the reduced dimension m could be different from source to target subjects. In practice, we use $m = 20$ coefficients for each body part. Note that this operation, as the body segmentation step, introduces additional inaccuracies in the overlapping regions that require post-processing (section 5.6.1).

5.5 Shape Pose Mapping

Given two groups of shape poses for two subjects and a set of key-pose correspondences between the groups, we want to estimate a function that maps poses from one group to the other. Since the objective is to transfer motions, hence sequences of poses, we expect this function to provide consistent and realistic temporal pose arrangements. Relying for that purpose on sparse key-poses only, as in Baran et al. [2009], is suboptimal when global semantic motion correspondences, e.g. walking or running, are known. We therefore first extend the set of key-pose correspondences to full pose correspondences between similar motions. Second, in order to better capture pose inter-dependencies, we perform a global non-linear regression over all pose correspondences. Our experiments demonstrate that both contribute to a better accuracy of motion transfers.

5.5.1 Temporal Correspondence Densification

For every pair of matching motions in the training set, sparse semantic correspondences between the source and target poses are provided by the user (see figure 5.3). Starting from these initial correspondences, our aim is to optimally propagate associations to the rest of the sequence by solving a shortest path problem using dynamic time warping. The costs of source and target pose associations are derived from a linear mapping between poses as learned from the initial correspondences. A linear model per motion is sufficient here to capture the information given by the very few initial associations (around 5 key-poses per sequence in our experiments).

Association Cost

Using the notations introduced in section 5.4.2, we hence assume that the mapping between the target and source pose vectors $\mathbf{y}$ and $\mathbf{x}$ of part k

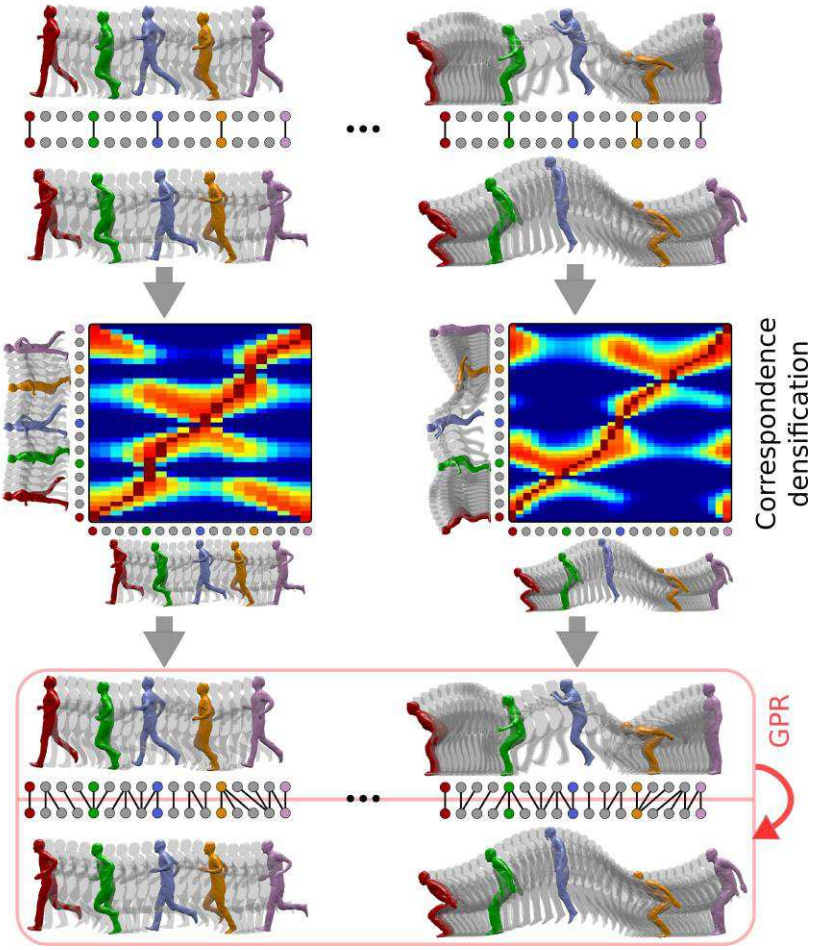


Figure 5.3 – Pose correspondence densification and mapping.

is linear, up to an observation noise vector $\boldsymbol{\epsilon}$, thus:

$$\mathbf{y} = \mathbf{M}\mathbf{x} + \boldsymbol{\epsilon}, \tag{5.4}$$

where $\mathbf{M}$ is the matrix of the linear transformation for part k. We assume that the additive noise follows a Gaussian distribution: $\boldsymbol{\epsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2\mathbf{I})$, where $\sigma^2 = 0.01$ in our experiments. Together with the above linear formulation, this assumption gives rise to the following likelihood for body part k:

$$p(\mathbf{Y}|\mathbf{X}, \mathbf{M}) = \mathcal{N}(\mathbf{M}\mathbf{X}, \sigma^2\mathbf{I}), \tag{5.5}$$

where matrices $\mathbf{X}$ and $\mathbf{Y}$ contain the stacked source and target initially matched key-poses: $\mathbf{X} = (\mathbf{x}_{1^s}, \ldots, \mathbf{x}_{L^s})$ and $\mathbf{Y} = (\mathbf{y}_{1^t}, \ldots, \mathbf{y}_{L^t})$ for part k, assuming we have L key-pose correspondences. Given a zero mean

identical Gaussian prior on rows of $\mathbf{M}$: $\text{row}(\mathbf{M}) \sim \mathcal{N}(\mathbf{0}, \mathbf{I})$, the posterior distribution over parameters in M can then be written as follows, using Bayes' rule:

$$\begin{aligned} p(\mathbf{M}|\mathbf{Y}, \mathbf{X}) &\propto p(\mathbf{Y}|\mathbf{X}, \mathbf{M})p(\mathbf{M}), \\ p(\mathbf{M}|\mathbf{Y}, \mathbf{X}) &= \mathcal{N}(\tfrac{1}{\sigma^2}\mathbf{Y}\mathbf{X}^T\mathbf{A}^{-T}, \mathbf{A}^{-1}), \end{aligned} \tag{5.6}$$

where $\mathbf{A} = \frac{1}{\sigma^2}\mathbf{X}\mathbf{X}^T + \mathbf{I}$. The predictive distribution for the function $\mathbf{M}\mathbf{x}$ at $\mathbf{x}$ is obtained by averaging over all matrices M with the Gaussian posterior in (5.6):

$$\begin{aligned} p(\mathbf{M}\mathbf{x}|\mathbf{x}, \mathbf{X}, \mathbf{Y}) &= \int_{\mathbf{M}} p(\mathbf{M}|\mathbf{X}, \mathbf{Y})\, \mathbf{M}\mathbf{x} \\ &= \mathcal{N}(\tfrac{1}{\sigma^2}\mathbf{Y}\mathbf{X}^T\mathbf{A}^{-T}\mathbf{x}, \mathbf{x}^T\mathbf{A}^{-1}\mathbf{x}\mathbf{I}). \end{aligned} \tag{5.7}$$

Hence for a given body part k, the probability of the target pose $\mathbf{y}_j$ to belong to the predictive distribution for $\mathbf{M}\mathbf{x}$ at the source pose $\mathbf{x}_i$ is then:

$$P_{i,j}^k = e^{-\frac{1}{2}(\mathbf{y}_j - \boldsymbol{\mu}_i)^T(\boldsymbol{\Sigma}_i)^{-1}(\mathbf{y}_j - \boldsymbol{\mu}_i)}, \tag{5.8}$$

where $\boldsymbol{\mu}_i = \frac{1}{\sigma^2}\mathbf{Y}\mathbf{X}^T\mathbf{A}^{-T}\mathbf{x}_i$ and $\boldsymbol{\Sigma}_i = \mathbf{x}_i^T\mathbf{A}^{-1}\mathbf{x}_i\mathbf{I}$. For the whole body, the probability, or cost, of this association between poses i and j is then the product of the latter probabilities over all the body parts:

$$P_{i,j} = \prod_{1 \le k \le N} P_{i,j}^k. \tag{5.9}$$

Dense Correspondence

Denoting by f_s and f_t the source and target sequence sizes, the next step is to find a map $\phi : [\![1, ..., f_s]\!] \to [\![1, ..., f_t]\!]$ between source and target sequence poses that maximizes the product of the association probabilities $\{(i, \phi(i))\}$:

$$\begin{aligned} \phi_* &= \underset{\phi}{\text{argmax}} \prod_i P_{i,\phi(i)}, \\ &= \underset{\phi}{\text{argmin}} \sum_i -\log\left(P_{i,\phi(i)}\right). \end{aligned} \tag{5.10}$$

When formulated as a minimization, as above, and taking into account some motion priors such as continuity and monotonicity, finding a mapping ϕ can be cast as a shortest path problem, solved using a dynamic programming algorithm Müller [2007]. The solution corresponds to a least cost path within a cost matrix (shown in figure 5.3) taking the negative log of the probability $P_{i,j}$ as value for node (i, j). In our implementation, we bound the path through the cost matrix to be continuous, not reserve

path, not include more than 3 vertical or horizontal consecutive nodes in a row, and also run through the initial key-pose correspondences.

5.5.2 Pose and Displacement Mapping

Using the densification approach presented before, we can augment the set of source and target pose correspondences using all the motion sequences present in the training datasets. This allows to benefit from both variability and redundancy in the shape pose training set. In the previous section, a linear model was used to map poses between the source and target in each motion sequence. We consider now the full set of corresponding poses, whose correspondence distribution is expected to be better captured by a more elaborate model, as demonstrated in our experiments.

Non-linear Model

For a given body part, we assume that the i-th parameter y_i of the target pose $\mathbf{y}$ can be related to a new source vector $\mathbf{x}$ with a non-linear function and up to an observation noise:

$$y_i = f_i(\mathbf{x}) + \varepsilon, \tag{5.11}$$

with $\varepsilon \sim \mathcal{N}(0, \sigma_n^2)$ and $\sigma_n^2 = 0.1$ in our experiments. We use a Gaussian Process (GP) to describe the distribution over such functions $f_i(\mathbf{x})$ given a training set.

A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution. It extends multivariate Gaussian distributions to infinite dimensionality. GP have shown to be efficient in solving similar regression problems on sparse motion representations. A GP is defined by its mean and covariance functions. Since body parts are first aligned, we assume the mean functions to be null. For the covariance functions, we introduce the following entities:

$$\begin{aligned} \mathbf{K} &= \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_F) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_F, \mathbf{x}_1) & \dots & k(\mathbf{x}_n, \mathbf{x}_F) \end{bmatrix}, \\ \mathbf{K}_* &= \begin{bmatrix} k(\mathbf{x}_*, \mathbf{x}_1) & \dots & k(\mathbf{x}_*, \mathbf{x}_F) \end{bmatrix}, K_{**} = k(\mathbf{x}_*, \mathbf{x}_*), \end{aligned} \tag{5.12}$$

where F is the total number of training poses, and $\mathbf{x}_*$ is a new input pose for a body part and $k(.,.)$ a kernel function. In our experiments, the

neural network covariance below outperforms other traditional kernels (see section 5.7):

$$k_{\text{nn}}(\mathbf{x}, \mathbf{x}') = \sigma_f^2 \sin^{-1}\left(\frac{\frac{1}{2l^2}\mathbf{x}^T\mathbf{x}'}{\sqrt{(1+\frac{1}{2l^2}\mathbf{x}^T\mathbf{x})(1+\frac{1}{2l^2}\mathbf{x}'^T\mathbf{x}')}}\right) \tag{5.13}$$

where the hyper-parameters l and σ_f are optimized using conjugate gradients Rasmussen and Williams [2005]. We also use the squared exponential and the linear kernels for a comparative study in section 5.7.

$$\begin{aligned} k_{\text{se}}(\mathbf{x}, \mathbf{x}') &= \sigma_f^2 \exp\left(-\tfrac{1}{2l^2}\left(\mathbf{x}-\mathbf{x}'\right)^T\left(\mathbf{x}-\mathbf{x}'\right)\right) \\ k_{\text{lin}}(\mathbf{x}, \mathbf{x}') &= \mathbf{x}^T\mathbf{x}' \end{aligned} \tag{5.14}$$

Using the GP prior on function f_i, the joint distribution of the observed target pose parameters $\{y_i\}$ over the training set and the predicted parameter $y_{i*} = f_i(\mathbf{x}_*) + \varepsilon$ at the new source pose $\mathbf{x}_*$ can be expressed as follows:

$$\begin{bmatrix} y_i \\ y_{i*} \end{bmatrix} \sim \mathcal{N}\left(\mathbf{0}, \begin{bmatrix} \mathbf{K}+\sigma_n^2\mathbf{I} & \mathbf{K}_*^T \\ \mathbf{K}_* & K_{**} \end{bmatrix}\right). \tag{5.15}$$

By conditioning the joint Gaussian prior distribution on the observations Rasmussen and Williams [2005], we obtain the posterior distribution for y_{i*}:

$$\begin{aligned} p(y_{i*}|\mathbf{X}, \mathbf{Y}_i, \mathbf{x}_*) \sim \mathcal{N}\big(&\mathbf{K}_*[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}\mathbf{Y}_i, \\ &K_{**} - \mathbf{K}_*[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}\mathbf{K}_*^T\big), \end{aligned} \tag{5.16}$$

where the matrix $\mathbf{X}$ contains the stacked values of the source part poses $\mathbf{x}$ in the training set and the vector $\mathbf{Y}_i$ contains the stacked values of the pose parameter y_i in the corresponding target pose in the training set. Finally, for a given input pose $\mathbf{x}_*$, we take the mean of this distribution as the predicted output value;

$$y_{i*} = \mathbf{K}_*[\mathbf{K}+\sigma_n^2\mathbf{I}]^{-1}\mathbf{Y}_i, \tag{5.17}$$

this for all the body part pose parameters $\{y_{i*}\}_{1\leq i\leq m}$ and over all body parts.

Rigid Displacement

As explained in section 5.4.2, a shape pose is characterized by its body part poses as well as its rigid displacement from the previous pose in the

considered motion sequence. This rigid displacement is modelled using a similar GPR approach that learns from all training correspondences the mappings between the source displacement vectors and each of the 6 target displacement vector values.

5.6 Shape Pose Reconstruction

Using the approach described in the previous sections we can predict body part poses and global rigid displacement for any input source pose. Since these predictions are independent, a post-processing step is required to combine them all in a consistent way. This includes body part correction, body part stitching and a global rigid displacement according to the prediction. We elaborate on the body part tasks below. Note also that final motions might require additional traditional post-processing such as self-intersection clean up, or foot-skating correction to ensure foot contacts don't appear to be slipping.

5.6.1 Body Part Correction

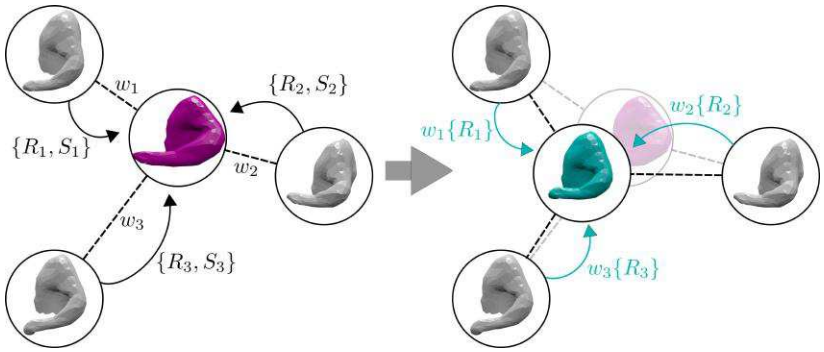


Figure 5.4 – Body part correction algorithm.

As a result of the reduced pose parametrization, mesh deformities can appear in the predicted body-part meshes such as shrinkage, distortion and local magnification. We propose in this section an example-based algorithm that solves these limitations. It relies on the assumption that mesh deformities result from strong non-isometric deformations, that we therefore try to factor out. The algorithm is composed of the following steps, illustrated is figure 5.4:

- The M nearest meshes to the predicted mesh are found in the training set, according to a simple vertex-to-vertex distance. In our experiments,

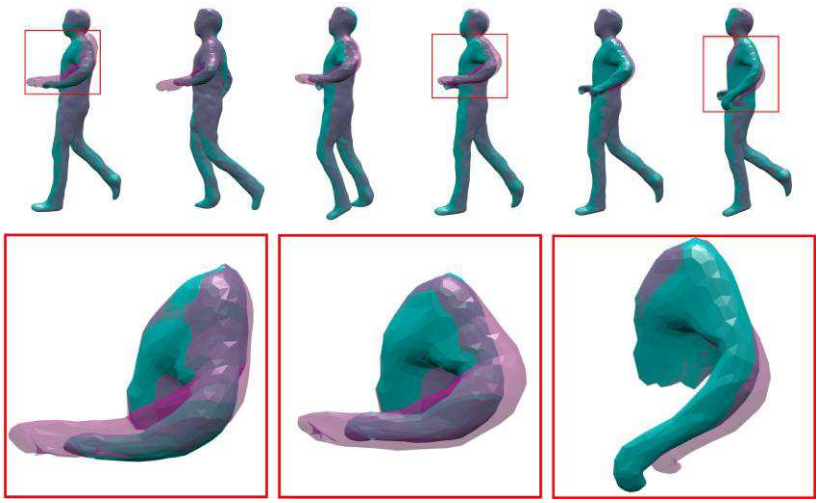


Figure 5.5 – Example of body part correction (in green).

3 nearest meshes were enough to obtain perceptually valid results. A weight w_i is associated to each nearest mesh, which decreases with the distance to the predicted mesh.

- Polar decompositions are performed between triangles on nearest meshes and their correspondents on the predicted mesh to obtain isometric $\{R_i\}$ (Rotation) and non-isometric $\{S_i\}$ (Shear/Scale) deformation components per triangle.
- Nearest mesh triangles are deformed with only the isometric per-face components $\{R_i\}$ and the corresponding deformed meshes are reconstructed using a Poisson based method Xu et al. [2005]. The resulting M deformed meshes are then combined using weights $\{w_i\}$.

Figure 5.5 shows a correction example where the algorithm helps correct shrinkage and local magnification in the prediction.

5.6.2 Body Part Stitching

Assuming we deal with zero-genus manifold meshes, drawing two non-intersecting curves on the mesh will result in an overlap region that is topologically equivalent to a cylinder, as can be visualized in figure 5.6 flattened with conformal mapping. For a pair on contiguous body parts, and after rigidly aligning the child parts to the root part with respect to the overlap region, we look for a closed curve within this shared overlap topology that we will use as a Dirichlet boundary condition for Poisson mesh merging algorithm Yu et al. [2004a] applied to these two parts. The topological curve that is more likely to produce seamless merges is the one

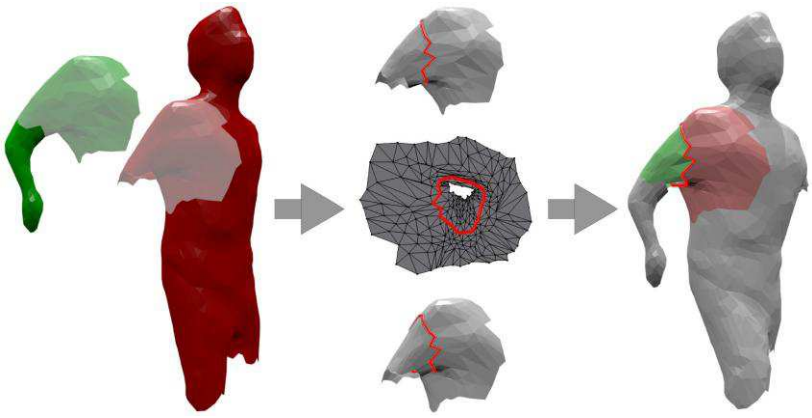


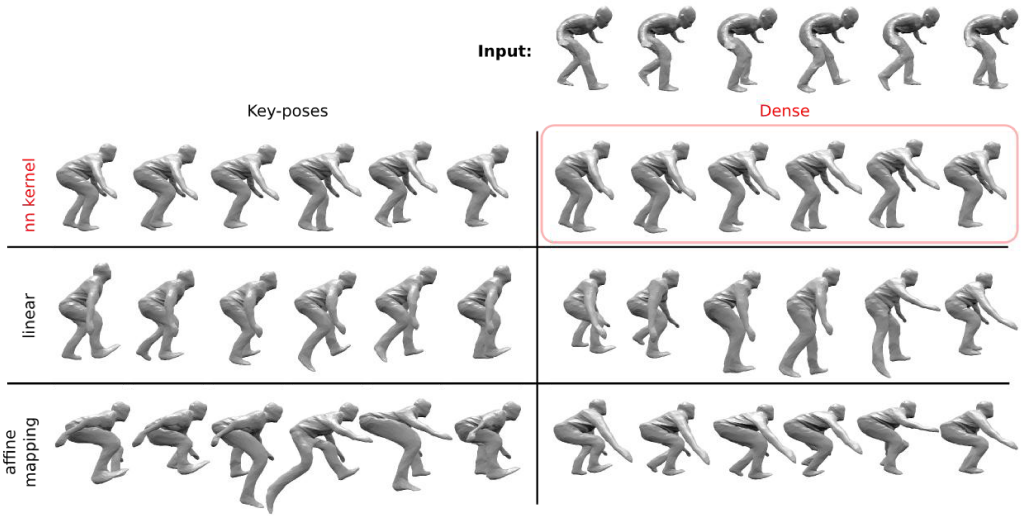
Figure 5.6 – Body part stitching.

with the least deformation cost between its two instantiations in the root and child geometries. Following Huang et al. [2007b], an approximation of this solution is found with a Dijkstra shortest path algorithm. This process is subsequently reiterated for all overlap regions to recover the full final body merged mesh.

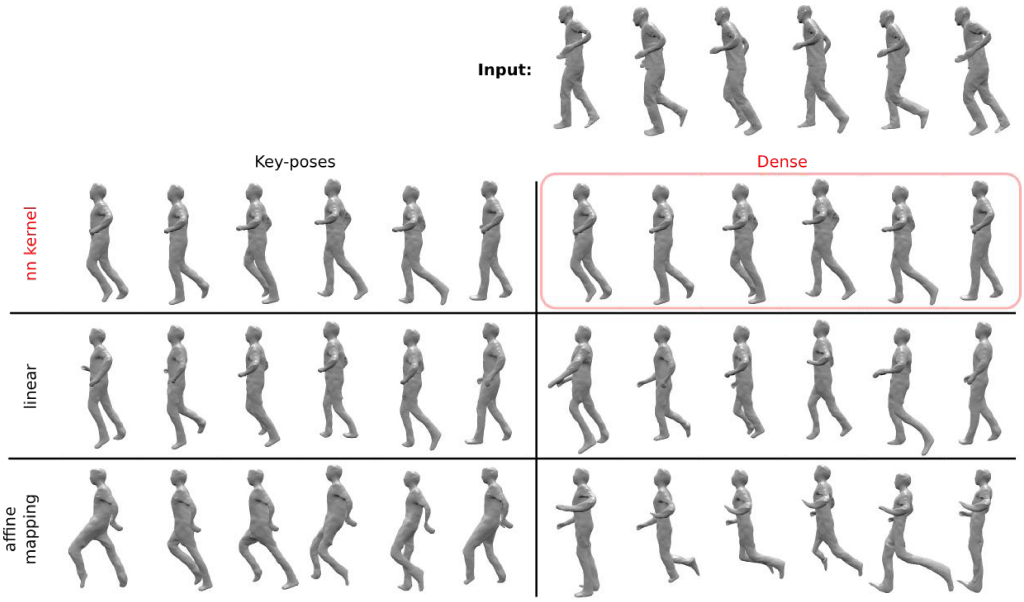
5.7 Evaluation

To demonstrate our method, we use two datasets containing temporally coherent mesh sequences of matching basic motions, and we attempt motion transfer from one dataset to the other in both ways. DAN Casas et al. [2014a] dataset has meshes with 2667 vertices and 5330 faces and was recorded at 25 fps. TOMAS Boukhayma and Boyer [2015] dataset has more uniform meshes with 5000 vertices and 10000 faces and was recorded at a higher frame rate; 50 fps. For the training, we match TOMAS's sequences *Walk*, *Run*, *Jump*, *Jump forward* and *Bend* with DAN's sequences *Walk*, *Jog*, *High jump*, *Long jump* and *Big box low* respectively. We initialize the key-pose correspondences with 5 frames for each sequence and end up with a total number of 350 pairs of frame correspondences after densification. As we show in figures 5.7,5.1 and this video, we succeed in transferring both motions that are semantically similar to the ones in the training set, like *Run2walk*, and other new motions not represented in the training set, such as *Duck*, *Push*, *Upstairs*, *Downstairs*.

For quantitative evaluations, we fully annotate the motion datasets with frame to frame correspondences, and we randomly split this ground-truth data into a training set and a cross-validation set. The training set



(a) Sequence *Duck*

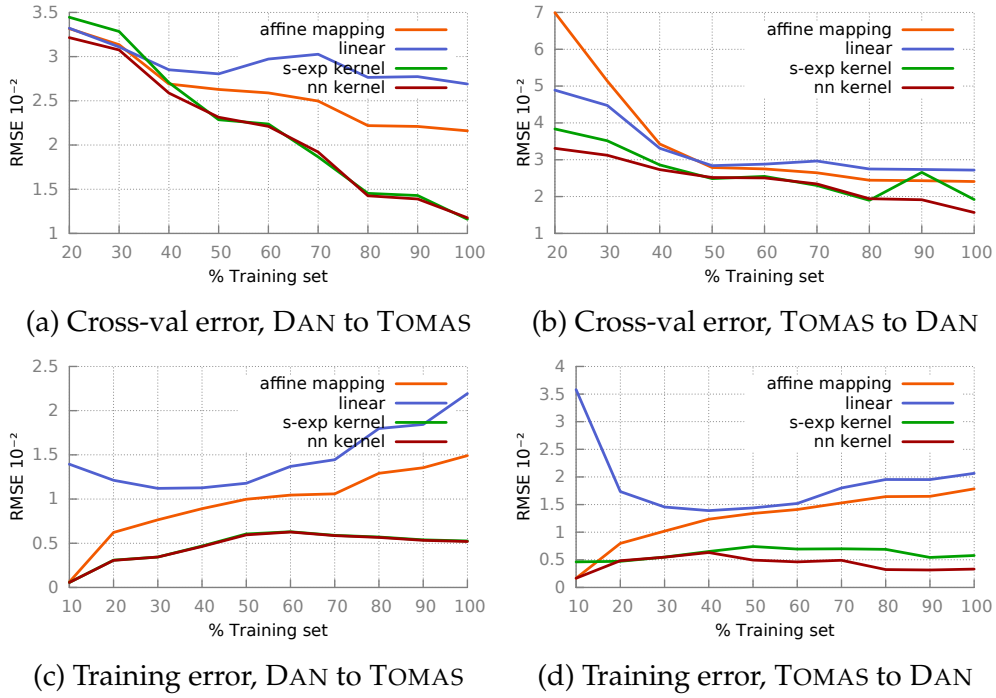


(b) Sequence *Run2walk*

Figure 5.7 – Transferring from TOMAS to DAN.

accounts for 70% of the pair samples, while the remainder belongs to the testing set. For a given pose regression method, we plot the training set error and the cross-validation set error as a function of the size of the training subset and in terms of root mean square of mesh per-vertex error. We carry these experiments for transfer in both direction, i.e. from TOMAS to DAN and from DAN to TOMAS.

5.7.1 Transfer Model



(a) Cross-val error, DAN to TOMAS

(b) Cross-val error, TOMAS to DAN

(c) Training error, DAN to TOMAS

(d) Training error, TOMAS to DAN

Figure 5.8 – Learning curves for various regression models.

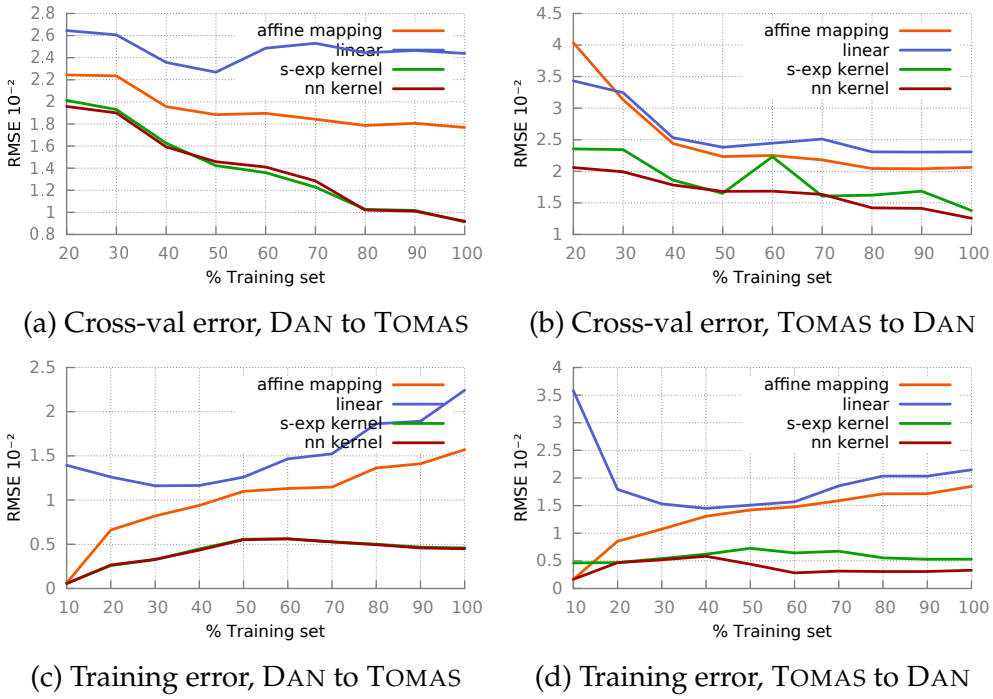
We compare our pose regression model to other methods both quantitatively and qualitatively. We use Gaussian Process regression with the neural network kernel, and compare it to the squared exponential kernel, the linear kernel and the method presented in Baran et al. [2009], which uses source and target affine spaces spanned with key-poses, and a mapping derived from these key-pose pair associations. As shown in figure 5.8, our method outperforms the rest for both training and testing errors.

The low cross-validation error curve of our method is demonstrated in the qualitative comparison through a visually superior generalization ability that we can witness in figure 5.7. For both examples, *Duck* and *Run2walk*, our method succeeds in extrapolating the best prediction corresponding to the input motion, while preserving the properties inherent to the target motion space. The other methods fail to achieve comparable results. We note that we show raw outputs in figure 5.7 for all methods alike, prior to the mesh enhancement process described in section 5.6.1 (see also this video).

Perceptually, we noticed that affine mapping performs particularly poorly in our training scenario, which consists of relatively large train-

ing examples obtained from dense temporal matching between several different motions. On the contrary, our strategy does not suffer from the training set extension thanks to the Gaussian model ability to better handle redundancy, variability and uncertainty in the associations.

5.7.2 Dense Correspondence



(a) Cross-val error, DAN to TOMAS

(b) Cross-val error, TOMAS to DAN

(c) Training error, DAN to TOMAS

(d) Training error, TOMAS to DAN

Figure 5.9 – Learning curves for various regression models. Training subset initialized with key-poses.

We evaluate the impact of using dense correspondence instead of sparse key-pose associations both qualitatively and quantitatively. To this end, we recalculate the learning curves in figure 5.8. Only this time, the training subset, randomly selected with gradually increased size throughout the experiment, is initialized with key-pose pairs. We can see in figure 5.9 that the generalization error globally decreases for all methods as we densify the correspondences, which accordingly translates into visually improved predictions in figure 5.7. The linear regression seems to be an exception to this behaviour. In fact, with more training examples, the linear model under-fits the learning set, which also results in poor predictions. We note also that cross-validation errors are globally lower in figure

5.9 which makes perfect sense as key-poses are meant to better encode variability in the data and hence yield better generalization.

5.7.3 Body Parts

(a) Cross-val error, DAN to TOMAS

(b) Cross-val error, TOMAS to DAN

(c) Training error, DAN to TOMAS

(d) Training error, TOMAS to DAN

Figure 5.10 – Learning curves for the GPR model with neural network kernel with and without body parts.

Using our regression method, we reiterate the learning curve experiments (figure 5.8) for two different pose representations: Using the whole body mesh, and using body parts. As we show in figure 5.10, independent transfer of body part motion compares favourably to the whole body approach both in training and testing errors, for equivalent total number of input and output dimensions.

5.8 Conclusion

We presented in this work a novel solution for surface motion capture transfer that doesn't require source and target mesh correspondence. We contributed a Gaussian Process regression model applied directly to mesh

data, and a correspondence densification scheme based on probabilistic dynamic time warping. Densifying the correspondences helps better capture motion style and variability. Concordantly, this richer variability is modelled more accurately with our non-linear regression. We also propose a user friendly algorithm for body parts separation and automatic stitching, and an example-based approach to improve predicted meshes with respect to the training set. This last heuristic compensates for the limitations of the linear parametrization of shape poses, which is mainly motivated by compatibility with standard regression kernels. A more elaborate regression model designed specifically for manifold valued data Banerjee et al. [2016] could be attempted as a next step. We could also consider transferring appearance information along with geometry.

Chapter 6

Shape Motion Variation Synthesis

Contents

6.1 Introduction

In this chapter, we address the task of generating an unlimited number of variations of a subject movement using a limited number of training frames of captured performance. As humans rarely perform similar actions in the exact same manner every time, variation in motion is an

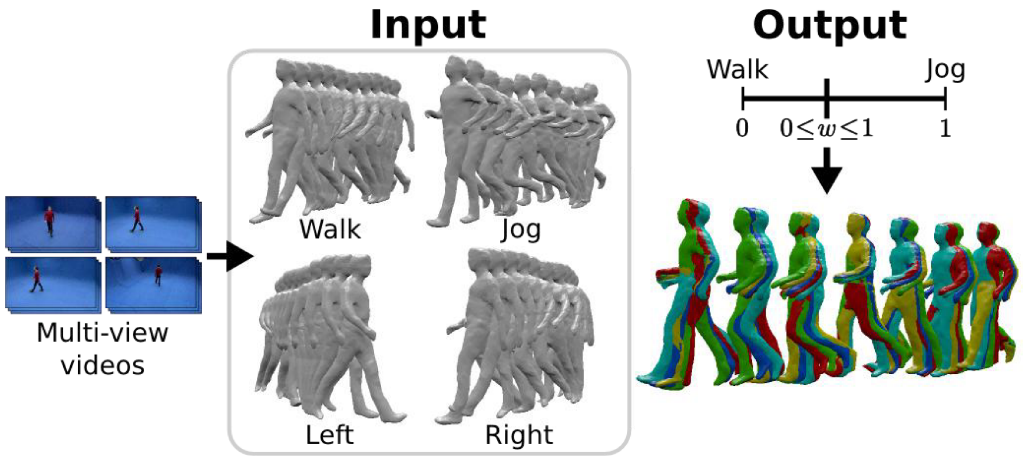


Figure 6.1 – Overview. Left: Multi-view acquisition of motion sequences of the same type (e.g. locomotion). Right: Variation synthesis for any of the input sequences or any blended version of them. 5 variations are overlapped, each with a different color.

essential component of animation realism. Unfortunately, current 4D data animation solutions Casas et al. [2014b], Boukhayma and Boyer [2015], Prada et al. [2016] merely replay input motion segments or blended versions of them. Such exact repetition of motion cycles can lead to unrealistic animations. Hence, a variation model that can generate even slight differences of the original motion cycles can improve the naturalness of the generated animations substantially, and provide new examples without the burden of motion acquisition.

Given a few examples of a particular type of motion as input, such as locomotion (walking, running, turning, etc.) or jumping (jumping far, close, high, low, etc.), a manifold of motion is learned and new variations are obtained by sampling from the later. The user can in particular generate variations of any of the input motions and also any blended version of them. While this problem has received some attention previously for sparse surface representations, namely traditional Motion Capture Wang et al. [2008b], Lau et al. [2009], Ma et al. [2010], to our knowledge this is the first work of its type that considers dense surface representations relevant to our capture situation.

Our main contributions in this chapter are as follows:

- First, we advocate the use of Gaussian Process Dynamical Models directly on mesh data to build a low dimensional embedding of mesh based motion sequences. This two-fold model learns a probabilistic mapping between latent coordinates and mesh vertex coordinates, along with a

second probabilistic mapping between successive frame latent coordinates. New motion examples are next generated by sampling from this manifold using a Hybrid Monte Carlo Markov Chain. The resulting sequences are statistically similar to the initialization of the Markov Chain but are not exact copies of it, as one could see in the latent space (figure 6.5) and the observation space (figures 6.8 and 6.9). This variation in the generated sequences stems from the following: variation in poses in the input data, the independent body part modelling scheme that we elaborate in section 6.4.2, the probabilistic mapping between the latent and the observation spaces, and finally the probabilistic mapping of dynamics in the latent space.

- Second, we propose an algorithm that allows generation of variations of any blended version of the input sequences. This process avoids costly non-linear mesh interpolation of many variations in the observation space, and sampling motion around latent trajectories outside the training set, which leads to degenerate samples. Instead, we learn the model with few pre-interpolated sequences and sample variations only around learned trajectories. For a given requested blending weight, the variations of the closest latent trajectories are interpolated with the appropriate proportions to generate the requested blended sequence variations.

We evaluate our work perceptually in section 6.8 and in the following video using a dataset of surface capture, with two different types of movements: locomotion and jumps. We succeeded in generating variations of the input motions and blended versions of them that are globally similar to the inputs but yet slightly different in both poses and timings. We also provide in section 6.7 numerical validation of the benefit of our motion parametrization scheme guided with pre-blended examples, compared to a simple trajectory interpolation in the latent space.

6.2 Related work

With the term *variation* of motion, we refer to new examples that look globally like the original sequences but differ slightly from them in body poses and their timings of occurrence, thus mimicking human behaviour richness and inexactitude when reproducing the same movement. Previous work on Motion Capture data Bodenheimer et al. [1999]Perlin [1995] attempts to generate variations merely with an additive noise component. However, biomechanical research Harris and Wolpert [1998] asserts that variation is rather a functional component of motion and not just noise. In addition, the work of Lau et al. [2009] on motion capture shows empiri-

cally that there are no guarantees that added noise, either arbitrary or with tuned distributions, matches well with the existing motion, which renders these methods prone to unrealistic results and not robust to automation. Following other works on Motion Capture Wang et al. [2008b], Lau et al. [2009], Ma et al. [2010], we use a data-based approach where variation comes from the data and not a separate additive component.

There are many works in the literature that use use few examples of skeletal human Motion Capture sequences to generate new variations. The work of Pullen and Bregler [2000, 2002] models the correlations between the degrees of freedom in motion data with a distribution, and synthesize new motions by sampling from this distribution. The authors in Lau et al. [2009] use Dynamic Bayesian Networks to model and simulate dynamics in *similar but slightly different* example motions. Ma et al. [2010] interpolates various examples of skeleton joint group motions with Universal Kriging. We follow the work of Wang et al. [2008b] and extend Gaussian Process Dynamical Models Wang et al. [2006] to Surface Motion Capture. The GPDM augments the Gaussian Process Latent Variable Model Lawrence [2004], Grochow et al. [2004] with a latent dynamical model, in addition to a probabilistic non-linear mapping from the latent space to the data space. The dynamic model enables prediction and adds regularization when modelling temporal data, and was used successfully for standard Motion Capture data synthesis.

Motion sequence parametrization Casas et al. [2011b], Kovar and Gleicher [2004b] is a key component in building parametric motion graphs Casas et al. [2014b], Heck and Gleicher [2007b], Gleicher et al. [2008] for interactive character control using skeletal and mesh data alike. For a given set of sequences exhibiting variation of the same type of movement, these sequences are temporally aligned and blended with various weights. However, each blending weight gives a unique output sequence. We extend this framework in this chapter to allow both motion blending and variation synthesis for blended motions in the latent motion space.

6.3 Controllable motion variation synthesis

Our approach considers as input 3D shape sequences of the same subject as acquired from multi-view acquisition systems (See figure 6.1), performing motions of the same type, such as locomotion or jumping. Shapes are represented by globally consistent 3D meshes. In practice, motion sequences are temporally pre-aligned to a reference sequence and shapes are represented as independent body parts. We proceed as follows:

- Motion variation: A Gaussian Process Dynamical Model is used to build a probabilistic low dimensional embedding from the input motions. We use a Hybrid Monte Carlo Markov Chain initialized with a latent motion trajectory to generate variants of this latter. Mesh motion sequences can then be obtained from latent trajectories through the Gaussian Process of pose reconstruction. Variation in the generated outputs comes thus from: variation in poses in the input data, body part independent modelling, probabilistic mapping from latent to observation space, and probabilistic dynamic modelling.
- Motion parametrization: for the sequences that the user wishes to parametrize, we learn the model with few non-linearly pre-interpolated intermediate sequences along with the original ones. We sample variations only around learned trajectories to avoid degenerate samples, and interpolate variations in the latent space of nearby latent trajectories, thus avoiding many non-linear mesh interpolations of variations in the mesh domain.

6.4 Shape pose representation

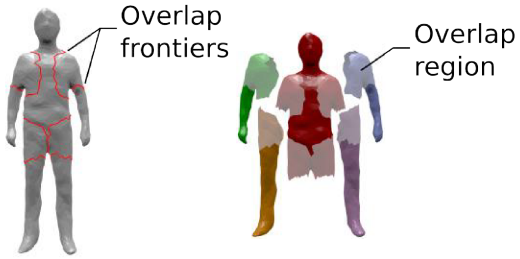


Figure 6.2 – Coarse body partition.

We represent shape in motion in the form of a 3D mesh $\mathcal{M}$ with consistent topology and connectivity. Shape motion can be characterized by elements such as sequential global rigid transformations and body part poses. While not fully independent, modelling limb and global body movements independently produces better results in our experiments. Consequently, we choose to learn motion for body elements using independent models, in a similar strategy to the one followed in chapter 5.

6.4.1 Rigid alignment

For every motion sequence of N frames: $\{\mathcal{M}_i\}_{1 \leq i \leq N}$, we use orthogonal Procrustese analysis to align all frames to a template mesh. The motion

sequence can hence be decomposed into a set of rigidly aligned meshes $\{\bar{\mathcal{M}}_i\}_{1\leq i\leq N}$ and relative rigid displacements between successive frames $\{\delta T_i, \delta R_i\}_{1\leq i\leq N}$, factored into elementary translations δT_i and rotations δR_i. These transformations are expressed with a 6-dimensional linear parametrization $(\delta T_i, \delta R_i) \mapsto (t_x, t_y, t_z, h_1, h_2, h_3)^T$, based on exponential maps for the rotation parameters h_k.

6.4.2 Body parts

Each aligned mesh $\bar{\mathcal{M}}_i$ is next decomposed into P body part submeshes $\{\mathcal{P}_i^k\}_{1\leq k\leq P}$. We use $P = 5$ parts in a tree structured hierarchy including a torso as the root, and a pair of arms and legs as children nodes. We adopt the coarse and overlapping body segmentation strategy proposed previously in chapter 5. As illustrated in figure 6.2, the user provides closed and non-intersecting curves that delimit each overlapping region between contiguous body parts on $\mathcal{M}$. During motion learning and prediction, each body part is augmented with the overlap regions shared with its neighbouring parts. The sampled output body part motions are merged in the end with an automatic algorithm that ensures seamless body part stitching as described in section 6.6.2. Considering body parts independently allows for learning their deformations more accurately, enriches variation in the finally stitched body mesh, and also reduces the dimensionality of our model inputs. Increased complexity for large input partial meshes can be compensated by adopting more body parts with fewer vertices, or any means of dimension reduction such as PCA.

Each body part $\mathcal{P}_i^k$ is finally represented with a vector $\mathbf{y}_i$ that stacks successively the three coordinates of every vertex in the part. The torso vector is appended additionally with the 6 global mesh displacement parameters.

6.5 Shape motion embedding

We learn a probabilistic low dimensional embedding of motion for each body part independently using the Gaussian Process Dynamical Model Wang et al. [2008b]. We build latent spaces of body part motion using temporally pre-aligned motion sequences that are logically compatible, such as locomotion movements with various speeds and directions, or jumping movements with varying heights and lengths.

6.5.1 Non-linear dynamic model

For a body part motion sequence $\{\mathbf{y}_i\}_{1\leq i\leq N}$, the GPDM comprises a non-linear probabilistic mapping f from the latent variables $\mathbf{x}_i \in \mathbb{R}^d$ to the meshes $\mathbf{y}_i \in \mathbb{R}^D$, parametrized with coefficients $\mathbf{A}$, and another mapping g between latent coordinates of consecutive frames, parametrized with coefficients $\mathbf{B}$:

$$\mathbf{y}_i = f(\mathbf{x}_i, \mathbf{A}) + \mathbf{n}_{y,i} \tag{6.1}$$
$$\mathbf{x}_i = g(\mathbf{x}_{i-1}, \mathbf{B}) + \mathbf{n}_{x,i} \tag{6.2}$$

where $\mathbf{n}_{x,i}$ and $\mathbf{n}_{y,i}$ are zero-mean white Gaussian noises.

Marginalizing over $\mathbf{A}$ MacKay [2003], Neal [2012] with isotropic Gaussian priors on its parameters yields the likelihood:

$$P(\mathbf{Y}|\mathbf{X}, \boldsymbol{\alpha}) = \frac{1}{\sqrt{(2\pi)^{ND}|\mathbf{K}_Y|^D}} \exp\left(-\frac{1}{2}\text{tr}\left(\mathbf{K}_Y^{-1}\mathbf{Y}\mathbf{Y}^T\right)\right) \tag{6.3}$$

where $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_N]^T$ is the matrix of training shape poses, $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T$ is the matrix of the associated latent positions. This likelihood is expressed using the kernel matrix $\mathbf{K}_Y$ whose coefficients $(\mathbf{K}_Y)_{1\leq i,j\leq N} = k_Y(\mathbf{x}_i, \mathbf{x}_j)$ are defined using the Radial Basis Function MacKay [2003]:

$$k_Y(\mathbf{x}, \mathbf{x}') = \alpha_1 \exp\left(-\frac{\alpha_2}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right) + \frac{\delta_{\mathbf{x},\mathbf{x}'}}{\alpha_3} \tag{6.4}$$

Hyperparameter vector $\boldsymbol{\alpha}$ comprises kernel parameters α_k and the variance of the additive noise $\mathbf{n}_{y,i}$.

Similarly, the density over latent trajectories can be obtained by marginalizing out $\mathbf{B}$ Wang et al. [2006] with isotropic Gaussian priors on its parameters:

$$P(\mathbf{X}|\boldsymbol{\beta}) = \frac{P(\mathbf{x}_1)}{\sqrt{(2\pi)^{(N-1)d}|\mathbf{K}_X|^d}} \exp\left(-\frac{1}{2}\text{tr}\left(\mathbf{K}_X^{-1}\mathbf{X}'\mathbf{X}'^T\right)\right) \tag{6.5}$$

where $\mathbf{X}' = [\mathbf{x}_2, \ldots, \mathbf{x}_N]^T$ and $\mathbf{x}_1$ is given an isotropic Gaussian prior. This joint probability is expressed using the kernel matrix $\mathbf{K}_X$ whose coefficients $(\mathbf{K}_X)_{1\leq i,j\leq N-1} = k_X(\mathbf{x}_i, \mathbf{x}_j)$ are defined using the linear and Radial Basis kernel:

$$k_X(\mathbf{x}, \mathbf{x}') = \beta_1 \exp\left(-\frac{\beta_2}{2}\|\mathbf{x} - \mathbf{x}'\|^2\right) + \beta_3\mathbf{x}^T\mathbf{x}' + \frac{\delta_{\mathbf{x},\mathbf{x}'}}{\beta_4} \tag{6.6}$$

Hyperparameter vector $\boldsymbol{\beta}$ comprises kernel parameters β_k and the variance of the additive noise $\mathbf{n}_{x,i}$.

The latent variables $\mathbf{X}$ and the hyperparameters $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ are estimated by minimizing the negative log of the posterior:

$$P(\mathbf{X}, \boldsymbol{\beta}, \boldsymbol{\alpha}|\mathbf{Y}) \propto P(\mathbf{Y}|\mathbf{X}, \boldsymbol{\alpha})P(\mathbf{X}|\boldsymbol{\beta})P(\boldsymbol{\alpha})P(\boldsymbol{\beta}) \tag{6.7}$$

Priors that favour small output scale are adopted for the hyperparameters: $P(\boldsymbol{\alpha}) = \Pi_k \alpha_k^{-1}$, $P(\boldsymbol{\beta}) = \Pi_k \beta_k^{-1}$. In our experiments, hyperparameters α_k and β_k are initialized with value 1, variances of noises $\mathbf{n}_{y,i}$ and $\mathbf{n}_{x,i}$ are initialized with value 0.1, and latent positions $\mathbf{X}$ are initialized with Principal Component Analysis coordinates. The latent space dimension is set to $d = 3$.

6.5.2 Multiple sequences

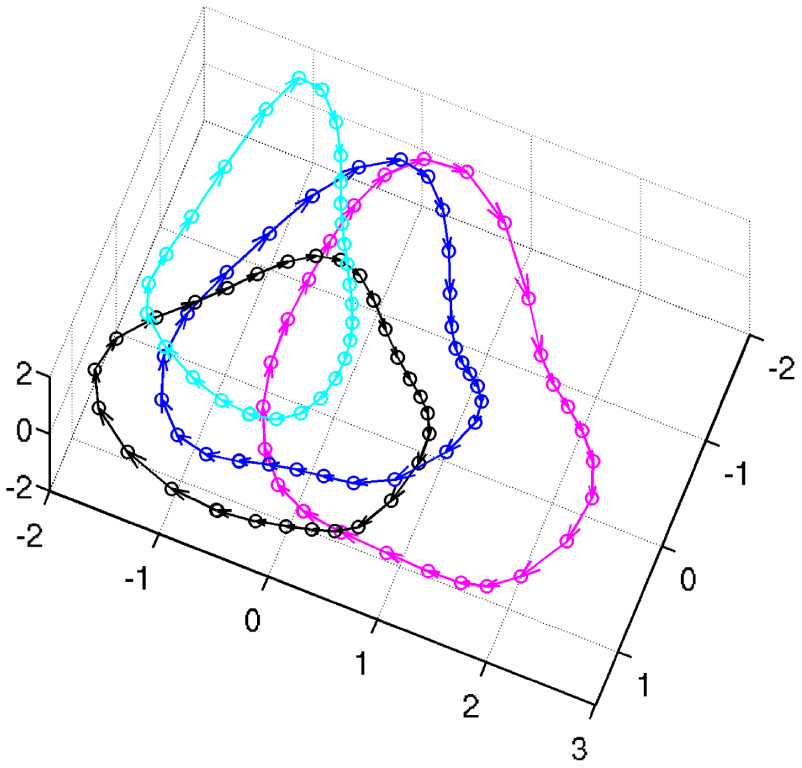


Figure 6.3 – Latent trajectories of the right leg, learned with sequences *Walk* (blue), *Jog* (black), *Left* (cyan) and *Right* (magenta).

The GPDM can naturally be extended to multiple motion sequences. After aligning them temporally to one reference sequence using Dynamic

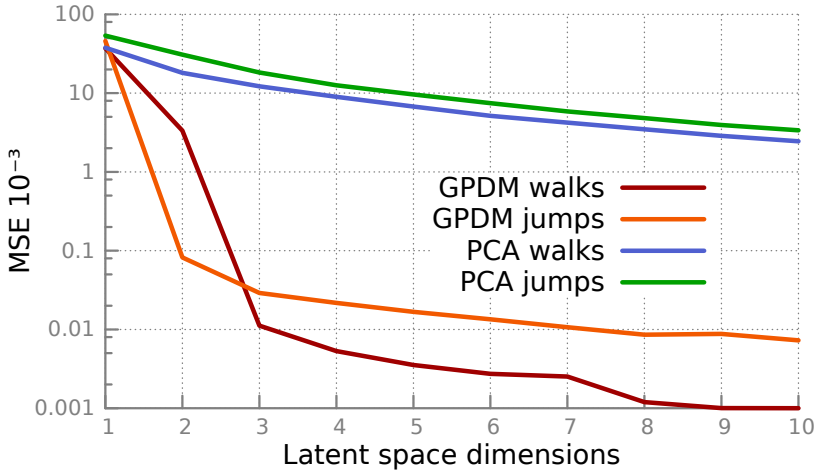


Figure 6.4 – Mean reconstruction error of the *"walk"* GPDM (built with sequences *Walk*, *Jog*, *Left*, *Right*) and the *"jumps"* GPDM (built with sequences *Jump long*, *Short*, *High*, *Low*), compared to PCA.

Time Warping Müller [2007], we concatenate the M training motion sequences $\mathbf{Y} = [\mathbf{Y}_1^T, \ldots, \mathbf{Y}_M^T]^T$. The associated latent position sequences $\{\mathbf{X}_j\}$ share the same latent space, $\mathbf{X}'$ comprises all but the first latent position for each sequence, and kernel matrix $\mathbf{K}_X$ is computed from all but the last latent position of each sequence. Figure 6.3 shows 4 latent trajectories of locomotion sequences of the right leg body part from DAN Casas et al. [2014b] dataset. Figure 6.4 shows vertex reconstruction errors of the training mesh sequences from two GPDMs built with walking and jumping movements from the same dataset, compared to PCA.

6.6 Shape motion sampling

Following the work of Wang et al. [2008b] on skeletal Motion Capture data, we synthesise variants of a motion sequence $\mathbf{Y}_j$ in the observation space by sampling variants of its subjacent trajectory $\mathbf{X}_j = [\mathbf{x}_1, \ldots, \mathbf{x}_N]$ using a Markov Chain Monte Carlo method in the latent space, initialized with a mean prediction sequence. Mesh motion sequences are finally obtained from latent motion samples through the Gaussian Process of pose reconstruction.

6.6.1 Mean prediction

We start from the initial position $\mathbf{x}_1$ of the latent trajectory and simulate the dynamical process one frame at a time until we reach the original trajectory length N using mean prediction, that is the density over a latent prediction $\hat{\mathbf{x}}_i$ conditioned on the previous one $\hat{\mathbf{x}}_{i-1}$ is Gaussian MacKay [2003]:

$$\hat{\mathbf{x}}_i \sim \mathcal{N}(\mu_X(\hat{\mathbf{x}}_{i-1}), \sigma_X^2(\hat{\mathbf{x}}_{i-1})\mathbf{I}) \tag{6.8}$$

$$\mu_X(\mathbf{x}) = \mathbf{X}'^T\mathbf{K}_X^{-1}\mathbf{k}_X(\mathbf{x}) \tag{6.9}$$

$$\sigma_X^2(\mathbf{x}) = \mathbf{k}_X(\mathbf{x},\mathbf{x}) - \mathbf{k}_X(\mathbf{x})^T\mathbf{K}_X^{-1}\mathbf{k}_X(\mathbf{x}) \tag{6.10}$$

where vector $\mathbf{k}_X(\mathbf{x})$ contains $k_X(\mathbf{x},\mathbf{x}_i)$ in its i-th entry, and $\mathbf{x}_i$ is the i-th training vector.

6.6.2 Latent sampling

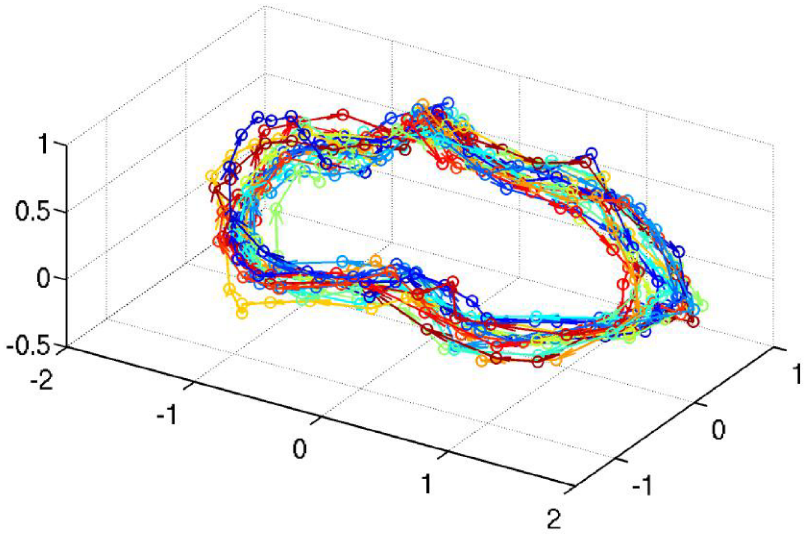


Figure 6.5 – 20 latent variations of sequence *Walk* for the torso.

Next, we use the resulting mean prediction sequence $\hat{\mathbf{X}} = [\hat{\mathbf{x}}_1, \ldots, \hat{\mathbf{x}}_n]^T$ to initialize a Markov Chain that draws multiple fair samples of entire trajectories $\tilde{\mathbf{X}} \sim P(\hat{\mathbf{X}}|\mathbf{x_1}, \mathbf{X}, \mathbf{Y}, \boldsymbol{\alpha}, \boldsymbol{\beta})$ offline using Hybrid Monte Carlo MacKay [2003]. Figure 6.5 shows 20 variations, each with a different color, of the latent trajectory of the torso body part from sequence *Walk* of DAN dataset. The latent space was learned with Locomotion sequences *walk*, *Jog*, *Left* and *Right* of this very dataset.

Pose reconstruction Finally for a latent position $\tilde{\mathbf{x}}$ in a sampled trajectory $\tilde{\mathbf{X}}$, the corresponding shape $\tilde{\mathbf{y}}$ is obtained by sampling from the predictive distribution of the Gaussian Process of pose reconstruction MacKay [2003]:

$$\tilde{\mathbf{y}} \sim \mathcal{N}(\mu_Y(\tilde{\mathbf{x}}), \sigma_Y^2(\tilde{\mathbf{x}})\mathbf{I}) \tag{6.11}$$

$$\mu_Y(\mathbf{x}) = \mathbf{Y}^T\mathbf{K}_Y^{-1}\mathbf{k}_Y(\mathbf{x}) \tag{6.12}$$

$$\sigma_Y^2(\mathbf{x}) = \mathbf{k}_Y(\mathbf{x},\mathbf{x}) - \mathbf{k}_Y(\mathbf{x})^T\mathbf{K}_Y^{-1}\mathbf{k}_Y(\mathbf{x}) \tag{6.13}$$

where vector $\mathbf{k}_Y(\mathbf{x})$ contains $k_Y(\mathbf{x},\mathbf{x}_i)$ in its i-th entry, and $\mathbf{x}_i$ is the i-th training vector.

After predicting all body parts $\{\tilde{\mathcal{P}}\}_{1\leq k\leq P}$ and global rigid displacement parameters (i.e. $\{\tilde{\mathbf{y}}^k\}_{1\leq k\leq P}$) for a given pose (i.e frame), we adopt the strategy proposed in Boukhayma et al. [2017] to stitch the body mesh back together automatically. We rigidly align each child part to the root part with respect to the overlap region, then we find the closed curve with the least deformation cost between its two instantiations in the root and child geometries, and use it as an optimal boundary for Poisson mesh merging Huang et al. [2007c] Yu et al. [2004b]. This process is reiterated for all overlap regions to recover the fully merged body mesh, which is positioned subsequently according the current global rigid displacement prediction.

6.7 Shape motion parametrization

Once we have learned the motion latent space, we can synthesise infinite variations of any sequence used to build this space using the method detailed in section 6.6. In this section, we present a method that allows variation synthesis for any blended version of logically compatible pair of input sequences, without the need for costly non-linear interpolation of many input sequence variations in the mesh domain.

Given a pair $\mathbf{Y}_1$ and $\mathbf{Y}_2$ of input sequences exhibiting variations of a common movement, such as a pair of walking and running sequences which both represent locomotion but with two different speeds, seminal work on mesh animation Casas et al. [2014b] proposes to synthesise new intermediate sequences of the same movement, ranging from the first motion $\mathbf{Y}_1$ to the second motion $\mathbf{Y}_2$, by blending the input sequence pair with weights w varying between 0 and 1: $\mathbf{Y}_w = (1-w)\mathbf{Y}_1 + w\mathbf{Y}_2$. However, for a given parameter value w, only one unique interpolated sequence

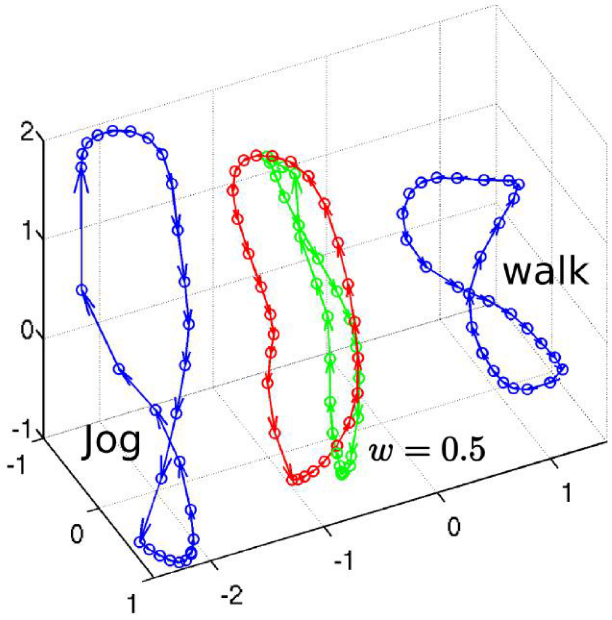
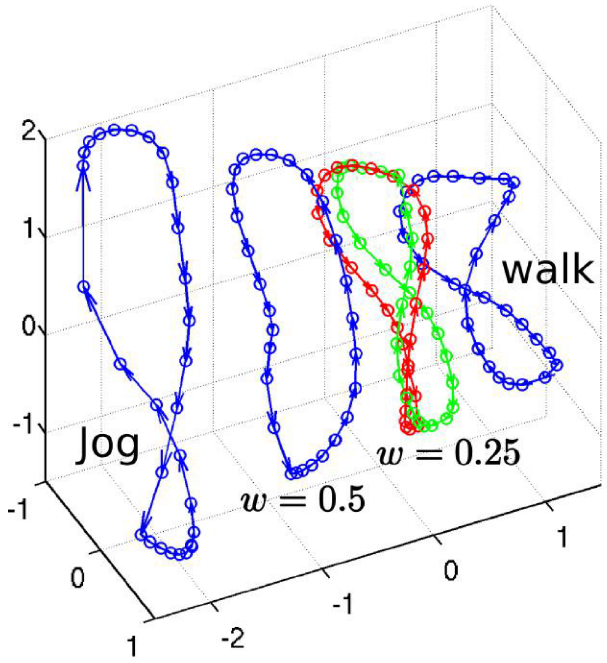


Figure 6.6 – In green: Interpolation of *Walk* (blue) and *Jog* (blue) in latent space. In red: Latent trajectory of non-linearly blended *Walk* and *Jog* in mesh domain.

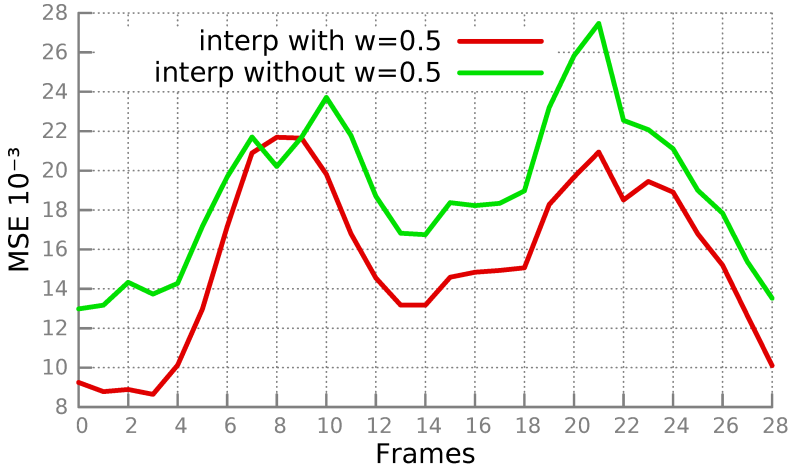
$\mathbf{Y}_w$ is obtained. Our goal is to perform a similar motion parametrization scheme, but in addition, we want to be able to generate infinite variations for each blended motion with a given parameter w.

One solution is to generate variations $\{\tilde{\mathbf{Y}}_1^l\}$ and $\{\tilde{\mathbf{Y}}_2^m\}$ of input sequence pair $\mathbf{Y}_1$ and $\mathbf{Y}_2$, and each element in the set of blended variations in the observation space $\{(1-w)\tilde{\mathbf{Y}}_1^l + w\tilde{\mathbf{Y}}_2^m\}_{l,m}$ could be considered as a variant of $\mathbf{Y}_w$. However this requires performing mesh non linear interpolation of a big number of sequence pairs for visually plausible outputs, since linear interpolation results in mesh distortions. The work of Casas et al. [2013] elaborates on the time cost of non-linear mesh interpolation, which can be achieved using standard algorithms such as Xu et al. [2006]. For interpolating meshes from DAN Casas et al. [2014b] dataset for instance, these approaches cost 3 orders of magnitude more than linear interpolation, which makes them in turn unfit for real time synthesis.

As an alternative solution, we could consider interpolation in the latent space. That is, we could attempt to generate variations of the blended trajectory $\mathbf{X}_w = (1-w)\mathbf{X}_1 + w\mathbf{X}_2$ in the latent space. Conversely, we could generate variations $\{\tilde{\mathbf{X}}_1^l\}$ and $\{\tilde{\mathbf{X}}_2^m\}$ of the latent trajectories $\mathbf{X}_1$ and $\mathbf{X}_2$, and each element in the set of blended variations in the latent space



(a) Obtaining latent sequence $w = 0.25$ by: (green) interpolating sequences *Walk* and *Jog*, (red) interpolating nearby sequences *Walk* and $w = 0.5$.



(b) Mean vertex distance between non-linearly interpolated mesh sequence $w = 0.25$ (ground-truth) and: (green) reconstructing interpolation of latent sequences *Walk* and *Jog*, (red) reconstructing interpolation of nearby latent sequences *Walk* and $w = 0.5$.

Figure 6.7 – Interpolation in latent space with and without intermediate learned latent trajectories.

$\{(1-w)\tilde{\mathbf{X}}_1^l + w\tilde{\mathbf{X}}_2^m\}_{l,m}$ could be considered as a variant of $\mathbf{X}_w$. However, these two strategies suffer from the two following shortcomings:

- Blended motions in the observation space are not necessarily represented by interpolated trajectories with the same proportions in the latent space. For instance in figure 6.6, we build a GPDM with 5 sequences: *Walk*, *Jog*, *Left*, *Right*, and a blended version of sequences *Walk* and *Jog* with $w = 0.5$. We notice that this sequence's latent trajectory (in red) does not coincide with the linear interpolation of *Walk* and *Jog* trajectories (in green) in the latent space, due to the over representation of walking like poses (*Walk*, *Left* and *Right*) in the training set in this example for instance.
- In our experiments, sampling variations from a latent trajectory that was not learned with the GPDM model, such as an interpolated trajectory $\mathbf{X}_w$, usually results in degenerate sequences. Hence we limit sampling initialization to trajectories of real sequences that took part of the training process.

In light of these observations, we propose the following solution to overcome the limitations above. We perform blending offline for few discriminative weighting values, e.g. $w \in W = \{0.25, 0.5, 0.75\}$. In our experiments, one intermediate value was enough ($W = \{0.5\}$) to obtain visually compelling results. The GPDM is learned with both the original sequences $\mathbf{Y}_1, \mathbf{Y}_2, \ldots, \mathbf{Y}_m$ and the blended sequences $\{\mathbf{Y}_w\}_{w \in W}$. Variations are generated for latent trajectories of all of these sequences offline. Then, for a new requested blend value $w^* \in [0, 1]$ we can obtain variations of sequence $\mathbf{Y}_{w^*}$ online as follows: First, we find the closest bounding values of w^* in W: $w^* \in [w_1, w_2]$ where $\mathbf{X}_{w=0} := \mathbf{X}_1$ and $\mathbf{X}_{w=1} := \mathbf{X}_2$. Second, we select two variants of the corresponding latent trajectories $\tilde{\mathbf{X}}_{w_1}^l$ and $\tilde{\mathbf{X}}_{w_2}^m$ randomly and blend them accordingly : $\tilde{\mathbf{X}}_{w^*} = \frac{w^*-w_2}{w_1-w_2}\tilde{\mathbf{X}}_{w_1}^l + \frac{w^*-w_1}{w_2-w_1}\tilde{\mathbf{X}}_{w_2}^m$. Finally, we generate an example shape sequence $\tilde{\mathbf{Y}}_{w^*}$.

Figure 6.7 shows that guiding interpolation of latent trajectories with nearby learned intermediate trajectories results in good approximations of non-linearly interpolated mesh sequences. In this particular example, we learn a GPDM with sequences *Walk*, *Jog*, *Left*, *Right* and a blended version of sequences *Walk* and *Jog* with $w = 0.5$. As shown in figure 6.7a, we generate a sequence $\mathbf{X}_{w=0.25}$ both from interpolating latent trajectories $\mathbf{X}_1$ (*Walk*) and $\mathbf{X}_2$ (*Jog*) that we plot in red, and interpolating nearby trajectories $\mathbf{X}_1$ and $\mathbf{X}_{w=0.5}$ that we plot in green. We realise that the reconstruction of the interpolation that uses intermediate sequence $\mathbf{X}_{w=0.5}$ is closer, in vertex distance, to the non-linearly blended sequence $\mathbf{Y}_{w=0.25}$ in the observation space, that we consider to be ground truth, as shown in figure 6.7b.

(a) Jumping long.

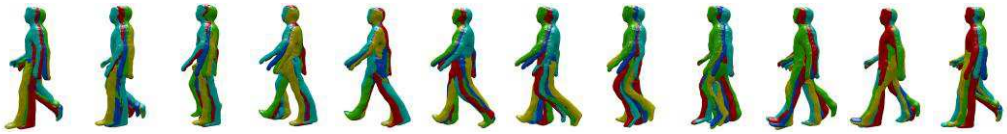
(b) $(1 - w)$ Jumping short + w Jumping long, $w = 0.5$.

(c) Jumping short.

Figure 6.8 – 5 overlapped Jumping variations, each with a different color.

(a) Jogging.

(b) $(1 - w)$ Walking + w Jogging, $w = 0.5$.



(c) Walking.

Figure 6.9 – 5 overlapped locomotion variations, each with a different color.

6.8 Results

We use DAN surface motion capture dataset Casas et al. [2014b] to evaluate our method. Meshes have 2667 vertices and 5330 faces and motions are recorded at 25 fps. We build two GPDMs using locomotion sequences and jumping sequences respectively.

Figure 6.8 shows results from the GPDM of jumping movements, which comprises sequences *Short jump*, *Long jump*, *Low jump*, *High jump*. All sequences are temporally aligned to *Short jump* with dynamic time warping guided with feet contact annotation. To demonstrate motion parametrization, we interpolate sequences *Short jump* and *Long jump* non-linearly with $w = 0.5$ and add this interpolated mesh sequence to the training sequences. As a result, we can generate variations of sequences *Short jump*, *Long jump*, *Low jump*, *High jump* and blended versions of sequences *Short jump* and *Long jump* with any weighting proportions. We show in this figure variations of training sequences and also blended sequences.

In figure 6.9, we show results from the GPDM of locomotion movements, which comprises sequences *Walk*, *Jog*, *Left turn*, *Right turn*. All sequences are temporally aligned to *Walk* with dynamic time warping guided with feet contact annotation. To demonstrate motion parametrization, we interpolate sequences *Walk* and *Jog* non-linearly with $w = 0.5$ and add this interpolated mesh sequence to the training sequences. As a result, we can generate variations of sequences *Walk*, *Jog*, *Left turn*, *Right turn* and blended versions of sequences *Walk* and *Jog* with any weighting proportions. We show in this figure variations of training sequences and also blended sequences.

As we can see in figures 6.8 and 6.9 and this video, the output variations are logically similar but slightly different from each other in poses and timings. The differences between the variations are big enough to be noticed by users, but still conserve the main characteristics of the base movement. The realism of the generated sequences matches that of the input ones, as poses and dynamics are overall sound and coherent.

6.9 Conclusion

We presented in this chapter a solution for generating infinite variations of a subject movement using few training sequences of surface motion capture, based on Gaussian Process Dynamical Models. We also contributed an algorithm that allows synthesis of variations for any blended

version of the input sequences without costly non-linear interpolation of many motion sequence variations in mesh domain. While the differences between a movement variations are easily noticeable, these generated motions are mostly visually plausible and match the realism level of the input sequences. As a next step, this work can be extended to model and synthesis both shape and appearance variation.

Chapter 7

Conclusions

Contents

7.1 Summary

In this thesis, we contributed solutions to some of the tasks related to the acquisition and exploitation of 4D surface capture data as obtained from multi-view set-ups, with a focus on corpus of moving humans. We obtain our temporally consistent 4D textured models through surface reconstruction and tracking, and then texture reprojection from the input views into atlas consistent appearance maps.

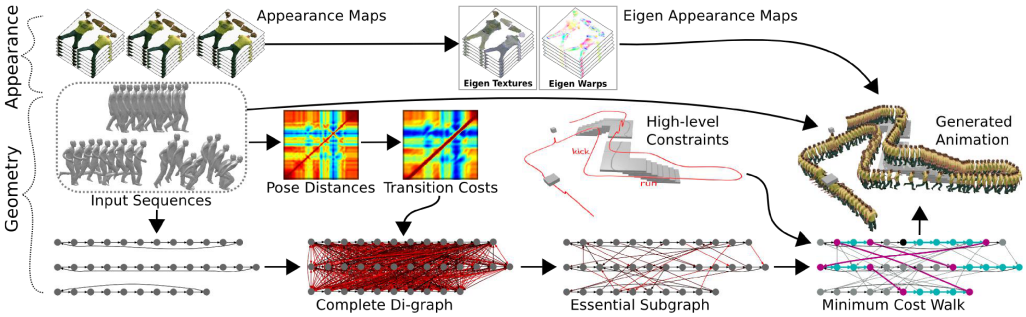


Figure 7.1 – Shape and appearance animation pipeline

Combining chapters 2, 3 and 4, we contributed an animation synthesis pipeline that takes as input temporally coherent meshes of elementary motions and their texture maps and outputs animations satisfying high-level

user constraints. As illustrated in figure 7.1, we start by computing shape pose distances between the dataset frames and then we use those to obtain an evaluation of dynamic transition costs between the dataset frames as elaborated in chapter 2. A graph representation of the input sequences is then augmented with additional optimal transitions with their respective costs, thus forming our animation organizing structure, the essential graph, which was introduced in chapter 3. Finally, a minimal cost walk in the essential graph with respect to the user guidance generates the desired output mesh sequence. Theses meshes are appended with appearances through interpolations within our subject-specific appearance representation introduced in chapter 4.

In chapter 4, we approached dynamic object appearance modelling from multiple views with a twofold strategy that solves the non-linearly of the problem first through 3D coarse geometric registration, then by factoring out the fine scale geometric inaccuracies through 2D realignment of appearances in the common texture map domain. The appearance of the model can then be compactly represented through the linear reduction of the realignment warps and the aligned appearances both through PCA.

In chapter 5, we addressed the problem of transferring motion between captured 4D models. Given 4D training sets for two subjects for which a sparse set of corresponding key-poses are known, we were able to transfer newly captured motions from one subject to the other. We built on Gaussian Process and used non-linear pose and displacement regression directly on mesh data without the need for source/target vertex correspondence, or any intermediate pose representation such as skeletons or control points.

In chapter 6, we addressed the problem of generating variations of captured 4D models automatically, in order to mimic human motion richness and inexactitude while performing the same action repeatedly, thus resulting in improved realism and augmented datasets with more motion samples. We built on Gaussian Process Dynamical models to model motion sequences and sample new variants of the input motions and any blended versions of them.

7.2 Future work

Regarding 4D animation synthesis, the quality of the rendered animations is limited by the quality of the 4D surface capture process, including surface reconstruction, surface tracking and appearance reconstruction. Although the Vision and Graphics state of the art is very promising with

this respect, we still have room to improve. Besides, popularizing this animation process and convincing the industrials of the utility of such data is still a work in progress, but we believe that with the emergence of Virtual Reality and Augmented Reality applications, such solutions will be of great use for the community.

This research aims also to ease the burden of surface motion capture and allow automatic synthesis of 3D textured shapes in motion with high realism. In fact, we proposed some attempts at augmenting 4D datasets through motion transfer and also variation synthesis. Once 4D datasets are enlarged enough through more capture and such robust automatic generalization methods, deep learning animation frameworks, such as the work of Holden et al. [2016] on motion capture data, might be attempted also on 4D data.

Bibliography

B. Allain, J.-S. Franco, and E. Boyer. An efficient volumetric framework for shape tracking. In *CVPR*, 2015. Cited on pages 4 and 78.

O. Arikan and D. A. Forsyth. Interactive motion generation from examples. *ACM Trans. Graph.*, 21(3):483–490, 2002. Cited on pages 11 and 49.

M. Banerjee, R. Chakraborty, E. Ofori, M. S. Okun, D. E. Viallancourt, and B. C. Vemuri. A nonlinear regression technique for manifold valued data with applications to medical image analysis. In *CVPR*, 2016. Cited on page 120.

I. Baran, D. Vlasic, E. Grinspun, and J. Popović. Semantic deformation transfer. In *ACM SIGGRAPH*, 2009. Cited on pages 102, 104, 105, 108, and 117.

A. H. Barr. Global and local deformations of solid primitives. In *ACM SIGGRAPH Papers*, 1984. Cited on page 14.

M. Ben-Chen, O. Weber, and C. Gotsman. Spatial deformation transfer. In *SCA*, 2009. Cited on page 103.

V. Blanz and T. Vetter. A morphable model for the synthesis of 3d faces. In *ACM SIGGRAPH*, 1999. Cited on pages 77 and 78.

B. Bodenheimer, A. V. Shleyfman, and J. K. Hodgins. The effects of noise on the perception of animated human running. In *Computer Animation and Simulation*, volume 99, 1999. Cited on page 123.

A. Boukhayma and E. Boyer. Video based Animation Synthesis with the Essential Graph. In *3DV*, 2015. Cited on pages 4, 6, 78, 115, and 122.

A. Boukhayma, V. Tsiminaki, J.-S. Franco, and E. Boyer. Eigen Appearance Maps of Dynamic Shapes. In *ECCV*, 2016. Cited on pages 4 and 6.

A. Boukhayma, J.-S. Franco, and E. Boyer. Surface motion capture transfer with gaussian process regression. In *CVPR*, 2017. Cited on page 131.

H. M. Briceño, P. V. Sander, L. McMillan, S. Gortler, and H. Hoppe. Geometry videos: A new representation for 3d animations. In *Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '03, pages 136–146, 2003. ISBN 1-58113-659-5. Cited on page 12.

A. Bruderlin and L. Williams. Motion signal processing. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 97–104, 1995. ISBN 0-89791-701-4. Cited on page 34.

C. Budd, P. Huang, M. Klaudiny, and A. Hilton. Global non-rigid alignment of surface sequences. *Int. J. Comput. Vision*, 102(1-3), 2013. Cited on pages 4 and 6.

C. Cagniart, E. Boyer, and S. Ilic. Free-from mesh tracking: a patch-based approach. In *CVPR*, 2010. Cited on page 77.

J. Carranza, C. Theobalt, M. A. Magnor, and H.-P. Seidel. Free-viewpoint video of human actors. *ACM Trans. Graph.*, 2003. Cited on page 77.

D. Casas, M. Tejera, J. Guillemaut, and A. Hilton. Parametric control of captured mesh sequences for real-time animation. In *Motion in Games - 4th International Conference*, 2011a. Cited on pages 40 and 51.

D. Casas, M. Tejera, J.-Y. Guillemaut, and A. Hilton. Parametric control of captured mesh sequences for real-time animation. In *MIG*, 2011b. Cited on page 124.

D. Casas, M. Tejera, J.-Y. Guillemaut, and A. Hilton. 4d parametric motion graphs for interactive animation. In *in Proc. of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. ACM, 2012. Cited on pages 13, 51, 52, 54, and 78.

D. Casas, M. Tejera, J.-Y. Guillemaut, and A. Hilton. Interactive animation of 4d performance capture. *TVCG*, 19(5), 2013. Cited on page 132.

D. Casas, M. Volino, J. Collomosse, and A. Hilton. 4d video textures for interactive character appearance. *Computer Graphics Forum (Proceedings Eurographics)*, 33(2), 2014a. Cited on pages 4 and 115.

D. Casas, M. Volino, J. Collomosse, and A. Hilton. 4D Video Textures for Interactive Character Appearance. *Computer Graphics Forum (Proc. of Eurographics)*, 2014b. Cited on pages 6, 78, 122, 124, 129, 131, 132, and 136.

D. Casas, C. Richardt, J. Collomosse, C. Theobalt, and A. Hilton. 4D Model Flow: Precomputed Appearance Alignment for Real-time 4D Video Interpolation. *Computer Graphics Forum*, 2015. Cited on page 78.

T. Chen, J.-Y. Zhu, A. Shamir, and S.-M. Hu. Motion-aware gradient domain video composition. *IEEE Transactions on Image Processing*, 2013. Cited on pages 83 and 105.

A. Collet, M. Chuang, P. Sweeney, D. Gillett, D. Evseev, D. Calabrese, H. Hoppe, A. Kirk, and S. Sullivan. High-quality streamable free-viewpoint video. *ACM Trans. Graph.*, 2015a. Cited on pages 77 and 78.

A. Collet, M. Chuang, P. Sweeney, D. Gillett, D. Evseev, D. Calabrese, H. Hoppe, A. Kirk, and S. Sullivan. High-quality streamable free-viewpoint video. *ACM Trans. Graph.*, 34(4), 2015b. Cited on page 3.

T. F. Cootes, G. J. Edwards, and C. J. Taylor. Active appearance models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2001. Cited on page 78.

E. de Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.-P. Seidel, and S. Thrun. Performance capture from sparse multi-view video. *ACM Trans. Graph.*, 2008. Cited on page 77.

P. E. Debevec, C. J. Taylor, and J. Malik. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *ACM SIGGRAPH*, 1996. Cited on page 77.

M. Dou, S. Khamis, Y. Degtyarev, P. Davidson, S. R. Fanello, A. Kowdle, S. O. Escolano, C. Rhemann, D. Kim, J. Taylor, P. Kohli, V. Tankovich, and S. Izadi. Fusion4d: Real-time performance capture of challenging scenes. *ACM Trans. Graph.*, 35(4), 2016. Cited on page 4.

M. Eisemann, B. De Decker, M. Magnor, P. Bekaert, E. de Aguiar, N. Ahmed, C. Theobalt, and A. Sellent. Floating textures. *Computer Graphics Forum (Proc. of Eurographics)*, 2008. Cited on page 77.

W.-W. Feng, B.-U. Kim, and Y. Yu. Real-time data driven deformation using kernel canonical correlation analysis. *ACM Trans. Graph.*, 27(3), 2008. Cited on pages 102 and 103.

J.-S. Franco and E. Boyer. Efficient Polyhedral Modeling from Silhouettes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2009. Cited on page 4.

J. Gall, C. Stoll, E. De Aguiar, C. Theobalt, B. Rosenhahn, and H.-P. Seidel. Motion capture using joint skeleton tracking and surface estimation. In *Proc. of CVPR*, 2009. Cited on page 13.

M. Gleicher, H. J. Shin, L. Kovar, and A. Jepsen. Snap-together motion: assembling run-time animations. In *ACM SIGGRAPH 2008 classes*, 2008. Cited on page 124.

B. Goldlücke, M. Aubry, K. Kolev, and D. Cremers. A super-resolution framework for high-accuracy multiview reconstruction. *International Journal of Computer Vision*, 2014. Cited on page 77.

K. Grochow, S. L. Martin, A. Hertzmann, and Z. Popović. Style-based inverse kinematics. *ACM TOG*, 23(3), 2004. Cited on page 124.

C. M. Harris and D. M. Wolpert. Signal-dependent noise determines motor planning. *Nature*, 394(6695), 1998. Cited on page 123.

R. Heck and M. Gleicher. Parametric motion graphs. In *Proc. of the 2007 Symposium on Interactive 3D Graphics and Games*, 2007a. Cited on pages 40 and 52.

R. Heck and M. Gleicher. Parametric motion graphs. In *ACM I3D*, 2007b. Cited on page 124.

D. Holden, J. Saito, T. Komura, and T. Joyce. Learning motion manifolds with convolutional autoencoders. In *SIGGRAPH Asia 2015 Technical Briefs*, 2015. Cited on page 51.

D. Holden, J. Saito, and T. Komura. A deep learning framework for character motion synthesis and editing. *ACM Transactions on Graphics (TOG)*, 35(4), 2016. Cited on pages 51 and 141.

P. Huang, J. Starck, and A. Hilton. Temporal 3d shape matching. In *Visual Media Production, 2007. IETCVMP. 4th European Conference on*, 2007a. Cited on page 13.

P. Huang, A. Hilton, and J. Starck. Human motion synthesis from 3d video. In *Proc. of CVPR*, 2009. Cited on pages 10, 13, 49, and 52.

X. Huang, H. Fu, O. K.-C. Au, and C.-L. Tai. Optimal boundaries for poisson mesh merging. In *SPM*, 2007b. Cited on pages 105 and 115.

X. Huang, H. Fu, O. K.-C. Au, and C.-L. Tai. Optimal boundaries for poisson mesh merging. In *SPM*, 2007c. Cited on page 131.

D. Q. Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009. Cited on page 17.

L. Ikemoto, O. Arikan, and D. Forsyth. Quick transitions with cached multi-way blends. In *Proc. of the 2007 Symposium on Interactive 3D Graphics and Games*, 2007. Cited on page 51.

M. Innmann, M. Zollhöfer, M. Nießner, C. Theobalt, and M. Stamminger. Volumedeform: Real-time volumetric non-rigid reconstruction. In *ECCV*, 2016. Cited on page 4.

D. L. James, C. D. Twigg, A. Cove, and R. Y. Wang. Mesh ensemble motion graphs: Data-driven mesh animation with constraints. *ACM Trans. Graph.*, 26(4), 2007. Cited on page 12.

J. Jia, J. Sun, C.-K. Tang, and H.-Y. Shum. Drag-and-drop pasting. *ACM Trans. Graph.*, 2006. Cited on pages 83 and 105.

M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3 d shape descriptors. In *Symposium on geometry processing*, 2003. Cited on page 13.

L. Kovar and M. Gleicher. Flexible automatic motion blending with registration curves. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2003. Cited on pages 11 and 67.

L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM Trans. Graph.*, 23(3), 2004a. Cited on pages 40 and 51.

L. Kovar and M. Gleicher. Automated extraction and parameterization of motions in large data sets. *ACM TOG*, 23(3), 2004b. Cited on page 124.

L. Kovar, M. Gleicher, and F. Pighin. Motion graphs. *ACM Trans. Graph.*, 21(3):473–482, July 2002a. ISSN 0730-0301. Cited on pages 11, 13, 30, 40, and 48.

L. Kovar, J. Schreiner, and M. Gleicher. Footskate cleanup for motion capture editing. In *Proc. of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2002b. Cited on page 32.

M. Lau, Z. Bar-Joseph, and J. Kuffner. Modeling spatial and temporal variation in motion data. In *ACM Transactions on Graphics (TOG)*, volume 28, page 171, 2009. Cited on pages 122, 123, and 124.

N. D. Lawrence. Gaussian process latent variable models for visualisation of high dimensional data. In *NIPS*, 2004. Cited on page 124.

B. Le and Z. Deng. Smooth skinning decomposition with rigid bones. In *Proceedings of the 2012 SIGGRAPH Asia Conference*, SA '12, New York, NY, USA, 2012. ACM. Cited on page 12.

J. Lee, J. Chai, P. S. A. Reitsma, J. K. Hodgins, and N. S. Pollard. Interactive control of avatars animated with human motion data. *ACM Trans. Graph.*, 21(3), 2002. Cited on pages 11 and 49.

V. S. Lempitsky and D. V. Ivanov. Seamless mosaicing of image-based texture maps. In *CVPR*, 2007. Cited on page 77.

B. Lévy, S. Petitjean, N. Ray, and J. Maillot. Least squares conformal maps for automatic texture atlas generation. *ACM. Trans. Graph.*, 2002. Cited on page 4.

C. Linz, C. Lipski, and M. Magnor. Multi-image interpolation based on graph-cuts and symmetric optical flow, 2010. Posters. Cited on page 83.

L. Lucas, P. Souchet, M. Ismael, O. Nocent, C. Niquin, C. Loscos, L. Blache, S. Prévost, and Y. Remion. Recover3d: A hybrid multi-view system for 4d reconstruction of moving actors. In *4th International Conference and Exhibition on 3D Body Scanning Technologies*, page 219. Cited on page 4.

W. Ma, S. Xia, J. K. Hodgins, X. Yang, C. Li, and Z. Wang. Modeling style and variation in human motion. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 21–30, 2010. Cited on pages 122 and 124.

D. J. MacKay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003. Cited on pages 127, 130, and 131.

D. Mahajan, F.-C. Huang, W. Matusik, R. Ramamoorthi, and P. N. Belhumeur. Moving gradients: a path-based method for plausible image interpolation. *ACM Trans. Graph.*, 2009. Cited on page 83.

N. Mantel. The detection of disease clustering and a generalized regression approach. *Cancer research*, 27(2 Part 1):209–220, 1967. Cited on page 18.

A. Menache. *Understanding motion capture for computer animation and video games*. Morgan kaufmann, 2000. Cited on page 52.

S. Menardais, R. Kulpa, F. Multon, and B. Arnaldi. Synchronization for dynamic blending of motions. In *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, 2004. Cited on pages 11 and 31.

M. Meyer, M. Desbrun, P. Schrder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds, 2002. Cited on page 27.

J. Min and J. Chai. Motion graphs++: a compact generative model for semantic motion analysis and synthesis. *ACM Transactions on Graphics (TOG)*, 31(6), 2012. Cited on page 51.

M. Müller. *Information Retrieval for Music and Motion*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2007. Cited on pages 32, 110, and 129.

A. Mustafa, H. Kim, J.-Y. Guillemaut, and A. Hilton. Temporally coherent 4d reconstruction of complex dynamic scenes. In *CVPR*, 2016. Cited on page 4.

R. M. Neal. *Bayesian learning for neural networks*, volume 118. Springer Science & Business Media, 2012. Cited on page 127.

R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *CVPR*, 2015. Cited on page 4.

K. Nishino, Y. Sato, and K. Ikeuchi. Eigen-texture method: Appearance compression and synthesis based on a 3d model. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2001. Cited on pages 76, 77, and 78.

P. Pérez, M. Gangnet, and A. Blake. Poisson image editing. *ACM Trans. Graph.*, 2003. Cited on pages 83 and 105.

K. Perlin. Real time responsive animation with personality. *TVCG*, 1(1), 1995. Cited on page 123.

K. Polthier and E. Preuß. *Identifying vector field singularities using a discrete Hodge decomposition*. 2003. Cited on page 23.

F. Prada, M. Kazhdan, M. Chuang, A. Collet, and H. Hoppe. Motion graphs for unstructured textured meshes. *ACM Trans. Graph.*, 35(4), 2016. Cited on pages 4, 13, 49, and 122.

K. Pullen and C. Bregler. Animating by multi-level sampling. In *Computer Animation*, 2000. Cited on page 124.

K. Pullen and C. Bregler. Motion capture assisted animation: Texturing and synthesis. *ACM TOG*, 21(3), 2002. Cited on page 124.

C. E. Rasmussen and C. K. I. Williams. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press, 2005. Cited on page 112.

C. Ren, L. Zhao, and A. Safonova. Human motion synthesis with optimization-based graphs. *Comput. Graph. Forum*, 29(2), 2010. Cited on page 50.

H. Rhodin, J. Tompkin, K. I. Kim, V. Kiran, H.-P. Seidel, and C. Theobalt. Interactive motion mapping for real-time character control. *Computer Graphics Forum (Proceedings Eurographics)*, 33(2), 2014. Cited on pages 102 and 104.

C. Rose, B. Guenter, B. Bodenheimer, and M. F. Cohen. Efficient generation of motion transitions using spacetime constraints. In *Proc. of ACM SIGGRAPH*, 1996. Cited on page 11.

C. Rose, M. F. Cohen, and B. Bodenheimer. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Comput. Graph. Appl.*, 18(5): 32–40, Sept. 1998. ISSN 0272-1716. Cited on pages 9, 10, and 11.

A. Safonova and J. K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Trans. Graph.*, 26(3), July 2007. ISSN 0730-0301. Cited on pages 11, 49, and 50.

H. Sakoe and S. Chiba. Readings in speech recognition. chapter Dynamic Programming Algorithm Optimization for Spoken Word Recognition, pages 159–165. 1987. ISBN 1-55860-124-4. Cited on page 32.

H. J. Shin and H. S. Oh. Fat graphs: constructing an interactive character with continuous controls. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, 2006. Cited on page 52.

O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, pages 109–116, 2007. ISBN 978-3-905673-46-3. Cited on page 23.

O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl, and H.-P. Seidel. Laplacian surface editing. In *Proc. of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 2004. Cited on page 14.

J. Starck and A. Hilton. Surface capture for performance-based animation. *IEEE Comput. Graph. Appl.*, 27(3), 2007. Cited on page 6.

C. Stoll, J. Gall, E. de Aguiar, S. Thrun, and C. Theobalt. Video-based reconstruction of animatable human characters. *ACM Trans. Graph.*, 29 (6), 2010. Cited on page 6.

R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Trans. Graph.*, 23(3), 2004a. Cited on page 14.

R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. In *ACM SIGGRAPH*, 2004b. Cited on pages 102 and 103.

G. J. Székely, M. L. Rizzo, N. K. Bakirov, et al. Measuring and testing dependence by correlation of distances. *The Annals of Statistics*, 35(6): 2769–2794, 2007. Cited on page 21.

J. Snchez Prez, E. Meinhardt-Llopis, and G. Facciolo. TV-L1 Optical Flow Estimation. *Image Processing On Line*, 2013. Cited on page 82.

L. M. Tanco and A. Hilton. Realistic synthesis of novel human movements from a database of motion capture examples. In *Proc. of the Workshop on Human Motion (HUMO'00)*, 2000. Cited on page 50.

M. Tejera and A. Hilton. Learning part-based models for animation from surface motion capture. In *3DV*, 2013. Cited on pages 104 and 107.

J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, and M. Nießner. Face2face: Real-time face capture and reenactment of rgb videos. In *CVPR*, 2016. Cited on page 103.

Y. Tong, S. Lombeyda, A. N. Hirani, and M. Desbrun. Discrete multi-scale vector field decomposition. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 445–452, 2003. ISBN 1-58113-709-5. Cited on pages 23 and 27.

V. Tsiminaki, J.-S. Franco, and E. Boyer. High resolution 3d shape texture from multiple videos. In *CVPR*, 2014. Cited on pages 4, 77, and 89.

T. Tung. Simultaneous super-resolution and 3D video using graph-cuts. 2008. Cited on page 77.

M. Turk and A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 1991a. Cited on page 87.

M. Turk and A. Pentland. Eigenfaces for recognition. *J. Cognitive Neuroscience*, 1991b. Cited on page 78.

D. Vlasic, I. Baran, W. Matusik, and J. Popović. Articulated mesh animation from multi-view silhouettes. *ACM Transactions on Graphics*, 27(3), 2008a. Cited on pages 6 and 13.

D. Vlasic, I. Baran, W. Matusik, and J. Popović. Articulated mesh animation from multi-view silhouettes. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pages 97:1–97:9, New York, NY, USA, 2008b. ACM. ISBN 978-1-4503-0112-1. Cited on page 16.

A. Vögele, M. Hermann, B. Krüger, and R. Klein. Interactive steering of mesh animations. In *SCA*, 2012. Cited on pages 102 and 103.

M. Volino, D. Casas, J. Collomosse, and A. Hilton. Optimal representation of multiple view video. In *BMVC*, 2014. Cited on pages 4, 77, and 92.

M. Waechter, N. Moehrle, and M. Goesele. Let there be color! large-scale texturing of 3d reconstructions. In *ECCV*, 2014. Cited on page 77.

H. Wang, R. Raskar, and N. Ahuja. Seamless video editing. In *ICPR*, 2004a. Cited on pages 83 and 105.

J. Wang and B. Bodenheimer. Synthesis and evaluation of linear motion transitions. *ACM Trans. Graph.*, 27(1), 2008. Cited on pages 11 and 38.

J. Wang, A. Hertzmann, and D. M. Blei. Gaussian process dynamical models. In *NIPS*, 2006. Cited on pages 124 and 127.

J. M. Wang, D. J. Fleet, and A. Hertzmann. Multifactor gaussian process models for style-content separation. In *ICML*, 2007. Cited on page 104.

J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *IEEE Trans. Pattern Anal. Mach. Intell.*, 30, 2008a. Cited on page 104.

J. M. Wang, D. J. Fleet, and A. Hertzmann. Gaussian process dynamical models for human motion. *PAMI*, 30(2), 2008b. Cited on pages 122, 124, 126, and 129.

R. Wang, L. Wei, E. Vouga, Q. Huang, D. Ceylan, G. Medioni, and H. Li. Capturing dynamic textured surfaces of moving targets. In *ECCV*, 2016. Cited on page 4.

Z. Wang, A. C. Bovik, H. R. Sheikh, and E. P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 2004b. Cited on pages 89 and 92.

A. Witkin and Z. Popovic. Motion warping. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 105–108, New York, NY, USA, 1995. ACM. ISBN 0-89791-701-4. Cited on page 10.

D. Xu, H. Zhang, Q. Wang, and H. Bao. Poisson shape interpolation. In *Proc. of the 2005 ACM Symposium on Solid and Physical Modeling*, 2005. Cited on pages 23 and 114.

D. Xu, H. Zhang, Q. Wang, and H. Bao. Poisson shape interpolation. *Graphical models*, 68(3), 2006. Cited on page 132.

K. Yamane, Y. Ariki, and J. Hodgins. Animating Non-Humanoid Characters with Human Motion Data. In *SCA*, 2010. Cited on pages 102, 103, and 104.

Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. In *ACM SIGGRAPH 2004 Papers*, 2004a. Cited on pages 14, 28, 105, and 114.

Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. *ACM TOG*, 23 (3), 2004b. Cited on page 131.

L. Zhao and A. Safonova. Achieving good connectivity in motion graphs. In *Proc. of the 2008 ACM/Eurographics Symposium on Computer Animation*, July 2008. Cited on pages 49 and 50.

C. Zheng. One-to-many: example-based mesh animation synthesis. In *The ACM SIGGRAPH / Eurographics Symposium on Computer Animation, SCA '13, Anaheim, CA, USA, July 19-21, 2013*, 2013. Cited on page 12.

K. Zhou, W. Xu, Y. Tong, and M. Desbrun. Deformation transfer to multi-component objects. *Computer Graphics Forum*, 29(2), 2010. Cited on page 104.

C. Zitnick, S. Kang, M. Uyttendaele, S. Winder, and R. Szeliski. High-quality video view interpolation using a layered representation. In *ACM SIGGRAPH*, 2004. Cited on page 77.

S. Zuffi and M. J. Black. The stitched puppet: A graphical model of 3D human shape and pose. In *CVPR*, 2015. Cited on page 104.