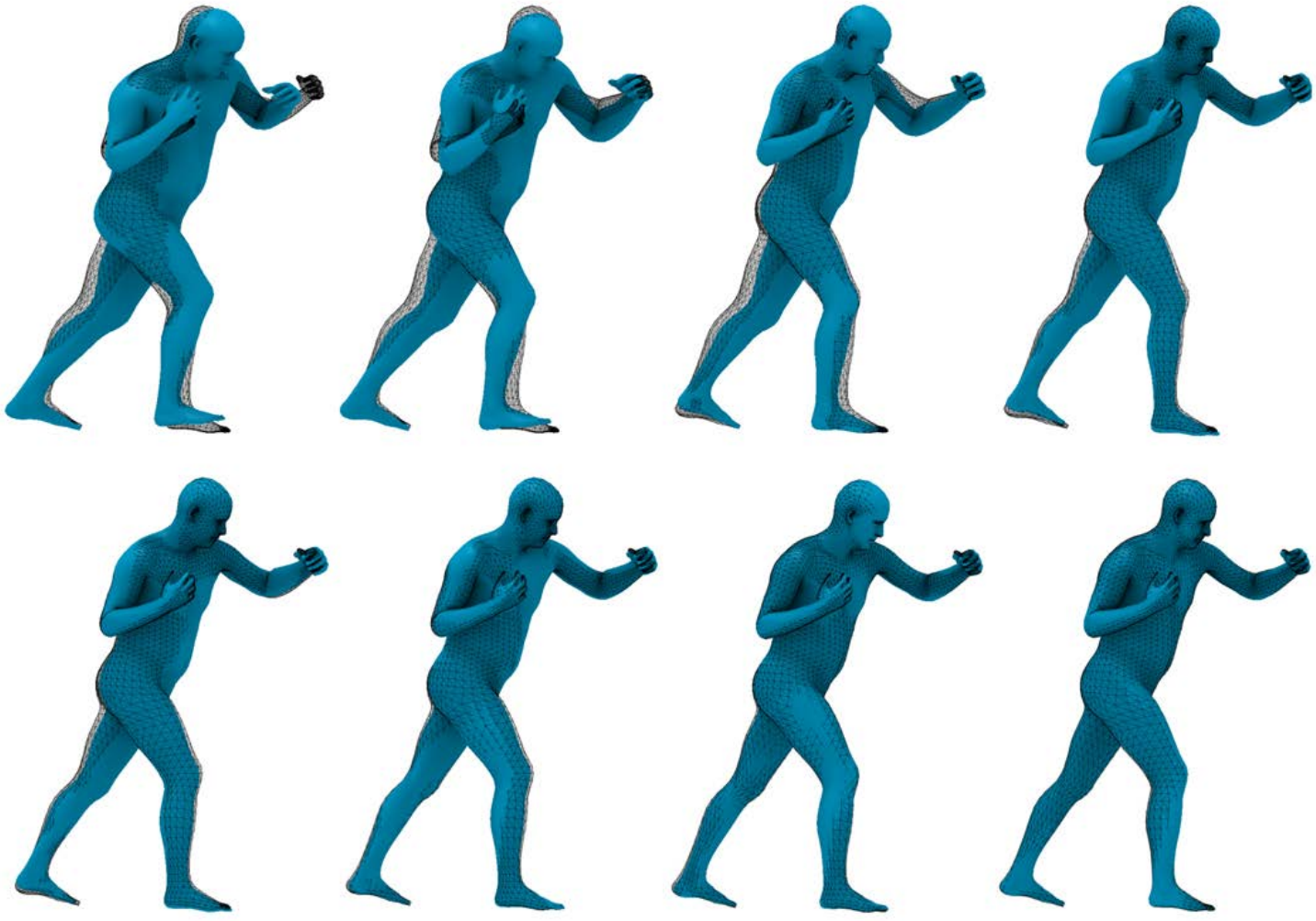
Quaternion PCA and sparse PCA for shape variability

Z.A. van Steijn



TUDelft

Quaternion PCA and sparse PCA for shape variability

by

Z.A. van Steijn

in partial fulfillment of the requirements for the degree of

Master of Science
in Computer Science

at the Delft University of Technology,
to be defended publicly on Monday October 5, 2020 at 11:00 AM.

Student number:	4460758	
Faculty:	EEMCS	
Department:	Computer Science	
Research group:	Computer Graphics and Visualization	
Project duration:	December, 2019 – October, 2020	
Thesis committee:	Dr. K.A. Hildebrandt,	TU Delft, supervisor
	Prof.dr. E. Eisemann,	TU Delft, chair
	Dr. C.C.S. Liem,	TU Delft

TU Delft

Preface

The completion of this thesis marks the end of my 5 year-long journey as a student at Delft University of Technology. A journey that has taught me a lot, both on a personal and professional level, and has left me with countless great memories.

First and foremost, I would like to express my gratitude towards my thesis supervisor Dr. Klaus Hildebrandt, without which this research project would not have been possible. Thank you for the guidance, the many insightful discussions and the invaluable feedback you have provided during the past year. In addition, I would like to thank the other thesis committee members, Prof.dr. Elmar Eisemann and Dr. Cynthia Liem, for taking the time to read and assess my work.

Writing a master's thesis is never an easy task, especially not during the global pandemic of COVID-19. Therefore, I would like to thank my roommates Jur and Thijs for keeping me sane and motivated during these strange times. Furthermore, I would like to thank my friends for providing me with some much-needed distractions this past year and for making my student years in Delft unforgettable.

Special thanks to my parents and my brother for their endless support over the years and, last but not least, Yoran for always putting a smile on my face and believing in me.

Z.A. van Steijn
Delft, September 2020

Nomenclature

$\mathbb{H}$	The skew field of quaternions
$\mathbb{R}$	The real field
$\overline{\mathbf{A}}$	Conjugated matrix
$\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$	The standard quaternion imaginary units
$\mathbf{A}^*$	Conjugate transposed matrix
$\mathbf{A}^T$	Transposed matrix
ADMM	Alternating direction method of multipliers [8]
CPCA	Complex principal component analysis
PCA	Principal component analysis
QPCA	Quaternion principal component analysis
QSPLOCS	Quaternion SPLOCS
QSPLOCS_lap	QSPLOCS with added Laplacian term
QSVD	Quaternion singular value decomposition
SPLOCS	Sparse Localized Deformation Components [39]
SPLOCS_lap	SPLOCS with added Laplacian term
SVD	Singular value decomposition

Contents

1 Introduction

1.1. Motivation

Nowadays, various 3D data acquisition techniques exist that are able to create detailed and high-resolution copies of real-world objects. Modeling and analyzing this digital geometric content is of interest in many different areas, ranging from computer animation to medical imaging. In the former field, methods for deforming 3D shapes and synthesizing new ones from a collection of shapes are studied. Medical imaging, on the other hand, is more concerned with analyzing the shape variations in order to detect abnormalities present in organs or tissues.

One of the problems related to the methods described above is that they often require the solution of a high-dimensional non-linear optimization problem. At the same time, users often expect to get real-time responses. Therefore, approximation algorithms are needed in order to provide a balance between accuracy and speed. For this reason, subspace construction methods have been introduced in the past. The objective of these methods is to construct a lower-dimensional approximation of the problem in an offline phase and to only solve this lower-dimensional system interactively.

Many techniques for constructing subspaces have been introduced, where this work focuses on the well-known *Principal Component Analysis* (PCA) method as well as a sparse PCA variant called *Sparse Localized Deformations Components* (SPLOCS) [39]. The objective of PCA is to decompose the data into components that describe as much variance as possible, while being orthonormal. The sparse PCA technique, on the other hand, tries to find components that are both sparse and localized, without having the orthonormality constraint. By only storing the first few components, a subspace is formed that represents the low-frequency deformations or variations present in the data samples.

When applying PCA or sparse PCA to geometric data, it is common to concatenate the x, y and z coordinates of each sample mesh into a single real-valued vector. As a result, the coordinates of each vertex are no longer treated as a unit, which means that we may lose the potential non-linear spatial relationship that is present among them. Another problem related to these methods is that they are not invariant to rigid motion of the data samples. This means that the quality of the subspace is heavily influenced by the alignment of the shapes. Therefore, the shapes have to be aligned in a preprocessing step, which is often done through rigid registration. This is, however, not always an easy task, especially when the shapes do not have a natural up direction, such as 3D models of an organ. Thus, ideally we would like to find a subspace construction method that is invariant to these rigid motions.

This work investigates whether quaternions can be utilized to solve these challenges. By combining quaternions with PCA or sparse PCA, we wish to construct a subspace that is able to describe a richer space of deformations using fewer components. More specifically, this work will explore the following research directions:

PCA

Can quaternions be used in combination with PCA in order to obtain a lower-dimensional subspace that is invariant to rigid motion of the samples? This research question is motivated by related work on complex PCA, which shows that complex PCA is invariant to rotations in the two-dimensional setting. Therefore, we investigate whether the same holds for quaternion PCA using three-dimensional data.

If quaternion PCA is able to construct an improved subspace, another important research topic is then ***how can the quaternion principal components be computed efficiently?*** A prominent method in this direction for real PCA is the method of snapshots, which learns the subspace from observations. We will explore whether a similar method can be performed using quaternions instead.

Finally, it is also desirable to have a subspace construction method that works on meshes with irregular triangulation. Thus, an additional question we ask is ***can we choose a scalar product such that quaternion PCA becomes robust to mesh irregularities?*** If this is not the case, deformations will mostly be found in areas of the mesh that have many small triangles as opposed to areas with fewer large triangles.

Sparse PCA

Since the sparse PCA technique acts on vertex displacements rather than vertex positions, it does not make sense to investigate rigid motion invariance for this method. However, due to the properties of quaternions, a question we can still ask is ***can quaternions be combined with SPLOCS in order to construct a subspace that is able to describe a richer space of deformations?***

One property of the SPLOCS method is that it uses a spatially varying regularization parameter to create localized components that are centered around certain vertices. These center positions are heavily influenced by the initialization phase, training samples and parameters set by the user. As a result, the obtained components do not always seem natural and can differ a lot based on these parameters. Therefore, an interesting question is ***can we replace the spatially varying regularization strengths in SPLOCS with an additional term based on the Laplacian in order to obtain components that are sparse, localized and smooth?***

And if this is possible, ***can this newly constructed sparse PCA technique also be implemented using quaternions?***

1.2. Contribution

This work combines quaternions with PCA and sparse PCA in order to improve results based on their real counterparts. To the best of our knowledge, this is one of the first researches that uses quaternion PCA in a geometry processing related topic. The main contributions can be summarized as follows:

1. First, we describe how existing methods for computing quaternion PCA (QPCA) can be applied to geometric data in Section 4.2.1. Furthermore, we show how the mass matrix can be incorporated into QPCA in order to obtain a quaternion subspace that is robust to mesh irregularities. This can be found in Section 4.2.3.

2. Next, we present a quaternion method of snapshots in Section 4.2.2, which gives us a much more computationally efficient way of computing the quaternion principal components. Additionally, we describe how the quaternion method of snapshots can be combined with the mass matrix in order to improve the computational efficiency of the mass-weighted QPCA technique.

3. Regarding sparse PCA, we derive a quaternion generalization of SPLOCS, which we call QSPLOCS. Additionally, we derive a quaternion ADMM based scheme for solving the resulting optimization problem. This is shown in Section 5.2.

4. Furthermore, we propose a new sparse PCA method in Section 5.3, inspired by SPLOCS, that introduces an additional term to the objective based on the Laplacian. We call this method

SPLOCS_lap. Additionally, we show how this method can be implemented using quaternions in Section 5.4.

5. After evaluating the performances of SPLOCS and SPLOCS_lap, we introduce a third sparse PCA method that uses both the spatially varying regularization strengths and the Laplacian term. This is discussed in Section 7.4. The method is shown to improve upon the smoothness aspect of SPLOCS and the sparsity and locality aspects of SPLOCS_lap. Additionally, we derive a quaternion version of this method.

6. Finally, we compare the performances of quaternion PCA and the quaternion sparse PCA methods to the performance of their real counterparts on a variety of different datasets in order to determine the accuracy and generality of the methods. These experiments are shown in Chapters 6 and 7. By additionally performing experiments on datasets that are not rigidly aligned, we show that quaternion PCA is more rigid motion invariant than PCA.

1.3. Thesis structure

This master's thesis is structured in the following way. First, Chapter 2 discusses related work and recent developments in the topics of PCA, sparse PCA, complex PCA and quaternion PCA. Next, the background chapter gives a brief introduction on the algebra of quaternions and describes the quaternion SVD method we use in this work. Furthermore, this chapter describes real PCA and complex PCA in more detail as well as the method of snapshots. In Chapter 4, we present the quaternion subspace construction method including the quaternion method of snapshots. Additionally, we show how to make this method robust to mesh irregularities and investigate whether this technique is invariant to rigid motion of the samples. Next, Chapter 5 describes a quaternion subspace construction method based on sparse PCA, which is inspired by the SPLOCS method. Additionally, we introduce a new sparse PCA technique that includes an additional term based on the Laplacian. The performances of the methods are discussed and compared in Chapters 6 and 7, where multiple experiments are performed. Finally, we discuss all findings of this work as well as limitations and future work in Chapter 8.

2

Related work

2.1. PCA and sparse PCA

Principal Component Analysis (PCA) is a well-known technique in the fields of computer graphics and geometry processing for construction a low-dimensional subspace from example shapes, poses or motions. This section briefly discusses the applications of PCA as well as its limitations. Thereafter, we discuss the sparse PCA techniques that have been introduced to solve some of these problems.

One obvious application of PCA is data compression. Alexa and Müller [1] were one of the first to apply PCA to animation sequences for the purpose of geometry compression. By only storing the first few components, they showed that very high compression ratios can be achieved, while accepting a small loss in animation accuracy. This method was subsequently improved by Karni and Gotsman [25], who exploited both the spatial and temporal correlations present in the dataset. Besides capturing the spatial correlations using PCA, the method exploits the temporal coherence present in the data by applying second-order linear prediction coding to the PCA coefficients. In the years following, many more data compression methods have been proposed that in some way make use of PCA.

As a second application, PCA has often been used to represent different facial features or expressions. PCA has proven to be very useful in this area, as facial motions can often be described by a linear path. Blanz and Vetter [6] were one of the first to apply PCA for creating morphable face models. By computing the average face and the principal components of a large number of 3D face scans, the authors constructed a parametric face model that is able to generate almost any face. The model can then be matched to a set of 2D images or 3D scans of faces resulting in a 3D reconstruction of the input face including textures, which is in full correspondence with the morphable face model.

More recently, Weise *et al.* [59] proposed a system for live facial puppetry, that enables real-time facial expression tracking and can transfer expressions to another person's face. This is done by first fitting a generic template to the actor's face. This personalized template is then tracked offline through a set of facial expressions, which is used as input for PCA. The resulting subspace can then be used online for real-time tracking of the facial expressions, which can then be transferred onto an arbitrary target face. Li *et al.* [31] proposed an alternative method for this, that does not require the lengthy offline training session that was needed in [59]. In an offline phase, the neutral face model of the actor is reconstructed, from which a set of initial blendshapes is automatically generated using the deformation transfer method from [52]. Next, the actor's face is tracked online and fitted onto the initial blendshapes, after which it is projected onto the adaptive PCA space. This adaptive PCA space consists of k anchor shapes derived from the initial blend shapes as well as l corrective shapes, which are learned online from new unseen expression samples.

Another area of research that has really benefited from the use of PCA is the area of medical imaging. Statistical shape analysis is a very important topic in this direction, as it can be used to detect abnormalities in a variety of organs or tissues. A PCA subspace can be constructed from either 2D or 3D medical scans in order to represent the shape variability present in the data. Bryan *et al.* [9] used PCA to obtain a statistical shape model of the femur bone. The goal of this study was to generate realistic femurs from this model as solid tetrahedral meshes with associated material properties. This large set of realistic femur models can then be used as input for new finite element studies. Previous

studies in this direction (such as orthopedic implant design studies) often led to dramatic differences in performance between patients due to the lack of available source data.

Grosgeorge *et al.* [17] proposed a method for segmenting the right ventricle in cardiac MRI scans, which is required for the assessment of cardiac function. PCA is used to construct a shape model from a set of right ventricle shapes that have been obtained from manual segmentation by an expert. Segmentation of new data is then performed by incorporating this shape model into the well-known graph cut framework. Promising results are obtained, however the method is not yet able to segment 3D MRI data. Desdouits *et al.* [15] used PCA to characterize the evolution of cavity geometry in dynamic protein systems. They performed a parallel PCA method on protein structures as well as their associated cavities. Using this method, the authors show that there are correlations between the evolutions of the cavities and the structures. Additionally, it can suggest how certain protein conformations can be modified in order to induce a given cavity geometry.

Many more studies exist that use PCA in a computer graphics or geometry processing related topic. For example, Kókai *et al.* [27] introduced a styling framework for creating early conceptual designs of automotive shapes. By training PCA on a set of example car shapes, that are represented using deformation gradients, users are able to pull, drag or edit points in order to obtain new automotive designs. In a complete opposite direction, Heeren *et al.* [19] applied PCA to analyse the shape of plant organs of different cultivars for the purpose of plant breeding. Statistical shape analysis was performed on 140 sugar beet roots of different cultivars in order to obtain a description of the variation in shape. This model is then used to classify unknown tap roots.

One drawback of the components found by PCA is that they are of global support and therefore often lack semantic meaning. This makes the PCA bases less suitable for shape editing and deformation, as the user often requires specific control over small parts of the mesh. For this reason, multiple methods have been introduced for extracting deformation components with local support.

Meyer and Anderson [36] proposed a method to solve this by applying Varimax rotations to the PCA modes. These Varimax rotations are used to alter the basis vectors without changing the spanned subspace. The basis vectors are rotated to maximize the variation of a small set of vertices, while driving the remaining variations towards zero. As a result, the components are much more localized and meaningful. An alternative method was introduced by Tena *et al.* [53], which is a region-based PCA variant. The method uses spectral clustering to divide the mesh into clusters of spatially close vertices. Thereafter, PCA is applied to each cluster individually, resulting in a collection of PCA submodels with shared boundaries. One drawback of this technique is that it is not known whether the clustering is optimal with respect to the deformation model. Furthermore, the model is constrained to the space learned from the training samples, which means that it is not possible to create exaggerated poses.

In 2006, Zou *et al.* [67] introduced standard sparse PCA, which adds a sparsity inducing norm in order to obtain components that only contain a few non-zero values. This method, however, does not take spatial constraints into account, which means that sparsity can occur at any vertex. Thus, even though the components are sparse, they are not necessarily localized around a small section of the mesh. By incorporating these spatial constraints, Neumann *et al.* [39] introduced the SPLOCS method, which is able to extract sparse components that are both localized and meaningful. One disadvantage of this method, however, is that the size and location of the local support region have to be modeled explicitly. Consequently, the resulting components can differ a lot based on these parameters and might not always look natural. A second disadvantage of SPLOCS is that it cannot handle deformations involving large-scale rotations well, as the method uses vertex displacements. In the past years, several approaches have been introduced that address these issues.

Bernard *et al.* [3] tried to tackle the first problem by incorporating regularization terms that simultaneously introduce sparsity and smoothness into the components. As a result, the local support regions can be modeled implicitly, without having to initialize their size and location. However, similar to SPLOCS the method cannot handle articulated motions well.

The limitation of SPLOCS regarding large-scale rotations can be addressed by using more advanced shape representations. Huang *et al.* [21] proposed a method that uses deformation gradients to represent the shape deformations. These deformation gradients are then decomposed to produce components that are both sparse and localized. The rotational components are treated non-linearly, which means that the method is able to handle larger rotations. However, due to the limitations of de-

formation gradients, rotations larger than 180° cannot be dealt with. Furthermore, the method cannot cope with global rotations, since deformation gradients are not an intrinsic geometric quantity.

Wang *et al.* [58] overcome these issues by describing the shapes as edge lengths and dihedral angles instead. Sparse localized components are then extracted using the SPLOCS method [39]. Unfortunately, this method also has some drawbacks. First, the method is insensitive to the scale of the deformations, which makes it sensitive to noise. Furthermore, the method only supports manipulation by modifying the weights of the components, which is not very intuitive. The latter issue was addressed by Liu *et al.* [34] by reformulating the example-based discrete shell deformation. This makes it possible to obtain new poses by simply dragging some of its vertices.

Recently, Sassen *et al.* [48] introduced Sparse Principal Geodesic Analysis (SPGA), which combines the advantages of PGA with those of SPLOCS. The method applies PGA on shape manifolds, which provides a way of obtaining non-linear models of shape variability that are invariant to rigid transformations of the examples poses. By using a ℓ_1 norm as regularizer, the resulting components describe localized deformations, such as the movement of a human shoulder. By defining sparsity as local distortions instead of local displacements, which is used in SPLOCS, the proposed SPGA method is able to handle articulated motions well.

2.1.1. Shape correspondence

Before being able to apply PCA to a dataset, it is important that the vertices of each sample mesh are in correspondence. By this we mean that each vertex of the sample mesh should be paired with the "same" vertex in the other meshes. Note that corresponding vertex pairs might not exist, for instance when the dataset contains a set of partial scans. For this reason, amongst others, shape correspondence is a very difficult problem to solve. The topic of shape correspondence is an own branch of research. A detailed overview of various proposed solutions to shape correspondence can be found in [55]. In this work, we simply assume that all datasets are in correspondence.

2.1.2. Rigid registration

Besides finding point correspondences, we additionally want the samples to be aligned rigidly before applying PCA. This can be achieved by using one of the many Iterative Closest Point (ICP) algorithms that have been introduced over the past three decades. ICP algorithms generally consist of six stages:

1. **Select** source points from one or both meshes.
2. **Match** points to points in other mesh.
3. **Weight** the corresponding pairs.
4. **Reject** outlier point pairs.
5. **Assign error metric** based on the point pairs.
6. **Minimize** error metric.

where the first four stages are related to finding the point correspondences. ICP was first introduced by Besl and McKay [4], who used a point-to-point metric for matching the vertices. This method, however, did not always give satisfactory results, for instance when the samples had a low overlap ratio. In the years following, hundreds of other ICP variants were proposed that affected one of the six stages of the algorithm. Chen and Medioni [10] altered the fifth stage of ICP by using a point-to-plane metric instead, which computes the the distance between the selected point and the tangent plane of the corresponding point. This method is shown to converge at a faster rate using less iterations and is able to get out of a local minimum.

As mentioned in the previous section, finding the corresponding points in stage 2 is a difficult and computationally expensive task. For this reason, Blais *et al.* [5] proposed a faster alternative based on reverse calibration. This method finds corresponding pairs by projecting a point onto the destination mesh using the destination mesh's range camera. Even though this method performs slightly worse in each iteration, it can be up to two orders of magnitude faster than standard closest-point matching. Thus, a trade-off between accuracy and speed has to be made. Another alternative algorithm for the second stage was proposed by Godin *et al.* [16], which restricts the method to only matching points with similar color. This method additionally weights the corresponding pairs based on their color.

A more extensive review of the many existing ICP algorithms can be found in [42, 45].

2.2. Complex PCA

Complex PCA (CPCA) has been used in a number of different applications due to its useful properties, such as its insensitivity to rotation about the origin of the plane. This section briefly discusses a few applications of CPCA in the fields of computer graphics and geometry processing.

Papaioannou *et al.* [40, 41] proposed a method for learning statistical shape priors in the complex domain. By storing the x and y coordinates of the 2D shapes as complex numbers, the CPCA model is shown to be able to include more information. Both horizontal and vertical poses are incorporated in the first complex component, while real PCA needs two components to represent the same deformations. Colosimo and Pacella [11] used CPCA in an industrial context and presented a case study for analyzing the straightness of cylinder axes. By measuring the 3D curve points at equally spaced locations along the z axis, CPCA can be used to model the x and y coordinates of the curve points. Due to the rotation invariance property of CPCA, the authors demonstrate that the registration preprocessing step, which is needed for PCA, can be omitted when using CPCA.

2.3. Quaternion PCA

Quaternion eigenvalue problems have been used in various areas of study, ranging from quantum mechanics [14] to spacecraft attitude dynamics [60]. However, research of these eigenvalue problems in the areas of computer graphics and computer vision seems lacking. Nevertheless, this section briefly discusses the methods that have been used to solve quaternion eigenvalue problems as well as how these methods are applied to various computer vision and graphics tasks.

2.3.1. Methods

Over the past two decades, various methods have been introduced for solving quaternion eigenvalue problems. One of the most used methods in the areas of computer vision and graphics is the method proposed by Le Bihan and Sangwine [29], who introduced a quaternion version of the well-known PCA method. This method, called QPCA, performs eigenvalue decomposition by applying quaternion singular value decomposition (QSVD). The authors obtain this decomposition by rewriting the quaternion matrix as a complex matrix and using classical algorithms for computing complex SVD. The complex representation of a quaternion matrix is called the *complex adjoint* matrix [65].

Since the introduction of the QPCA technique above, multiple other methods for solving quaternion eigenvalue problems have been published. Sangwine and Le Bihan [47] proposed an alternative method for computing the quaternion SVD based on a transformation of the quaternion matrix to bidiagonal form using quaternion Householder transformations. Instead of computing the QSVD indirectly by applying classical algorithms on the complex adjoint matrix, which requires twice as much storage and might lead to a loss in accuracy, this publication introduces a somewhat direct quaternion method of computing the SVD of a quaternion matrix. More details on how this method works are covered in Section 3.4.

A year later, the same authors introduced an implicit Jacobi algorithm for computing the SVD of quaternion matrices directly using quaternion arithmetic [30]. First, the authors proposed an extension of Jacobi rotations to 2×2 quaternionic Hermitian matrices, which can then be used for the diagonalization of quaternion matrices. Based on this, an extension of the implicit Jacobi SVD algorithm for quaternion matrices is introduced, which is more accurate, but slower than the direct QSVD method based on quaternion Householder transformations [47].

In 2013, Jia *et al.* [23] presented a structure-preserving algorithm for solving right eigenvalue problems of Hermitian quaternion matrices. By taking into account the structures of the $4n \times 4n$ real representation, the authors are able to construct a structure-preserving method for the tridiagonalization of this real matrix, which reduces the $4n$-dimensional problem to an n-dimensional problem.

Using the same idea, Li *et al.* [32] introduced a fast structure-preserving algorithm for computing the quaternion SVD. This algorithm evaluates the singular values by applying structure-preserving bidiagonalization of the real representation.

2.3.2. Applications

Quaternion eigenvalue problems have been applied to a variety of computer vision tasks in order to solve problems related to traditional PCA. When dealing with color images, traditional PCA was often applied to each color channel separately after which the results were fused together. However, this

completely ignores the spatial relationship between the different color channels, which could potentially contain valuable information. This issue was addressed by using quaternions instead, as they provide a nice way of describing each pixel of a color image as a pure quaternion:

$$q = R\boldsymbol{i} + G\boldsymbol{j} + B\boldsymbol{k}$$

As a result, color can be processed as a unit, rather than as 3 separate channels. Below we briefly discuss some of the computer vision tasks that make use of quaternion eigenvalue problems.

Shi and Funt [49] proposed an algorithm consisting of three stages for the purpose of texture segmentation of color images. In the first stage features are extracted by applying QPCA to a set of square sub-windows taken from the input image. The training data from the sub-windows is then projected onto the first principal component of the QPCA space. Next, k-means clustering is applied to the feature vectors yielding k centroids that describe the features of the texture clusters. Clusters that are statistically similar are merged in the final stage in order to avoid over-segmentation. Unfortunately, the performance of this approach is very dependent on the parameter selection.

For the purpose of vessel segmentation, Shi *et al.* [50] introduced a method that estimates the curvature in color images. By introducing a quaternion representation of the Hessian matrix, the authors are able to detect lines and edges in color images directly, without having to treat each color channel separately. This Hessian matrix is used as input for QSVD in order to find the principal curvature. This way, it is possible to better separate the vessel structures from the background of the image. This same quaternion Hessian matrix was also used by Mirzaalian *et al.* [37] for detecting malignant melanoma from skin images. Streak structures visible in the dermoscopic images can be enhanced and thereafter classified as regular or irregular depending on their intensity, texture and color distribution.

Another topic of interest is color image denoising. Xu *et al.* [62, 63] proposed a quaternion-based sparse dictionary learning algorithm called K-QSVD. The authors demonstrate that the proposed algorithm can be used for color image reconstruction, denoising and inpainting. For all three use-cases, the K-QSVD method is shown to improve performances based on traditional K-SVD methods. One disadvantage of this method, however, is that the same dictionary is used for all pixels, making the method unsuitable for large and complex images. More recently, Yu *et al.* [64] published a different technique for color image denoising based on quaternion weighted nuclear norm minimization. This method involves applying QSVD to a large set of noisy patch matrices, which improves results both quantitatively and qualitatively. However, this method is less computationally efficient due to the many QSVD computations.

Quaternion eigenvalue problems have also been applied to a number of security related topics, such as identification and image watermarking. Xu and Guo [61] proposed a method for identification using palm-print recognition. Multi-spectral images, obtained by illuminating palm-prints under four different wavelengths, can be used to obtain more information about palm-prints, since different light wavelengths penetrate through different skin layers. These images can then be described using quaternions by storing the pixel values of the four wavelengths in the four parts of the quaternion. Features can then be extracted by applying QPCA on the regions of interest, which can be used for classification. Lang *et al.* [28] published a method for protecting the authenticity, integrity and copyright ownership of digital images. This is done in multiple phases. First, the method selects the four most prominent feature points from each quarter of the image. The pixels inside the quadrangle spanned by the four points are then put into a matrix, which is called the watermarks carrier. QPCA is applied to this matrix and the original dataset is subsequently projected onto this space. Finally, the projected data matrix is combined with a private key and re-projected back to the original space. The authors show that the method is able to withstand a variety of geometrical attacks (translation/rotation/scaling/skewing) as well as some signal processing attacks (salt and pepper noise). However, the method is very computationally expensive.

Quaternion eigenvalue problems have hardly been used in three-dimensional computer graphics applications. To the best of our knowledge, it has only been applied in combination with the quaternion Dirac operator. Crane *et al.* [12] used this operator to compute conformal transformations of 3D triangle meshes. By solving a quaternion eigenvalue problem related to the quaternion Dirac operator, the authors are able to project the result of any arbitrary deformation onto a surface that is approximately conformal, while preserving the textures. In some cases a trade-off between conformality and

approximation quality has to be made, for instance when the deformation contains a large amount of shear.

Liu *et al.* [33] presented a quaternion relative Dirac operator, which is purely extrinsic, and introduced a family of differential operators by interpolating between the relative Dirac operator and the Laplace-Beltrami operator, which is purely intrinsic. The idea behind this is that the user can select operators that are more intrinsic or extrinsic depending on the use-case. QPCA can then be applied for extracting features. The extrinsic relative Dirac operator is shown to be good at distinguishing surfaces with similar base shape but different details. Furthermore, it can distinguish man-made objects from organic shapes with high accuracy. One drawback of this method is that there is no way to select the most appropriate differential operator automatically.

3

Background

Before we can investigate the various questions posed at the start of this work, it is important to first understand the basics of PCA and quaternion PCA. Therefore, this chapter first summarizes the concepts of PCA as well as rigid shape alignment. Thereafter, Section 3.2 briefly discusses complex PCA and its property of rotation invariance. Next, we give an introduction to quaternion numbers and their properties in Section 3.3, which can then be used to explain a method for quaternion SVD in Section 3.4.

3.1. Principal component analysis

PCA is one of the oldest and best known techniques for statistical analysis. It is a popular linear transformation method that transforms data to a new coordinate system, such that the variance is maximized along as few axes as possible, while keeping the axes orthogonal [24]. The first axis, also called the first *principal component*, explains the largest possible variance present in the dataset. Each succeeding component accounts for as much of the remaining variance as possible, while being uncorrelated (i.e. orthogonal) to the previous components.

Since most of the variance is contained in the first few components, PCA can be used to reduce the dimensionality of large datasets. By only keeping the first d components that together explain a large percentage of the variance, the dimensionality of the dataset can be reduced, while preserving as much information as possible.

PCA is often computed using a closely related linear algebra concept called *singular value decomposition* (SVD). SVD is able to factor any real matrix as:

$$\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T$$

where $\mathbf{U}$ and $\mathbf{V}$ are orthogonal matrices containing the eigenvectors (i.e. principal components) of $\mathbf{Y}\mathbf{Y}^T$ and $\mathbf{Y}^T\mathbf{Y}$ respectively and $\mathbf{\Sigma}$ contains the eigenvalues of $\mathbf{Y}$. Thus, the matrix $\mathbf{U}$ contains principal components of the covariance matrix of the dataset, which is exactly what PCA aims to find.

One drawback of PCA is the significant computing resources it requires in terms of processing time and memory usage. Nowadays, the mesh models used are often very detailed and high-dimensional. As a result, PCA would require the solution of a very large eigenvalue problem for the covariance matrix of the data. This could require multiple gigabytes of memory and many hours of computing.

In order to overcome this problem, Sirovich [51] proposed the *method of snapshots*. This method improves the computational complexity in cases where the number of vertices is much larger than the number of samples, which is often the case in geometry processing applications. Instead of solving the eigensystem for the covariance matrix of size $\mathbf{Y}\mathbf{Y}^T \in \mathbb{R}^{3n\times 3n}$, where n is the number of vertices, one only needs to consider the matrix $\mathbf{Y}^T\mathbf{Y} \in \mathbb{R}^{f\times f}$, where f is the number of samples. This smaller matrix is known as the Gram matrix. The principal components of the covariance matrix can then be deducted from the solution of the smaller eigenvalue problem.

The method of snapshots can also be combined with the mass matrix in order to get a mass-weighted PCA [56]. A mass matrix is a diagonal matrix that for each vertex stores a third of the combined

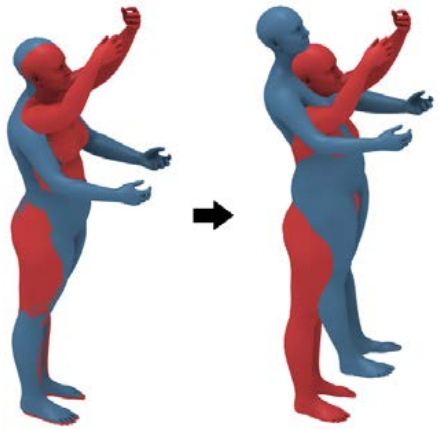


Figure 3.1: Two data samples before (left) and after (right) applying rigid registration. The red mesh is taken as the reference sample.

area of all triangles adjacent to it. Mass-weighted PCA can be used to correct for meshes with irregular triangulation. The resulting principal components will then be robust against remeshing, coarsening or refining of the meshes.

3.1.1. Rigid shape alignment

In the field of geometry processing, subspaces are often constructed to analyze the shape variations and/or scales present in the dataset. A shape can be defined as all the geometrical information that remains when location, scale and rotational effects are filtered out of the object [26]. Thus, when analyzing the data we often want to filter out the rigid motions (i.e. translations and rotations) of the samples.

One property of PCA is that it is not invariant to these rigid motions. Thus, in order to obtain a meaningful subspace using PCA, one has to often apply rigid registration to the samples as a preprocessing step. This method determines the rigid motion for each sample that minimizes the squared distance to a selected reference sample. By applying the optimal rigid motion to each sample, the shapes will be approximately aligned, which can then be used as an input for PCA. Different methods used to achieve this are discussed in Section 2.1.2.

Rigid registration, however, does not always give desirable results, especially on datasets where the samples do not have a natural up direction or when the data samples contain many non-rigid transformations. An example of the latter can be found in Figure 3.1. Since the distance between the vertices in the arms of the two samples is large, applying rigid registration will rotate the blue sample backwards in order to minimize the total squared distance between the points.

3.2. Complex PCA

When working with three-dimensional geometric data, PCA is often applied by concatenating the three components of each data sample. As a result, we may lose the notion that they form a spatial vector. When we are instead working with two-dimensional data, such as a collection of 2D shapes, this problem can be solved by describing each vertex as a complex number instead. Thereafter, complex PCA (CPCA) can be applied to the data, which is computed in a similar way. The only difference between PCA and CPCA is that CPCA employs the conjugate transpose instead and thus uses a Hermitian covariance matrix.

CPCA is shown to lead to improved performance in comparison to real PCA [11, 41]. Using less components the method is able to give a better approximation of the training data samples. This property is nicely visualized in Figure 3.2, taken from [41], which shows the first four components of the PCA and CPCA subspaces. The first component of CPCA (top and bottom image) is able to incorporate both the horizontal and vertical poses of the face. Meanwhile, PCA needs two components to express these same motions.

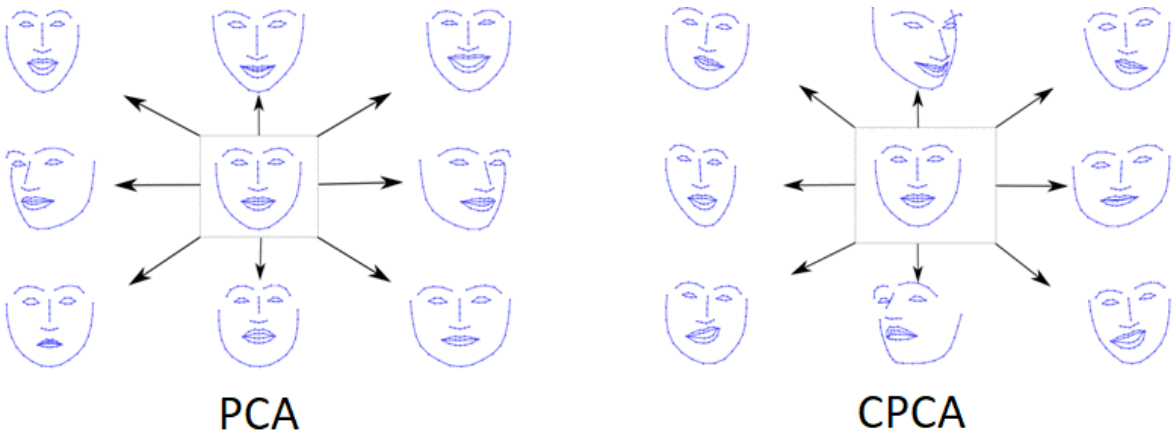


Figure 3.2: First 4 principal components of PCA (left) and CPCA (right). Top and bottom show component 1, left and right show component 2 and the main diagonals show components 3 and 4. Taken from [41].

One important property of CPCA is its insensitivity to variability of the shape with respect to arbitrary rotations around the origin of the complex plane [11]. In other words, this means that CPCA produces the same subspace no matter how the samples are rotated in 2D space. The reason for this is as follows: Let $\mathbf{Y} \in \mathbb{C}^{n \times f}$ be our data matrix, where each sample contains n 2D points. The matrix $\mathbf{Y}_{rot}$ then represents the dataset where each vector is rotated by an arbitrary angle θ_i:

$$\mathbf{Y}_{rot} = \mathbf{YR}, \qquad \mathbf{R} = \text{diag}([\exp(\boldsymbol{i}\theta_1), \cdots, \exp(\boldsymbol{i}\theta_f)])$$

As it turns out, the covariance matrices of these two data matrices are equal:

$$\begin{aligned} \mathbf{Y}_{rot}\mathbf{Y}_{rot}^* &= \mathbf{YRR}^*\mathbf{Y}^* \\ &= \mathbf{YY}^* \end{aligned}$$

Since CPCA is computed from the covariance matrix, this proves that CPCA is indeed invariant to rotations around the origin. As a result, the preliminary step of rigidly aligning each sample, as has to be done for PCA, can be omitted for CPCA.

3.3. The algebra of quaternions

Before investigating how to combine PCA with quaternions, it is important to first understand what quaternion numbers are and what we can do with them. Therefore, this section provides a brief introduction on quaternions and quaternion matrices and discusses how quaternions can be used to describe rotations in 3D space.

3.3.1. Quaternions

Quaternions were first introduced by the mathematician Hamilton in 1843 [18]. They are essentially an extension to complex numbers, where a quaternion $q \in \mathbb{H}$ consists of one real part and three imaginary parts:

$$q = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$$

where $a, b, c, d \in \mathbb{R}$ and $\boldsymbol{i}, \boldsymbol{j}, \boldsymbol{k}$ are three imaginary units satisfying:

$$\begin{aligned} &\boldsymbol{i}^2 = \boldsymbol{j}^2 = \boldsymbol{k}^2 = \boldsymbol{ijk} = -1 \\ &\boldsymbol{ij} = -\boldsymbol{ji} = \boldsymbol{k}, \quad \boldsymbol{jk} = -\boldsymbol{kj} = \boldsymbol{i}, \quad \boldsymbol{ki} = -\boldsymbol{ik} = \boldsymbol{j} \end{aligned}$$

A quaternion with no real part ($a = 0$) is said to be a *pure* quaternion. The conjugate $\overline{q}$ of a quaternion is given by: $\overline{q} = a - b\boldsymbol{i} - c\boldsymbol{j} - d\boldsymbol{k}$ and its norm is: $\|q\| = \sqrt{q\overline{q}} = \sqrt{\overline{q}q} = \sqrt{a^2 + b^2 + c^2 + d^2}$. The inverse of a quaternion is then given by: $q^{-1} = \overline{q}/\|q\|^2$. Thus, a property of unit quaternions is that their inverse is equal to their conjugate.

A characteristic property of quaternions is that they are non-commutative under multiplication. This means that for two quaternions $q_1, q_2 \in \mathbb{H}$:

$$q_1 q_2 \neq q_2 q_1$$

This will be an important property throughout this work. An exception to this rule applies when the imaginary part of either q_1 or q_2 is zero, i.e. when $q_i \in \mathbb{R}$. In this case quaternions are commutative under multiplication:

$$qa = aq \qquad q \in \mathbb{H},\ a \in \mathbb{R}$$

In order to explain how quaternion multiplication works, it is useful to know that quaternions can be represented as an order pair, where r and $\vec{v}$ denote the scalar and vector parts respectively:

$$q = [r,\ \vec{v}] = [a,\ \begin{pmatrix} b \\ c \\ d \end{pmatrix}]$$

The formulas for addition and multiplication are then as follows:

$$\begin{aligned} [r_1,\ \vec{v}_1] + [r_2,\ \vec{v}_2] &= [r_1 + r_2,\ \vec{v}_1 + \vec{v}_2] \\ [r_1,\ \vec{v}_1] \cdot [r_2,\ \vec{v}_2] &= [r_1 r_2 - \langle \vec{v}_1, \vec{v}_2 \rangle,\ r_1 \vec{v}_2 + r_2 \vec{v}_1 + \vec{v}_1 \times \vec{v}_2] \end{aligned} \tag{3.1}$$

where "$\langle \cdot, \cdot \rangle$" denotes the scalar product and "$\times$" denotes the vector product.

3.3.2. Quaternion vectors and matrices

The scalar product of two quaternion vectors $\mathbf{x}, \mathbf{y} \in \mathbb{H}^n$ is defined as:

$$\langle \mathbf{x},\ \mathbf{y} \rangle = \mathbf{x}^* \mathbf{y} = \sum_{i=1}^{n} \overline{x}_i y_i$$

where "$*$" denotes the quaternion conjugate transpose operator, i.e. $\mathbf{x}^* = (\overline{\mathbf{x}})^T = \overline{x^T}$. One of the properties of this scalar product is that $\langle \mathbf{x},\ \mathbf{y} \rangle = \overline{\langle \mathbf{y},\ \mathbf{x} \rangle}$. Similar to real vectors, quaternion vectors are said to be orthogonal if $\langle \mathbf{x},\ \mathbf{y} \rangle = 0$. Using this scalar product, the norm for quaternion vectors is defined as:

$$\|\mathbf{x}\| = \sqrt{\langle \mathbf{x},\ \mathbf{x} \rangle}$$

which can be used to measure the distance between two quaternion vectors $\mathbf{x}$ and $\mathbf{y}$ as $\|\mathbf{x} - \mathbf{y}\|$.

Let $\mathbf{X} \in \mathbb{H}^{n \times n}$ be a square matrix containing quaternion entries. This matrix is said to be Hermitian if $\mathbf{X}^* = \mathbf{X}$, unitary if $\mathbf{X}^* \mathbf{X} = \mathbf{I}$ and invertible if $\mathbf{X}\mathbf{Y} = \mathbf{Y}\mathbf{X} = \mathbf{I}$ for some quaternion matrix $\mathbf{Y} \in \mathbb{H}^{n \times n}$. Quaternion matrices also have some other properties, which will be of importance later on in this work. Let $\mathbf{X} \in \mathbb{H}^{m \times n}$ and $\mathbf{Y} \in \mathbb{H}^{n \times o}$, then [65]:

1. $(\overline{\mathbf{X}})^T = \overline{(\mathbf{X}^T)}$
2. $(\mathbf{X}\mathbf{Y})^* = \mathbf{Y}^* \mathbf{X}^*$
3. $\overline{\mathbf{X}\mathbf{Y}} \neq \overline{\mathbf{X}}\ \overline{\mathbf{Y}}$ (in general)
4. $(\mathbf{X}\mathbf{Y})^T \neq \mathbf{Y}^T \mathbf{X}^T$ (in general)

Since standard packages for numerical linear algebra do not support quaternion matrices, a real matrix $\mathbf{X}' \in \mathbb{R}^{4m \times 4n}$ can be constructed from any quaternion matrix $\mathbf{X} \in \mathbb{H}^{m \times n}$. This is done by replacing each entry $q = a + b\boldsymbol{i} + c\boldsymbol{j} + d\boldsymbol{k}$ by the 4×4 matrix:

$$\begin{pmatrix} a & -b & -c & -d \\ b & a & -d & c \\ c & d & a & -b \\ d & -c & b & a \end{pmatrix}$$

The real transpose of matrix $\mathbf{X}'$ corresponds to the conjugate transpose of the original quaternion matrix $\mathbf{X}$, i.e. $\mathbf{X}'^T = \mathbf{X}^*$. When solving eigenvalue problems with this real representation, each eigenvector of the quaternion matrix is represented by four eigenvectors of the real matrix, which differ only by an arbitrary multiplicative constant $q \in \mathbb{H}$.

3.3.3. Rotations using quaternions

In Section 3.3.1 we explained how a quaternion can be represented as an ordered pair. Using this notation the product of two quaternions could be expressed in a nice and structured way, as shown in Equation (3.1). Now, we will use this to prove that quaternions can be used to rotate points in 3D-space.

Let q_{rot} be a unit-norm quaternion, which will be used to rotate points:

$$q_{rot} = [\cos\theta,\ \sin\theta\vec{v}]$$

Here, θ denotes the angle of the rotation, while the unit vector $\vec{v}$ denotes the rotational axis. Furthermore, let p be the pure quaternion point that we wish to rotate:

$$p = [0,\ \vec{p}]$$

A proper rotated version of p can be obtained by right and left multiplying the point by the unit quaternion q_{rot} and its conjugate respectively:

$$\begin{aligned}
p_{rot} &= q_{rot} \cdot p \cdot \overline{q}_{rot} \\
&= [\cos\theta,\ \sin\theta\vec{v}] \cdot [0,\ \vec{p}] \cdot [\cos\theta,\ -\sin\theta\vec{v}] \\
&= [-\sin\theta\,\langle\vec{v},\vec{p}\rangle,\ \cos\theta\vec{p} + \sin\theta\vec{v}\times\vec{p}] \cdot [\cos\theta,\ -\sin\theta\vec{v}] \\
&= [-\sin\theta\cos\theta\,\langle\vec{v},\vec{p}\rangle + \langle\cos\theta\vec{p} + \sin\theta\vec{v}\times\vec{p},\ \sin\theta\vec{v}\rangle, \\
&\qquad \sin^2\theta\,\langle\vec{v},\vec{p}\rangle\,\vec{v} + \cos\theta(\cos\theta\vec{p} + \sin\theta\vec{v}\times\vec{p}) - (\cos\theta\vec{p} + \sin\theta\vec{v}\times\vec{p})\times(\sin\theta\vec{v})] \\
&= [\sin^2\theta\,\langle\vec{v}\times\vec{p},\ \vec{v}\rangle, \\
&\qquad \sin^2\theta\,\langle\vec{v},\vec{p}\rangle\,\vec{v} + \cos^2\theta\vec{p} + \sin\theta\cos\theta\vec{v}\times\vec{p} - \sin\theta\cos\theta\vec{p}\times\vec{v} - \sin^2\theta(\vec{v}\times\vec{p})\times\vec{v}] \\
&= [\sin^2\theta\,\langle\vec{p},\ \vec{v}\times\vec{v}\rangle, \\
&\qquad \sin^2\theta\,\langle\vec{v},\vec{p}\rangle\,\vec{v} + \cos^2\theta\vec{p} + 2\sin\theta\cos\theta\vec{v}\times\vec{p} + \sin^2\theta\vec{v}\times(\vec{v}\times\vec{p})] \\
&= [0,\ \sin^2\theta\,\langle\vec{v},\vec{p}\rangle\,\vec{v} + \cos^2\theta\vec{p} + 2\sin\theta\cos\theta\vec{v}\times\vec{p} + \sin^2\theta\,\langle\vec{v},\vec{p}\rangle\,\vec{v} - \sin^2\theta\,\langle\vec{v},\vec{v}\rangle\,\vec{p}] \\
&= [0,\ 2\sin^2\theta\,\langle\vec{v},\vec{p}\rangle\,\vec{v} + \cos^2\theta\vec{p} - \sin^2\theta\vec{p} + 2\sin\theta\cos\theta\vec{v}\times\vec{p}] \\
&= [0,\ (1-\cos 2\theta)\,\langle\vec{v},\vec{p}\rangle\,\vec{v} + \cos 2\theta\vec{p} + \sin 2\theta\vec{v}\times\vec{p}]
\end{aligned} \tag{3.2}$$

The result we obtain is exactly the same as Rodrigues' rotation formula around the $\vec{v}$ axis with an angle of 2θ. Thus, p_{rot} gives us a proper rotated version of the point p.

3.4. Quaternion SVD

Similar to real PCA, quaternion PCA can also be computed using singular value decomposition. One straightforward way of computing this quaternion SVD (QSVD) is by rewriting the $n \times m$ quaternion matrix as a $2n \times 2m$ complex-valued matrix [29], also known as the complex adjoint matrix [65]. The quaternion singular values and vectors can then be obtained by applying any classical SVD algorithm for complex matrices. This indirect method of computing QSVD can, however, lead to a loss of accuracy and involves twice as much work as a direct quaternion method. This has to do with the fact that each quaternion is represented by four complex numbers in the complex adjoint matrix, while only two complex numbers are required to describe it.

In 2006, Sangwine and Le Bihan [47] proposed an alternative to the indirect method described above. This method bidiagonalizes any quaternion matrix to a real bidiagonal matrix by applying a simple recursive algorithm that uses quaternion Householder transformations. Classical real PCA can then be applied to obtain the singular values and vectors. Even though this method still does not compute the QSVD directly, it does not rely on the complex adjoint matrix and thus does not require twice the amount of computations. Furthermore, it still makes use of traditional real SVD routines that are well tested and known to be computationally efficient.

The algorithm for the bidiagonalization of a quaternion matrix $\mathbf{Y}$ can be found in Algorithm 1. Here, the result $\mathbf{Z}$ can be an upper or lower bidiagonal real matrix depending on the relative magnitudes of n and m. After having computed this matrix, one can compute the SVD of $\mathbf{Z}$ and use the result to find the singular values and vectors of $\mathbf{Y}$. This is shown in Algorithm 2.

Algorithm 1: Bidiagonalization of quaternion matrix $\mathbf{Y}$.

Data: Data matrix $\mathbf{Y} \in \mathbb{H}^{n \times m}$.
Result: $\mathbf{L} \in \mathbb{H}^{n \times n}$, $\mathbf{Z} \in \mathbb{R}^{n \times m}$, $\mathbf{R}^* \in \mathbb{H}^{m \times m}$

if $m \leq n$ **then**
  `Bidiagonalize`$(\mathbf{Y}, \mathbf{L}, \mathbf{Z}, \mathbf{R})$;
else
  `Bidiagonalize`$(\mathbf{Y}^*, \mathbf{R}, \mathbf{Z}, \mathbf{L})$;
  $\mathbf{Z} \leftarrow \mathbf{Z}^T$;
end

Function `Bidiagonalize`(*Input:* $\mathbf{Y}$, *Output:* $\mathbf{L}$, $\mathbf{Z}$, $\mathbf{R}$)**:**
  $\mathbf{v} \leftarrow \begin{bmatrix} 1 & 0 & \cdots & 0 \end{bmatrix}^T$;
  $\mathbf{L} \leftarrow \mathrm{H}(\mathbf{Y}_{:,1}, \mathbf{v})$; `// Householder transformation.`
  $\mathbf{Z} \leftarrow \mathbf{L}\mathbf{Y}$;
  $\mathbf{R} \leftarrow \mathbb{I}^{m \times m}$;

  if $m > 1$ **then**
    `Bidiagonalize`$(\mathbf{Y}^*_{:,2:m}, \mathbf{R}', \mathbf{Z}', \mathbf{L}')$;
    $\mathbf{L} \leftarrow \mathbf{L}'\mathbf{L}$;
    $\mathbf{Y}_{:,2:m} \leftarrow (\mathbf{Z}')^T$;
    $\mathbf{R}_{2:m,2:m} \leftarrow \mathbf{R}'$;

  $\mathbf{Z} \leftarrow r(\mathbf{Y})$; `// Scalar part of` $\mathbf{Y}$.

Function `H`(*Input:* $\mathbf{y} \in \mathbb{H}^n$, $\mathbf{v} \in \mathbb{R}^n$, *Output:* $\mathbf{H} \in \mathbb{H}^{n \times n}$)**:**
  Find unit quaternion scalar z and quaternion vector $\mathbf{u} \in \mathbb{H}^n$ with $\|\mathbf{u}\| = \sqrt{2}$ such that:
  $z(\mathbb{I} - \mathbf{u}\mathbf{u}^*)\mathbf{y} = \|\mathbf{y}\|\mathbf{v}$;

  $\mathbf{H} \leftarrow z(\mathbb{I} - \mathbf{u}\mathbf{u}^*)$;

Algorithm 2: Quaternion singular value decomposition.

Data: Data matrix $\mathbf{Y} \in \mathbb{H}^{n \times m}$.
Result: $\mathbf{U} \in \mathbb{H}^{n \times n}$, $\mathbf{\Sigma} \in \mathbb{R}^{n \times m}$, $\mathbf{V}^* \in \mathbb{H}^{m \times m}$

Bidiagonalize $\mathbf{Y}$ using Algorithm 1 to decompose $\mathbf{Y}$ into $\mathbf{L}\mathbf{Z}\mathbf{R}^*$;

Compute SVD of $\mathbf{Z}$ to obtain $\mathbf{U}', \mathbf{\Sigma}, \mathbf{V}'$ such that:
$\mathbf{Z} = \mathbf{U}'\mathbf{\Sigma}(\mathbf{V}')^*$;

$\mathbf{U} \leftarrow \mathbf{L}\mathbf{U}'$;
$\mathbf{V}^* \leftarrow (\mathbf{V}')^*\mathbf{R}^*$;

4

Quaternion PCA

This chapter investigates how quaternion PCA (QPCA) can be applied to geometric data in order to describe the shape variability present in a dataset. First, we describe how the sample meshes can be represented using pure quaternions. Using this notation, we then specify how existing methods for quaternion PCA can be applied to this geometric data in Section 4.2. Furthermore, this section includes the derivation of a quaternion method of snapshots as well as a mass-weighted QPCA technique. We believe that this is the first work to introduce these two computationally efficient methods. After discussing the quaternion subspace construction, we explain how samples can be projected onto this subspace in Section 4.3. Finally, Section 4.4 discusses whether the rigid motion invariance property holds for the presented QPCA techniques.

4.1. Samples

Quaternions provide us a nice way to describe 3D meshes. By encoding the x, y and z coordinates of each vertex as a pure quaternion, a mesh of n vertices can be described by a vector of quaternions $\mathbf{y} \in \mathbb{H}^n$. Thus, if we have f poses or snapshots, this can be described by a matrix of quaternions $\mathbf{Y} \in \mathbb{H}^{n \times f}$. In order to obtain meaningful results, it is required that the vertices of each sample are in correspondence. The datasets used in this work already satisfy this requirement. However, methods of obtaining such vertex correspondences are briefly discussed in Section 2.1.1.

4.2. Subspace construction

A mesh containing n vertices has $3n$ degrees of freedom (i.e. the vertex positions). In many cases, a high correlation is present between the vertices of the samples. Therefore, a low-dimensional subspace can be constructed that is optimized for providing a good approximation of the data samples. This subspace is found by constructing a basis matrix $\mathbf{U}$ that minimizes the approximation of the samples:

$$\begin{aligned} \arg\min_{\mathbf{u}_j} \sum_{i=1}^{f} \| \mathbf{y}_i - \sum_{j=1}^{d} \mathbf{u}_j \langle \mathbf{u}_j, \mathbf{y}_i \rangle \|^2 \\ \text{s.t. } \langle \mathbf{u}_a, \mathbf{u}_b \rangle = \delta_{ab} \end{aligned} \tag{4.1}$$

This optimization problem can be solved using a quaternion version of PCA. The remainder of this section is built up as follows. First, we describe a general way of performing quaternion PCA (QPCA) in the section below, which is taken from previous work [29, 47]. Due to the inefficiency of this method in solving the objective above, we contribute by introducing a quaternion method of snapshots in Section 4.2.2. We show that this method significantly improves the computational efficiency of QPCA. Next, we combine this newly introduced quaternion method of snapshots with the mass matrix in order to obtain a mass-weighted QPCA technique in Section 4.2.3. This makes the method more robust to coarsening or refining of the sample meshes.

4.2.1. Quaternion principal component analysis (QPCA)

Given a dataset $\mathbf{Y} \in \mathbb{H}^{n \times f}$, one can build the covariance matrix $\mathbf{\Gamma} \in \mathbb{H}^{n \times n}$ as:

$$\mathbf{\Gamma} = \mathbf{Y}\mathbf{Y}^* \tag{4.2}$$

Performing QPCA now consists of decomposing $\mathbf{\Gamma}$ using quaternion eigenvalue decomposition. Due to the non-commutative property of quaternion multiplication, two kinds of eigenvalue problems exist, namely left- and right eigenvalue problems. These two problems are given by $\mathbf{\Gamma}\mathbf{u}_i = \lambda_i \mathbf{u}_i$ and $\mathbf{\Gamma}\mathbf{u}_i = \mathbf{u}_i \lambda_i$ respectively. In this work, however, there is no difference between these two eigenvalue problems. Since $\mathbf{\Gamma}$ is a quaternionic Hermitian matrix, the eigenvalues λ_i will be real-valued [43]. Left- or right multiplication of a quaternion by a real number yields the same result and therefore these two eigenvalue problems are identical. Rewriting the eigenvalue problem into matrix form gives:

$$\mathbf{\Gamma} = \mathbf{U}\mathbf{\Lambda}\mathbf{U}^* \tag{4.3}$$

where $\mathbf{U} \in \mathbb{H}^{n \times n}$ is a unitary matrix that contains the eigenvectors of $\mathbf{\Gamma}$ and $\mathbf{\Lambda} \in \mathbb{R}^{n \times n}$ is a diagonal matrix containing the eigenvalues. Equation (4.3) can be computed using a quaternion version of singular value decomposition (QSVD). The data matrix $\mathbf{Y}$ admits a QSVD given as:

$$\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{4.4}$$

Here, $\mathbf{U} \in \mathbb{H}^{n \times n}$ and $\mathbf{V} \in \mathbb{H}^{f \times f}$ are unitary matrices containing the eigenvectors of $\mathbf{\Gamma}$ and $\mathbf{Y}^*\mathbf{Y}$ respectively and $\mathbf{\Sigma}$ is a diagonal matrix containing the singular values of $\mathbf{Y}$. Furthermore, $\mathbf{\Sigma}$ is related to $\mathbf{\Lambda}$ in Equation (4.3) by $\mathbf{\Lambda} = \mathbf{\Sigma}^2$. This is easily shown by:

$$\begin{aligned} \mathbf{\Gamma} &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*)(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*)^* = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^*\mathbf{V}\mathbf{\Sigma}^*\mathbf{U}^* = \mathbf{U}\mathbf{\Sigma}^2\mathbf{U}^* \\ \mathbf{Y}^*\mathbf{Y} &= (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*)^*(\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*) = \mathbf{V}\mathbf{\Sigma}^*\mathbf{U}^*\mathbf{U}\mathbf{\Sigma}\mathbf{V}^* = \mathbf{V}\mathbf{\Sigma}^2\mathbf{V}^* \end{aligned} \tag{4.5}$$

By storing only the first d eigenvectors of $\mathbf{U}$ that together explain the largest part of the variance present in the data, we can approximate the samples while strongly reducing the degrees of freedom. The number of elements d can be chosen such that the corresponding basis satisfies a prescribed approximation goal:

$$\frac{\sum_{i=1}^{d} \lambda_i}{\sum_{i=1}^{f} \lambda_i} \tag{4.6}$$

where λ_i are the eigenvalues from $\mathbf{\Lambda}$. Depending on the application, this value should lie close to 1. Note that this only guarantees the approximation error on the set of training samples and not on other unseen test samples.

Sangwine and Le Bihan [47] proposed a method for computing the QSVD based on bidiagonalization to a real matrix using quaternion Householder transformations, which is described in Section 3.4. We decided to use this method as it is shown to be more efficient than other indirect methods of computation. Furthermore, this method is readily available in the MATLAB toolbox `qtfm` [46].

4.2.2. Quaternion method of snapshots

When applying QPCA to the samples, the number of vertices n is often much larger than the number of frames f, especially for high resolution meshes. In that case, the computation of QPCA would require the solution of a very large eigenvalue problem for the dense covariance matrix $\mathbf{\Gamma} \in \mathbb{H}^{n \times n}$. For PCA this problem was solved by applying the method of snapshots, where instead of solving the eigensystem for the covariance matrix, one only needs to consider the Gram matrix [51]. In this section we introduce a quaternion method of snapshots, which is heavily inspired by the real method of snapshots. To the best of our knowledge, we are the first to derive and use this quaternion method.

The quaternion Gram matrix of a data matrix $\mathbf{Y} \in \mathbb{H}^{n \times f}$ is defined as $\mathbf{Y}^*\mathbf{Y} \in \mathbb{H}^{f \times f}$. Equation (4.5) shows that the right eigenvalue problem for this matrix is given by:

$$\mathbf{Y}^*\mathbf{Y}\mathbf{v}_i = \mathbf{v}_i \lambda_i \tag{4.7}$$

This smaller eigenvalue problem can be solved using QPCA, as described in the previous section. By rewriting Equation (4.4), we know that:

$$\mathbf{Y}\mathbf{v}_i = \mathbf{u}_i \sigma_i \tag{4.8}$$

with $\lambda_i = \sigma_i^2$. Therefore, we can obtain the QPCA basis $\mathbf{u}_i$ from the solution of the smaller eigenvalue problem of $\mathbf{Y}^*\mathbf{Y}$ by setting:

$$\begin{aligned} \mathbf{u}_i &= \mathbf{Y}\mathbf{v}_i \sqrt{\lambda_i}^{-1} \\ &= \frac{1}{\sqrt{\lambda_i}} \mathbf{Y}\mathbf{v}_i \end{aligned} \tag{4.9}$$

Note that the eigenvalues λ_i are real-valued. Therefore, it is allowed to move the $\sqrt{\lambda_i}^{-1}$ term forward.

Applying the quaternion method of snapshots can significantly improve the computational complexity of the subspace construction. As an example, suppose that we have $f = 20$ samples, each consisting of $n = 10,000$ vertices. In this case, the normal QPCA method needs to solve a quaternion eigenvalue problem for the large covariance matrix of size $10,000 \times 10,000$. This can take a considerable amount of time and might even fail, depending on the QPCA implementation. On the other hand, the quaternion method of snapshots only requires the solution of a quaternion eigenvalue problem for the Gram matrix of size 20×20. Thereafter, the 20 eigenvectors of the covariance matrix can be obtained by performing 20 quaternion matrix multiplications, as described in Equation (4.9). This comes down to performing nf^2 quaternion multiplications, $nf(f-1)$ quaternion additions and f scalar multiplications.

4.2.3. Mass-weighted quaternion method of snapshots

The subspace found by QPCA might not be satisfactory when the training meshes have irregular triangulation. Imagine a dataset where the left side of each mesh contains many triangles, while the right side only contains a few triangles. In that case, QPCA will prioritize finding basis vectors that accurately represent the left side of the meshes, as the shape variation summed over the triangles is much larger in that area. For real PCA, this problem was solved by computing a mass-orthonormal basis instead, using the mass matrix, that corrects for these mesh irregularities. In this section, we derive a quaternion version of this mass-weighted PCA technique. Furthermore, we show that this mass-orthonormal quaternion basis can also be computed using the more efficient quaternion method of snapshots.

The mass-weighted QPCA subspace can be found by solving an adjusted version of the optimization problem in Equation (4.1):

$$\begin{aligned} \arg\min_{\mathbf{u}_j} \sum_{i=1}^{f} \| \mathbf{y}_i - \sum_{j=1}^{d} \mathbf{u}_j \langle \mathbf{u}_j, \mathbf{y}_i \rangle_M \|_M^2 \\ \text{s.t. } \langle \mathbf{u}_a, \mathbf{u}_b \rangle_M = \delta_{ab} \end{aligned} \tag{4.10}$$

Here, the mass-weighted scalar product and norm are denoted by $\langle \mathbf{x}, \mathbf{y} \rangle_M = \mathbf{x}^*\mathbf{M}\mathbf{y}$ and $\|\mathbf{x}\|_M = \sqrt{\langle \mathbf{x}, \mathbf{x} \rangle_M}$ respectively. Furthermore, $\mathbf{M} \in \mathbb{R}^{n \times n}$ denotes the mass matrix, which is a diagonal matrix that for each vertex stores a third of the combined area of the triangles containing it. The sample on which the mass matrix is computed can be chosen manually. However, it can also be selected automatically, e.g. as the sample that has the median triangle areas. Using the mass matrix allows us to obtain a basis matrix that is robust against remeshing, coarsening or refining of the meshes.

The problem from Equation (4.10) can be solved by applying QPCA on the covariance matrix of the dataset $\mathbf{M}^{1/2}\mathbf{Y} \in \mathbb{H}^{n \times f}$:

$$\begin{aligned} \mathbf{\Gamma}' &= \mathbf{M}^{1/2}\mathbf{Y}(\mathbf{M}^{1/2}\mathbf{Y})^* \\ &= \mathbf{M}^{1/2}\mathbf{Y}\mathbf{Y}^*(\mathbf{M}^{1/2})^* \\ &= \mathbf{M}^{1/2}\mathbf{Y}\mathbf{Y}^*\mathbf{M}^{1/2} \end{aligned}$$

The data matrix $\mathbf{M}^{1/2}\mathbf{Y}$ then admits a QSVD given by:

$$\mathbf{M}^{1/2}\mathbf{Y} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \tag{4.11}$$

Here, $\mathbf{U}$ and $\mathbf{V}$ are unitary matrices storing the eigenvectors of the new covariance matrix $\mathbf{\Gamma}'$ and mass-weighted Gram matrix $\mathbf{Y}^*\mathbf{M}\mathbf{Y}$ respectively. However, it should be noted that we still want to obtain the orthonormal basis of $\mathbf{\Gamma}$, from Equation (4.2), and not of $\mathbf{\Gamma}'$. This basis can easily be found by rewriting Equation (4.11) as:

$$\begin{aligned} \mathbf{M}^{1/2}\mathbf{Y} &= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^* \\ \mathbf{Y} &= (\mathbf{M}^{-1/2}\mathbf{U})\mathbf{\Sigma}\mathbf{V}^* \\ \mathbf{Y} &= \mathbf{U}'\mathbf{\Sigma}\mathbf{V}^* \qquad \text{with } \mathbf{U}' = \mathbf{M}^{-1/2}\mathbf{U} \end{aligned} \tag{4.12}$$

Thus, $\mathbf{U}'$ is the mass-orthonormal basis that solves the mass-weighted optimization problem. It is also possible to adapt the newly introduced quaternion method of snapshots in order to solve this. Similar to Equation (4.7), the eigenvalue problem for the mass-weighted Gram matrix is defined as:

$$\mathbf{Y}^*\mathbf{M}\mathbf{Y}\mathbf{v}_i = \mathbf{v}_i\lambda_i \tag{4.13}$$

Furthermore, the eigenvalue problem related to Equation (4.12) can be written as:

$$\mathbf{Y}\mathbf{v}_i = \mathbf{u}'_i\sigma_i \tag{4.14}$$

Thus, the mass-orthonormal basis $\mathbf{U}'$ can also be computed from the solution of the smaller eigenvalue problem of the mass-weighted Gram matrix by setting:

$$\begin{aligned} \mathbf{u}'_i &= \mathbf{Y}\mathbf{v}_i\sqrt{\lambda_i}^{-1} \\ &= \frac{1}{\sqrt{\lambda_i}}\mathbf{Y}\mathbf{v}_i \end{aligned} \tag{4.15}$$

where again $\lambda_i = \sigma_i^2$.

4.3. Projection onto quaternion subspace

After having constructed the quaternion subspace, we can project any number of samples onto this subspace by finding the weight matrix $\mathbf{W} \in \mathbb{H}^{d\times f}$ that minimizes:

$$\begin{aligned} \arg\min_{w_{ji}} \sum_{i=1}^{f} \|\mathbf{y}_i - \sum_{j=1}^{d} \mathbf{u}_j w_{ji}\|^2 &= \arg\min_{\mathbf{w}_i} \sum_{i=1}^{f} \|\mathbf{y}_i - \mathbf{U}\mathbf{w}_i\|^2 \\ &= \arg\min_{\mathbf{W}} \|\mathbf{Y} - \mathbf{U}\mathbf{W}\|_F^2 \end{aligned} \tag{4.16}$$

where $\|\cdot\|_F^2$ denotes the Frobenius norm, $\mathbf{U} \in \mathbb{H}^{n\times d}$ contains the first d columns of the orthonormal basis and $\mathbf{Y} \in \mathbb{H}^{n\times f}$ contains the f samples that we wish to project onto the QPCA space. Note that these samples do not necessarily have to be the same samples that were used for the construction of the subspace.

Since the basis constructed with QPCA is orthonormal, a closed-form solution exists for finding the desired weights:

$$\mathbf{W} = \mathbf{U}^*\mathbf{Y} \tag{4.17}$$

Thus, a projection of the samples onto the QPCA subspace is found by:

$$\begin{aligned} \mathbf{Y}_{proj} &= \mathbf{U}\mathbf{W} \\ &= \mathbf{U}\mathbf{U}^*\mathbf{Y} \end{aligned} \tag{4.18}$$

If $\mathbf{U}$ would instead be a mass-orthonormal basis, as described in Section 4.2.3, the optimal weights are:

$$\mathbf{W} = \mathbf{U}^*\mathbf{M}\mathbf{Y} \tag{4.19}$$

In that case, the projection of the samples onto the subspace can be found by:

$$\begin{aligned} \mathbf{Y}_{proj} &= \mathbf{U}\mathbf{W} \\ &= \mathbf{U}\mathbf{U}^*\mathbf{M}\mathbf{Y} \end{aligned} \tag{4.20}$$

Since the number of components d is typically smaller than the number of vertices n, the projected data matrix $\mathbf{Y}_{proj}$ will not only contain pure quaternions. In other words, besides the x, y and z components of the vertices that are stored in the imaginary parts of the quaternions, the quaternions will also have an additional non-zero real part. Since we are not able to visualize these real parts, as this would require a four-dimensional space for viewing, we simply set these real parts to zero. Furthermore, setting these real parts to zero will also minimize the optimization problem shown in Equation (4.16), since the data matrix $\mathbf{Y}$ only contains pure quaternions.

4.4. Rigid motion invariance

As mentioned in Section 3.1.1, one of the problems of PCA is that it is not invariant to rigid motions of the data samples. This means that the quality of the subspace found by PCA heavily depends on the alignment of the data samples. Therefore, rigid registration is often applied beforehand in order to approximately align the samples.

This section investigates whether the rigid registration preprocessing step can be omitted when QPCA is used instead. In other words, this section tries to answer the following question: Is the subspace found by QPCA invariant to rigid motion of the input samples? Here, a rigid motion is defined as a translation, a rotation or a combination of both. Note that we are mostly interested in the rotational aspect, which we discuss in Section 4.4.2.

4.4.1. Translation invariance

Similar to the PCA approach, translation invariance in QPCA can be achieved by simply centering each sample around zero. This is done by subtracting the centroid of the shape from the shape itself:

$$\mathbf{y}_i' = \mathbf{y}_i - y_{mean,i} \quad \text{for } i = 1, \dots, f \tag{4.21}$$

The centroid of a shape $\mathbf{y}_i$ can easily be computed using quaternions by:

$$y_{mean,i} = \frac{1}{n} \sum_{j=1}^{n} y_{ji} \tag{4.22}$$

When using mass-weighted QPCA, one can also define the centroid of the shape as its center of mass:

$$y_{mean,i} = \frac{1}{\|\mathbf{1}\|_M^2} \sum_{j=1}^{n} m_{jj} \cdot y_{ji} \tag{4.23}$$

Here, $\mathbf{M}$ denotes the mass-matrix and $\|\cdot\|_M$ is the mass-weighted scalar norm.

4.4.2. Rotation invariance

Ideally, we would like the constructed subspace to be independent to rotations of the data samples. In other words, we would like to obtain the same subspace whether we compute it on f sample frames $\mathbf{y}_i$ or compute it on f rotated sample frames $q_i \mathbf{y}_i \overline{q}_i$ for some unit quaternions q_i. Unfortunately, we could not fully achieve this. In order to be able to represent all possible rotations of a sample, a combination of both left- and right multiplication with a unit quaternion is needed, as seen in Section 3.3.3. For this reason, it is not possible to create a subspace that is rotation invariant. This has to do with the fact that the original samples and rotated samples do not give the same covariance matrix:

$$\begin{aligned}\mathbf{Y}\mathbf{Y}^* &= \sum_i^f \mathbf{y}_i\mathbf{y}_i^* \\ \mathbf{Y}_{rot}\mathbf{Y}_{rot}^* &= \sum_i^f (q_i\mathbf{y}_i\overline{q}_i)(q_i\mathbf{y}_i\overline{q}_i)^* \\ &= \sum_i^f q_i\mathbf{y}_i\overline{q}_i q_i\mathbf{y}_i^*\overline{q}_i \\ &= \sum_i^f q_i\mathbf{y}_i\mathbf{y}_i^*\overline{q}_i\end{aligned} \tag{4.24}$$

Since the quaternion subspace is constructed from the covariance matrix, we know that $\mathbf{Y}$ and $\mathbf{Y}_{rot}$ will not generate the same subspace. However, frames that have been multiplied with a unit quaternion q_i only from the right, i.e. $\mathbf{y}_i q_i$, do give us the same subspace as the subspace constructed from $\mathbf{y}_i$. In this case, the covariance matrix is the same for both datasets:

$$\begin{aligned}\mathbf{Y}_{right}\mathbf{Y}_{right}^* &= \sum_i^f (\mathbf{y}_i q_i)(\mathbf{y}_i q_i)^* \\ &= \sum_i^f \mathbf{y}_i q_i\overline{q}_i\mathbf{y}_i^* \\ &= \sum_i^f \mathbf{y}_i\mathbf{y}_i^*\end{aligned} \tag{4.25}$$

Thus, any pair of covariance matrices will be equal if the frames are right multiplied by arbitrary unit quaternions. This means that the quaternion subspace is invariant to multiplication with a unit quaternion from the right. In order to explain which transformations can be represented in this subspace, we have to take another look at quaternion multiplication. Multiplying a point p with a unit quaternion q_{rot} from the right gives:

$$\begin{aligned}p_{right} &= p \cdot \overline{q}_{rot} \\ &= [0,\ \vec{p}] \cdot [\cos\theta,\ -\sin\theta\vec{v}] \\ &= [\sin\theta\,\langle\vec{p},\vec{v}\rangle,\ \cos\theta\vec{p} - \sin\theta\vec{p}\times\vec{v}] \\ &= [\sin\theta\,\langle\vec{v},\vec{p}\rangle,\ \cos\theta\vec{p} + \sin\theta\vec{v}\times\vec{p}]\end{aligned} \tag{4.26}$$

In cases where $\vec{v}$ is perpendicular to $\vec{p}$, i.e. when $\vec{v} \perp \vec{p}$, the real term in Equation (4.26) disappears and point p_{right} already describes a proper rotated version of p. However, when the two vectors are not perpendicular, a proper rotated version can only be obtained by additionally left-multiplying the result by q_{rot}. In Section 3.3.3 we showed that this results in:

$$\begin{aligned}p_{rot} &= q_{rot} \cdot p \cdot \overline{q}_{rot} \\ &= [0,\ (1-\cos 2\theta)\,\langle\vec{v},\vec{p}\rangle\,\vec{v} + \cos 2\theta\vec{p} + \sin 2\theta\vec{v}\times\vec{p}]\end{aligned} \tag{3.2}$$

If we ignore the real parts, we can see that Equation (4.26) and (3.2) differ only by the term $(1 - \cos 2\theta)\,\langle\vec{v},\vec{p}\rangle\,\vec{v}$. This term actually projects $\vec{p}$ onto the axis of rotation $\vec{v}$, as can be seen in Figure 4.1, and scales it by $1 - \cos 2\theta$. Thus, when only right-multiplication by q_{rot} is applied to each point of a mesh, the resulting mesh will be a "correctly" rotated mesh apart from the scaling of the points along the axis of rotation $\vec{v}$. An example of this can be found in Figure 4.2, where the mesh seems to be perfectly rotated when looking straight through the axis of rotation. However, when viewed from another angle, the incorrect scaling along the axis of rotation becomes visible.

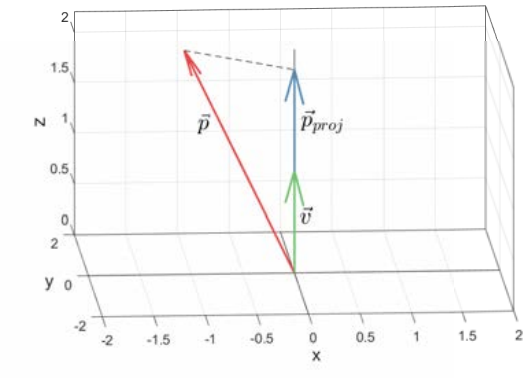


Figure 4.1: Visualization of the term $\vec{p}_{proj} = \langle \vec{v}, \vec{p} \rangle \vec{v}$, where $\vec{p}$ contains the vertex coordinates of point p and $\vec{v}$ denotes the axis of rotation.

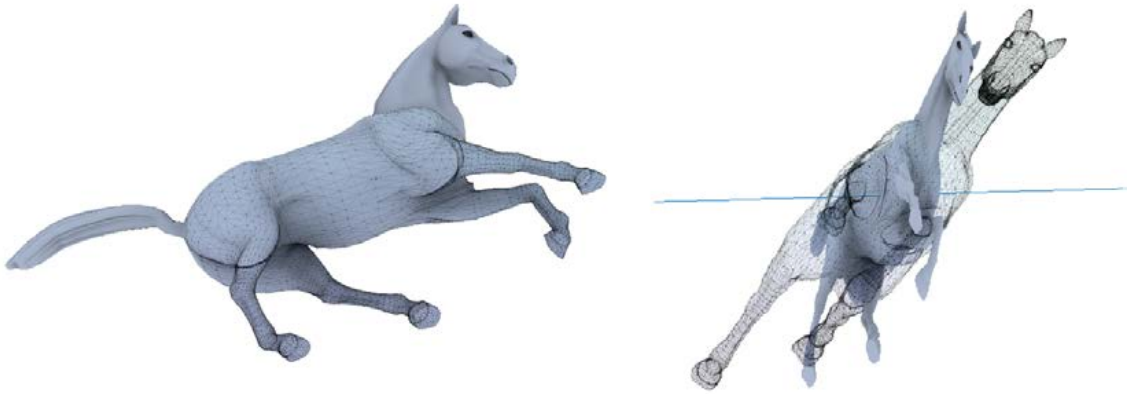


Figure 4.2: Right-multiplied sample $\mathbf{y}_i q_i$ (gray) versus proper rotated sample $q_i \mathbf{y}_i \overline{q}_i$ (transparent) viewed through the axis of rotation (left) and from another angle (right).

It should be noted that the optimal unit quaternion to right-multiply each point with in order to best approximate the rotated sample is not necessarily the same as the unit quaternion that was used to obtain the rotated sample. Instead, the optimal unit quaternion is given by:

$$\begin{aligned} q_{opt} &= \frac{\mathbf{y}_i^* \mathbf{y}_{rot,i}}{\|\mathbf{y}_i^* \mathbf{y}_{rot,i}\|} \\ &= \frac{\mathbf{y}_i^* q_i \mathbf{y}_i \overline{q}_i}{\|\mathbf{y}_i^* q_i \mathbf{y}_i \overline{q}_i\|} \\ &= \frac{\mathbf{y}_i^* q_i \mathbf{y}_i \overline{q}_i}{\|\mathbf{y}_i^* \mathbf{y}_i\|} \end{aligned} \tag{4.27}$$

This optimal quaternion is visualized in Figure 4.3, where q_{opt} is used to obtain the right-multiplied sample instead of the q_i from Figure 4.2.

In conclusion, we can state that the subspace found by QPCA is not rigid motion invariant. It is, however, invariant to right-multiplication with a unit quaternion. Therefore, both the original sample as well as any right-multiplied sample can be represented in the same subspace. In practice, this means that we are able to describe any rotation up to an incorrect scaling along the axis of rotation, as shown in Figure 4.2. By selecting a different unit quaternion, and thus a different axis of rotation, we can approximate the rotated sample even better, as shown in Figure 4.3. Thus, even though the QPCA technique will not give a subspace that is fully rotational invariant, and thus rigid motion invariant, it will most definitely be able to filter out the rotational aspect of the samples better than PCA is able to.

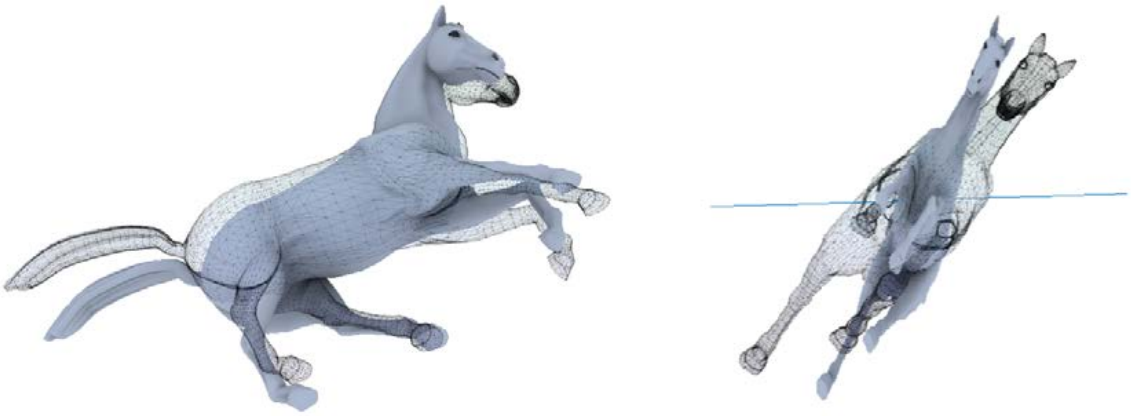


Figure 4.3: Right-multiplied sample $\mathbf{y}_i q_{opt}$ (gray) versus proper rotated sample $q_i \mathbf{y}_i \overline{q}_i$ (transparent) viewed through the axis of rotation (left) and from another angle (right). The right-multiplied sample uses the optimal unit quaternion from Equation (4.27).

5

Quaternion sparse PCA

One disadvantage of the PCA and QPCA methods described in Chapter 4 is that the components found by these methods are of global support. This means that each component deforms every vertex of the mesh. As a result, only the first few components can typically be interpreted. In some settings it would be preferable to have sparse and localized components, where each component only impacts a small part of the mesh. This way, the components are much more interpretable and each vertex of the mesh is deformed by a small set of components only. Another added benefit of this sparsity is that less storage space is required to store the components. This chapter describes ways of obtaining sparse and localized components using both real numbers and quaternions, inspired by a sparse PCA technique called the Sparse Localized Deformation Components (SPLOCS) method [39].

The chapter is structured as follows. First, we briefly review the original SPLOCS method in Section 5.1. Thereafter, we introduce a quaternion SPLOCS method, which takes advantage of the correlation between the x, y and z coordinates of the vertices. This is derived in Section 5.2. Next, we introduce a new sparse PCA method in Section 5.3, which adds an additional Laplacian term to the optimization problem of SPLOCS. The quaternion version of this new sparse PCA method is then described in Section 5.4. Details on how the samples are projected onto the sparse subspaces are given in Section 5.5. Finally, Section 5.6 discusses the tunable parameters that are present in the sparse PCA methods.

5.1. Sparse localized deformation components (SPLOCS)

Before going into how the SPLOCS method can be implemented using quaternions, we first give a brief summary of the original SPLOCS method. For a more detailed description, we refer to [39].

Similar to the PCA and QPCA methods, the SPLOCS method aims to construct a low-dimensional subspace that is optimized for providing a good approximation of the sample frames. However, instead of imposing an orthogonality constraint on the components, SPLOCS introduces a sparsity inducing term as regularizer on the components. Thus, SPLOCS aims to minimize the following optimization problem:

$$\arg\min_{\mathbf{U},\mathbf{W}} \|\mathbf{Y} - \mathbf{U}\mathbf{W}\|_F^2 + \Omega(\mathbf{U}) \qquad \text{s.t. } \mathcal{V}(\mathbf{W}) \tag{5.1}$$

where $\mathbf{U} \in \mathbb{R}^{3n \times d}$ contains the d components, $\mathbf{W} \in \mathbb{R}^{d \times f}$ contains their corresponding weights, $\Omega(\mathbf{U})$ is the regularizer and the constraint $\mathcal{V}$ makes sure that the weights of each component do not grow too large:

$$\max(|\mathbf{w}_i|) = 1 \qquad \text{for } i = 1, \ldots, d \tag{5.2}$$

It is important to note that the dataset $\mathbf{Y}$ does not contain the vertex positions of each frame, as was the case in PCA and QPCA. Instead, the method expresses the vertex coordinates as the residual displacements to a "rest shape" $\bar{\mathbf{y}}$ of the mesh, which could for instance be the first sample or the average over all samples. Thus, the dataset is defined as:

$$\mathbf{Y} \leftarrow \mathbf{Y} - \bar{\mathbf{y}} \tag{5.3}$$

where the rest shape is subtracted from each column of $\mathbf{Y}$. Since the dataset is defined in terms of residual displacements, the component matrix $\mathbf{U}$ will also be defined in such a way. Therefore, the

sparsity inducing term $\Omega(\mathbf{U})$ can be described as:

$$\Omega(\mathbf{U}) = \sum_{i=1}^{n}\sum_{j=1}^{d} \Lambda_{ij}\|\mathbf{u}_{ij}\|_2 \tag{5.4}$$

where $\mathbf{u}_{ij}$ is a triplet corresponding to the x, y and z displacements of vertex i in component j and Λ_{ij} is the regularization strength. By using a ℓ_1/ℓ_2 norm in the sparsity inducing term, which is a form of group sparsity, we make sure that the x, y and z dimensions of each vertex vanish simultaneously.

If the regularization strength would be constant for all vertices and components, the resulting components would be sparse, but not necessarily localized. The sparsity inducing term does not specify which vertex displacements should vanish. For this reason, SPLOCS employs a spatially varying regularization parameter based on the distances to the center of each component. The method stipulates that each component should be centered around the vertex showing the largest displacement within that component. Thereafter, the method computes the geodesic distances of each vertex in the rest shape to this center vertex, using the heat method [13], and linearly maps and clips these values to the range $[0, 1]$:

$$\Lambda_{ij} = \lambda \cdot \begin{cases} 0 & \text{if } d_{ij} < d_{\min} \\ 1 & \text{if } d_{ij} > d_{\max} \\ \frac{d_{ij}-d_{\min}}{d_{\max}-d_{\min}} & \text{otherwise} \end{cases} \tag{5.5}$$

Using this definition, vertices that lie close to the center of a component will have a low regularization strength, while vertices that lie further away will have a larger value for Λ_{ij}. As a result, the sparsity inducing term $\Omega(\mathbf{U})$ will only apply to vertices that lie far away and thus only these displacements will vanish.

The optimization problem shown in Equation (5.1) is solved using a refinement method that iteratively fixes either $\mathbf{W}$ or $\mathbf{U}$ and solves for the other. Optimizing Equation (5.1) with respect to the weights $\mathbf{W}$ is a constrained linear least squares problem, while optimization of the components $\mathbf{U}$ is tackled using the Alternating Direction Method of Multipliers (ADMM) [8].

5.2. Quaternion SPLOCS (QSPLOCS)

The SPLOCS method needs some adjustments in order for it to be implemented using quaternions. In this section we introduce a quaternion version of this method, which we call QSPLOCS, and take a closer look at its implementation details. Similar to the SPLOCS method, QSPLOCS aims to minimize the following optimization problem:

$$\arg\min_{\mathbf{U},\mathbf{W}} \|\mathbf{Y} - \mathbf{U}\mathbf{W}\|_F^2 + \Theta(\mathbf{U}) \qquad \text{s.t. } \mathcal{V}(\mathbf{W}) \tag{5.6}$$

where $\mathbf{Y} \in \mathbb{H}^{n\times f}$, $\mathbf{U} \in \mathbb{H}^{n\times d}$ and $\mathbf{W} \in \mathbb{H}^{d\times f}$ are quaternion matrices describing the dataset, components and their weights respectively, $\Theta(\mathbf{U})$ is the regularizer and the constraints $\mathcal{V}$ are defined in the same way as in the real case from Equation (5.2). As before, the vertex coordinates are expressed as residual displacements to a rest shape $\bar{\mathbf{y}}$. The sparsity inducing term of SPLOCS, however, which uses the ℓ_1/ℓ_2 group norm, can be simplified when using quaternions. Since the x, y and z displacements of each vertex are already grouped together in a quaternion, applying a simple ℓ_1 norm on the quaternion components will have the same effect as applying the ℓ_1/ℓ_2 norm in the real case. Thus, the regularizer of QSPLOCS can be written as:

$$\Theta(\mathbf{U}) = \sum_{i=1}^{n}\sum_{j=1}^{d} \Lambda_{ij}|u_{ij}| \tag{5.7}$$

Here, the regularization strengths Λ_{ij} are defined in the same way as in Equation (5.5). The optimization problem of Equation (5.6) is again solved using an iterative refinement method and ADMM for the optimization of the components. Since we are dealing with quaternions and a ℓ_1 norm, there are some differences in the implementation of ADMM for QSPLOCS. Details on how this is computed are discussed in Section 5.2.2. The following section first discusses the initialization phase of QSPLOCS.

Algorithm 3: Initialization phase of QSPLOCS.

Data: Data matrix $\mathbf{Y}$, regularization strength matrix $\mathbf{\Lambda}$ and regularization parameter λ.
Result: Components $\mathbf{U}$ and their corresponding weights $\mathbf{W}$.

$\mathbf{R} = \mathbf{Y}$;
for $i = 1, \dots, d$ **do**
  $j \leftarrow$ index of vertex showing largest displacement over all samples in $\mathbf{R}$;
  $\mathbf{W}_{i,:} \leftarrow \mathcal{V}(\mathbf{R}_{j,:})$;
  $\mathbf{U}_{:,i} \leftarrow (\mathbf{R}\mathbf{W}_{i,:}^{*} \odot (1 - \mathbf{\Lambda}_{:,i}/\lambda)) \,/\, \|\mathbf{W}_{i,:}\|^2$;
  $\mathbf{R} \leftarrow \mathbf{R} - \mathbf{U}_{:,i}\mathbf{W}_{i,:}$;
end

5.2.1. Initialization phase

We initialize the components and weights of QSPLOCS in a similar way as the real SPLOCS method does [39]. The components are initialized in an iterative way, where each component is centered around the vertex that shows the largest displacement over all samples in the residual. This is shown in Algorithm 3. Note that $\odot$ stands for the element-wise product operator.

5.2.2. Optimization phase

After the components have been initialized, the next step is to optimize the components and their corresponding weights. Due to the non-convexity of Equation (5.6) and the spatially varying regularization parameters Λ_{ij}, the problem cannot easily be optimized directly. Therefore, the problem is optimized using an iterative refinement method that alternates between optimizing for $\mathbf{W}$ and $\mathbf{U}$.

The optimization of the weights given fixed $\mathbf{U}$ is a linear least squares problem and is solved, similarly to real SPLOCS [39], using a simple block-coordinate descent algorithm. The optimization of $\mathbf{U}$ given fixed $\mathbf{W}$ is tackled using convex optimization. SPLOCS showed quick convergence when using ADMM to solve this, which is a first order optimization method. For this reason, QSPLOCS is also optimized using ADMM, so that we are able to compare both methods later on.

Before being able to solve Equation (5.6) with respect to $\mathbf{U}$ using ADMM, we first need to modify this optimization problem by introducing a dual variable $\mathbf{D} \in \mathbb{H}^{n \times d}$. The objective can then be rewritten as:

$$\arg\min_{\mathbf{U},\mathbf{D}} \|\mathbf{Y} - \mathbf{U}\mathbf{W}\|_F^2 + \Theta(\mathbf{D}) \qquad \text{s.t. } \mathbf{U} - \mathbf{D} = \mathbf{0} \tag{5.8}$$

Algorithm 4 shows the three steps of ADMM that are iterated in order to optimize the objective above. The first step, optimizing for $\mathbf{U}$, is a linear least squares problem. The derivative of these terms using quaternion algebra are:

$$\frac{\partial}{\partial \mathbf{U}} \left(\|\mathbf{Y} - \mathbf{U}\mathbf{W}\|_F^2 + \frac{\rho}{2}\|\mathbf{U} - \mathbf{D} + \mathbf{V}\|_F^2 \right) = 2 \cdot \overline{\mathbf{U}\mathbf{W}\mathbf{W}^*} - 2 \cdot \overline{\mathbf{Y}\mathbf{W}^*} + \rho(\overline{\mathbf{U}} - \overline{\mathbf{D}} + \overline{\mathbf{V}}) \tag{5.9}$$

Setting the derivative to zero, we obtain the optimal $\mathbf{U}$:

$$\begin{aligned} 2 \cdot \overline{\mathbf{U}\mathbf{W}\mathbf{W}^*} - 2 \cdot \overline{\mathbf{Y}\mathbf{W}^*} + \rho(\overline{\mathbf{U}} - \overline{\mathbf{D}} + \overline{\mathbf{V}}) &= 0 \\ 2 \cdot \mathbf{U}\mathbf{W}\mathbf{W}^* + \rho\mathbf{U} &= 2 \cdot \mathbf{Y}\mathbf{W}^* + \rho(\mathbf{D} - \mathbf{V}) \\ \mathbf{U} &= (2 \cdot \mathbf{Y}\mathbf{W}^* + \rho(\mathbf{D} - \mathbf{V}))\,(2 \cdot \mathbf{W}\mathbf{W}^* + \rho\mathbb{I})^{-1} \end{aligned} \tag{5.10}$$

Since the $(2 \cdot \mathbf{W}\mathbf{W}^* + \rho\mathbb{I})$ term does not depend on $\mathbf{U}$ and thus does not change during ADMM, we are able to factorize this term using Cholesky decomposition. We compute this using a structure-preserving method for quaternion Cholesky decomposition [57].

The second step, optimizing for the dual variable $\mathbf{D}$, corresponds to the proximal operator of the ℓ_1 norm Θ. A closed-form solution of this can be found in [8]:

$$\begin{aligned} d_{ij} \leftarrow \text{prox}_{\Theta}(u_{ij} + v_{ij}) &= \left(1 - \frac{\Lambda_{ij}}{\rho|u_{ij} + v_{ij}|}\right)_+ (u_{ij} + v_{ij}) \\ &= \text{sign}(u_{ij} + v_{ij}) \left(|u_{ij} + v_{ij}| - \frac{\Lambda_{ij}}{\rho}\right)_+ \end{aligned} \tag{5.11}$$

Algorithm 4: Optimization of $\mathbf{U}$ using ADMM for QSPLOCS.

Data: Data matrix $\mathbf{Y}$, weight matrix $\mathbf{W}$, penalty parameter ρ and number of ADMM iterations k.
Result: Optimized components $\mathbf{U}$.

$\mathbf{V} \leftarrow \mathbf{0}$;
$\mathbf{D} \leftarrow \mathbf{U}$;
for $i = 1, \dots, k$ **do**
  $\mathbf{U} \leftarrow \arg\min_{\mathbf{U}} \|\mathbf{Y} - \mathbf{U}\mathbf{W}\|_F^2 + \frac{\rho}{2}\|\mathbf{U} - \mathbf{D} + \mathbf{V}\|_F^2$;
  $\mathbf{D} \leftarrow \arg\min_{\mathbf{D}} \Theta(\mathbf{D}) + \frac{\rho}{2}\|\mathbf{U} - \mathbf{D} + \mathbf{V}\|_F^2$;
  $\mathbf{V} \leftarrow \mathbf{V} + \mathbf{U} - \mathbf{D}$;
end

where $(\cdot)_+ = \max(\cdot, 0)$. This optimization of the dual variable $\mathbf{D}$ differs from its optimization in SPLOCS, since SPLOCS uses the proximal operator of the ℓ_1/ℓ_2 group norm.

5.3. SPLOCS using Laplacian term (SPLOCS_lap)

Even though the SPLOCS and QSPLOCS methods are able to extract components that are both sparse and localized, the resulting components found by these methods are heavily influenced by the initialization phase and the varying regularization strengths Λ_{ij}. This has a few consequences. First, the vertices around which the components are centered in the initialization phase will change depending on the sample set used for training and the values for d_{min} and d_{max}. Additionally, due to the way $\Omega(\mathbf{U})$ and $\Theta(\mathbf{U})$ are defined and the use of ADMM as solver, these center locations do not move around a lot during the optimization phase. Finally, the size of the components is heavily influenced by the d_{min} and d_{max} values set by the user. These parameters often need to be adjusted manually to get good results on different datasets.

Ideally, we would like to find a sparse PCA method that gives components that are naturally localized, around parts of the mesh that make sense, without having to define these spatially varying regularization strengths Λ_{ij}. This section discusses whether this can be achieved by adding an additional regularization term to the optimization problem based on the Laplacian. This Laplacian term compares the displacement value of each vertex to that of its neighbors and penalizes whenever this difference is large. The idea is that this will, in combination with the ℓ_1/ℓ_2 group norm, give components that are sparse (due to the group norm) and also localized and smooth (due to the added Laplacian term).

By adding the new Laplacian term and removing the spatially varying regularization strengths Λ_{ij}, we end up with the following optimization problem:

$$\arg\min_{\mathbf{U},\mathbf{W}} \|\mathbf{Y} - \mathbf{U}\mathbf{W}\|_F^2 + \frac{\lambda_1}{2}\operatorname{Tr}(\mathbf{U}^T\mathbf{L}\mathbf{U}) + \lambda_2 \sum_{i=1}^{n}\sum_{j=1}^{d} \|\mathbf{u}_{ij}\|_2 \qquad \text{s.t. } \mathcal{V}(\mathbf{W}) \tag{5.12}$$

where the ℓ_1/ℓ_2 group norm is defined as before, without the varying regularization strengths, the constraints $\mathcal{V}$ are defined in Equation (5.2) and the matrix $\mathbf{L} \in \mathbb{R}^{3n\times 3n}$ contains the cotangent weights:

$$\begin{aligned} l_{ii} &= -\sum_j l_{ij} \\ l_{ij} &= \begin{cases} -\frac{1}{2}(\cot(\alpha_{ij}) + \cot(\beta_{ij})) & \text{for all neighbors } j \text{ of } i \\ 0 & \text{for all other entries} \end{cases} \end{aligned} \tag{5.13}$$

By computing the cotangent weight matrix on the rest shape, instead of computing the traditional Laplacian matrix, we get a Laplacian term that is invariant to meshes with irregular triangulation. Note that each row of $\mathbf{L}$ from Equation (5.13) is repeated three times and shifted right, so that the Laplacian term acts on the x, y and z coordinates of each vertex simultaneously. Sections 5.3.1 and 5.3.2 specify how the components and weights from Equation (5.12) are initialized and optimized. We will refer to this method as SPLOCS_lap.

5.3.1. Initialization phase

The results in Section 7.3.2 and the discussion in Chapter 8 show that the SPLOCS_lap method is still very dependent on the initialization phase. Different methods of initializing the components will yield very different results. Therefore, we decided to initialize the components in the same way as SPLOCS was initialized. Thus, the initialization phase of SPLOCS_lap still makes use of the spatially varying regularization strengths Λ_{ij} from Equation (5.5). This term is then removed in the optimization phase and replaced by the Laplacian term.

5.3.2. Optimization phase

Similar to SPLOCS, we optimize the problem using an iterative refinement method, which optimizes for $\mathbf{W}$ and $\mathbf{U}$ in an alternating fashion. The optimization of the weights given fixed components is solved in the same way as in real SPLOCS, using a block-coordinate descent algorithm. The optimization of $\mathbf{U}$ is again tackled using ADMM, where we now introduce two dual variables $\mathbf{D}, \mathbf{E} \in \mathbb{R}^{3n \times d}$ due to the added Laplacian term. Adding an additional dual variable for solving ADMM is inspired by [38]. The objective from Equation (5.12) with respect to $\mathbf{U}$ can then be reformulated as:

$$\arg\min_{\mathbf{U},\mathbf{D},\mathbf{E}} \|\mathbf{Y} - \mathbf{UW}\| + \frac{\lambda_1}{2} \operatorname{Tr}(\mathbf{D}^T \mathbf{L} \mathbf{D}) + \lambda_2 \sum_{i=1}^{n} \sum_{j=1}^{d} \|\mathbf{e}_{ij}\|_2 \qquad \text{s.t. } \mathbf{U} - \mathbf{D} = \mathbf{0}, \quad \mathbf{U} - \mathbf{E} = \mathbf{0} \tag{5.14}$$

This is solved using ADMM by iterating the four steps shown in Algorithm 5. The first step, solving for $\mathbf{U}$, can again be solved by taking the derivative and setting it to zero. The derivate of the first step is as follows:

$$\frac{\partial}{\partial \mathbf{U}} \left(\|\mathbf{Y} - \mathbf{UW}\|_F^2 + \frac{\rho}{2} \left\| \begin{bmatrix} \mathbf{U} \\ \mathbf{U} \end{bmatrix} - \begin{bmatrix} \mathbf{D} \\ \mathbf{E} \end{bmatrix} + \begin{bmatrix} \mathbf{V_D} \\ \mathbf{V_E} \end{bmatrix} \right\|_F^2 \right) = 2 \cdot \mathbf{UWW}^T - 2 \cdot \mathbf{YW}^T + \rho(2 \cdot \mathbf{U} - \mathbf{D} + \mathbf{V_D} - \mathbf{E} + \mathbf{V_E}) \tag{5.15}$$

By setting this derivative to zero, we obtain the optimal $\mathbf{U}$:

$$\begin{aligned} 2 \cdot \mathbf{UWW}^T - 2 \cdot \mathbf{YW}^T + \rho(2 \cdot \mathbf{U} - \mathbf{D} + \mathbf{V_D} - \mathbf{E} + \mathbf{V_E}) &= 0 \\ 2 \cdot \mathbf{UWW}^T + 2\rho \cdot \mathbf{U} &= 2 \cdot \mathbf{YW}^T + \rho(\mathbf{D} - \mathbf{V_D} + \mathbf{E} - \mathbf{V_E}) \\ \mathbf{U} &= (2 \cdot \mathbf{YW}^T + \rho(\mathbf{D} - \mathbf{V_D} + \mathbf{E} - \mathbf{V_E}))(2 \cdot \mathbf{WW}^T + 2\rho \cdot \mathbb{I})^{-1} \end{aligned} \tag{5.16}$$

Similar to SPLOCS and QSPLOCS, we use Cholesky decomposition to factorize the $(2 \cdot \mathbf{WW}^T + 2\rho \cdot \mathbb{I})$ term. Optimizing with respect to the dual variable $\mathbf{D}$ is again a linear least squares problem, which can be solved by setting the derivative to zero. The derivative of this second step of Algorithm 5 is given by:

$$\frac{\partial}{\partial \mathbf{D}} \left(\frac{\lambda_1}{2} \operatorname{Tr}(\mathbf{D}^T \mathbf{L} \mathbf{D}) + \frac{\rho}{2} \|\mathbf{U} - \mathbf{D} + \mathbf{V_D}\|_F^2 \right) = \lambda_1 \cdot \mathbf{LD} + \rho(\mathbf{D} - \mathbf{U} - \mathbf{V_D}) \tag{5.17}$$

The optimal $\mathbf{D}$ is then obtained by:

$$\begin{aligned} \lambda_1 \cdot \mathbf{LD} + \rho(\mathbf{D} - \mathbf{U} - \mathbf{V_D}) &= 0 \\ \lambda_1 \cdot \mathbf{LD} + \rho \cdot \mathbf{D} &= \rho(\mathbf{U} + \mathbf{V_D}) \\ \mathbf{D} &= (\lambda_1 \cdot \mathbf{L} + \rho \mathbb{I})^{-1} \rho(\mathbf{U} + \mathbf{V_D}) \end{aligned} \tag{5.18}$$

Finally, optimization for the dual variable $\mathbf{E}$ is achieved using the proximal operator of the ℓ_1/ℓ_2 group norm [2], similar to the one used in SPLOCS. The only difference is that this version uses a fixed weight λ_2 instead of the spatially varying regulation strengths:

$$e_{ij} \leftarrow \text{prox}(\mathbf{u}_{ij} + \mathbf{v}_{ij}^{\mathbf{E}}) = \left(1 - \frac{\lambda_2}{\rho \|\mathbf{u}_{ij} + \mathbf{v}_{ij}^{\mathbf{E}}\|_2}\right)_+ (\mathbf{u}_{ij} + \mathbf{v}_{ij}^{\mathbf{E}}) \tag{5.19}$$

Algorithm 5: Optimization of $\mathbf{U}$ using ADMM for SPLOCS_lap.

Data: Data matrix $\mathbf{Y}$, initialized component matrix $\mathbf{U}$, weight matrix $\mathbf{W}$, Laplacian matrix $\mathbf{L}$, λ_1, λ_2, penalty parameter ρ and number of ADMM iterations k.
Result: Optimized components $\mathbf{U}$.

$\mathbf{V} = \begin{bmatrix}\mathbf{V_D}\\ \mathbf{V_E}\end{bmatrix} \leftarrow \begin{bmatrix}\mathbf{0}\\ \mathbf{0}\end{bmatrix}$;
$\mathbf{D} \leftarrow \mathbf{U}$;
$\mathbf{E} \leftarrow \mathbf{U}$;
for $i = 1, \dots, k$ **do**
  $\mathbf{U} \leftarrow \arg\min_{\mathbf{U}} \|\mathbf{Y} - \mathbf{UW}\|_F^2 + \frac{\rho}{2}\left\|\begin{bmatrix}\mathbf{U}\\ \mathbf{U}\end{bmatrix} - \begin{bmatrix}\mathbf{D}\\ \mathbf{E}\end{bmatrix} + \mathbf{V}\right\|_F^2$;
  $\mathbf{D} \leftarrow \arg\min_{\mathbf{D}} \frac{\lambda_1}{2}\operatorname{Tr}(\mathbf{D}^T\mathbf{LD}) + \frac{\rho}{2}\|\mathbf{U} - \mathbf{D} + \mathbf{V_D}\|_F^2$;
  $\mathbf{E} \leftarrow \arg\min_{\mathbf{E}} \lambda_2 \sum_{i=1}^{n}\sum_{j=1}^{d} \|\mathbf{e}_{ij}\|_2 + \frac{\rho}{2}\|\mathbf{U} - \mathbf{E} + \mathbf{V_E}\|_F^2$;
  $\mathbf{V} \leftarrow \begin{bmatrix}\mathbf{V_D}\\ \mathbf{V_E}\end{bmatrix} + \begin{bmatrix}\mathbf{U}\\ \mathbf{U}\end{bmatrix} - \begin{bmatrix}\mathbf{D}\\ \mathbf{E}\end{bmatrix}$;
end

5.4. QSPLOCS using Laplacian term (QSPLOCS_lap)

Similar to what we showed in QSPLOCS, it is also possible to implement SPLOCS_lap using quaternions. We will refer to this method as QSPLOCS_lap. This section briefly discusses the changes with respect to SPLOCS_lap.

The quaternion optimization problem containing the new Laplacian term is:

$$\arg\min_{\mathbf{U},\mathbf{W}} \|\mathbf{Y} - \mathbf{UW}\|_F^2 + \frac{\lambda_1}{2}\operatorname{Tr}(\mathbf{U}^*\mathbf{LU}) + \lambda_2 \sum_{i=1}^{n}\sum_{j=1}^{d} |u_{ij}| \qquad \text{s.t. } \mathcal{V}(\mathbf{W}) \tag{5.20}$$

where the differences with SPLOCS_lap are the use of the conjugate transpose operator and the replacement of the group norm by the simpler ℓ_1 norm. The Laplacian matrix $\mathbf{L} \in \mathbb{R}^{n\times n}$ is defined in the same way as in Equation 5.13, which can be seen as a quaternionic matrix with imaginary parts equal to zero. For reasons discussed in Section 5.3.1, the components and weights of this method are initialized in the exact same way as in the QSPLOCS method, shown in Algorithm 3.

The optimization phase again solves for $\mathbf{U}$ and $\mathbf{W}$ in an iterative fashion, where the components are optimized using ADMM. The steps of solving this are shown in Algorithm 6 and are very similar to the steps performed in SPLOCS_lap (Algorithm 5). The first step, solving for $\mathbf{U}$, has the following derivative:

$$\frac{\partial}{\partial \mathbf{U}}\left(\|\mathbf{Y} - \mathbf{UW}\|_F^2 + \frac{\rho}{2}\left\|\begin{bmatrix}\mathbf{U}\\ \mathbf{U}\end{bmatrix} - \begin{bmatrix}\mathbf{D}\\ \mathbf{E}\end{bmatrix} + \begin{bmatrix}\mathbf{V_D}\\ \mathbf{V_E}\end{bmatrix}\right\|_F^2\right) = 2\cdot\overline{\mathbf{UWW}^*} - 2\cdot\overline{\mathbf{YW}^*} + \rho(2\cdot\overline{\mathbf{U}} - \overline{\mathbf{D}} + \overline{\mathbf{V_D}} - \overline{\mathbf{E}} + \overline{\mathbf{V_E}}) \tag{5.21}$$

Solving for zero gives the optimal $\mathbf{U}$, which is essentially the same as Equation (5.16) apart from the conjugate transpose operator:

$$\begin{aligned} 2\cdot\overline{\mathbf{UWW}^*} - 2\cdot\overline{\mathbf{YW}^*} + \rho(2\cdot\overline{\mathbf{U}} - \overline{\mathbf{D}} + \overline{\mathbf{V_D}} - \overline{\mathbf{E}} + \overline{\mathbf{V_E}}) &= 0 \\ 2\cdot\mathbf{UWW}^* + 2\rho\cdot\mathbf{U} &= 2\cdot\mathbf{YW}^* + \rho(\mathbf{D} - \mathbf{V_D} + \mathbf{E} - \mathbf{V_E}) \\ \mathbf{U} &= (2\cdot\mathbf{YW}^* + \rho(\mathbf{D} - \mathbf{V_D} + \mathbf{E} - \mathbf{V_E}))(2\cdot\mathbf{WW}^* + 2\rho\cdot\mathbb{I})^{-1} \end{aligned} \tag{5.22}$$

Again, we use quaternion Cholesky decomposition to factorize the term containing the weight matrix. In a similar way, one can show that the optimal $\mathbf{D}$ is found in the exact same way as in SPLOCS_lap, shown in Equation (5.18).

Algorithm 6: Optimization of $\mathbf{U}$ using ADMM for QSPLOCS_lap.

Data: Data matrix $\mathbf{Y}$, initialized component matrix $\mathbf{U}$, weight matrix $\mathbf{W}$, Laplacian matrix $\mathbf{L}$, λ_1, λ_2, penalty parameter ρ and number of ADMM iterations k.
Result: Optimized components $\mathbf{U}$.

$\mathbf{V} = \begin{bmatrix}\mathbf{V_D}\\ \mathbf{V_E}\end{bmatrix} \leftarrow \begin{bmatrix}\mathbf{0}\\ \mathbf{0}\end{bmatrix}$;
$\mathbf{D} \leftarrow \mathbf{U}$;
$\mathbf{E} \leftarrow \mathbf{U}$;
for $i = 1, \dots, k$ **do**
  $\mathbf{U} \leftarrow \arg\min_{\mathbf{U}} \|\mathbf{Y} - \mathbf{UW}\|_F^2 + \frac{\rho}{2}\left\|\begin{bmatrix}\mathbf{U}\\ \mathbf{U}\end{bmatrix} - \begin{bmatrix}\mathbf{D}\\ \mathbf{E}\end{bmatrix} + \mathbf{V}\right\|_F^2$;
  $\mathbf{D} \leftarrow \arg\min_{\mathbf{D}} \frac{\lambda_1}{2}\mathrm{Tr}(\mathbf{D}^*\mathbf{LD}) + \frac{\rho}{2}\|\mathbf{U} - \mathbf{D} + \mathbf{V_D}\|_F^2$;
  $\mathbf{E} \leftarrow \arg\min_{\mathbf{E}} \lambda_2 \sum_{i=1}^{n}\sum_{j=1}^{d}|\mathbf{e}_{ij}| + \frac{\rho}{2}\|\mathbf{U} - \mathbf{E} + \mathbf{V_E}\|_F^2$;
  $\mathbf{V} \leftarrow \begin{bmatrix}\mathbf{V_D}\\ \mathbf{V_E}\end{bmatrix} + \begin{bmatrix}\mathbf{U}\\ \mathbf{U}\end{bmatrix} - \begin{bmatrix}\mathbf{D}\\ \mathbf{E}\end{bmatrix}$;
end

Finally, the optimal $\mathbf{E}$ can be found using proximal operator of the ℓ_1 norm, similar as in QSPLOCS:

$$\begin{aligned} e_{ij} \leftarrow \text{prox}(u_{ij} + v_{ij}^{\mathbf{E}}) &= \left(1 - \frac{\lambda_2}{\rho|u_{ij} + v_{ij}^{\mathbf{E}}|}\right)_+ (u_{ij} + v_{ij}^{\mathbf{E}}) \\ &= \text{sign}(u_{ij} + v_{ij}^{\mathbf{E}})\left(|u_{ij} + v_{ij}^{\mathbf{E}}| - \frac{\lambda_2}{\rho}\right)_+ \end{aligned} \tag{5.23}$$

5.5. Projection onto constructed subspace

After obtaining the d components using one of the four methods discussed in this chapter, we can now project any set of samples onto this subspace by finding weight matrix $\mathbf{W}$ that minimizes:

$$\arg\min_{\mathbf{W}} \|\mathbf{Y} - \mathbf{UW}\|_F^2 \tag{5.24}$$

where $\mathbf{Y}$ contains the samples that we wish to project onto the sparse PCA subspace and $\mathbf{U}$ contains the d components. Note that $\mathbf{Y}$ should again be defined in terms of residual displacements from the rest shape $\overline{\mathbf{y}}$. A closed-form solution of this problem exists by finding the derivative with respect to $\mathbf{W}$ and setting this to zero:

$$\frac{\partial}{\partial \mathbf{W}}\|\mathbf{Y} - \mathbf{UW}\|_F^2 = 2\mathbf{U}^T\mathbf{UW} - 2\mathbf{U}^T\mathbf{Y} \tag{5.25}$$

$$\begin{aligned} 2\mathbf{U}^T\mathbf{UW} - 2\mathbf{U}^T\mathbf{Y} &= 0 \\ \mathbf{W} &= (\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{Y} \end{aligned} \tag{5.26}$$

When quaternions are used instead of real numbers, the optimal weights can be found by replacing the transpose operators in Equation (5.26) with the conjugate transpose. Note that the optimal weight computation differs to that from PCA and QPCA, as seen in Section 4.3, since the components found by those methods form an orthonormal basis. In that case, the optimal weight computation is simplified, since $(\mathbf{U}^T\mathbf{U})^{-1}$ is simply equal to the identity matrix.

The projection of the sample set onto the subspace is found by:

$$\begin{aligned} \mathbf{Y}_{proj} &= \mathbf{UW} \\ &= \mathbf{U}(\mathbf{U}^T\mathbf{U})^{-1}\mathbf{U}^T\mathbf{Y} \end{aligned} \tag{5.27}$$

This projection can then be defined in terms of vertex positions by adding the rest shape $\bar{\mathbf{y}}$ back to each column:

$$\mathbf{Y}_{proj} \leftarrow \mathbf{Y}_{proj} + \bar{\mathbf{y}} \tag{5.28}$$

In the quaternionic setting it is not guaranteed that $\mathbf{Y}_{proj}$ only contains pure quaternions. Since the real parts of these quaternions will only result in a higher reconstruction error, we simply set the real part of each quaternion to zero. This was also done for PCA and QPCA, as described in Section 4.3.

5.6. Tunable parameters

The four sparse subspace construction methods described in this chapter all take a number of user-specified parameters. For SPLOCS and QSPLOCS these parameters are:

- The required number d of deformation components.
- The sample that is used as the rest shape.
- d_{min} and d_{max}, the minimal and maximal geodesic distances for the support map.
- λ, the weight of the sparsity inducing norm.
- ρ, the penalty parameter used in ADMM.
- The number of iterations of ADMM and the number of iterations of the total optimization phase.

Likewise, SPLOCS_lap and QSPLOCS_lap use the following parameters:

- The required number d of deformation components.
- The sample that is used as the rest shape.
- d_{min} and d_{max}, the minimal and maximal geodesic distances for the support map (only used during initialization).
- λ_1, the weight of the Laplacian term.
- λ_2, the weight of the sparsity inducing norm.
- ρ, the penalty parameter used in ADMM.
- The number of iterations of ADMM and the number of iterations of the total optimization phase.

By making some changes to the datasets prior to constructing the subspaces, the selection of some of these parameters can be made a lot easier. First, we rescale each dataset so that the samples all fit inside a $[-0.5,\ 0.5]$ box, where each sample is centered around zero. This helps in selecting the d_{min} and d_{max} parameters, as it allows us to use the same distance values on each dataset. The spatially varying regularization strengths are then computed on the rest shape of this scaled version.

Next, we normalize $\mathbf{Y}$ by its standard deviation after subtracting the rest shape from each sample. This makes the algorithm more invariant to the different amounts of deformations present in different datasets. As a result, the selection of the weights for the Laplacian and sparsity inducing terms is simplified. This enables us to use the same weights on all datasets that contain approximately the same number of vertices. In cases where one dataset contains more vertices than another, we can correct for this difference by adjusting the weight of the Laplacian term. Both the data term and the sparsity inducing term scale approximately linearly with the number of vertices. The Laplacian term, on the other hand, remains almost constant when more vertices are added. Therefore, we only have to adjust the weight of this Laplacian term in order to correct for datasets containing a different number of vertices.

Strategies for adjusting the penalty parameter ρ are discussed in [8]. However, we found that fixing this parameter to $\rho = 10$ already works well on all the datasets we considered. Furthermore, we found that iterating ADMM 20 times and the total optimization phase 5 times gives acceptable results. This is further discussed in Sections 7.2.1 and 7.3.1, where convergence plots of different datasets are shown.

6

Results quaternion PCA

In this chapter, we evaluate the effectiveness of the QPCA based subspace. First, we perform tests to assess the reconstruction quality of both training data and unseen test data. This is done using quantitative evaluation in Section 6.2.1 as well as a visual comparison in Section 6.2.2. The former section also includes an overview of the computation times. Next, we perform similar tests in Section 6.3 to see how well the proposed method performs when all the training and test samples have been randomly rigidly rotated and translated in space. In these experiments, we evaluate to what extent the QPCA subspace is invariant to rigid motion of the samples. All these experiments are evaluated on both PCA and QPCA, after which a comparison is made. Finally, we investigate whether the mass-weighted QPCA technique is indeed robust to mesh irregularities. These experiments are performed in Section 6.4.

6.1. Datasets

In order to properly asses the performance and generality of the QPCA method, we compare it to PCA on a variety of captured datasets. The first dataset we used is the DFAUST dataset [7]. This dataset contains high-resolution scans of human subjects in motion, captured at 60 fps. Furthermore, we used the TCD Hands [20] and ACCAD [54] datasets, which are part of the larger AMASS database [35]. AMASS is a relatively new database that fits captured motion onto the SMPL+H model [44], which is a human model that can capture detailed full body and hand motions. The TCD Hands dataset specifically focuses on these hand motions, while the ACCAD dataset contains a range of full body motions. In order to make the comparisons more general, we also tested the methods on various animal datasets [52] as well as on a captured face dataset [66]. Note that the motion captures described above are already rigidly aligned, since the captures are taken from people and animals standing on a level floor.

For the sake of clarity, all experiments in this chapter will be performed on two datasets only:

- “C5 - Walk to run”, taken from ACCAD [54].
- “horse-gallop”, taken from the animal dataset [52].

Results of these experiments on various other datasets can be found in Appendix A. All experiments are performed using a MATLAB implementation. For quaternion algebra, we utilize the quaternion MATLAB toolbox `qtfm` [46]. Furthermore, the meshes are visualized with the help of the MATLAB toolbox for geometry processing, called `gptoolbox` [22].

6.2. Experiments on rigidly aligned datasets

Before investigating how well the QPCA technique performs on data that has not been rigidly aligned in a preprocessing step, we want to assess how the method compares to PCA when data is rigidly aligned. Since quaternions are able to describe rotations better than real numbers, it is interesting to examine whether the QPCA subspace is able to better describe the non-rigid motions (i.e. local rotations) present in the data.

First, the methods are compared quantitatively in Section 6.2.1. The reconstruction accuracies of both methods are evaluated and the percentage of variance explained by each component is given. This section also includes a comparison of computation times on various datasets. Next, a visual comparison in given in Section 6.2.2.

6.2.1. Quantitative evaluation

As a first experiment, we evaluate the reconstruction accuracies on rigidly aligned datasets. For this, we train both PCA and QPCA on the two datasets mentioned in Section 6.1, where each sample is centered around zero as described in Section 4.4.1. The samples are then reconstructed using a varying number of components, after which the reconstruction error is computed. The error function used is based on the root-mean-square error:

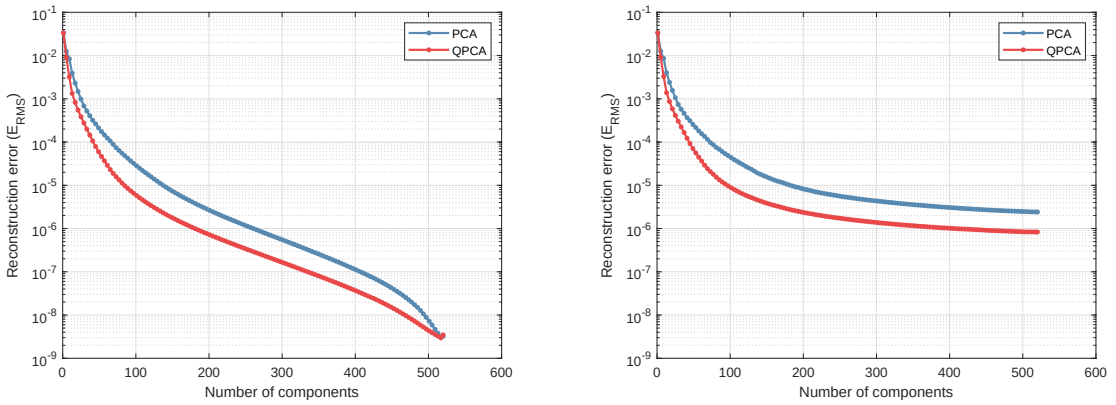
$$E_{RMS} = \sqrt{\frac{\sum_{i=1}^{f} \left\| \mathbf{y}_i - \mathbf{y}_i^{proj} \right\|_M^2}{f \cdot \|\mathbf{1}\|_M^2}} \tag{6.1}$$

where $\mathbf{Y}$ is the input data, $\mathbf{Y}_{proj}$ is the projection of $\mathbf{Y}$ onto the subspace using a varying number of components, $\mathbf{1}$ denotes the all-ones vector and $\|\cdot\|_M^2$ is the mass-weighted ℓ_2 norm. The methods are evaluated using 5-fold cross validation. This means that both PCA and QPCA are trained on 80% of the data, while the remainder of each run is used to test on unseen data.

Results of this experiment performed on the “C5 - Walk to run” and “horse-gallop” datasets can be found in Figures 6.1 and 6.2 respectively. Furthermore, the variance distributions of these two motion captures are plotted in Figures 6.3 and 6.4, where the left side shows the percentage of variance explained by each component taken from the singular values and the right side shows the cumulative percentage of variance explained. Results obtained from other datasets can be found in Appendices A.1.1 and A.1.2.

Figures 6.1 and 6.2 show that QPCA is able to reconstruct the frames more accurately using fewer components than PCA can. The difference in terms of reconstruction accuracy seems to be even slightly higher on unseen test data. It should be noted that the orthonormal basis found by QPCA is four-dimensional, while the basis found by PCA is not. This means that the QPCA method requires $4/3$ times more memory to store the basis vectors than PCA does. Thus, it is useful to also plot the reconstruction error with respect to the memory needed to store the basis vectors. This is shown in Figures 6.5 and 6.6. From these figures it becomes clear that the reconstruction accuracy of QPCA is in most cases still higher than PCA when storage capacity is limited, especially on the unseen test data. When more components are selected on the training data, PCA seems to be the preferred method on these two datasets.

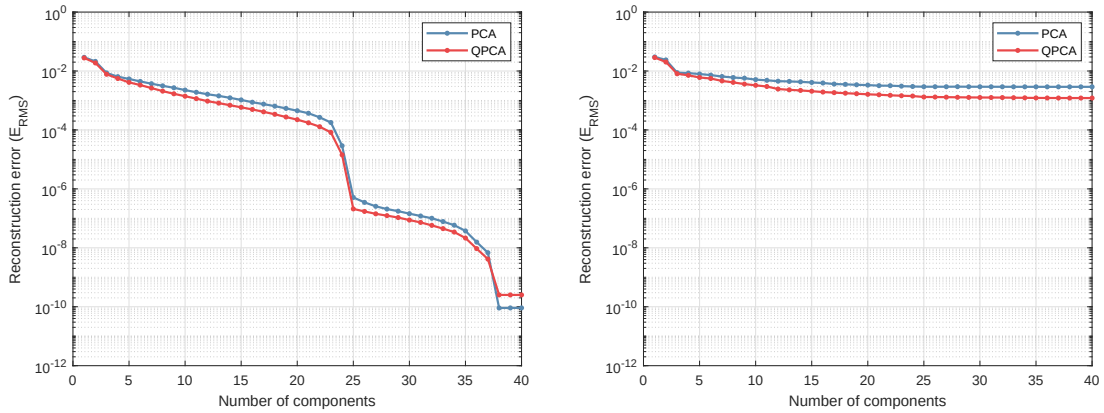
In conclusion, we can say that QPCA is able to describe the non-rigid motions in the datasets better than PCA is able to. Using the same amount of components, QPCA is able to describe a richer space of deformations, leading to an increase in reconstruction accuracy. Even if we take into account the memory space required to store the components, QPCA is often still the preferred method, especially when applied to unseen test data. This indicates that the subspace construction method indeed benefits from the use of quaternions, as we are able to store more information using the same amount of storage. This improvement can be explained by the properties of quaternions, that take into account the correlation between the x, y and z coordinates of the vertices.



(a) E_{RMS} error computed on 520 training frames.

(b) E_{RMS} error computed on 130 (unseen) test frames.

Figure 6.1: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the “C5 - Walk to run” motion capture from ACCAD [54]. 5-fold cross validation is used.



(a) E_{RMS} error computed on 40 training frames.

(b) E_{RMS} error computed on 9 (unseen) test frames.

Figure 6.2: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the “horse-gallop” motion capture from the animal dataset [52]. 5-fold cross validation is used.

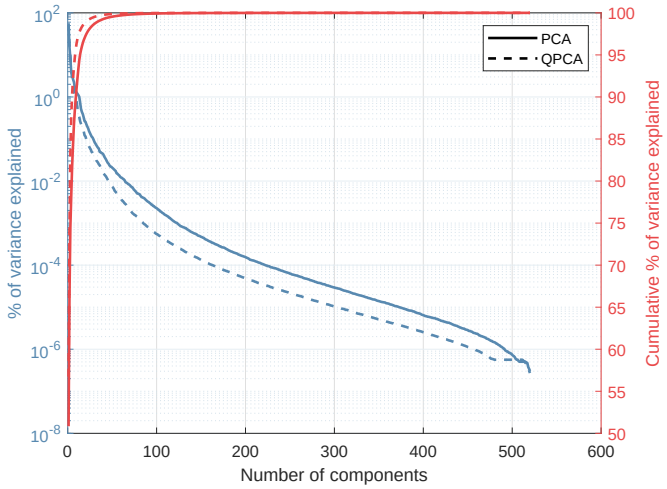


Figure 6.3: (Cumulative) percentage of variance explained per component, computed on the “C5 - Walk to run” motion capture from ACCAD [54].

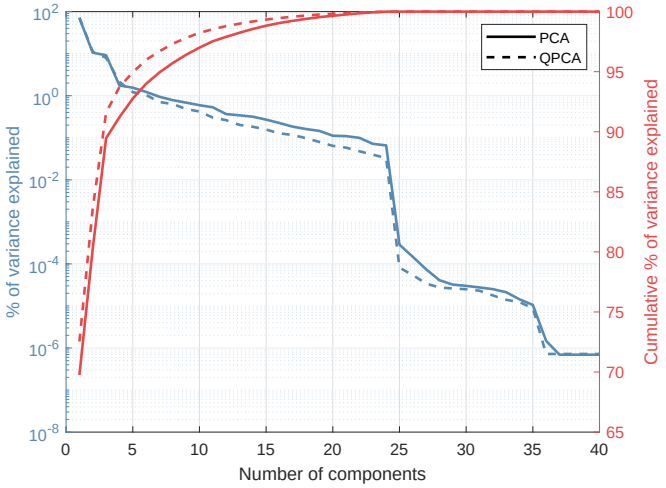
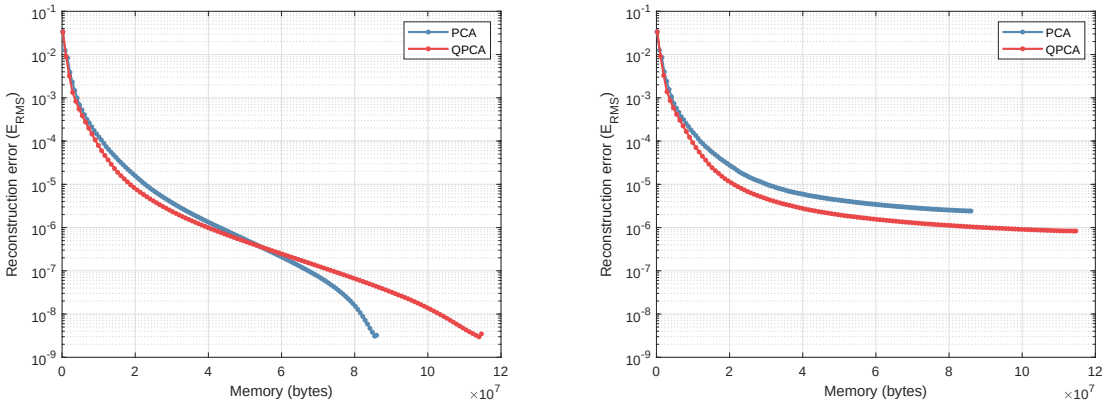


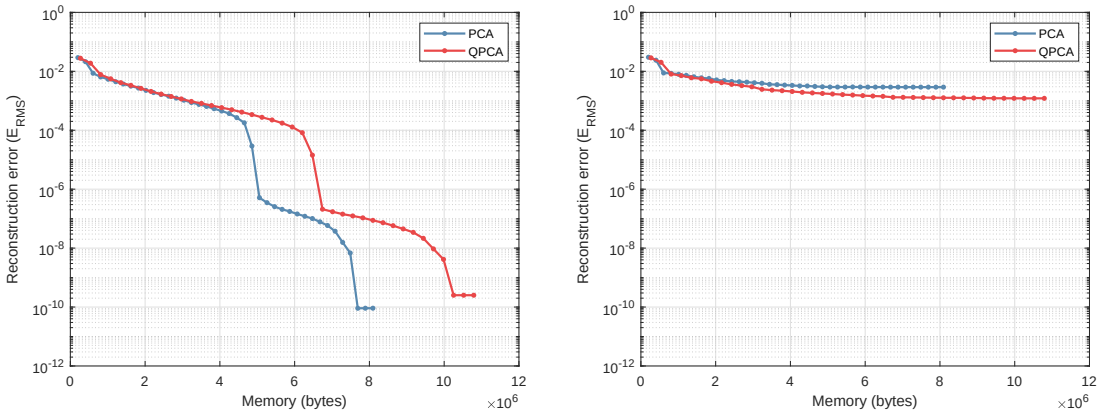
Figure 6.4: (Cumulative) percentage of variance explained per component, computed on the “horse-gallop” motion capture from [52].



(a) E_{RMS} error computed on 520 training frames.

(b) E_{RMS} error computed on 130 (unseen) test frames.

Figure 6.5: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the “C5 - Walk to run” motion capture from ACCAD [54]. 5-fold cross validation is used.



(a) E_{RMS} error computed on 40 training frames.

(b) E_{RMS} error computed on 9 (unseen) test frames.

Figure 6.6: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the “horse-gallop” motion capture from the animal dataset [52]. 5-fold cross validation is used.

As a second experiment, it is also interesting to compare the methods based on their computation time. Table 6.1 shows the computation times of both methods on a variety of datasets. The average times were taken over five runs, where both methods make use of the (quaternion) method of snapshots. Furthermore, all measurements are performed on the same laptop.

As expected, PCA is faster on all datasets considered, which is a result of quaternion multiplication being slower than real multiplication. Furthermore, the SVD methods used by both PCA and QPCA, which are applied to the Gram matrix, also differ in speed. The SVD method used in real PCA is a built-in function in MATLAB, which has been optimized to perfection. The SVD function used in QPCA, however, is part of the `qtfm` MATLAB toolbox [46] and based on [47]. Thus, this algorithm is most likely less efficient. Since subspace construction is often performed in an offline phase, the difference in computation times is not a very large issue. The largest observed computation time of 45.18s is still very reasonable.

Table 6.1: Comparison of computation times when using PCA and QPCA. The table shows the number of vertices n, the number of frames f and the computation times for both methods using the (quaternion) method of snapshots. i7 2.50GHz, 8GB RAM, Matlab implementation.

Mesh	n	f	Times for PCA	Times for QPCA
“horse-gallop” [52]	8431	49	0.04s	0.63s
“face” [66]	23725	385	0.34s	6.94s
“Sign D poses” [20]	6890	601	0.22s	34.38s
“C5 - Walk to run” [54]	6890	650	0.30s	45.18s
“E5 - Hook left poses” [54]	6890	327	0.08s	3.60s
“jumping-jacks-50022” [7]	6890	308	0.07s	3.18s
“jiggle-on-toes-50004” [7]	6890	239	0.04s	1.75s

6.2.2. Visual comparison

Besides performing a quantitative evaluation, one can also make a visual comparison between the two methods. This makes it possible to interpret the different components better and to understand the differences between the PCA and QPCA subspaces. For consistency purposes, we perform all visualization experiments on the “C5 - Walk to run” dataset only. More visualization results can be found in Appendix A.1.3.

As a first experiment, we visualize five principal components of both methods. Since we are working with vertex positions instead of vertex displacements, the first component actually represents the mean shape of the dataset. Therefore, we actually plot the projection onto the first component in combination with the other components. Figure 6.7 shows five components of PCA, where each component is displayed with weights $-\sqrt{\lambda_i}$ and $\sqrt{\lambda_i}$. Here, λ_i denotes the ith eigenvalue. Visualization of the QPCA components can be found in Figures 6.8-6.11. Since each principal component and corresponding weight is quaternion-valued and can thus be represented by a four-dimensional real space, it is not possible to visualize each component in one single image. Therefore, we plot models corresponding to $-\sqrt{\lambda_i}$ and $\sqrt{\lambda_i}$ along the real part and the three imaginary parts of the ith principal component. Each sample projected onto one of these modes can then be represented as a linear combination of these four deformations.

When comparing Figure 6.7 to Figure 6.8 we notice that the 2nd component from PCA shows approximately the same deformation as the real part (left image) of the 2nd QPCA component. This means that besides describing the same deformation as the PCA component, the QPCA component is additionally able to describe three other deformations corresponding to multiplication with weights containing the three imaginary parts. Therefore, QPCA is able to describe a broader range of deformations in this single component. This same property also applies to the 4th component, as can be seen by comparing the 4th PCA component to the QPCA component in Figure 6.9. However, from the 5th component onwards, this property does not seem to hold anymore. If we compare the 5th component of both methods, we notice that these do not look similar. This has to do with the fact that the 5th PCA component is already partially described by the previous QPCA components. The elongated right leg of PCA is, for instance, also visible along the third dimension of the 2nd QPCA component. Thus, the QPCA method is able to describe more variance by showing a different deformation. This is why the 5th QPCA component, shown in Figure 6.10, shows some resemblance to the 6th PCA component. Thus, the first 5 QPCA components are able to approximately describe the same set of deformations

as the first 6 PCA components. This is yet another indicator that the QPCA method is able to describe a richer space of deformations using less components. If we compare higher component numbers, such as the 16th component of both methods, no similarities can be noticed anymore.

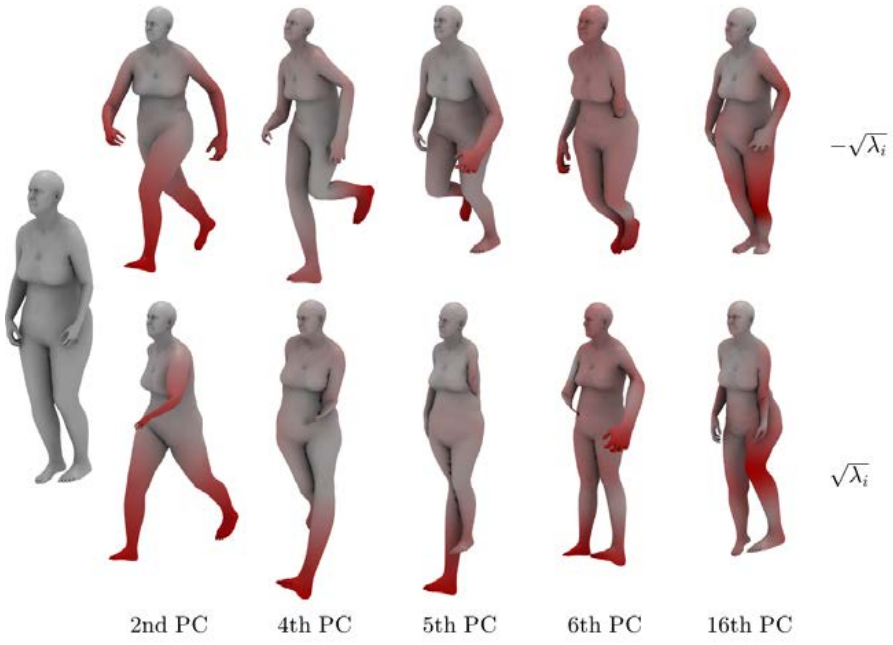


Figure 6.7: Visualization of PCA: 5 modes of deformation on the "C5 - Walk to run" motion capture from ACCAD [54]. Models corresponding to $-\sqrt{\lambda_i}$ (top) and $\sqrt{\lambda_i}$ (bottom) along the ith principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

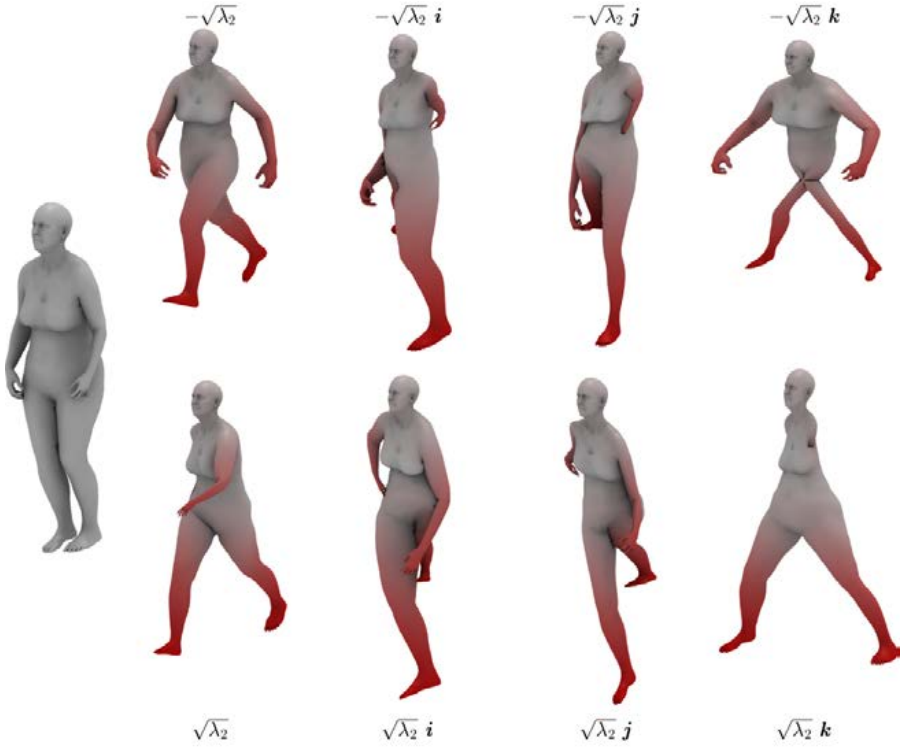


Figure 6.8: Visualization of QPCA: The 2nd quaternion mode of deformation on the "C5 - Walk to run" motion capture from ACCAD [54]. Models corresponding to $-\sqrt{\lambda_2}$ (top) and $\sqrt{\lambda_2}$ (bottom) along the four dimensions of the 2nd principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

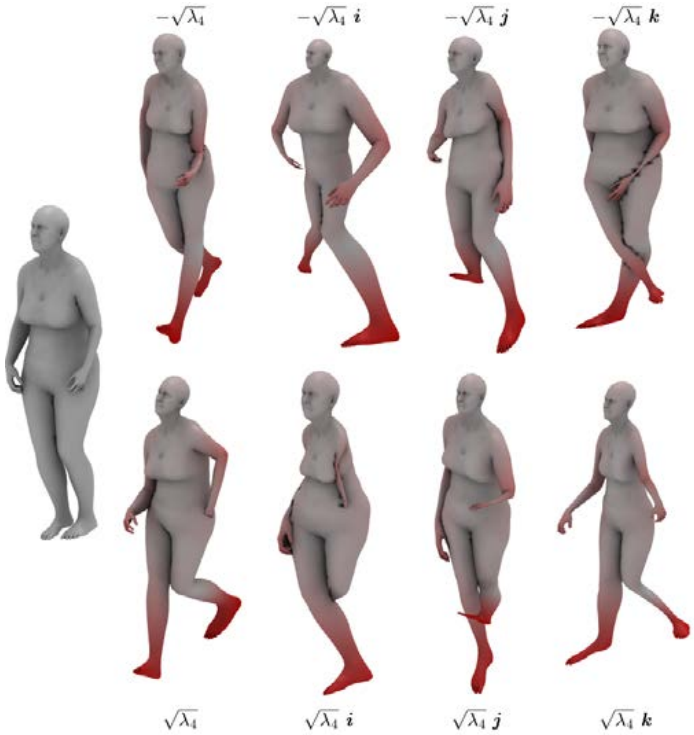


Figure 6.9: Visualization of QPCA: The 4th quaternion mode of deformation on the “C5 - Walk to run” motion capture from ACCAD [54]. Models corresponding to $-\sqrt{\lambda_4}$ (top) and $\sqrt{\lambda_4}$ (bottom) along the four dimensions of the 4th principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

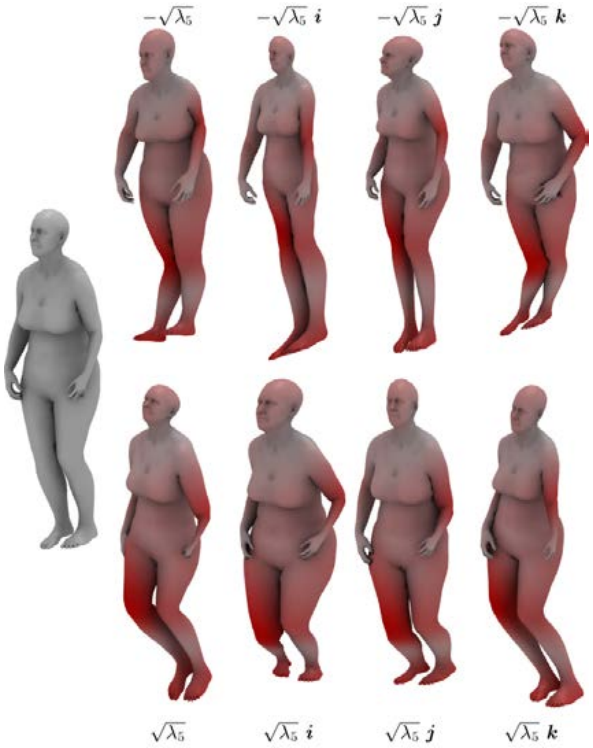


Figure 6.10: Visualization of QPCA: The 5th quaternion mode of deformation on the “C5 - Walk to run” motion capture from ACCAD [54]. Models corresponding to $-\sqrt{\lambda_5}$ (top) and $\sqrt{\lambda_5}$ (bottom) along the four dimensions of the 5th principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

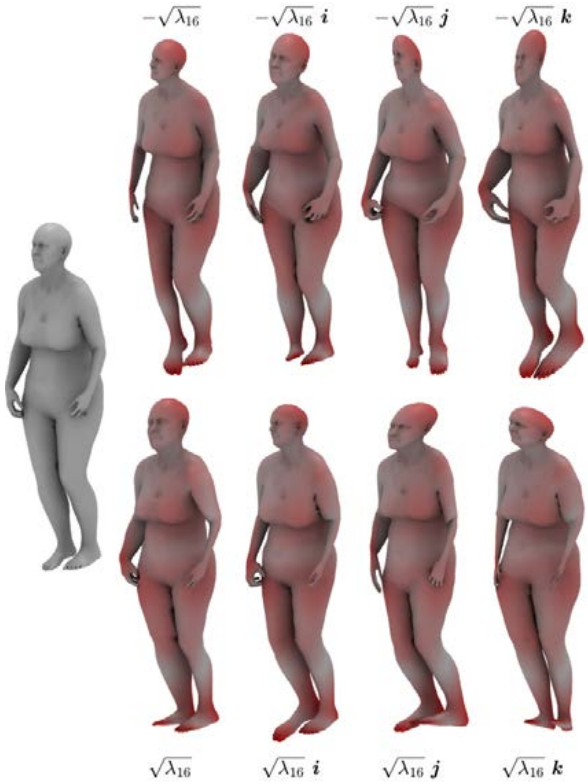


Figure 6.11: Visualization of QPCA: The 16th quaternion mode of deformation on the “C5 - Walk to run” motion capture from ACCAD [54]. Models corresponding to $-\sqrt{\lambda_{16}}$ (top) and $\sqrt{\lambda_{16}}$ (bottom) along the four dimensions of the 16th principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

As a second experiment, in order to make the differences between the methods more clear, we project a single sample of the dataset onto the first 1 to 9 components from both PCA and QPCA. This is visualized in Figures 6.12 and 6.13, where the projection onto the subspace (solid color) as well as the original frame (transparent) are shown.

It is evident from this figure that QPCA is able to reconstruct the sample better using less components. Using only 7 components, the QPCA projection is already better aligned with the original frame than the PCA projection is using 9 components. Another interesting observation is that the first component of QPCA, which represents the mean shape of the capture sequence, is already slightly tilted forward to match the running pose. Thus, it seems that QPCA is able to describe small rotations in its components.

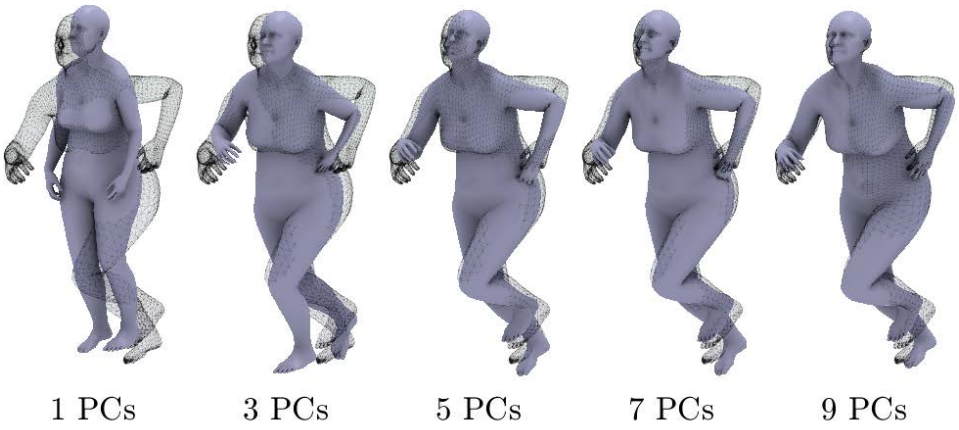


Figure 6.12: Visualization of PCA: Projection of one sample of the “C5 - Walk to run” motion capture from ACCAD [54] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

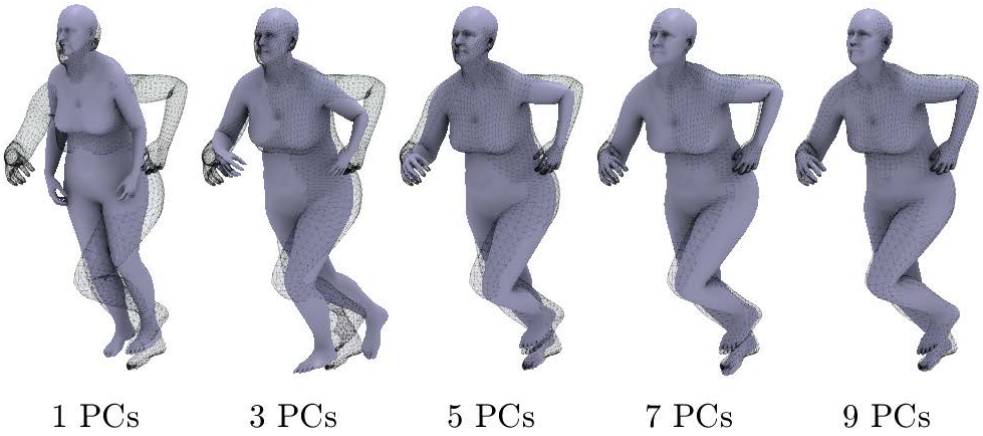


Figure 6.13: Visualization of QPCA: Projection of one sample of the “C5 - Walk to run” motion capture from ACCAD [54] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

6.3. Experiments on non-rigidly aligned datasets

Now that the methods have been compared on rigidly aligned datasets, it is interesting to investigate the performance of both methods on data that is not rigidly aligned. Since QPCA is invariant to right-multiplication with a unit quaternion, as described in Section 4.4, we expect this method to outperform PCA on these types of non-rigidly aligned datasets. In order to evaluate this, multiple experiments are performed similar to the ones performed in Section 6.2.

6.3.1. Quantitative evaluation

In the first experiment, we take a single frame of the “C5 - Walk to run” motion capture and create 100 copies of it that have been randomly rigidly translated and rotated in space. Thereafter, PCA and QPCA are applied and the frames are reconstructed using a varying number of components. The result of this is shown in Figure 6.14.

As expected, PCA needs 9 components in order to describe all randomly rotated frames, since this is equal to the number of elements in the 3-dimensional rotation matrix. The QPCA basis, however, only needs 3 components to describe any rotated frame.

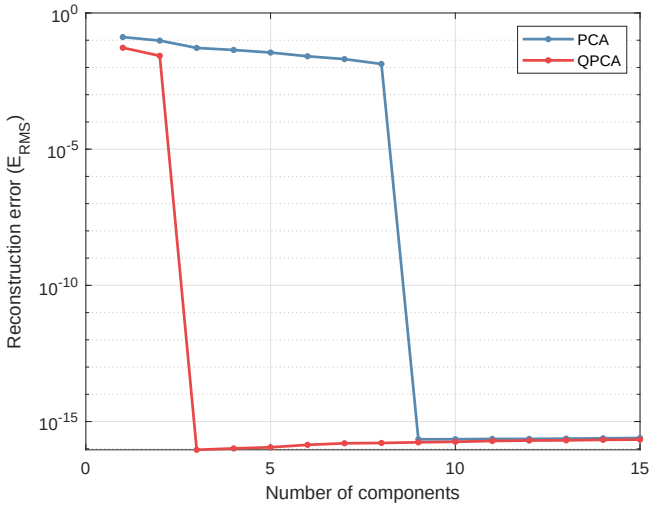
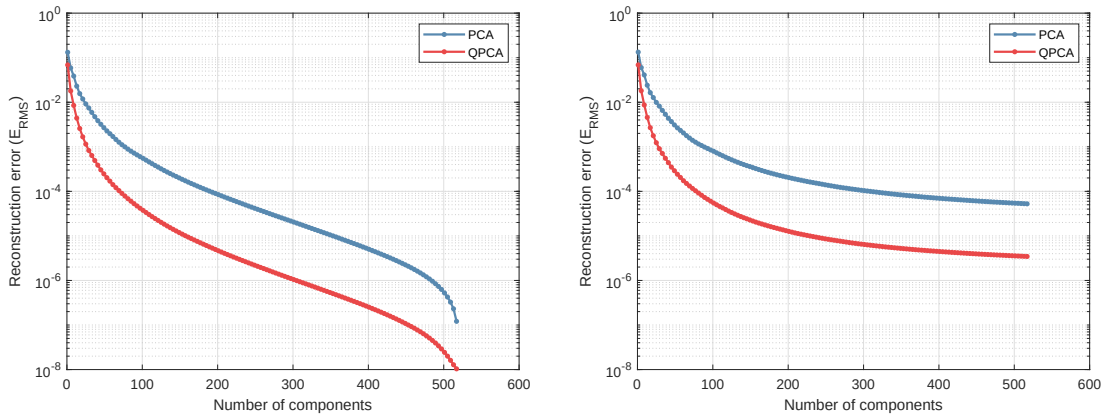


Figure 6.14: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the first frame of the “C5 - Walk to run” motion capture from ACCAD [54]. 5-fold cross validation is used.

As a second experiment, instead of taking one sample and creating copies that have been rigidly transformed, we take all samples of the dataset and apply a random rigid motion to each one. Then, PCA and QPCA are applied to the data and the reconstruction errors are computed, which is plotted in Figures 6.15 and 6.16. The variance distribution is displayed in Figures 6.17 and 6.18. Furthermore, a comparison in terms of memory needed to store the eigenvectors is given in Figures 6.19 and 6.20. Results on other datasets can be found in Appendices A.2.1 and A.2.2.

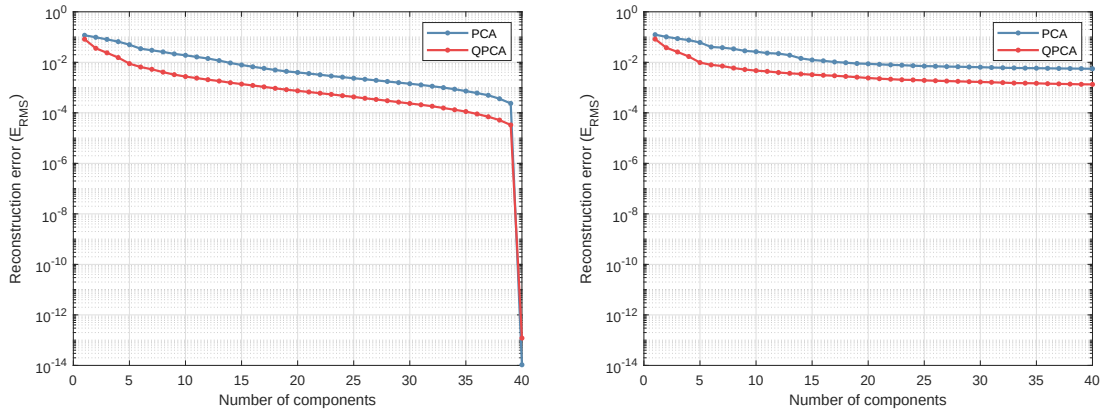
These figures really show the benefit of using QPCA on data that is not rigidly aligned. In comparison to PCA, the reconstruction accuracy is shown to improve by more than one order of magnitude for “C5 - Walk to run” and by almost an order of magnitude for “horse-gallop”. Even when comparing the methods based on the amount of memory needed to store the components, we see a large difference in performance. Thus, we can conclude that the QPCA subset is able to describe rigid rotations better and using less components than PCA. However, if we compare the QPCA results to the results of the rigidly aligned experiment, shown in Figures 6.1 and 6.2, we notice that the reconstruction errors on the rigidly aligned datasets are still lower. This indicates that QPCA is not completely rigid motion invariant and, therefore, we are not able to completely omit the rigid registration preprocessing step. Nevertheless, this does mean that QPCA will perform better on data that has not been perfectly rigidly aligned, as is often the case.



(a) E_{RMS} error computed on 520 training frames.

(b) E_{RMS} error computed on 130 (unseen) test frames.

Figure 6.15: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the “C5 - Walk to run” motion capture from ACCAD [54], where a random rigid motion is applied to each frame. 5-fold cross validation is used.



(a) E_{RMS} error computed on 40 training frames.

(b) E_{RMS} error computed on 9 (unseen) test frames.

Figure 6.16: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the “horse-gallop” motion capture from the animal dataset [52], where a random rigid motion is applied to each frame. 5-fold cross validation is used.

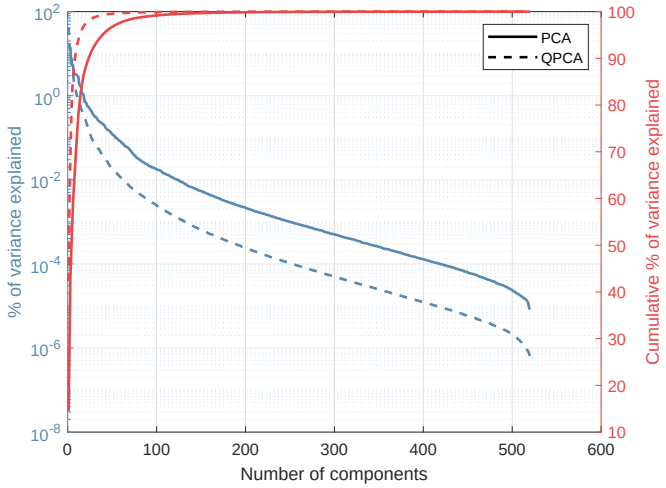


Figure 6.17: (Cumulative) percentage of variance explained per component, computed on the “C5 - Walk to run” motion capture from ACCAD [54]. A random rigid motion is applied to each sample of the dataset.

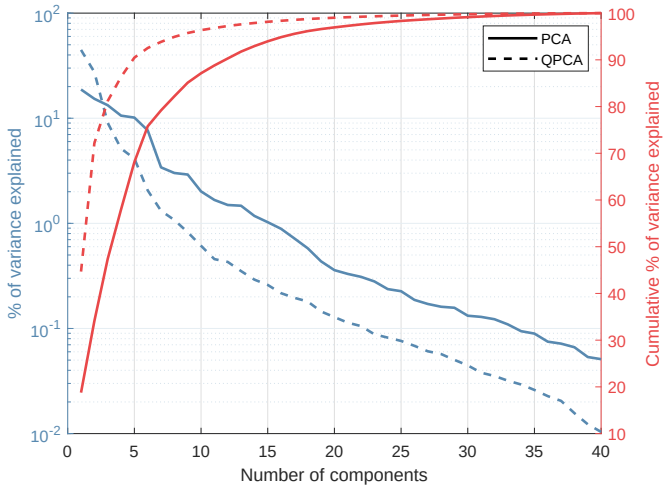
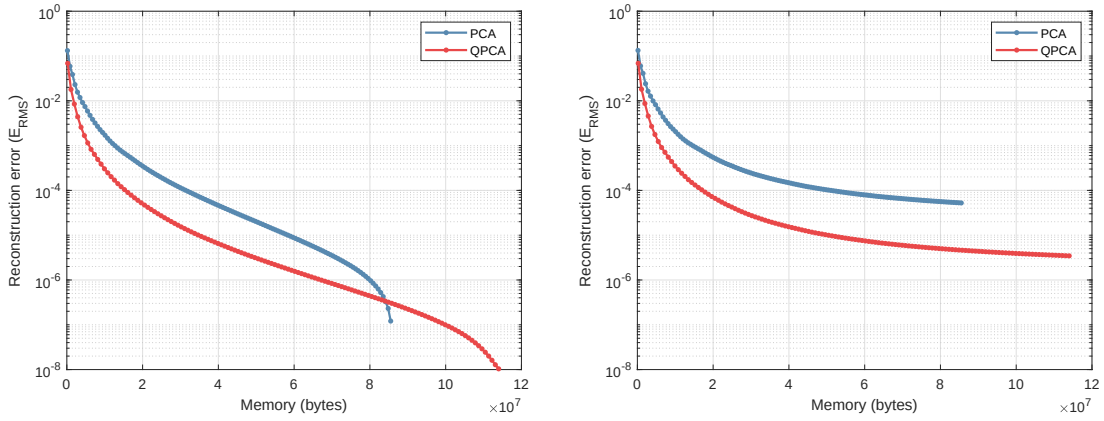


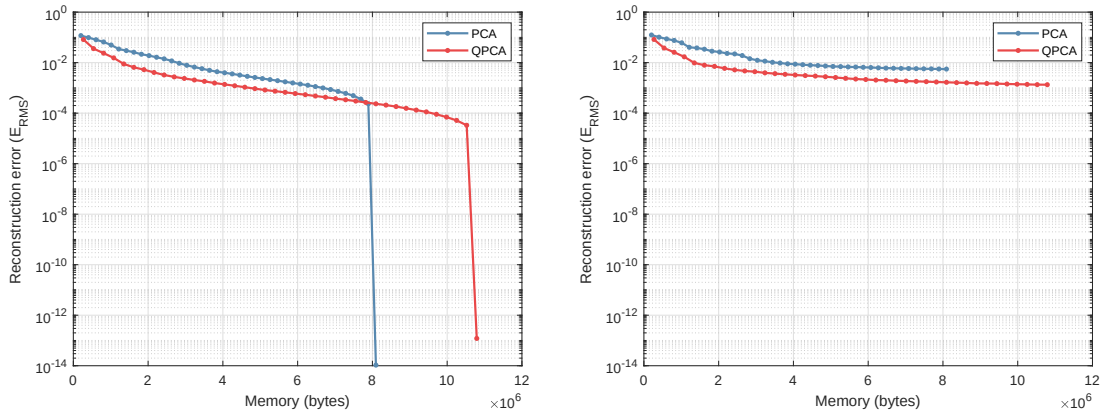
Figure 6.18: (Cumulative) percentage of variance explained per component, computed on the “horse-gallop” motion capture from the animal dataset [52]. A random rigid motion is applied to each sample of the dataset.



(a) E_{RMS} error computed on 520 training frames.

(b) E_{RMS} error computed on 130 (unseen) test frames.

Figure 6.19: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the “C5 - Walk to run” motion capture from ACCAD [54], where a random rigid motion is applied to each frame. 5-fold cross validation is used.



(a) E_{RMS} error computed on 40 training frames.

(b) E_{RMS} error computed on 9 (unseen) test frames.

Figure 6.20: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "horse-gallop" motion capture from the animal dataset [52], where a random rigid motion is applied to each frame. 5-fold cross validation is used.

Similar to what we showed in Section 6.2.1, we can also compare the computation times of both methods on non-rigidly aligned datasets. This is shown in Table 6.2. The computation times are very similar to the ones obtained on the rigidly aligned datasets. Again, PCA is shown to be more computationally efficient on all datasets considered for reasons discussed before.

Table 6.2: Comparison of computation times when using PCA and QPCA. The table shows the number of vertices n, the number of frames f and the computation times for both methods using the (quaternion) method of snapshots. i7 2.50GHz, 8GB RAM, Matlab implementation.

Mesh	n	f	Times for PCA	Times for QPCA
"horse-gallop" [52]	8431	49	0.03s	0.53s
"face" [66]	23725	385	0.34s	7.72s
"Sign D poses" [20]	6890	601	0.28s	35.23s
"C5 - Walk to run" [54]	6890	650	0.31s	46.29s
"E5 - Hook left poses" [54]	6890	327	0.09s	3.88s
"jumping-jacks-50022" [7]	6890	308	0.07s	3.37s
"jiggle-on-toes-50004" [7]	6890	239	0.04s	1.93s

6.3.2. Visual comparison

Similar to the experiments performed in Section 6.2.2, we can also visually compare the two methods on data that is not rigidly aligned. However, since all samples are randomly rotated in space, the resulting components will not be very interpretable. Therefore, it does not make sense to visualize the first five components of both methods. Instead, we only perform the second experiment, where a single sample is projected onto the first 1 to 9 principal components for both PCA and QPCA. Figures 6.21 and 6.22 show the results of this experiment on the "C5 - Walk to run" dataset. As before, more visual comparisons can be found in Appendix A.2.3.

Figures 6.21 and 6.22 appear to be in agreement with the results from Figure 6.14, where it was shown that PCA needs nine components to describe any rotated version of a sample, while QPCA only needs three components. From the visual comparison it seems that the first nine (PCA) or three (QPCA) components are used to get a "proper" rotation, while the remaining components are used to resemble the shape of the sample.

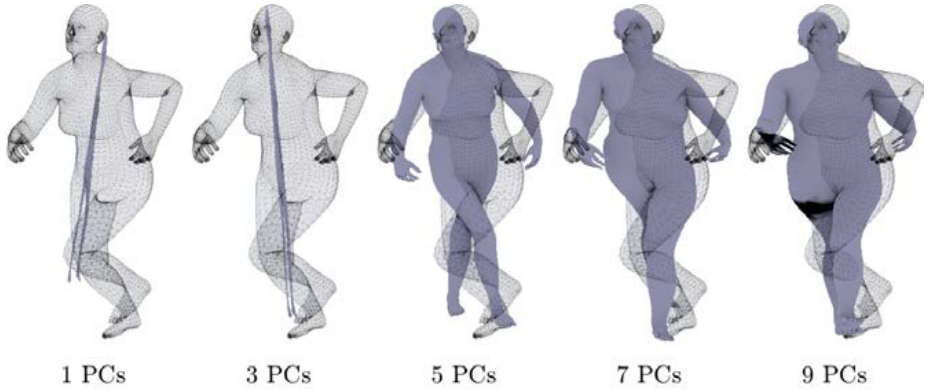


Figure 6.21: Visualization of PCA: Projection of one sample of the “C5 - Walk to run” motion capture from ACCAD [54] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

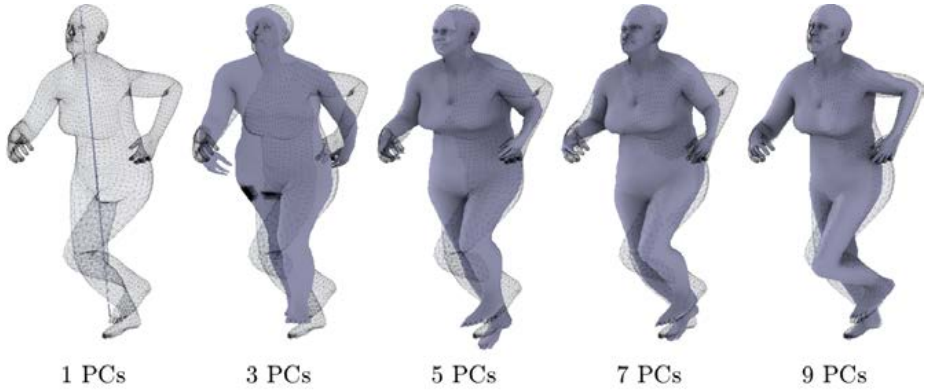


Figure 6.22: Visualization of QPCA: Projection of one sample of the “C5 - Walk to run” motion capture from ACCAD [54] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

6.4. Experiments on dataset with irregular triangulation

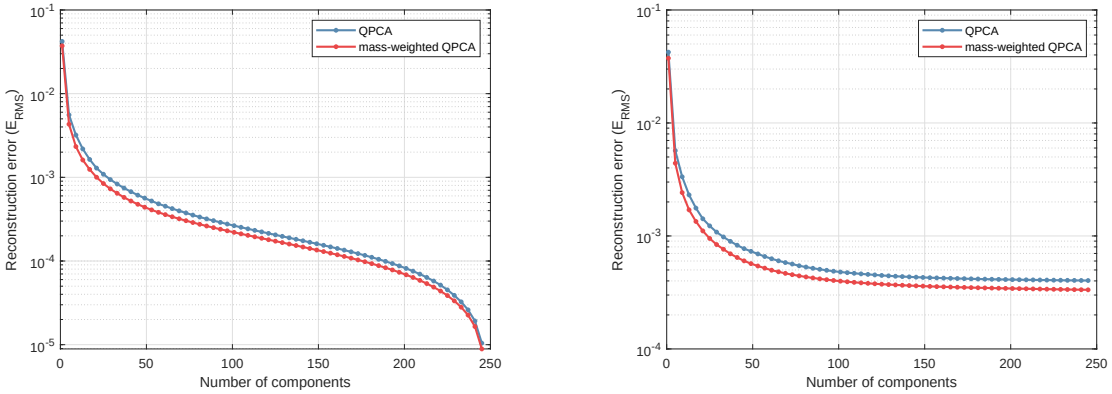
In Section 4.2.3 we introduced a mass-weighted QPCA method, which incorporates the mass matrix into QPCA in order to construct a mass-orthonormal basis. The idea is that this would make the method robust against remeshing, coarsening or refining of the meshes. In this section we perform various experiments on a dataset with irregular triangulation in order to investigate whether the method is indeed robust to meshes with irregular triangulation. The dataset we use in these experiments is the “jumping-jacks-50022” dataset from DFAUST [7], since this dataset has symmetric deformations on the left and right side of the mesh.

As a preprocessing step, we refine the left side of all sample meshes using subdivision. An example of this is shown in Figure 6.23. Next, we train both QPCA and mass-weighted QPCA on this dataset and evaluate the reconstruction accuracy of both methods. Figure 6.24 shows the results of this experiment. As before, 5-fold cross validation is used.

From this figure it becomes clear that the mass-weighted QPCA method is indeed able to reconstruct the samples more accurately than normal QPCA. This applies to both training data and unseen test data. However, it is not yet clear whether the mass-weighted QPCA technique is fully robust against mesh irregularities. This can be evaluated by comparing the results to results obtained on the original dataset, which has regular triangulation. If both datasets give the same reconstruction accuracy, this would indicate that mass-weighted QPCA is indeed robust against remeshing.

The reconstruction accuracy of the original “jumping-jacks-50022” dataset is plotted in Figure A.7 of Appendix A.1.1. By comparing this figure to Figure 6.24, we notice that mass-weighted QPCA on the irregular dataset gives approximately the same reconstruction accuracy as normal QPCA on the regular dataset. This indicates that mass-weighted QPCA is indeed robust to mesh irregularities.

Figure 6.23: Sample of the “jumping-jacks-50022” dataset from DFAUST [7], where the left side of the mesh is subdivided.



(a) E_{RMS} error computed on 247 training frames.

(b) E_{RMS} error computed on 61 (unseen) test frames.

Figure 6.24: Reconstruction error (y-axis) with respect to the number of components (x-axis) on the “jumping-jacks-50022” motion capture from DFAUST [7], where the left side of each sample is refined. 5-fold cross validation is used.

In order to get a better understanding of the differences between QPCA and mass-weighted QPCA, we can make a visual comparison between both methods. We do this by projecting a single sample of the irregular “jumping-jacks-50022” dataset onto the first 1 to 9 principal components of QPCA and mass-weighted QPCA. Figures 6.25 and 6.26 show the results of this experiment, where the projection onto the subspace (solid color) is shown as well as the original frame (transparent).

These figures are really able to capture the difference between both methods. The original QPCA method clearly prioritizes the accurate reconstruction of the left side of the mesh. Using only five components the left side of the projected sample already seems to align perfectly with the original frame. The right side of the mesh, however, is not even aligned correctly using nine components. The reason for this difference has to do with the fact that the left side of the sample meshes contain many more vertices. Therefore, QPCA is able to capture a larger amount of variance in the first few components by prioritizing the left side. The method can gain less by accurately reconstructing the right side of the sample meshes and, therefore, this has a lower priority.

Mass-weighted QPCA, however, multiplies each vertex by its mass before applying QPCA. As a result, vertices with a larger triangular area get a higher priority and vice versa. This means that QPCA will no longer prioritize the left side of the sample meshes. Figure 6.26 seems to be in agreement with this theory. When using five components, we see that both the left and the right side of the projected sample are not yet perfectly aligned. This indicates that the left side is no longer prioritized. Furthermore, when using nine components we are able to reconstruct the sample more accurately than the normal QPCA method is able to.

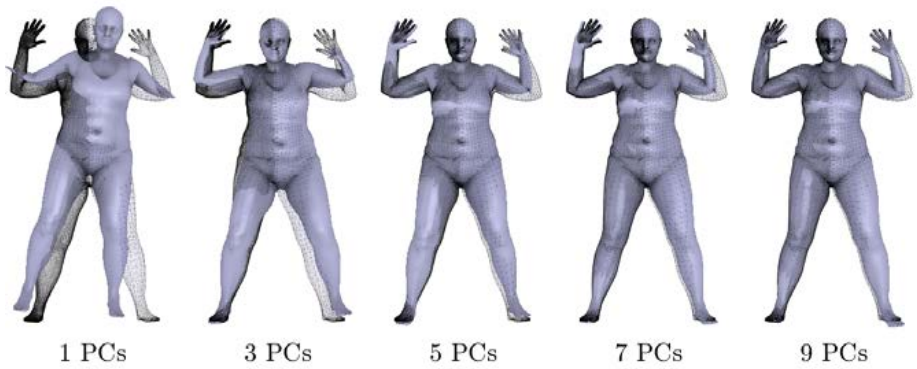


Figure 6.25: Visualization of QPCA: Projection of one sample of the “jumping-jacks-50022” motion capture from DFAUST [7] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

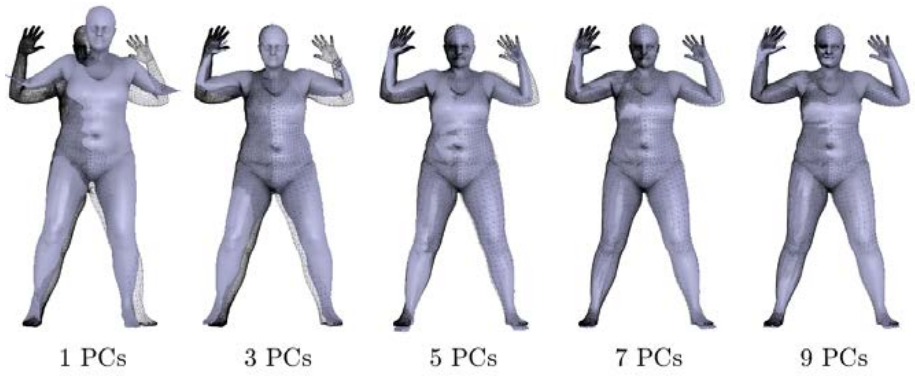


Figure 6.26: Visualization of mass-weighted QPCA: Projection of one sample of the “jumping-jacks-50022” motion capture from DFAUST [7] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

7

Results quaternion sparse PCA

The quaternion sparse PCA techniques described in Chapter 5 are designed to find components that are both sparse and localized. As a result, the obtained components require less storage space and should be more interpretable than the components found by normal PCA and QPCA. In this chapter, we evaluate both the real sparse PCA methods and their quaternion counterparts. Through a number of experiments, we aim to answer the questions posed in Chapter 1.

The chapter is organized as follows. First, we discuss the datasets that will be used in the experiments as well as the values of the tunable parameters. Next, we perform experiments on the SPLOCS and QSPLOCS methods in Section 7.2. This section includes both quantitative experiments, where the methods are compared on reconstruction accuracy, sparsity, convergence rates and computation times, and a number of visual experiments. The same experiments are then performed on SPLOCS_lap and QSPLOCS_lap in Section 7.3.

7.1. Datasets and parameter selection

All experiments in this chapter are performed on a variety of datasets. For consistency, we use the same datasets as described in Section 6.1. It will be especially interesting to evaluate the sparse PCA methods on the face dataset [66], which typically shows a lot of linear deformations, as well as on the TCD Hands dataset [20], in which only a small portion of the mesh moves around (arms/hands) while the remainder remains almost constant over the samples. For this reason, we perform all experiments in this chapter on the following datasets:

- “horse-gallop”, taken from the animal dataset [52].
- “face”, taken from [66].
- “Sign D poses”, taken from TCD Hands [20].

Experiments performed on other datasets can be found in Appendix B.

The sparse PCA methods all contain a fair amount of tunable parameters. Prior to performing the experiments, a number of preprocessing steps are applied as discussed in Section 5.6. That is, all datasets are rescaled and normalized by their standard deviation after subtracting the rest shape. Since all samples are rescaled to fit in a $[-0.5,\ 0.5]$ box, the d_{min} and d_{max} parameters are fixed at 0.2 and 0.6 respectively. Next, the rest shape is simply taken as the first sample of the datasets, as we found that the first sample often shows a neutral pose. The penalty parameter ρ is set to 10 on all experiments, as this appeared to give good results. Furthermore, the number of iterations for ADMM is set to 20, as convergence was often observed around this time, and the total optimization phase is iterated 5 times.

As discussed in Section 5.6, both the data term and the sparsity inducing term scale approximately linearly with the number of vertices. The Laplacian term, on the other hand, stays approximately constant. Therefore, we can use the same weights on all datasets and only have to adjust the weight of the

Laplacian term in order to correct for this. Through trial and error, tested on the “horse-gallop” dataset, the weights are set as follows:

- $\lambda_1 = \frac{n}{250}$ (Laplacian term).
- $\lambda_2 = 3$ (sparsity inducing term).

7.2. SPLOCS and QSPLOCS

Before investigating the effect of the added Laplacian term, we first evaluate the performance of the original SPLOCS method and compare it to the performance of the proposed QSPLOCS method. We do this by first analyzing the methods quantitatively, where we compare the reconstruction accuracy, sparsity and convergence rates of both methods. Here, we also give a comparison of the computation times. All these experiments are performed in the section below. Afterwards, we visually compare the methods in Section 7.2.2 in order to assess the interpretability of the components as well as how much the components depend on the initialization phase. An additional experiment is performed in this section, where the methods are evaluated on their ability to handle articulated rotations.

7.2.1. Quantitative evaluation

As a first experiment we evaluate the reconstruction accuracy on the selected datasets. For this we use the same error function as before, shown in Equation (6.1). We plot the average over 2 runs, where 80% of the samples are used for training and the remaining samples are used to evaluate the performance on unseen test data. Since we are interested in obtaining components that are sparse, it makes sense to additionally plot the sparsity error of the components. The error function we use is:

$$E_{sparsity} = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{d} m_{ii} \cdot \|\mathbf{u}_{ij}\|_2}{d \cdot \|\mathbf{1}\|_M^2}} \tag{7.1}$$

where $\mathbf{M}$ denotes the mass matrix, $\mathbf{U}$ contains the real components and d denotes the number of components. Note that the sparsity error for QSPLOCS can be written in a slightly different way due to its simplified ℓ_1 norm:

$$E_{sparsity} = \sqrt{\frac{\sum_{i=1}^{n}\sum_{j=1}^{d} m_{ii} \cdot |\mathbf{u}_{ij}|}{d \cdot \|\mathbf{1}\|_M^2}} \tag{7.2}$$

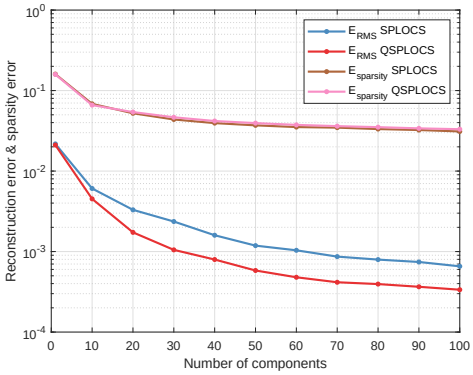
Results of this experiment on the three datasets can be found in Figures 7.1-7.3. Results obtained on other datasets can be found in Appendix B.1.1. From the results it becomes clear that QSPLOCS is able to achieve higher reconstruction accuracies using less components than SPLOCS. This higher reconstruction accuracy is observed on all three types of datasets, containing either linear motions (“face”) or more non-linear rotational motions (“horse-gallop” and “Sign D poses”). If we look at Figure 7.3, we notice that the E_{RMS} values fluctuate a lot depending on the number of components used. This is especially noticeable for QSPLOCS. One explanation for this could be the fact that the ADMM method, which is used in both methods, is a first order method and can end up in a non-optimal local minimum. Since the “Sign D poses” dataset only contains motions in the arm and hand regions, while the rest of the body remains almost constant, the methods will try to center almost all components around these arm and hand regions. This makes it harder to find a global minimum.

Figures 7.1-7.3 also show the sparsity errors of the three datasets. From the plots it becomes clear that the average sparsity error goes down when more components are added. This is as expected, since a trade-off is made between reconstruction accuracy and sparsity. When only a few components are computed, the methods can gain more in terms of a better data approximation and therefore accept the higher sparsity penalty. However, when more components are added you reach a point where the sparsity term is higher than what can be gained in terms of a better data approximation. This results in components that are more sparse.

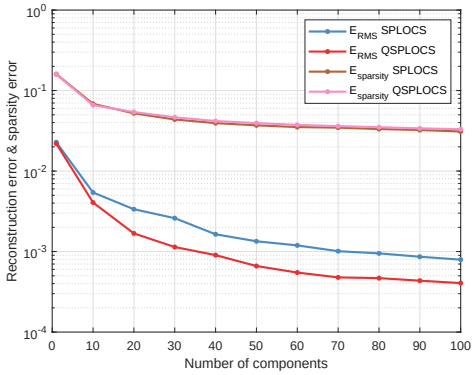
On the “face” and “Sign D poses” datasets QSPLOCS is shown to have a lower sparsity error. This means that QSPLOCS is able to give components that are more sparse than SPLOCS, while also achieving a higher reconstruction accuracy. A slightly higher sparsity error is observed for QSPLOCS on the “horse-gallop” dataset in Figure 7.1. However, the difference in terms of reconstruction accuracy

between SPLOCS and QSPLOCS is also higher for this dataset. Thus, this again visualizes the trade-off that is made between reconstruction accuracy and sparsity.

As before, we can also plot the reconstruction and sparsity errors against the amount of memory needed to store the components. This is shown in Figures 7.4-7.6. From these figures we can conclude that QSPLOCS is still able to describe a richer space of deformations, even when storage capacity is limited.

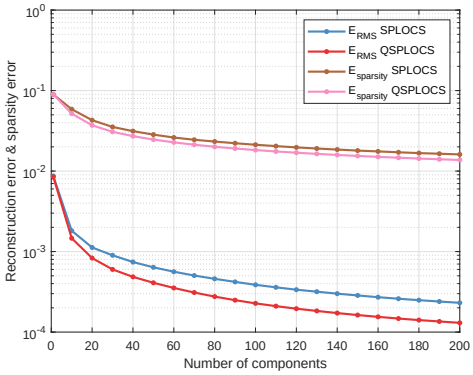


(a) E_{RMS} and $E_{sparsity}$ computed on 40 training frames.

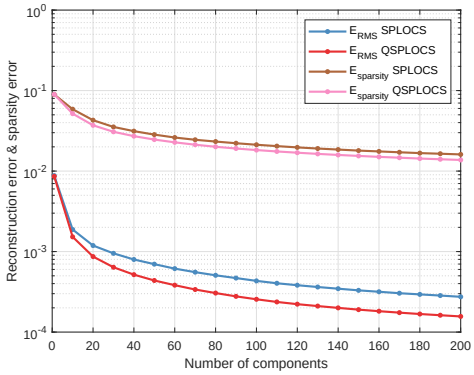


(b) E_{RMS} and $E_{sparsity}$ computed on 9 (unseen) test frames.

Figure 7.1: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "horse-gallop" motion capture from the animal dataset [52].

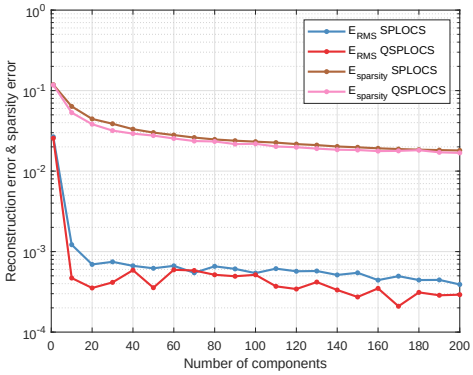


(a) E_{RMS} and $E_{sparsity}$ computed on 308 training frames.

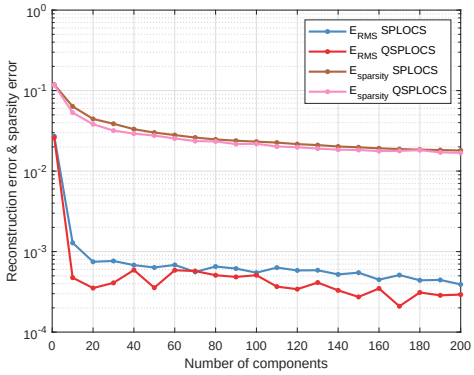


(b) E_{RMS} and $E_{sparsity}$ computed on 77 (unseen) test frames.

Figure 7.2: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "face" motion capture from [66].

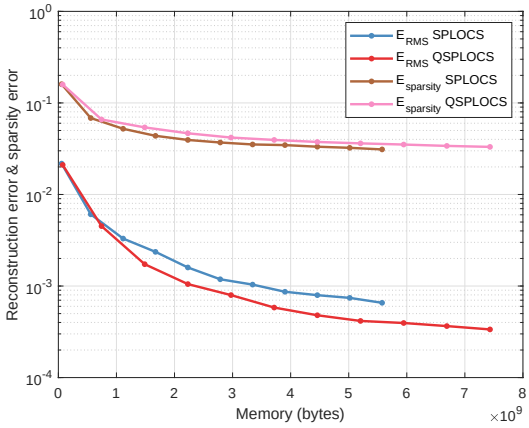


(a) E_{RMS} and $E_{sparsity}$ computed on 481 training frames.

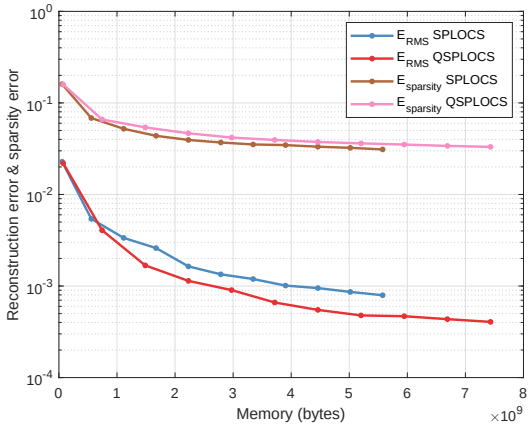


(b) E_{RMS} and $E_{sparsity}$ computed on 120 (unseen) test frames.

Figure 7.3: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "Sign D poses" motion capture from the TCD Hands dataset [20].

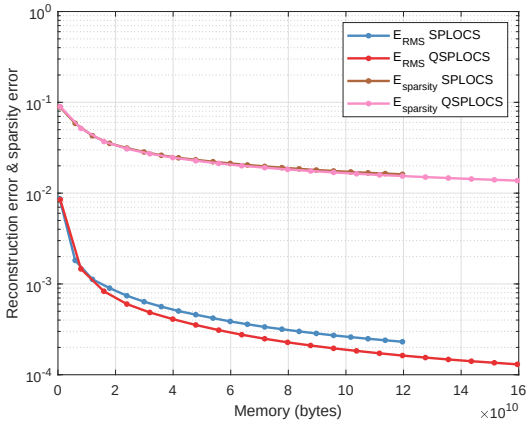


(a) E_{RMS} and $E_{sparsity}$ computed on 40 training frames.

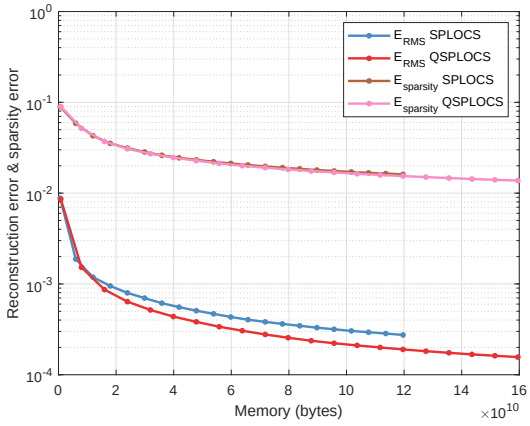


(b) E_{RMS} and $E_{sparsity}$ computed on 9 (unseen) test frames.

Figure 7.4: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "horse-gallop" motion capture from the animal dataset [52].

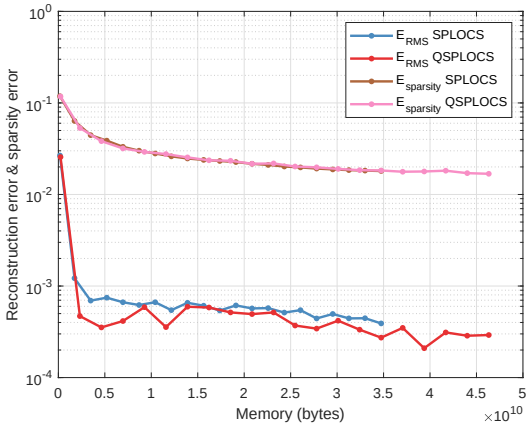


(a) E_{RMS} and $E_{sparsity}$ computed on 308 training frames.

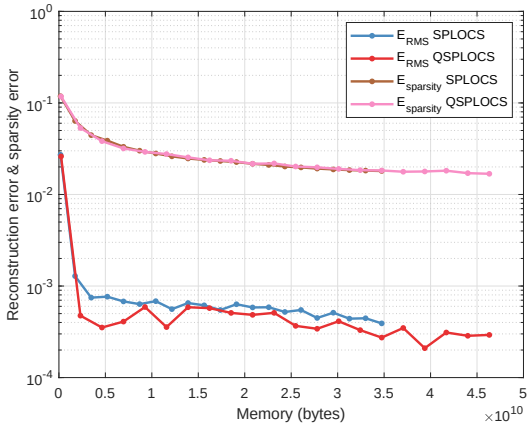


(b) E_{RMS} and $E_{sparsity}$ computed on 77 (unseen) test frames.

Figure 7.5: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "face" motion capture from [66].



(a) E_{RMS} and $E_{sparsity}$ computed on 481 training frames.

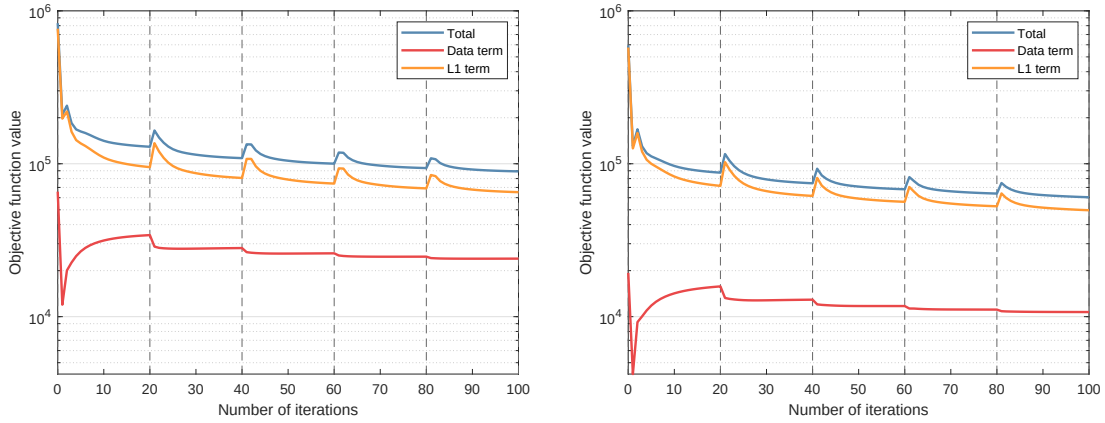


(b) E_{RMS} and $E_{sparsity}$ computed on 120 (unseen) test frames.

Figure 7.6: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "Sign D poses" motion capture from the TCD Hands dataset [20].

Besides looking at the reconstruction accuracy, it is also interesting to compare the convergence rates of both methods. Since we observed similar convergence behavior on all datasets considered, we will only discuss the convergence rates on the “face” dataset, which is shown in Figure 7.7.

For constructing the convergence plots, we applied SPLOCS and QSPLOCS to the dataset using $d = 100$ components. Thereafter, we plotted the value of each term in the objective function after each iteration. The peaks that are visible after every 20 iterations are due to the weight optimization step. Both SPLOCS and QSPLOCS show similar convergence behavior. The data term and sparsity inducing term are both shown to decrease at every iteration (apart from the weight update step). The QSPLOCS method is able to achieve a lower objective function value, both on the data term and the sparsity inducing term. This indicates that the quaternion method is able to obtain components that give a higher reconstruction accuracy, while being more sparse.



(a) Convergence of SPLOCS.

(b) Convergence of QSPLOCS.

Figure 7.7: Objective function value (y-axis) with respect to the number of iterations (x-axis) on the “face” motion capture from [66].

Finally, it is also interesting to compare the methods based on their computation time. Since quaternions can be rewritten as a 4×4 real matrix, multiplication of two quaternions takes 16 real multiplications and 12 additions. Thus, we expect the computation times of QSPLOCS to be considerable higher than those of SPLOCS. Table 7.1 shows the computation times on a variety of datasets. The average time is taken over two runs.

From the table it becomes clear that SPLOCS is indeed faster than QSPLOCS on all datasets considered. The computation times seem to be dependent on both the number of vertices and the number of frames contained in the dataset. On the “horse-gallop” dataset, SPLOCS is shown to be on average 1.7 times faster than QSPLOCS. On the “C5 - Walk to run” dataset, which contains more frames but less vertices, SPLOCS is 3.4 times as fast. Thus, the frame count appears to have a large effect on the computation times of QSPLOCS. Even though the methods show a large difference in computation times, this is not a huge issue. Since subspaces are typically computed in an offline phase, 23 minutes computation time, which is the largest value shown in the table, is still acceptable.

Table 7.1: Comparison of computation times when using SPLOCS and QSPLOCS. The table shows the number of vertices n, the number of frames f and the computation times using a varying number of components d on a variety of datasets. i7 2.50GHz, 8GB RAM, Matlab implementation.

Mesh	n	f	Times for $d = 10$ comp.		Times for $d = 50$ comp.		Times for $d = 100$ comp.	
			SPLOCS	QSPLOCS	SPLOCS	QSPLOCS	SPLOCS	QSPLOCS
“horse-gallop” [52]	8431	49	12.07s	17.63s	61.28s	98.61s	119.96s	199.22s
“face” [66]	23725	385	59.54s	146.79s	287.24s	708.43s	574.55s	1415.40s
“Sign D poses” [20]	6890	601	18.07s	53.59s	86.36s	271.95s	173.02s	541.37s
“C5 - Walk to run” [54]	6890	650	17.99s	56.36s	90.35s	285.24s	177.25s	572.15s
“E5 - Hook left poses” [54]	6890	327	13.07s	32.40s	64.51s	168.06s	128.01s	334.89s
“jumping-jacks-50022” [7]	6890	308	13.25s	33.12s	62.85s	161.85s	124.93s	322.51s
“jiggle-on-toes-50004” [7]	6890	239	11.84s	26.57s	58.64s	140.12s	116.56s	278.73s

7.2.2. Visual comparison

Similar to the experiments performed in Sections 6.2.2 and 6.3.2, we can also make a visual comparison between the two sparse PCA techniques. This enables us to better evaluate the sparsity and locality of both methods. We perform these experiments on all three datasets described at the beginning of this chapter. For these experiments, the methods were applied to obtain $d = 50$ sparse components. Visualizations of other datasets can be found in Appendix B.1.2

The SPLOCS and QSPLOCS methods aim to find components that approximate the training samples as good as possible while being sparse and localized. Thus, as opposed to the components found by PCA and QPCA, which are ordered based on their eigenvalues, the sparse PCA components do not have a clear ordering. This means that the 1st component is not necessarily more important than the 20th component. In addition, this means that the components found by SPLOCS do not necessarily describe the same deformations as the components found by QSPLOCS.

As a first experiment, we visualize and compare a few components of both SPLOCS and QSPLOCS. We do this by selecting 4 SPLOCS components at random and finding the QSPLOCS components that show a similar motion. Note that not all real components have a quaternion component showing the same motion and vice versa. Since the components do not have an eigenvalue that can be used as weight for the visualization, we select the weight of each component manually.

The components of SPLOCS for the “horse-gallop” dataset are shown in Figure 7.8. The corresponding quaternion components can be found in Figures 7.9-7.12. Both SPLOCS and QSPLOCS are able to produce components that are sparse and localized. However, the amount of sparsity present in the components can vary between the two methods. For example, the deformation shown in Figure 7.10 is slightly more sparse than its real counterpart. One reason for this could be that each quaternion component has 4 dimensions, while the real components only have 3. This means that the sparsity error of a quaternion component would be slightly higher than that of its real counterpart on components that cover the same area of a mesh. Consequently, the area covered by the quaternion component could decrease during the optimization phase.

Similar to what we observed for PCA and QPCA, the deformations described by the SPLOCS components are also contained in the QSPLOCS subspace. When comparing Figure 7.8 to Figure 7.9, we notice that the first SPLOCS component shows the same deformation as the QSPLOCS component along the fourth dimension ($\boldsymbol{k}$). In addition, the quaternion component describes three other deformations. This property also applies to the other visualized components.

Visualizations of SPLOCS on the other two datasets are found in Figures 7.13 and 7.18 and the corresponding QSPLOCS components can be found in Figures 7.14-7.17 and Figures 7.19-7.22 respectively. Similar properties as described above can be observed in these datasets. Interestingly enough, the deformation shown in the SPLOCS component is not always described by the same dimension of the QSPLOCS component. On the “face” dataset, for instance, the SPLOCS components show approximately the same deformations as the QSPLOCS components along the second dimension ($\boldsymbol{i}$). On the “Sign D poses” dataset this is mostly shown along the fourth dimension ($\boldsymbol{k}$) of the QSPLOCS components, except for the component in Figure 7.22, which describes the real motion along its first dimension. It should be noted that this does not hold for all component of both methods. As mentioned above, some deformations shown in the SPLOCS components are not present in the set of QSPLOCS components and vice versa. An example of this is shown in Figure 7.16, in which the QSPLOCS component shows quite different deformations than the 12th SPLOCS component from Figure 7.13.

One apparent property of the two methods is that the components do not always look smooth or natural. Two examples of this are the 10th SPLOCS component of the “horse-gallop” dataset, shown in Figure 7.8, and the 4th QSPLOCS component of the “Sign D poses” dataset, shown in Figure 7.20. The deformations described by these components show a very sharp boundary at the neck region. As a result, the motions do not feel very natural. Another example is shown in Figure 7.19, where the deformation along the first dimension shows a wide forearm while the elbow region does not change in size. The reason for these unnatural sharp boundaries is the spatially varying regularization strengths used in both methods. Using these regularization strengths, only vertices that lie far away from the center of the component are penalized by the sparsity inducing term. Other vertices are less penalized by this term and some vertices are not penalized at all. This can result in components with a sharp boundary, since the center vertices are able to freely describe any large motion, while the sparsity

penalty at the boundaries might be too large to describe any small motion.

Thus, in conclusion, we can say that the SPLOCS and QSPLOCS methods are both able to obtain components that are sparse and localized. Oftentimes, the deformation described by the SPLOCS component is also present in the space spanned by its quaternion variant. The area covered by both components can, however, vary in size. Due to the spatially varying regularization parameters, it can happen that the deformations described by the components show sharp boundary edges. Thus, the components are not always smooth or natural looking.

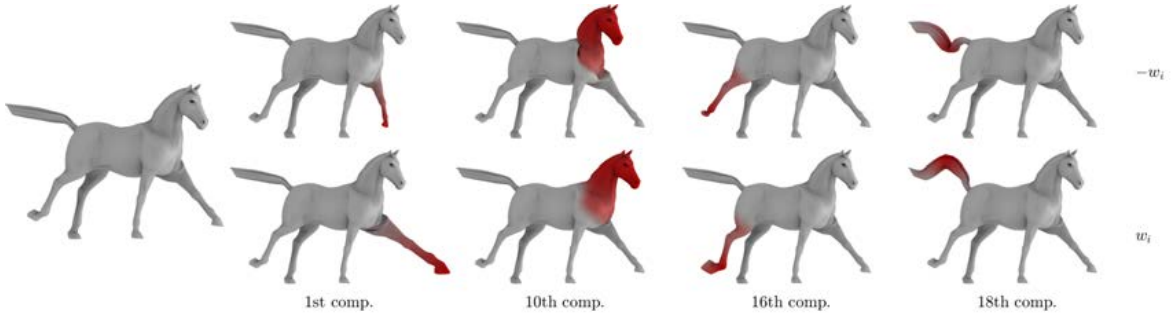


Figure 7.8: Visualization of SPLOCS: 4 sparse components on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

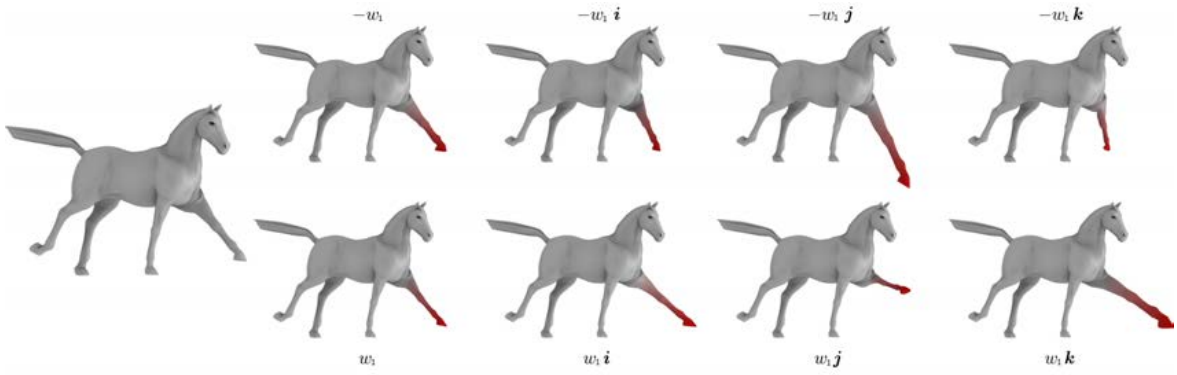


Figure 7.9: Visualization of QSPLOCS: The 1st quaternion sparse component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_1$ (top) and w_1 (bottom) along the four dimensions of the 1st component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

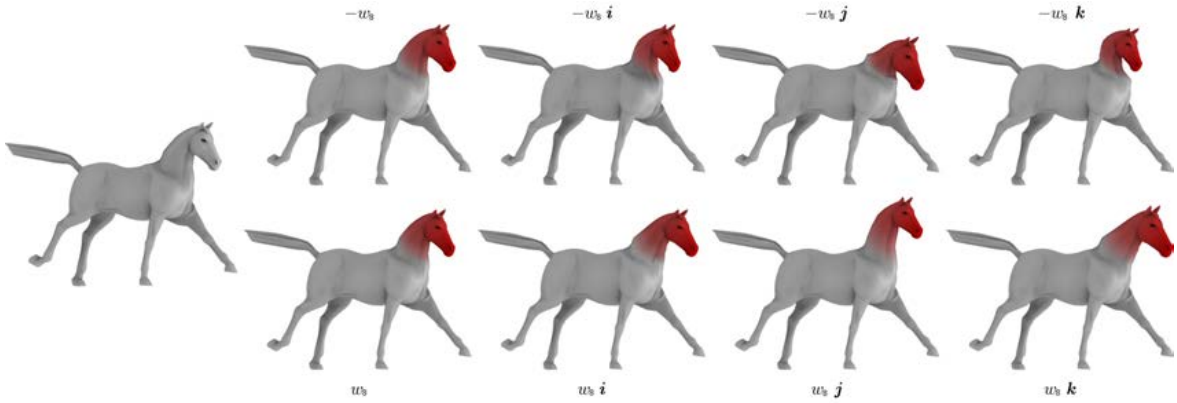


Figure 7.10: Visualization of QSPLOCS: The 8th quaternion sparse component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_8$ (top) and w_8 (bottom) along the four dimensions of the 8th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

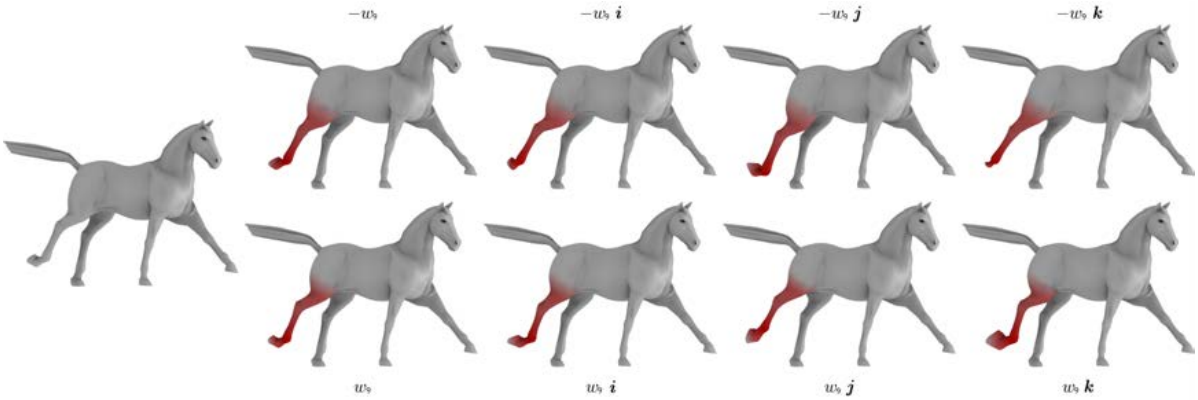


Figure 7.11: Visualization of QSPLOCS: The 9th quaternion sparse component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_9$ (top) and w_9 (bottom) along the four dimensions of the 9th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

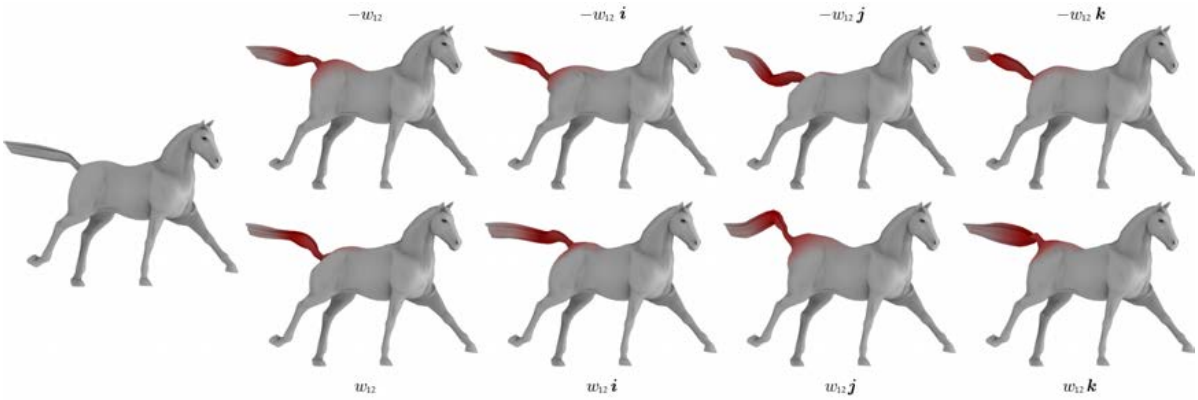


Figure 7.12: Visualization of QSPLOCS: The 12th quaternion sparse component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_{12}$ (top) and w_{12} (bottom) along the four dimensions of the 12th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

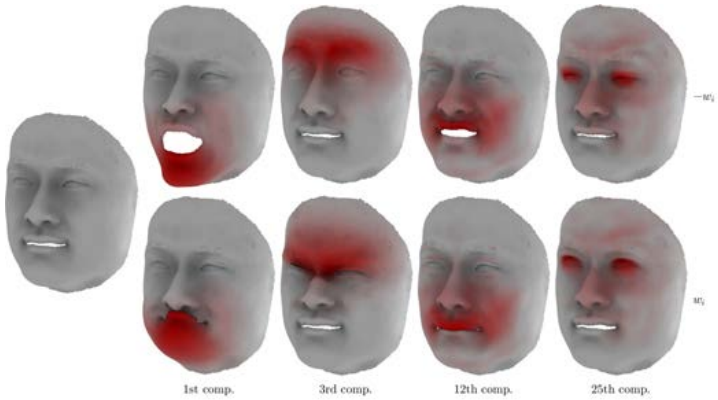


Figure 7.13: Visualization of SPLOCS: 4 sparse components on the “face” motion capture from [66]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

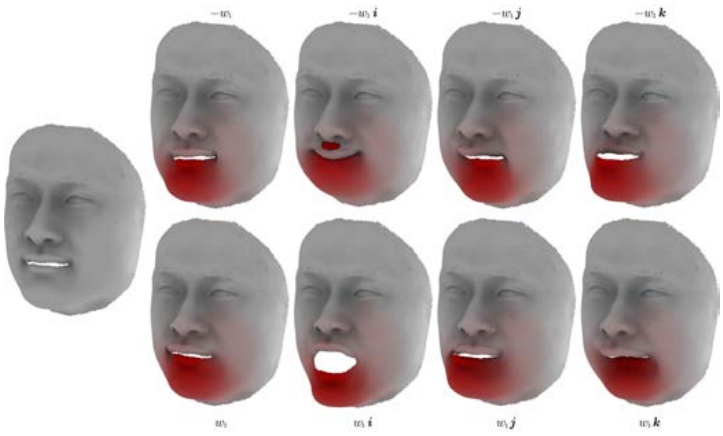


Figure 7.14: Visualization of QSPLOCS: The 1st quaternion sparse component on the "'face"' motion capture from [66]. Models corresponding to $-w_1$ (top) and w_1 (bottom) along the four dimensions of the 1st component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

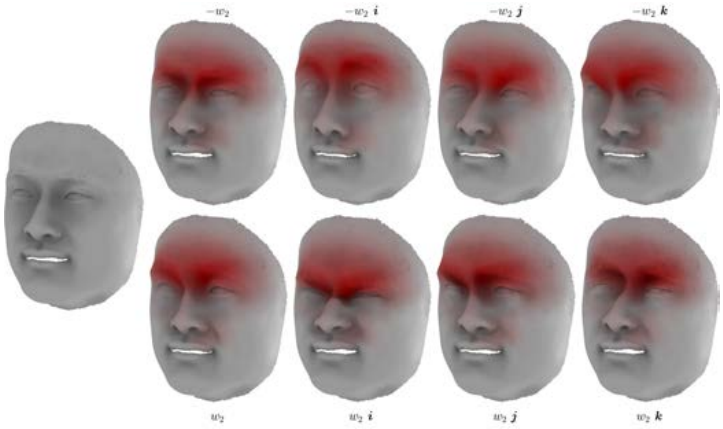


Figure 7.15: Visualization of QSPLOCS: The 2nd quaternion sparse component on the "'face"' motion capture from [66]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

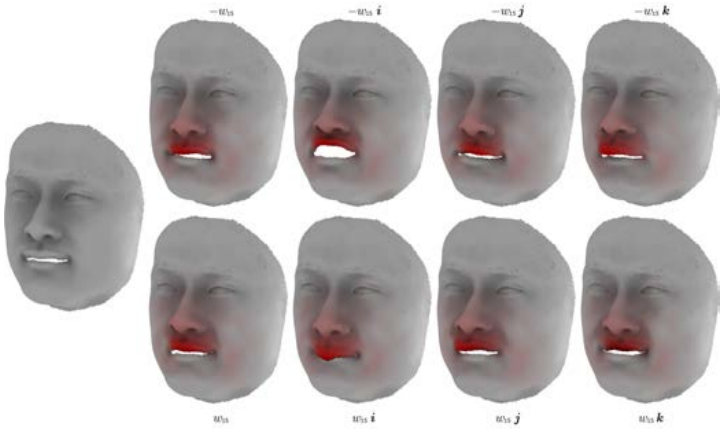


Figure 7.16: Visualization of QSPLOCS: The 15th quaternion sparse component on the "'face"' motion capture from [66]. Models corresponding to $-w_{15}$ (top) and w_{15} (bottom) along the four dimensions of the 15th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

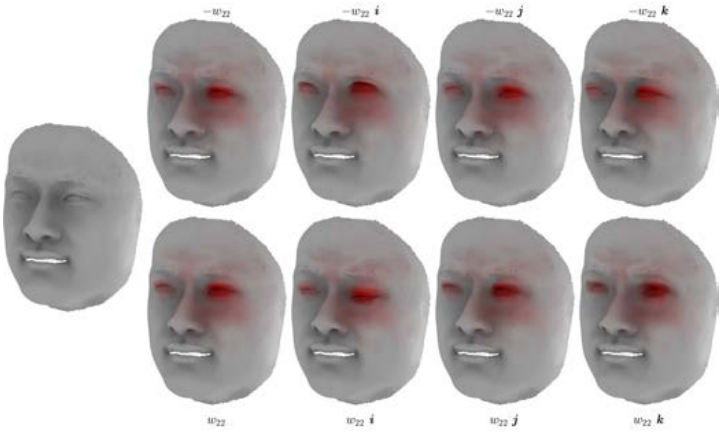


Figure 7.17: Visualization of QSPLOCS: The 22nd quaternion sparse component on the "'face"' motion capture from [66]. Models corresponding to $-w_{22}$ (top) and w_{22} (bottom) along the four dimensions of the 22nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

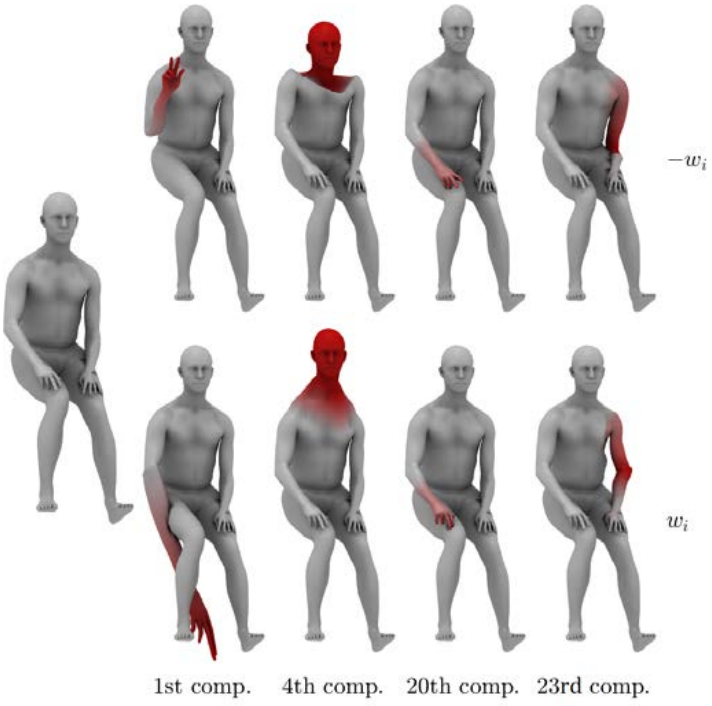


Figure 7.18: Visualization of SPLOCS: 4 sparse components on the "Sign D poses" motion capture from TCD Hands[20]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

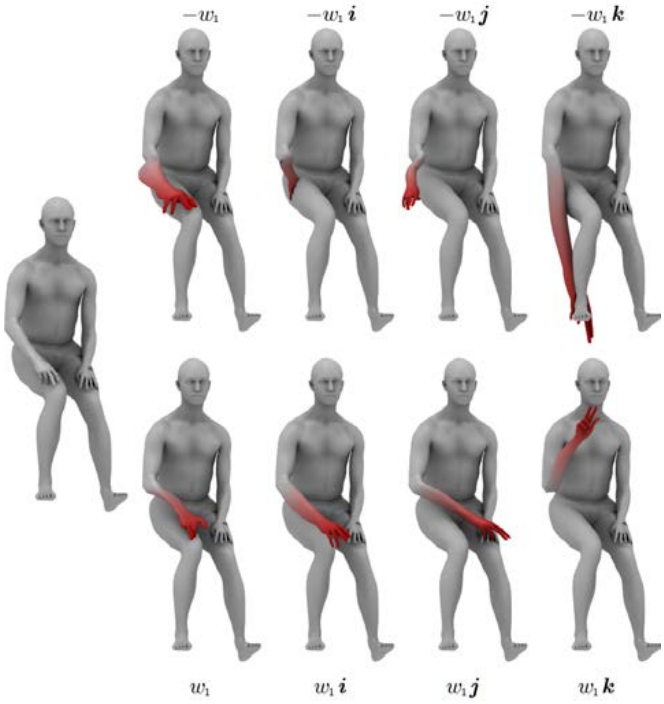


Figure 7.19: Visualization of QSPLOCS: The 1st quaternion sparse component on the "'Sign D poses"' motion capture from TCD Hands [20]. Models corresponding to $-w_1$ (top) and w_1 (bottom) along the four dimensions of the 1st component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

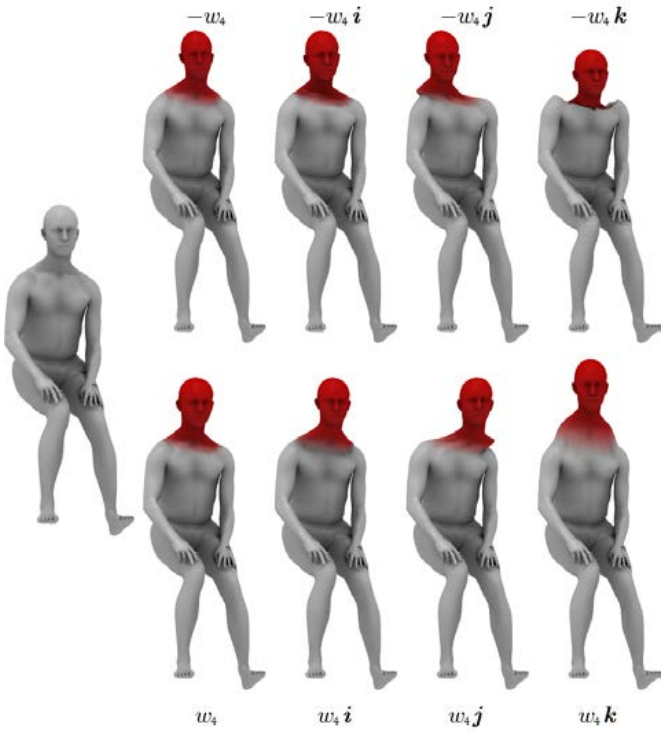


Figure 7.20: Visualization of QSPLOCS: The 4th quaternion sparse component on the "'Sign D poses"' motion capture from TCD Hands [20]. Models corresponding to $-w_4$ (top) and w_4 (bottom) along the four dimensions of the 4th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

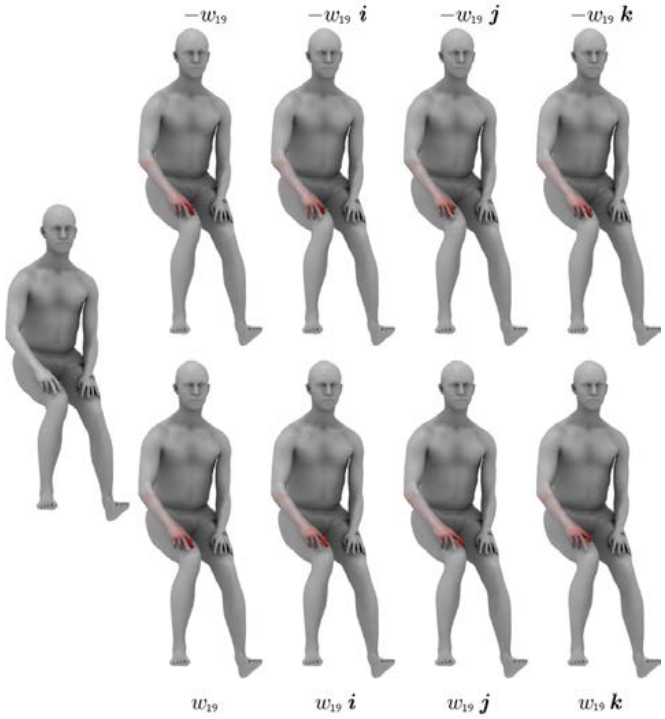


Figure 7.21: Visualization of QSPLOCS: The 19th quaternion sparse component on the "'Sign D poses"' motion capture from TCD Hands [20]. Models corresponding to $-w_{19}$ (top) and w_{19} (bottom) along the four dimensions of the 19th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

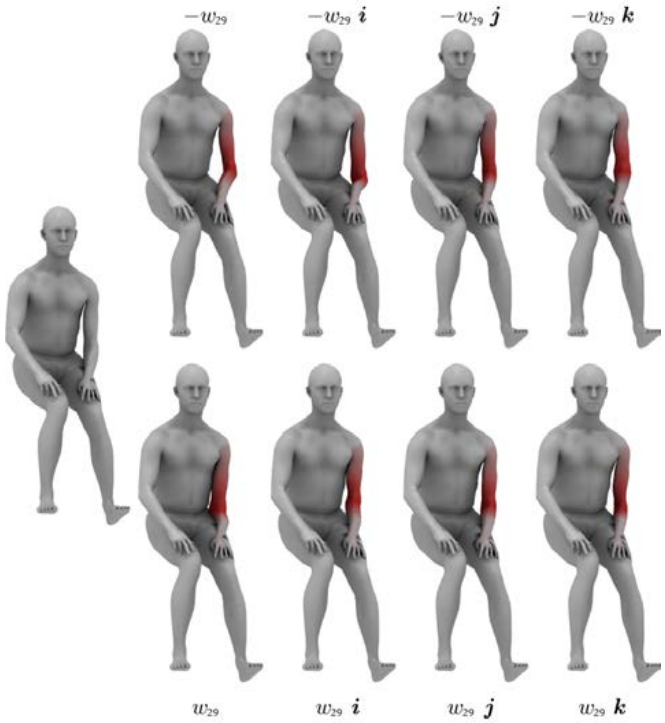


Figure 7.22: Visualization of QSPLOCS: The 29th quaternion sparse component on the "'Sign D poses"' motion capture from TCD Hands [20]. Models corresponding to $-w_{29}$ (top) and w_{29} (bottom) along the four dimensions of the 29th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

As a second visual experiment, we want to investigate the influence of the initialization phase on the components that are obtained. Since we found that the ADMM method used in the optimization phase does not progress that much from its initial values, we expect the method to be very dependent on the initialization phase. In this experiment, we applied the methods to various datasets using a random initialization. For SPLOCS this means that the entries in each component got assigned a random value between $[-1, 1]$. For QSPLOCS, each entry was filled with a uniformly distributed unit quaternion. Results of this are shown in Figures 7.23 and 7.26 for SPLOCS and in Figures 7.24, 7.25, 7.27 and 7.28 for QSPLOCS. Since other datasets showed similar results, we will limit our discussion to these two datasets only.

From the figures we can immediately tell that the results obtained using random initialization are far from optimal. Especially on the "horse-gallop" dataset we notice many artifacts. Many of the components, such as the 4th SPLOCS component in Figure 7.23 and the 1st QSPLOCS component in Figure 7.24, show a lot of noise due to the random initialization. This is far from ideal and indicates that ADMM is indeed not able to reach a good solution. However, not all components show this noise. Some components do seem nicely sparse and localized, such as the 19th SPLOCS component from the "horse-gallop" dataset. Other components, however, are less localized or lack interpretable meaning, such as the 13th SPLOCS component from Figure 7.26.

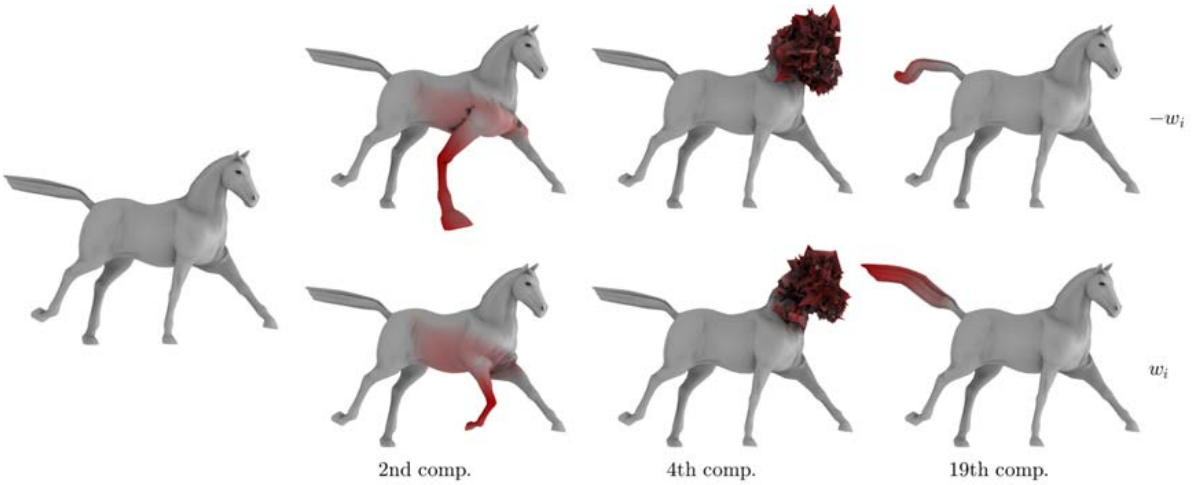


Figure 7.23: Visualization of SPLOCS using random initialization: 3 sparse components on the "horse-gallop" motion capture from the animal dataset [52]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

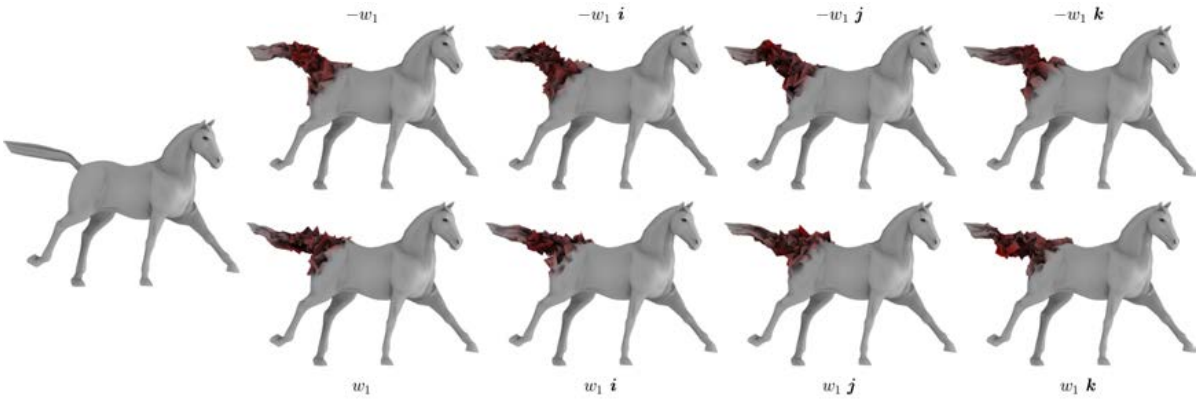


Figure 7.24: Visualization of QSPLOCS using random initialization: The 1st quaternion sparse component on the "horse-gallop" motion capture from the animal dataset [52]. Models corresponding to $-w_1$ (top) and w_1 (bottom) along the four dimensions of the 1st component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

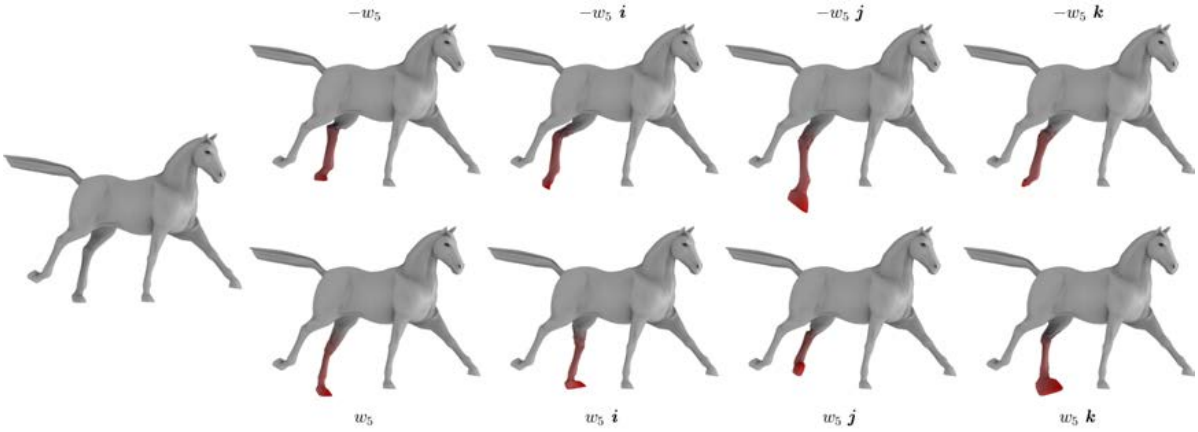


Figure 7.25: Visualization of QSPLOCS using random initialization: The 5th quaternion sparse component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_5$ (top) and w_5 (bottom) along the four dimensions of the 5th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

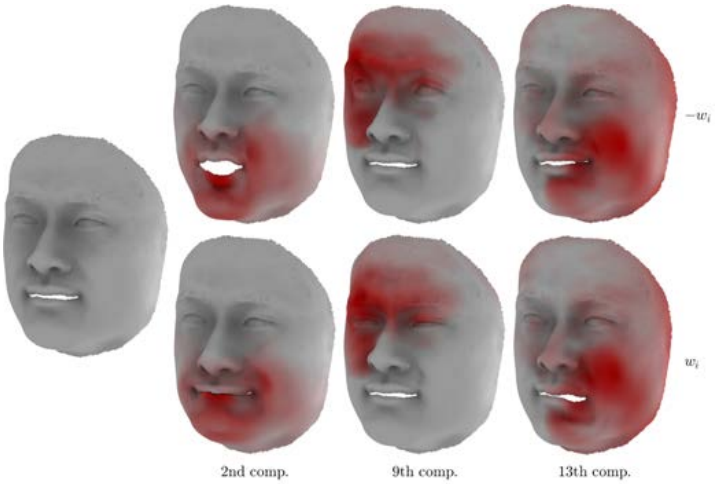


Figure 7.26: Visualization of SPLOCS using random initialization: 3 sparse components on the “face” motion capture from [66]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

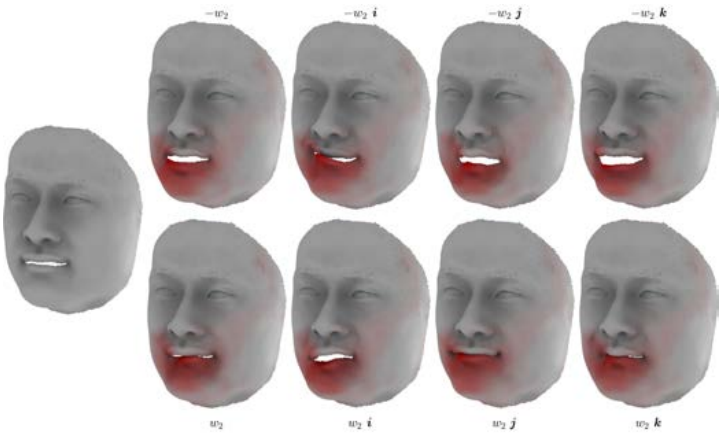


Figure 7.27: Visualization of QSPLOCS using random initialization: The 2nd quaternion sparse component on the “‘face”’ motion capture from [66]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

Figure 7.28: Visualization of QSPLOCS using random initialization: The 17th quaternion sparse component on the "'face"' motion capture from [66]. Models corresponding to $-w_{17}$ (top) and w_{17} (bottom) along the four dimensions of the 17th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

One known limitation of SPLOCS is that the method cannot handle articulated rotations well [39]. Since the components are linear, the method is not able to describe the curvilinear paths present in the datasets, such as the rotating limbs in the "horse-gallop" dataset. Since each quaternion QSPLOCS component can be seen as a linear combination of four real deformations, it is interesting to investigate whether the QSPLOCS method is able to better describe these rotations. We do this by visualizing some components using a few different weights. These weights are selected as follows. For SPLOCS, we simply take the largest and smallest weights that were observed in the training data as well as some values in between. QSPLOCS is, however, harder to visualize, since many different quaternion-valued weights are possible. We do this by taking the weights from the training data that have the largest, median and smallest value along each of the four dimensions. This gives us a maximum of 12 projected samples to visualize, from which we remove samples that have a similar weight.

We perform this experiment on the "horse-gallop" dataset only, as this dataset contains a lot of articulated rotations. Figures 7.29 and 7.30 show the results on two SPLOCS components and Figures 7.31 and 7.32 show the corresponding QSPLOCS components. As expected, we see that the SPLOCS components indeed follow a linear path. This introduces some unwanted artifacts, where the components shows a shrinkage of the leg or tail along the middle of the path. The QSPLOCS components, on the other hand, seem to be able to describe a somewhat curvilinear path. The deformations do not follow a linear path. However, since many different quaternion weights are possible for QSPLOCS, it is not easy to describe this path. The figures show two approximate lines that seem to follow the motion of the selected samples. The deformations, however, do not necessarily move the mesh in a single direction. From the back-view, we can see that the components also move the mesh from left to right. Thus, it is very hard to properly describe these motions. The QSPLOCS method also introduces some new artifacts, where some configurations again show a shrunken or skewed version of the leg or tail.

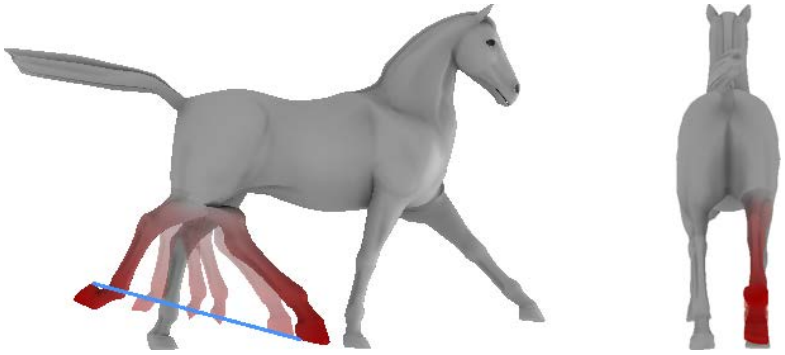


Figure 7.29: Visualization of SPLOCS on the "horse-gallop" dataset [52] showing its limitation regarding articulated rotations. Models corresponding to the largest and smallest weights present in the training data are displayed as well as some values in between. The experiment is performed on the 2nd component.

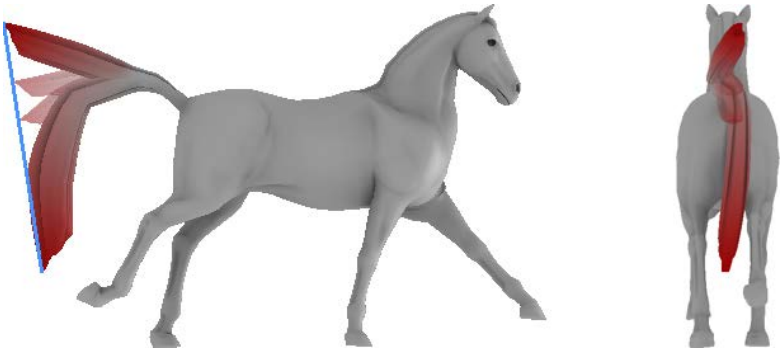


Figure 7.30: Visualization of SPLOCS on the “horse-gallop” dataset [52] showing its limitation regarding articulated rotations. Models corresponding to the largest and smallest weights present in the training data are displayed as well as some values in between. The experiment is performed on the 5th component.

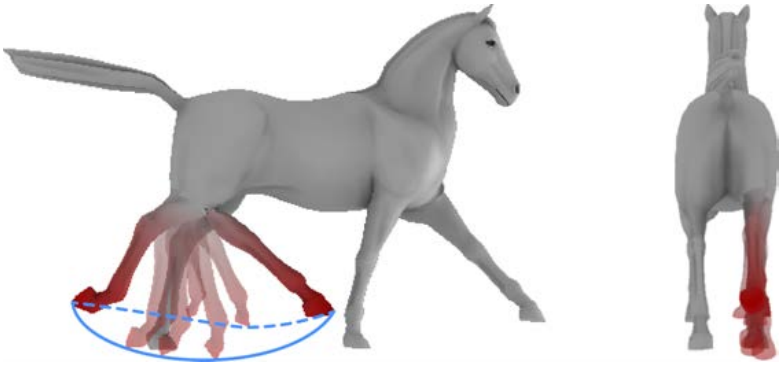


Figure 7.31: Visualization of QSPLOCS on the “horse-gallop” dataset [52] showing how it handles articulated rotations. Models corresponding to the largest, median and smallest weights along each dimension present in the training data are displayed. The experiment is performed on the 2nd component.

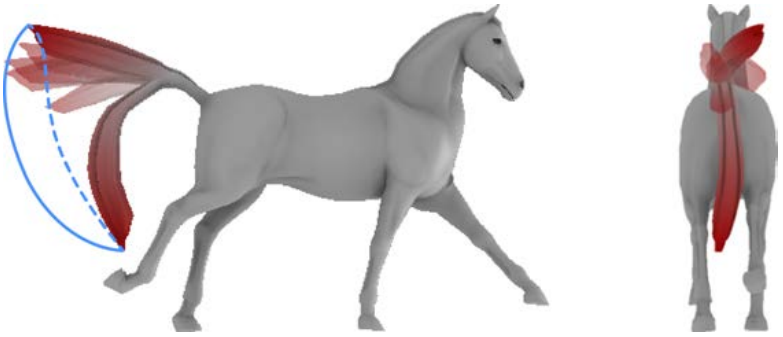


Figure 7.32: Visualization of QSPLOCS on the “horse-gallop” dataset [52] showing how it handles articulated rotations. Models corresponding to the largest, median and smallest weights along each dimension present in the training data are displayed. The experiment is performed on the 5th component.

7.3. SPLOCS_lap and QSPLOCS_lap

After having evaluated the original SPLOCS method and its quaternion variant QSPLOCS, it is interesting to evaluate the effect of the added Laplacian term. As mentioned in Section 5.3, one of the properties of SPLOCS is that it uses varying regularization strengths based on the d_{min} and d_{max} distances set by the user. This means that during optimization, only vertices that lie far away from the center of the components are penalized by the sparsity term. We found that this results in components that are not always very smooth or natural looking, as shown in the previous section. Furthermore, this results in components that are heavily influenced by the initialization phase of the method.

By getting rid of these spatially varying regularization strengths and instead introducing an additional term based on the Laplacian, which penalizes large differences in values between neighboring vertices, the goal was to obtain naturally smooth, sparse and localized components without having to explicitly specify the size and location of the components. In this section we perform various experiments to investigate this. First, we perform a quantitative evaluation in the section below, where we compare SPLOCS_lap and QSPLOCS_lap based on their reconstruction accuracy, sparsity, convergence rates and computation times. Next, we perform visual analysis in Section 7.3.2, where we evaluate the sparsity, locality and smoothness of the components. Furthermore, we perform experiments to investigate to what extent the methods are dependent on the initialization phase and whether the methods are able to handle articulated rotations.

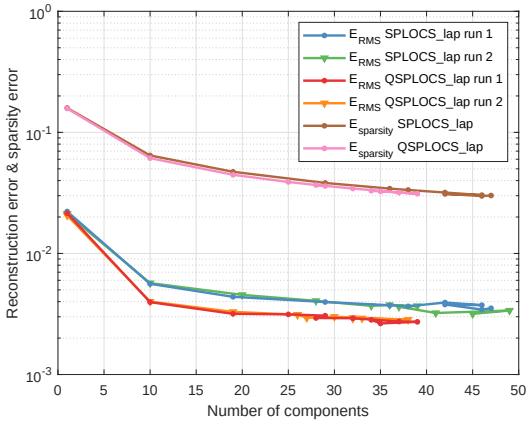
7.3.1. Quantitative evaluation

As before, we evaluate the reconstruction accuracy and sparsity error of the SPLOCS_lap and QSPLOCS_lap methods on the three datasets described at the start of this chapter. We apply both methods two times using a varying number of components, where 80% of the data is used for training. Results of this are shown in Figures 7.33-7.35. In order to avoid clutter, we only plot the sparsity error of the first run. Results obtained on other datasets can be found in Appendix B.2.1

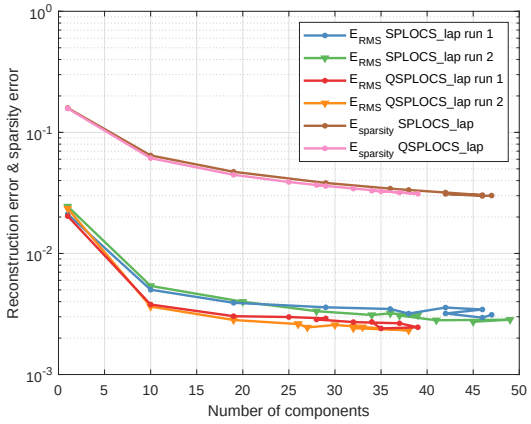
Immediately, we can notice a lot of differences when comparing these plots to those obtained by SPLOCS and QSPLOCS in Figures 7.1-7.3. First of all, we have plotted the reconstruction error of each run separately instead of taking the average. This has to do with the way in which SPLOCS_lap and QSPLOCS_lap are designed. As mentioned above, the original SPLOCS method uses spatially varying regularization parameters that do not penalize vertices that lie close to the center of the components. Thus, it is possible to improve the data approximation in that area without having any penalty. For SPLOCS_lap and QSPLOCS_lap, on the other hand, we do not have this term based on the distance to the center. The sparsity inducing term penalizes each non-zero valued vertex in the components. As a result, it can happen that the value of the sparsity inducing term is larger than what can be gained in terms of a better data approximation. In those cases, the entire component will go to zero. Since each run uses different training samples, and because ADMM is prone to end up in non-optimal local minima, the number of non-zero components can vary per run. Therefore, we are not able to take the average over all runs.

A second consequence of this is that there is a maximum number of components that can be obtained by the methods. For the “horse-gallop” dataset, shown in Figure 7.33, we are not able to obtain more than ± 50 real components or ± 40 quaternion components. All remaining components are zero vectors due to the trade-off between reconstruction accuracy and sparsity. The same applies to the “Sign D poses” dataset, shown in Figure 7.35, where only ± 55 real and ± 45 quaternion non-zero components can be found. On the other hand, the “face” dataset from Figure 7.34 is able to get the full amount of non-zero components. The reason why some datasets are able to get many components, while other datasets are not, has to do with the motions present in the dataset. As mentioned before, the face dataset shows many linear deformations, where the entire face is in motion. In this case, the improved performance in terms of data approximation always outweighs the penalty introduced by the sparsity and Laplacian terms. Therefore, the components will not go to zero everywhere. The “Sign D poses” dataset, on the other hand, shows very detailed motions in the hand and arm regions, while the remainder of the body remains almost constant. As a result, components will mostly be found in these significant areas, since the sparsity inducing term will be too high in other regions. More components can be found by lowering the weight of the sparsity inducing term, however the resulting components will then also be less sparse. A trade-off needs to be made.

In comparison to SPLOCS_lap, QSPLOCS_lap shows improved performances both in terms of reconstruction accuracy and sparsity on all three datasets. This indicates that the quaternion variant is again able to describe a richer space of deformations. Figures 7.36-7.38 show the reconstruction and sparsity errors plotted against the memory needed to store the components. One interesting observation from these figures is that both methods are able to get approximately the same amount of non-zero components in terms of memory needed to store them. This does not hold for the “face” dataset for reasons discussed above.

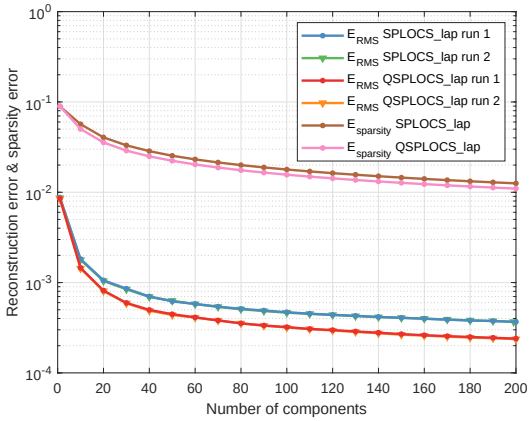


(a) E_{RMS} and $E_{sparsity}$ computed on 40 training frames.

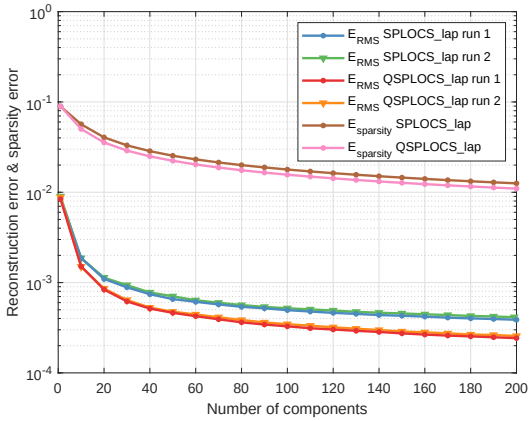


(b) E_{RMS} and $E_{sparsity}$ computed on 9 (unseen) test frames.

Figure 7.33: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "horse-gallop" motion capture from the animal dataset [52].

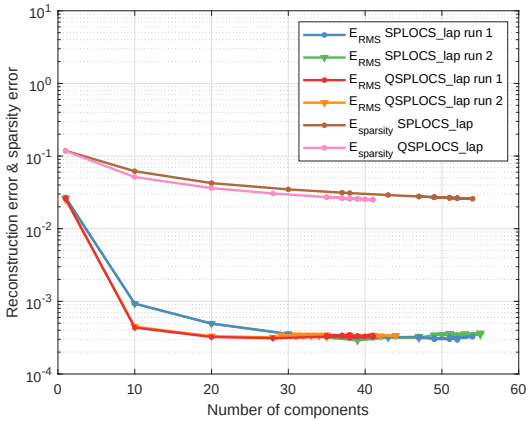


(a) E_{RMS} and $E_{sparsity}$ computed on 308 training frames.

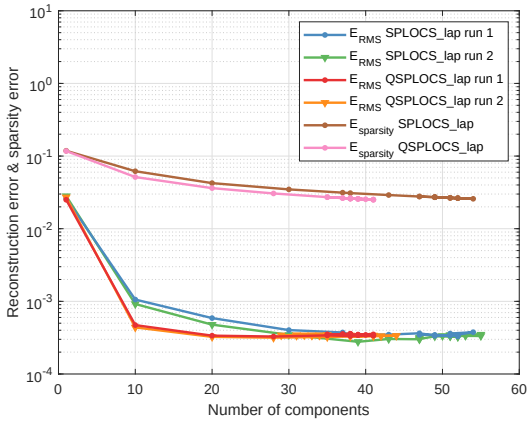


(b) E_{RMS} and $E_{sparsity}$ computed on 77 (unseen) test frames.

Figure 7.34: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "face" motion capture from [66].

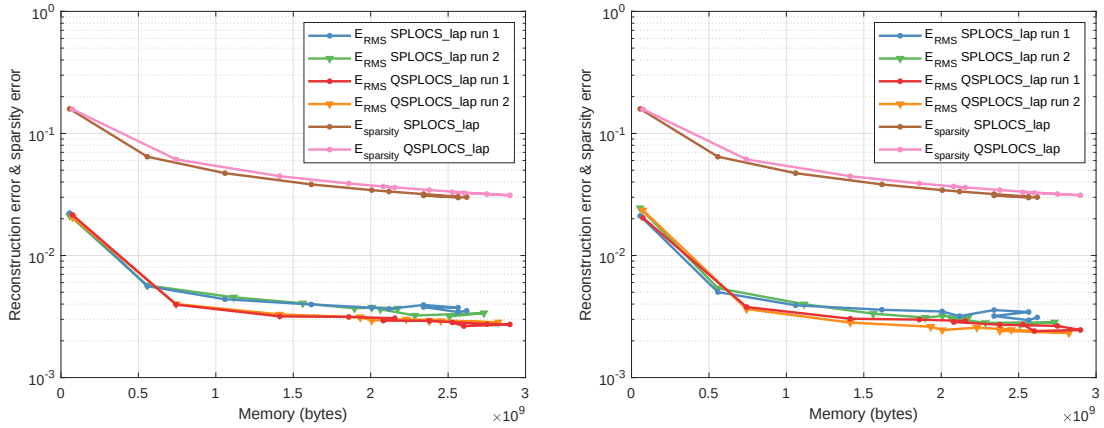


(a) E_{RMS} and $E_{sparsity}$ computed on 481 training frames.



(b) E_{RMS} and $E_{sparsity}$ computed on 120 (unseen) test frames.

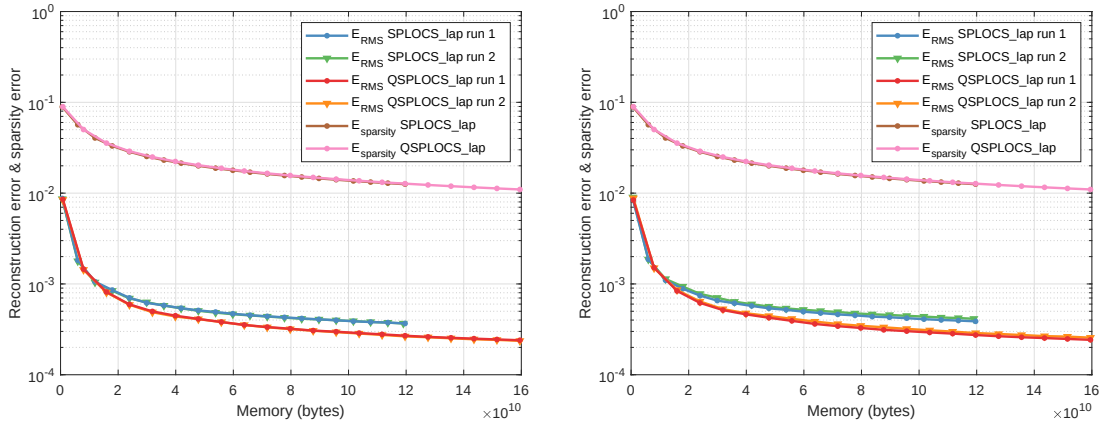
Figure 7.35: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "Sign D poses" motion capture from the TCD Hands dataset [20].



(a) E_{RMS} and $E_{sparsity}$ computed on 40 training frames.

(b) E_{RMS} and $E_{sparsity}$ computed on 9 (unseen) test frames.

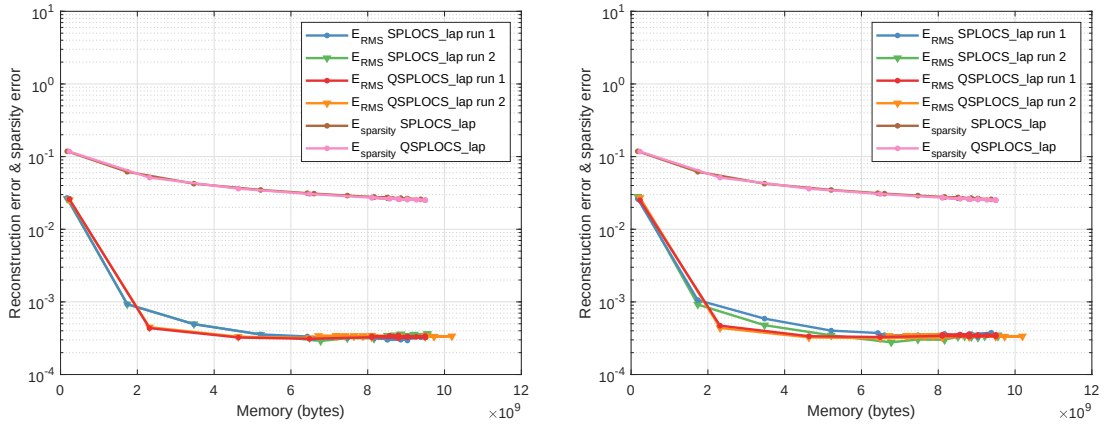
Figure 7.36: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "horse-gallop" motion capture from the animal dataset [52].



(a) E_{RMS} and $E_{sparsity}$ computed on 308 training frames.

(b) E_{RMS} and $E_{sparsity}$ computed on 77 (unseen) test frames.

Figure 7.37: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "face" motion capture from [66].



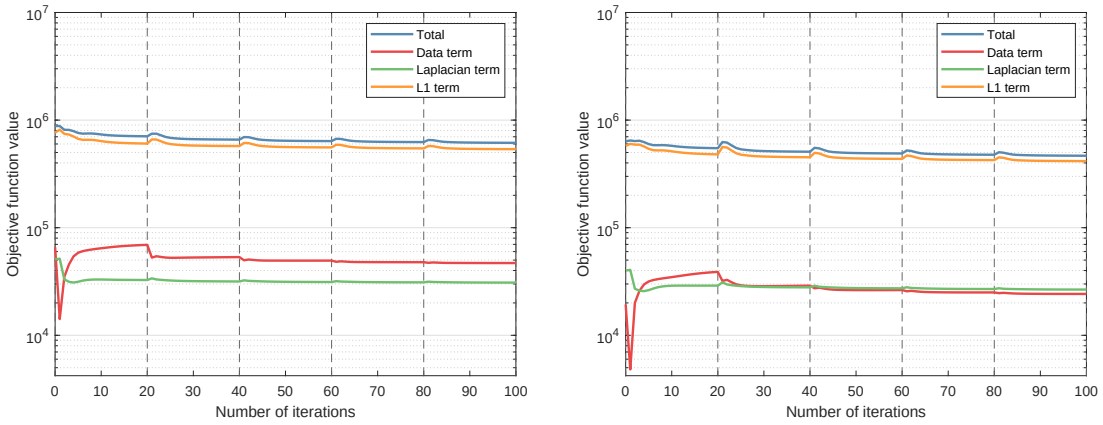
(a) E_{RMS} and $E_{sparsity}$ computed on 481 training frames.

(b) E_{RMS} and $E_{sparsity}$ computed on 120 (unseen) test frames.

Figure 7.38: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "Sign D poses" motion capture from the TCD Hands dataset [20].

As a second experiment, we again compare the convergence rates of the two methods. This is shown in Figure 7.39 for the "face" dataset, where $d = 100$ components were selected.

As before, a slight peak can be observed after every 20 iterations due to the weight update step. Nevertheless, the total objective function value appears to go down on all other iterations for both methods. QSPLOCS_lap appears to have a slightly lower objective value function for both the data term and the sparsity inducing term. This means that the quaternion method is able to get components that are more sparse, while also being able to give a better data approximation.



(a) Convergence of SPLOCS_lap.

(b) Convergence of QSPLOCS_lap.

Figure 7.39: Objective function value (y-axis) with respect to the number of iterations (x-axis) on the "face" motion capture from [66].

As a final quantitative experiment, we compare SPLOCS_lap and QSPLOCS_lap based on their computation times on various datasets. This is shown in Table 7.2. As before, the average time is taken over two runs and the algorithms are run on the same laptop.

Similar to what we saw for SPLOCS and QSPLOCS in Table 7.1, SPLOCS_lap is again shown to be faster than its quaternion variant on all datasets considered. The difference in speed between the two methods is now, however, even more pronounced. On the "horse-gallop" dataset, SPLOCS_lap is now shown to be on average 3.1 times faster than QSPLOCS_lap. On the "C5 - Walk to run" dataset, SPLOCS_lap is 4.0 times as fast on average. When we compare the results to those obtained by SPLOCS and QSPLOCS, we observe that the quaternion variants only differ slightly in computation times. The QSPLOCS_lap method is slightly slower due to the added step of optimizing the Laplacian term. When comparing the real methods, however, we notice that SPLOCS_lap is actually faster than the original SPLOCS method. This has to do with the fact that we removed the spatially varying regularization parameters. These parameters had to be computed for all components and after each weight optimization iteration using the heat method [13]. Thus, it appears that for the real methods it is more computationally efficient to optimize the Laplacian term in each ADMM iteration than it is to compute the spatially varying regularization strengths after each weight optimization iteration. For the quaternion methods, the opposite holds. One reason for this is the inefficient implementation of the Laplacian term optimization step from Equation 5.18. Since the `qtfm` toolbox does not contain a function for solving a system of quaternion linear equations, we solve this by rewriting the quaternion system as a real system and using built-in MATLAB functions for solving a system of real linear equations. Thereafter, the resulting matrix is converted back to quaternions again. Thus, the QSPLOCS_lap computation times could be improved by finding a more efficient way of computing this step.

Table 7.2: Comparison of computation times when using SPLOCS_lap (SPLOCS in table) and QSPLOCS_lap (QSPLOCS in table). The table shows the number of vertices n, the number of frames f and the computation times using a varying number of components d on a variety of datasets. i7 2.50GHz, 8GB RAM, Matlab implementation.

Mesh	n	f	Times for $d = 10$ comp.		Times for $d = 50$ comp.		Times for $d = 100$ comp.	
			SPLOCS	QSPLOCS	SPLOCS	QSPLOCS	SPLOCS	QSPLOCS
“horse-gallop” [52]	8431	49	13.22s	27.79s	35.34s	114.54s	64.43s	219.29s
“face” [66]	23725	385	54.27s	174.03s	189.81s	749.73s	362.15s	1502.10s
“Sign D poses” [20]	6890	601	16.52s	60.28s	63.37s	260.92s	108.10s	404.65s
“C5 - Walk to run” [54]	6890	650	17.50s	62.84s	69.01s	303.68s	133.65s	563.78s
“E5 - Hook left poses” [54]	6890	327	11.60s	38.83s	45.65s	182.80s	81.51s	299.46s
“jumping-jacks-50022” [7]	6890	308	11.97s	39.86s	44.47s	179.56s	82.47s	332.59s
“jiggle-on-toes-50004” [7]	6890	239	10.38s	32.75s	37.49s	135.89s	65.15s	231.58s

7.3.2. Visual comparison

Perhaps even more important than the quantitative analysis is a visual comparison of the two newly proposed sparse PCA techniques. The goal of this is to see whether the proposed methods including the Laplacian term are indeed able to obtain components that are sparse and localized, while also being smooth and natural looking. Since the Laplacian term penalizes neighboring vertices with a large difference in value, we expect to solve the problem related to the sharp boundaries that was observed for the SPLOCS and QSPLOCS methods in Section 7.2.2. All experiments in this section are performed on the same three datasets, where the methods were applied to obtain $d = 50$ sparse components. Results obtained from other datasets are shown in Appendix B.2.2.

First, we visualize a few components of both methods in order to assess how sparse, localized and smooth the components are. Since we are also interested in knowing whether the newly proposed methods are able to solve problems related to the original SPLOCS method, we do this by selecting the four SPLOCS_lap and QSPLOCS_lap components that describe a similar motion as the components visualized in Section 7.2.2. Note that in some cases we were not able to find components showing a similar motion. As before, the weights corresponding to the components are selected manually.

The SPLOCS_lap components are visualized in Figures 7.40, 7.45 and 7.50 and the corresponding QSPLOCS_lap components can be found in Figures 7.41-7.44, 7.46-7.49 and 7.51-7.54 respectively. From these figures it becomes clear that the two methods are indeed able to get some components that are sparse and localized. When comparing SPLOCS_lap to its quaternion variant, we notice similar properties to those observed for SPLOCS and QSPLOCS in Section 7.2.2. First, the deformations described by the SPLOCS_lap components are often also described along one of the dimensions of the corresponding QSPLOCS_lap component. However, the area covered by the motion of the real component and its quaternion variant can differ. An example of this is the 4th component of the “Sign D poses” dataset shown in Figures 7.50 and 7.52. The SPLOCS_lap component shows a deformation in the head and torso area while the corresponding quaternion component shows a deformation in the head and arm regions.

Even though many of the components are sparse and localized, the area covered by the components can vary a lot in size. Some components cover a large area, such as the 12th SPLOCS_lap component from the “face” dataset shown in Figure 7.45, while other components are too sparse and localized, such as the 19th QSPLOCS_lap components from the “Sign D poses” dataset in Figure 7.53. This is as expected from the proposed methods, since they no longer make use of the spatially varying regularization strengths. Furthermore, natural looking deformations can in theory cover a range of different area sizes. Nevertheless, not all components obtained by the proposed methods describe a natural motion, such as the latter example described above, where only a single fingertip is in motion.

Another drawback of the proposed methods is that not all components are localized. An example of this is given in Figure 7.43, where the deformation not only affects the hind leg of the horse, but additional moves its two hooves. One potential cause of this could be the optimizer used by the two methods. ADMM is a first order method that can end up in non-optimal local minima. Throughout the experiments performed on all four sparse PCA methods, we noticed that ADMM typically does not progress that much and therefore does not always reach a good solution. If a better solution would be reached, the deformations in the hooves would most likely be added to another component which would result in a lower objective function value.

When comparing the components obtained by the proposed methods to the components computed by SPLOCS and QSPLOCS in Section 7.2.2, we notice that the added Laplacian term is indeed able to produce components that are more smooth and natural looking. One example of this is shown in Figure 7.52, which can be compared to the corresponding QSPLOCS components shown in Figure 7.20. Where the QSPLOCS component shows a deformation in the head region with a very sharp and unnatural boundary below the neck, the corresponding QSPLOCS_lap component shows a smooth transition throughout the deformation. Thus, the sharp boundaries that were observed in the SPLOCS and QSPLOCS components are no longer present. However, this example also demonstrates that the deformations can be too smooth, resulting in implausible and crooked face shapes.

In conclusion, this experiment has given us a lot of insight on the properties of SPLOCS_lap and QSPLOCS_lap and their drawbacks. The methods are able to obtain components that are in many cases shown to be sparse, localized and smooth. As before, the deformation described by the real component is oftentimes also present in the space spanned by its corresponding quaternion component. The areas covered by the components can vary a lot in size, which can be beneficial in datasets showing both large and small deformations. In some cases, however, the methods are shown to obtain components that are too localized and only move a few vertices of the mesh. Another drawback is that some components are not localized at all, meaning that they can impact different areas of the mesh that are not necessarily connected. This problem could potentially be solved by using another optimizer that uses second order information.

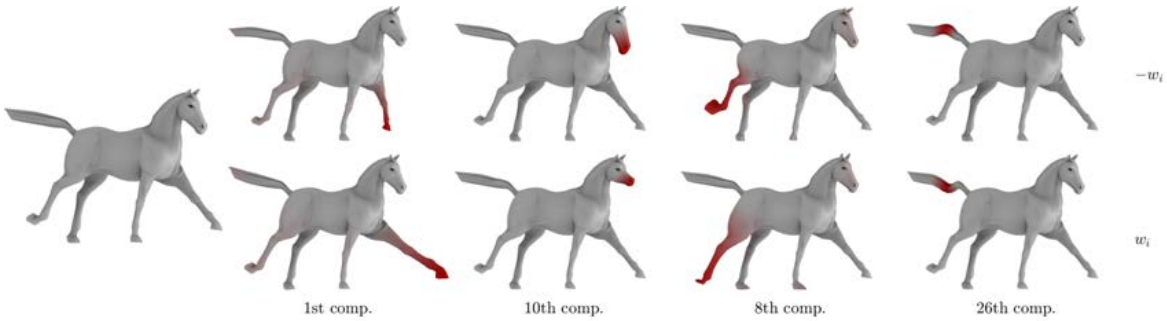


Figure 7.40: Visualization of SPLOCS_lap: 4 sparse components on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

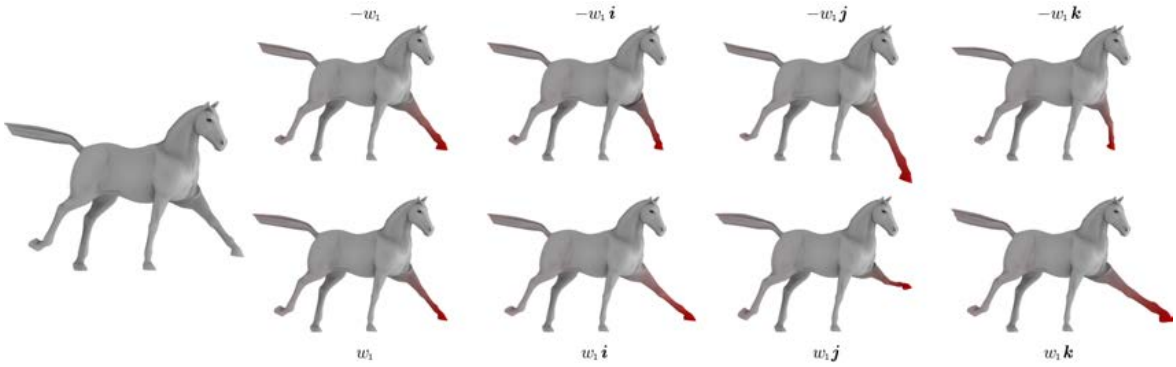


Figure 7.41: Visualization of QSPLOCS_lap: The 1st quaternion sparse component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_1$ (top) and w_1 (bottom) along the four dimensions of the 1st component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

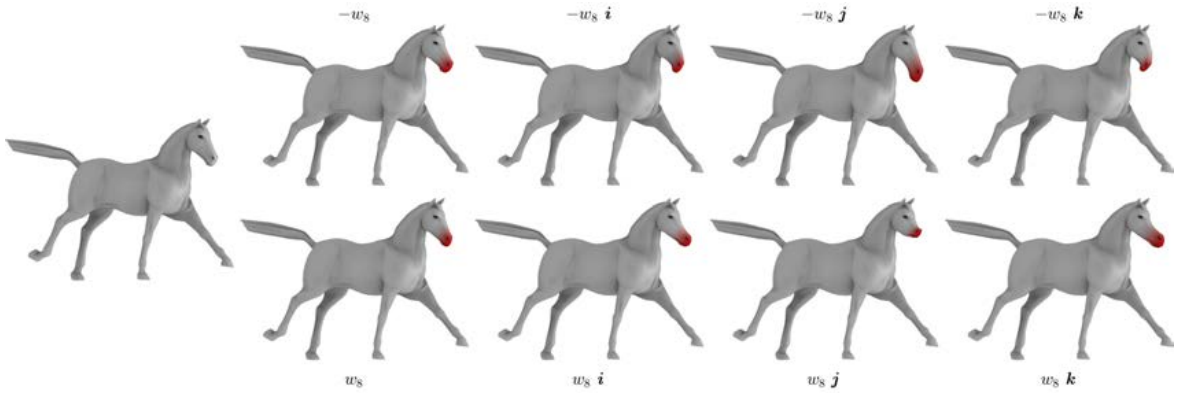


Figure 7.42: Visualization of QSPLOCS_lap: The 8th quaternion sparse component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_8$ (top) and w_8 (bottom) along the four dimensions of the 8th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

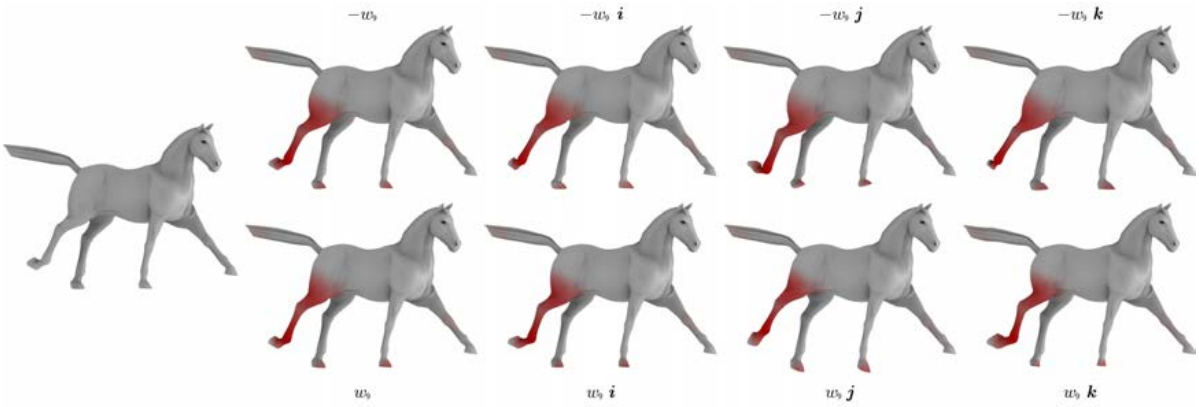


Figure 7.43: Visualization of QSPLOCS_lap: The 9th quaternion sparse component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_9$ (top) and w_9 (bottom) along the four dimensions of the 9th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

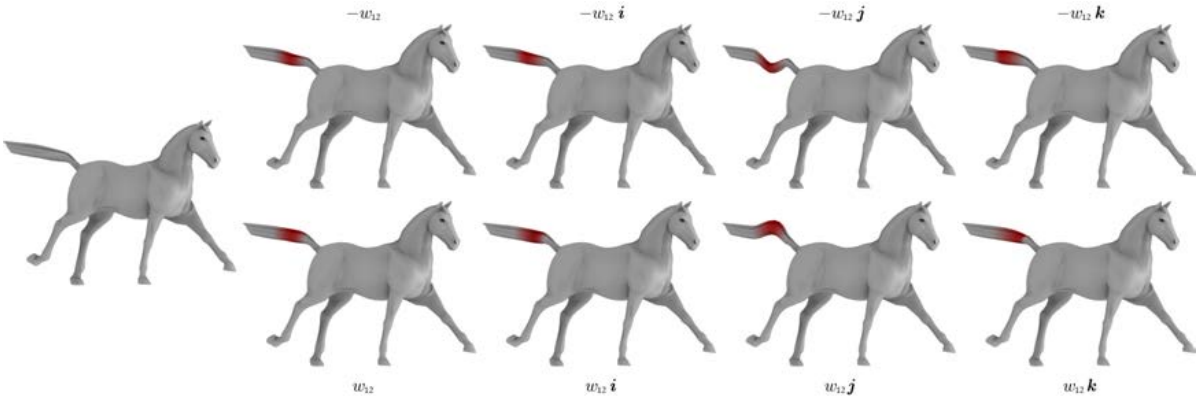


Figure 7.44: Visualization of QSPLOCS_lap: The 12th quaternion sparse component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_{12}$ (top) and w_{12} (bottom) along the four dimensions of the 12th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

Figure 7.45: Visualization of SPLOCS_lap: 4 sparse components on the “face” motion capture from [66]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

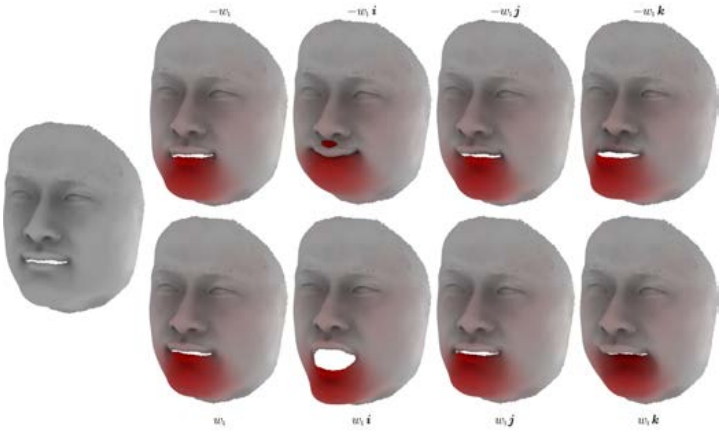


Figure 7.46: Visualization of QSPLOCS_lap: The 1st quaternion sparse component on the “‘face”’ motion capture from [66]. Models corresponding to $-w_1$ (top) and w_1 (bottom) along the four dimensions of the 1st component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

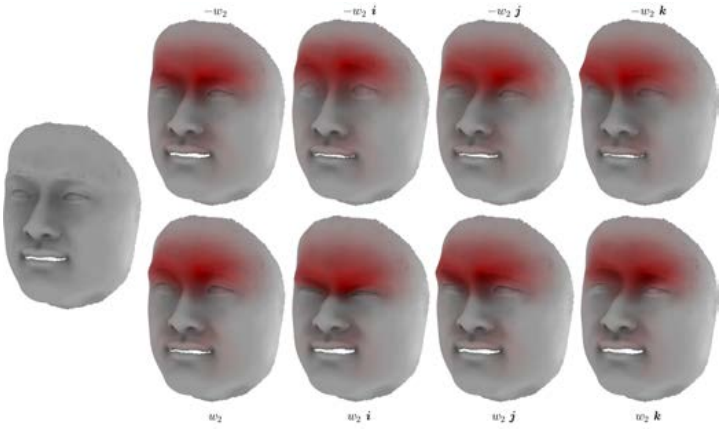


Figure 7.47: Visualization of QSPLOCS_lap: The 2nd quaternion sparse component on the “‘face”’ motion capture from [66]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

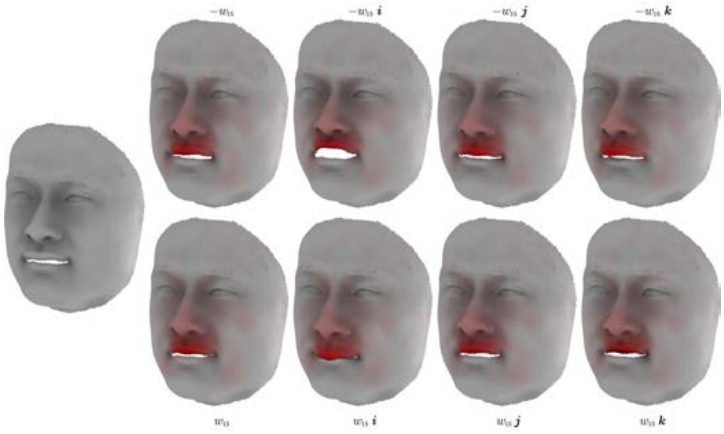


Figure 7.48: Visualization of QSPLOCS_lap: The 15th quaternion sparse component on the "'face"' motion capture from [66]. Models corresponding to $-w_{15}$ (top) and w_{15} (bottom) along the four dimensions of the 15th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

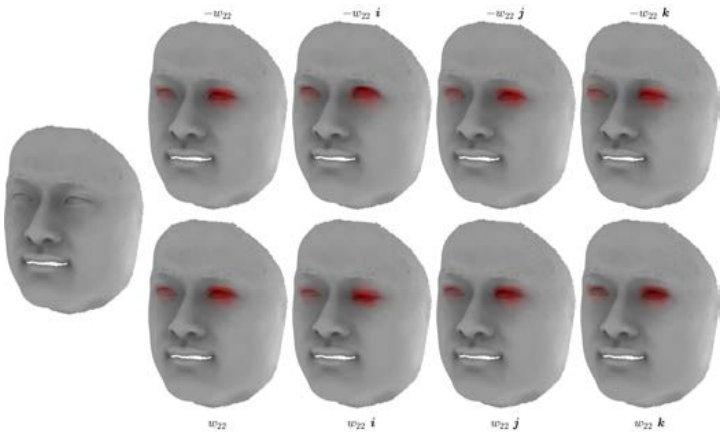


Figure 7.49: Visualization of QSPLOCS_lap: The 22nd quaternion sparse component on the "'face"' motion capture from [66]. Models corresponding to $-w_{22}$ (top) and w_{22} (bottom) along the four dimensions of the 22nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

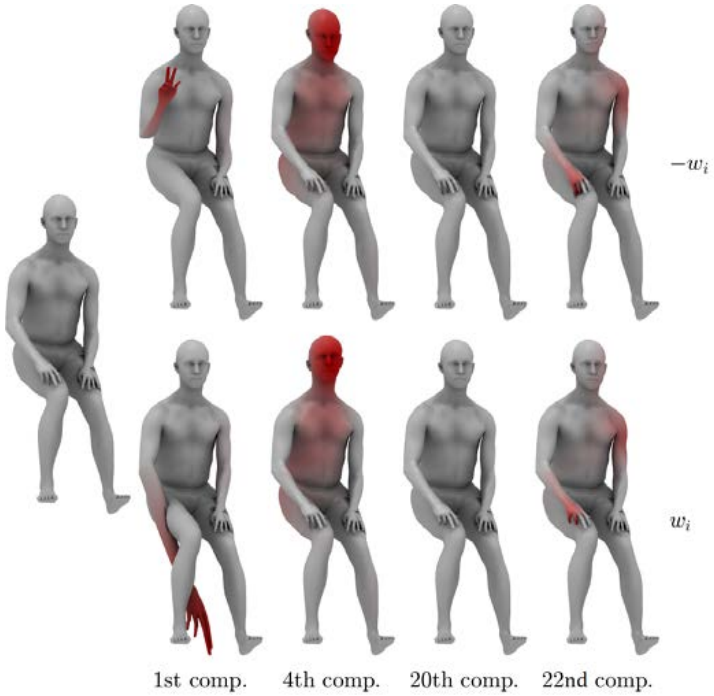


Figure 7.50: Visualization of SPLOCS_lap: 4 sparse components on the “Sign D poses” motion capture from TCD Hands[20]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

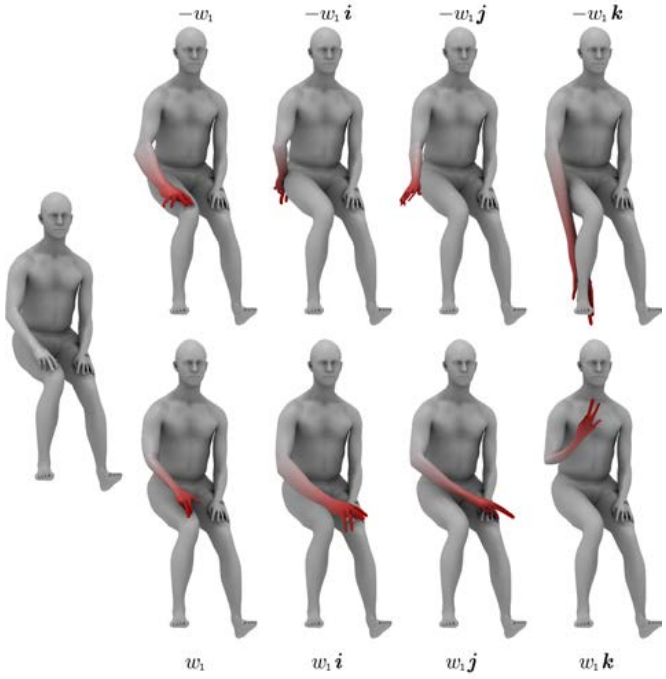


Figure 7.51: Visualization of QSPLOCS_lap: The 1st quaternion sparse component on the “‘Sign D poses”’ motion capture from TCD Hands [20]. Models corresponding to $-w_1$ (top) and w_1 (bottom) along the four dimensions of the 1st component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

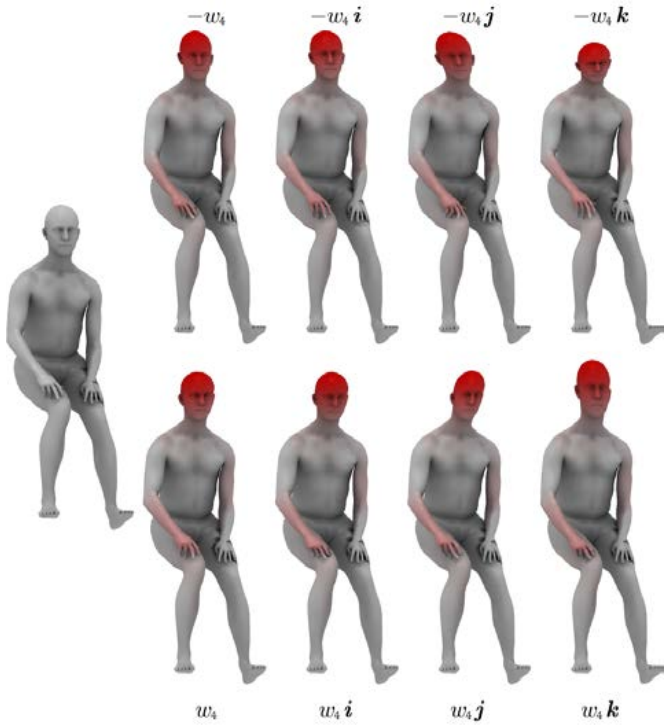


Figure 7.52: Visualization of QSPLOCS_lap: The 4th quaternion sparse component on the "'Sign D poses"' motion capture from TCD Hands [20]. Models corresponding to $-w_4$ (top) and w_4 (bottom) along the four dimensions of the 4th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

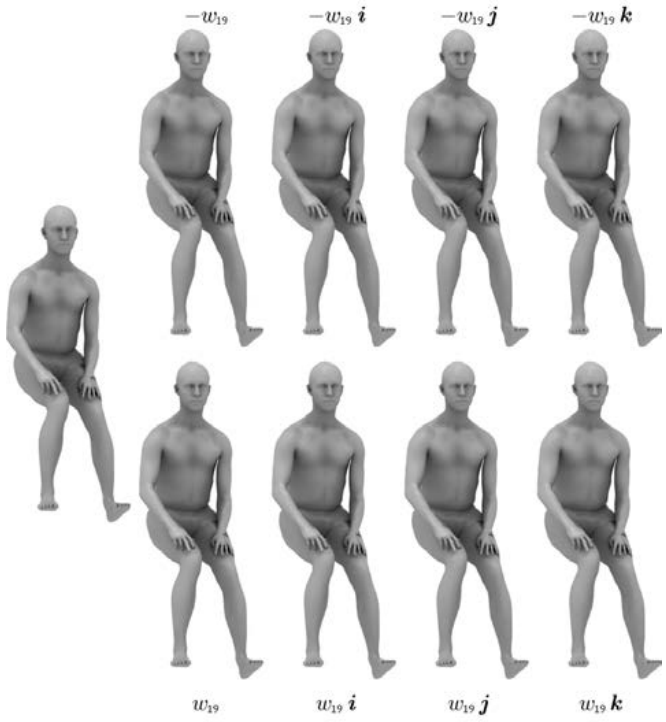


Figure 7.53: Visualization of QSPLOCS_lap: The 19th quaternion sparse component on the "'Sign D poses"' motion capture from TCD Hands [20]. Models corresponding to $-w_{19}$ (top) and w_{19} (bottom) along the four dimensions of the 19th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

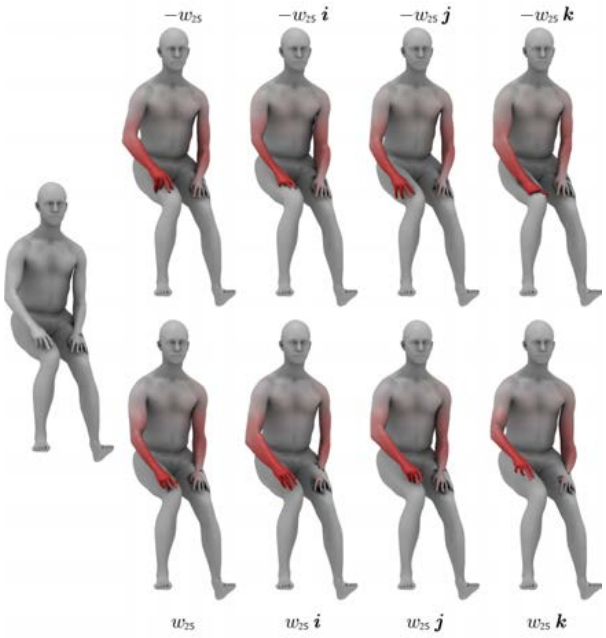


Figure 7.54: Visualization of QSPLOCS_lap: The 25th quaternion sparse component on the "'Sign D poses"' motion capture from TCD Hands [20]. Models corresponding to $-w_{25}$ (top) and w_{25} (bottom) along the four dimensions of the 25th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

Since the ADMM optimizer is shown to not progress that much, the result is very dependent on the initialization of the components. As a second experiment, we aim to visualize this dependency by applying the two proposed methods using a random initialization phase similar to the experiment performed in Section 7.2.2. For SPLOCS and QSPLOCS, we observed that random initialization can lead to very noisy components or components that are positioned strangely and therefore seem unnatural. For these proposed methods, we expect to solve the problem related to noise due to the added Laplacian term. Results obtained on the "horse-gallop" and "face" datasets are shown in Figures 7.55 and 7.58 for SPLOCS and in Figures 7.56, 7.57, 7.59 and 7.60 for QSPLOCS.

From the figures it becomes clear that the added Laplacian term indeed gets rid of the noise problem that was present for SPLOCS and QSPLOCS. The components are still smooth and in some cases also sparse. However, since all components are initialized randomly and since we do not use the spatially varying regularization strengths, we observe that many of the components are not restricted to one small area of the mesh. Most of the components are shown to deform many different regions, such as the 1st SPLOCS_lap component in Figure 7.55 and the 22nd QSPLOCS_lap component in Figure 7.60. This again shows that ADMM is not able to reach a good solution, since a lower objective function value could most likely be obtained by moving the deformations of each region to another component.

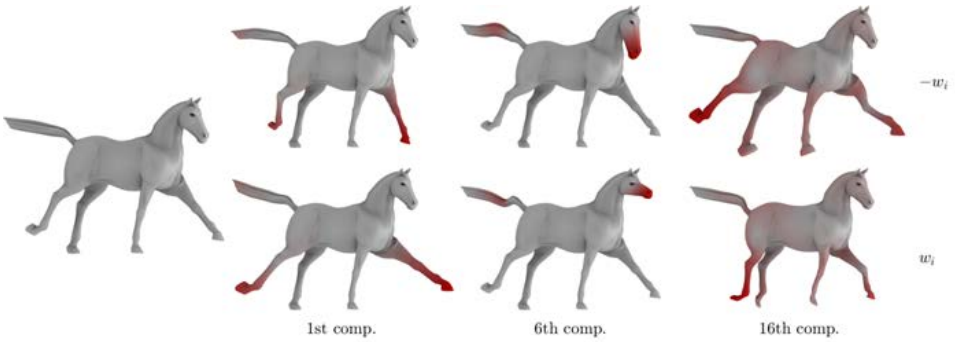


Figure 7.55: Visualization of SPLOCS_lap using random initialization: 3 sparse components on the "horse-gallop" motion capture from the animal dataset [52]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

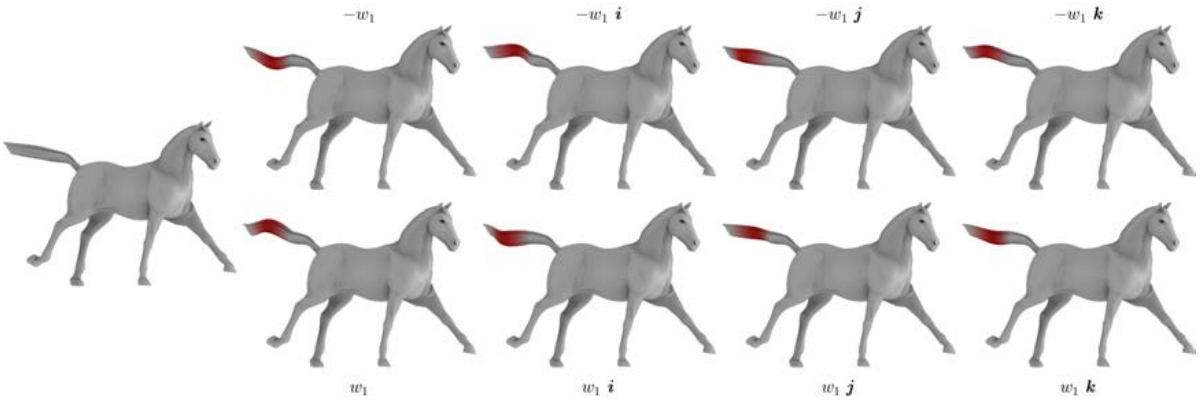


Figure 7.56: Visualization of QSPLOCS_lap using random initialization: The 1st quaternion sparse component on the "horse-gallop" motion capture from the animal dataset [52]. Models corresponding to $-w_1$ (top) and w_1 (bottom) along the four dimensions of the 1st component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

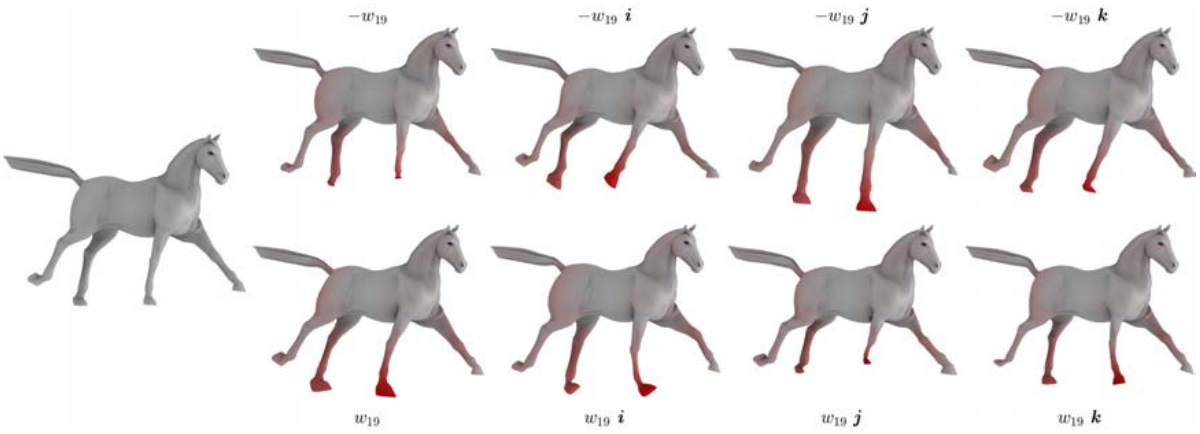


Figure 7.57: Visualization of QSPLOCS_lap using random initialization: The 19th quaternion sparse component on the "horse-gallop" motion capture from the animal dataset [52]. Models corresponding to $-w_{19}$ (top) and w_{19} (bottom) along the four dimensions of the 19th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

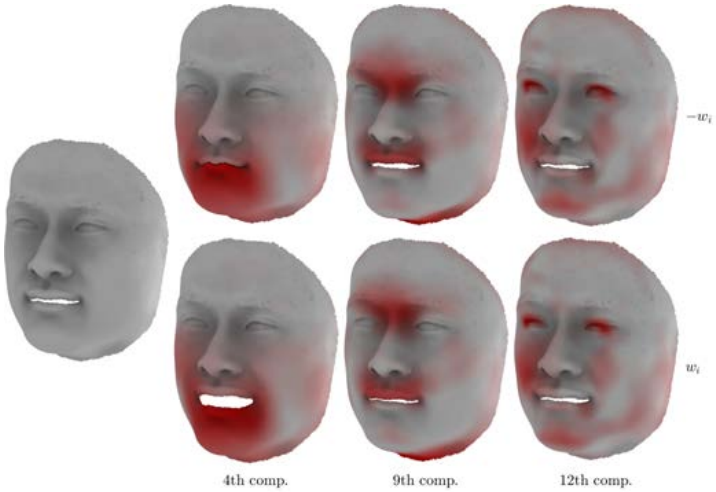


Figure 7.58: Visualization of SPLOCS_lap using random initialization: 3 sparse components on the "face" motion capture from [66]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

Figure 7.59: Visualization of QSPLOCS_lap using random initialization: The 1st quaternion sparse component on the "'face"' motion capture from [66]. Models corresponding to $-w_1$ (top) and w_1 (bottom) along the four dimensions of the 1st component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

Figure 7.60: Visualization of QSPLOCS_lap using random initialization: The 22nd quaternion sparse component on the "'face"' motion capture from [66]. Models corresponding to $-w_{22}$ (top) and w_{22} (bottom) along the four dimensions of the 22nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

Similar to the experiment performed on SPLOCS and QSPLOCS in the previous section, it is also interesting to see how well SPLOCS_lap and QSPLOCS_lap are able to handle articulated rotations. Since the components of SPLOCS_lap are also linear, we expect to see similar artifacts as observed in SPLOCS, since the components will not be able to describe the curvilinear paths present in some datasets. As before, we perform this experiment on the "horse-gallop" dataset only.

The experiment performed on SPLOCS_lap is shown in Figures 7.61 and 7.62 and the results of QSPLOCS_lap are shown in Figures 7.63 and 7.64. The results are very similar to what we observed for SPLOCS and QSPLOCS. As expected, the SPLOCS_lap components follow a linear path, which introduces linearization artifacts. The QSPLOCS_lap components seem to be able to describe the articulated rotations better. Using different weights, the components are able to follow a curvilinear path. However, similar to QSPLOCS, it is very hard to properly describe this path due to the four dimensional weights. As can be seen in Figure 7.63, multiple different paths can be observed, where some configurations introduce new artifacts.

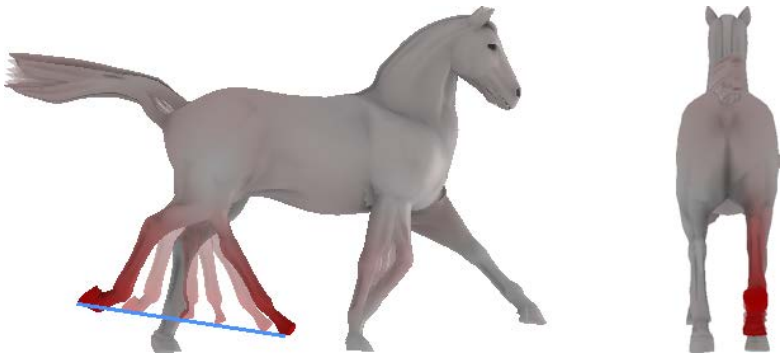


Figure 7.61: Visualization of SPLOCS_lap on the “horse-gallop” dataset [52] showing its limitation regarding articulated rotations. Models corresponding to the largest and smallest weights present in the training data are displayed as well as some values in between. The experiment is performed on the 2nd component.

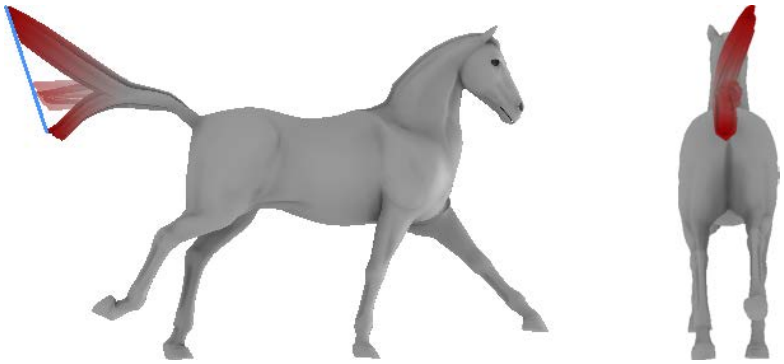


Figure 7.62: Visualization of SPLOCS_lap on the “horse-gallop” dataset [52] showing its limitation regarding articulated rotations. Models corresponding to the largest and smallest weights present in the training data are displayed as well as some values in between. The experiment is performed on the 5th component.

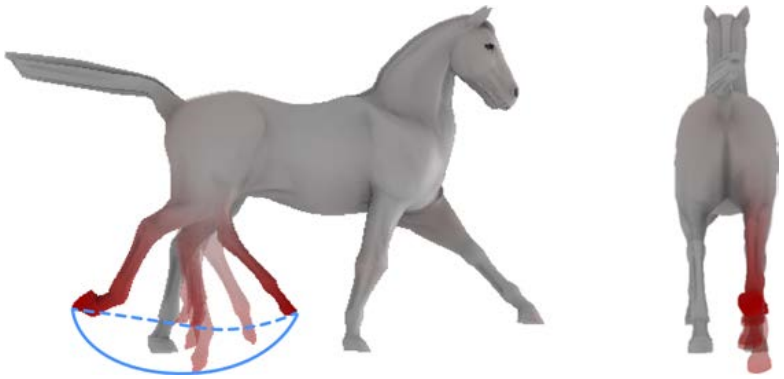


Figure 7.63: Visualization of QSPLOCS_lap on the “horse-gallop” dataset [52] showing how it handles articulated rotations. Models corresponding to the largest, median and smallest weights along each dimension present in the training data are displayed. The experiment is performed on the 2nd component.

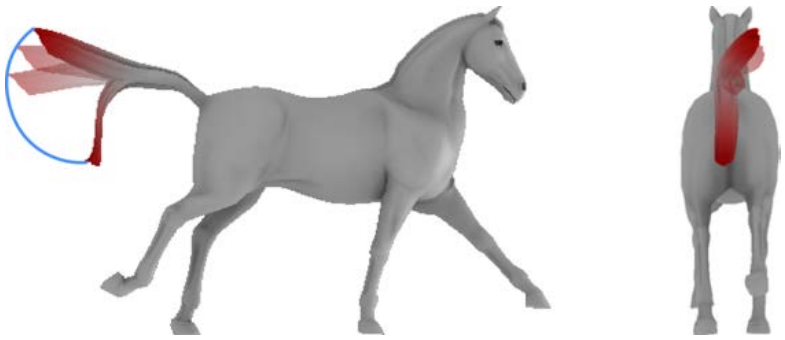


Figure 7.64: Visualization of QSPLOCS_lap on the "horse-gallop" dataset [52] showing how it handles articulated rotations. Models corresponding to the largest, median and smallest weights along each dimension present in the training data are displayed. The experiment is performed on the 5th component.

7.4. Combined method

From the experiments performed in the previous sections we can conclude that all discussed sparse PCA techniques have their advantages and disadvantages. The SPLOCS and QSPLOCS methods are able to generate components that are sparse and only affect a small part of the mesh. In some cases, however, the deformations described by the components lack smoothness and can therefore look unnatural. Furthermore, the performance of the methods is very dependent on the initialization phase. These issues are partially due to the spatially varying regularization strengths that are used in both methods.

The SPLOCS_lap and QSPLOCS_lap methods, which replace these spatially varying regularization strengths with an additional term based on the Laplacian, are able to improve on some of the aspects mentioned above. Due to the Laplacian term, the components are able to look much more smooth and natural. Furthermore, this term gets rid of the noise problem that was observed for SPLOCS and QSPLOCS using a random initialization. However, since the spatially varying regularization strengths are removed in these methods, the components found by these methods are in many cases either too sparse and localized, or not localized at all. Thus, it appears that the Laplacian term in combination with the sparsity inducing term are not able to enforce the components to consistently be both sparse and localized.

In this section, we investigate whether we can construct a new sparse PCA technique, which takes the positive aspects of all methods discussed above. We do this by including both the spatially varying regularization strengths and the Laplacian term in the optimization problem. The former will give components that are both sparse and localized, while the latter term will make sure that these components are smooth. The computation times of this combined method and its quaternion variant are given in Table 7.3. As before, the average computation time is taken over two runs. From the table we observe that the combined method has higher computation times than both SPLOCS and SPLOCS_lap. The same holds for the quaternion variants. This is as expected, since we now have to compute the spatially varying regularization strengths after each weight iteration as well as solve Equation 5.18 in each ADMM iteration. The difference in computation times is, however, still reasonable. The largest observed value in the table, 1652.70s for the quaternion combined method, is only $10 - 15\%$ higher than the computation times taken from QSPLOCS_lap and QSPLOCS respectively.

Table 7.3: Comparison of computation times when using the combined method (comb. 1) and its quaternion variant (comb. 2). The table shows the number of vertices n, the number of frames f and the computation times using a varying number of components d on a variety of datasets. i7 2.50GHz, 8GB RAM, Matlab implementation.

Mesh	n	f	Times using $d = 10$ comp.		Times using $d = 50$ comp.		Times using $d = 100$ comp.	
			comb. 1	comb. 2	comb. 1	comb. 2	comb. 1	comb. 2
"horse-gallop" [52]	8431	49	26.57s	30.20s	68.83s	143.07s	147.36s	287.80s
"face" [66]	23725	385	73.92s	177.78s	300.34s	823.81s	590.51s	1652.70s
"Sign D poses" [20]	6890	601	21.06s	61.98s	92.24s	299.14s	177.69s	586.16s

We evaluate the combined method by visually comparing it to SPLOCS and SPLOCS_lap. Thereafter, we show a few example components obtained by its quaternion variant. These experiments are performed on the three datasets mentioned at the start of this chapter. Figures 7.65-7.73 show example components of SPLOCS, SPLOCS_lap and the combined method. We do this by selecting components from the combined technique at random and finding the SPLOCS and SPLOCS_lap components that describe a similar deformation. Results show that the combined method is indeed able to generate components that are sparse, localized and smooth. If we look at Figure 7.65, we notice that the combined method shows a smoother variant of the deformation described by SPLOCS. The sharp boundary at the neck has disappeared due to the Laplacian term. Furthermore, the method covers a larger area than the corresponding SPLOCS_lap method, which seems to be too sparse and localized. Similar examples can be found in Figures 7.67, 7.71 and 7.72. The combined method can also result in components that are better localized, as can be seen in Figure 7.66. Here, the SPLOCS component shows additional movement around the chest area of the horse, whereas the combined method only deforms the leg region. Differences between SPLOCS and the combined method are less noticeable on the "face" dataset.

Next, we visualize the components of the quaternion combined method. We do this by selecting the components that show similar deformations to the ones visualized for the real combined method. This is shown in Figures 7.74-7.82. It seems that most of the components obtained by the quaternion combined method are sparse, localized and smooth. Only the component shown in Figure 7.79 seems to be a bit less natural than the components shown in Figure 7.70. The deformation is mostly located on the left eye, while it would be more natural to have the component deform both eyes equally.

In conclusion, we can say that this combined method shows much potential. In comparison to SPLOCS the method is able to obtain similar components with added smoothness near the boundaries. This gets rid of the sharp transitions that were present in SPLOCS. Furthermore, all obtained components only affect a small part of the mesh. This was often not the case for SPLOCS_lap, which yielded many components that were either too sparse and localized or not localized at all. Thus, using the current implementation and solver, this combined method seems to be the preferred choice. This also applies to the quaternion combined method, which shows similar properties.

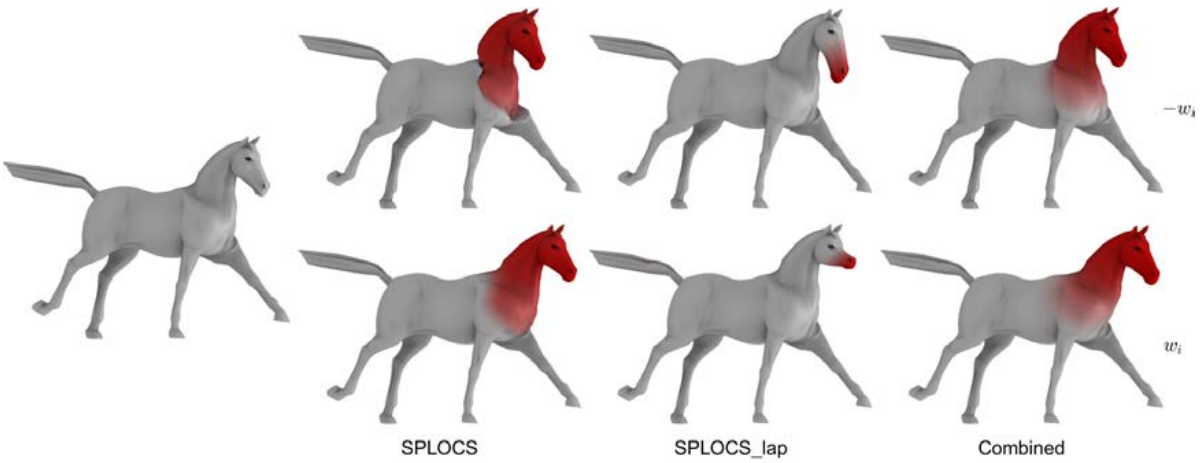


Figure 7.65: Visualization of SPLOCS (left), SPLOCS_lap (middle) and the combined sparse PCA technique (right): Example component on the "horse-gallop" motion capture from the animal dataset [52]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

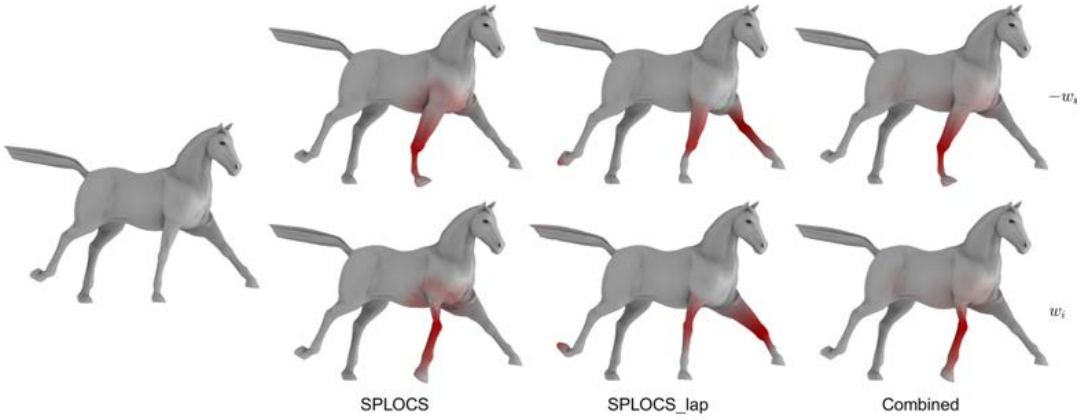


Figure 7.66: Visualization of SPLOCS (left), SPLOCS_lap (middle) and the combined sparse PCA technique (right): Example component on the "horse-gallop" motion capture from the animal dataset [52]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

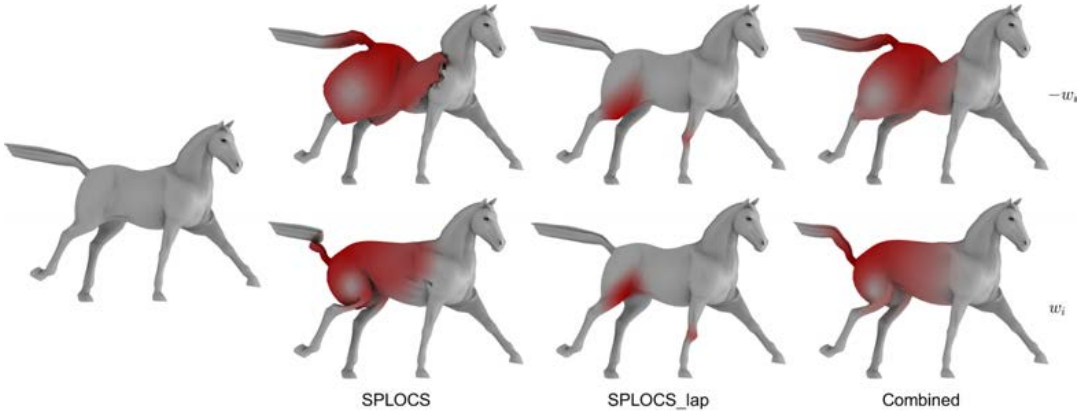


Figure 7.67: Visualization of SPLOCS (left), SPLOCS_lap (middle) and the combined sparse PCA technique (right): Example component on the "horse-gallop" motion capture from the animal dataset [52]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

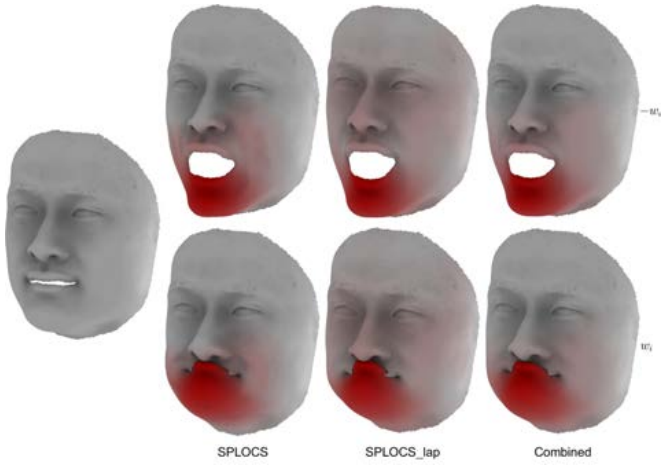


Figure 7.68: Visualization of SPLOCS (left), SPLOCS_lap (middle) and the combined sparse PCA technique (right): Example component on the "face" motion capture from [66]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

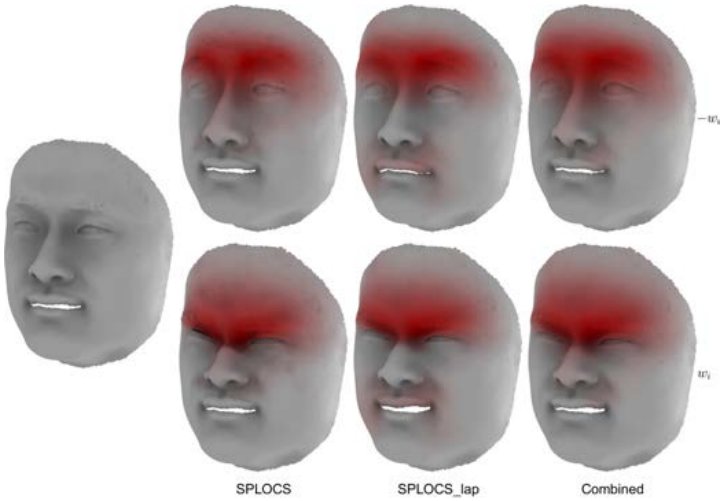


Figure 7.69: Visualization of SPLOCS (left), SPLOCS_lap (middle) and the combined sparse PCA technique (right): Example component on the “face” motion capture from [66]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

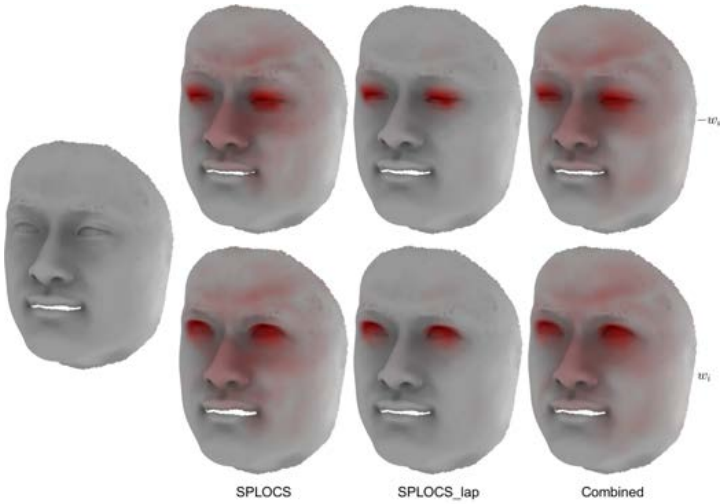


Figure 7.70: Visualization of SPLOCS (left), SPLOCS_lap (middle) and the combined sparse PCA technique (right): Example component on the “face” motion capture from [66]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

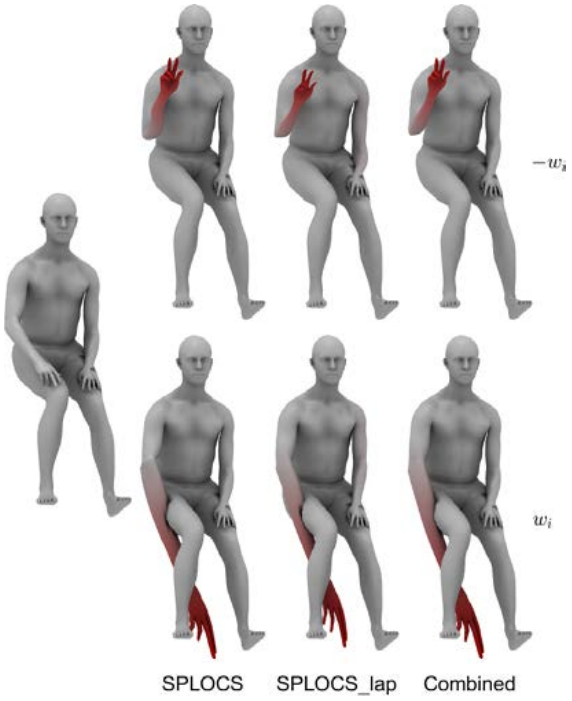


Figure 7.71: Visualization of SPLOCS (left), SPLOCS_lap (middle) and the combined sparse PCA technique (right): Example component on the “Sign D poses” motion capture from the TCD Hands dataset [20]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

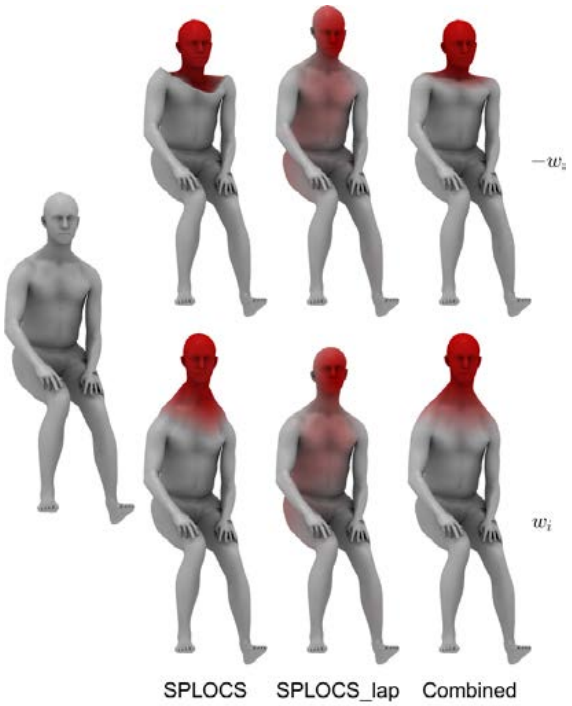


Figure 7.72: Visualization of SPLOCS (left), SPLOCS_lap (middle) and the combined sparse PCA technique (right): Example component on the “Sign D poses” motion capture from the TCD Hands dataset [20]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

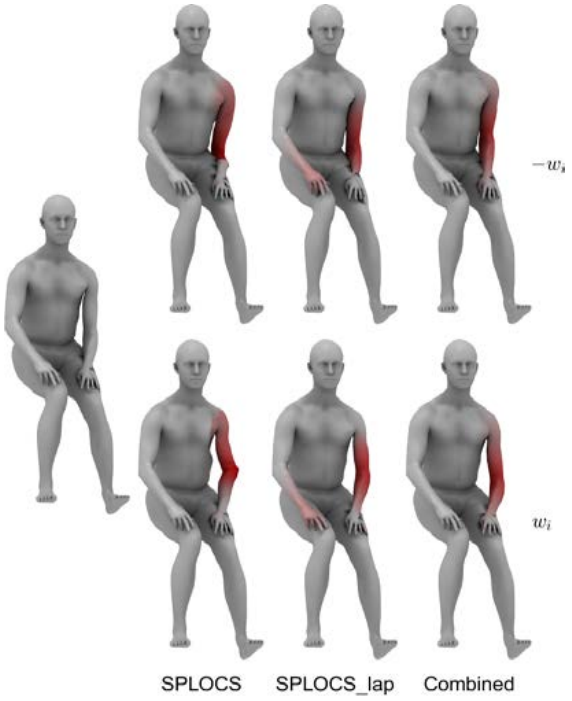


Figure 7.73: Visualization of SPLOCS (left), SPLOCS_lap (middle) and the combined sparse PCA technique (right): Example component on the “Sign D poses” motion capture from the TCD Hands dataset [20]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

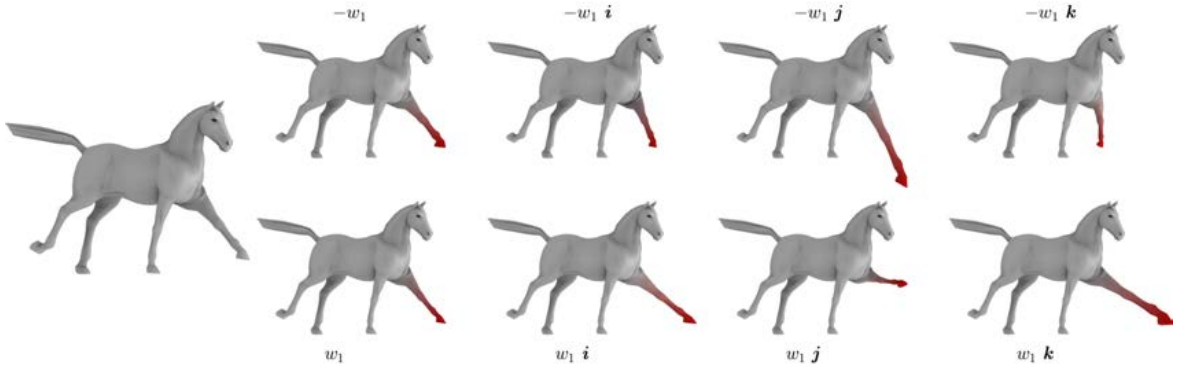


Figure 7.74: Visualization of the quaternion combined sparse PCA technique: Example component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

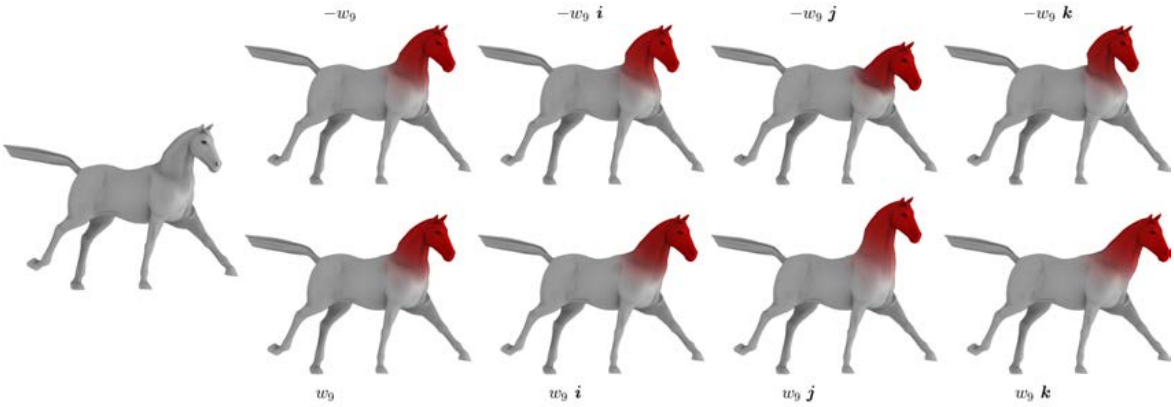


Figure 7.75: Visualization of the quaternion combined sparse PCA technique: Example component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

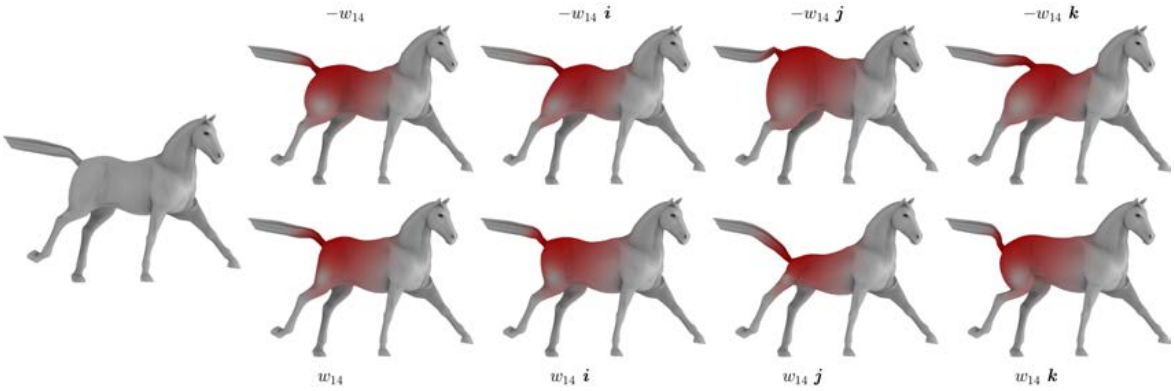


Figure 7.76: Visualization of the quaternion combined sparse PCA technique: Example component on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

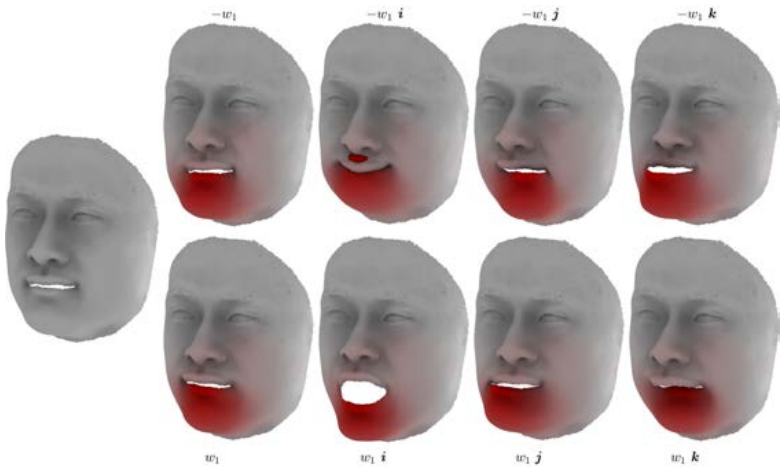


Figure 7.77: Visualization of the quaternion combined sparse PCA technique: Example component on the “face” motion capture from [66]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

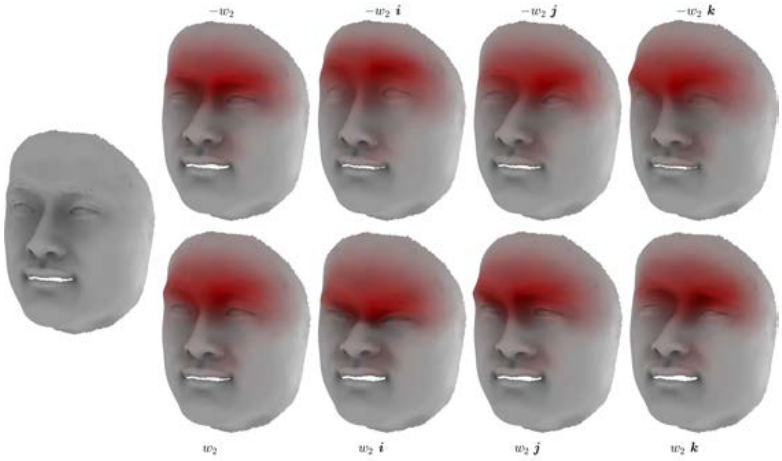


Figure 7.78: Visualization of the quaternion combined sparse PCA technique: Example component on the “face” motion capture from [66]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

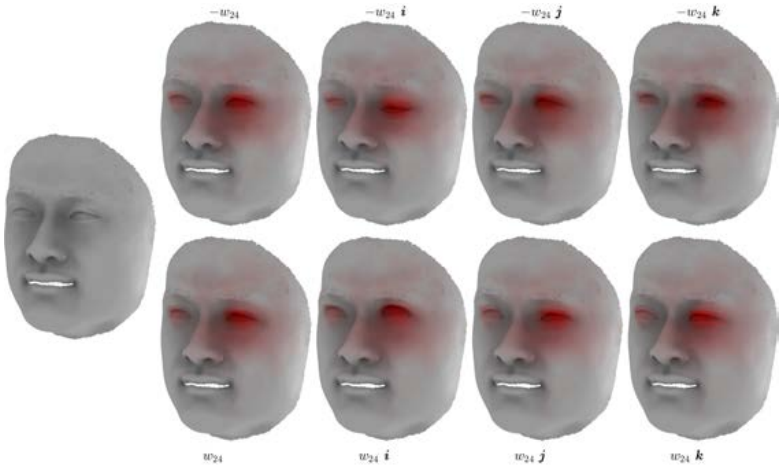


Figure 7.79: Visualization of the quaternion combined sparse PCA technique: Example component on the “face” motion capture from [66]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

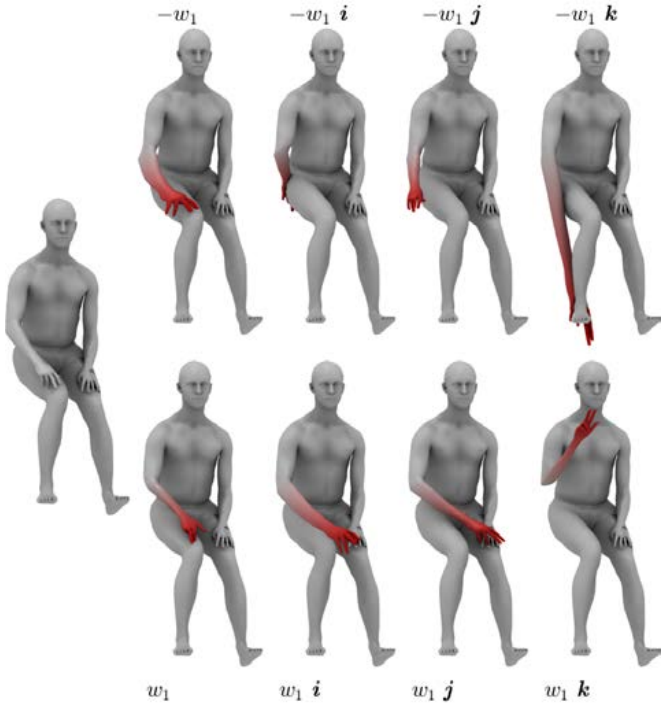


Figure 7.80: Visualization of the quaternion combined sparse PCA technique: Example component on the "Sign D poses" motion capture from the TCD Hands dataset [20]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

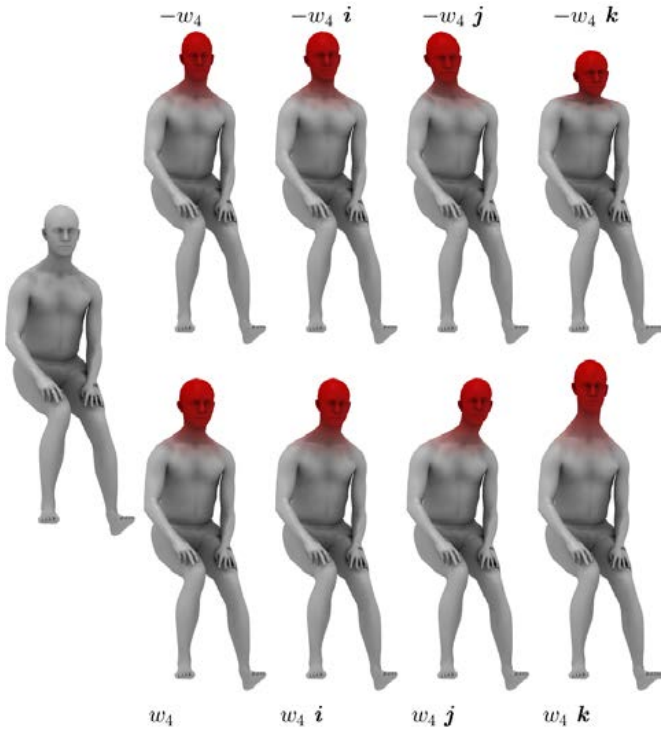


Figure 7.81: Visualization of the quaternion combined sparse PCA technique: Example component on the "Sign D poses" motion capture from the TCD Hands dataset [20]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

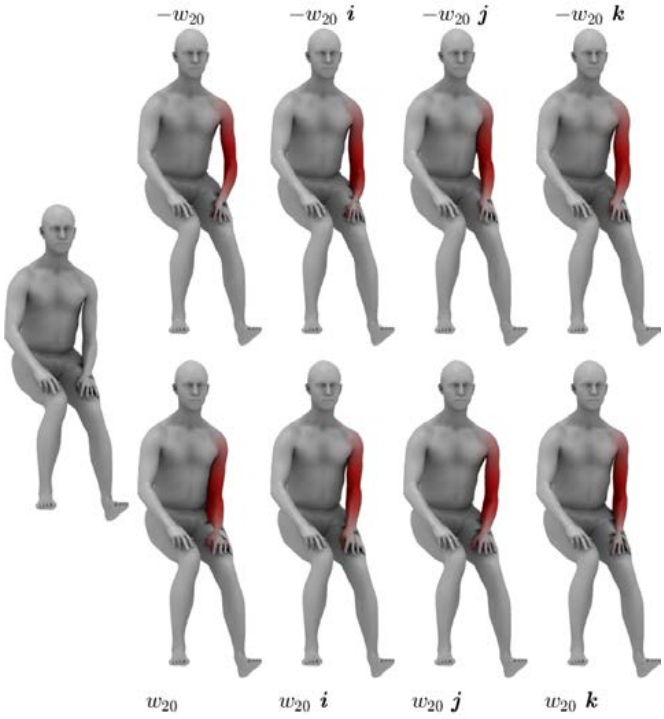


Figure 7.82: Visualization of the quaternion combined sparse PCA technique: Example component on the “Sign D poses” motion capture from the TCD Hands dataset [20]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

8

Discussion & conclusion

The main goal of this work was to investigate whether quaternions could be used in combination with PCA or sparse PCA in order to solve some of the challenges related to these methods in geometry processing applications. By deriving a number of theoretical properties and performing various experiments, we are now able to answer the questions posed at the start of this work. These questions are answered in Section 8.1, where we discuss the main findings and limitations of the proposed methods. Based on this discussion, we then suggest topics for future research in Section 8.2.

8.1. Discussion

Some of the main limitations of PCA and sparse PCA were their variance to rigid motion, resulting in the need of a rigid alignment preprocessing step, as well as the fact that the correlations between the x, y and z coordinates are not taken into account. Through a number of research questions, we aimed to investigate whether quaternions could be utilized to solve these issues. Now, we will review each research question in more detail and discuss the findings, limitations and conclusions that can be made.

8.1.1. PCA

Can quaternions be used in combination with PCA in order to obtain a lower-dimensional subspace that is invariant to rigid motion of the samples?

For this research direction we were most interested in the rotational aspect. Since quaternions can only be rotated by applying both right and left multiplication using a unit quaternion, we found that PCA using quaternions is not invariant to rigid motion. However, through the use of quaternion linear algebra we were able to prove that QPCA is invariant to right-multiplication with a unit quaternion. This means that all rotations can be described in the space up to an incorrect scaling along the axis of rotation. Through multiple experiments, we were able to show that this property leads to a huge increase in performance on data that has not been rigidly aligned in a preprocessing step. Where PCA needs nine components to describe rotations, QPCA is shown to only need three. On various datasets, QPCA is shown to improve the reconstruction accuracy achieved by PCA by more than one order of magnitude. Nevertheless, the reconstruction accuracy of QPCA on those datasets is still lower than the reconstruction accuracy achieved by the same method on datasets that are rigidly aligned. Thus, we are not able to completely omit the rigid registration preprocessing step. However, since rigid registration is often no easy task and does not always align the samples perfectly, we do believe that the use of QPCA will be beneficial.

How can the quaternion principal components be computed efficiently?

When applying PCA or QPCA to geometric data, we often see that the number of vertices present in the data meshes is much larger than the number of samples to work with. This makes the computation of the principal components very inefficient, as it requires the solution of a very large eigenvalue problem related to the covariance matrix. For PCA we know that this problem was solved by an algorithm called the method of snapshots, which can compute the same principal components by only solving

the eigenvalue problem related to the much smaller Gram matrix. In this work, we successfully derived a quaternion method of snapshots, which vastly improves the computational efficiency of QPCA on this type of geometric data. Even though this algorithm makes it possible to compute the quaternion principal components in a fraction of the time it would take without this algorithm, we show that QPCA is still much slower than PCA on all datasets considered in this work. The reason for this is that quaternion multiplication is slower than multiplication with real numbers. Furthermore, the quaternion linear algebra toolbox used by QPCA is just not as efficient as the built-in MATLAB functions used by real PCA. Since the subspaces are often constructed in an offline phase, we do not perceive this difference in speed as a large issue.

Can we choose a scalar product such that quaternion PCA becomes robust to mesh irregularities?
It is in fact possible to choose a scalar product that makes QPCA robust against remeshing, coarsening or refining of the meshes. This can be achieved by adding the mass matrix into optimization problem, which is a diagonal matrix that for each vertex stores one third of the combined area of all triangles containing it. The purpose of this matrix is to assign smaller weights to vertices that lie close to each other and larger weights to vertices covering a large area. In this work, we described how this mass-weighted QPCA problem can be solved. Additionally, we introduced a slightly altered version of the quaternion method of snapshots, which greatly improves the computational efficiency of this method. Through quantitative and visual experiments we showed that this mass-weighted QPCA method indeed leads to improved performances on datasets with irregular triangulation. By assigning different weights to each vertex, vertices lying close together are no longer prioritized as was the case with normal QPCA.

8.1.2. Sparse PCA

Can quaternions be combined with SPLOCS in order to construct a subspace that is able to describe a richer space of deformations?
The SPLOCS method aims to find components that are able to approximate the data as good as possible, while also being sparse and localized. By making various small changes to the original SPLOCS method, we showed how the method can be implemented using quaternions. One of these changes was the use of a simplified sparsity inducing term, where QSPLOCS makes it possible to replace the ℓ_1/ℓ_2 group norm by a ℓ_1 norm. Through a number of experiments we evaluated the performance of the QSPLOCS method. On all datasets considered, we showed that QSPLOCS is able to achieve a higher reconstruction accuracy than SPLOCS, while in many cases also being more sparse and localized. Thus, QSPLOCS is able to improve performances on both aspects of the trade-off between accuracy and sparsity. However, this does come at the cost of a larger computation time, since QSPLOCS is shown to be slower on all datasets considered. The visual experiments provided us with many insights on the similarities and differences between the components obtained by both methods. In many cases we found that the deformation described by the real component was also present in the four dimensional space of a quaternion component. These quaternion components were then additionally able to describe three other deformations. Furthermore, we showed that the quaternion components are able to better describe the articulated rotations present in some datasets. Where the SPLOCS components are only able to follow a linear path, resulting in unwanted artifacts, the QSPLOCS components are able to describe some curvilinear paths. However, since many different configurations are possible due to the quaternion weights, this can also introduce new artifacts. Nevertheless, we can still conclude that QSPLOCS is indeed able to describe a richer space of deformations.

Can we replace the spatially varying regularization strengths in SPLOCS with an additional term based on the Laplacian in order to obtain components that are sparse, localized and smooth?
The Laplacian term uses a cotangent weight matrix in order to penalize neighboring vertices in the components that have a large difference in value. This term must be added to the optimization when the spatially varying regularization strengths are removed in order to obtain components that are not only sparse, but also localized. The idea was that without this term, the sparsity inducing term would only enforce sparsity onto the components, without specifying where this sparsity should occur. We showed that this new optimization problem, which we called SPLOCS_lap, can be solved by using a revised version of ADMM that introduces an additional dual variable. Results showed that this method has positive aspects, but also some negative ones. Due to the removal of the spatially varying regularization

strengths, we often observed that some components go to zero everywhere. As a result, we were only able to obtain a certain number of non-zero components on many of the datasets considered. Furthermore, we noticed a huge difference in sparsity and locality between the different components. Where some components did indeed look sparse, localized and smooth, other components were either too sparse or not sparse and local at all. One positive aspect of this method, however, is that the components are indeed more smooth, which can result in the components looking more natural. Some of the components obtained by the original SPLOCS method had very sharp looking boundary edges. These rough edges are smoothed out nicely by SPLOCS_lap. Additionally, the Laplacian term fixes the noise problem that was present in SPLOCS when using a random initialization. Overall, we can say that even though this method shows potential, the performance in its current form is still inadequate. Some of these challenges could potentially be improved by using a different solver.

As a final experiment, we wanted to see whether we could combine the SPLOCS and SPLOCS_lap methods in order to get components that are sparse, localized and smooth using the current ADMM solver. This new sparse PCA technique uses both the spatially varying regularization strengths and the added Laplacian term. Results show that this method is indeed able to generate sparse and localized components, similar to SPLOCS, with added smoothness due to the Laplacian. Thus, this method shows a lot of potential and for now seems to be the preferred choice.

Can this newly constructed sparse PCA technique also be implemented using quaternions?
Similar to how SPLOCS was implemented using quaternions, we also showed how the SPLOCS_lap method can be combined with quaternions in order to obtain QSPLOCS_lap. Experiments showed that this method is again able to achieve both higher reconstruction accuracies and lower sparsity errors than its real counterpart. This, however, comes at the cost of higher computation times. Furthermore, similar properties to SPLOCS_lap are visible, where some components are too sparse and localized, while others are not localized at all. Since each QSPLOCS_lap component describes a larger percentage of the shape variability, we notice that this method is able to obtain less non-zero components than SPLOCS_lap. More components could be obtained by lowering the weight of the sparsity inducing term. This would, however, also give components that are less sparse, thus a trade-off needs to be made. Similar to QSPLOCS, we observed that the method is able to better describe the curvilinear paths present in some datasets. The SPLOCS_lap components, on the other hand, are only able to follow a linear path.

We additionally constructed a new quaternion sparse PCA technique that uses both the spatially varying regularization strengths and the Laplacian term. This method is able to construct quaternion components that are sparse, localized and smooth. Due to the drawbacks of QSPLOCS_lap in its current form, this combined method seems to be the preferred choice.

8.2. Future work

Through various experiments we have shown that it can be very beneficial to use quaternion PCA and sparse PCA in a geometry processing application. The methods are able to describe a richer space of deformations using less components and can solve some of the problems related to real PCA and sparse PCA. The x, y and z coordinates of the vertices are now treated as a unit and the quaternion components are able to better describe the rotations, both global and local, that are present in the datasets.

This work focused specifically on quaternion PCA and sparse PCA, showing their benefits on a variety of different datasets for generality. For future research, we are interested to see how these methods could be applied in various computer graphics and geometry processing applications. One example topic that could be of interest is non-rigid registration, where one applies QPCA to a template model and uses this to fit the template to for instance a stream of depth images or a set of ct scans. Other topics of interest could be deformation-based modeling or shape interpolation. The former example provides the user with a set of tools for modifying a geometry object, which can be especially useful using quaternion sparse PCA. This makes it possible for the user to modify small parts of the mesh without affecting the whole mesh.

Some applications might require fast computation times for constructing the subspaces. Experiments showed that quaternion PCA and sparse PCA are unfortunately not as computationally efficient

as their real counterparts. This is partly due to the quaternion linear algebra package that is used, called `qtfm` [46]. This package is simply not as optimized as the built-in real linear algebra package. Thus, to make these quaternion methods more effective, one could look for a more efficient quaternion linear algebra toolbox.

Through the experiments performed in Chapter 7, we showed that the ADMM optimizer is in many cases not able to reach a good solution. The optimizer does not progress much from its initial values, making the methods very dependent on their initialization phase. Furthermore, we showed that for SPLOCS_lap and QSPLOCS_lap this can lead to components that are not localized at all. Thus, for future work one could look at whether it is possible to implement these methods using another solver, which uses second order information. During this research project, we made an attempt to solve this using quadratic programs with inequality constraints, similar to what was done in [48]. This, however, turned out to be very difficult to solve due to the ℓ_1/ℓ_2 group norm used in the methods as well as the added Laplacian term. Thus, further research is needed in this direction.

Finally, one could investigate whether the mass matrix can also be combined with the sparse PCA and quaternion sparse PCA methods similar to what we showed for QPCA. This would make the methods more robust against meshes with irregular triangulation. Furthermore, this could potentially simplify the selection of the tunable parameters.

A

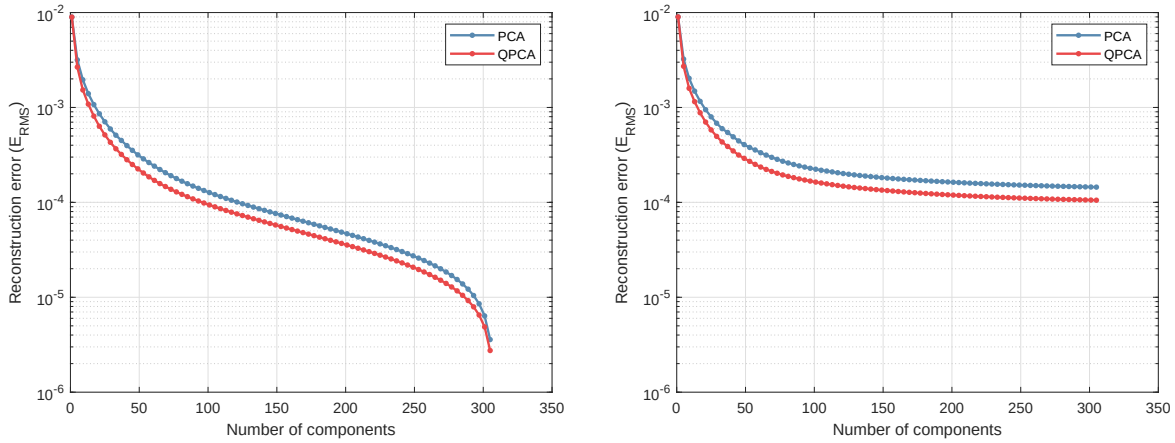
Additional results PCA

In this appendix, we perform some of the experiments from Chapter 6 on the following datasets:

- “face” from [66].
- “Sign D poses” from TCD Hands [20].
- “E5 - Hook left poses” from ACCAD [54].
- “jumping-jacks-50022” from DFAUST [7].
- “jiggle-on-toes-50004” from DFAUST [7].

A.1. Experiments on rigidly aligned datasets

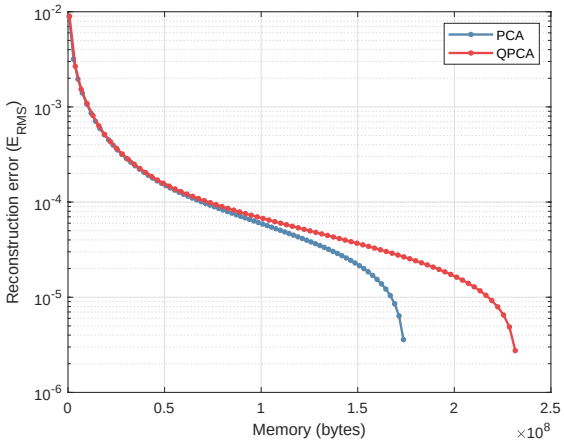
A.1.1. Quantitative evaluation - Reconstruction accuracy



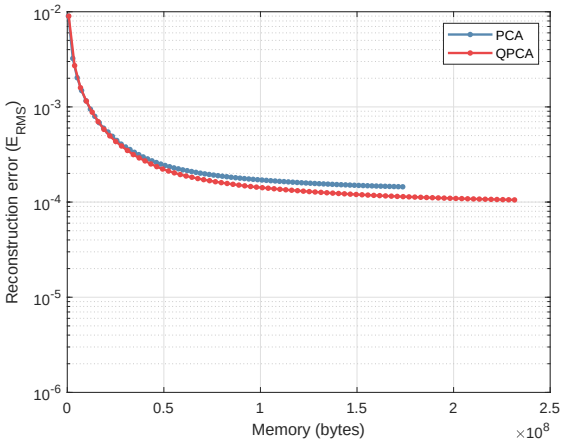
(a) E_{RMS} error computed on 308 training frames.

(b) E_{RMS} error computed on 77 (unseen) test frames.

Figure A.1: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the “face” motion capture from [66]. 5-fold cross validation is used.

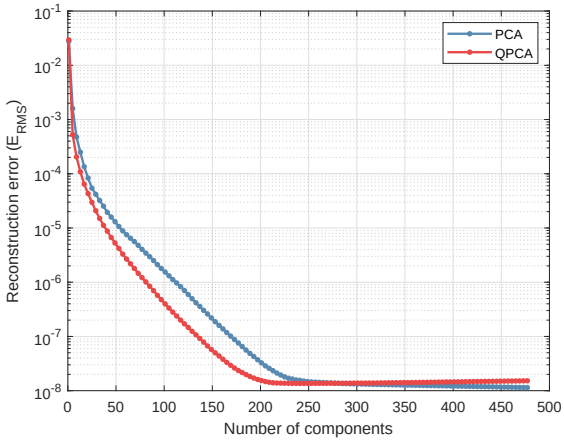


(a) E_{RMS} error computed on 308 training frames.

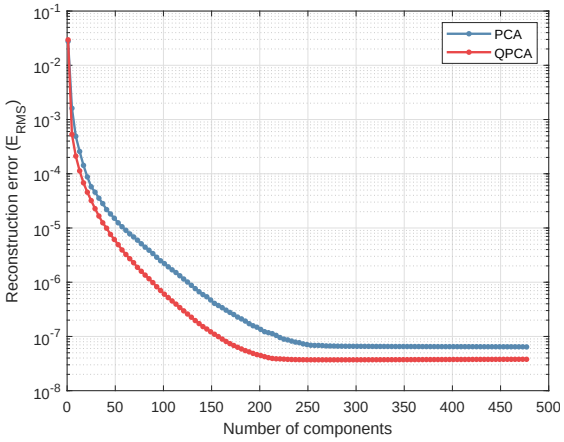


(b) E_{RMS} error computed on 77 (unseen) test frames.

Figure A.2: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "face" motion capture from [66]. 5-fold cross validation is used.

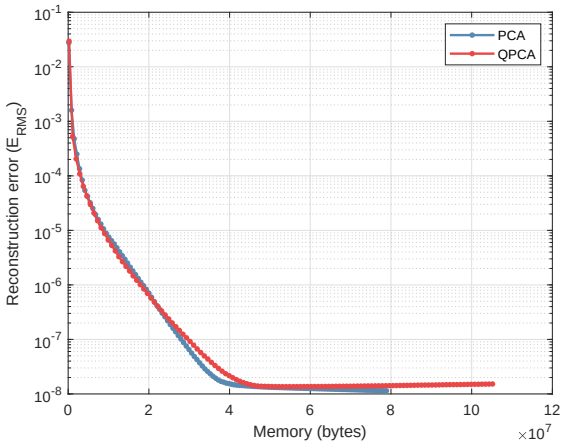


(a) E_{RMS} error computed on 481 training frames.

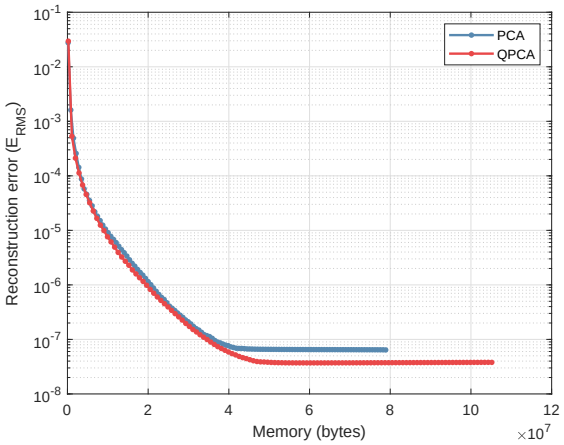


(b) E_{RMS} error computed on 120 (unseen) test frames.

Figure A.3: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the "Sign D poses" motion capture from TCD Hands [44]. 5-fold cross validation is used.

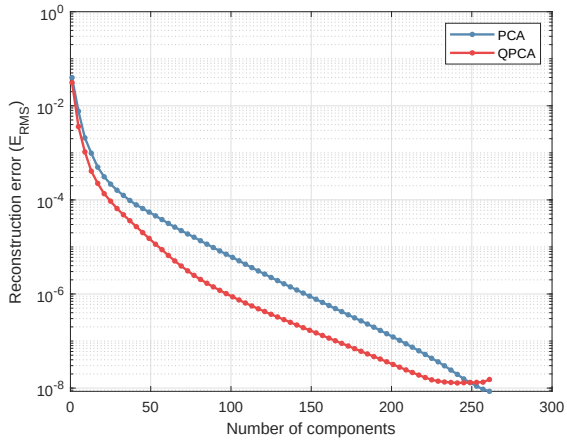


(a) E_{RMS} error computed on 481 training frames.

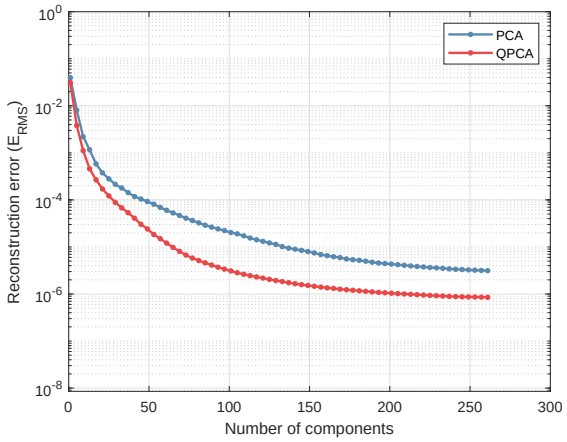


(b) E_{RMS} error computed on 120 (unseen) test frames.

Figure A.4: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "Sign D poses" motion capture from TCD Hands [44]. 5-fold cross validation is used.

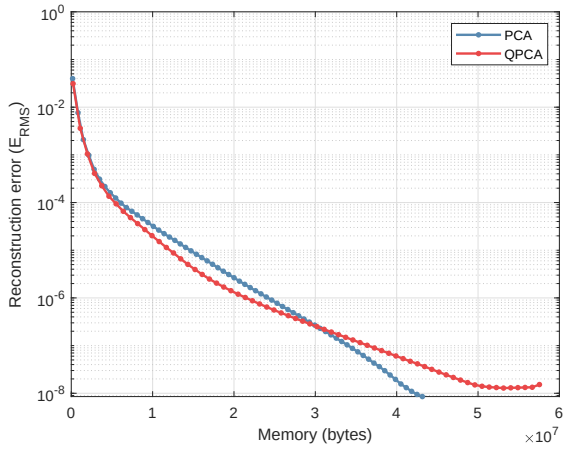


(a) E_{RMS} error computed on 262 training frames.

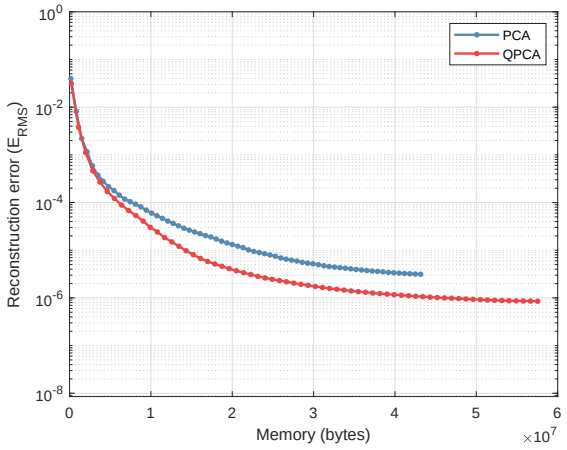


(b) E_{RMS} error computed on 65 (unseen) test frames.

Figure A.5: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the "E5 - Hook left poses" motion capture from ACCAD [54]. 5-fold cross validation is used.

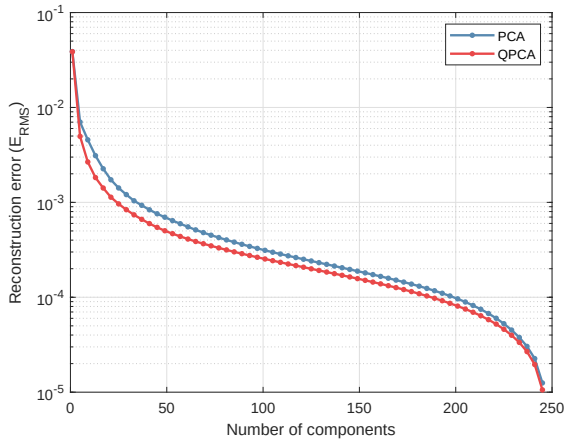


(a) E_{RMS} error computed on 262 training frames.

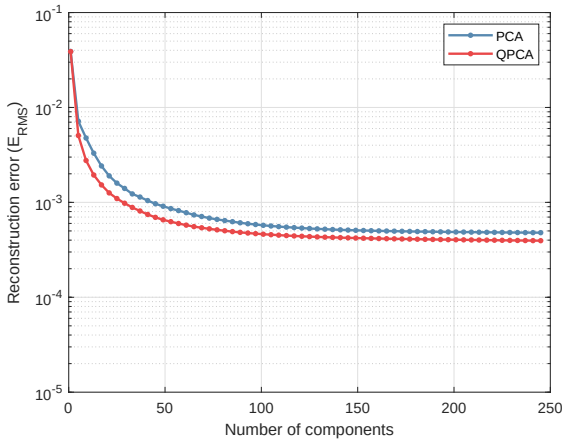


(b) E_{RMS} error computed on 65 (unseen) test frames.

Figure A.6: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "E5 - Hook left poses" motion capture from ACCAD [54]. 5-fold cross validation is used.

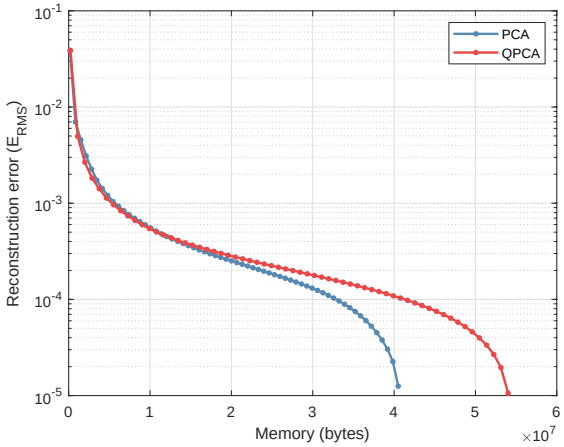


(a) E_{RMS} error computed on 247 training frames.

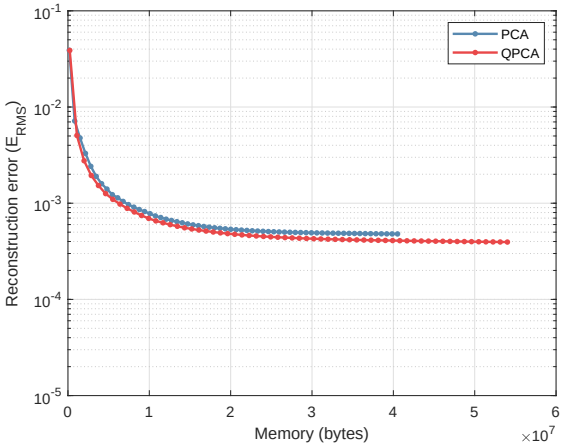


(b) E_{RMS} error computed on 61 (unseen) test frames.

Figure A.7: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the "jumping-jacks-50022" motion capture from DFAUST [7]. 5-fold cross validation is used.

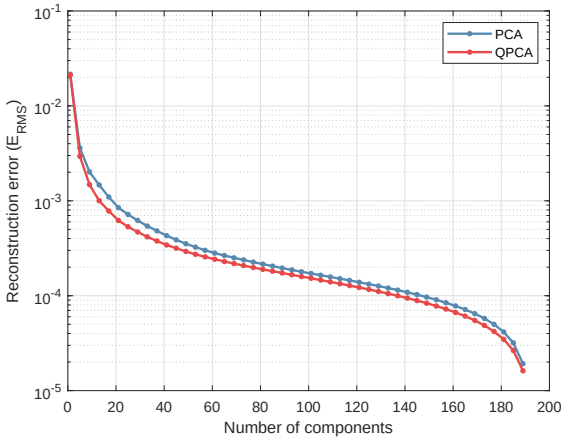


(a) E_{RMS} error computed on 247 training frames.

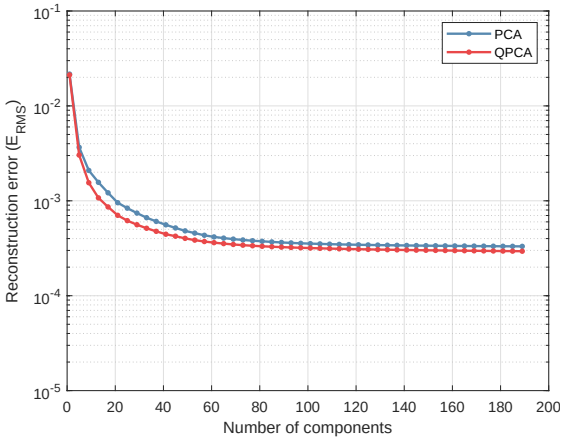


(b) E_{RMS} error computed on 61 (unseen) test frames.

Figure A.8: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "jumping-jacks-50022" motion capture from DFAUST [7]. 5-fold cross validation is used.

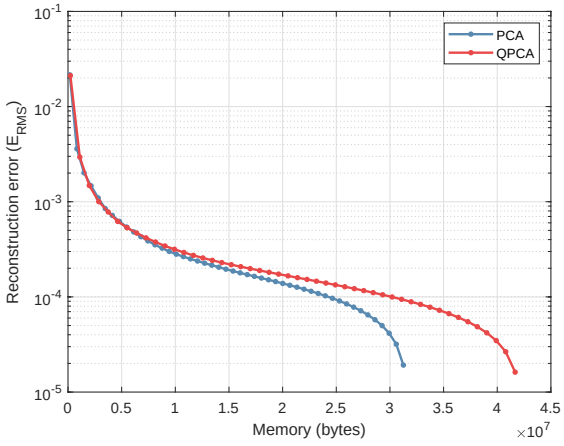


(a) E_{RMS} error computed on 192 training frames.

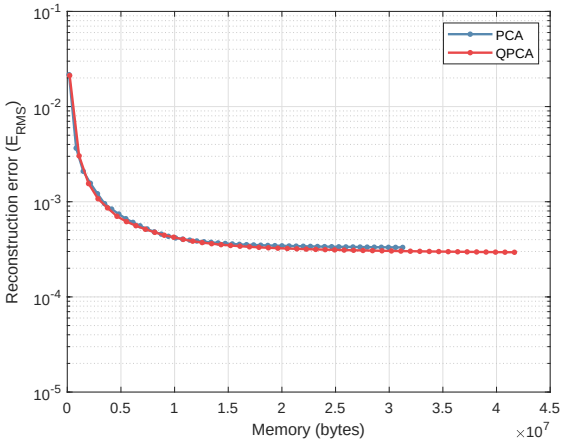


(b) E_{RMS} error computed on 47 (unseen) test frames.

Figure A.9: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. 5-fold cross validation is used.



(a) E_{RMS} error computed on 192 training frames.



(b) E_{RMS} error computed on 47 (unseen) test frames.

Figure A.10: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. 5-fold cross validation is used.

A.1.2. Quantitative evaluation - Variance distribution

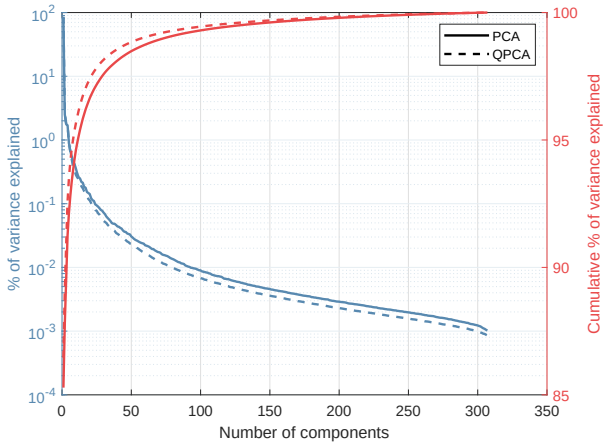


Figure A.11: (Cumulative) percentage of variance explained per component, computed on the “face” motion capture from [66].

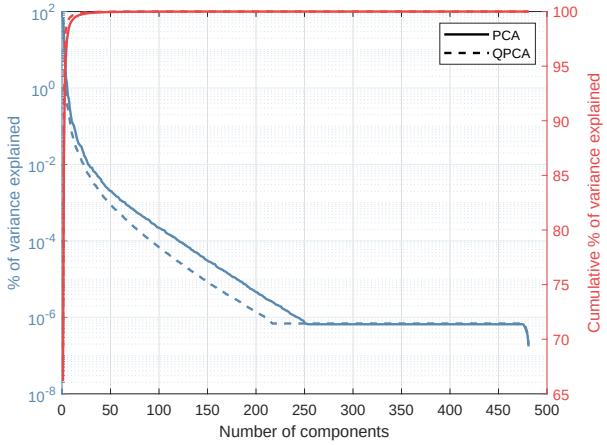


Figure A.12: (Cumulative) percentage of variance explained per component, computed on the “Sign D poses” motion capture from TCD Hands [44].

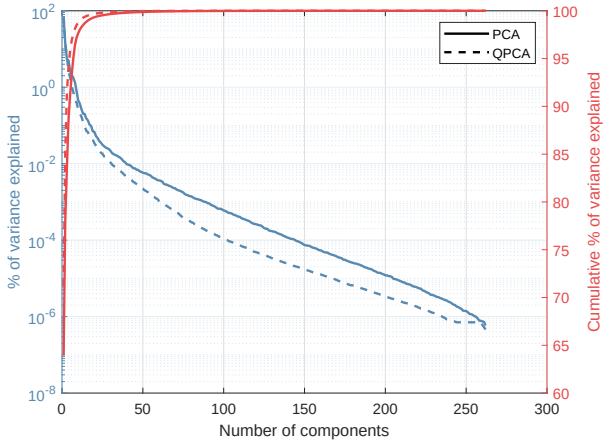


Figure A.13: (Cumulative) percentage of variance explained per component, computed on the “E5 - Hook left poses” motion capture from ACCAD [54].

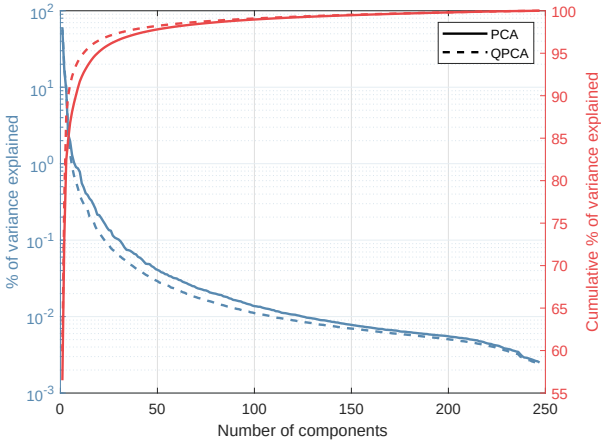


Figure A.14: (Cumulative) percentage of variance explained per component, computed on the “jumping-jacks-50022” motion capture from DFAUST [7].

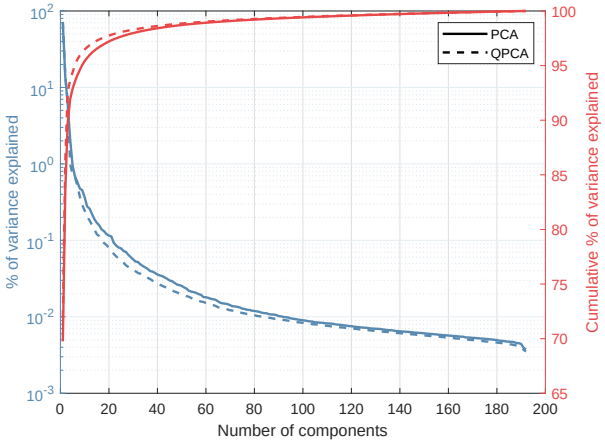


Figure A.15: (Cumulative) percentage of variance explained per component, computed on the “jiggle-on-toes-50004” motion capture from DFAUST [7].

A.1.3. Visual comparison

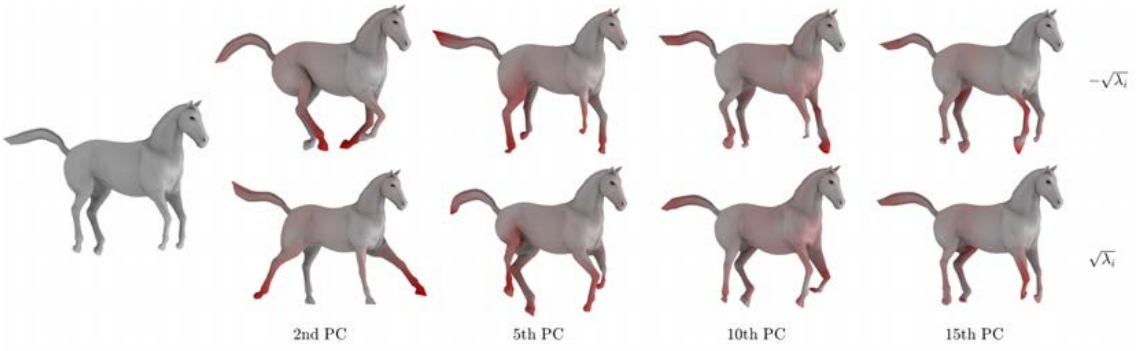


Figure A.16: Visualization of PCA: 4 modes of deformation on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-\sqrt{\lambda_i}$ (top) and $\sqrt{\lambda_i}$ (bottom) along the ith principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

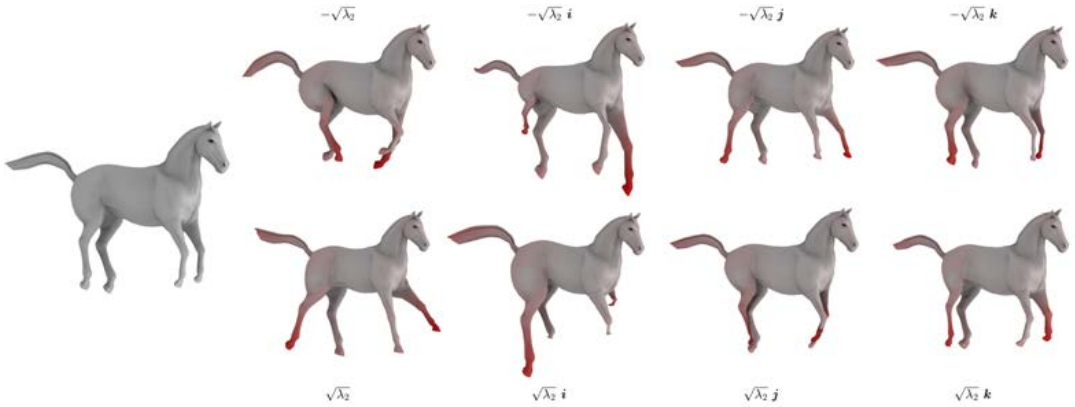


Figure A.17: Visualization of QPCA: The 2nd quaternion mode of deformation on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-\sqrt{\lambda_2}$ (top) and $\sqrt{\lambda_2}$ (bottom) along the four dimensions of the 2nd principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

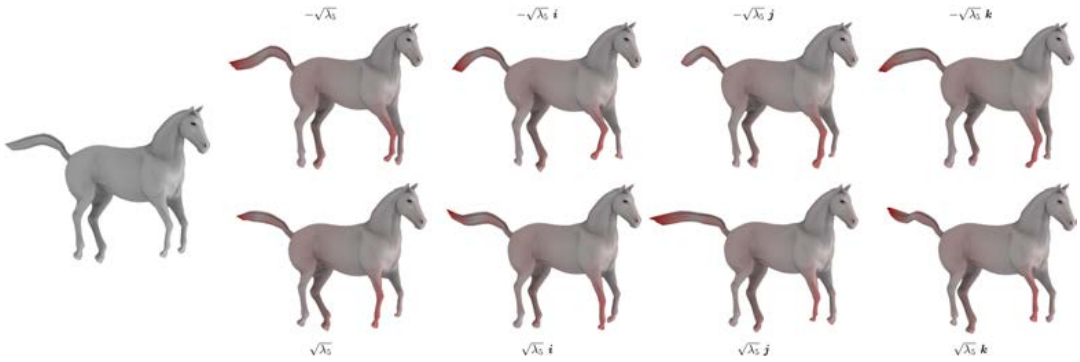


Figure A.18: Visualization of QPCA: The 5th quaternion mode of deformation on the “horse-gallop” motion capture from the animal dataset [52]. Models corresponding to $-\sqrt{\lambda_5}$ (top) and $\sqrt{\lambda_5}$ (bottom) along the four dimensions of the 5th principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

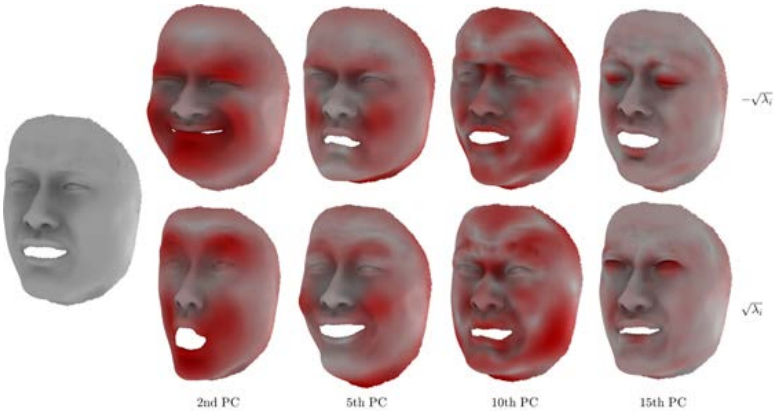


Figure A.19: Visualization of PCA: 4 modes of deformation on the “face” motion capture from [66]. Models corresponding to $-\sqrt{\lambda_i}$ (top) and $\sqrt{\lambda_i}$ (bottom) along the ith principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

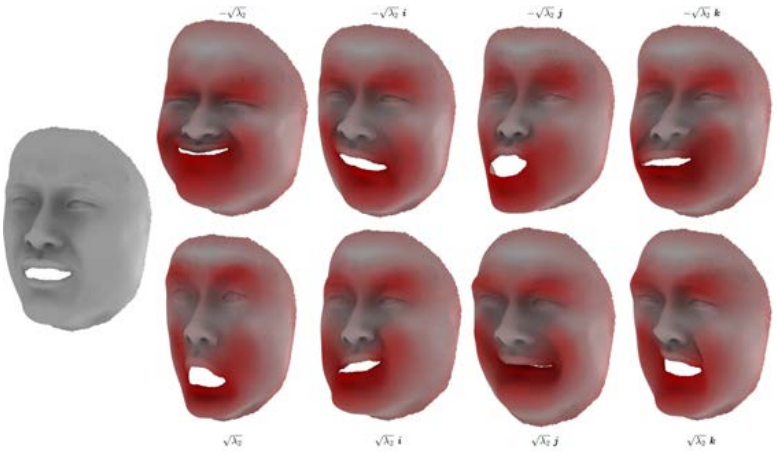


Figure A.20: Visualization of QPCA: The 2nd quaternion mode of deformation on the “face” motion capture from [66]. Models corresponding to $-\sqrt{\lambda_2}$ (top) and $\sqrt{\lambda_2}$ (bottom) along the four dimensions of the 2nd principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

Figure A.21: Visualization of QPCA: The 5th quaternion mode of deformation on the “face” motion capture from [66]. Models corresponding to $-\sqrt{\lambda_5}$ (top) and $\sqrt{\lambda_5}$ (bottom) along the four dimensions of the 5th principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

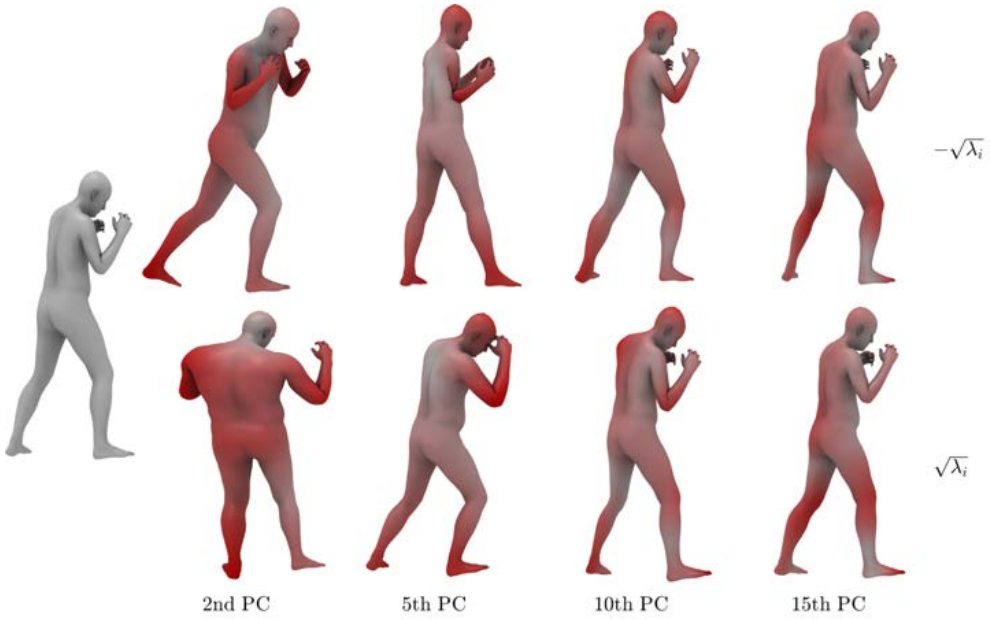


Figure A.22: Visualization of PCA: 4 modes of deformation on the “E5 - Hook left poses” motion capture from ACCAD [54]. Models corresponding to $-\sqrt{\lambda_i}$ (top) and $\sqrt{\lambda_i}$ (bottom) along the ith principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

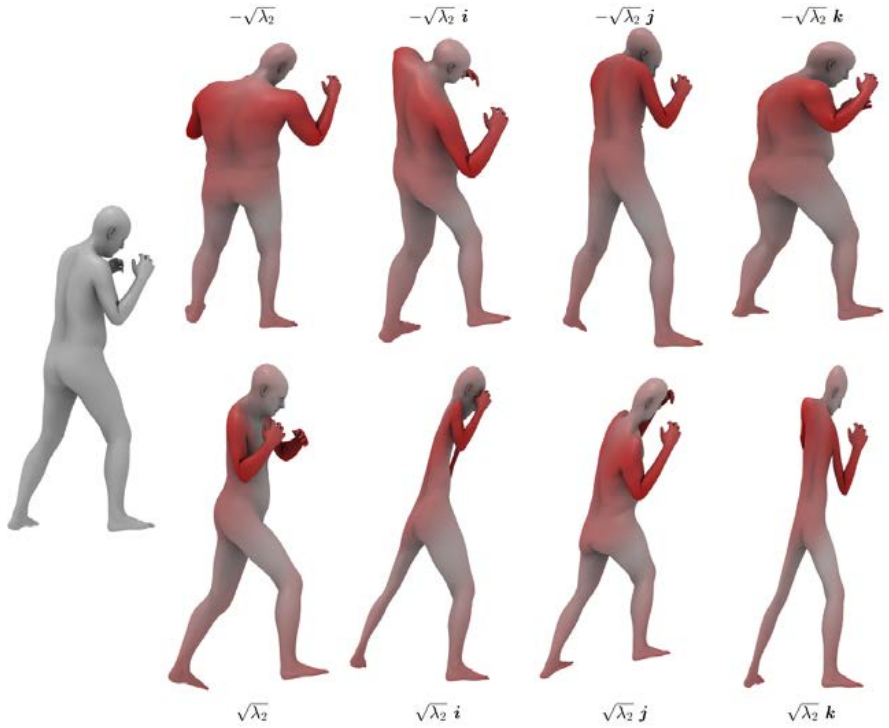


Figure A.23: Visualization of QPCA: The 2nd quaternion mode of deformation on the “E5 - Hook left poses” motion capture from ACCAD [54]. Models corresponding to $-\sqrt{\lambda_2}$ (top) and $\sqrt{\lambda_2}$ (bottom) along the four dimensions of the 2nd principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

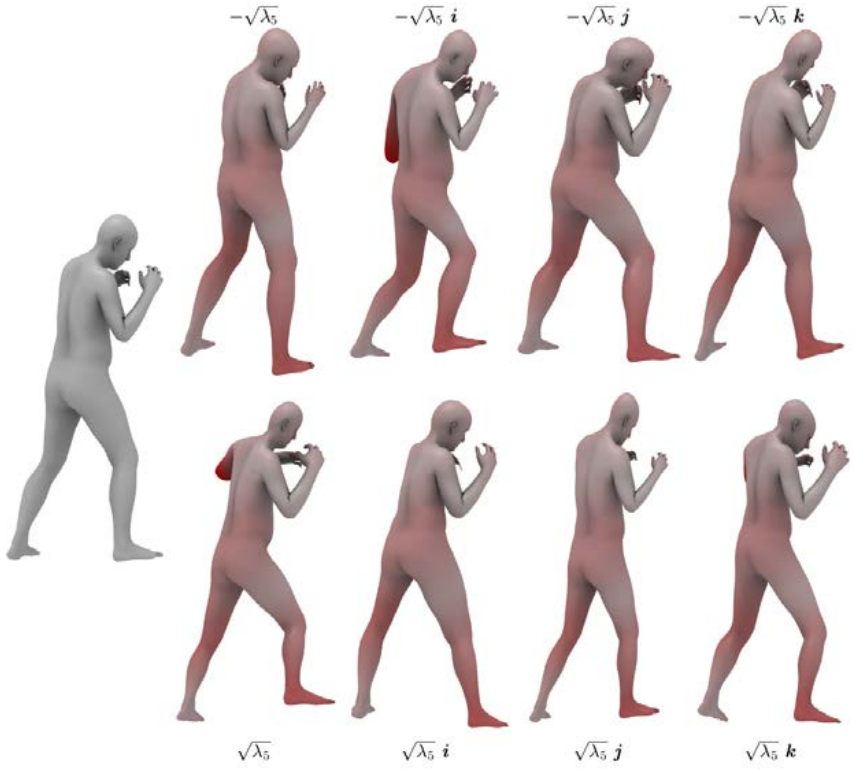


Figure A.24: Visualization of QPCA: The 5th quaternion mode of deformation on the “E5 - Hook left poses” motion capture from ACCAD [54]. Models corresponding to $-\sqrt{\lambda_5}$ (top) and $\sqrt{\lambda_5}$ (bottom) along the four dimensions of the 5th principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

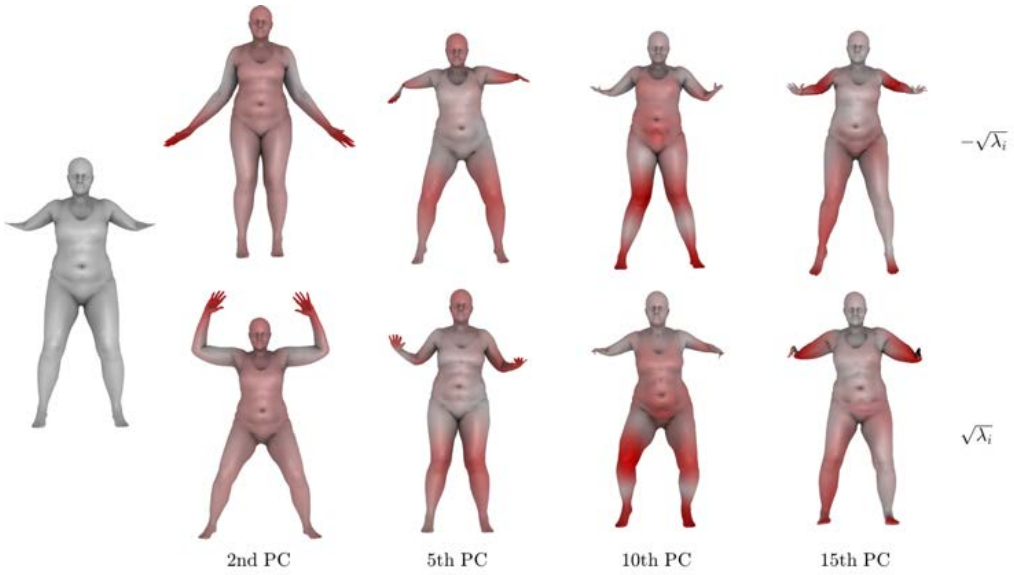


Figure A.25: Visualization of PCA: 4 modes of deformation on the “jumping-jacks-50022” motion capture from DFAUST [7]. Models corresponding to $-\sqrt{\lambda_i}$ (top) and $\sqrt{\lambda_i}$ (bottom) along the ith principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

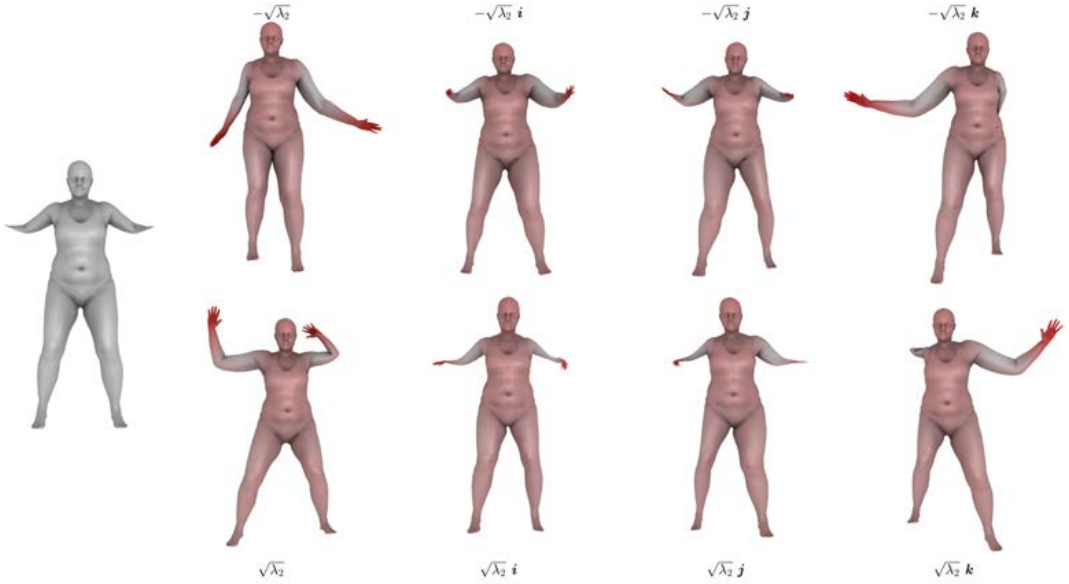


Figure A.26: Visualization of QPCA: The 2nd quaternion mode of deformation on the “jumping-jacks-50022” motion capture from DFAUST [7]. Models corresponding to $-\sqrt{\lambda_2}$ (top) and $\sqrt{\lambda_2}$ (bottom) along the four dimensions of the 2nd principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

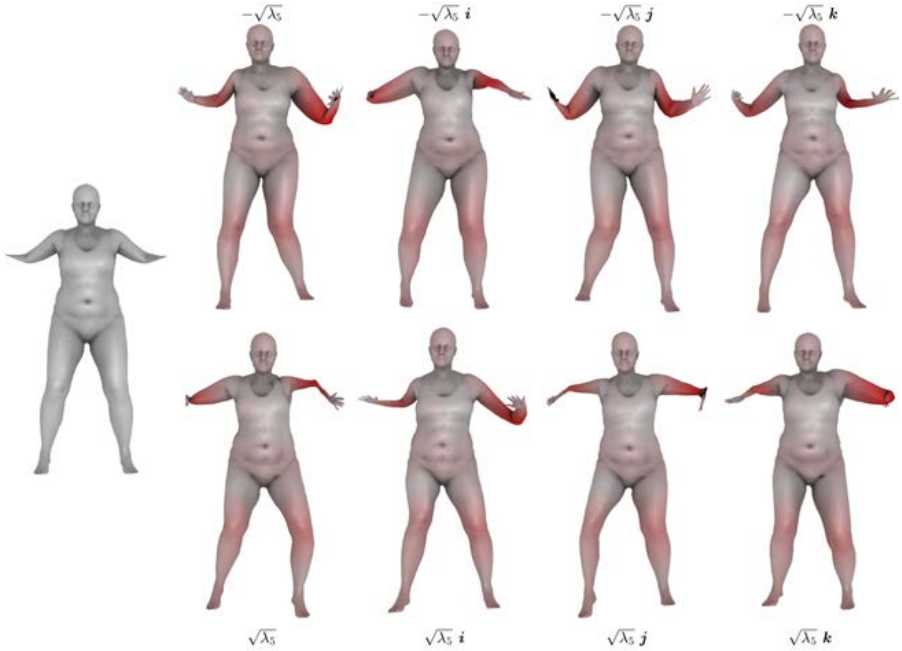


Figure A.27: Visualization of QPCA: The 5th quaternion mode of deformation on the “jumping-jacks-50022” motion capture from DFAUST [7]. Models corresponding to $-\sqrt{\lambda_5}$ (top) and $\sqrt{\lambda_5}$ (bottom) along the four dimensions of the 5th principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

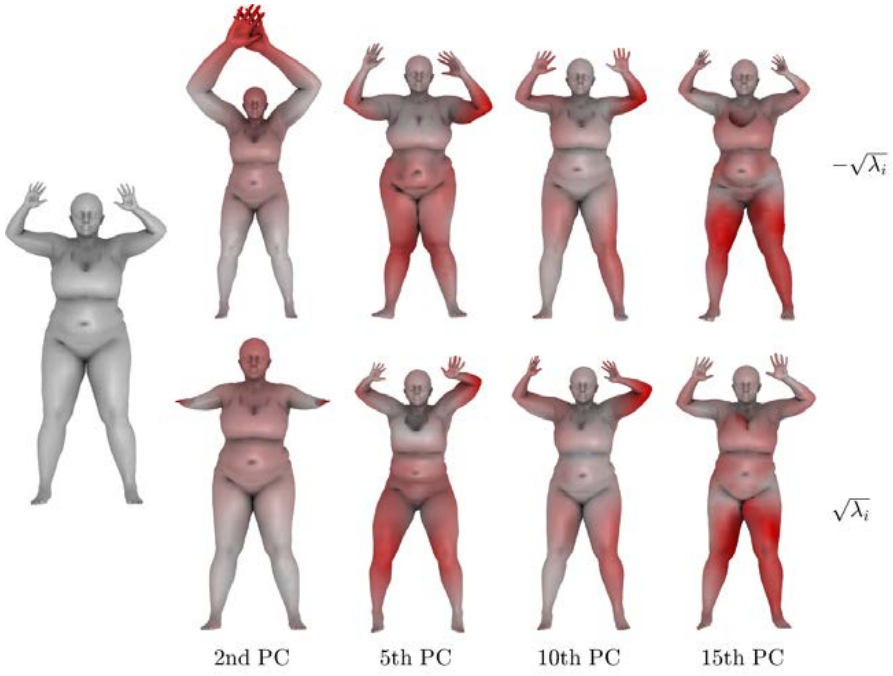


Figure A.28: Visualization of PCA: 4 modes of deformation on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. Models corresponding to $-\sqrt{\lambda_i}$ (top) and $\sqrt{\lambda_i}$ (bottom) along the ith principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

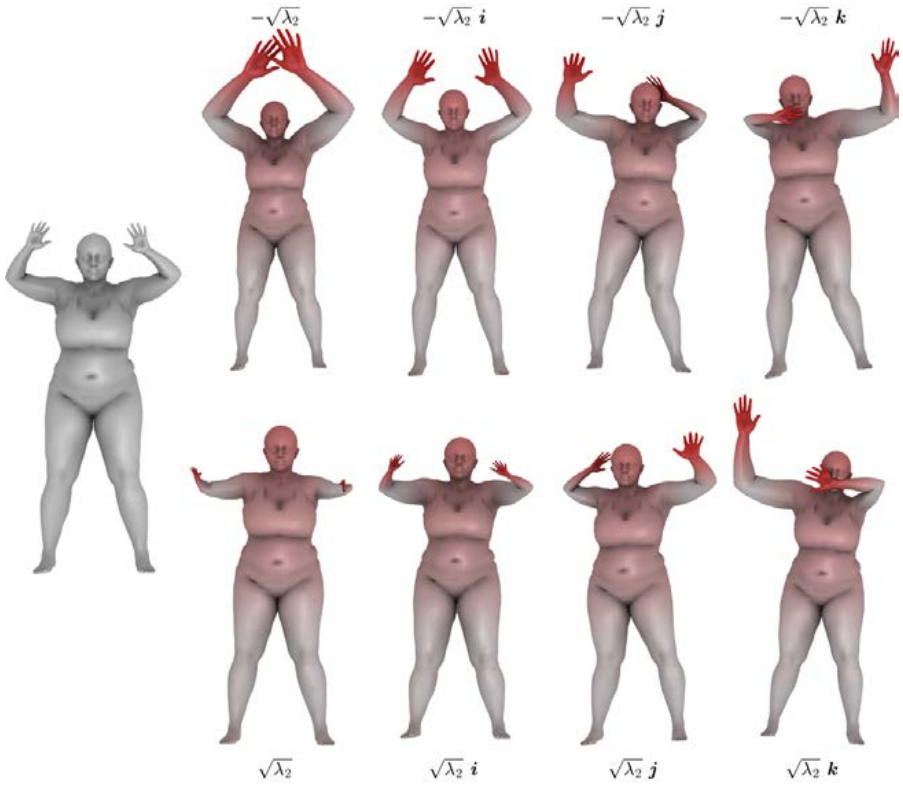


Figure A.29: Visualization of QPCA: The 2nd quaternion mode of deformation on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. Models corresponding to $-\sqrt{\lambda_2}$ (top) and $\sqrt{\lambda_2}$ (bottom) along the four dimensions of the 2nd principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

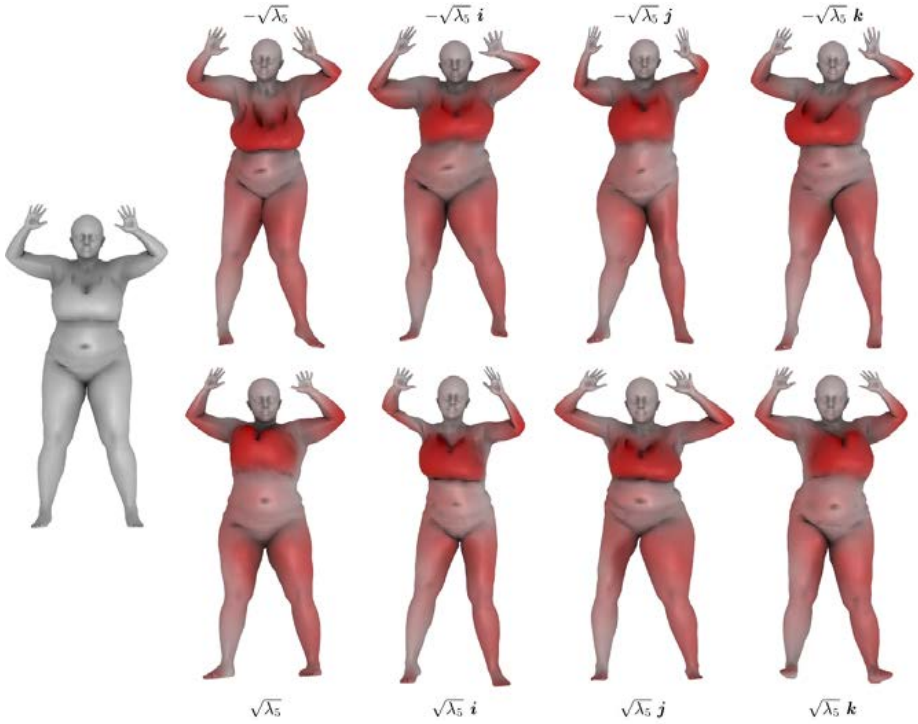
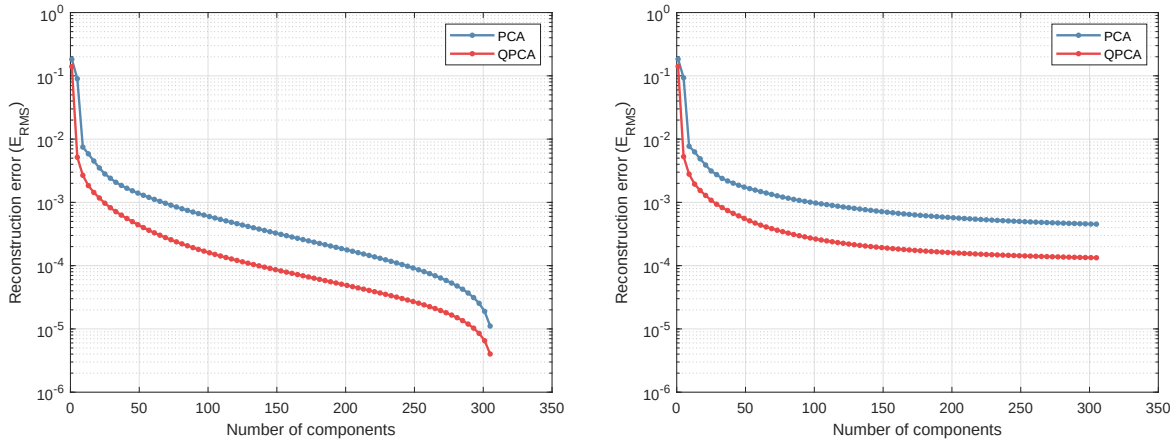


Figure A.30: Visualization of QPCA: The 5th quaternion mode of deformation on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. Models corresponding to $-\sqrt{\lambda_5}$ (top) and $\sqrt{\lambda_5}$ (bottom) along the four dimensions of the 5th principal component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

A.2. Experiments on non-rigidly aligned datasets

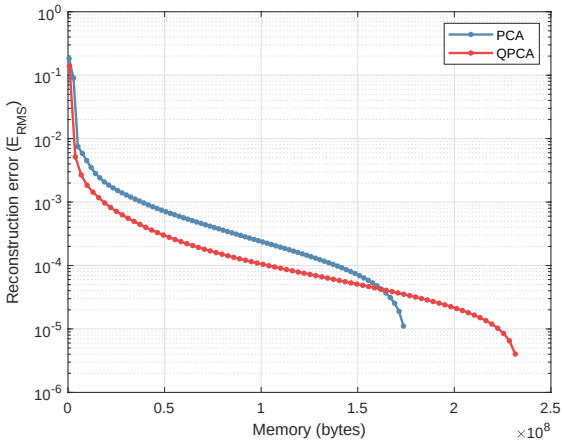
A.2.1. Quantitative evaluation - Reconstruction accuracy



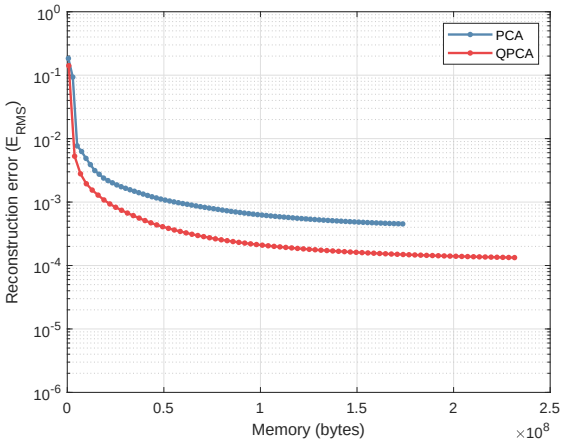
(a) E_{RMS} error computed on 308 training frames.

(b) E_{RMS} error computed on 77 (unseen) test frames.

Figure A.31: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the "face" motion capture from [66], where a random rigid motion is applied to each frame. 5-fold cross validation is used.

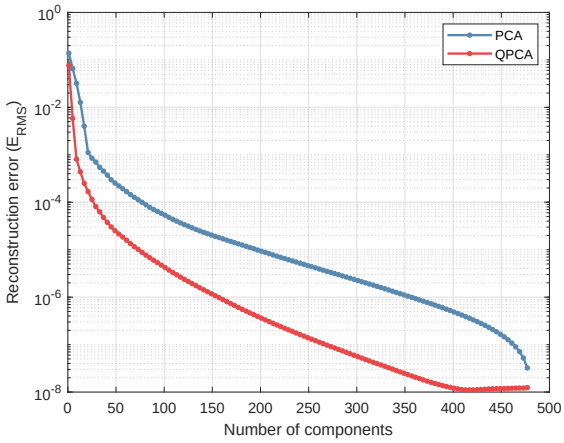


(a) E_{RMS} error computed on 308 training frames.

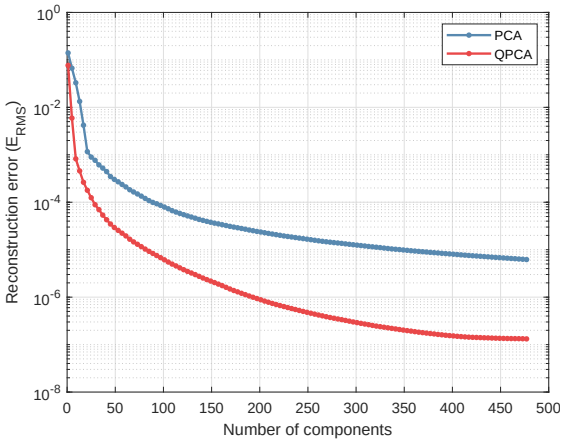


(b) E_{RMS} error computed on 77 (unseen) test frames.

Figure A.32: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "face" motion capture from [66], where a random rigid motion is applied to each frame. 5-fold cross validation is used.

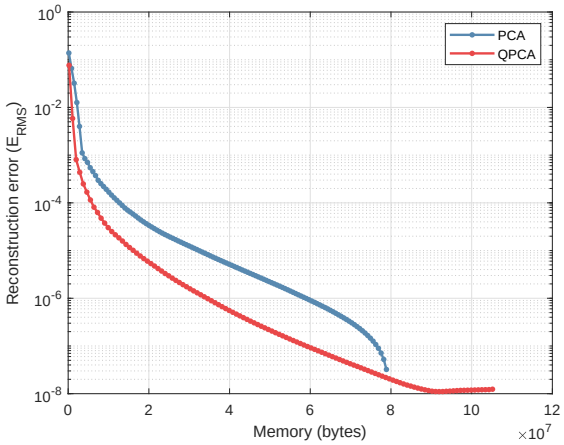


(a) E_{RMS} error computed on 481 training frames.

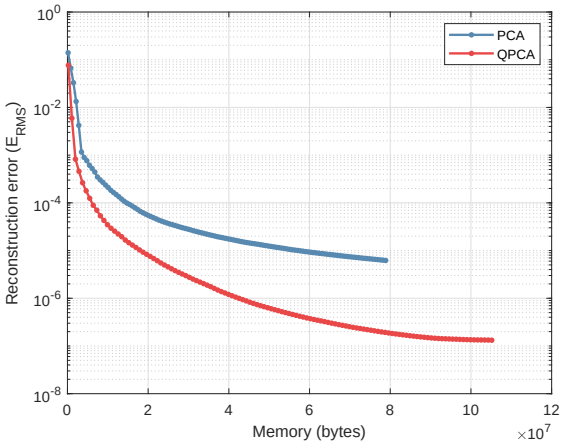


(b) E_{RMS} error computed on 120 (unseen) test frames.

Figure A.33: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the "Sign D poses" motion capture from TCD Hands [44], where a random rigid motion is applied to each frame. 5-fold cross validation is used.

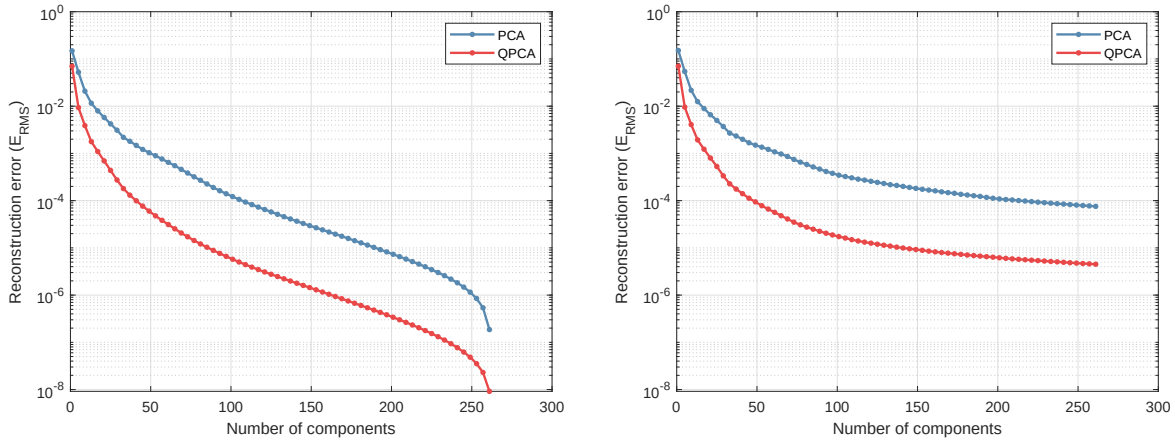


(a) E_{RMS} error computed on 481 training frames.



(b) E_{RMS} error computed on 120 (unseen) test frames.

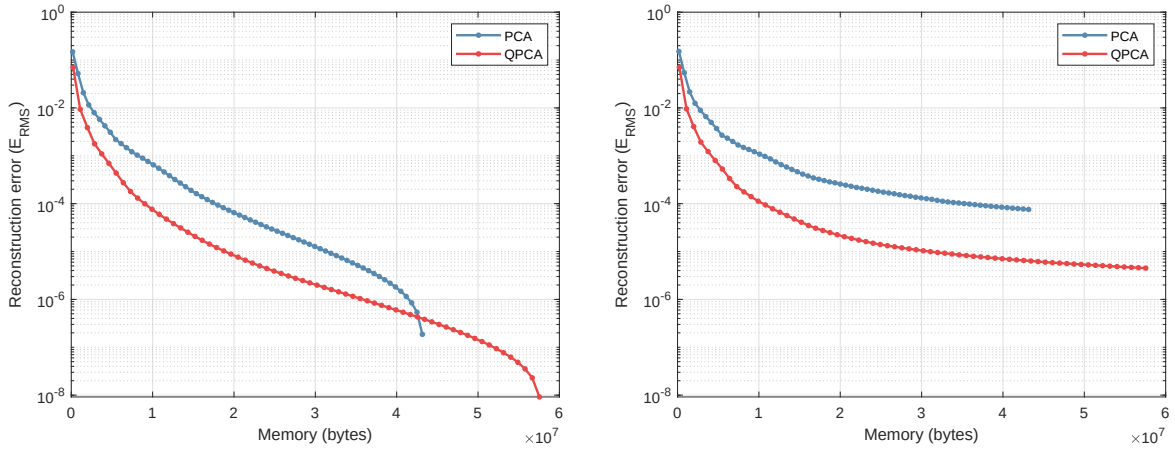
Figure A.34: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "Sign D poses" motion capture from TCD Hands [44], where a random rigid motion is applied to each frame. 5-fold cross validation is used.



(a) E_{RMS} error computed on 262 training frames.

(b) E_{RMS} error computed on 65 (unseen) test frames.

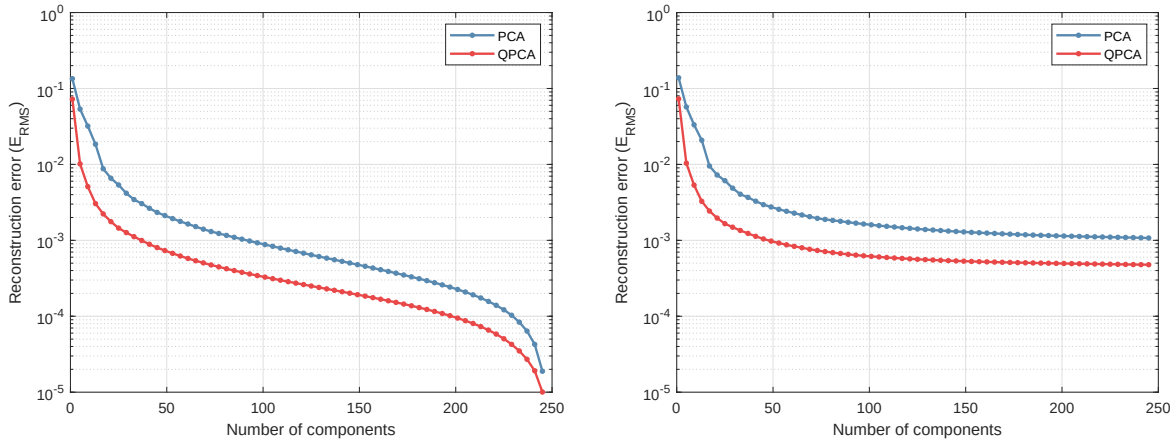
Figure A.35: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the "E5 - Hook left poses" motion capture from ACCAD [54], where a random rigid motion is applied to each frame. 5-fold cross validation is used.



(a) E_{RMS} error computed on 262 training frames.

(b) E_{RMS} error computed on 65 (unseen) test frames.

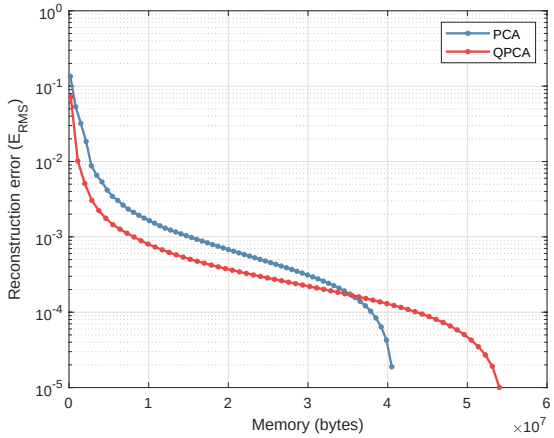
Figure A.36: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "E5 - Hook left poses" motion capture from ACCAD [54], where a random rigid motion is applied to each frame. 5-fold cross validation is used.



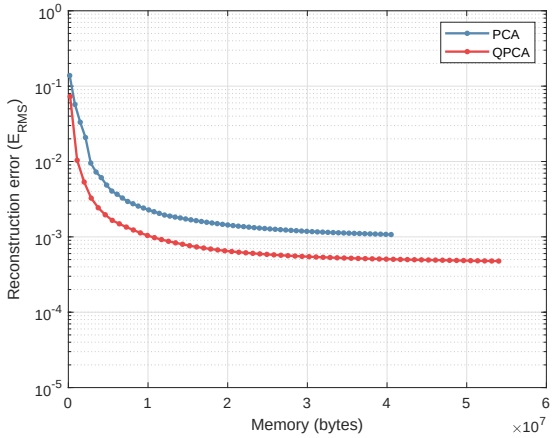
(a) E_{RMS} error computed on 247 training frames.

(b) E_{RMS} error computed on 61 (unseen) test frames.

Figure A.37: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the "jumping-jacks-50022" motion capture from DFAUST [7], where a random rigid motion is applied to each frame. 5-fold cross validation is used.

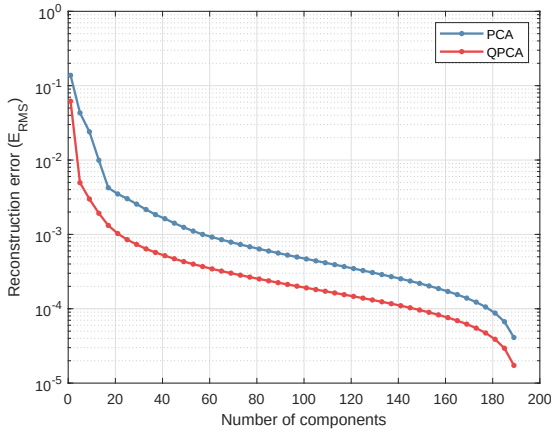


(a) E_{RMS} error computed on 247 training frames.

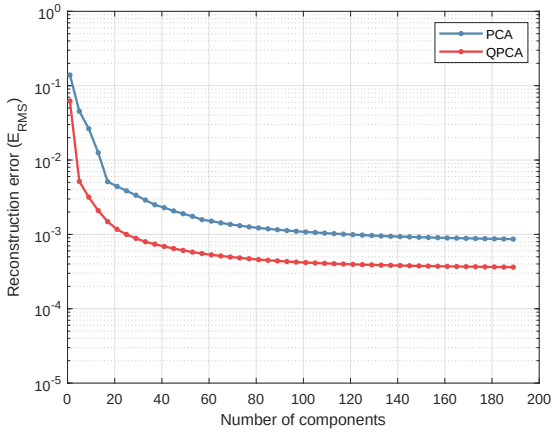


(b) E_{RMS} error computed on 61 (unseen) test frames.

Figure A.38: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "jumping-jacks-50022" motion capture from DFAUST [7], where a random rigid motion is applied to each frame. 5-fold cross validation is used.

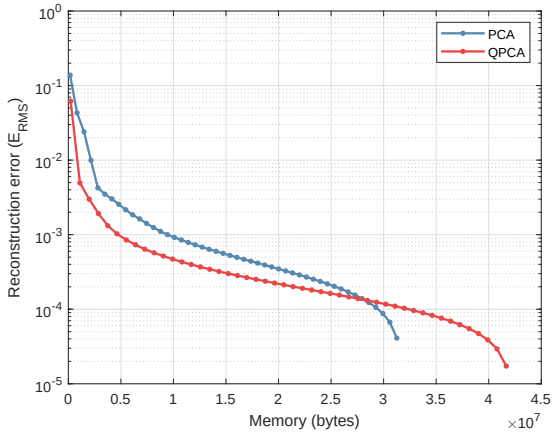


(a) E_{RMS} error computed on 192 training frames.

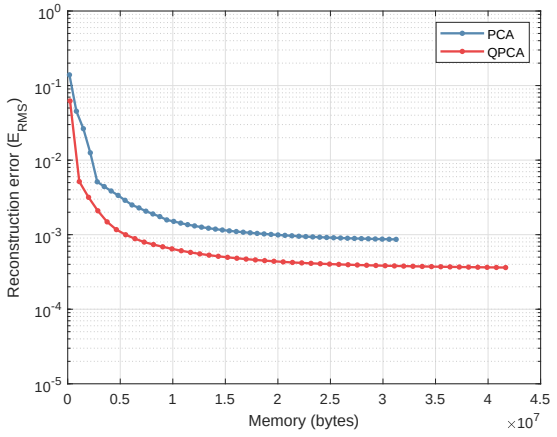


(b) E_{RMS} error computed on 47 (unseen) test frames.

Figure A.39: Reconstruction error (y-axis) with respect to the number of components used (x-axis) on the "jiggle-on-toes-50004" motion capture from DFAUST [7], where a random rigid motion is applied to each frame. 5-fold cross validation is used.



(a) E_{RMS} error computed on 192 training frames.



(b) E_{RMS} error computed on 47 (unseen) test frames.

Figure A.40: Reconstruction error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "jiggle-on-toes-50004" motion capture from DFAUST [7], where a random rigid motion is applied to each frame. 5-fold cross validation is used.

A.2.2. Quantitative evaluation - Variance distribution

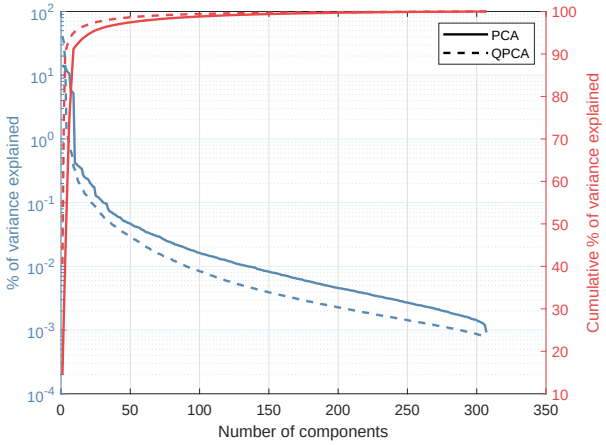


Figure A.41: (Cumulative) percentage of variance explained per component, computed on the “face” motion capture from [66]. A random rigid motion is applied to each sample of the dataset.

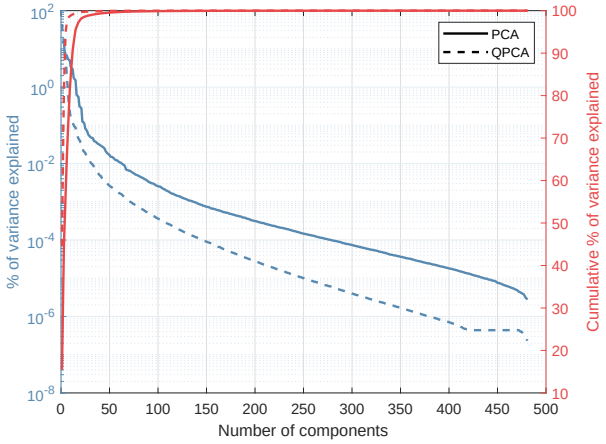


Figure A.42: (Cumulative) percentage of variance explained per component, computed on the “Sign D poses” motion capture from TCD Hands [44]. A random rigid motion is applied to each sample of the dataset.

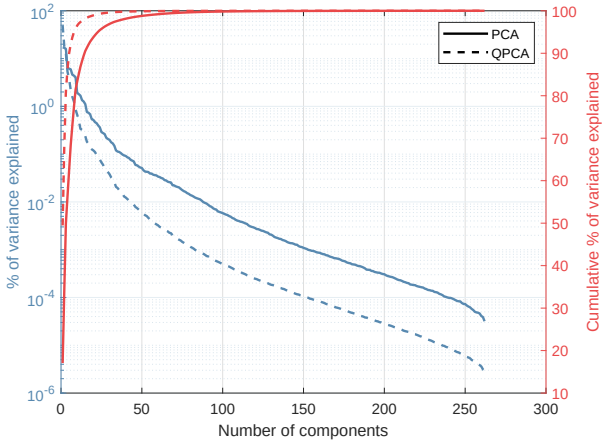


Figure A.43: (Cumulative) percentage of variance explained per component, computed on the “E5 - Hook left poses” motion capture from ACCAD [54]. A random rigid motion is applied to each sample of the dataset.

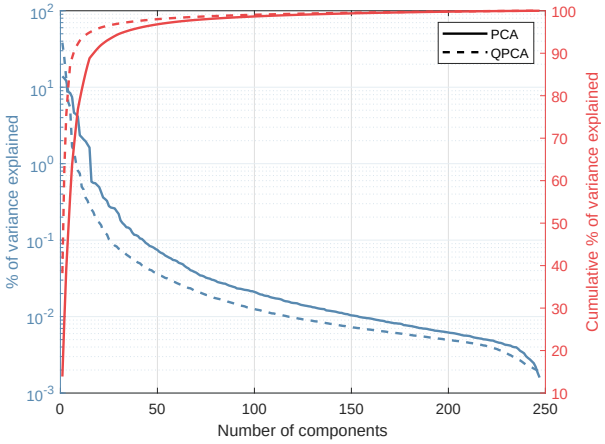


Figure A.44: (Cumulative) percentage of variance explained per component, computed on the “jumping-jacks-50022” motion capture from DFAUST [7]. A random rigid motion is applied to each sample of the dataset.

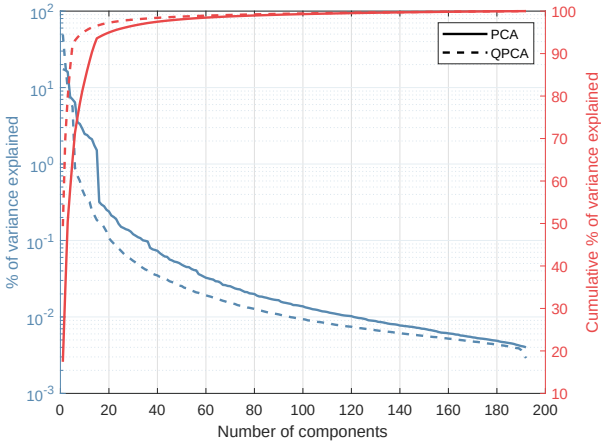


Figure A.45: (Cumulative) percentage of variance explained per component, computed on the “jiggle-on-toes-50004” motion capture from DFAUST [7]. A random rigid motion is applied to each sample of the dataset.

A.2.3. Visual comparison

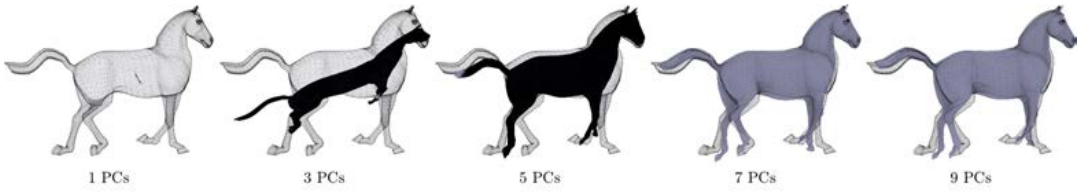


Figure A.46: Visualization of PCA: Projection of one sample of the “horse-gallop” motion capture from the animal dataset [52] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

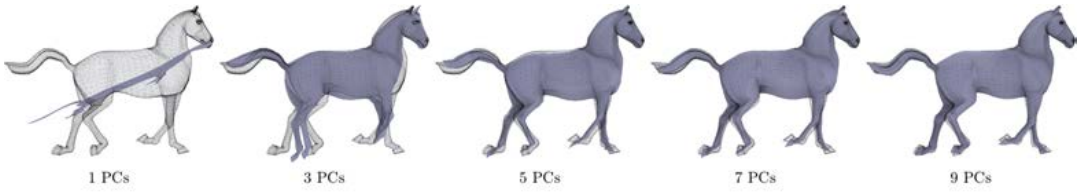


Figure A.47: Visualization of QPCA: Projection of one sample of the “horse-gallop” motion capture from the animal dataset [52] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

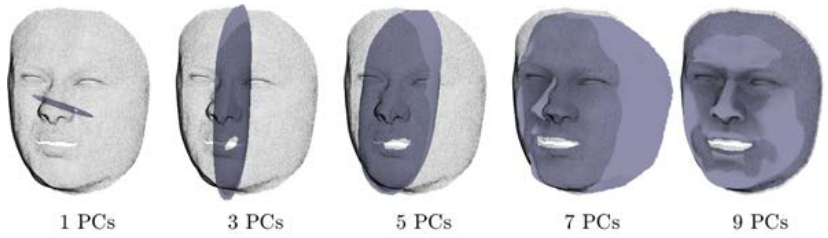


Figure A.48: Visualization of PCA: Projection of one sample of the “face” motion capture from [66] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

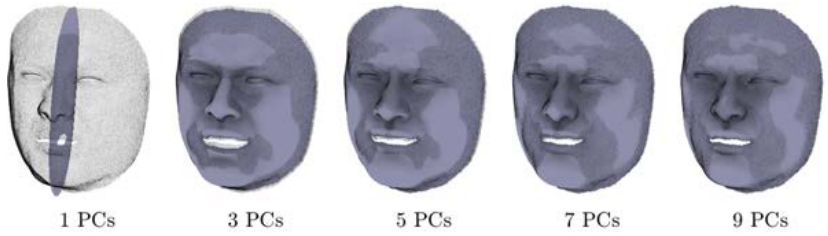


Figure A.49: Visualization of QPCA: Projection of one sample of the “face” motion capture from [66] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

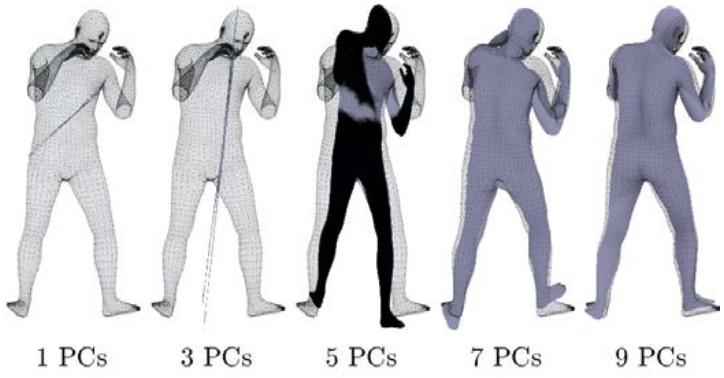


Figure A.50: Visualization of PCA: Projection of one sample of the “E5 - Hook left poses” motion capture from ACCAD [54] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

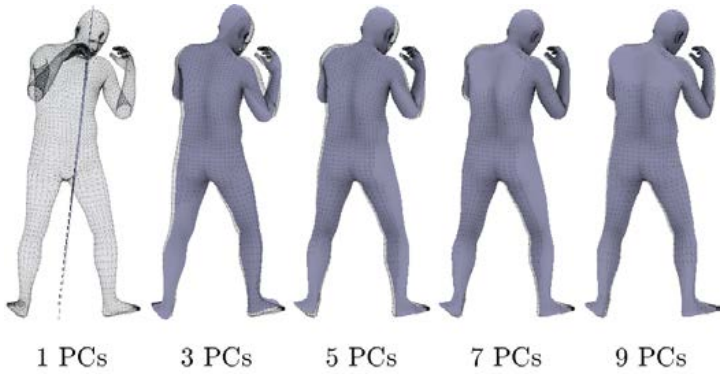


Figure A.51: Visualization of QPCA: Projection of one sample of the “E5 - Hook left poses” motion capture from ACCAD [54] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

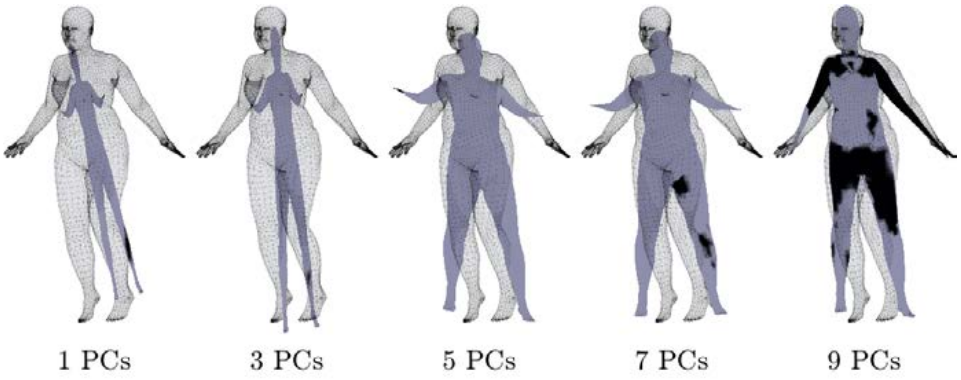


Figure A.52: Visualization of PCA: Projection of one sample of the “jumping-jacks-50022” motion capture from DFAUST [7] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

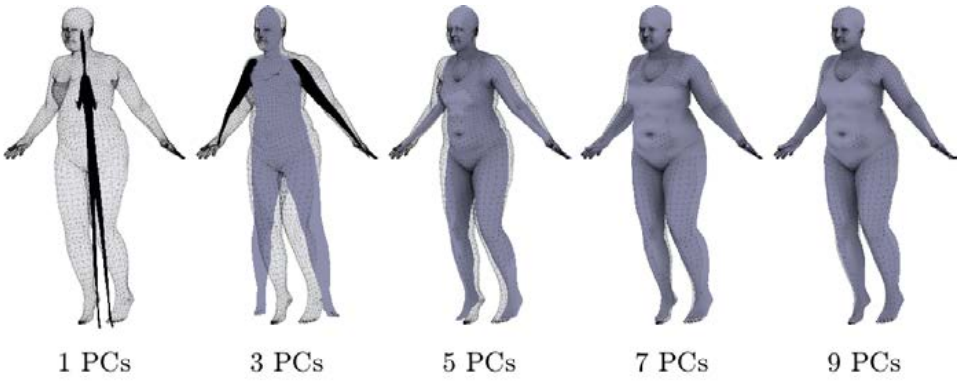


Figure A.53: Visualization of QPCA: Projection of one sample of the “jumping-jacks-50022” motion capture from DFAUST [7] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

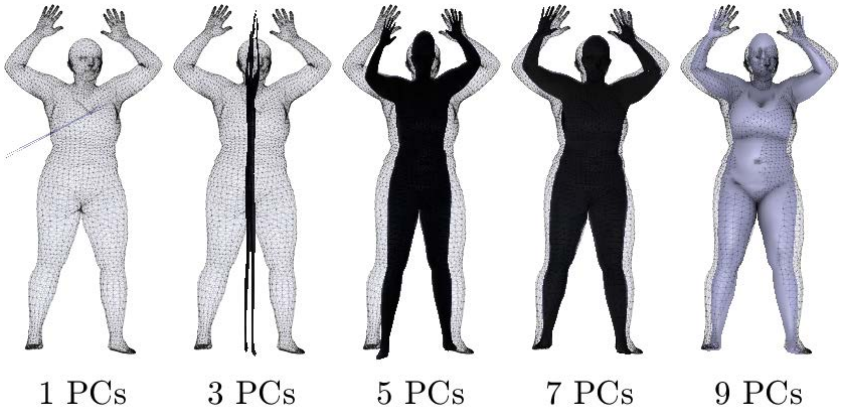


Figure A.54: Visualization of PCA: Projection of one sample of the “jiggle-on-toes-50004” motion capture from DFAUST [7] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

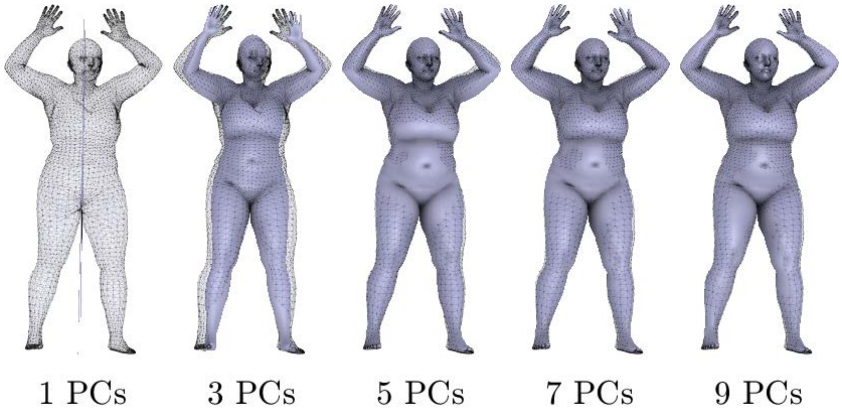


Figure A.55: Visualization of QPCA: Projection of one sample of the “jiggle-on-toes-50004” motion capture from DFAUST [7] onto the subspaces consisting of 1 (left) to 9 (right) principal components stacked on top of each other.

B

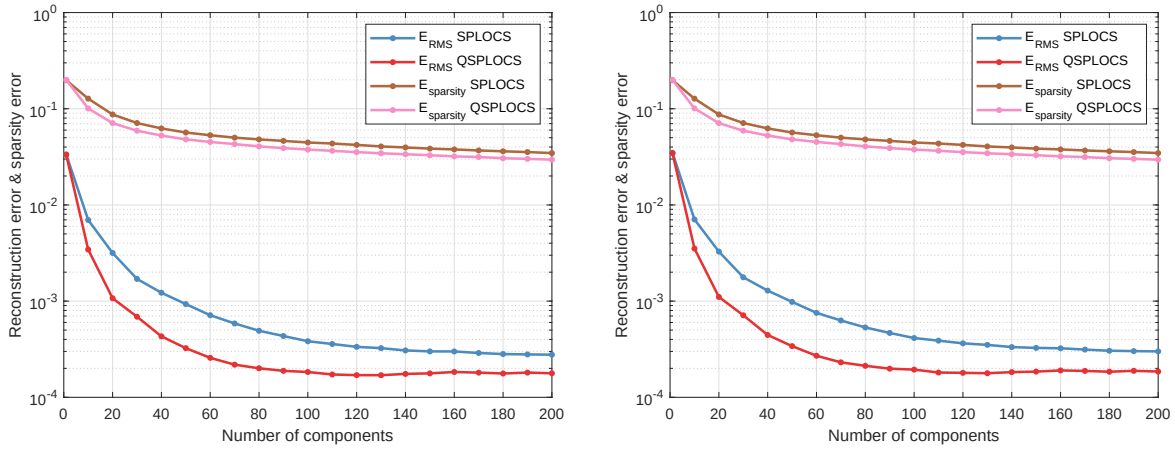
Additional results sparse PCA

In this appendix, we perform some of the experiments from Chapter 7 on the following datasets:

- “C5 - Walk to run” from ACCAD [54].
- “E5 - Hook left poses” from ACCAD [54].
- “jumping-jacks-50022” from DFAUST [7].
- “jiggle-on-toes-50004” from DFAUST [7].

B.1. SPLOCS and QSPLOCS

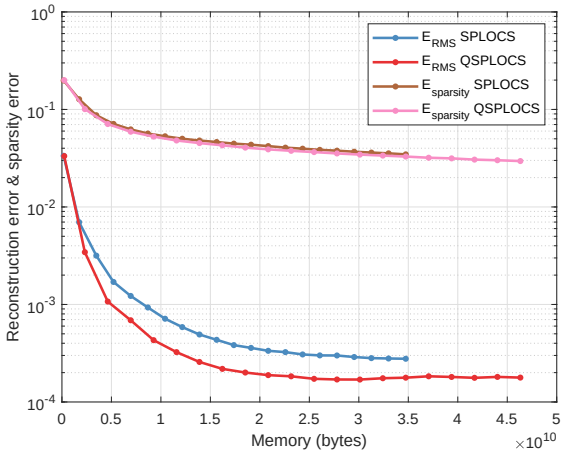
B.1.1. Quantitative evaluation - Reconstruction accuracy and sparsity error



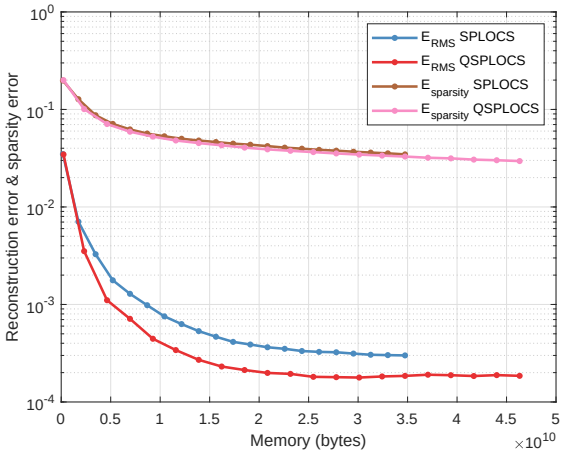
(a) E_{RMS} and $E_{sparsity}$ errors computed on 520 training frames.

(b) E_{RMS} and $E_{sparsity}$ errors computed on 130 (unseen) test frames.

Figure B.1: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the “C5 - Walk to run” motion capture from ACCAD [54].

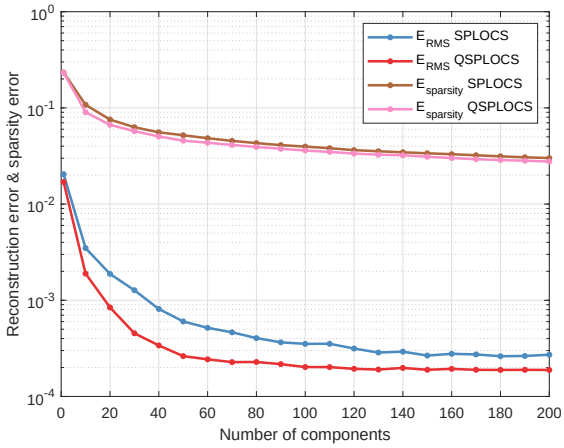


(a) E_{RMS} and $E_{sparsity}$ computed on 520 training frames.

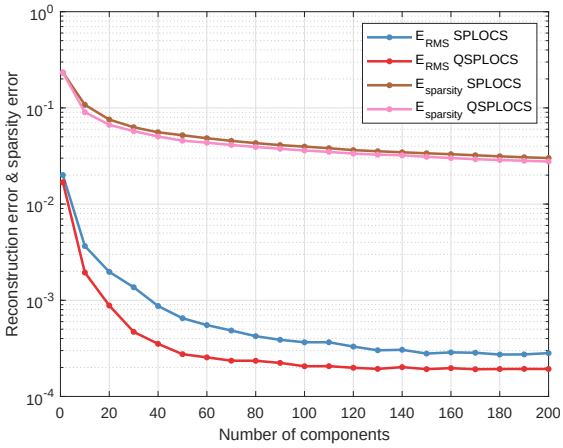


(b) E_{RMS} and $E_{sparsity}$ computed on 130 (unseen) test frames.

Figure B.2: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "C5 - Walk to run" motion capture from the ACCAD [54].

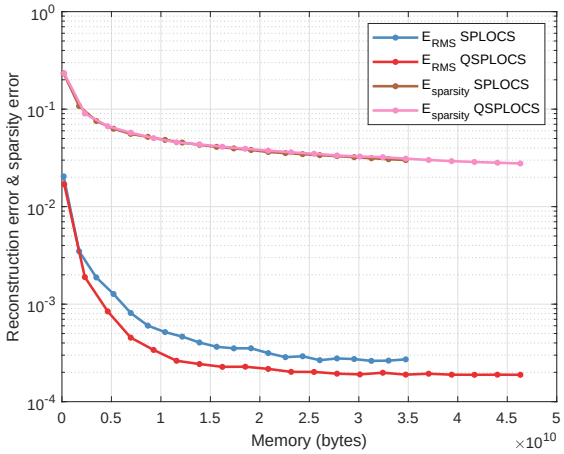


(a) E_{RMS} and $E_{sparsity}$ computed on 262 training frames.

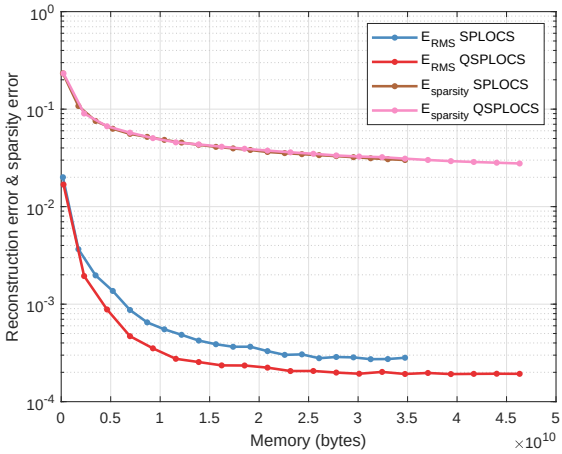


(b) E_{RMS} and $E_{sparsity}$ computed on 65 (unseen) test frames.

Figure B.3: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "E5 - Hook left poses" motion capture from ACCAD [54].

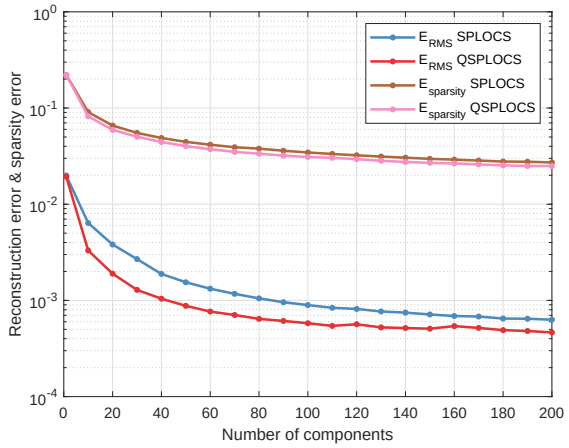


(a) E_{RMS} and $E_{sparsity}$ computed on 262 training frames.

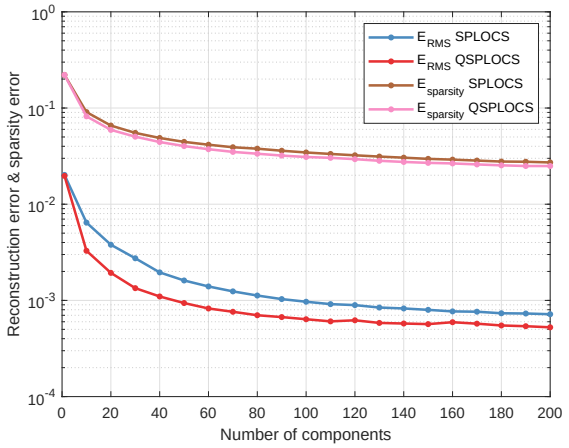


(b) E_{RMS} and $E_{sparsity}$ computed on 65 (unseen) test frames.

Figure B.4: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "E5 - Hook left poses" motion capture from the ACCAD [54].

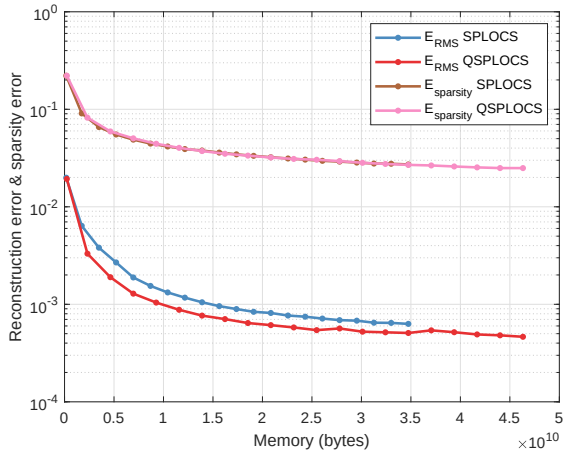


(a) E_{RMS} and $E_{sparsity}$ computed on 247 training frames.

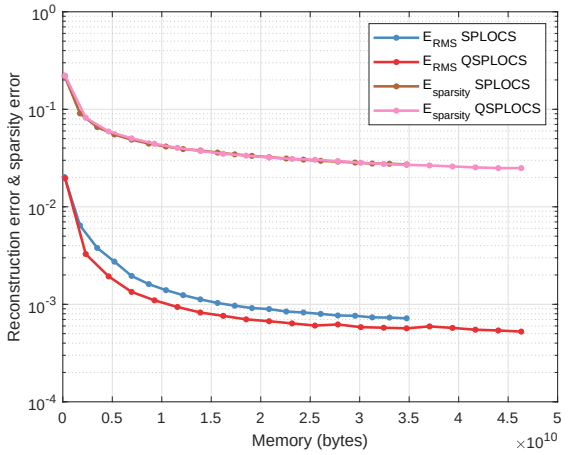


(b) E_{RMS} and $E_{sparsity}$ computed on 61 (unseen) test frames.

Figure B.5: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "jumping-jacks-50022" motion capture from DFAUST [7].

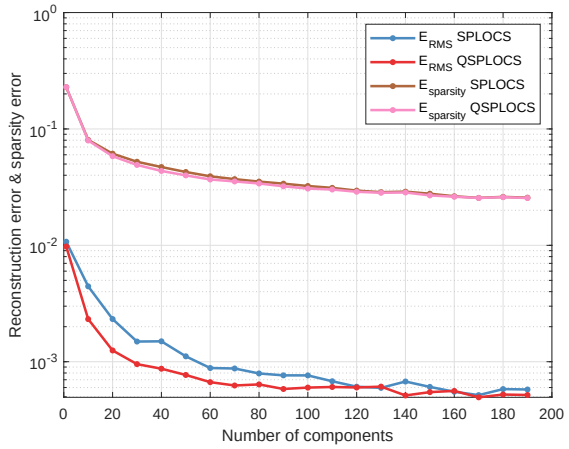


(a) E_{RMS} and $E_{sparsity}$ computed on 247 training frames.

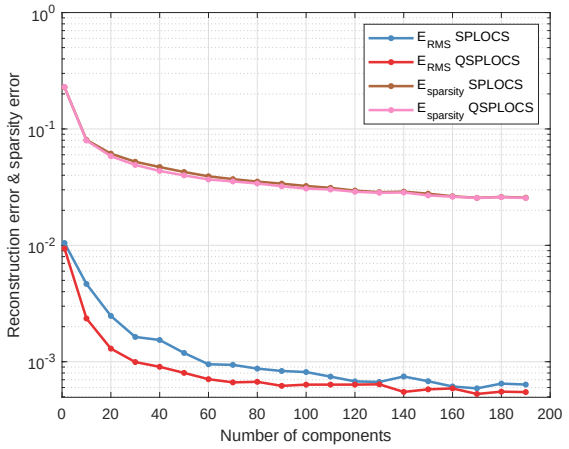


(b) E_{RMS} and $E_{sparsity}$ computed on 61 (unseen) test frames.

Figure B.6: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "jumping-jacks-50022" motion capture from the DFAUST [7].

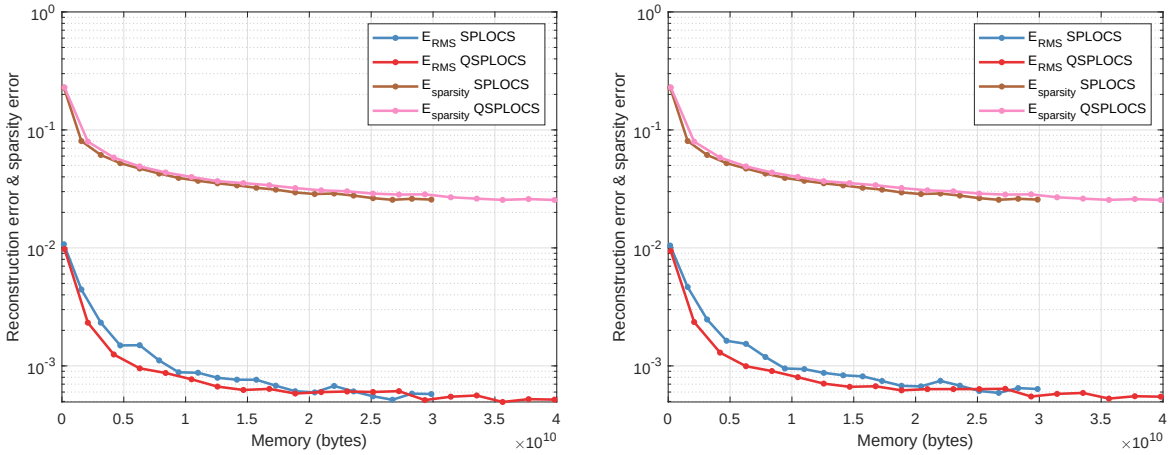


(a) E_{RMS} and $E_{sparsity}$ computed on 192 training frames.



(b) E_{RMS} and $E_{sparsity}$ computed on 47 (unseen) test frames.

Figure B.7: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "jiggle-on-toes-50004" motion capture from DFAUST [7].



(a) E_{RMS} and $E_{sparsity}$ computed on 192 training frames.

(b) E_{RMS} and $E_{sparsity}$ computed on 47 (unseen) test frames.

Figure B.8: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "jiggle-on-toes-50004" motion capture from the DFAUST [7].

B.1.2. Visual comparison

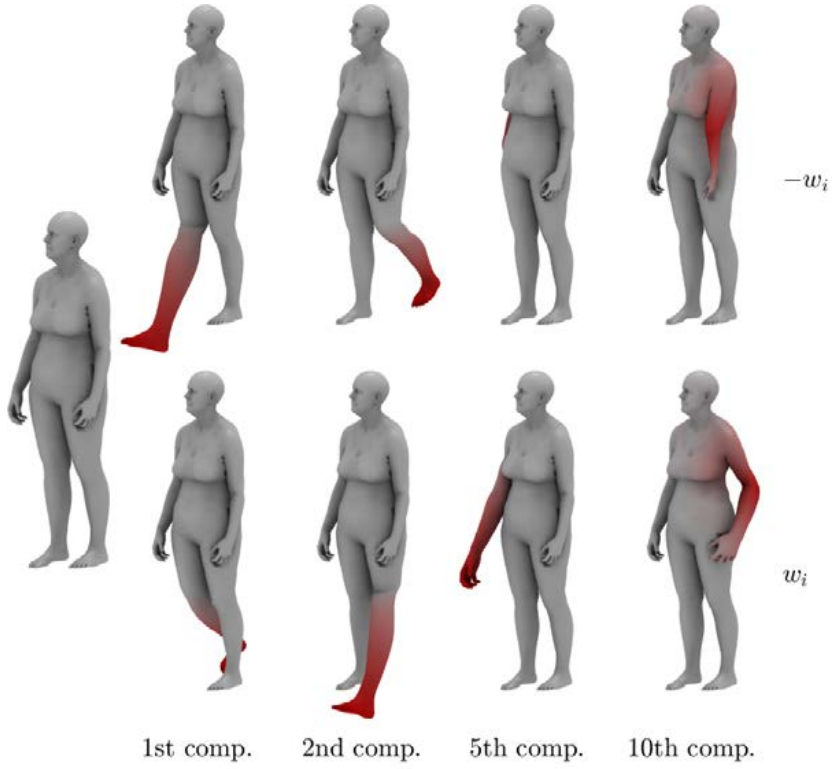


Figure B.9: Visualization of SPLOCS: 4 sparse components on the "C5 - Walk to run" motion capture from ACCAD [54]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

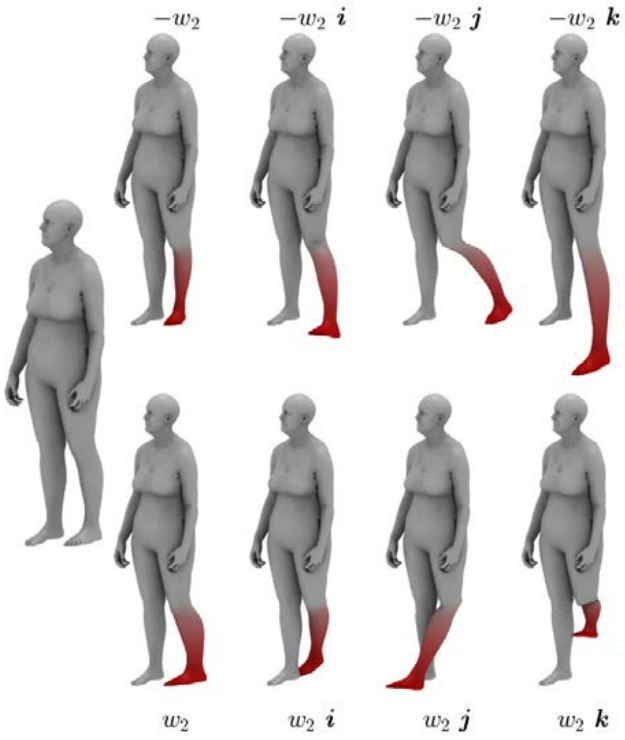


Figure B.10: Visualization of QSPLOCS: The 2nd quaternion sparse component on the “C5 - Walk to run” motion capture from ACCAD [54]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

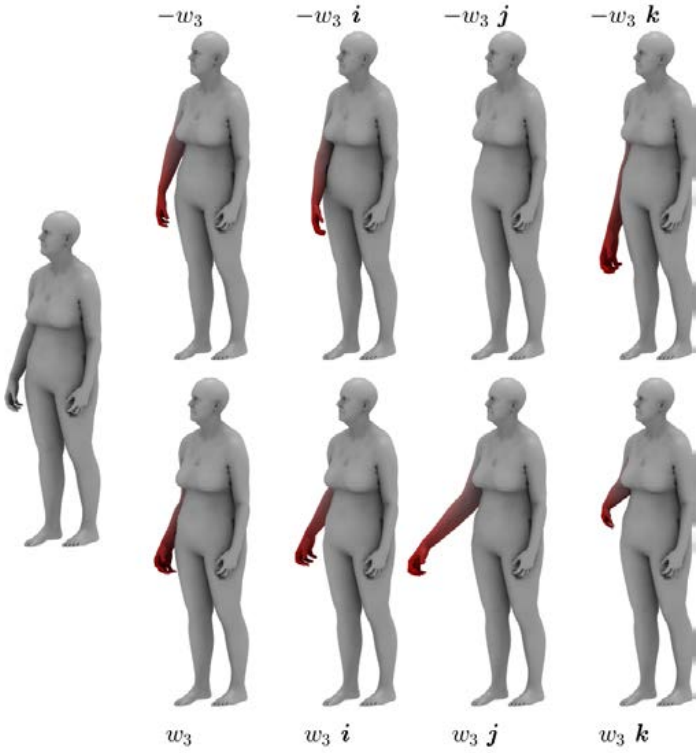


Figure B.11: Visualization of QSPLOCS: The 3rd quaternion sparse component on the “C5 - Walk to run” motion capture from ACCAD [54]. Models corresponding to $-w_3$ (top) and w_3 (bottom) along the four dimensions of the 3rd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

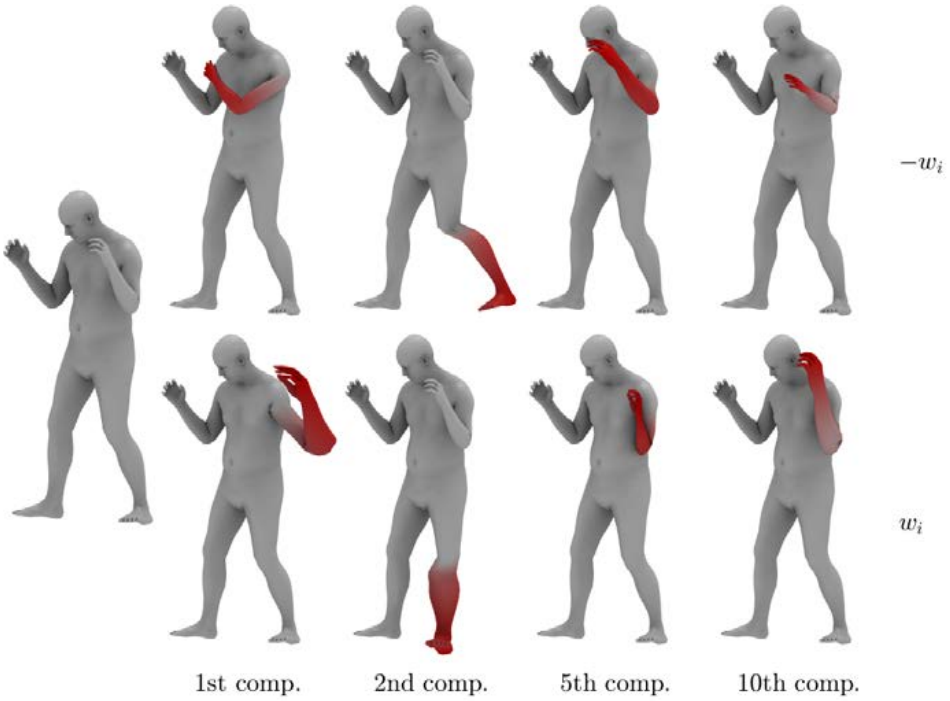


Figure B.12: Visualization of SPLOCS: 4 sparse components on the “E5 - Hook left poses” motion capture from ACCAD [54]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

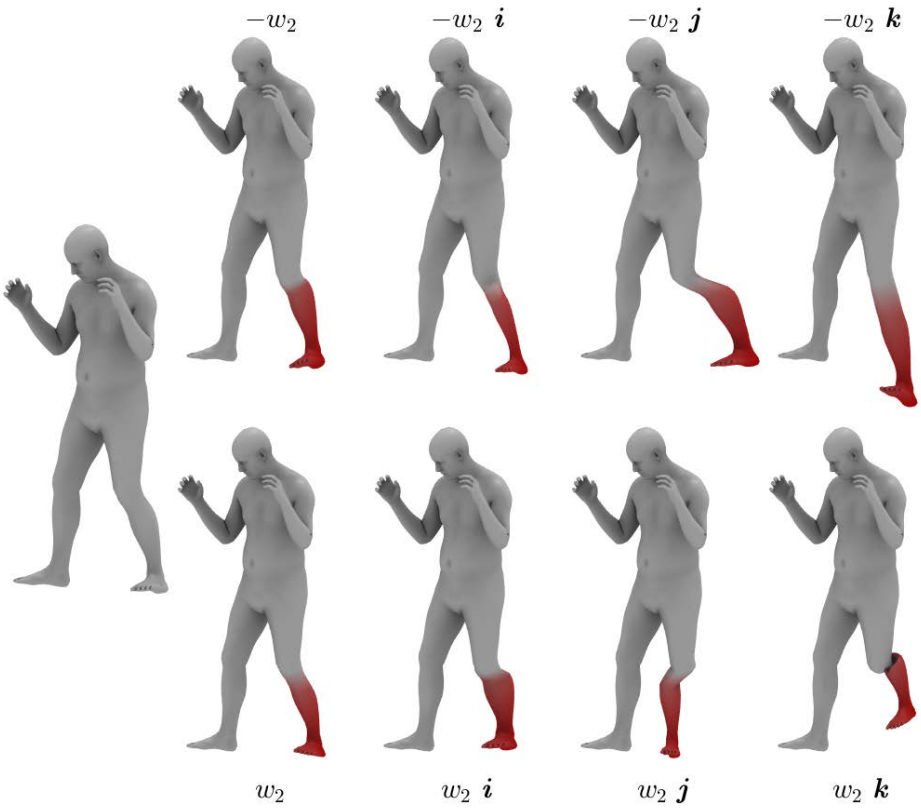


Figure B.13: Visualization of QSPLOCS: The 2nd quaternion sparse component on the “E5 - Hook left poses” motion capture from ACCAD [54]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

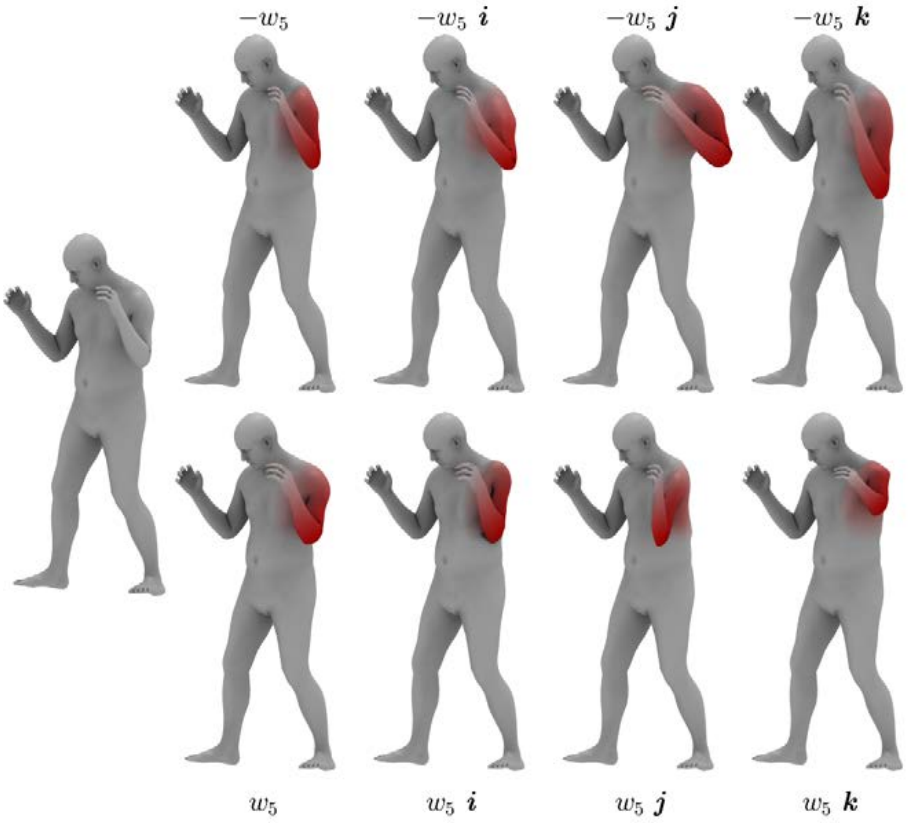


Figure B.14: Visualization of QSPLOCS: The 5th quaternion sparse component on the "E5 - Hook left poses" motion capture from ACCAD [54]. Models corresponding to $-w_5$ (top) and w_5 (bottom) along the four dimensions of the 5th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

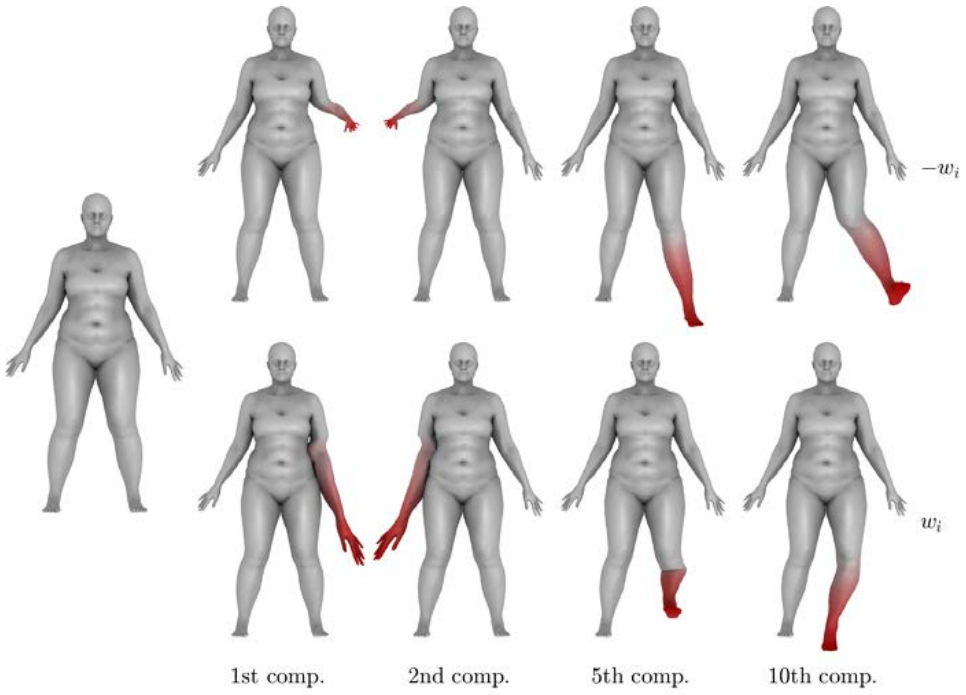


Figure B.15: Visualization of SPLOCS: 4 sparse components on the "jumping-jacks-50022" motion capture from DFAUST [7]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

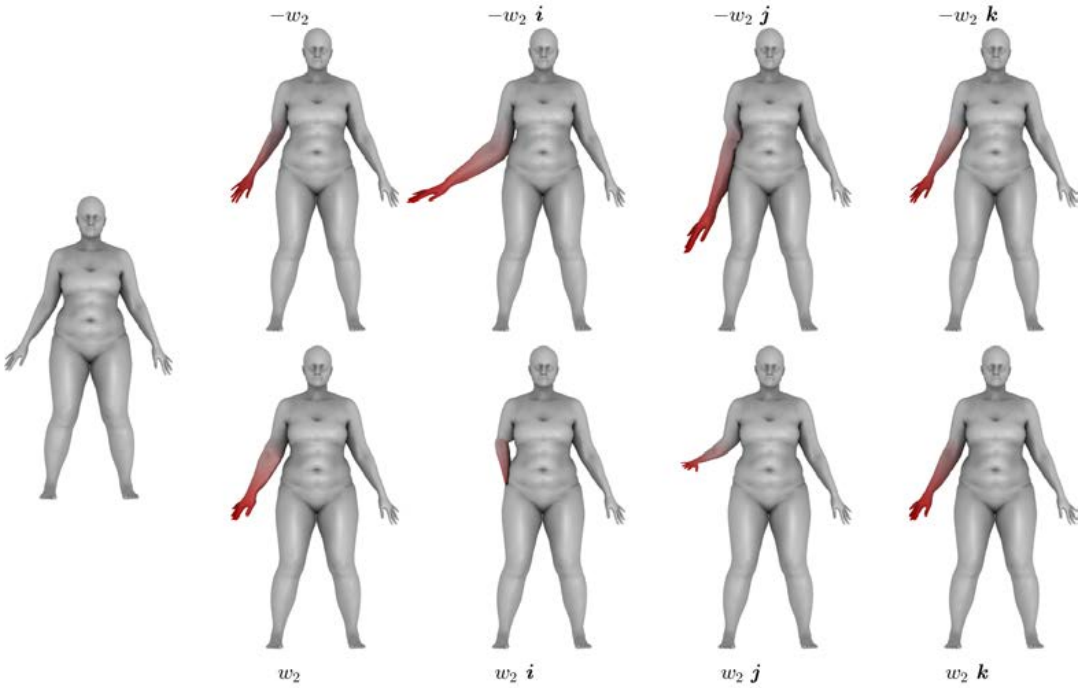


Figure B.16: Visualization of QSPLOCS: The 2nd quaternion sparse component on the "jumping-jacks-50022" motion capture from DFAUST [7]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

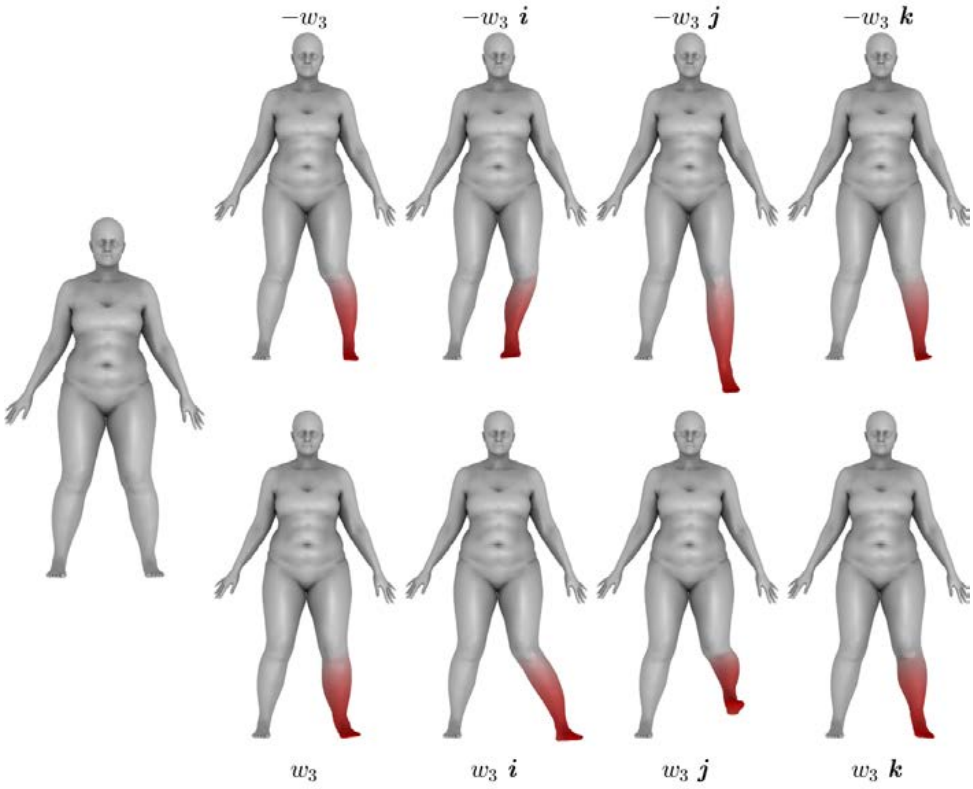


Figure B.17: Visualization of QSPLOCS: The 3rd quaternion sparse component on the "jumping-jacks-50022" motion capture from DFAUST [7]. Models corresponding to $-w_3$ (top) and w_3 (bottom) along the four dimensions of the 3rd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

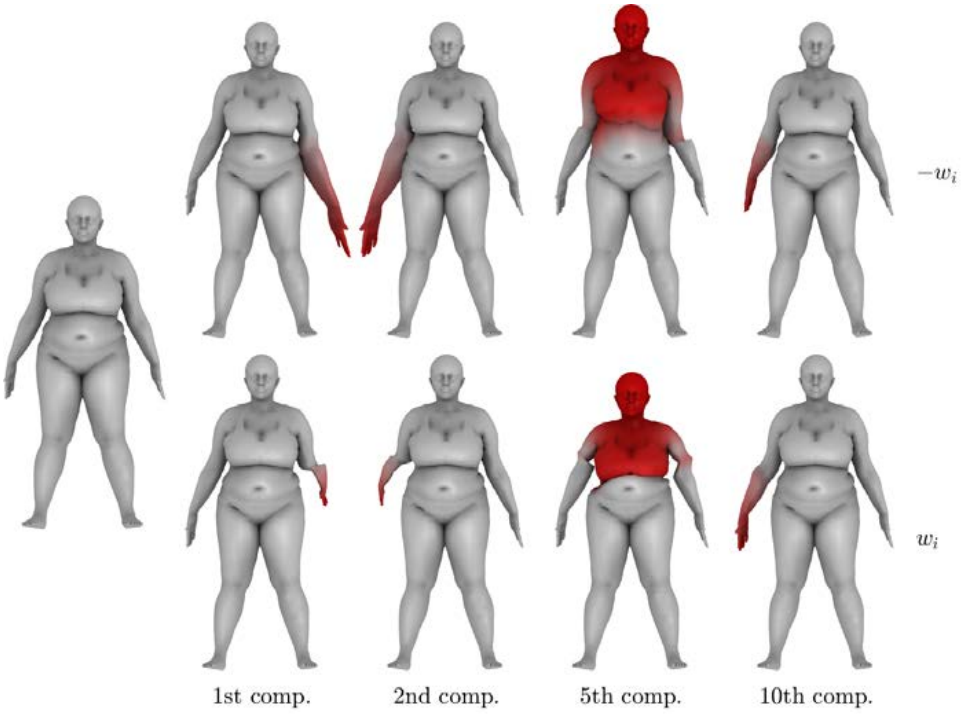


Figure B.18: Visualization of SPLOCS: 4 sparse components on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

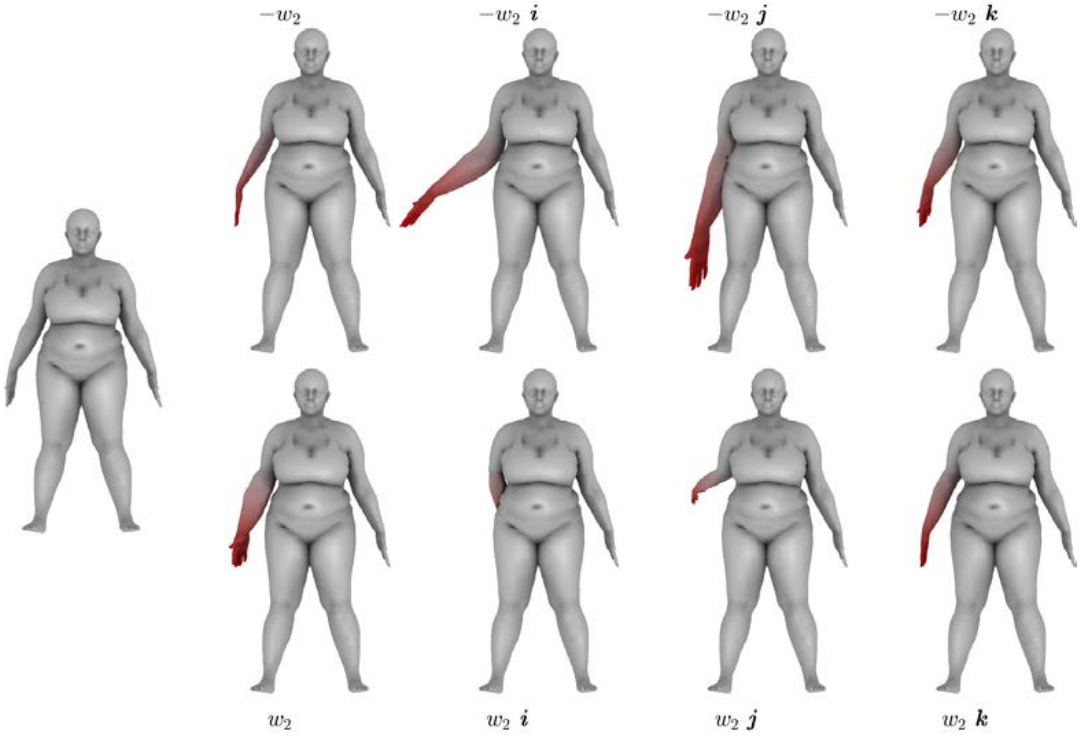


Figure B.19: Visualization of QSPLOCS: The 2nd quaternion sparse component on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

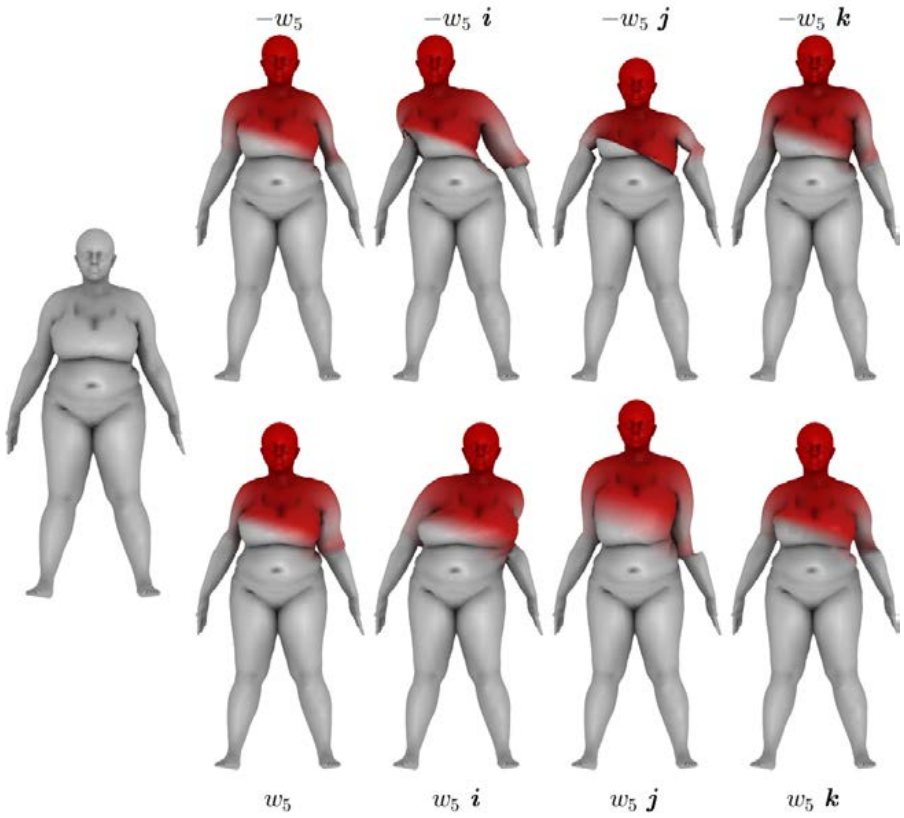
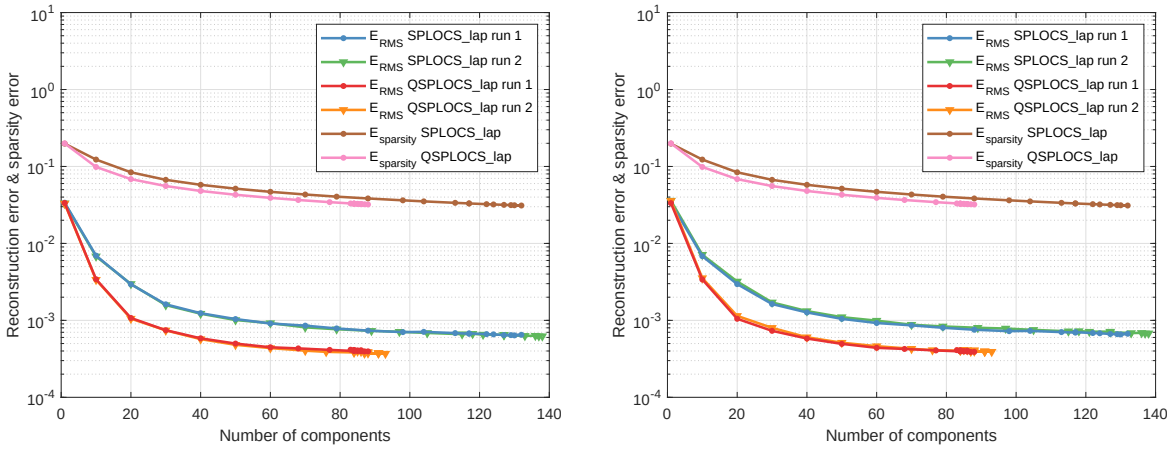


Figure B.20: Visualization of QSPLOCS: The 5th quaternion sparse component on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. Models corresponding to $-w_5$ (top) and w_5 (bottom) along the four dimensions of the 5th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

B.2. SPLOCS_lap and QSPLOCS_lap

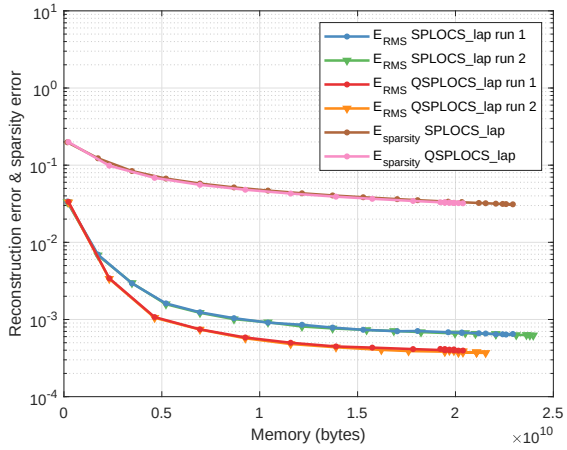
B.2.1. Quantitative evaluation - Reconstruction accuracy and sparsity error



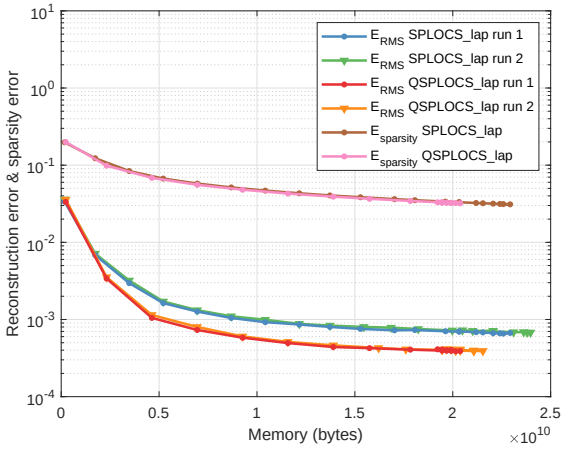
(a) E_{RMS} and $E_{sparsity}$ errors computed on 520 training frames.

(b) E_{RMS} and $E_{sparsity}$ errors computed on 130 (unseen) test frames.

Figure B.21: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "C5 - Walk to run" motion capture from ACCAD [54].

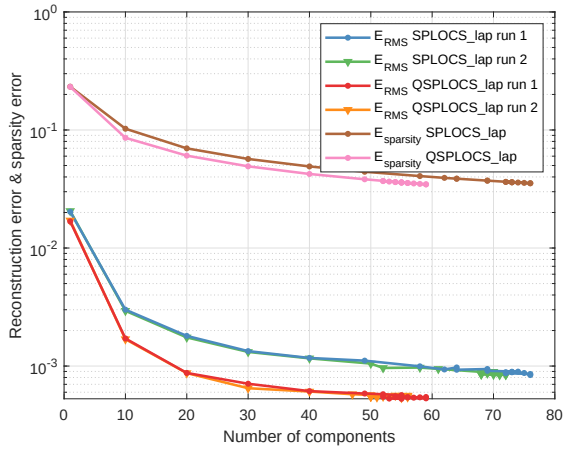


(a) E_{RMS} and $E_{sparsity}$ computed on 520 training frames.

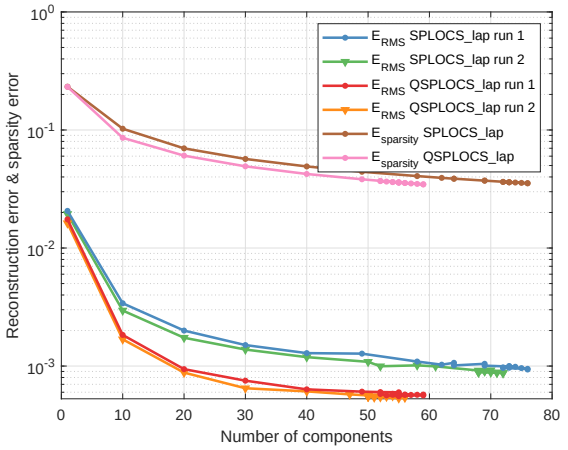


(b) E_{RMS} and $E_{sparsity}$ computed on 130 (unseen) test frames.

Figure B.22: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "C5 - Walk to run" motion capture from the ACCAD [54].

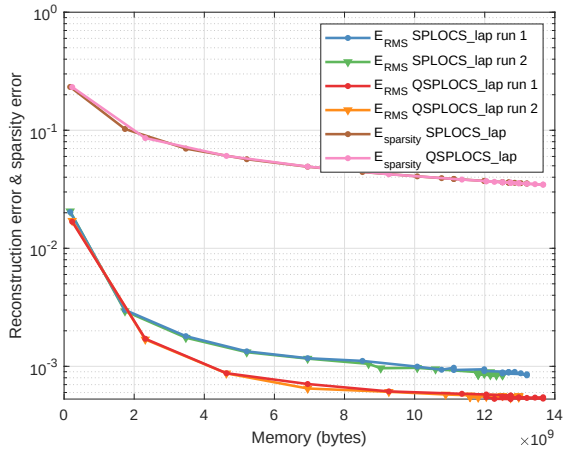


(a) E_{RMS} and $E_{sparsity}$ computed on 262 training frames.

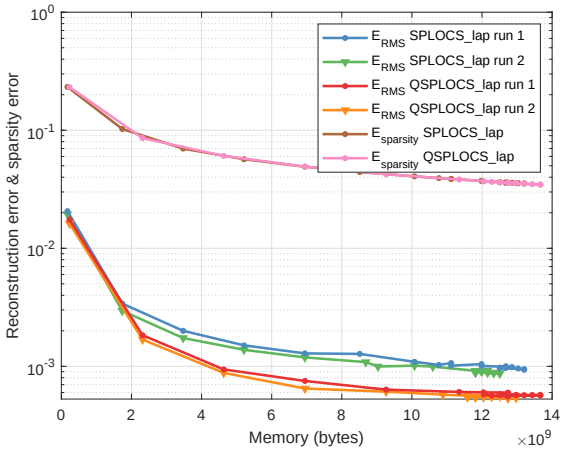


(b) E_{RMS} and $E_{sparsity}$ computed on 65 (unseen) test frames.

Figure B.23: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "E5 - Hook left poses" motion capture from ACCAD [54].

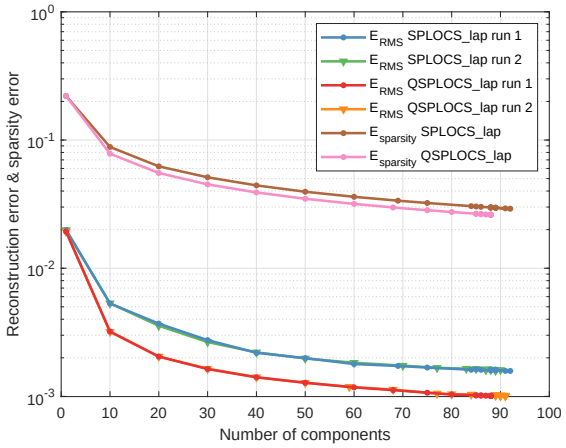


(a) E_{RMS} and $E_{sparsity}$ computed on 262 training frames.

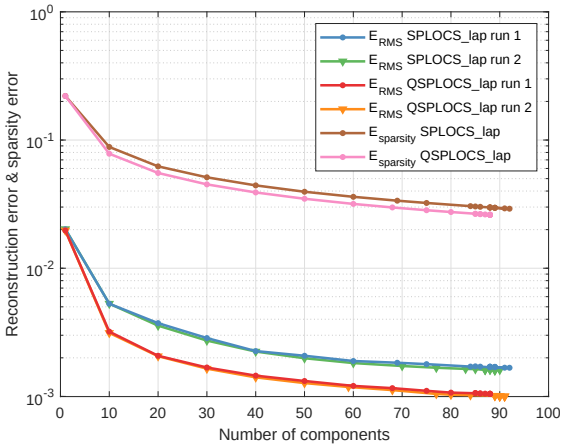


(b) E_{RMS} and $E_{sparsity}$ computed on 65 (unseen) test frames.

Figure B.24: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "E5 - Hook left poses" motion capture from the ACCAD [54].

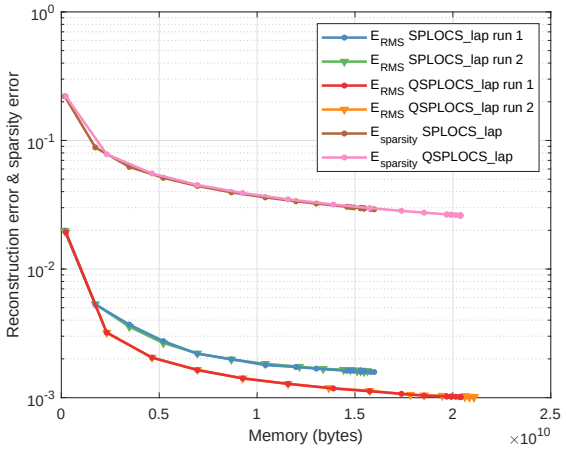


(a) E_{RMS} and $E_{sparsity}$ computed on 247 training frames.

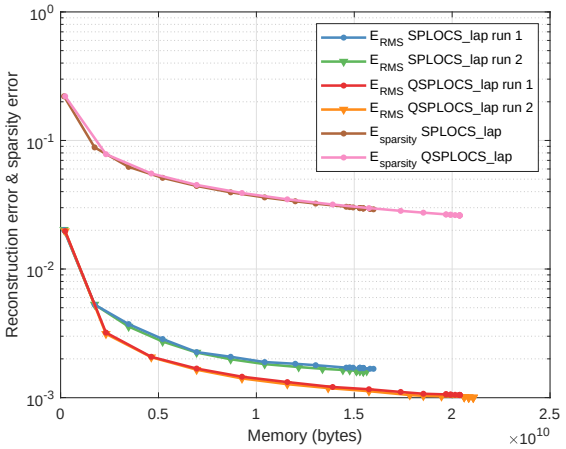


(b) E_{RMS} and $E_{sparsity}$ computed on 61 (unseen) test frames.

Figure B.25: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "jumping-jacks-50022" motion capture from DFAUST [7].

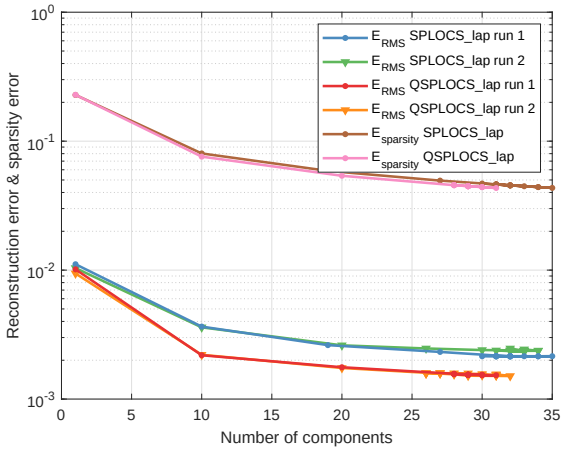


(a) E_{RMS} and $E_{sparsity}$ computed on 247 training frames.

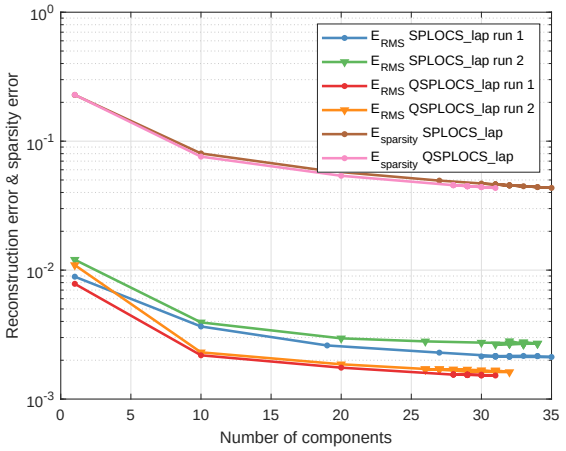


(b) E_{RMS} and $E_{sparsity}$ computed on 61 (unseen) test frames.

Figure B.26: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "jumping-jacks-50022" motion capture from the DFAUST [7].

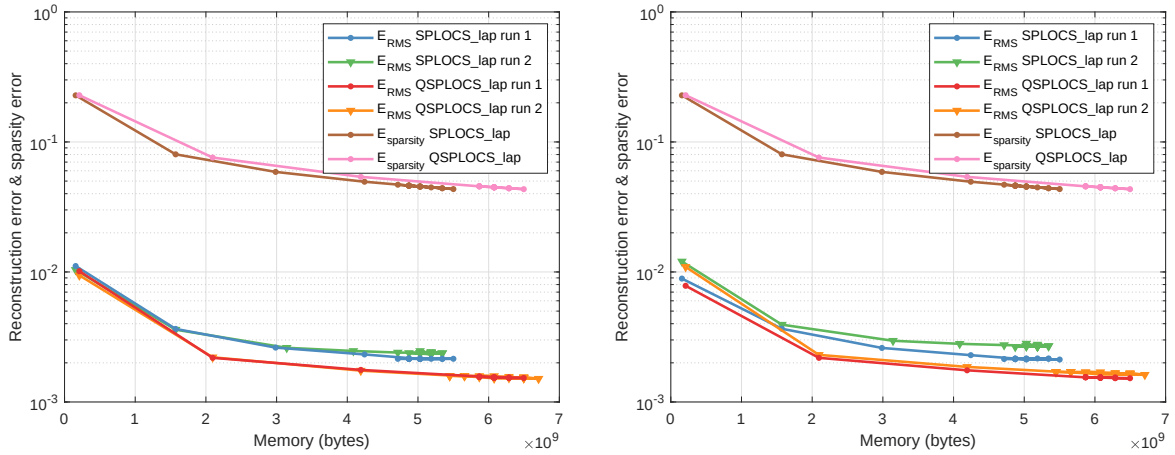


(a) E_{RMS} and $E_{sparsity}$ computed on 192 training frames.



(b) E_{RMS} and $E_{sparsity}$ computed on 47 (unseen) test frames.

Figure B.27: Reconstruction error and sparsity error (y-axis) with respect to the number of components used (x-axis) on the "jiggle-on-toes-50004" motion capture from DFAUST [7].



(a) E_{RMS} and $E_{sparsity}$ computed on 192 training frames.

(b) E_{RMS} and $E_{sparsity}$ computed on 47 (unseen) test frames.

Figure B.28: Reconstruction error and sparsity error (y-axis) with respect to the memory needed to store the number of components (x-axis) on the "jiggle-on-toes-50004" motion capture from the DFAUST [7].

B.2.2. Visual comparison

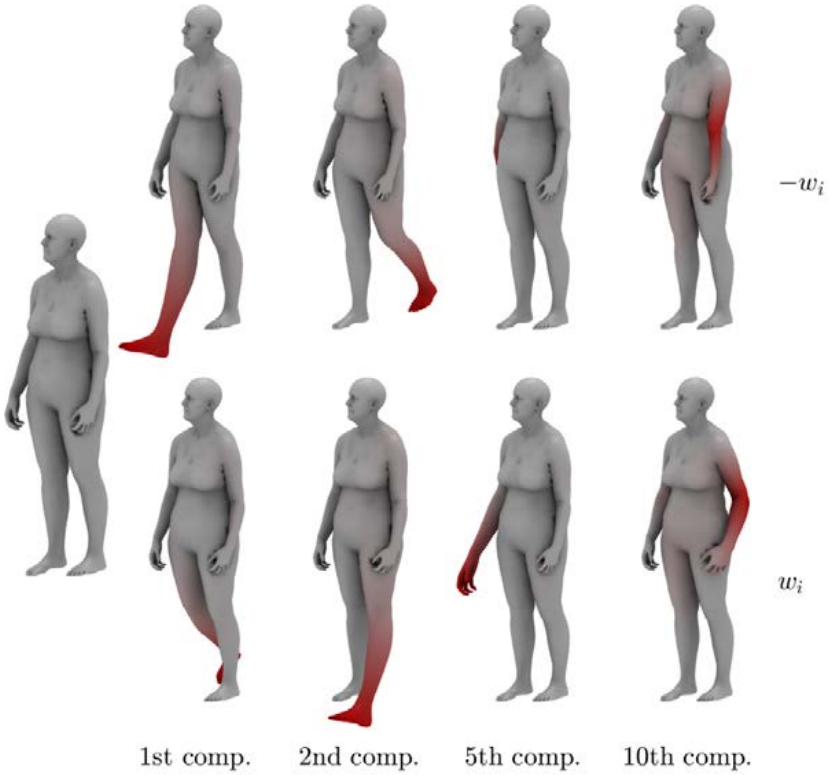


Figure B.29: Visualization of SPLOCS: 4 sparse components on the "C5 - Walk to run" motion capture from ACCAD [54]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

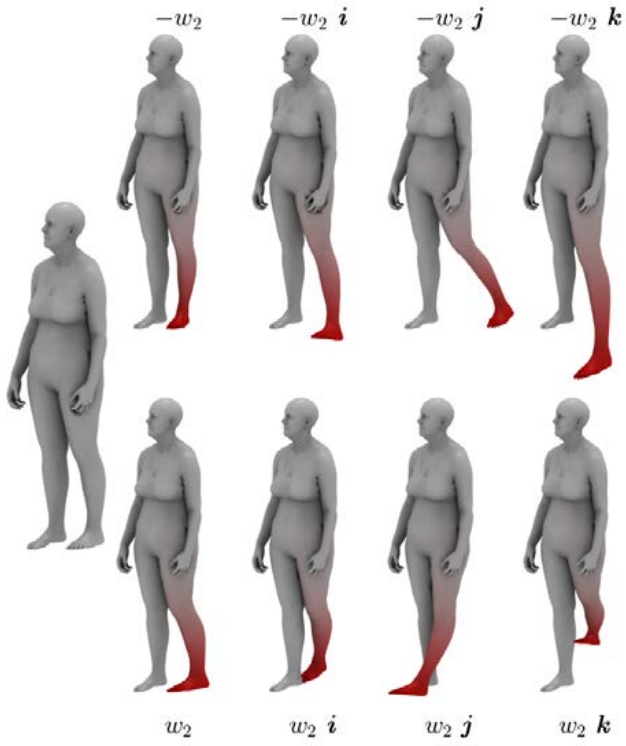


Figure B.30: Visualization of QSPLOCS: The 2nd quaternion sparse component on the “C5 - Walk to run” motion capture from ACCAD [54]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

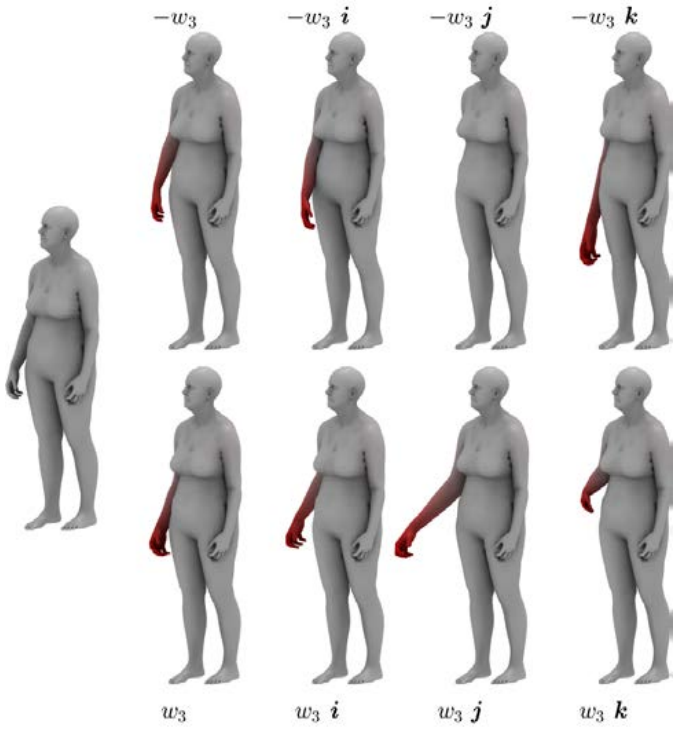


Figure B.31: Visualization of QSPLOCS: The 3rd quaternion sparse component on the “C5 - Walk to run” motion capture from ACCAD [54]. Models corresponding to $-w_3$ (top) and w_3 (bottom) along the four dimensions of the 3rd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

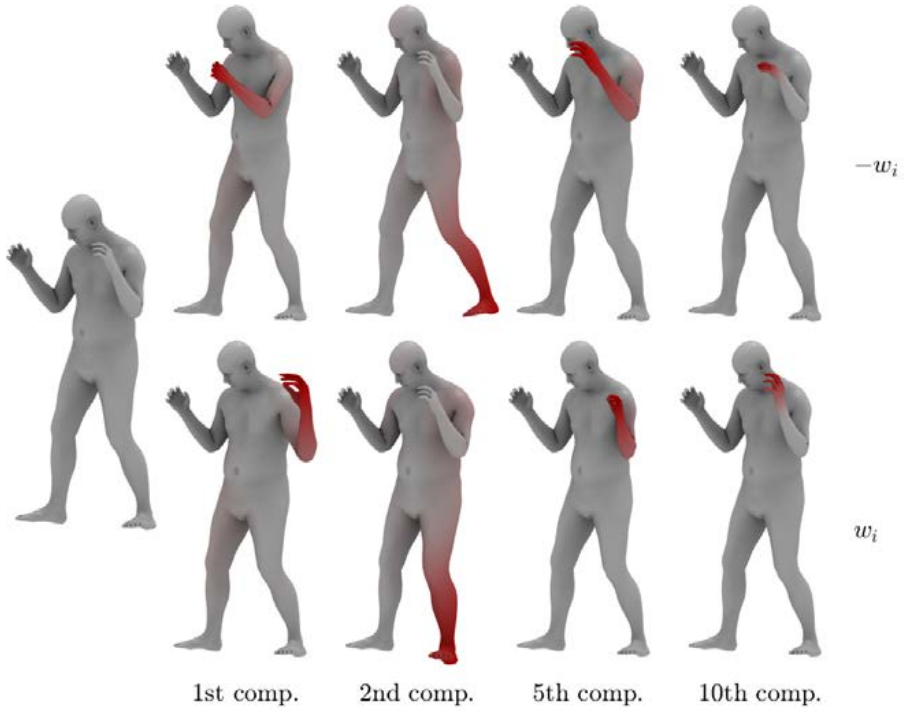


Figure B.32: Visualization of SPLOCS: 4 sparse components on the “E5 - Hook left poses” motion capture from ACCAD [54]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

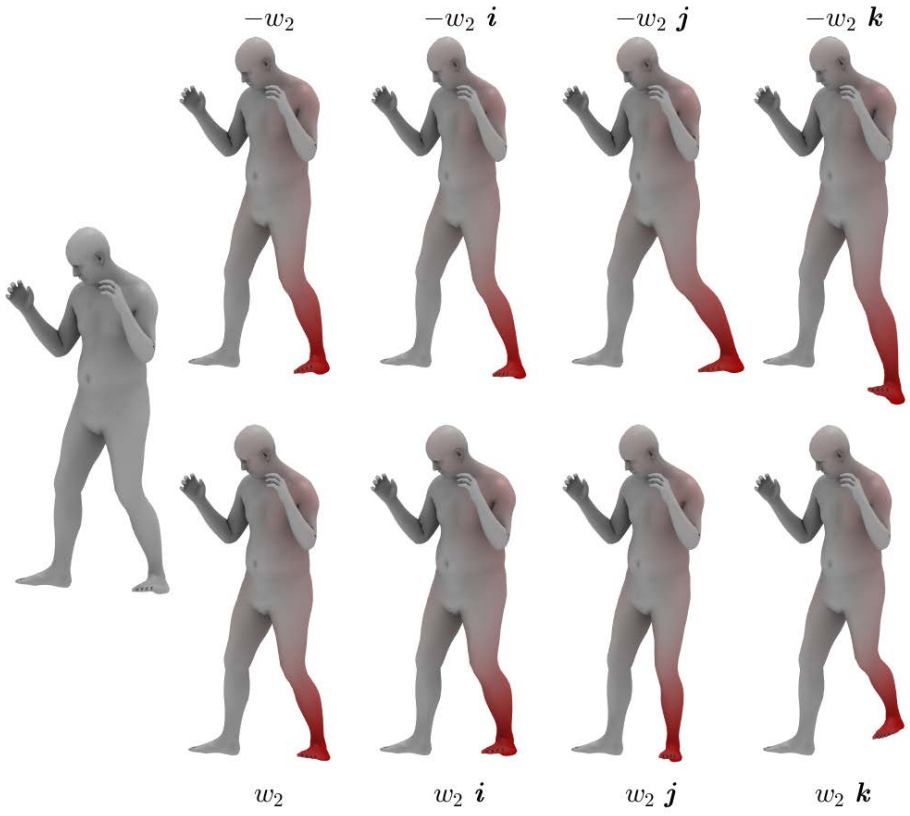


Figure B.33: Visualization of QSPLOCS: The 2nd quaternion sparse component on the “E5 - Hook left poses” motion capture from ACCAD [54]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

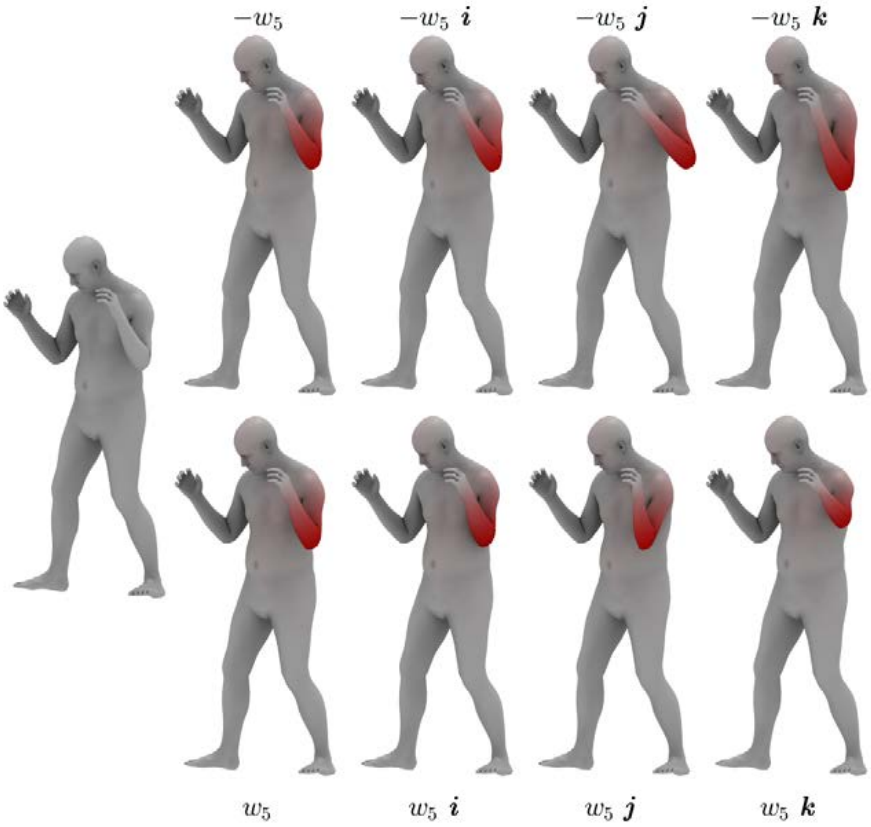


Figure B.34: Visualization of QSPLOCS: The 5th quaternion sparse component on the "E5 - Hook left poses" motion capture from ACCAD [54]. Models corresponding to $-w_5$ (top) and w_5 (bottom) along the four dimensions of the 5th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

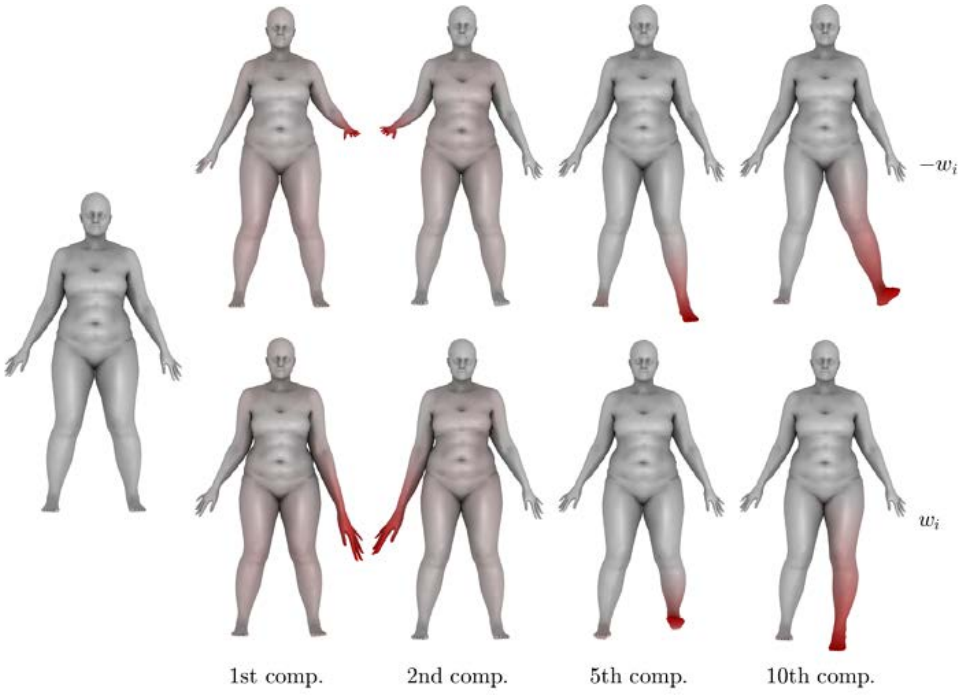


Figure B.35: Visualization of SPLOCS: 4 sparse components on the "jumping-jacks-50022" motion capture from DFAUST [7]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

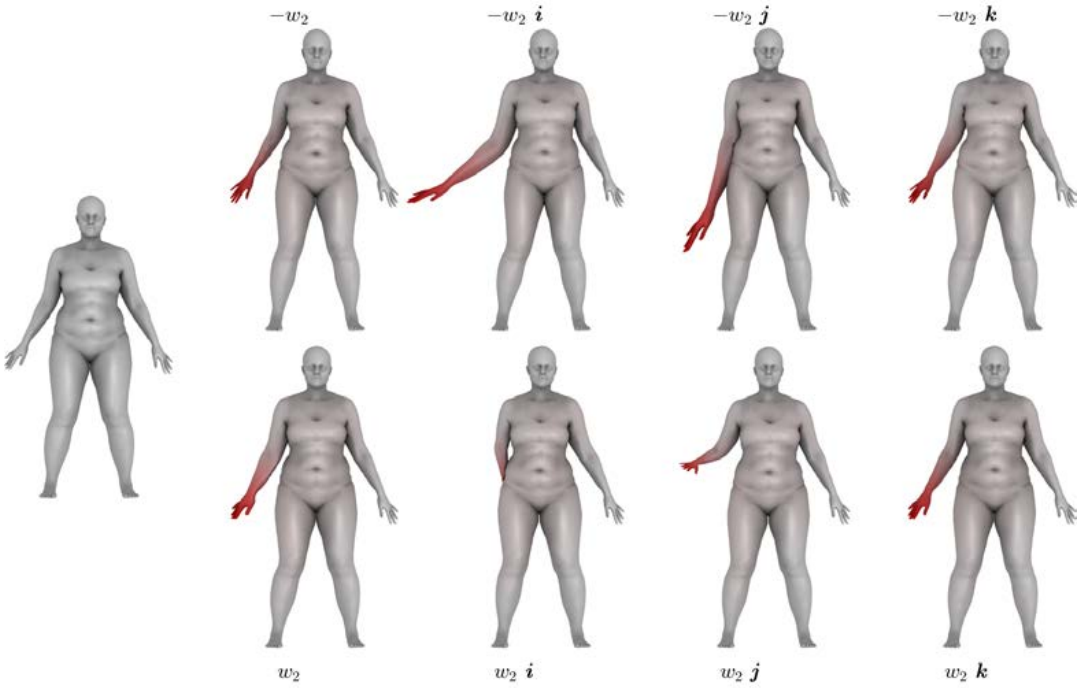


Figure B.36: Visualization of QSPLOCS: The 2nd quaternion sparse component on the "jumping-jacks-50022" motion capture from DFAUST [7]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

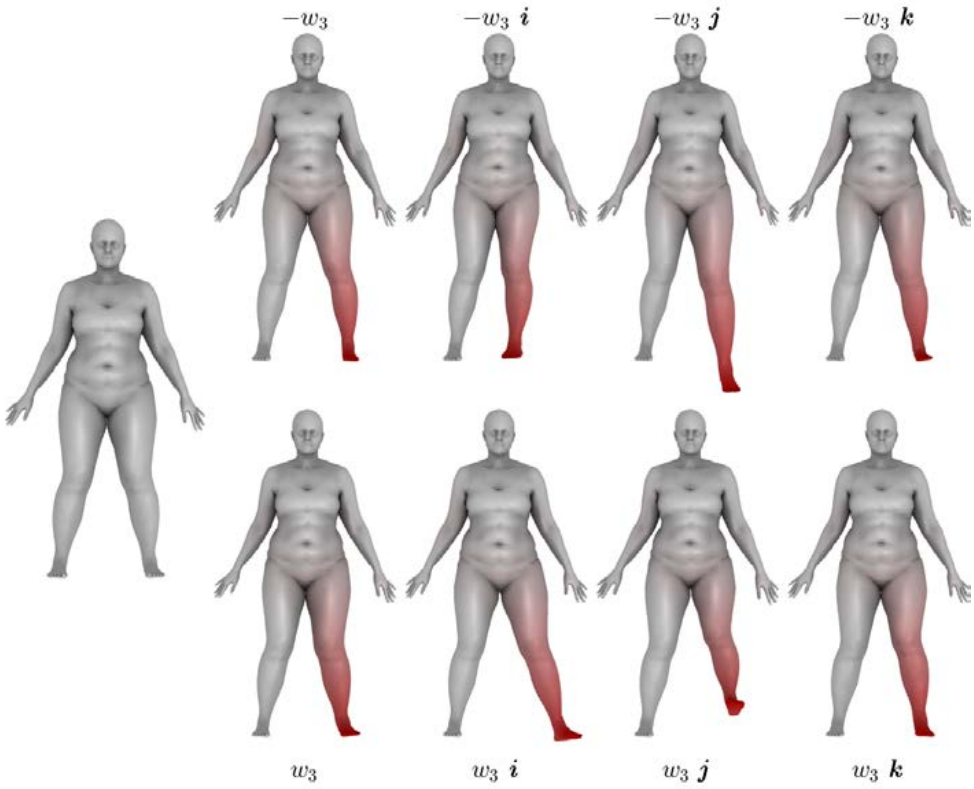


Figure B.37: Visualization of QSPLOCS: The 3rd quaternion sparse component on the "jumping-jacks-50022" motion capture from DFAUST [7]. Models corresponding to $-w_3$ (top) and w_3 (bottom) along the four dimensions of the 3rd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

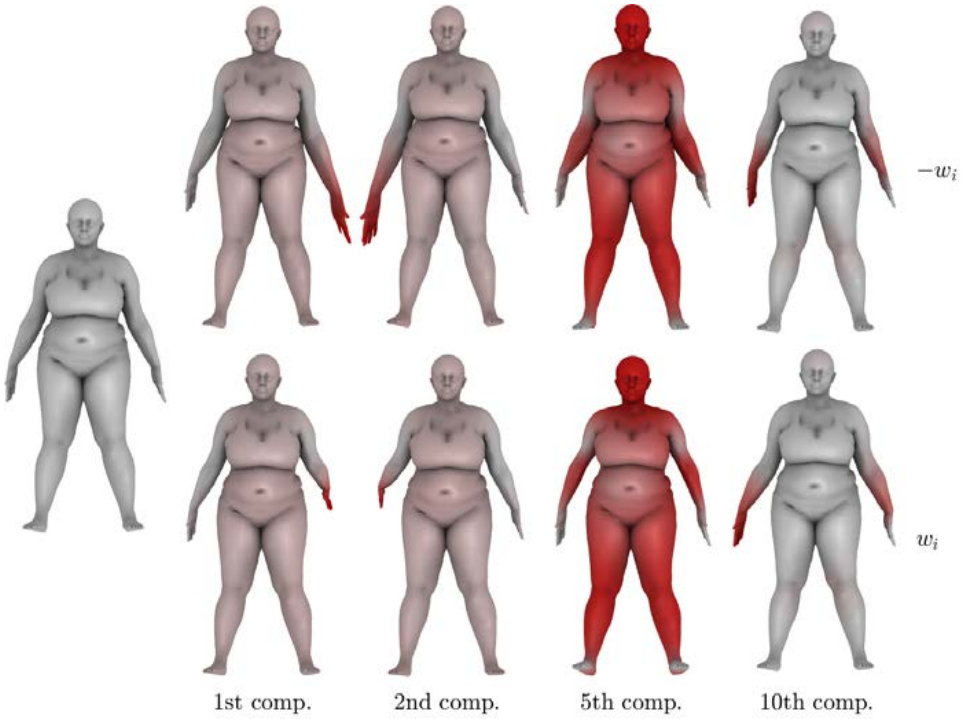


Figure B.38: Visualization of SPLOCS: 4 sparse components on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. Models corresponding to $-w_i$ (top) and w_i (bottom) along the ith component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

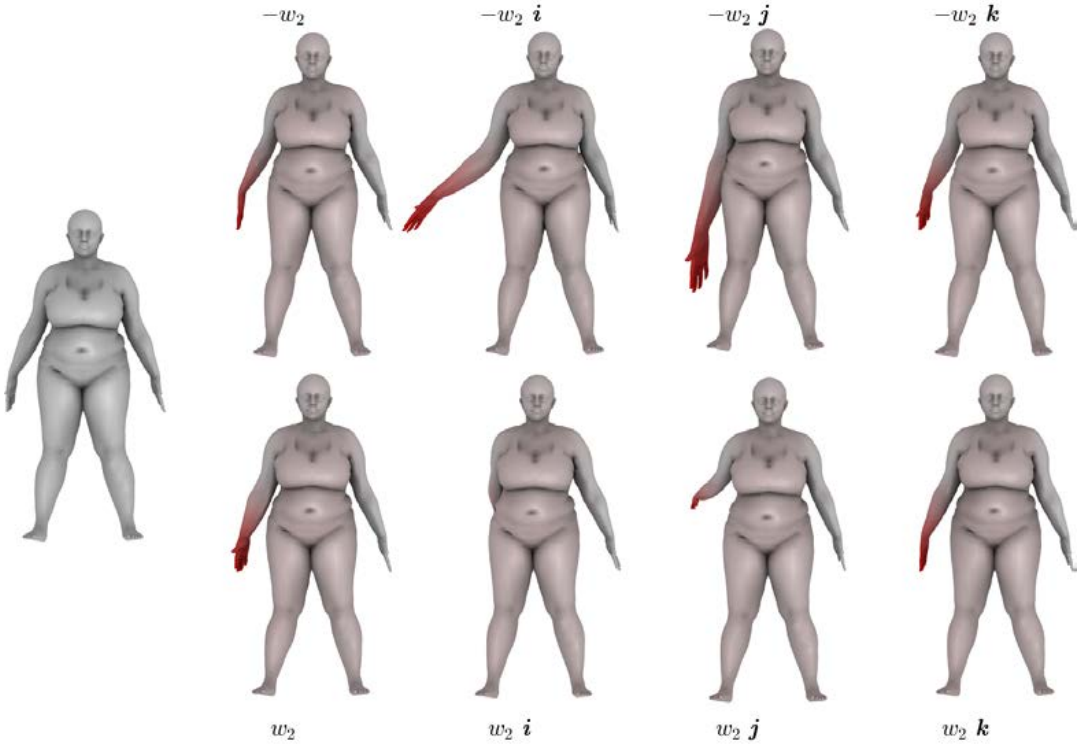


Figure B.39: Visualization of QSPLOCS: The 2nd quaternion sparse component on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. Models corresponding to $-w_2$ (top) and w_2 (bottom) along the four dimensions of the 2nd component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

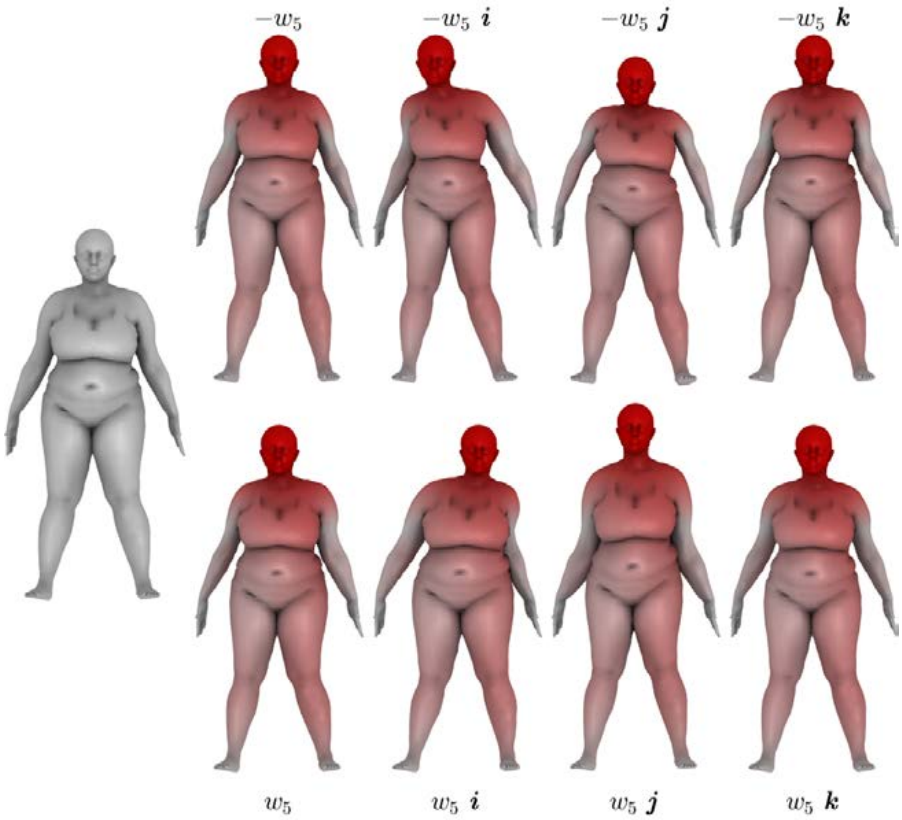


Figure B.40: Visualization of QSPLOCS: The 5th quaternion sparse component on the "jiggle-on-toes-50004" motion capture from DFAUST [7]. Models corresponding to $-w_5$ (top) and w_5 (bottom) along the four dimensions of the 5th component are displayed. Color coding shows the magnitude of vertex displacements inside the components.

Bibliography

[1] Alexa, M., and Müller, W. Representing Animations by Principal Components. *Computer Graphics Forum 19*, 3 (sep 2000), 411–418.

[2] Bach, F., Jenatton, R., Mairal, J., and Obozinski, G. Optimization with sparsity-inducing penalties. *Foundations and Trends in Machine Learning 4*, 1 (2011), 1–106.

[3] Bernard, F., Gemmar, P., Hertel, F., Goncalves, J., and Thunberg, J. Linear shape deformation models with local support using graph-based structured matrix factorisation. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition* (oct 2016), vol. 2016-Decem, IEEE Computer Society, pp. 5629–5638.

[4] Besl, P. J., and McKay, N. D. A Method for Registration of 3-D Shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence 14*, 2 (1992), 239–256.

[5] Blais, G., and Levine, M. D. Registering Multiview Range Data to Create 3D Computer Objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence 17*, 8 (1995), 820–824.

[6] Blanz, V., and Vetter, T. A morphable model for the synthesis of 3D faces. *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1999* (1999), 187–194.

[7] Bogo, F., Romero, J., Pons-Moll, G., and Black, M. J. Dynamic FAUST: Registering human bodies in motion. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)* (July 2017).

[8] Boyd, S., Parikh, N., Chu, E., Peleato, B., and Eckstein, J. Distributed optimization and statistical learning via the alternating direction method of multipliers, 2010.

[9] Bryan, R., Surya Mohan, P., Hopkins, A., Galloway, F., Taylor, M., and Nair, P. B. Statistical modelling of the whole human femur incorporating geometric and material properties. *Medical Engineering and Physics 32*, 1 (2010), 57–65.

[10] Chen, Y., and Medioni, G. Object modeling by registration of multiple range images. In *Proceedings - IEEE International Conference on Robotics and Automation* (1991), vol. 3, Publ by IEEE, pp. 2724–2729.

[11] Colosimo, B. M., and Pacella, M. Analyzing the effect of process parameters on the shape of 3D profiles. *Journal of Quality Technology 43*, 3 (2011), 169–195.

[12] Crane, K., Schröder, P., and Pinkall, U. Spin Transformations of Discrete Surfaces. *ACM Transactions on Graphics 30*, 4 (2011), 1–10.

[13] Crane, K., Weischedel, C., and Wardetzky, M. Geodesics in heat: A new approach to computing distance based on heat flow. *ACM Transactions on Graphics 32*, 5 (sep 2013), 1–11.

[14] De Leo, S., and Scolarici, G. Right eigenvalue equation in quaternionic quantum mechanics. *Journal of Physics A: Mathematical and General 33*, 15 (apr 2000), 2971–2995.

[15] Desdouits, N., Nilges, M., and Blondel, A. Principal Component Analysis reveals correlation of cavities evolution and functional motions in proteins. *Journal of Molecular Graphics and Modelling 55* (2015), 13–24.

[16] Godin, G., Rioux, M., and Baribeau, R. Three-dimensional registration using range and intensity information. In *Videometrics III* (oct 1994), S. F. El-Hakim, Ed., vol. 2350, SPIE, pp. 279–290.

[17] Grosgeorge, D., Petitjean, C., Dacher, J. N., and Ruan, S. Graph cut segmentation with a statistical shape model in cardiac MRI. *Computer Vision and Image Understanding 117*, 9 (2013), 1027–1035.

[18] Hamilton, W. R. On quaternions, or on a new system of imaginaries in algebra. *The London, Edinburgh and Dublin Philosophical Magazine and Journal of Science* (1843).

[19] Heeren, B., Paulus, S., Goldbach, H., Kuhlmann, H., Mahlein, A. K., Rumpf, M., and Wirth, B. Statistical shape analysis of tap roots: a methodological case study on laser scanned sugar beets. *BMC bioinformatics 21*, 1 (jul 2020), 335.

[20] Hoyet, L., Ryall, K., McDonnell, R., and O'Sullivan, C. Sleight of hand: perception of finger motion from reduced marker sets. In *Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games* (New York, NY, USA, 2012), I3D '12, ACM, pp. 79–86.

[21] Huang, Z., Yao, J., Zhong, Z., Liu, Y., and Guo, X. Sparse Localized Decomposition of Deformation Gradients. *Computer Graphics Forum 33*, 7 (oct 2014), 239–248.

[22] Jacobson, A., et al. gptoolbox: Geometry processing toolbox, 2018. Available at: http://github.com/alecjacobson/gptoolbox/.

[23] Jia, Z., Wei, M., and Ling, S. A new structure-preserving method for quaternion Hermitian eigenvalue problems. *Journal of Computational and Applied Mathematics 239*, 1 (feb 2013), 12–24.

[24] Jolliffe, I. T. Principal Component Analysis, Second Edition. *Encyclopedia of Statistics in Behavioral Science 30*, 3 (2002), 487.

[25] Karni, Z., and Gotsman, C. Compression of soft-body animation sequences. *Computers and Graphics (Pergamon) 28*, 1 (2004), 25–34.

[26] Kendall, D. G. Shape manifolds, procrustean metrics, and complex projective spaces. *Bulletin of the London Mathematical Society 16*, 2 (1984), 81–121.

[27] Kókai, I., Finger, J., Smith, R. C., Pawlicki, R., and Vetter, T. Example-based conceptual styling framework for automotive shapes. *Sketch-Based Interfaces and Modeling 2007 - ACM SIGGRAPH/Eurographics Symposium Proceedings 1* (2007), 37–44.

[28] Lang, F.-n., Zhou, J.-l., Cang, S., Yu, H., and Shang, Z. A self-adaptive image normalization and quaternion PCA based color image watermarking algorithm. *Expert Systems with Applications 39*, 15 (nov 2012), 12046–12060.

[29] Le Bihan, N., and Sangwine, S. J. Quaternion principal component analysis of color images. In *IEEE International Conference on Image Processing* (2003), vol. 1, pp. 809–812.

[30] Le Bihan, N., and Sangwine, S. J. Jacobi method for quaternion matrix singular value decomposition. *Applied Mathematics and Computation 187*, 2 (apr 2007), 1265–1271.

[31] Li, H., Yu, J., Ye, Y., and Bregler, C. Realtime facial animation with on-the-fly correctives. *ACM Transactions on Graphics 32*, 4 (2013).

[32] Li, Y., Wei, M., Zhang, F., and Zhao, J. A fast structure-preserving method for computing the singular value decomposition of quaternion matrices. *Applied Mathematics and Computation 235* (may 2014), 157–167.

[33] Liu, H. T. D., Jacobson, A., and Crane, K. A Dirac Operator for Extrinsic Shape Analysis. *Computer Graphics Forum 36*, 5 (aug 2017), 139–149.

[34] Liu, Y., Li, G., Wang, Y., Nie, Y., and Mao, A. Discrete shell deformation driven by adaptive sparse localized components. *Computers and Graphics (Pergamon) 78* (2019), 76–86.

[35] Mahmood, N., Ghorbani, N., F. Troje, N., Pons-Moll, G., and Black, M. J. Amass: Archive of motion capture as surface shapes. In *The IEEE International Conference on Computer Vision (ICCV)* (Oct 2019).

[36] Meyer, A. R. F., and Anderson, M. Key Point Subspace Acceleration and Soft Caching. *ACM Trans. Graph 26* (2007), pages.

[37] Mirzaalian, H., Lee, T. K., and Hamarneh, G. Learning features for streak detection in dermoscopic color images using localized radial flux of principal intensity curvature. In *2012 IEEE Workshop on Mathematical Methods in Biomedical Image Analysis* (jan 2012), IEEE, pp. 97–101.

[38] Neumann, T., Varanasi, K., Theobalt, C., Magnor, M., and Wacker, M. Compressed manifold modes for mesh processing. *Eurographics Symposium on Geometry Processing 33*, 5 (aug 2014), 35–44.

[39] Neumann, T., Varanasi, K., Wenger, S., Wacker, M., Magnor, M., and Theobalt, C. Sparse localized deformation components. *ACM Transactions on Graphics 32*, 6 (nov 2013).

[40] Papaioannou, A. *Component analysis of complex-valued data for machine learning and computer vision tasks*. PhD thesis, Imperial College London, 2017.

[41] Papaioannou, A., Antonakos, E., and Zafeiriou, S. Complex representations for learning statistical shape priors. In *25th European Signal Processing Conference, EUSIPCO 2017* (oct 2017), vol. 2017-Janua, Institute of Electrical and Electronics Engineers Inc., pp. 1180–1184.

[42] Pomerleau, F., Colas, F., and Siegwart, R. *A Review of Point Cloud Registration Algorithms for Mobile Robotics*, vol. 4. Now Publishers, 2015.

[43] Rodman, L. *Topics in quaternion linear algebra*, vol. 9781400852. 2014.

[44] Romero, J., Tzionas, D., and Black, M. J. Embodied hands: Modeling and capturing hands and bodies together. *ACM Transactions on Graphics, (Proc. SIGGRAPH Asia) 36*, 6 (Nov. 2017).

[45] Rusinkiewicz, S., and Levoy, M. Efficient variants of the ICP algorithm. *Proceedings Third International Conference on 3-D Digital Imaging and Modeling* (2001), 145–152.

[46] Sangwine, S. J., and Le Bihan, N. Quaternion Toolbox for Matlab®. [Online], 2005. Software library available at: http://qtfm.sourceforge.net/.

[47] Sangwine, S. J., and Le Bihan, N. Quaternion singular value decomposition based on bidiagonalization to a real or complex matrix using quaternion Householder transformations. *Applied Mathematics and Computation 182*, 1 (nov 2006), 727–738.

[48] Sassen, J., Hildebrandt, K., and Rumpf, M. Nonlinear Deformation Synthesis via Sparse Principal Geodesic Analysis. *Computer Graphics Forum 39*, 5 (aug 2020), 119–132.

[49] Shi, L., and Funt, B. Quaternion color texture segmentation. *Computer Vision and Image Understanding 107*, 1-2 (jul 2007), 88–96.

[50] Shi, L., Funt, B., and Hamarneh, G. Quaternion color curvature. In *Final Program and Proceedings - IS and T/SID Color Imaging Conference* (2008), pp. 338–341.

[51] Sirovich, L. Turbulence and the dynamics of coherent structures. I: Coherent structures. *Quarterly of Applied Mathematics 45*, 3 (1987), 561–571.

[52] Sumner, R. W., and Popović, J. Deformation transfer for triangle meshes. In *ACM SIGGRAPH 2004 Papers, SIGGRAPH 2004* (2004), pp. 399–405.

[53] Tena, J. R., De La Torre, F., and Matthews, I. Interactive region-based linear 3D face models. *ACM Transactions on Graphics 30*, 4 (2011), 76.

[54] The Ohio State University. ACCAD MoCap. Retrieved from https://accad.osu.edu/research/motion-lab/mocap-system-and-data.

[55] van Kaick, O., Zhang, H., Hamarneh, G., and Cohen-Or, D. A survey on shape correspondence. *Eurographics Symposium on Geometry Processing 30*, 6 (2011), 1681–1707.

[56] von Radziewsky, P., Eisemann, E., Seidel, H., and Hildebrandt, K. Optimized subspaces for deformation-based modeling and shape interpolation. *Computers & Graphics 58* (aug 2016), 128–138.

[57] Wang, M., and Ma, W. A structure-preserving algorithm for the quaternion Cholesky decomposition. *Applied Mathematics and Computation 223* (oct 2013), 354–361.

[58] Wang, Y., Li, G., Zeng, Z., and He, H. Articulated-Motion-Aware Sparse Localized Decomposition. *Computer Graphics Forum 36*, 8 (dec 2017), 247–259.

[59] Weise, T., Li, H., Van Gool, L., and Pauly, M. Face/off: Live facial puppetry. In *Computer Animation, Conference Proceedings* (New York, New York, USA, 2009), ACM Press, pp. 7–16.

[60] Wu, J., Zhou, Z., Gao, B., Li, R., Cheng, Y., and Fourati, H. Fast Linear Quaternion Attitude Estimator Using Vector Observations. *IEEE Transactions on Automation Science and Engineering 15*, 1 (jan 2018), 307–319.

[61] Xu, X., and Guo, Z. Multispectral palmprint recognition using quaternion principal component analysis. In *Emerging Techniques and Challenges for Hand-Based Biometrics, ETCHB 2010* (2010).

[62] Xu, Y., Yu, L., Xu, H., Zhang, H., and Nguyen, T. Vector sparse representation of color image using quaternion matrix analysis. *IEEE Transactions on Image Processing 24*, 4 (apr 2015), 1315–1329.

[63] Yu, L., Xu, Y., Xu, H., and Zhang, H. Quaternion-based sparse representation of color image. In *Proceedings - IEEE International Conference on Multimedia and Expo* (2013).

[64] Yu, Y., Zhang, Y., and Yuan, S. Quaternion-based weighted nuclear norm minimization for color image denoising. *Neurocomputing 332* (mar 2019), 283–297.

[65] Zhang, F. Quaternions and matrices of quaternions. *Linear Algebra and Its Applications 251* (jan 1997), 21–57.

[66] Zhang, L., Snavely, N., Curless, B., and Seitz, S. M. Spacetime faces: High resolution capture for modeling and animation. In *ACM SIGGRAPH 2004 Papers, SIGGRAPH 2004* (2004), pp. 548–558.

[67] Zou, H., Hastie, T., and Tibshirani, R. Sparse principal component analysis. *Journal of Computational and Graphical Statistics 15*, 2 (2006), 265–286.