Vector Graph Representation for 3D Meshes and Algorithms for Mesh Deformation, Enveloping, Interpolation and Morphing

by

PRASHANT MANSHUKHBHAI DOMADIYA
201521010

A Thesis Submitted in Partial Fulfilment of the Requirements for the Degree of

DOCTOR OF PHILOSOPHY
to

Dhirubhai Ambani Institute of Information and Communication Technology

April, 2021

Declaration

I hereby declare that

i) the thesis comprises of my original work towards the degree of Doctor of Philosophy at Dhirubhai Ambani Institute of Information and Communication Technology and has not been submitted elsewhere for a degree,

ii) due acknowledgment has been made in the text to all the reference material used.

Prashant Manshukhbhai Domadiya

Certificate

This is to certify that the thesis work entitled VECTOR GRAPH REPRESENTATION FOR 3D MESHES AND ALGORITHMS FOR MESH DEFORMATION, ENVELOPING, INTERPOLATION AND MORPHING has been carried out by PRASHANT MANSHUKHBHAI DOMADIYA for the degree of Doctor of Philosophy at *Dhirubhai Ambani Institute of Information and Communication Technology* under our supervision.

Suman Mitra

Prof. Suman K Mitra
Thesis Supervisor

Prof. Pratik Shah (IIIT, Vadodara)
Thesis Co-Supervisor

Acknowledgments

During Ph.D., many people have contributed something directly or indirectly to my work. I grab this opportunity to thank them. First of all, I thank to the almighty to give me an opportunity to pursue my dreams. I thank to my both guides, *Prof. Pratik Shah (IIITV)* and *Prof. Suman Mitra (DAIICT)* for their valuable guidance and support. Both have gave me full freedom to pursue my work and also taught me valuable lessons of life. With them, I've also learnt the other valuable skills such as academic management and teaching. I thank to Prof. Aditya tatu, Prof M.V. Joshi and Dr. Sumukh Bansal for their their suggestions and comments. I also thank to anonymous reviewers for their time and valuable comments.

I can't express my gratitude towards to my parents, *Manshukhbhai* and *Nirmalaben*, in words. Thank you, both of you to let me pursue my dreams. A special thank to my wife *Mamta* to stand by me every time on my odds and even during Ph.D.. I thank my family members Mihir, Grishma, Aradhya, Payal, Yogesh, Bhavesh, Kailash, Hetal, Hardik, Kavya, Nakul, Nandan, Tanish and Trisha for their moral support. I thank to my mother in-law, *Jayshriben* and my father in-law late *Pravinbhai* for their support.

I thank *TATA Consultancy Services Research Scholar Program* (TCS-RSP) to grant me the scholarship for four years. Without their support, it is hard for me to fulfil my financial requirements during Ph.D.. I thank to Blender developers to create a wonderful and creative open source software that helps me to realize my work. I express my gratitude towards the Python developers, Stack Overflow, Stack Exchange and Blender developer community for their help during implementation.

I thank to my joyful friends Kamal, Sumukh, Parth, Jignesh and Rishikant to

be with me all the time. I also thank to my seniors Dr. Gitam Shikkenawis, Dr. Ashish Phophaliya, Dr. Milind Padalkar, Dr. Purvi Koringa and Dr. Vandana for their motivation and help. I also thank to my colleagues Miral, Anjali, Tusar, Sujatha and Pranav to be my tea buddies. I also thank to my friends Rathin, Nikita Bhungaliya, Bhavika, Nilay, Bharti, Nikita Joshi and Sagar to be my hangout partners. I thank to DAIICT support staff for their help.

I greatly thank to DAIICT cricket team, Om, Amrit, Saheb, Archit, Yash, Nandan, Kush, Yogesh, Jaimin, Arvind, Nisarg Samarth Parikh, Samarth Kothari, Kenil, Ajay, Keval, Jeel, Siddharthsinh, Bhargav, Rutul, Khushil, Aman, Shyam, Tirth, Yuvraj, Manthan and other members to be considering me as part of the team. I can pursue my work joyfully and continuously for five years because the cricket helps me to be healthy, mentally and physically. Without all of you, that might not be possible.

Contents

Abstract

The performance of the tools developed for graphics and animation depends on the underlying representation of the geometric model of the objects. The geometric model in turn describes the geometry of the real world object. In this work, we propose a unified framework for mesh editing based on the Vector Graph (VG) representation of the geometric model. The VG representation is an instance of the barycentric coordinates. It represents various geometric models such as triangular, quad and hybrid meshes as a collection of vectors. For various mesh editing applications, the deformations of the geometric models are considered in terms of the deformation of the VG representation. For this task, the deformation of a vector of the VG is modelled in terms of rotation and scaling which are orientation preserving transformations. Moreover, the composition is commutative and forms a matrix group which is also a smooth manifold. The simplicity of the VG allows for the straight forward formulations of mesh editing approaches, such as Deformation Transfer, Enveloping, Interpolation and Morphing. For the proposed method, the major computation is involved in solving an overdetermined system of linear equations with appropriate constraints. The efficient methods exist for solving this systems at a large scale. This ensures the real-time execution of the algorithm for large geometric models.

In this work, the proposed algorithms for Deformation Transfer, enveloping, interpolation and morphing using the VG representations are compared qualitatively and quantitatively with state-of-the-art methods. In addition, we have proposed a face selection algorithm which reduces the computational complexity without compromising on the quality of editing. This can be applied to other face based formulations also. Apart from computational efficiency, based on the experiments, it is encouraging to note that (a) the proposed Deformation Transfer method preserves the shape and the geometric details of the target mesh for a wide range of deformations, (b) the enveloped mesh using proposed enveloping method can be edited in real-time to generate complex poses without artifacts, (c) qualitatively the proposed interpolation scheme performs at par with the existing approaches and is computationally real-time compared to others and (d) proposed morphing is qualitatively similar to the other approaches.

In this work, our objective is two fold. We first propose computationally efficient algorithms which ensure smooth and realistic deformations of the mesh for various applications and then implement these algorithms in Blender as prototypes for interactive user experience. The simplicity of the VG is reflected in the implementation of all the proposed algorithms in Blender 2.82. The created Blender add-ons are easy to use. We have included a detailed user manual for the developed add-ons as a part of the thesis.

List of Tables

List of Figures

CHAPTER 1

Introduction

In graphics and animation, the representation of geometric objects plays a crucial role in manipulating their geometry. In practice, the data related to the object of interest is acquired from scanners (sensors) and is stored as a collection of 3D points, traditionally referred to as a point cloud. Based on some geometric criteria, these points are connected to form edges and faces (and volume elements). A suitable collection of points, edges, and faces (volume elements) describes a geometric model of the object. If the faces are triangles, then it is called a triangular mesh. Similarly, if the mesh is a collection of different types of n-gons, then it is called a hybrid mesh. The choice of model depends on the application; for example, skeletons are used in action and activity recognition. The triangular, quad, and hybrid meshes are used in animation and computer graphics.

In computer graphics, an animator manipulates the geometry of a mesh to create an animation sequence. Each element of a mesh, such as vertices and faces, should be manipulated during this process. A manipulation approach should induce smooth and realistic deformation of the mesh, and it should also be computationally efficient. Hence, mesh manipulation algorithms are designed based on a trade-off between qualitative and computational performances. In literature, two major approaches are proposed to perform mesh manipulation tasks. In the first approach, the mesh is manipulated directly in its original domain ($\mathbb{R}^3$) by manipulating its elements with given constraints. This approach leads to smooth and realistic deformation of the mesh, but it creates an undue overload on computation due to iterative optimization process. In another approach, a suitable differential representations [13] of a mesh is defined. Each element of this dif-

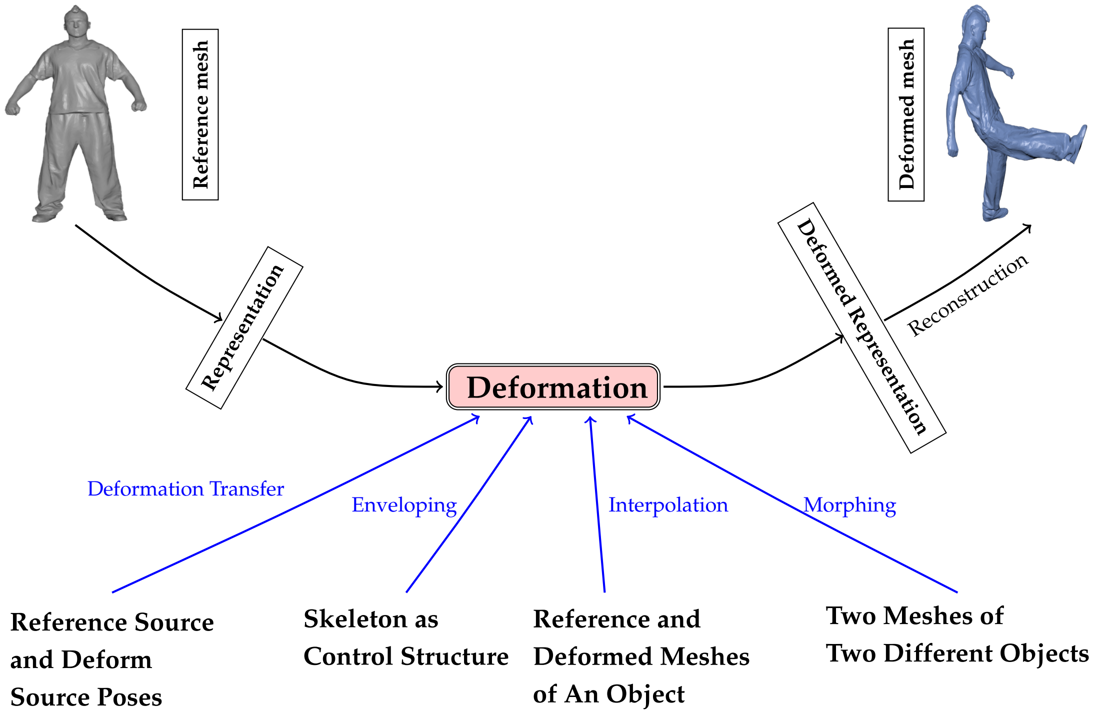

Reference mesh
Deformed mesh
Representation
Deformation
Deformed Representation
Reconstruction
Deformation Transfer
Enveloping
Interpolation
Morphing
Reference Source and Deform Source Poses
Skeleton as Control Structure
Reference and Deformed Meshes of An Object
Two Meshes of Two Different Objects

Figure 1.1: The illustration of deformation based approach for the mesh manipulation. The deformation is computed and manipulated differently for various applications.

ferential representation is then defined resulting in desired manipulation of the mesh. A variety of representations are reported in literature to compute either the face deformation [1, 14, 15] or the vertex deformation [10, 16, 17, 18]. This process is illustrated in Figure 1.2. The element deformation is manipulated differently depending upon the applications and representations. From the deformed representation, the mesh can easily be reconstructed back by solving a system of linear equations with suitable constraints.

In this work, we explore four mesh editing tasks; Deformation Transfer, Enveloping, Interpolation and morphing using *Vector Graph*. These editing tasks can be used in many computer graphics and computer vision applications. The Deformation transfer and enveloping can be used in gaming, animation, motion capture, sport training, action recognition etc.. The interpolation can be used in gaming, motion estimation in occlusion condition, object tracking etc.. The morphing can be used in animation, object transformation visualization, understanding evolution of a shape etc.. Various challenges involved in these tasks and their wide range of applications motivate us to present our contribution. In Deformation Transfer method, the geometric details of the target should be preserved during transfer of deformation from source meshes. In the enveloping task, the mesh should be related with aligned control structure such that the deformed control structure should produce smooth deformed mesh and the defined relationship should offer compatibility with existing hardware. Moreover, the enveloped mesh should be deformed in real-time. The interpolation and morphing method should produce smooth and realistic transition from one mesh to another. We try to tackle challenges involve in these applications using *Vector Graph* representation as a tool. We introduce all the editing tasks in the following sections.

1.1 Deformation Transfer

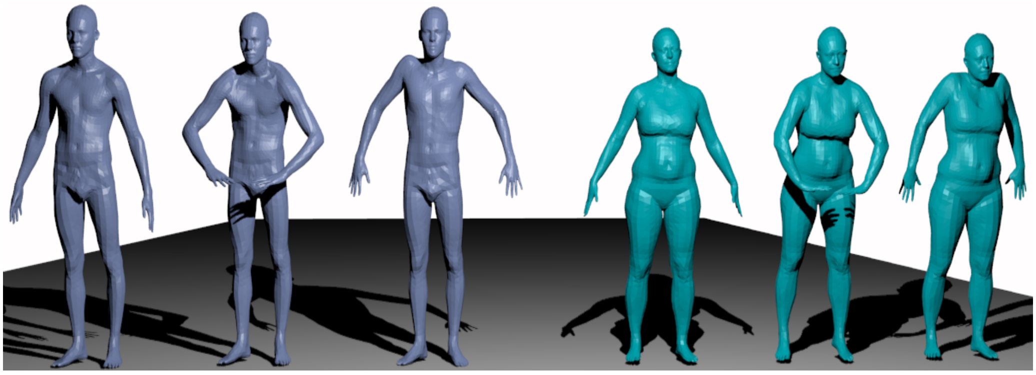


Figure 1.2: Demonstration of the deformation transfer : The action of a man (Source) is transferred to the woman (Target).

The Deformation Transfer is a process by which the deformation of a source mesh is transferred to the target mesh. As an example of Deformation Transfer, in Figure 4.1, the poses of a female actor (on the right) are generated according to the poses of a male actor (on the left). It generates the target poses that are deformed similar to the corresponding source poses with minimum human intervention. In general, the Deformation Transfer is performed in two phases; first, a dense set of correspondence between the reference source and the target is identified from the sparse set of user selected markers, and in the second phase, the deformation of the source is transferred to the target with the help of the known correspondence.

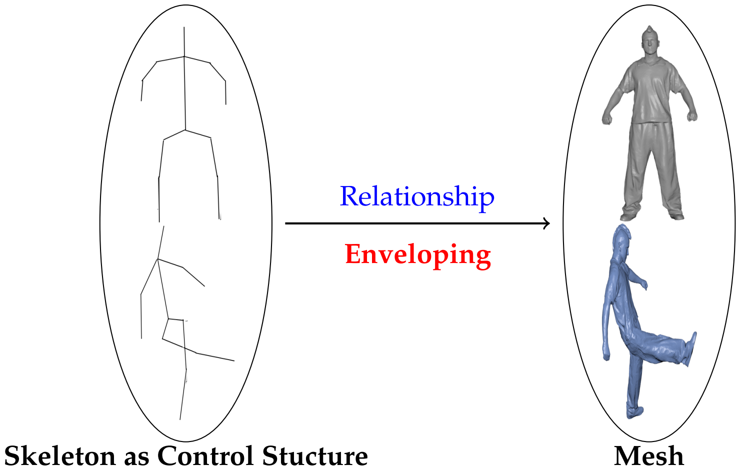


Figure 1.3: Establishing the relationship between the mesh and the control structure is known as *Enveloping*

As shown in Figure 1.2, the deformation computed from the source is applied to the target reference mesh in the Deformation Transfer application. A Deformation Transfer method should ensure that the geometric details of the target mesh are preserved during deformation transfer.

In literature, the methods reported for deformation transfer are primarily for triangular meshes and are not directly applicable to skeleton and hybrid meshes. However, these methods can be applied to quad or hybrid meshes by splitting quad faces into triangles. Moreover, various representations such as Deformation Gradient [1], Lie body [2] are also employed to compute the triangle deformation as a linear transformation. The linear transformation is not an orientation preserving transformation. Hence, the geometric details of the target can not be preserved when the source mesh is deformed with large shear. In Deformation Transfer problem, the source and the target are different, and so are their motion properties such as the temporal positions of poses and motion trajectories. During Deformation Transfer, these properties should be adjusted appropriately for the target sequence. However, existing methods requires some amount of user interaction to adjust for these properties.

1.2 Enveloping

Deforming a mesh by deforming the associated control structure based on an established relationship is a standard and an efficient technique practised by skilled animators for years. Generally, the control structure includes a skeleton, point handles, cages, region handles or combination of them. Establishing the relationship between the mesh and the control structure is known as *Enveloping* as illustrated in Figure 1.3. The enveloped mesh can be deformed by deforming the associated control structure. As shown in Figure 1.2, the deformation of the enveloped mesh is manipulated by manipulating the skeleton deformation. The established relationship should produce smooth mesh deformation, encourage real-time performance, and offer compatibility with existing hardware. Because of simplicity, real-time performance, and compatibility with high-performance computing hardware, the Linear Blend Skinning (LBS) [6] is the widely used enveloping technique. In this, a linear relationship is established between the mesh deformation and the control structure deformation. During the editing process, each vertex of the enveloped mesh is deformed independently to its neighbours by deforming the skeleton bones. Such individualistic vertex deformation may not preserve the mesh properties such as edge length. This relationship, if not preserved, results in inconsistency artifacts after deformation.

In other approaches, the control structure and the vertex deformation are related indirectly via differential representations. The vertex deformation which is computed using a differential representation, is further constrained to ensure smooth deformation of the mesh. However, these constraints introduce nonlinearity which not only affects the computational time but also makes an enveloping method hard to be compatible with the high-performance computing hardware. For real-time performance, the mesh is segmented into patches. It is expected that the vertices of a patch undergo similar deformation. The patches are generated based on similarities using machine learning algorithms such as K-means [8] and mean-shift [19]. The qualitative performance of this approach depends upon the accuracy of the segmentation. Moreover, the segmentation method may depend on the user input, i.e. the number of segments. As stated earlier, all the enveloping techniques usually trade deformation quality for computation.

1.3 Interpolation

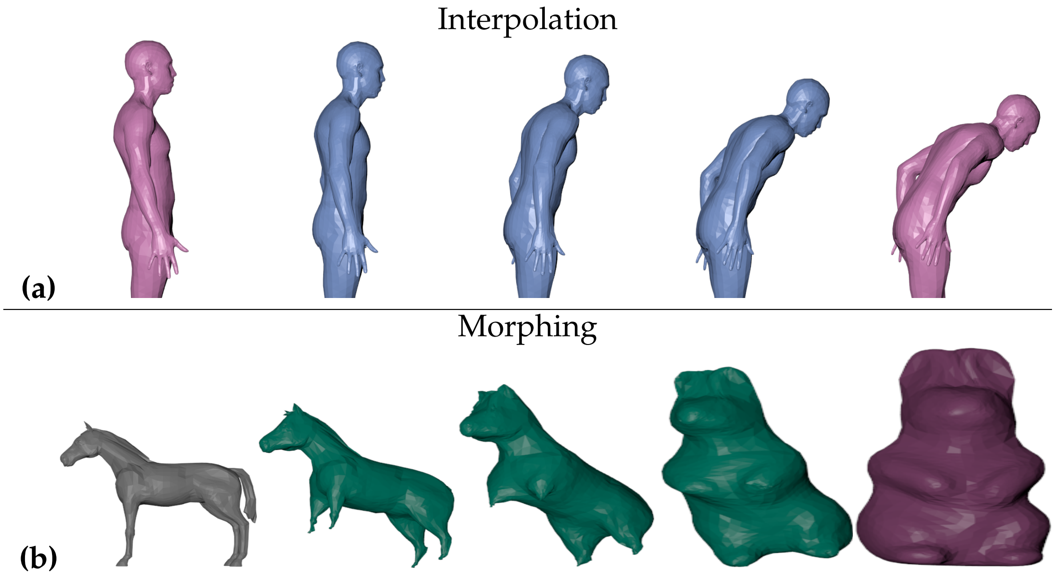


Figure 1.4: **(a)** The intermediate poses of bending action of a man are generated using a *interpolation* method. **(b)** The horse is morphed to bunny. The intermediate poses are generated using a *morphing* method.

An animation sequence is a set of consecutive poses of a mesh (an animation character), and the transition between two consecutive poses is smooth and continues. Generating such a set is a tedious task. Hence, an animator creates a small set of poses (keyframes) by deforming the reference mesh. The intermediate poses between two keyframes are then generated by one of the shape interpolation methods. In morphing, the intermediate poses are generated between two meshes of different geometric objects. As shown in Figure 1.2, the deformation is interpolated and applied to the reference mesh for the interpolation and morphing applications. The interpolation and morphing are demonstrated in Figure 1.4 as well. The shape interpolation method should generate realistic and natural-looking deformation on the interpolated poses, and it should also be computationally real-time at the same time.

For some shape interpolation approaches, the meshes are represented in a the high dimensional shape space, and then a trajectory between a pair of points is approximated in the shape space. A point on the trajectory is an interpolated pose. The trajectory can be approximated in form of a geodesic. The linear interpolation is the simplest and fastest method, but it suffers from shrinkage problem. This problem can be addressed by introducing constraints (as a non-linearity) into the trajectory approximation process. However, the added constraints make the trajectory approximation process more complex and computationally demanding. Moreover, this approach can't be adopted directly for multi-pose interpolation and the morphing.

In another approach, the shape interpolation is viewed as the interpolation of the deformation of the mesh. In this case, the mesh is first represented by a differential representation to compute the element (vertices or faces) deformations. The element deformation forms a manifold, and the shape interpolation is performed on the tangent space of the manifold. This approach generates smooth and natural interpolated poses, and it can also be adopted directly for multi-pose interpolation and the morphing. In both the applications, the deformation of each element is processed independently to achieve the desired mesh deformation.

1.4 Problem Identification

We, in this work, propose a unified differential representation for various geometric models. It is suitable for various mesh editing tasks such as deformation transfer, enveloping, interpolation, and morphing. This representation is referred to as the *vector graph* (VG) representation. In VG, each face of a mesh is represented by a set of vectors, and each vector originates from the centroid of the face and points towards the corresponding vertex of that face. Since a mesh is a collection of faces, its VG representation is a collection of vectors. The VG is a universal representation in the sense that it represents various types of meshes such as triangular, quad, or hybrid by a collection of vectors. Moreover, from the VG, it is straight forward to reconstruct the corresponding mesh by solving an overdetermined system of linear equations. The reconstruction process helps us to deform the mesh by deforming its VG representation. The VG is deformed by deforming its vectors. In this work, the deformation of a vector is modelled as a composition of the orientation preserving transformations, i.e. rotation and scaling. This in turn help to preserve the geometric details of the target for a wide range of deformations in the Deformation Transfer application. Moreover, the composition is commutative and forms a matrix group which is a smooth manifold. Similar to other representations, the VG is a translation-invariant representation. The VG representation is described in details in chapter 3.

In this work, we have proposed two Deformation Transfer methods by computing the shape deformation and the pose deformation using the VG representation. The shape deformation is computed between the reference source and the reference target meshes. It is then transferred to deformed source pose to get deformed target pose. This approach is computationally efficient compared to the traditional approaches, but it fails to preserve the target shape for large deformation. It also fails to keep the planarity of the quad faces. The above limitations are overcome by using the pose deformation for the Deformation Transfer application. The pose deformation is computed between the reference, and the deformed source poses using the VG representation. It is then transferred to the target reference mesh via established correspondence to get deformed target pose. In this approach, the orientation preserving property of the vector deformation preserves the geometric details of the target for a wide range of deformations. Moreover, the added planarity constraint in the reconstruction process preserves the planarity of the quad faces. After performing the Deformation Transfer, the temporal properties of the target sequence are refined using *Poisson Interpolation*. The Deformation Transfer application is described in chapter 4.

Next, we have explored a data-driven enveloping method using the VG. Here, we work with the skeleton as a control structure. The skeleton deformation is a smooth manifold similar to the vector deformation manifold. During enveloping, a map between the skeleton deformation to the vector deformation is established by computing its parameters using a set of example poses available in the database. The enveloped mesh is then deformed by deforming the skeleton through the established map which leads to smooth and meaningful deformation of the mesh. The deformation of the enveloped mesh is computationally equivalent LBS [6] and qualitatively similar to non-linear methods. The enveloping application is described in chapter 5.

We have also explored the shape interpolation and morphing using the VG by interpolating vector deformation similar to the traditional approaches. However, the shape interpolation is computationally demanding due to element-wise operations. To address this issue, we exploit the approach used for enveloping. Here, the mesh segmented into patches serves as a low resolution structure. A map between patch deformation manifold and vector deformation manifold is established by computing its parameters using example poses in the dataset. First, the patch deformation is interpolated and then the established map maps it to the vector deformation for the shape interpolation. Since the dimension of the patch deformation manifold is relatively smaller, the proposed interpolation is computationally real-time and qualitatively comparable existing methods. The established map can also be used for the real-time deformation transfer application. The interpolation and morphing applications are described in chapter 6.

1.5 Our Contribution

- We have proposed a unified VG representation in this work.
- We have formulated each of the following mesh editing tasks in the VG framework. (a) Deformation Transfer, (b) Enveloping, (c) Interpolation and (d) Morphing.
- To reduce the computational time due to redundancy, we have proposed a face selection algorithm. This algorithm can be used with any face-based method for mesh editing to reduce the computational time without compromising on the quality.
- We show that the proposed method for Deformation Transfer preserves the

shape and the geometric details of the target for a wide range of mesh deformations. We also point out the major difference between the proposed method and the existing Deformation Transfer methods, namely the dot product property.

- With the results of experiments carries out, we show the effectiveness of the proposed enveloping scheme which can be used for real-time editing of meshes using the control structure.

- Further, we show that the proposed interpolation scheme is qualitatively comparable with state-of-the-art methods and is real-time.

- We have implemented all the proposed methods in Blender as Add-ons for an interactive experience. The Blender add-on user manual has been included as part of the thesis.

1.5.1 Publications

- Prashant Domadiya, Pratik Shah, Suman K Mitra, “Vector Graph Representation for Deformation Transfer Using Poisson Interpolation", IEEE Winter Conference on Applications of Computer Vision, Lake Tahoe, California, USA, March, 2018.

- Prashant Domadiya, Pratik Shah, Suman K Mitra, “Guided Deformation Transfer", ACM SIGGRAPH European Conference on Visual Media Production, London, UK, December, 2019.

1.5.2 Submitted

- Prashant Domadiya, Pratik Shah, Suman K Mitra, “Real-Time Enveloped Mesh Editing Using Vector Graph Representation", Submitted to ACM Transaction on Graphics, 29 July, 2020.

1.5.3 Under Preparation

- Prashant Domadiya, Pratik Shah, Suman K Mitra, “The Real-Time Mesh Interpolation, Deformation Transfer and Morphing".

1.6 Organization of the Thesis

In chapter 2, we present the detailed survey of the existing methods of Deformation Transfer, Enveloping, Interpolation and Morphing. We also include the survey of other applications such as motion editing, style transfer and path editing. We describe the Vector Graph representation in detail in chapter 3. In section 3.1, we describe the mesh reconstruction process from the VG. Moreover, we show by experiment that the vector graph is a consistent representation. We, in the section 3.2, propose the face selection algorithm to reduce the computational time. In section 3.3, we also present the two possible way of computing vector deformation in terms of composition of rotation and scaling transformations. We present various properties of the vector deformation composition in the section 3.4. These properties make the VG unique compared to other representations.

In chapter 4, we propose two approaches for Deformation Transfer using shape deformation and pose deformation. We present the proposed Deformation Transfer framework using the shape deformation in section 4.1 in details. In this section, we also describe the computation of the shape deformation. In section 4.1.2, we present Poisson interpolation with boundary conditions followed by guided deformation transfer in section 4.1.3. Here, we show that the proposed guided Deformation Transfer preserves temporal properties of the target mesh sequence. In section 4.1.4, we provide the experimental details, results, comparison with the state-of-the-art methods and limitations of this approach. We present the proposed Deformation Transfer framework using the pose deformation in section 4.2 in details. In this section, we also describe the computation of the pose deformation. In section 4.2.2, we present the importance of dot product property of the deformation to explore the Deformation Transfer for wide range of deformation. We show that the added planarity constraint in the reconstruction process can preserve the planarity of the quad faces of the hybrid mesh in section 4.2.3. We also show that the Poisson interpolation can also be employed for pose deformation in section 4.2.5.

We describe the proposed enveloping approach in chapter 5 in details. In section 5.1, we describe the proposed approach for control structure (skeleton) assignment to the reference mesh and computation of the bone deformation. We then define the vector and skeleton deformation groups in section 5.2. In section 5.3, we describe proposed enveloping process and establish a map between vector and skeleton deformation groups. After establishing this map, we formulate the mesh deformation associated with skeleton deformation in section 5.4. Here, we show that the relationship between skeleton and mesh deformations using the VG is similar to that of the LBS. Finally, in section 5.5 we compare proposed method with the four state-of-the-art methods in terms of quality of deformation and computational time.

In chapter 6, we propose two methods for the interpolation and morphing. We first describe the standard deformation based interpolation and morphing method using the VG and compare these methods with other methods in section 6.1. We then explore the learning based interpolation approach using the VG. In this approach, we first segment the meshes into patches and computation of the patch deformation as described in section 6.2. We then define a map between the vector and patch deformation groups and compute the map parameters in section 6.3. After establishing this map, we formulate the mesh deformation associated with patch deformation. In section 6.4 and 6.5, we formulate real-time interpolation and deformation transfer using established map. We also show that the interpolation and deformation transfer can be performed on morphed mesh using established map in section 6.6. Finally, in section 6.7 we compare proposed interpolation and deformation transfer methods with other state-of-the-art methods in terms of quality of deformation and computational time.

We implemented all the proposed methods in the Blender as add-ons using Python. We document the implementation as user manual in the chapter 7. In this chapter, we first describe system requirement, dependent Python libraries and basic setup procedure to use Blender add-ons. We then describe the step-by-step process to use all the add-ons developed for Deformation Transfer, enveloping, interpolation and morphing. Finally, we conclude the thesis in chapter 8 with possible directions towards the future work.

Chapter 2

Literature Survey

In literature, different representations of the geometric models have been proposed for different applications. For example, deformation gradient is used [1] for purpose of deformation transfer, Manifold Representation [20] (i.e. Lie Body) is used for Deformation Transfer [2] and for mesh editing [14]. In [10], Linear Rotation Invariant coordinates are used for interpolation task. It is important to note that a specific representation may work well for a task and may not be effective for others. Both Manifold Representation and Linear Rotation Invariant are computationally inefficient for enveloping and interpolation compared to the VG representation and representation used in Deformation Gradient [1]. For enveloping, Linear Blend Skinning (LBS) [6] and its variant DeepLBS [7] are widely used due to their simplicity and real-time behaviour. Since, in this work, we focus on the VG representation and wish to demonstrate its utility in all the above mentioned mesh editing tasks, we provide a brief review of the work carried out in Deformation Transfer, enveloping, interpolation, and morphing. The survey captures the salient points of the state-of-the-art algorithms.

2.1 Deformation Transfer

From literature, it is evident that Deformation Transfer has attracted considerable attention in computer graphics and computer vision. A variety of geometric models of a real-world object such as skeleton and hybrid meshes make the Deformation Transfer problem challenging. Adjusting temporal properties (position and motion trajectory) and spatial properties (shape) of the target during Deformation Transfer are two of the major challenges. A survey of the work done in the field of deformation transfer is presented next.

Sumner and Popović in [1] proposed a deformation transfer method for triangulated meshes. Initially, with little user interaction, a dense set of correspondence map is established between target and source meshes vertices using the selected sparse correspondence set. Next, the deformation between corresponding triangles of reference source mesh and deformed source mesh is computed independently for each triangle. The deformations are composed of affine transformations. These deformations of selected source triangles are then transferred to the entire target mesh via a correspondence map established earlier with consistency constraints to preserve the topology of the target. A face is represented by two different vectors and a normal. This representation can not represent the line segment, quad, and n-gon faces. This method preserves spatial property (shape) of the target but requires user interaction to adjust for the temporal properties. Zhou et al., in [21], extends the above method to multi-component triangular meshes.

In [5], a semantic deformation transfer method is proposed. It preserves semantic motion property of the target (i.e., transferring walking action of a human to flamingo). To begin with, all meshes of the source and the target are represented by the patch-based Linear Rotation Invariant coordinated [10]. Using standard linear algebra techniques like interpolation and projection, a new target pose is reconstructed. This method can handle large deformations and noisy meshes. However, the representation is specific for triangular meshes only, and full user interaction is required to adjust for the temporal properties.

A geometric modelling method is proposed in [22]. The authors suggest projecting a shape (mesh) in a higher dimensional space called the shape space. For the Deformation Transfer problem, a geodesic is generated based on the Riemannian metric which preserves isometric property in shape space between source reference pose and its deformed pose. This method is also extended to the quad mesh by stiffening quad face into two triangles. However, stiffening by the edge at a diagonal of the quad face adds a constraint to the deformation of vertices of added edge. Such constraint restricts vertices to deform w.r.t opposite diagonal.

Moreover, as mentioned in [22], self-collision is difficult to avoid during Deformation Transfer without user intervention.

Shabayek et al. in [2] extended the Lie body manifold representation of the triangular mesh presented in [14] for deformation transfer. In contrast to the mesh in Euclidean space, its Lie body manifold representation has certain advantages: it requires a minimum degree of freedom, and the non-physical deformation is not involved due to the positive determinant of the deformation. However, this method is limited to the triangular meshes and requires full user interaction to adjust for the temporal properties.

In [23], Anguelov et al. propose a data-driven shape completion method that converts partial mesh/pose to complete mesh/pose retaining the shape. The pose model handles the pose variations, and the shape model handles the shape variations. By varying PCA parameters learned from different example shape models, a set of shape models is generated to get the desired shape variations. The method accomplishes two sequential tasks: (1) given a partial scan of a person whose pose is not present in the database; a whole body surface completion is carried out, and (2) applying this method on time series motion capture data to animate the human action. This method adjusts the temporal and spatial properties of the target. However, it can't handle for skeleton, quad, and n-gon faces because a face is represented by two different vectors and a normal.

Young et al. proposed a dual-domain deformation transfer method in [24]. In this method, authors suggest converting triangular meshes into dual meshes [25]. The affine transformation between corresponding vertices of reference and deformed dual meshes of the source are computed. The affine transformations computed at each dual vertex are transferred to reference dual mesh of the target via correspondence. The resultant deformed dual mesh of the target is converted back to the original triangular mesh. The dual mesh is not defined for the skeleton, and user interaction is required to adjust the temporal properties of the target sequence.

The mesh may consist of a group of unorganized triangles i.e., polygon soups and multicomponent meshes, or volumetric representation as tet-mesh. Ben-

Chen in [26] proposed a Deformation Transfer method suitable for such representations as tetrahedral meshes, polygon soup, multicomponent meshes etc.. In this method, a mesh is embedded in a space (a cage) which is made up of uniformly structured triangles. Deforming the space also deforms the unstructured mesh is the key idea of this method. Initially, the space deformation between the spaces, into which reference and deformed source meshes are embedded, is approximated by variational harmonic map (VHM) using harmonic functions. The space deformation is transferred to the target space through a sparse set correspondence assigned by the user. The deformation of the target space also deforms the target reference mesh resulting in deformed target mesh. Similarly, Chen et al. in [27] also propose a cage based Deformation Transfer method for organized and unorganized triangular meshes. Unlike the VHM, Greens' coordinates [28] have been used to compute the deformation. Here, the source sequence is a motion capture data. Hence, the deformation is not transferred from the source, but it is computed at the target, considering motion capture data as handles. These methods are specific to organized or unorganized triangular and tetrahedron meshes. It could not adjust the temporal properties of the target.

Yu at el. [29] proposed a mesh editing method based on Poisson interpolation. This method is used for interactive deformation computation, mesh cloning, and smoothing. Xu et al. in [11] extended this work to mesh morphing. The Poisson equation is a partial differential equation that describes the potential field on charge distribution. It is not limited to Physics but has found applications in image processing, graphics, and geometry processing. Patrick at el. in [30] proposed an image editing method using Possison interpolation. By varying the guidance field, authors have shown various types of editing, such as seamless cloning, monochrome transfer, illumination change, feature exchange, etc..

2.2 Enveloping

Linear Blend Skinning (LBS) [6] is one of the widely used enveloping techniques, and it is implemented in almost all the game engines and graphics tools. In LBS, the deformation of each vertex of a mesh is modeled as a linear combination of the deformation of skeleton bones. Since LBS is linear in nature, it can easily be implemented on the high-performance computing hardware. Though it is fast and simple, it introduces artifacts such as *candy wrapper, joint collapse, and volume loss*. Moreover, since the vertices are deformed independently by deforming skeleton bones, the inter-vertex relationship may change during the mesh deformation. This leads to the inconsistency artifacts in the deformed mesh. Wang et al. in [31], modified vertex-bone deformation relationship defined in the LBS by replacing single weight assigned to each bone deformation with multiple weights. This multi-weight approach overcomes *candy-wrapper and joint collapse* with an increase in the computational cost. Mohr et al. in [32] further extend the LBS by approximating additional bones for the skeleton to resolve the muscle bulging issue. In addition, the affinity constraint imposed on weights overcomes the weight over-fitting problem.

Kavan et al., in [33], replaces affine transformations representing the bone and the vertex deformations in the LBS by rigid transformations to address the inconsistency artifacts. In this method, the bone and vertex rotations are represented by quaternions. Next, the vertex quaternion is approximated by blending quaternions of skeleton bones with convex weights. The choice of center of a vertex rotation isn't arbitrary; rather, it is predicted by solving the least squares optimization problem. This approach can avoid artifacts such as joint collapse and "candy-wrapper" again with an increased computational requirement. Approximating center of rotation for each vertex individually is a computationally expensive task. Kavan et al. [34] use dual quaternions to represent the rigid transformations. The dual quaternion is coordinate-invariant; moreover, the convex combination of a set of dual quaternions is also a dual quaternion. Hence, the vertex deformation approximated using a rigid transformation of skeleton bones is a rigid transformation. The dual quaternion introduces non-linearity in computation; hence it is computationally demanding.

Mukai et al. [35] extend the LBS by introducing helper bone in addition to the skeleton to simulate muscle bulge. In [7], authors extend the LBS by approximating non-linear part of a vertex deformation by a neural network. Introducing sparseness followed by simplification in the network improvises the computation efficiency of the network. The results show the increment in the quality of the enveloping. However a large amount of data is needed to train the neural network. To summarize, the vertex deformation is related to the skeleton bone deformation forsaking inter-vertex properties such as edge length in the LBS and its variants. Such individualistic vertex deformation introduces the inconsistency artifacts during the mesh deformation.

In [8], the energy function defined in [36, 37, 38] is combined with the energy function defined for the LBS. Further, the vertex deformation computed using the differential representation is constrained to be As-Rigid-As-Possible (ARAP). The ARAP constraint preserves the edge lengths and hence avoids the inconsistency artifacts. With this constraint, the mesh can be deformed using an abstract set of skeleton bones as well. Wang et al. [9] proposed a non-linear enveloping method to address the “candy wrapper" and muscle bulging issues inherent to the LBS. In this, the deformation of a triangular face of a mesh is computed as a linear transformation using the deformation gradient (DG) [1], and the mesh deformation is represented by deformation of all triangular faces collectively. The relationship between triangular deformation and skeleton bone deformation is established using non-linear rotation regression. Since the deformation gradient uses the vertex neighbors to compute the deformation, the smooth deformation on the mesh is achieved without inconsistency artifacts. In both cases, the presence of non-linearity imposes the additional computational requirement. It is improvised by segmenting the mesh into parts undergoing similar deformation and simplifying the mesh reconstruction process. It is obvious that the quality of the mesh deformation depends on the efficacy of the segmentation method and the number of segments. Identifying the appropriate number of segments is a challenging task. Moreover, computational overload increases with the number of segments.

In [19], authors have proposed a skinning method to deform thousands of meshes in real-time. The skinning process is split into two stages. First, the mean-shift algorithm [39, 40] is employed to segment out a mesh into patches. Each segmented patch (called proxy bone) contains a set of similarly deformed faces, and its deformation is computed using only strongly related triangles. Second, the bone-vertex relationship is established by assigning convex weights along with the sparsity constraint on the neighborhood. The mesh editing is difficult to organize due to unorganized bone hierarchy. This method is improved in [41] by merging both processes in one optimization problem with soft non-negativity, affinity, sparseness, and orthogonality constraints. Le and colleagues, in [42], further improvised the skinning method proposed in [19] by constraining the unified cost function with hard constraints. The qualitative and quantitative performances of these methods heavily depend upon the segmentation method to identify the appropriate number of proxy bones. However, it is difficult to solve the trade off between the quality of deformation and computational load.

2.3 Interpolation and Morphing

Various shape interpolation and morphing [43, 44, 37, 45] are proposed to generate intermediate poses. Alexa at el. [46] suggests an interpolation scheme for 2D and 3D triangular meshes. The deformation of a triangular face is computed as the linear transformation, which is further decomposed into rotation and shear matrices for the interpolation. Similarly, in [11], the linear transformation of a vertex is decomposed into rotation and shear matrices using QR decomposition for interpolation. In both cases, the rotation is interpolated by interpolating the rotational angle, and the shear is interpolated linearly. The mesh is then reconstructed by solving a constraint optimization problem. Since the deformation is computed for each face or vertex, the computational complexity increases with the resolution of the mesh. Moreover, both the methods [46, 11] suffer from artifacts when the rotation is large ($> 180^\circ$).

In [12], the author proposed a mesh interpolation and morphing method based on Lie body representation [14]. The linear transformation of a triangular face is decomposed into the rotation, scaling, and a positive definite upper triangular matrix. The large deformation is handled by identifying affected triangles.

Moreover, this interpolation method is directly adopted to perform the morphing between two different meshes. To deal with the large deformation, in [47], a vertex rotation is approximated from a set of relative rotations computed w.r.t its neighbors by solving a least square problem iteratively. The relative rotations are interpolated in pursuance of the shape interpolation, so this method becomes computationally complex compared to interpolating direct face rotation [46, 11]. A subspace interpolation method [48] is proposed for real-time interpolation by projecting a mesh into a lower-dimensional subspace for a given set of poses. The subspace formation is a computationally inefficient process for a given set of poses, and the computational cost increases with the number of poses. Moreover, this process must be repeated for a new set of poses. The approximated subspace may not be adopted for other applications such as morphing and deformation transfer.

Linear Rotation Invariant coordinate [10] is proposed for mesh deformation and interpolation. A vertex deformation is represented as fundamental forms which are further used for interpolation. This method promotes as-rigid-as-possible deformation of the mesh without explicit constraints, although it fails to handle large deformations. The face-based Linear Rotation Invariant coordinate [15] extends the reach of Linear Rotation Invariant coordinate to the large deformation. In this method, the triangular face deformation is computed w.r.t its neighbors as the linear transformation, which is further decomposed into rotation and shear for interpolation. Like patch-based Linear Rotation Invariant coordinates, the relative angle and stretch for a triangular face are interpolated in isometry-invariant coordinate (IIC) [16] to perform interpolation and morphing. The interpolated mesh is reconstructed from IIC iteratively with edge length constraints. Due to relative face deformation, this method can handle large deformation efficiently. A rigid motion invariant coordinate; Pyramid coordinate [17] is proposed for various mesh editing tasks such as interpolation, mesh deformation, and morphing. It comprises a set of pairs of angle and edge length for each vertex computed from its one-ring neighbors. The angle and edge length are interpolated linearly. The interpolated mesh is reconstructed iteratively by solving a non-linear system. The Pyramid coordinate preserves the geometry and shape of the mesh for various mesh editing tasks. Similarly, in [18], Mark Alexa proposes differential coordinates for morphing and mesh deformation. The differential coordinates capture the local geometry of the mesh for various mesh editing tasks. All these methods [10, 15, 16, 17, 18] are computationally inefficient due to element-wise operations. Moreover, the computational cost increases with the resolution of the mesh.

The mean value coordinates are one of the widely used barycentric coordinate first proposed in [49] for mesh deformation and interpolation. This coordinates are generalized for 3D closed triangular meshes in [50] to perform various mesh editing tasks. In this, the weight computation process is generalized to 3D meshes. The mean value coordinates are a special case of barycentric coordinates. In [51], authors have proposed the complex barycentric coordinates to deform and interpolate the planar shapes. The vector graph representation is a barycentric representation of a face. Unlike other barycentric coordinates [49, 50, 51], it can be used for enveloping and deformation transfer. Moreover, the VG representation is not limited to triangular meshes only. It can easily be extended to quad and hybrid meshes.

In [22], the interpolation is performed by projecting meshes into higher dimensional shape space followed by approximating a geodesic with as-isometric-as-possible and as-rigid-as-possible constraints. A multi-scale approach [52] is proposed for as-consistence-as-possible shape interpolation. In this method, the mesh resolution is reduced from fine to coarse resolution levels. At each level, the edge length and dihedral angles of each triangular face are computed and interpolated. The reconstruction process keeps neighbor patches as consistent as possible. This approach [52] is further simplified in [20] by abandoning the hierarchical process and simplifying energy function. In [53], a multi-resolution Mean-Shift clustering algorithm is proposed to segment the triangular mesh into near-rigid components. A hierarchical structure then connects the rigid components. The pose deformation computed from the hierarchical structure and the local deformation computed for each triangle is interpolated to get intermediate poses. All these methods [22, 52, 20, 53] can deal with large deformation, but they are computationally burdensome due to the multi-resolution approach, added constraints, and complex reconstruction process.

2.4 Other Mesh Editing Applications

2.4.1 Style Transfer

In the style transfer, the action style (i.e., angry, excited, etc.) of the source is transferred to the target. In [54], the authors proposed a data-driven frequency domain approach for style transfer that can handle heterogeneous motions. It also can transfers style to actions that are not present in the database. Such capability turns out to be a database enhancement method. However, this method may fail if there is a significant difference among input action and actions in the database as noted in the [54]. In [55], authors have proposed style transfer methods based on Iterative Motion Warping (IMW). This method finds a correspondence between input joints angle curve with output joint angle curve. The IMW consists of time and space warp. This method produces quality result and its performance is nearly real-time. In [56], the authors proposed a data-driven style transfer method based on regression models. In the database, the motion is labeled in terms of style and actions. The regression parameters are estimated for the style transfer using K-nearest neighbors for labeled data. The authors also introduce a mixture of auto regressive models (MAR) to handle unlabeled and heterogeneous motions. This method can also transfer style to the motion which is significantly different from motions in the database.

2.4.2 Motion Editing

Various motion editing methods [57, 58, 59] are proposed to edit the motion properties of the existing animation. In [60], the authors have proposed an interactive motion editing technique using a sketch interface, and combining inverse kinematics with multi-level B-spline interpolation. The spatial and the temporal editing are carried out using this method. The user can edit the motion trajectories interactively just with brush strokes using newly introduced sketch spaces; local, global, and dynamics. The spatial motion of the meshes can be edited using their rig models. The cardinal spline [61] interpolation is used for motion editing technique proposed in [62]. In this method, the user can define the key frames and specify the constraints at each key frames interactively. The trajectories traced out by human motion capture data is considered as motion curve. The new motion is generated using proposed interpolation by editing the user defined key frames. In [63], the authors propose a motion editing and retargeting technique for human-like-figure considering inter and intra frame relationships. The intra-frame relationship can be edited by solving an inverse kinematic problem with constraints. In this approach, Degree of Freedoms (DoFs) at joints are reduced so that the Inverse Kinematics become computationally efficient. The hierarchical B-spline [64] is used for the inter-frame motion editing. The B-spline maintains the smooth inter-frame relationship.

In [65], the authors have proposed a motion graph for motion generation and path synthesis of motion capture data. The motion graph is a directed graph in which a node indicates the transition from one motion to another, and the directed edge indicates a clip of annotated motion. The smooth transition from one motion to another is created by blending two clips. Using motion graph, the motion is generated on the user-specified path by taking a graph walk such that it causes minimum transition error. In [66], the author proposed a motion retargeting method on different characters of diverse physic. During retargeting, specific features of the motion i.e., frequency, end-effector contact, are preserved by adding space-time constraints. In the next chapter, we present the proposed vector graph representation in detail.

CHAPTER 3

Vector Graph Representation

The 3D scanner scans the real-world object and stores it as a collection of the sample points. Such a set of points is known as a point cloud. Connecting neighboring points of a point cloud by edges generate faces. Combining such faces, a 3D surface is generated which is known as a mesh. The faces of a mesh can be triangles, quad, or convex polygons. The Skeleton of a real-world object consists of special feature points (for example, a human skeleton consists of head, upper torso, hands, legs, etc.) known as vertices, and two vertices are conditionally connected by a link with each other (for example, a head vertex is connected only to the upper torso and not with hands). Various types of meshes are shown in the Figure 3.1.

The mesh and the skeleton are examples of graphs, considering vertices as nodes and edges as links. Both the face of a mesh and link of the skeleton can be represented by the vectors originating from the centroid of that face and pointing towards the vertices. If a face or a link contains n vertices, then n vectors represent the face (for skeleton $n = 2$ and for triangular mesh $n = 3$). In the rest of the

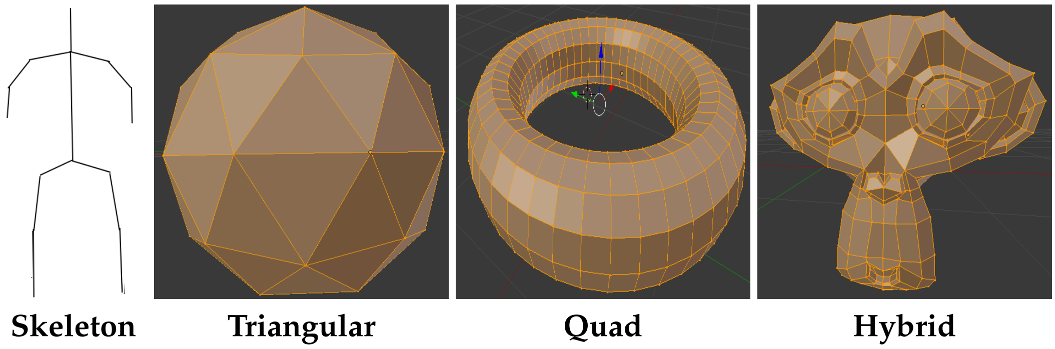

Skeleton **Triangular** **Quad** **Hybrid**

Figure 3.1: Various types of meshes.

sections, we refer to the triangular mesh and skeleton as the vector graphs (VG). Figure 3.2 shows the vector graphs of the line segment, the triangle, the quad and the polygon faces.

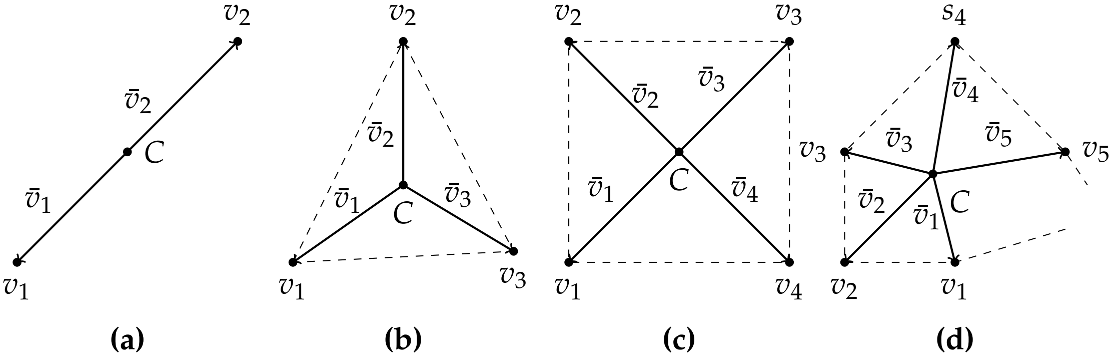


Figure 3.2: The vector graph representation of a face; (**a**) the line segment, (**b**) the triangle, (**c**) the quad and (**d**) the polygonal.

We represent a mesh using its VG representation. In Figure 3.2, we show vector graph representations of a skeleton bone, a triangular, a quad and a polygonal faces. For example, let's consider a polygon face f consisting of vertices $[v_1, v_2, \ldots, v_r]$ where r is the number of vertices in a polygon. The VG representation of f is,

$$[\bar{v}_1, \bar{v}_2, \ldots, \bar{v}_r] = [v_1 - C_f, v_2 - C_f, \ldots, v_r - C_f] \qquad \text{or} \tag{3.1}$$

$$\begin{bmatrix} \bar{v}_1 \\ \bar{v}_2 \\ \vdots \\ \bar{v}_r \end{bmatrix} = \begin{bmatrix} v_1 - C_f \\ v_2 - C_f \\ \vdots \\ v_r - C_f \end{bmatrix} \tag{3.2}$$

where, $C_f = \frac{1}{r}\sum_{i=1}^{r} v_i$ is the centeroid of the face f. All other faces are represented by the VG representation similarly. Let's consider a mesh V consists of q vertices and l number of faces. The VG representation of the mesh is denoted as $\bar{V}$ which consist of n number of vectors. The vector-vertex relation is defined as,

$$VA = \bar{V} \qquad \text{or} \qquad BV = \bar{V}. \tag{3.3}$$

$$\left[\begin{array}{c} v_1, v_2, \cdots, v_q \end{array}\right] A = \left[\begin{array}{c} \bar{v}_1, \bar{v}_2, \cdots, \bar{v}_n \end{array}\right] \quad \text{or} \quad B \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_q \end{bmatrix} = \begin{bmatrix} \bar{v}_1 \\ \bar{v}_2 \\ \vdots \\ \bar{v}_n \end{bmatrix} \tag{3.4}$$

where, $V \in \mathbb{R}^{3\times q}$ (or $V \in \mathbb{R}^{3q}$) and $\bar{V} \in \mathbb{R}^{3\times n}$ (or $\bar{V} \in \mathbb{R}^{3n}$) consist of vertices and vectors. Matrix $A_{q\times n}$ (or $B_{3n\times 3q}$) can be formed using mesh connectivity. Let's consider the face i $f_i = (x_1, x_2, \ldots, x_r)$ to understand the formation of both A and B where, $x_j \in \{1, 2, \ldots, q\}$ is the index of a vertex of the mesh V. Both the matrices are formed as,

$$A_{q\times n} = \begin{array}{cccccccc|c}
1 & \cdots & i+1 & i+2 & \cdots & i+r & \cdots & n & \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \\
0 & \cdots & 1-\frac{1}{r} & 0 & \cdots & 0 & \cdots & 0 & x_1 \\
0 & \cdots & 0 & 1-\frac{1}{r} & \cdots & 0 & \cdots & 0 & x_2 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \vdots & \vdots & \\
0 & \cdots & 0 & 0 & \cdots & 1-\frac{1}{r} & \cdots & 0 & x_r \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots &
\end{array} \tag{3.5}$$

$$B_{3n\times 3q} = \begin{array}{ccccccc|c}
 & 3x_1+1:3x_1+3 & & 3x_2+1:3x_2+3 & & 3x_r+1:3x_r+3 & & \\
\cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 0 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\
\cdots & I_3 - \frac{1}{r}\mathbb{1} & \cdots & 0 & \cdots & 0 & \cdots & 3i+1:3i+3 \\
\cdots & 0 & \cdots & I_3 - \frac{1}{r}\mathbb{1} & \cdots & 0 & \cdots & 3i+4:3i+6 \\
\vdots & \vdots & \vdots & \vdots & \ddots & \vdots & \cdots & \vdots \\
\cdots & 0 & \cdots & 0 & \cdots & I_3 - \frac{1}{r}\mathbb{1} & \cdots & 3i+3r+1:3i+3r+3 \\
\vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \cdots & \vdots \\
\cdots & 0 & \cdots & 0 & \cdots & 0 & \cdots & 3n
\end{array} . \tag{3.6}$$

where, I_3 and $\mathbb{1}$ represent 3×3 identity matrix and 3×3 matrix whose all the elements are 1 respectively. Note that the first row of above matrices (A and B) represent the indices of columns.

3.1 Consistency of the VG Representation and Reconstruction

From a given VG, it is straight forward to reconstruct the mesh. The reconstruction process involves solving equation (3.3) which is an over-determined system of linear equations. Since the VG is a translation-invariant representation ($\bar{V} = VA = (V + t)A$ where, t is the translation applied on the mesh), the reconstructed mesh is not unique. Hence, we can reconstruct a mesh from its VG representation by introducing positional constraints to equation (3.3) as,

$$\begin{aligned} &\underset{V}{argmin} \quad \|VA - \bar{V}\|_2 \\ &\text{subject to} \quad v_l = y_l,\ l \in C \end{aligned} \qquad \text{or} \qquad \begin{aligned} &\underset{V}{argmin} \quad \|BV - \bar{V}\|_2 \\ &\text{subject to} \quad v_l = y_l,\ l \in C \end{aligned} \tag{3.7}$$

where, C is a set of indies of the constrained vertices. Since the position of constraint vertices have already been fixed, they can be removed from the cost function. The modified cost function is,

$$\underset{\hat{V}}{argmin} \quad \|\hat{V}\hat{A} - \tilde{V}\|_2 \qquad \text{or} \qquad \underset{\hat{V}}{argmin} \quad \|\hat{B}\hat{V} - \tilde{V}\|_2 \tag{3.8}$$

where, $\tilde{V} = \bar{V} - V_c A_l$ (or $\tilde{V} = \bar{V} - B_l V_c$). Here, $V_c \in \mathbb{R}^{3\times|C|}$ (or $V_c \in \mathbb{R}^{3|C|}$) consists of constrain vertices. The matrix A_l (or B_l) contains rows (or columns) of A (or B) corresponding to constrained vertices. Matrices $\hat{A}$ (or $\hat{B}$) is derived from A (or B) by removing rows (or columns) corresponding to constraint vertices. The close-form solution for equation (3.8) is,

$$\hat{V} = \bar{V}\hat{A}^+ - V_c\hat{A}_l^+ \qquad \text{or} \qquad \hat{V} = \hat{B}^+\bar{V} - \hat{B}_l^+ V_c. \tag{3.9}$$

Here, matrices $\hat{A}^+ = \hat{A}'(\hat{A}\hat{A}')^{-1}$ (or $\hat{B}^+ = (\hat{B}'\hat{B})^{-1}\hat{B}'$) and $\hat{A}_l^+ = A_l\hat{A}^+$ (or $\hat{B}_l^+ = \hat{B}^+B_l$) depend upon the mesh structure and can be precomputed. The constrained vertices, $V_c \in \mathbb{R}^{3\times|C|}$ (or $V_c \in \mathbb{R}^{3|C|}$), are appended later to $\hat{V}$ to get all vertices (V) of the reconstructed mesh. A representation of the mesh should be consistent,

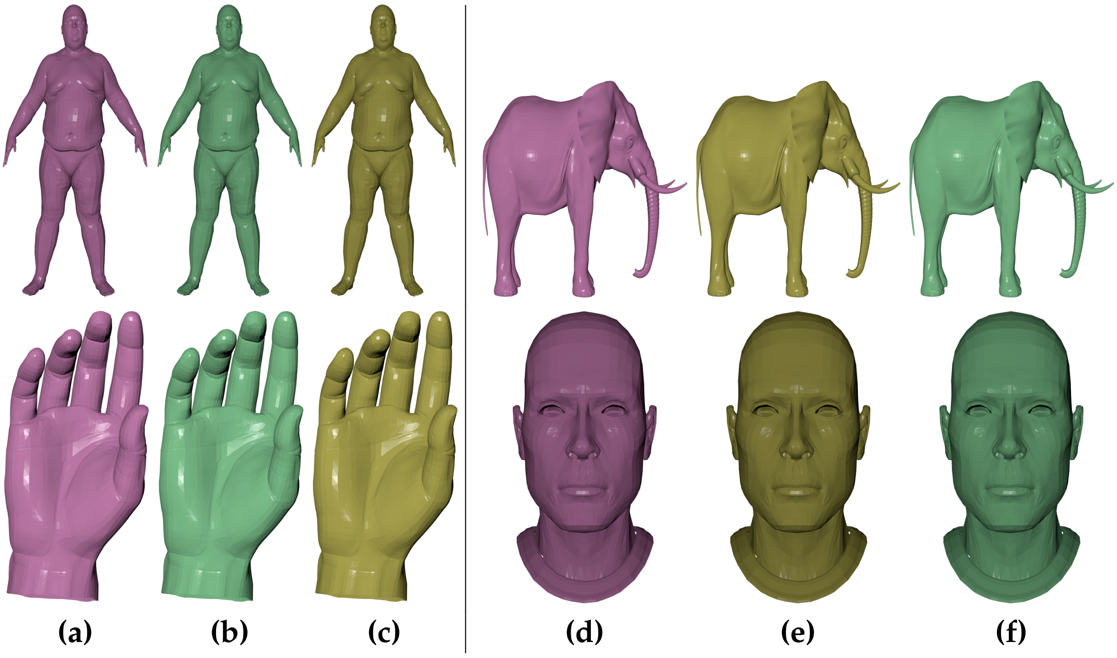


Figure 3.3: The meshes of four different objects (**(a)** and **(d)**) are regenerated by processing all the faces (**(b)** and **(e)**) and selected faces (**(c)** and **(f)**) from their VG representation.

.

which means it should regenerate the mesh accurately. To check the consistency, various meshes are regenerated from its VG representation as shown in Figure 3.3. The root sum squared error is computed as a quantitative analysis as shown in Table 3.1. Both qualitative and quantitative analysis show that the VG is a consistent representation from which the meshes can be regenerated accurately.

Model)	Face Type	Faces		Error (10^{-13})		Reconstruction Time(ms)	
		Full	Half	Full	Half	Full	Half
Woman	Triangular	19938	10342	0.88	4.86	10.41	5.29
Head	Triangular	31620	16158	1.54	2.64	16.95	9.15
Elephant	Triangular	84638	43686	8.11	17.8	47.14	26.21
50002	Triangular	13776	7125	1.45	2.53	6.96	3.69
Hand	Quad	8279	6624	0.12	0.37	4.45	4.08

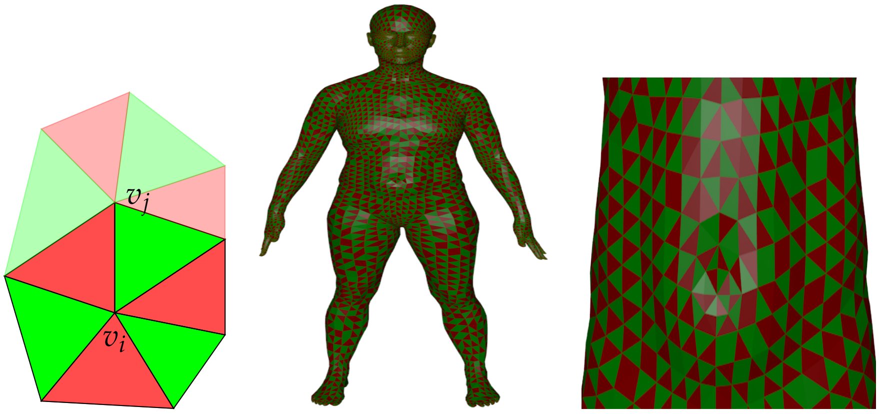
Table 3.1: Quantitative analysis of the reconstructed meshes shown in Figure 3.3 from their VG representation.

Algorithm 1: Face Selection Algorithm

```
Result: A set of selected faces $L_f$
$L_f = \emptyset$;
for $v_i \in V$ do
    find set ($O_i$) of one-ring faces of $v_i$;
    $A_f = L_f \cap O_i$ # already selected faces;
    $R_f = O_i - A_f$ # Remaining faces;
    for $j \in R_f$ do
        $E$ = set of edges of face $j$;
        if E doesn't contains edges of set of faces $A_f$ or $A_f = \emptyset$ then
            $A_f = A_f \cup \{j\}$;
        end
    end
    $L_f = L_f \cup A_f$;
end
```

3.2 Selection of Faces

The equation (3.3) is an over-determine system of linear equations because a vertex is belong to many faces. Hence the computation cost can be reduced by reducing the number of equations. Randomly removing equations may make the system under-determine or may affect the quality of reconstructed mesh. Here, we propose a rule based face (i.e. number of equations) selection algorithm. For each vertex of the given mesh, there corresponds a set which contains faces that share this vertex. In that set, there are faces amongst which there is no common edge. We first select such faces for a vertex of a mesh. Next, we select faces for neighbour vertices of this vertex considering already selected faces similarly. This process is repeated till the entire vertex set is exhausted. The selection process is explained in Algorithm 1. For the triangular meshes, the number of selected faces are approximately half the total number of faces. As shown in Figure 3.3, the reconstructed meshes using selected faces are visually similar to ground truth mesh and reconstructed mesh. The root sum squared error in Table 3.1 for reconstructed meshes is very similar to the reconstructed meshes using all the faces. As shown in Table 3.1, the number of selected faces of triangular meshes are half so is reconstruction time without compromising the quality.



(a) (b) (c)

Figure 3.4: **(a)** Face selection process for vertices v_i and v_j. Faces in green are selected faces. **(b)** Selected faces on a mesh. **(c)** A zoomed patch of the mesh with selected faces (in green) using the face selection algorithm.

3.3 Computing Deformation

To model the deformation of a mesh, we compute the deformation of all the vectors of its VG representation. Such local deformations collectively represent the mesh deformation. Let's denote $\bar{v}_{ri}$ and $\bar{v}_{di}$ as reference and deformed vectors respectively. The relation between $\bar{v}_{ri}$ and $\bar{v}_{di}$ is defined as,

$$\bar{v}_{di} = a_i J_i \bar{v}_{ri} \tag{3.10}$$

where, J_i and a_i are scaling and rotation transformations. Next, we will show computation of the scaling a_i and rotation J_i.

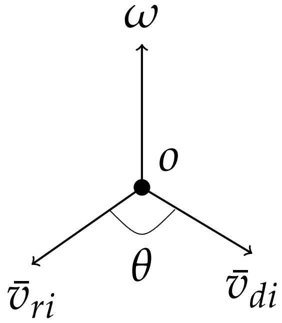


Figure 3.5: The graphical representation of computation of rotation using Rodrigues' formula.

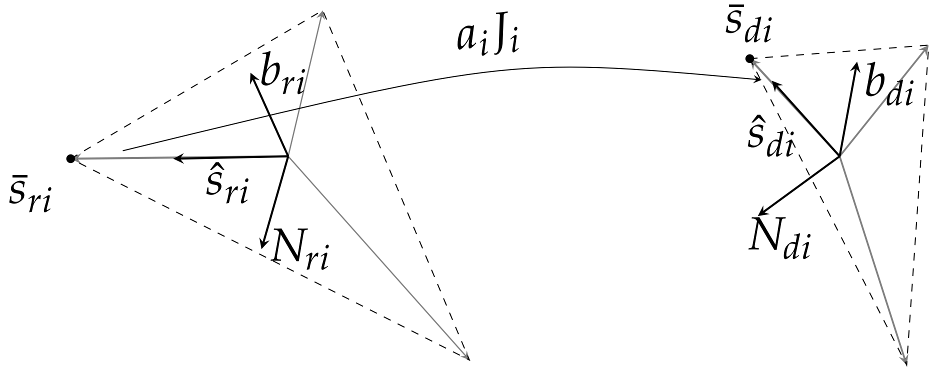


Figure 3.6: The deformation of the vector $\bar{v}_{ri}$ to the vector $\bar{v}_{di}$ is computed as composition of scaling (a_i) and rotation J_i by forming orthonormal frames.

Rodrigues' Formula:

The rotation matrix that rotate $\bar{v}_{ri}$ towards $\bar{v}_{di}$ is defined using Rodrigues' formula as follow,

$$J_i = \begin{bmatrix} t\omega_x^2 + e & t\omega_x\omega_y - g\omega_z & t\omega_x\omega_z + g\omega_y \\ t\omega_x\omega_y + g\omega_z & t\omega_y^2 + e & t\omega_y\omega_z - g\omega_x \\ t\omega_x\omega_z - g\omega_y & t\omega_y\omega_x + g\omega_x & t\omega_z^2 + e \end{bmatrix} \text{ and } a_i = \frac{\|\bar{v}_{di}\|_2}{\|\bar{v}_{ri}\|_2} \quad (3.11)$$

where, $\omega = (\omega_x, \omega_y, \omega_z) = \frac{\bar{v}_{ri} \times \bar{v}_{di}}{\|\bar{v}_{ri} \times \bar{v}_{di}\|_2}$, $e = cos\theta$, $g = sin\theta$ and $t = 1 - cos\theta$. θ is the angle between both the vectors as shown in Figure 3.5.

Orthonormal Frames:

The orthonormal frames are formed for both the vectors using normals of their corresponding faces as shown in Figure 3.6. The orthonormal frame for both $\bar{v}_{ri}$ and $\bar{v}_{di}$ is formed as,

$$F_{ri} = [\hat{v}_{ri}, b_{ri}, N_{ri}] \text{ and } F_{di} = [\hat{v}_{di}, b_{di}, N_{di}]. \quad (3.12)$$

Here, $\hat{v}_{ri}$ and $\hat{v}_{di}$ are unit vectors corresponding to $\bar{v}_{ri}$ and $\bar{v}_{di}$ respectively. The vectors N_{ri} and N_{di} are unit normals of the reference and the deformed faces respectively. b_{ri} and b_{di} are their unit binormal vectors. The transformation of the frame F_{ri} to F_{di} is defined as,

$$J_i F_{ri} = F_{di} \rightarrow J_i = F_{di} F'_{ri} \text{ and } a_i = \frac{\|\bar{v}_{di}\|_2}{\|\bar{v}_{ri}\|_2}. \tag{3.13}$$

where, the transformation matrix J_i is a rotation matrix (see proof in Appendix A.4). a_i is the scaling transformation.

3.4 Properties of Vector Deformation

Various properties of the Vector Graph representation makes it useful for various applications.

3.4.1 Orientation of Vectors

For Deformation Transfer application, the transformation computed using a representation should be orientation preserving transformation because it is an essential property which plays an important role in preserving the shape of the target for a wide range of deformations. In [1, 14, 2, 18], the triangle deformation is modelled locally as the linear transformation which is not orientation preserving transformation. Let $\bar{x}_r$ and $\bar{y}_r$ are two vectors, and $\bar{x}_d$ and $\bar{y}_d$ are their corresponding deformed vectors. The relation between $\bar{x}_r$ and $\bar{x}_d$ can be defined as $\bar{x}_d = a_x J_x \bar{x}_r$ as explained in section 3.3. The orientation between $\bar{x}_r$ and $\bar{y}_r$ is,

$$cos^{-1}(\hat{x}'_r \hat{y}_r) = \theta_r \tag{3.14}$$

where, $\hat{x}_r = \bar{x}_r / \|\bar{x}_r\|$ and $\hat{y}_r = \bar{y}_r / \|\bar{y}_r\|$ are unit vectors. Applying deformations J_x and a_x on the vector $\bar{y}_r$, we get $\bar{y}_d = a_x J_x \bar{y}_r$ where, $a_x = \|\bar{y}_d\| / \|\bar{y}_r\| = \|\bar{x}_d\| / \|\bar{x}_r\|$. The orientation between $\bar{x}_d$ and $\bar{y}_d$ is

$$\begin{aligned} \theta_d = cos^{-1}(\hat{x}'_d \hat{y}_d) = cos^{-1}(\frac{(a_x J_x \bar{x}_r)'}{\|\bar{x}_d\|} \frac{(a_x J_x \bar{y}_r)}{\|\bar{y}_d\|}) \\ = cos^{-1}((\frac{\bar{x}_r}{\|\bar{x}_r\|})'(\frac{\bar{y}_r}{\|\bar{y}_r\|})) = cos^{-1}(\hat{x}'_r \hat{y}_r) = \theta_r. \end{aligned} \quad (\because J'_i J_i = I_3) \tag{3.15}$$

Hence, the transformation composition, rotation and scaling, preserves the orientation.

3.4.2 Set of Deformation as a Manifold

As noted in [14], a triangular deformation computed as the linear transformation should form a manifold because it is vital for statistical analysis, distance measurement, and interpolation of meshes. The transformation which does not form a manifold may lead to non-physical deformation on the mesh. Using the VG representation, the mesh deformation is modelled as a collection of composition of rotation and scaling of each vector. Hence, the mesh deformation using VG also forms a group.

Definition 1. The vector deformation group $g_v \in G_v$, is a tuple,

$$g_v = (a_1 J_1, a_2 J_2, \ldots, a_n J_n) \tag{3.16}$$

where, $a_i \in R^+$ and $J_i \in SO(3)$. The composition map,

$$G_v \times G_v \mapsto G_v, \tag{3.17}$$

$$(a_1^1 J_1^1,\ a_2^1 J_2^1,\ \ldots,\ a_n^1 J_n^1) \times (a_1^2 J_1^2,\ a_2^2 J_2^2,\ \ldots,\ a_n^2 J_n^2) \mapsto (a_1^1 a_1^2 J_1^1 J_1^2,\ a_2^1 a_2^2 J_2^1 J_2^2,\ \ldots,\ a_n^1 a_n^2 J_n^1 J_n^2) \tag{3.18}$$

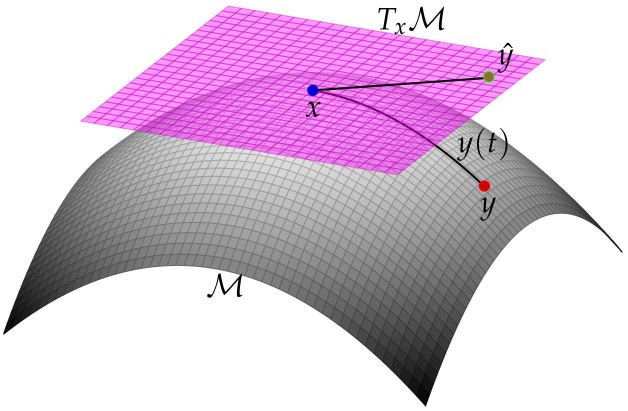


Figure 3.7: Manifold $\mathcal{M}$ and tangent space $T_x\mathcal{M}$ at point x.

The group G_v is a subgroup of the general linear group $(G_l(3))$ of invertible matrices which is a Lie group. Hence, G_v is a subgroup of the Lie group. Since the Lie group is a smooth manifold, the G_v is also a smooth manifold denoted as $\mathcal{M}$. The dimension of the manifold $\mathcal{M}$ is $4N_v$ where, N_v is the number of vectors in the VG representation of a mesh. Let x and y are points on the manifold $\mathcal{M}$. The tangent space at point $x \in \mathcal{M}$ is denoted as $T_x\mathcal{M}$. Both the manifold and tangent space $T_x\mathcal{M}$ are related by exponential and logarithmic maps. The logarithm ($log : y \mapsto \hat{y}$) connects manifold to tangent space whilst the exponential ($exp : \hat{y} \mapsto y$) connects tangent space to manifold. The point y on manifold is related to tangent space $T_x\mathcal{M}$ as $y = x * exp(log(x^{-1}y)) = x * exp(\hat{y})$ as shown in Figure 3.7. The geodesic between point x and y on manifold $\mathcal{M}$ is computed as $y(t) = x * exp(t * log(x^{-1}y))$, where $t \in [0, 1]$.

3.4.3 Simplification Due to Commutativity of Scaling and Rotation

The commutative composition of a transformation computed by a mesh representation simplifies the proposed enveloping (see transition from equation (5.26) to equation (5.27) and interpolation (see transition from equation (6.17) to equation (6.18) processes such that it becomes computationally real-time. Such simplification is not possible for other representations [1, 14, 10, 47, 16]. In some of these representations, the triangle deformation computed as the linear transformation is represented as composition of the rotation and positive definite symmetric matrices using QR-decomposition. In [14], the linear transformation is represented as composition of rotation, scaling, and positive-definite upper triangular matrix. These compositions are not commutative. In VG, the vector deformation composition, rotation and scaling, is commutative as,

$$a_i J_i = J_i a_i \tag{3.19}$$

where, a_i and J_i are scaling and rotation of the vector i.

Chapter 4

Deformation Transfer

In graphics and animation, animating the target object according to an example (source object) animation sequence is a challenging problem. Moreover, this process involves highly skilled graphic developers. Deformation Transfer is a process that transfers the action of a source object to the target object. Deformation Transfer generates an action sequence (of poses) of the target object similar to the action sequence of the source object with minimum human intervention. As an example of Deformation Transfer, in Figure 4.1, the poses of a female actor (on the right) are generated according to the poses of a male actor (on the left). Deformation Transfer to be effective, should transfer the deformation of the source to the target automatically and should also preserve the target shape.

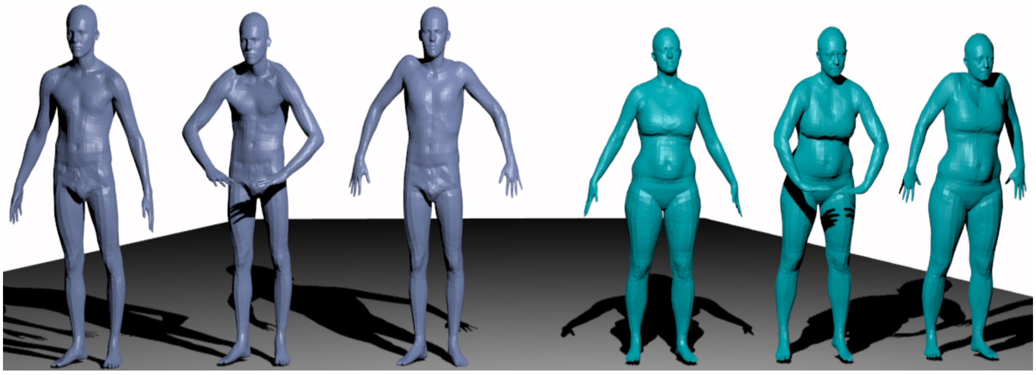


Figure 4.1: Demonstration of the deformation transfer : The action of a man (Source) is transferred to the woman (Target).

It is necessary that the Deformation Transfer algorithm preserves geometric details of the target object during the process of deformation transfer. In practice, various geometric models such as polygon/tet-mesh and skeleton are used for various applications. For example, skeletons are used in action and activity recognition whereas triangular, quad hybrid meshes are used in animations and computer graphics to model 3D objects. Handling variety of geometric models, adjusting temporal positions and motion trajectory, and preserving the target features are challenges for a Deformation Transfer method.

The Deformation Transfer is carry out in two phases; first, a dense set of correspondence between the reference source and the target is identified from the sparse set of user selected markers and in second phase, the deformation of source is transferred to target with the help of known correspondence. We, in proposed method, assume that the dense set of correspondence is available. However, if such correspondence is not available, then any known mesh registration method can be used for establishing the correspondence for example [1, 67]. In proposed approach, the mesh is represented by *vector graph* (VG) representation which consists of a collection of vectors for various types of meshes such as skeleton, triangular, quad and hybrid meshes. A vector deformation is decomposed into rotation and scaling. In this work, the Deformation Transfer problem is explored by computing the shape and the pose deformations which capture shape and pose variations respectively using the VG representation. The Deformation Transfer using shape deformation causes artifacts on the target mesh when the deformation is large. It also fails to preserve the planarity of the quad faces. These limitations are addressed in the Deformation Transfer using the pose deformation which characterizes the pose variation in the source mesh. Here, we also bring out the differences between this approach and the other Deformation Transfer methods proposed in [1] and [2]. It is straight forward to move from the VG to the mesh and vice versa. Using Deformation Transfer, the approximated poses of the target sequence has the similar temporal properties as the source sequence. These properties of the target sequence are further refined using *Poisson Interpolation* by modifying the temporal gradient information.

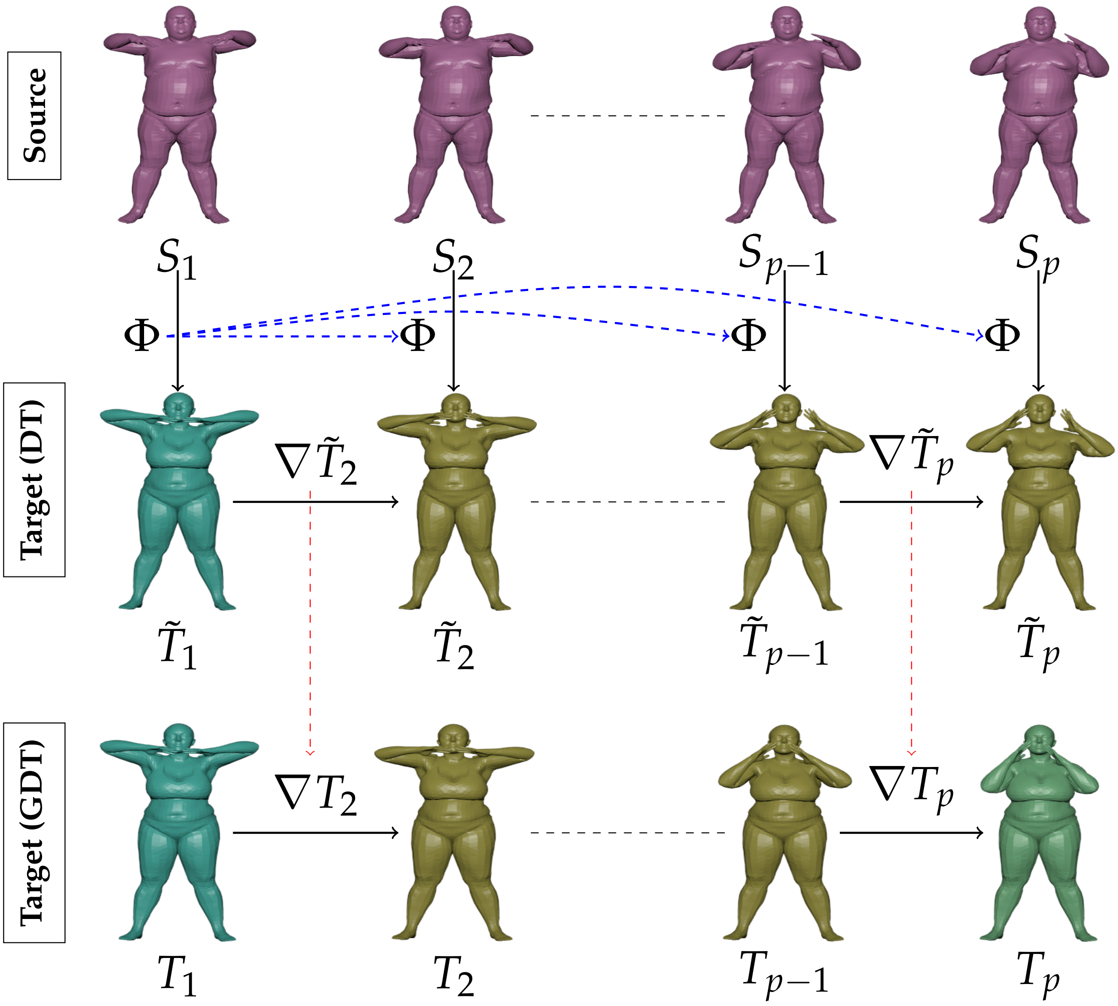


Figure 4.2: Deformation Transfer using shape deformation. The source sequence $\{S_1, S_2, \ldots, S_{p-1}, S_p\}$ and initial(T_1) and final (T_p) poses of target are input to the Deformation Transfer framework. The shape deformation, Φ, is computed from the source and the target reference poses. Target sequences $\{\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_{p-1}, \tilde{T}_p\}$ and $\{T_1, T_2, \ldots, T_{p-1}, T_p\}$ are generated by transferring Φ to corresponding deform source poses and by refining temporal properties using the Poisson interpolation. ∇T_d and $\nabla\tilde{T}_d$ are temporal gradients of pose d of the target sequence estimated by the Deformation Transfer and the Poisson interpolation respectively.

4.1 Deformation Transfer Using Shape Deformation

Figure 4.2 outlines the Deformation Transfer approach using the shape deformation (Φ) which captures the shape difference between the source and the target reference meshes. It is computed using the VG representation through established correspondences. The computed shape deformation is then transferred to the deformed source poses to get corresponding deformed target poses $\{\tilde{T}_2, \tilde{T}_3, \ldots, \tilde{T}_{p-1}, \tilde{T}_p\}$ as shown in Figure 4.2. These processes are explained in detail in the following sections.

4.1.1 Shape Deformation

Here, the shape deformation computation process is explained in detail. Let's denote i^{th} reference source and reference target vectors as $\bar{s}_{ri}$ and $\bar{t}_{ri}$ as illustrated in Figure 4.3(a). The relation between both the vectors is defined as,

$$\bar{t}_{ri} = b_i R_i \bar{s}_{ri} \tag{4.1}$$

where b_i and R_i are scaling factor and rotation matrix respectively. The rotation is computed using Rodrigues' rotation formula as explained in section 3.3. The scaling is computed as $b_i = \|\bar{t}_{ri}\| / \|\bar{s}_{ri}\|$. Similarly, we compute the scaling factor and the rotation matrix for each source-target vector pair.

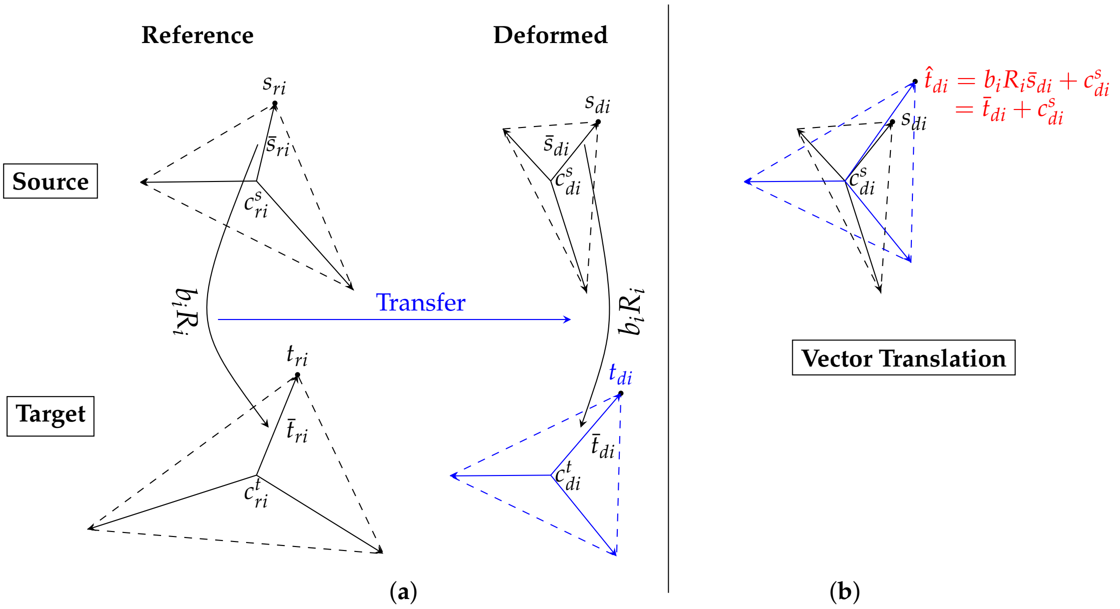


Figure 4.3: (**a**) The deformations b_i and R_i are computed from i^{th} source and target reference vectors. The deformations are then transferred to the corresponding i^{th} deformed source vector to get deformed target vector. (**b**) The computed deform target vector $\bar{t}_{di}$ is translated to the centroid of the corresponding source face.

Deformation Transfer

Our objective is to estimate unknown the target sequence using the shape deformation. For this task, the shape deformation is applied to the VG representation of the deformed source poses. The reconstruction process approximates the deform mesh from its VG representation. In this process, the deform target vector *i* is initially translated to centroids of corresponding source face as (see Figure 4.3(b)),

$$b_i R_i \bar{s}_{di} + c^s_{di} = \hat{t}_{di} \tag{4.2}$$

where, c^s_{di} is the centroid of the deform source vector i. A similar relationship can be derived for all other vectors of the deform target vectors. Collectively, all n equations are combined in the following form,

$$\Phi \bar{S}_d + C^s_d = \Phi B S_d + D S_d = (\Phi B + D) S_d = M S_d = \hat{T}_d \tag{4.3}$$

where, Φ is the shape deformation. The matrices B and D are $3n \times 3q$ matrices

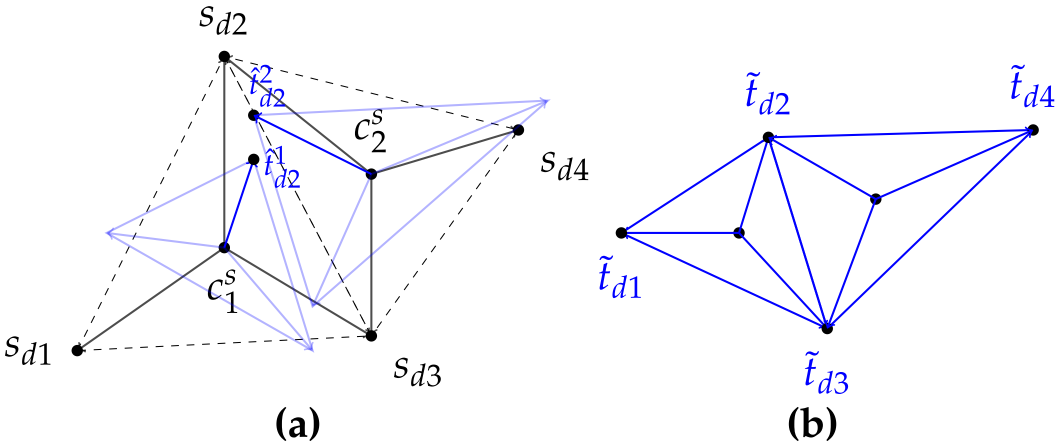


Figure 4.4: The mesh reconstruction process. **(a)** Translated deform target vectors ($\hat{t}^1_{d2}$ and $\hat{t}^2_{d2}$) to the centroids of corresponding source faces (c^s_1 and c^s_2). Note that the deform target faces are disconnect. **(b)** The reconstructed deform target mesh vertices $\tilde{t}_{di}$.

such that,

$$\Phi = \begin{bmatrix} b_1R_{_1} & 0 & \dots & 0 \\ 0 & b_2R_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & b_nR_n \end{bmatrix}, BS_d = \begin{bmatrix} \bar{s}_1 \\ \bar{s}_2 \\ \vdots \\ \bar{s}_n \end{bmatrix}, \ DS_d = \begin{bmatrix} c^s_{d1} \\ c^s_{d2} \\ \vdots \\ c^s_{dn} \end{bmatrix} \text{ and } \hat{T}_d = \begin{bmatrix} \hat{t}_{d1} \\ \hat{t}_{d2} \\ \vdots \\ \hat{t}_{dn} \end{bmatrix} \tag{4.4}$$

where, c^s_{di} is the centroid of a face in which source vector i lays. The column matrix $\hat{T}_d$ contains vertices of the disjoint deform target faces as shown in Figure 4.4(a). For the proper mesh, the disjoint faces should be connected. The fully connected deform target mesh as $\tilde{T}_d$ is related with $\hat{T}_d$ as,

$$E\tilde{T}_d = \hat{T}_d \tag{4.5}$$

where matrix E is a incident matrix and it can be derived from the mesh connectivity. The deformed target mesh ($\tilde{T}_d$) is derived by solving equation (4.5) in the least square sense as,

$$\tilde{T}_d = E^+\hat{T}_d = E^+MS_d \tag{4.6}$$

where, $\tilde{T}_d$ is a fully connected mesh as illustrated in Figure 4.4(b). The matrix $E^+ = (E'E)^{-1}E'$ is a pseudo inverse of E. The formation of matrix E and E^+ can be understood with the following example. Let's consider a VG with four vertices $\{x_0, x_1, x_2, x_3\}$ and two faces $F = [[0,1,2],[0,3,2]]$. Here, $n = 3$, $q = 2$, $m = 4$ and matrices,

$$E = \begin{bmatrix} I_3 & 0 & 0 & 0 \\ 0 & I_3 & 0 & 0 \\ 0 & 0 & I_3 & 0 \\ I_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & I_3 \\ 0 & 0 & I_3 & 0 \end{bmatrix}, D = \frac{1}{3}\begin{bmatrix} \mathbb{1} & \mathbb{1} & \mathbb{1} & 0 \\ \mathbb{1} & \mathbb{1} & \mathbb{1} & 0 \\ \mathbb{1} & \mathbb{1} & \mathbb{1} & 0 \\ \mathbb{1} & 0 & \mathbb{1} & \mathbb{1} \\ \mathbb{1} & 0 & \mathbb{1} & \mathbb{1} \\ \mathbb{1} & 0 & \mathbb{1} & \mathbb{1} \end{bmatrix}, B = E - D, E^+ = \begin{bmatrix} \frac{1}{2}I_3 & 0 & 0 & \frac{1}{2}I_3 & 0 & 0 \\ 0 & I_3 & 0 & 0 & 0 & 0 \\ 0 & 0 & \frac{1}{2}I_3 & 0 & 0 & \frac{1}{2}I_3 \\ 0 & 0 & 0 & 0 & I_3 & 0 \end{bmatrix}. \tag{4.7}$$

where, I_3 is the (3×3)identity matrix and $\mathbb{1}$ is a 3×3 matrix whose elements are all 1. It is clear that $\tilde{t}_{di}$ is an average of vertices shared by disjoint faces ($\hat{t}^1_{d2}$ and $\hat{t}^2_{d2}$) as shown in Figure 4.4. The reconstruction process combines all the disjoint faces to get the proper mesh.

4.1.2 Poisson Interpolation

In this section, we discuss the Poisson interpolation proposed in [30, 11]. Consider a set $S \subset \mathbb{R}^2$, $\Omega \subset S$, a vector field $g : \Omega \to \mathbb{R}^2$ and $\partial\Omega$ the boundary of the set Ω. Let f^* be a known scalar function defined over S and f be an unknown scalar function defined over Ω. The Poisson interpolation finds the unknown scalar function as a solution to the optimization problem,

$$\underset{f}{argmin} \int\int_{\Omega} |\nabla f - g|^2 \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{4.8}$$

where $\nabla \cdot = [\frac{\partial \cdot}{\partial x}, \frac{\partial \cdot}{\partial y}]$. The equation (4.8) is the Euler-Lagrange equation and its optimal solution is the well known Poisson equation with Dirichlet boundary conditions,

$$\Delta f = div \cdot g \text{ over } \Omega \text{ with } f|_{\partial\Omega} = f^*|_{\partial\Omega} \tag{4.9}$$

where, Δ is the Laplacian operator and $div \cdot g$ is the divergence of the vector field g. The unknown function f can be derived by solving the equation (4.9).

Effect of Boundary Conditions:

Let's consider a bunch of connected source particles $\mathcal{S}$ is moving in $\mathbb{R}^3$. The connection amongst them is in the form of either a line, a triangle, or a quad at an instance of time. We wish to generate similar trajectories for a bunch of the target particles $\mathcal{P}$, given their initial and final positions (boundary conditions). In this problem, we consider two types of boundaries: different initial and final positions (*Non-similar boundaries*), and the same initial and final positions (*Similar boundaries*). The connection between estimated target trajectories should be the same as the source at an instance of time. This problem can be formulated as Poisson

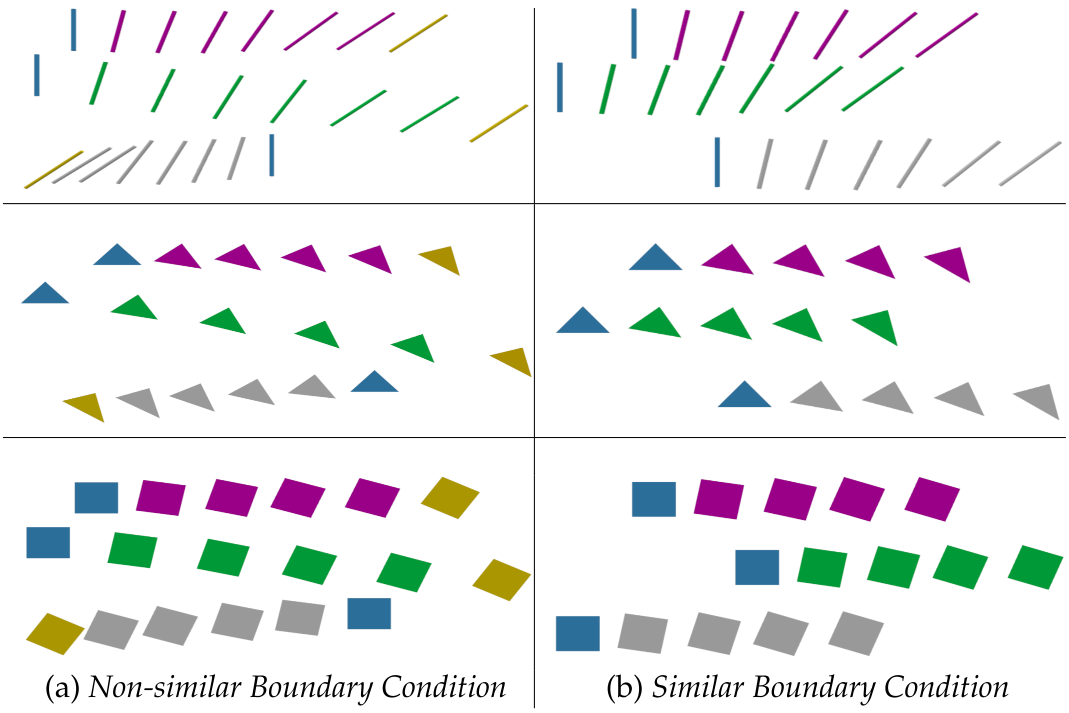
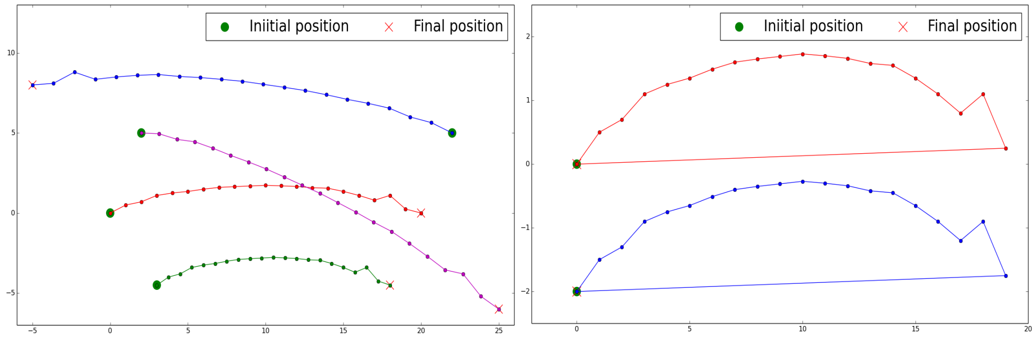

(a) *Non-similar Boundary Condition*
(b) *Similar Boundary Condition*

Figure 4.5: The trajectories of the bunch of target particles $\mathcal{P}$ (in green, and gray) are estimated using the trajectories of the bunch of connected source particles $\mathcal{S}$ (in pink) with (a) *non-similar boundary condition* and (b) *similar boundary condition*. The blue and yellow are initial and final boundaries respectively.

interpolation. The experimental results of Poisson interpolation are presented in Figure 4.5. In this experiment, we have considered two, three, and four particles in a bunch with linear, triangular, and quadrangular connectivity. It is interesting to observe that Poisson interpolation maintains the shape of trajectories and connections amongst the target particles $\mathcal{P}$ similar to $\mathcal{S}$. It also adjusts their relative positions and motion directions depending upon their boundary conditions.

Let $\{\mathcal{S}_1, \mathcal{S}_2, \ldots, \mathcal{S}_h\}$ is a set of temporal positions of the source particles $\mathcal{S}$ on their trajectories and $\{\mathcal{P}_1, \mathcal{P}_h\}$ is the set of boundary positions of target particles $\mathcal{P}$. Estimated temporal positions of target particles $\mathcal{P}$ at y using the Poisson interpolation with *non-similar boundaries*, is computed using equation (4.9) (see Appendix A.2 for derivation) as,

$$\mathcal{P}_y = \mathcal{S}_y + \frac{y-1}{h-1}\mathcal{P}_h + \frac{h-y}{h-1}\mathcal{P}_1 - \frac{y-1}{h-1}\mathcal{S}_h - \frac{h-y}{h-1}\mathcal{S}_1. \tag{4.10}$$



(a)

(b)

Figure 4.6: The trajectories of a target particle $\mathcal{P}$ (in blue, green, and magenta) estimated using the trajectory of a source particle $\mathcal{S}$ (in red) with (a) *non-similar boundary constraint* and (b) *similar boundary constraint*.

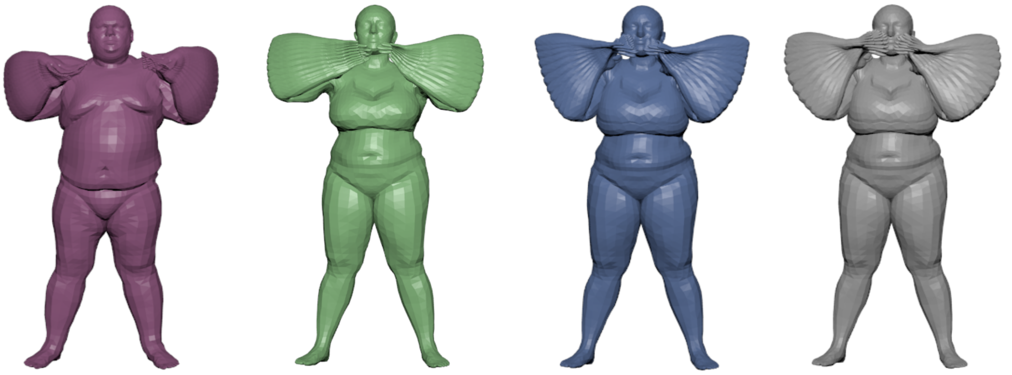
For *similar boundaries* ($\mathcal{P}_1 = \mathcal{P}_h$ and $\mathcal{S}_1 = \mathcal{S}_h$), above equation is rewritten as,

$$\mathcal{P}_y = \mathcal{S}_y + \mathcal{P}_1 - \mathcal{S}_1. \tag{4.11}$$

Hence, for *similar boundaries*, the temporal positions of particles $\mathcal{P}$ get the same translation $\mathcal{P}_1 - \mathcal{S}_1$ resulting in particles $\mathcal{P}$ have the same trajectories as the $\mathcal{S}$ (see Figure 4.5(b)). For *non-similar boundaries*, temporal positions and so the motion directions of particles $\mathcal{P}$ is adjusted according to equation (4.10) but shapes of trajectories remain similar as $\mathcal{S}$ (see Figure 4.5(a)). It is the same case for the single particle as shown in Figure 4.6.

4.1.3 Guided Deformation Transfer

The estimated target sequence $\{\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_p\}$ has similar deformations, motion trajectory and pose position as the source as shown in Figure 4.7(a) and (b). Since both the source and the target are different animation characters, these properties should be adjusted accordingly for the target. The objective here is to estimate unknown target sequence $\{T_1, T_2, T_3, \ldots, T_{p-1}, T_p\}$ which consists of refined temporal properties. The Poisson interpolation can perform this task based on given initial and final poses T_1 and T_p. Since we have transformed sequence $\{\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_p\}$, its temporal gradient information is used as guiding vector field. In terms of the Poisson interpolation frame work explained in section 4.1.2, we



(**a**) Source (**b**) DT (**c**) GDT (**d**) Ground Truth

Figure 4.7: (**a**) Source sequence. (**b**) The target sequence generated by Deformation Transfer (DT) without Poisson interpolation. (**c**) The target sequence generated by Guided Deformation Transfer (GDT). The Poisson interpolation can adjust temporal properties of the target based on boundary conditions using temporal gradients of trajectories of vertices sequence estimated by the Deformation Transfer (DT). (**d**) Available target sequence in the Dyna dataset [4] considered as ground truth.

consider $X = \{T_1, T_2, \ldots, T_p\}$, $\Omega = \{T_2, \ldots, T_{p-1}\}$ and $\partial\Omega = \{T_1, T_p\}$. The deformation transfer problem can now be put in the form of Poisson interpolation as follows: Given X, Ω and $\partial\Omega$ above, generate $\{T_2, T_3, \ldots, T_{p-1}\}$, with the condition that the temporal gradient ∇T_k is the same as $\nabla \tilde{T}_k$. The objective function can be formulated as follows,

$$\underset{T_k \in \Omega}{argmin} \sum_{k=2}^{p-1} \|\nabla \tilde{T}_k - \nabla T_k\|_2^2 \text{ with } T_1, T_p \in \delta\Omega. \tag{4.12}$$

From equation (4.6), we can rewrite above equation as,

$$\underset{T_k \in \Omega}{argmin} \sum_{k=2}^{p-1} \|\nabla (E^+ M S_k) - \nabla T_k\|_2^2 \text{ with } T_1, T_p \in \delta\Omega. \tag{4.13}$$

where, the matrices E^+ and M can be computed as explained in the section 4.1.1. To solve for T_k, setting the gradient of the objective function to zero, gives,

$$\forall k,\ \Delta T_k = \Delta(E^+ M S_k) = E^+ M(\Delta S_k)\ (\because \Delta \text{ is linear operator}) \tag{4.14}$$

where, Δ is a Laplacian operator so it is linear operator. $\Delta T_k = 2T_k - T_{k+1} - T_{k-1}$ and $\Delta S_k = 2S_k - S_{k+1} - S_{k-1}$ are denoted as Laplacian of T_k and S_k respectively. Equation (4.14) is the Poisson equation with known boundary constraints T_1 and T_p. Since the generated system of linear equations (4.14) is symmetric and has a unique solution, this can be solved by the Gauss-Seidel iterative successive over relaxation method as follows,

$$\forall k,\ T_k^{(x+1)} = (1-\omega)T_k^x + \frac{\omega}{2}(E^+ M(\Delta S_k) + T_{k-1}^x + T_{k+1}^x) \tag{4.15}$$

where, ω is a constant, we choose $\omega = 1.9$ experimentally. Hence, the refined target sequence can directly be computed from the source sequence. The equation (4.14) can be simplified similar to equation (4.10)) as follows,

$$T_k = E^+ M(S_k - \frac{p-k}{p-1}S_1 - \frac{k-1}{p-1}S_p) + \frac{p-k}{p-1}T_1 + \frac{k-1}{p-1}T_p. \tag{4.16}$$

In case of *non-similar boundary condition*, Poisson interpolation adjusts motion trajectory and pose position based on guided field and boundary conditions. For *similar boundary condition* ($T_1 = T_p$ and $S_1 = S_p$) above equation is modified to

$$T_k = E^+ MS_k + E^+ MS_1 + T_1 = E^+ MS_k + \tilde{T}_1 - T_1 = E^+ MS_k + d. \tag{4.17}$$

For *similar boundary condition*, the Poisson interpolation doesn't modify motion trajectories of target sequence estimated by the Deformation Transfer, but it adjusts the pose position according to translation (d). Figure 4.7 shows the effect of Poisson interpolation on estimated target action sequences. Note that, for Deformation Transfer without Poisson interpolation, the temporal properties of the target are similar to the source as shown in Figure 4.7(b). Poisson interpolation with *similar boundary condition* generates the same sequence, but all poses are translated by d. The Deformation Transfer using Poisson interpolation with *non-similar boundary condition* adjusts temporal properties of the target according to boundary poses so the target sequence is close to the ground truth as shown in Figure 4.7(c) and (d). The temporal properties of the target sequence are adjusted using the source guidance field (i.e., ∇S_k), so this method is called Guided Deformation Transfer.

Person (no. of actions × no. of poses in a sequence)	Proposed method		DG [1]		Semantic [5]		Lie Body [2]	
	avg	max	avg	max	avg	max	avg	max
Person 1 (13 × 9)	0.98	10.09	3.99	26.85	5.06	46.32	2.47	28.61
Person 2 (12 × 9)	0.95	10.02	5.01	27.07	5.25	40.50	3.03	30.84
Person 3 (10 × 9)	1.28	12.78	5.38	30.39	5.62	56.50	3.41	37.01
Person 4 (13 × 9)	0.95	9.82	3.79	21.18	5.37	43.73	2.39	29.28
Person 5 (12 × 9)	0.91	8.92	4.15	22.59	3.83	29.28	1.89	22.46
Person 6 (13 × 9)	1.45	25.03	4.18	22.21	5.48	49.67	2.50	30.46
Person 7 (12 × 9)	0.98	8.79	3.95	24.51	4.28	45.69	2.26	25.26
Person 8 (13 × 9)	0.85	6.48	4.23	23.77	5.66	40.76	2.64	21.58
Person 9 (12 × 9)	1.05	9.14	4.54	26.47	5.99	50.49	2.67	27.79

Table 4.1: Comparison of proposed method with state-of-the-art methods in terms of average and maximum of root sum squared errors computed at each vertex of all the estimated poses of a target performing several actions.

Action (no. of persons × no. of poses in a sequence)	Proposed method		DG [1]		Semantic [5]		Lie Body [2]	
	avg	max	avg	max	avg	max	avg	max
one leg jump (9× 9)	0.55	3.64	4.10	14.51	2.99	20.17	1.59	13.69
one leg loose (9× 9)	0.37	2.87	3.04	12.65	3.31	20.87	1.28	10.65
chicken wings(8× 9)	1.02	8.27	3.11	20.15	3.34	22.92	2.21	24.60
shake hips (9× 9)	1.78	6.41	4.48	23.77	2.89	15.45	2.84	21.58
jiggle on toes (9× 9)	0.72	6.08	4.10	26.69	3.36	40.50	2.01	30.84
jumping jacks (7× 9)	1.90	12.44	4.99	20.53	14.32	55.83	4.56	30.46
hips (8× 9)	0.12	0.92	4.17	11.68	2.31	10.22	0.73	4.61
shake shoulders (8× 9)	1.14	6.42	4.44	16.40	3.45	20.17	2.95	16.26
shake arms (8× 9)	1.96	10.02	4.49	21.19	6.62	34.21	3.94	29.28
running on spot (9× 9)	2.10	12.77	6.93	30.40	12.94	56.50	6.34	37.01
light hopping stiff (9× 9)	0.39	3.49	2.90	9.01	1.59	17.59	0.77	5.06
punching (8× 9)	2.12	25.03	5.12	21.42	8.97	49.67	3.25	25.26
knees (9× 9)	0.29	8.92	4.50	19.87	3.02	35.24	1.37	22.66

Table 4.2: Comparison of proposed method with state-of-the-art methods in terms of average and maximum of root sum squared errors computed at each vertex of all the estimated poses of an action performed by several targets.

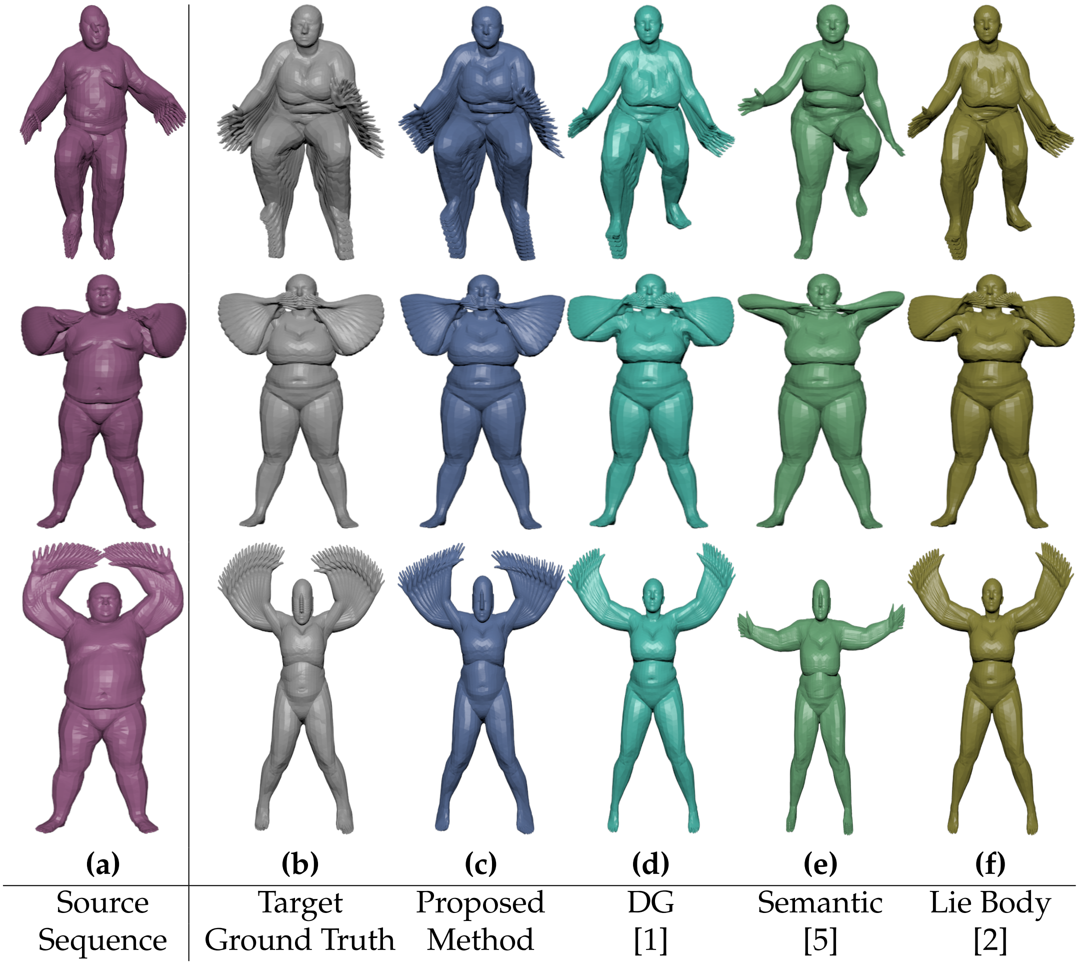


Figure 4.8: **(a)** Sources action sequences *running on spot*, *chicken wings* and *jumping jacks* (top to bottom). (**(b)** and **(g)**) Natural action sequences (ground truth) of two different individuals from dataset [4]. (**(c)** and **(h)**) Estimated action sequences by proposed method. (**(d)** and **(i)**) Estimated sequence by SumPop [1]. (**(e)** and **(j)**) Estimated sequence by Semantic [5]. (**(f)** and **(k)**) Estimated sequence by Manifold [2].

4.1.4 Results and Discussion

The Deformation Transfer is implemented in Python 3.7 and tested on a core-i7-2.8GHz computer. We have developed a deformation transfer add-on for 3D mesh processing software *Blender 2.81*. All the results excluding skeleton, Figure 4.5 and Figure 4.6, are generated using blender implementation. The code is available at `https://github.com/prashantdomadiya/DeformationTransferToolBlender`.

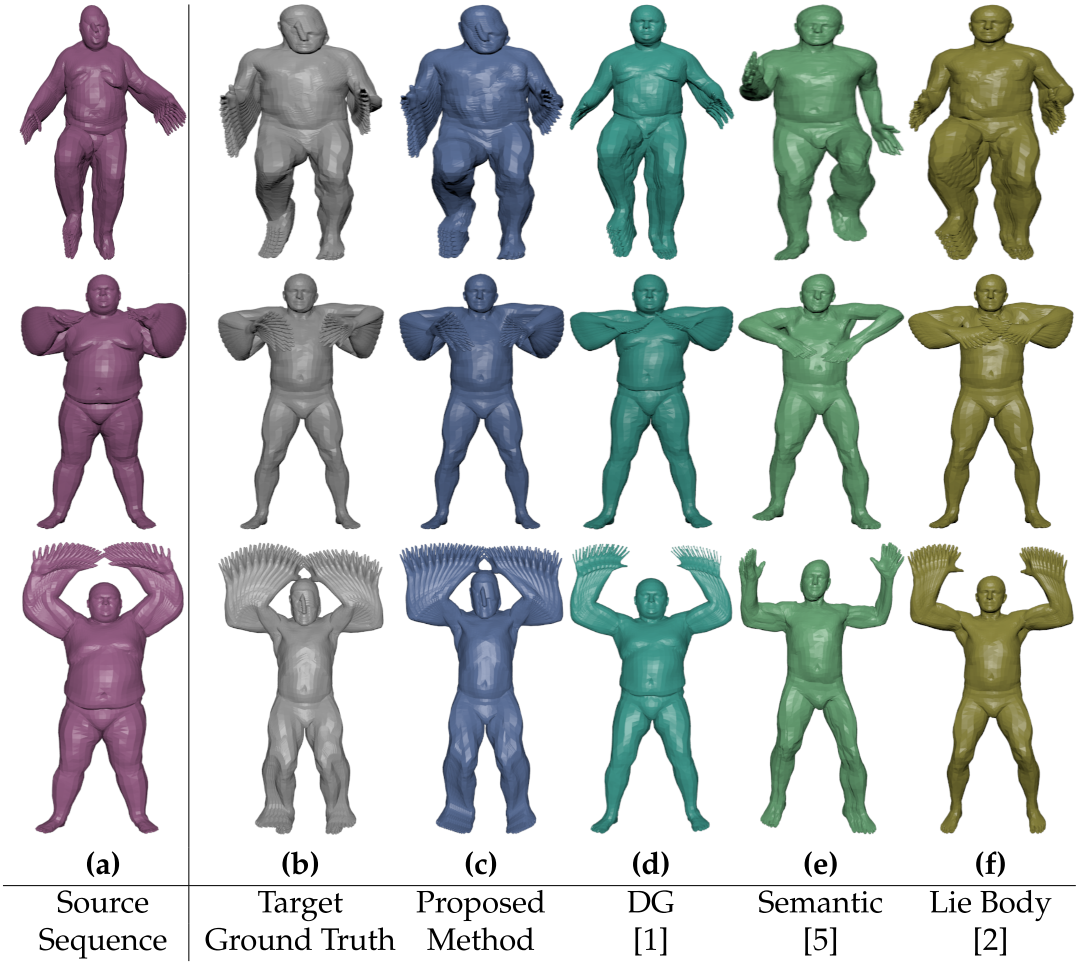


Figure 4.9: **(a)** Sources action sequences *running on spot*, *chicken wings* and *jumping jacks* (top to bottom). (**(b)** and **(g)**) Natural action sequences (ground truth) of two different individuals from dataset [4]. (**(c)** and **(h)**) Estimated action sequences by proposed method. (**(d)** and **(i)**) Estimated sequence by Deformation Gradient (DG) [1]. (**(e)** and **(j)**) Estimated sequence by Semantic [5]. (**(f)** and **(k)**) Estimated sequence by Lie Body [2].

Experiments on Mesh Dataset

We have experimented extensively on the triangular mesh dataset [4]. This dataset consists of triangular meshes of 10 different individuals performing 13 different actions. Each action sequence contains 9 triangular meshes. However, in this dataset, some action sequences of a few individuals are not available. Each mesh in this dataset has 6890 vertices and 13776 faces. The coordinate values of vertices are in meter (m) (real-world scans of humans).

In experiments, an action performed by an individual is taken as the source sequence. A source performing three different actions *running on spot, chicken wings* and *jumping jacks* is shown in Figure 4.8(a) and 4.9(a). Figure 4.8(b) and Figure 4.9(b) are the target sequences taken from the dataset [4] and it is considered as the ground truth. Figure 4.8(c-f) and Figure 4.9(c-f) are the estimated sequences of the target individuals by proposed method, Deformation Gradient [1], Semantic [5] and Lie Body [2] respectively. Figure 4.8 and 4.9 indicate qualitative comparison of the proposed method (with *non-similar boundary condition*) with three state-of-the art methods ([1, 5, 2]) for natural action sequences available in the dataset [4]. Table 4.1 and 4.2, presents the quantitative comparison of the proposed method with three state-of-the-art methods over the same dataset. We have used the vertex to vertex root sum squared errors in coordinates values as a measure of comparison. The average and maximum of root sum squared error are taken over all poses of a sequence under consideration. Average and maximum of root sum squared errors for a person performing several actions and for an action performed by several persons are tabled in Table 4.1 and Table 4.2 respectively.

Modules	Proposed Method	DG[1]	Semantic[5]	Lie Body[2]
Preprocessing	10.9	2.58	4.5 (per pose)	0.89
Pose Estimation	0.025	1.20	10	1.12

Table 4.3: Computation time is in seconds for all methods. The numbers in the table indicate the average time required for preprocessing and pose estimation stages.

Observe that, in Figure 4.8 and 4.9, the temporal properties vary for the different targets performing the same action as performed by the source. As explained in section 4.1.3, the Poisson interpolation not only adjusts temporal properties but also preserves the shape by incorporating shape deformation. As a result, the generated target sequence is close to natural motion sequence as shown in Figure 4.8(c) and Figure 4.9(c). The same is evident from Table 4.1 and 4.2 where the average and maximum of root sum squared errors are also small compared to three state-of-the-art methods. For the methods in [1] and [2], local deformations of the source sequences are directly transferred to the target with smoothness constraints so it generates similarly deformed target sequence as the source. As a result, the target sequence imitates the same motion properties as the source rather

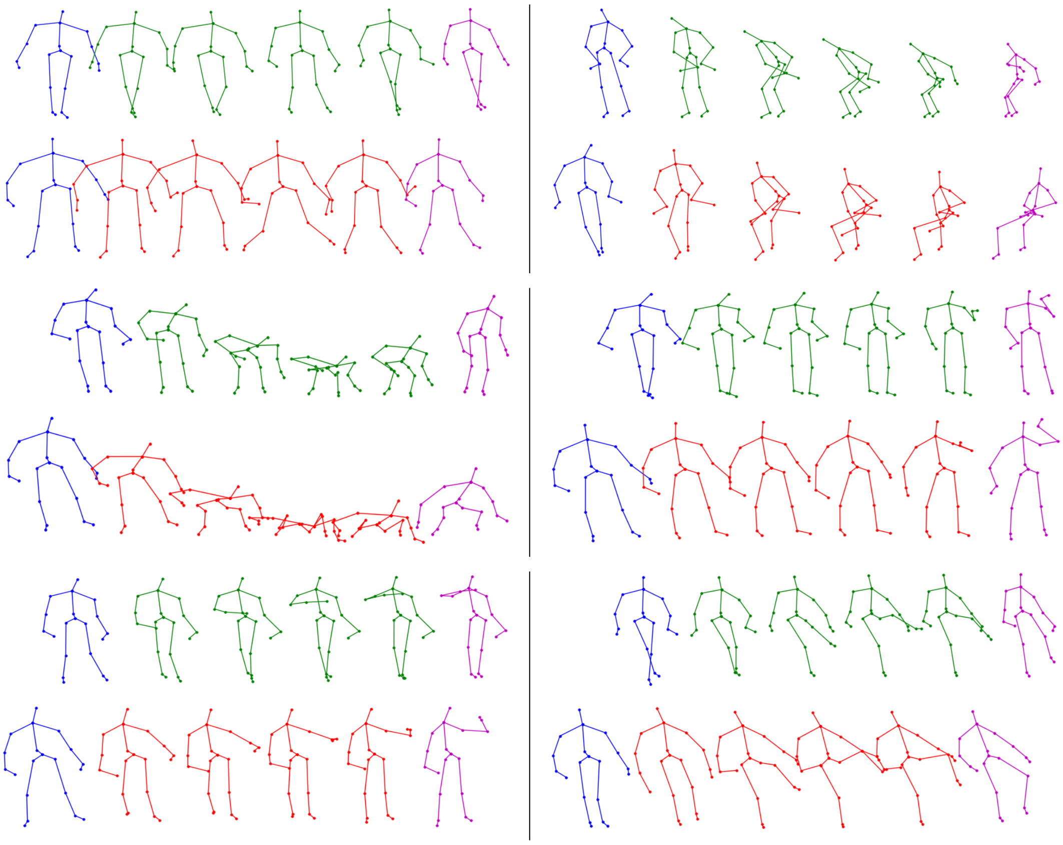


Figure 4.10: Deformation transfer with *non-similar boundary constraint* for different actions *walk*, *seat*, *grab*, *phone*, *watch clock*, *scratch head*, *cross arms*, *kick* (top to bottom left to right): Source sequence is shown in green and the estimated target sequence is shown in red. The initial poses and the final poses are shown in blue and magenta respectively.

than adjusting it according to the target. The method in [5] is able to maintain the shape and motion direction of the target because the source deformation is not directly adopted but its projection on the previous pose is adopted. However, due to this strategy, the deformation applied to the estimated poses of the target is highly dependent on the deformation of the previous target poses. As a result, if the deformations in previous target poses are small, this method fails to adjust the motion of the target (see the Figure 4.8(e) and 4.9(e)) which is also reflecting high root sum squared errors.

Table 4.3 shows the comparison of implemented methods in terms of time required to perform the deformation transfer on this dataset for each method. The preprocessing step is required only once to estimate the shape deformation for proposed method, generate adjacency matrix for Deformation Gradient [1] and

Lie Body [2], and compute linear rotation invariant coordinates, for example, poses (we have taken 5) for Semantic [5]. The estimation step includes the cost associated with estimation of each pose of the target and also the cost of mesh reconstruction.

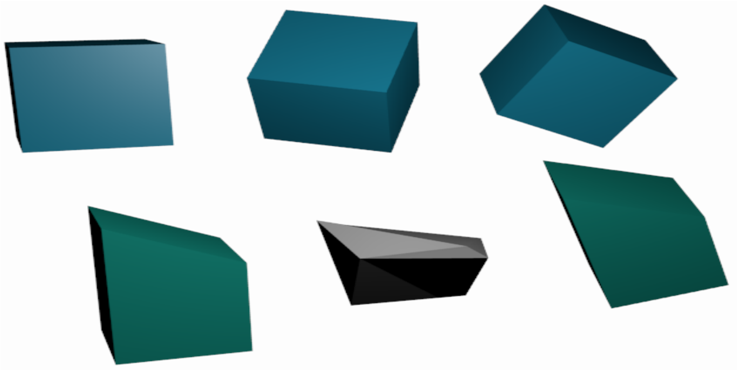


Figure 4.11: Deformation transfer on quad mesh. The deformation for a cube (Top row) is applied to the another mesh (bottom row).

Experiments on Skeleton dataset

The experiment on the UPCV action dataset [68] shows that the proposed method can also be applied to the skeletons. This dataset contains twenty skeletons of the different individual with twenty joints performing ten different actions. In Figure 4.10, we have shown 8 such action sequence of the target skeleton generated by the proposed method using source skeleton sequences.

The proposed method transfer deformation from the source to the target skeleton sequence by adjusting the speed and motion direction without changing its shape. For instance , in *watch clock* and *scratch head* sequences in Figure 4.10, the source performs action using right hand while the target performs same action by left hand. For such cases also, the estimated poses maintain the left-hand action.

Limitations

Despite the impressive results on triangular meshes and skeleton pose, the proposed method causes artifact when the deformation is large as shown in Figure 4.21. Moreover, it doesn't guarantee the planarity of the quad faces as shown in

Figure 4.11. In the next Deformation Transfer approach, the pose deformation can handle the large deformation, and added planarity constraint preserves the planarity of polygon faces.

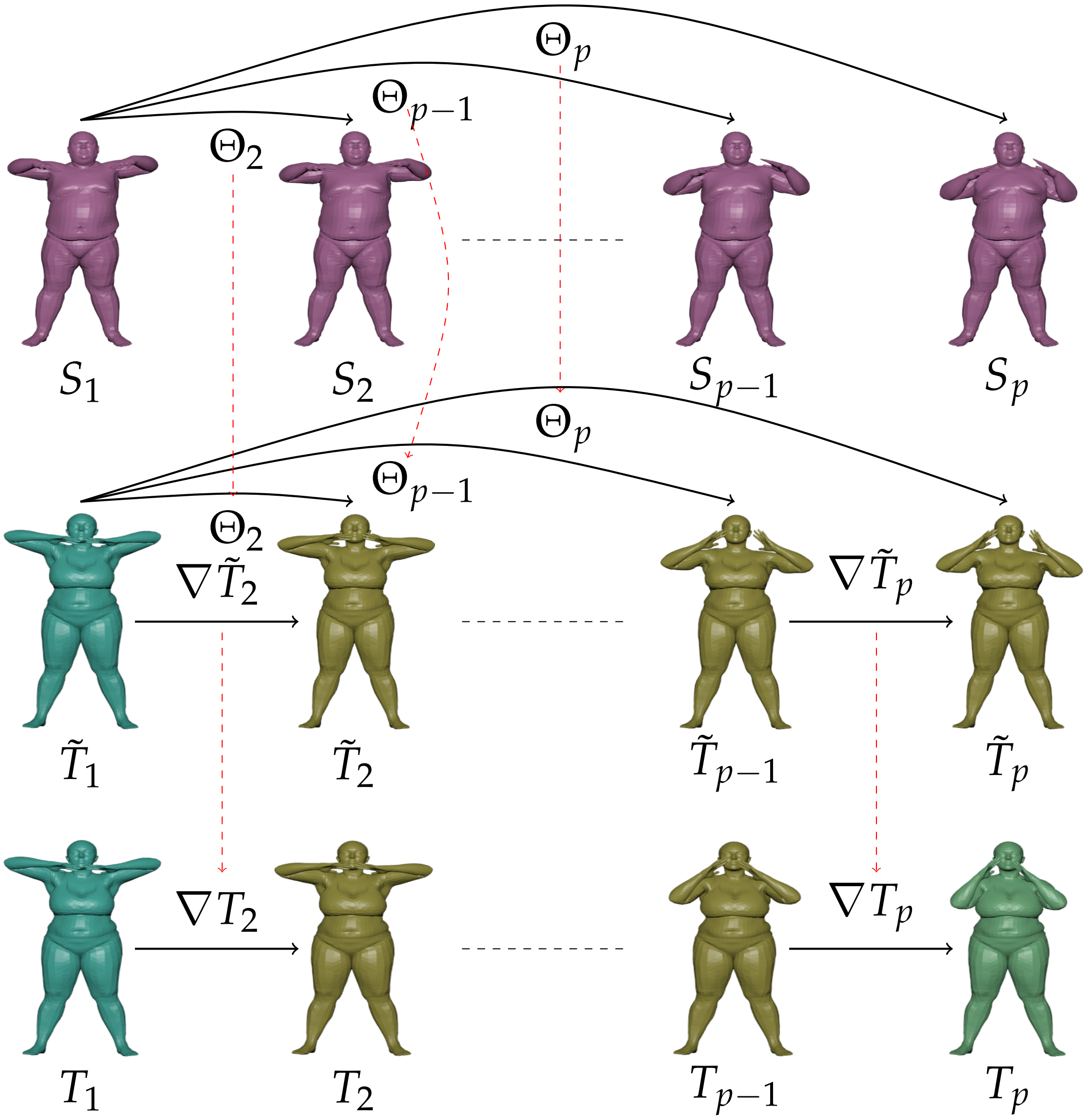


Figure 4.12: The Deformation Transfer framework using the pose deformation. The source sequence $\{S_1, S_2, \ldots, S_{p-1}, S_p\}$ and reference (T_1) and final (T_p) poses of target are input to the Deformation Transfer framework. The pose deformation Θ_d characterized the deformation of the reference source pose S_1 to S_d. The target sequences $\{\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_{p-1}, \tilde{T}_p\}$ and $\{T_1, T_2, \ldots, T_{p-1}, T_p\}$ are generated by the Deformation Transfer and the Guided Deformation Transfer. ∇T_d and $\nabla \tilde{T}_d$ are temporal gradients of pose d of the target sequence estimated by the Deformation Transfer and target sequence estimated by the Guided Deformation Transfer respectively.

4.2 Deformation Transfer Using Pose Deformation

Figure 4.12 outlines the Deformation Transfer framework using the pose deformation. Let's consider a sequence of p source poses $\{S_1, S_2, \ldots, S_{p-1}, S_p\}$. Each pose in the sequence can be formed by deforming the reference source pose and this deformation is characterized by the pose deformation (Θ_k). The computed pose deformation for each deformed source pose is then transferred to the reference target pose to get target sequence $\{\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_{p-1}, \tilde{T}_p\}$. This target sequence is in a way deformed similar to the source sequence. The temporal gradients of the target poses are further refined using the Poisson interpolation to adjust for temporal properties of the target sequence such as individual motion trajectory. The sequence $\{T_1, T_2, \ldots, T_{p-1}, T_p\}$ is the refined target motion sequence using the Poisson interpolation as shown in the Figure 4.12.

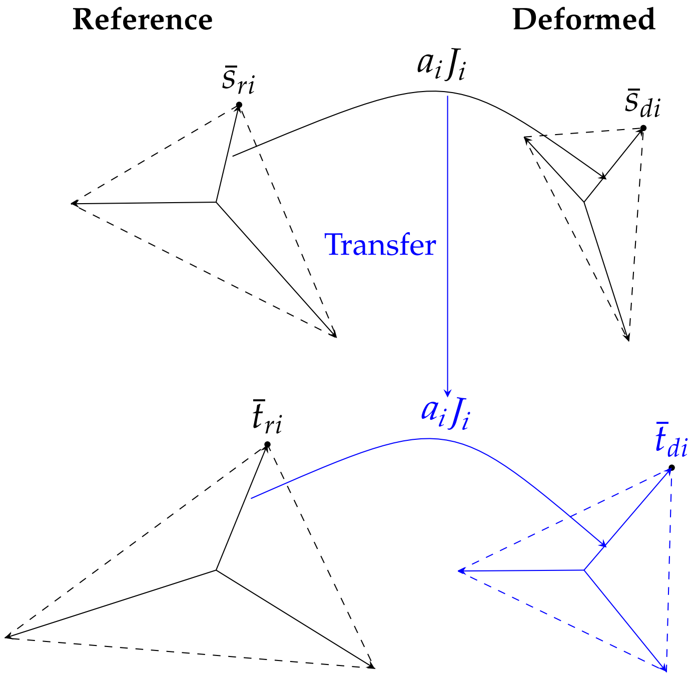


Figure 4.13: The deformations a_i and J_i computed from source reference and deformed vectors, are transferred to the corresponding target reference vectors.

4.2.1 Pose Deformation and Deformation Transfer

The pose deform captures the pose variation of the source. It is computed from the source reference and its deform pose using the VG representation, and it is then transferred to the reference target pose. The process is illustrated in Figure 4.13. Let's consider the reference and the deform source vectors $\bar{s}_{ri}$ and $\bar{s}_{di}$. The deformation of the vector $\bar{s}_{ri}$ to the vector $\bar{s}_{di}$ is characterized as,

$$\bar{s}_{di} = a_i J_i \bar{s}_{ri} \tag{4.18}$$

where b_i and R_i are scaling factor and rotation matrix respectively. The rotation is computed by forming orthonormal frames as explained in section 3.3. The scaling is computed as $a_i = \|\bar{s}_{di}\| / \|\bar{s}_{ri}\|$. The computed deformation $(a_i J_i)$ is then transferred to the corresponding reference target vector $\bar{t}_{ri}$ to get deform target vector $\bar{t}_{di}$ as,

$$\bar{t}_{di} = a_i J_i \bar{t}_{ri}. \tag{4.19}$$

Similarly, the deformation computed from all the source vectors is then transferred to the corresponding target reference vectors. The correspondence can be established using a mesh registration method such as [1]. The pose deformation (Θ) is formed as,

$$\Theta = \begin{bmatrix} a_1 J_1 & 0 & \dots & 0 \\ 0 & a_2 J_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & a_n J_n \end{bmatrix} \tag{4.20}$$

where, n is the number of vectors in the VG representation of the target mesh. The computed pose deformation Θ transforms the VG representation of reference target pose $(\bar{T}_r)$ to the VG representation of deformed target pose $(\bar{T}_d)$. The relationship is defined as,

$$\bar{T}_d = B\tilde{T}_d = \Theta \bar{T}_r \tag{4.21}$$

where $\tilde{T}_d$ is the deformed target mesh and matrix B is computed from the mesh connectivity as explained in section 3.1. The deform target mesh $\tilde{T}_d$ is approximated by following the reconstruction process.

Reconstruction

Since the VG is a translation invariant representation, the mesh can be reconstructed anywhere in $\mathbb{R}^3$. For this reason, the equation (4.21) is an overdetermined system of linear equations. This problem can be solved by imposing positional constraints during the mesh reconstruction process as,

$$\begin{aligned} \underset{\tilde{T}_d}{\text{argmin}} \quad & \|B\tilde{T}_d - \bar{T}_d\|_2^2 \\ \text{subject to} \quad & \tilde{t}_{di} = v \end{aligned} \tag{4.22}$$

where, $\tilde{T}_d$ is a fully connected mesh in $\mathbb{R}^3$ and $\tilde{t}_{di}$ is a constrained vertex of deformed target mesh $\tilde{T}_d$. Here, we set the constraint vertex $\tilde{t}_{di}$ at the global origin $0 \in \mathbb{R}^3$. The matrix B is computed from the target mesh connectivity as shown in equation 3.6. Above optimization problem can be now solved as explained in section 3.1. The reconstructed deformed target mesh is

$$\hat{T}_d = \hat{B}^+(\bar{T}_d - B_l v) = \hat{B}^+(\bar{T}_d - B_l 0) = \hat{B}^+\bar{T}_d \tag{4.23}$$

where, the matrices $\hat{B}$ and B_l are computed from the mesh connectivity as explained in section 3.1. The constraint vertex $\tilde{t}_{di}$ is appended to $\hat{T}_d$ to get fully connected deform target mesh $\tilde{T}_d$.

4.2.2 Dot Product Property

One of the main objectives of the Deformation Transfer is to preserve the geometric details of the target for the wide range of deformations. Here, we explain the importance of dot product property for this context. As explained in section 3.4.1, the vector deformation decomposition, rotation, and scaling, preserves the dot product property for deformation transfer. The dot product between deform

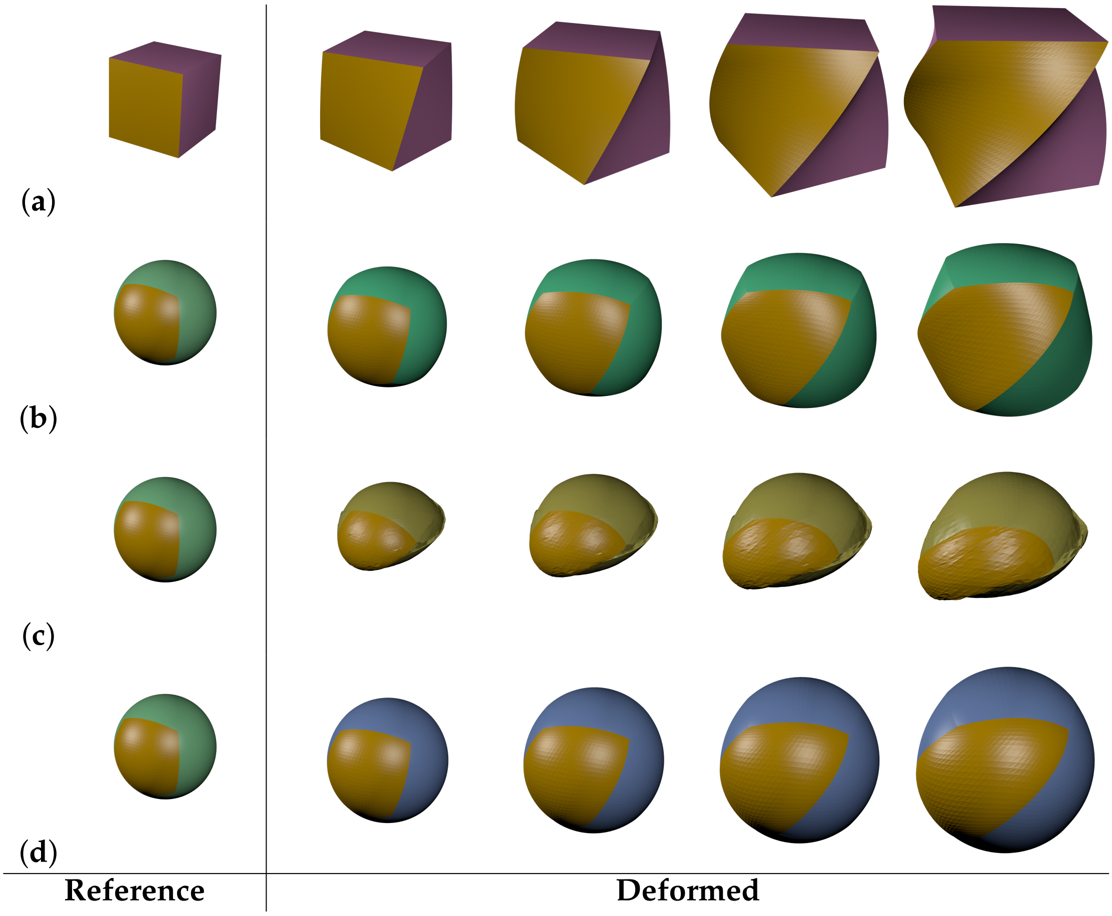


Figure 4.14: (**a**) A source sequence of a cube (a triangular mesh) in which twisting and scaling effects is applied. Deformation from the source (cube) is transfered to the target (sphere) (**b**) by Deformation Gradient [1],(**c**) by Lie Body [2] and (**d**) by proposed method (only Deformation Transfer).

source ($\bar{s}_{di}$) and target ($\bar{t}_{di}$) is computed as,

$$||\bar{s}'_{di}||\,||\bar{t}_{di}||cos\theta_{di} = \bar{s}'_{di}\bar{t}_{di} = (a_iJ_i\bar{s}_{ri})'(a_iJ_i\bar{t}_{ri}) = a_i^2\bar{s}'_{ri}\bar{t}_{ri} = \frac{||\bar{s}_{di}||}{||\bar{s}_{ri}||}\frac{||\bar{t}_{di}||}{||\bar{t}_{ri}||}||\bar{s}_{ri}||\,||\bar{t}_{ri}||cos\theta_{ri} \tag{4.24}$$

$$cos\theta_{di} = cos\theta_{ri} \tag{4.25}$$

where, θ_{ri} is the angle between reference source and target vectors, and θ_{di} is the angle between deformed source and target vectors. Each vertex of a face can be deformed independently and it, in turn, deforms a face. In [1], such deformation is modeled as the linear transformation (Q) which may not preserve the dot product property. As a result, it may not maintain the orientation difference,

$$v_k' u_k = v_1' Q' Q u_1 \neq c^2 v_1' u_1. \tag{4.26}$$

Refer to [1] for notations. To maintain the dot product property, the deformation transferred to the target face should be in the form of either rotation, scaling, or composition of them. The VG allows us to model the deformation of each vector (vertex w.r.t centroid) in the form of rotation and scaling. In Lie Body [2], the triangle deformation is decomposed into the rotation, scaling, and positive definite matrix as suggested in Lie body representation [14]. Since the positive definite matrix is not orientation preserving transformation in this decomposition, the triangle deformation can't preserve the dot product property. Moreover, in Lie Body [2], the transformation that transforms a source triangle to its canonical form, is transferred to the target reference. If corresponding reference the source and target triangles are in different orientations, then such transformation won't convert the reference target face to its canonical form. As a result, the undesirable artifacts occur in the target mesh as shown in Figure 4.14(c).

This is a major difference between the methods proposed in Deformation Gradient [1], Lie Body [2], and the VG based deformation transfer methods. To highlight the effect of orientation difference preserving property, the deformation (scaling and twisting) of a cube is transferred to a sphere in Figure 4.14. The Deformation Transfer using Deformation Gradient [1], can not preserve the dot product property, so the estimated deformed sphere doesn't have the same relationship with the corresponding deformed cube. As scaling and twisting increases, the sphere is morphed into more of a cube like structure as shown in Figure 4.14(b). The proposed method preserves the dot product property, and so the shape of the sphere can be preserved for these deformations (for more results, refer Figure 4.16).

4.2.3 Processing Hybrid Meshes

During the Deformation Transfer, the face planarity may get disturbed for hybrid meshes due to the non-planarity present in the source. Moreover, the mesh reconstruction may also disturb the planarity of faces. As noted in [69], the quad faces of a mesh should be planar as possible. Optimization in equation (4.22) doesn't guarantee planarity of quad faces of mesh $\tilde{T}_k$. Hence, planarity constraint should be added to optimization problem (4.22) to reduce non-planarity of reconstructed faces. It is easy to impose the planarity constraint in the reconstruction process using the VG representation. For a planar face, the vector i should be perpendicular to normal (M_{ki}) of the face. Hence, reconstruction process is constrained by adding this term to the cost function of equation (4.22) as follows,

$$\begin{aligned} &\underset{\tilde{T}_d}{\operatorname{argmin}} \quad w_1 \| B\tilde{T}_d - \bar{T}_d \|_2^2 + w_2 \sum_i |M_{di} \cdot \tilde{t}_{di}|^2 \\ &\text{subject to} \quad \tilde{t}_{di} = v \end{aligned} \tag{4.27}$$

where, w_1 and w_2 are weights. M_{di} is a unit vector computed as the average of perpendicular unit vectors of successive pairs of vectors of the face. $|M_{di} \cdot \tilde{t}_{di}|$ represents the dot product between $M_d i$ and $\tilde{t}_{di}$. The planarity constraint tries to change vectors in such a way that the dot product with M_{di} becomes close to zero (trying to make M_{di} as unit normal of the face). We choose $w_1 = 0.05$ and $w_2 = 1$ experimentally, because assigning a higher weight to planarity constraint allows it to modify positions of vertices to improve planarity. Above optimization can be rewritten as,

$$\begin{aligned} &\underset{\tilde{T}_d}{\operatorname{argmin}} \quad w_1 \| B\tilde{T}_d - \bar{T}_d \|_2^2 + w_2 \| E_d \tilde{T}_d \|_2^2 \\ &\text{subject to} \quad \tilde{t}_{di} = v \end{aligned} \tag{4.28}$$

where, $E_d = M_d B$. Matrix M_d is a sparse matrix containing normals of all faces (for formation of M_d see Appendix A.1). Similar to equation (4.22), we can remove positional constraint as follows,

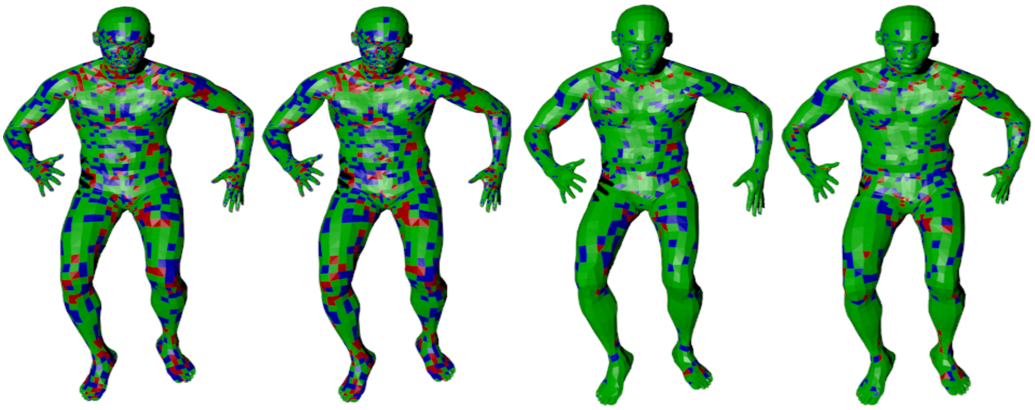
$$\underset{\hat{T}_d}{\operatorname{argmin}} \quad w_1 \| \hat{B}\hat{T}_d - b_d \|_2^2 + w_2 \| \hat{E}_d \hat{T}_d - e_d \|_2^2 \tag{4.29}$$

where, $\hat{E}_d$ is void of columns from $3i+1$ to $3i+3$, $e_d = -E(:, 3i+1 : 3i+3)v$ and $b_d = \bar{T}_d - B_l v$. Taking derivative with respect to $\hat{T}_d$ in above equation (see the derivation in Appendix A.1), we get,

$$(w_1 \hat{B}'\hat{B} + w_2 \hat{E}_d'\hat{E}_d)\hat{T}_d = X_d\hat{T}_d = w_1\hat{B}'b_d + w_2\hat{E}_d'e_d = Y_d \tag{4.30}$$

$$\hat{T}_d = X_d^{-1}Y_d. \tag{4.31}$$

For experiment purposes, we convert triangular meshes of dataset [4] into hybrid meshes using the Blender tool *TrisToQuad* allowing nontriangles to merge up to some threshold. As a result, input meshes would now contain non-planar faces. Due to the reconstruction process and non-planarity present in input meshes, target meshes estimated by triangular face based methods in Deformation Gradient [1] and Lie Body [2] contain non-planar faces. Proposed method reduces non-planarity while estimating target faces as shown in Figure 4.15(c).



(**a**) DG (**b**) Lie Body (**c**) DT (**d**) GDT

Figure 4.15: Planarity error in faces indicated by color code: green-error less than 0.05, blue- error between 0.05 to 0.1, and red- error greater than 0.1. These thresholds are set from Fig. 4.18. Estimated target meshes by (**a**) Deformation Gradient (DG) [1] and (**b**) Lie Body [2] (we convert triangles into quad by Blender tool *TrisToQuad*). (**c**) The Proposed method prevent penetration of non-planarity into estimated target meshes. (**d**) Poisson interpolation doesn't alter the planarity of a mesh much.

4.2.4 Correspondence and Selection of Faces

The proposed Deformation Transfer method applies to nonidentical source and target meshes also. The effectiveness of the same is evident from the results presented in Figure 4.17. Here, we have established correspondence using Deformation Gradient [1]. Though we assume that the correspondence between source and target faces is available, it is not a requirement since we can establish correspondence based on methods available in the literature, one such method is proposed in [1]. Here, we employ the face selection method as explained in section 3.2 for the Deformation Transfer process to save computation overload. This process is repeated until the entire vertex set is exhausted. The selection process is explained in Algorithm 1. For triangular meshes, the number of selected faces are approximately half the total number of faces. In the Dyna dataset [4], each mesh has 13776 triangular faces and 6890 vertices. Using the above mentioned process, we identify 7038 target faces. Since number of faces are approximately half so is the computational time (see Table 4.5). We arrange indices of selected faces of the target, and their corresponding source faces as two column vectors in such a way that indices of the corresponding faces remain at identical places.

4.2.5 Guided Interpolation

The estimated target sequence $\{\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_p\}$ has similar deformations and motion trajectory as the source. However, these properties vary with the target and should be adjusted accordingly. The objective here is to estimate unknown target sequence $\{T_1, T_2, T_3, \ldots, T_{p-1}\ , T_p\}$ which consists of refined temporal properties. Poisson interpolation performs this task based on given initial and final poses T_1 and T_p. Since we have the target sequence $\{\tilde{T}_1, \tilde{T}_2, \ldots, \tilde{T}_p\}$, its temporal gradient information is then used as guiding vector field. In terms of the Poisson interpolation frame work explained in [30], we consider $X = \{T_1, T_2, \ldots, T_p\}$, $\Omega = \{T_2, \ldots, T_{p-1}\}$ and $\partial\Omega = \{T_1, T_p\}$. The deformation transfer problem can now be put in the form of Poisson interpolation as follows: Given X, Ω and $\partial\Omega$ above, generate $\{T_2, T_3, \ldots, T_{p-1}\}$, with the condition that the temporal gradient

∇T_k is the same as $\nabla \tilde{T}_k$. The objective function can be formulated as follows,

$$J(T_k) = \underset{T_k \in \Omega}{argmin} \sum_{k=2}^{p-1} \|\nabla \tilde{T}_k - \nabla T_k\|_2^2 \text{ with } T_1, T_p \in \delta\Omega. \tag{4.32}$$

To solve for T_k, setting the gradient of the objective function to zero, gives,

$$\forall k,\ \Delta T_k = \Delta \tilde{T}_k \tag{4.33}$$

where, Δ is a Laplacian operator. $\Delta T_k = 2T_k - T_{k+1} - T_{k-1}$ and $\Delta \tilde{T}_k = 2\tilde{T}_k - \tilde{T}_{k+1} - \tilde{T}_{k-1}$ are denoted as Laplacian of T_k and $\tilde{T}_k$ respectively. Equation (4.33) is Poisson equation with known boundary constraints T_1 and T_p. Since the generated system of linear equations (4.33) is symmetric and has a unique solution, this can be solved by the Gauss-Seidel iterative successive over relaxation method as follows,

$$\forall k,\ T_k^{(x+1)} = (1-\omega)T_k^x + \frac{\omega}{2}(\Delta \tilde{T}_k + T_{k-1}^x + T_{k+1}^x) \tag{4.34}$$

where, ω is a constant, we choose $\omega = 1.9$ experimentally. There is close form solution of equation (4.33) available but the computational complexity is higher than iterative process when number of poses are large. Computational complexity of iterative and closed-form methods are $9mpi$ (i is number of iterations) and $p^3 + (3m-2)p^2 - 6m - 1$ respectively.

4.2.6 Results and Discussion

The implementation of the proposed method is tested on a core-i7-2.8 GHz computer with 8GB memory. All the experiments are carried out using an addon created for Blender 2.82 in Python 3.7. The implementation works well with different types of geometric models including skeletons and triangular/quad/hybrid meshes. Please find our Blender addon at https://github.com/prashantdomadiya/Guided-Deformation-Transfer.

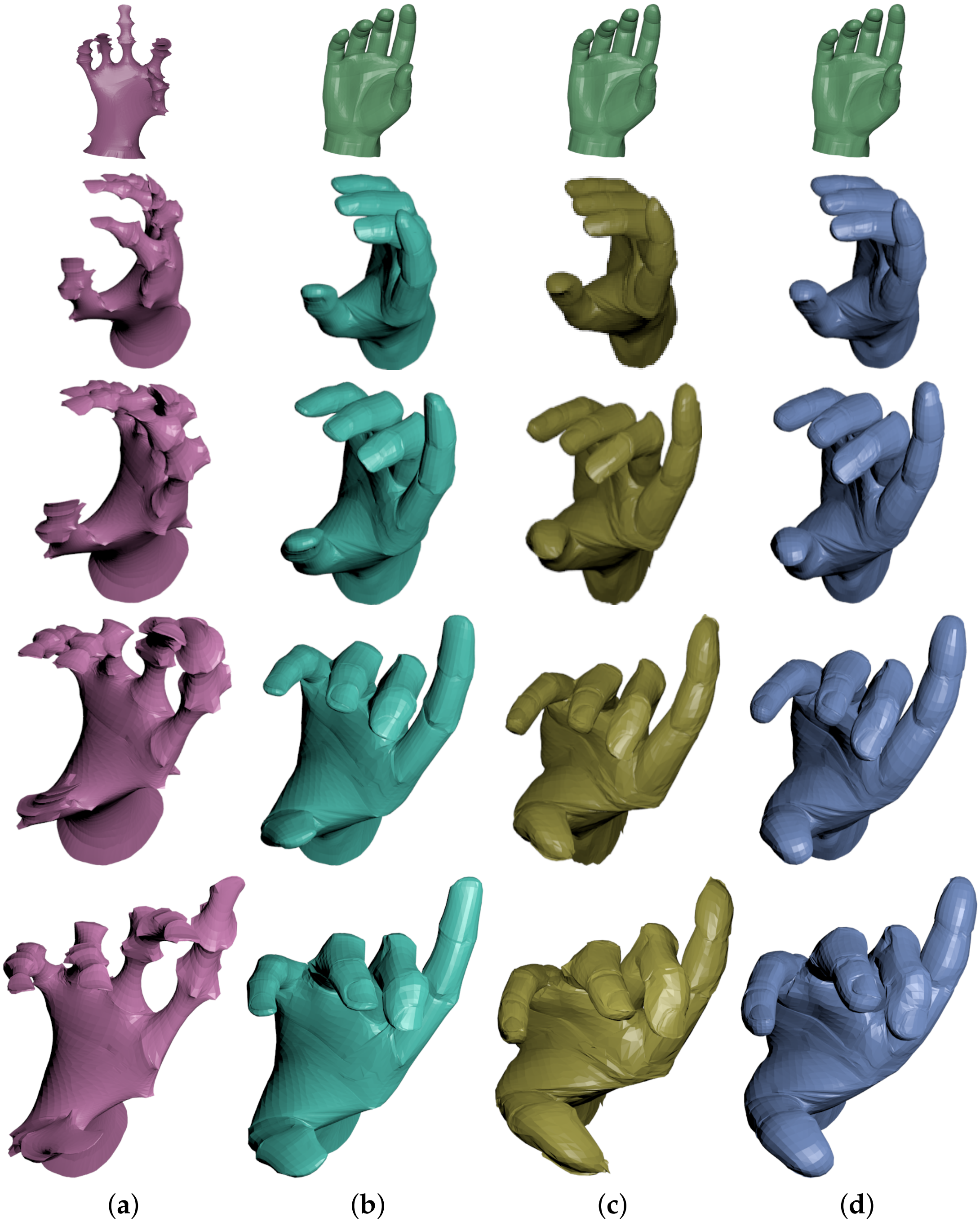


Figure 4.16: **(a)** Source: A devil's hand becoming big and twist is increasing with pose. Estimated target (a hand) sequences (**b**) Deformation Gradient [1], (**c**) Lie Body [2] and (**d**) by proposed Deformation Transfer.

Experiments on Triangular Mesh

In Figure 4.17, we have shown comparison of proposed method with two state-of-the-art methods. In this experiment, most of the faces of the source sequence

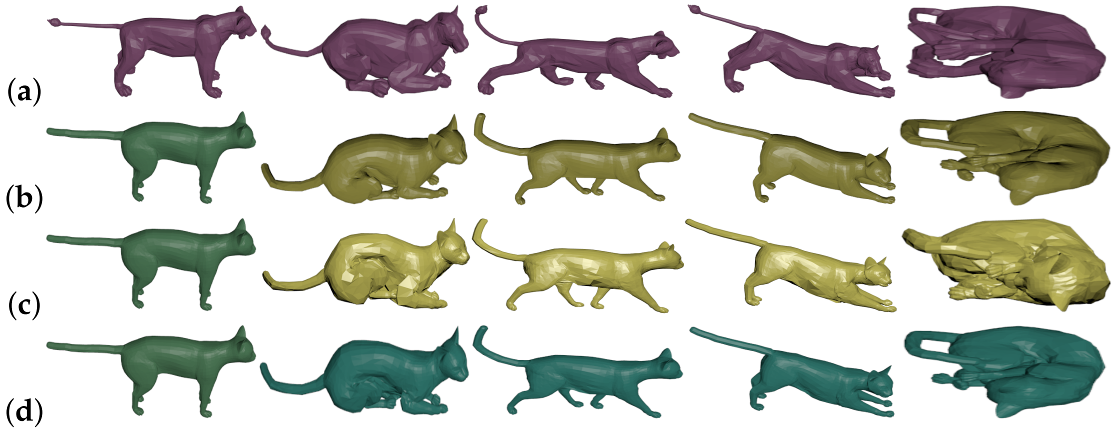


Figure 4.17: (**a**) Source: Deformation sequence of a lion. Deformation from lion poses is transferred to cat initial pose (left most) by (**b**) Deformation Gradient [1], (**c**) Lie Body [2] and (**d**) proposed method (only Deformation Transfer) .

experience transformations such as *scaling and rotation*, so the transformation matrix computed using either Deformation Gradient [1] or Lie Body [2] preserves the dot product property. As a result, estimated sequences by the proposed Deformation Transfer method and by two state-of-the-art methods Deformation Gradient [1] and Lie Body[2] are quite similar. However, as explained in section 4.2.2, these methods fail to preserve the dot product property for other transformations resulting in the shape of the target can't be maintained as evident in Figure 4.16. In Figure 4.16, estimated target hand poses by Deformation Gradient [1], and Lie Body [2] have lost their geometric details (observe the fingertips and palm lines) with an increase in the size respectively. In both the experiments, the proposed method preserves features of the target (shape and geometric details).

To compare proposed Deformation Transfer and Guided Deformation Transfer methods, we have experimented on Dyna dataset [4] which consists of triangular meshes of 13 actions performed by 10 persons. A few actions of some individuals are not available in the dataset. Each mesh in the dataset consists of 13776 triangular faces and 6890 vertices. The co-ordinates of vertices are in meters. For experiments, we consider 11 poses per sequence. In other words, for an action sequence of an individual we consider the action sequence characterized by 11 poses. The correspondence between vertices and faces between two different poses are available for this dataset. For quantitative comparison, we use root sum squared er-

Table 4.4: Comparison of proposed Deformation Transfer (DT) and Guided Deformation Transfer (GDT) methods in terms of average and maximum of root sum squared errors. These errors are computed at each vertex of all the estimated poses for an action performed by several targets. **Note:** All errors are in *mm*.

Action (no. of	GDT		GDT		DT	
persons × no. of	(faces=7038)		(faces=13776)		(faces=13776)	
poses in a sequence)	avg	max	avg	max	avg	max
one leg jump (9× 9)	0.89	4.81	0.8	4.75	1.77	14.08
one leg loose (9× 9)	0.79	4.00	0.79	3.9	1.50	11.72
chicken wings(8× 9)	1.29	8.90	1.29	8.81	2.39	24.71
shake hips (9× 9)	1.31	8.25	1.32	6.80	2.96	22.06
jiggle on toes (9× 9)	1.02	6.29	1.02	6.10	2.24	30.96
jumping jacks (7× 9)	1.64	10.58	1.66	10.51	4.96	30.69
hips (8× 9)	0.69	3.08	0.69	3.08	1.05	5.95
shake shoulders (8× 9)	1.36	6.33	1.34	6.31	3.05	16.16
shake arms (8× 9)	2.14	12.51	1.96	9.54	4.02	29.34
running on spot (9× 9)	2.26	13.39	2.19	13.34	6.38	37.09
light hopping stiff (9× 9)	0.80	3.58	0.80	3.57	1.08	6.25
punching (8× 9)	1.95	19.60	1.91	17.47	3.32	25.57
knees (9× 9)	0.78	8.94	0.78	8.94	1.61	23.16

rors between vertex coordinates of estimated and ground truth sequences. The measure of comparison is the average root sum squared and maximum root sum squared taken over all the poses of the sequence. Average and maximum of root sum squared errors for an action performed by several persons are presented in Table 4.4. As explained in section 4.2.5, Guided Deformation Transfer adjusts motion trajectories of estimated target sequences by proposed Deformation Transfer such that they are as similar as ground truth. As the result, root sum squared error of Guided Deformation Transfer is less compared to the Deformation Transfer.

Table 4.5 compares computation time required by each of the methods to perform deformation transfer on various source target pairs. The preprocessing step is required only once to generate the adjacency matrices for all methods. The proposed face selection algorithm can be applied to all the methods to reduce the computation time. The number of selected faces by the algorithm along with reduction in time are shown in brackets in respective columns. As shown in Table 4.5, the number of faces are reduced to approximately 49% and so is the computation time. The preprocessing time for Manifold Representation is lower than the

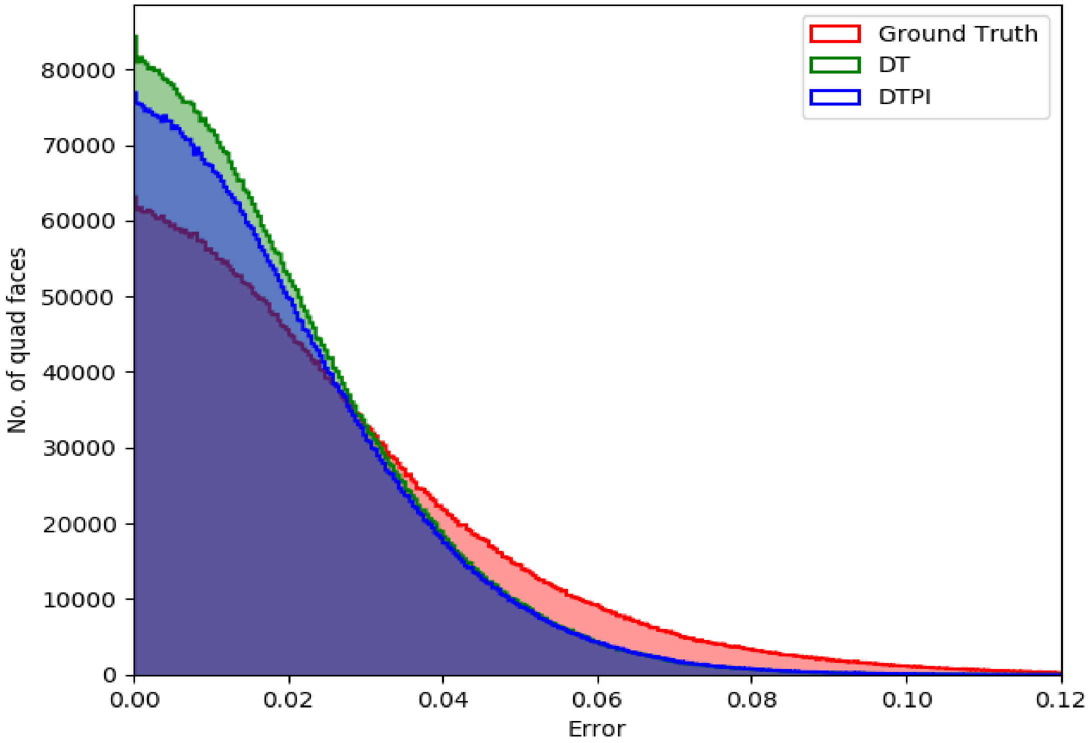


Figure 4.18: Combined histogram of planarity error in quad faces for ground truth target sequence converted from triangular meshes of dataset [4] using *'Tris to Quads'* in Blender (in red), generated hybrid target mesh sequence by proposed Deformation Transfer (in green) and Guided Deformation Transfer (in blue).

Source-Target pair	# faces (# selected faces)	Preprocessing			Pose estimation		
		DG [1]	Lie Body[2]	Proposed	DG [1]	Lie Body[2]	Proposed
Cat-lion	14110 (7426)	2.74 (1.44)	0.91 (0.47)	2.43 (1.31)	1.19 (0.63)	2.41 (1.27)	2.39 (1.27)
Devil-human hands	15789 (8075)	2.74 (1.43)	1.02 (0.53)	2.71 (1.41)	1.28 (0.67)	2.94 (1.51)	2.59 (1.33)
Alien-human faces	31620 (16456)	8.73 (4.56)	1.97 (1.04)	5.38 (2.81)	2.44 (1.29)	15.79 (8.23)	5.22 (2.73)
Bob-BigB	20000 (10272)	4.38 (2.27)	1.26 (0.66)	3.61 (1.88)	1.57 (0.83)	4.65 (2.41)	3.30 (1.72)
Dyna models	13776 (7069)	2.58 (1.32)	0.89 (0.46)	2.36 (1.20)	1.20 (0.63)	2.36 (1.21)	2.30 (1.18)
Dyna models (Hybrid meshes)	7929 (5349)	NA	NA	6.82 (5.09)	NA	NA	1.59 (1.25)

Table 4.5: Comparison of computational time of Deformation Transfer (DT) methods in seconds. The preprocessing is required only once for each new sequence. The values in brackets in column 2 is the number of selected faces. All the time durations are in seconds. The number in brackets in columns 3-8 indicate computation time required while working with selected faces only.

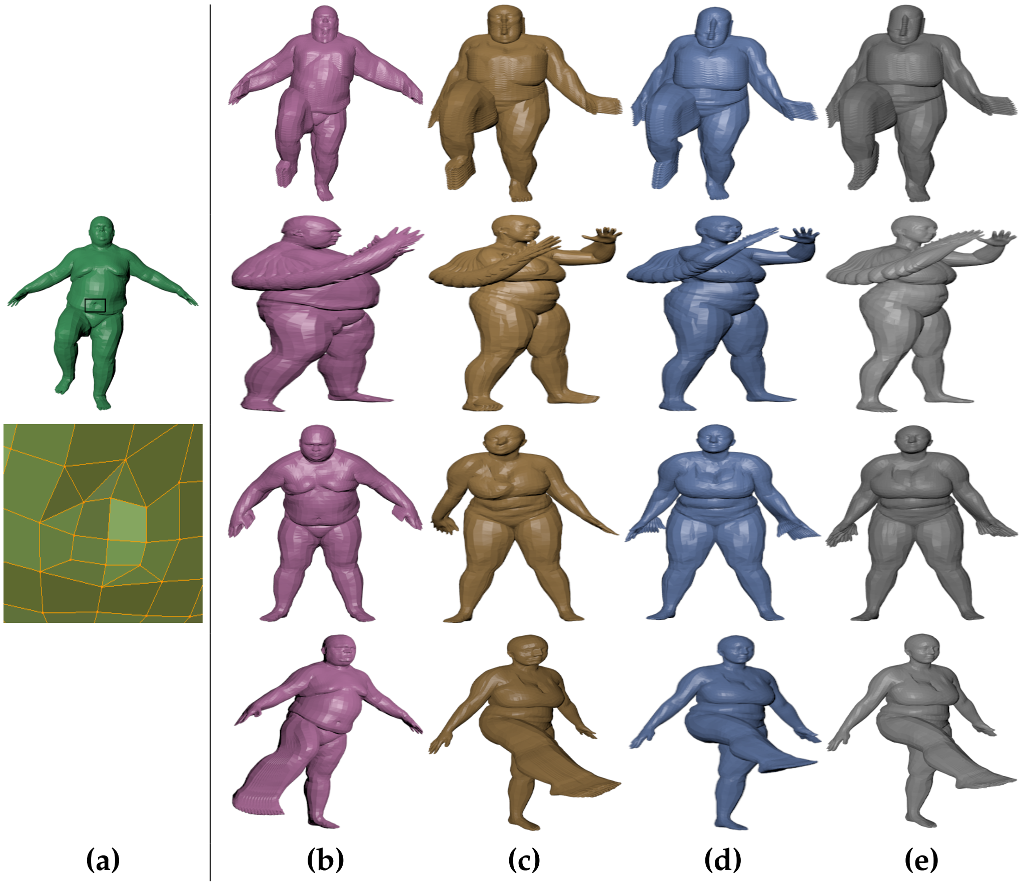


Figure 4.19: (**a**) A hybrid mesh and its example patch. (**b**) Four action sequences of hybrid meshes of the source. Generated action sequences of hybrid meshes by proposed Deformation Transfer (DT) (**c**) and Guided Deformation Transfer (GDT) (**d**) with *non-similar boundary condition*. Ground truth available in Dyna dataset [4] (**e**).

proposed method and Deformation Gradient due to relatively simpler structure of connectivity matrix but the pose estimation time is higher due to complex computation of deformation. The pose estimation time of the proposed method is higher than the Deformation Gradient because the degree of freedom per triangular face of VG is higher than Deformation Gradient.

Experiments on Hybrid Mesh

During experiments, we have converted triangular meshes of the dataset [4] to hybrid meshes using *Blender* tool *'Tris to Quads'* as shown in Figure 4.19 (column 1). The tool merges two triangles into a quad when the angle between them is less than a threshold of 40°. With a fixed threshold, the converted hybrid representation of a triangular mesh (13776 faces) contains approximately 7500 faces.

We have estimated 13 different action sequences performed by 9 different individuals using the proposed method. Each sequence consists of 9 estimated poses and two boundary poses. The estimated hybrid mesh, on average, consists of approximately 7500 faces. Four different source action sequences (namely *one leg jump, punching, shake shoulders* and *knees*) are shown in Figure 4.19(b). In Figure 4.19(c) and (d), target sequences generated by Deformation Transfer and Guided Deformation Transfer with *non-similar boundary condition* are presented respectively. Observe that the temporal position and motion trajectory of the target are adjusted according to the boundary poses. As a result, sequence in Figure 4.19(d) is closer to ground truth sequence in Figure 4.19(e). Table 4.5 compares computational efforts of proposed Deformation Transfer method in seconds for triangular and hybrid mesh inputs. The computational complexity of processing the hybrid mesh is higher than the triangular mesh due to added planarity constraint.

To test for face planarity, planarity error at each face of estimated pose, is computed. Let $X = \{x_1, x_2, \ldots, x_n\}$ be the set of vectors of VG representation of a face. N is a unit vector computed as an average of perpendicular vectors of consecutive pairs of vectors of set X. We define the planarity error of a reconstructed face as $e_p = \frac{1}{n}\sum_i |\hat{x}_i \cdot N| = \frac{1}{n}\sum_i |cos\beta_i|$ where, $\hat{x}_i = x_i / \|x_i\|_2$ and β_i is angle between $\hat{x}_i$ and N. The error varies between 0 to 1. The face starts loosing its planarity as e_p moves close to 1. In a hybrid mesh sequence derived from a triangular mesh sequence, each pose has a different number of faces. Moreover, due to the threshold, a pair of non-planar triangles are also converted to the quad face resulting in non-planar quad faces. Figure 4.18 is a histogram of planarity error present in quad faces of hybrid meshes derived from triangular meshes of the Dyna dataset [4]. Figure 4.18, shows the planarity error present in the estimated hybrid mesh sequences by the proposed Deformation Transfer method. Due to added planarity constraint, the proposed method prevents non-planarity from penetrating into estimated target poses. This is evident from the histogram with a large peak near zero error with a lower variation. Considering experimental results, the proposed method improves the planarity of the quad face. The Poisson interpolation may

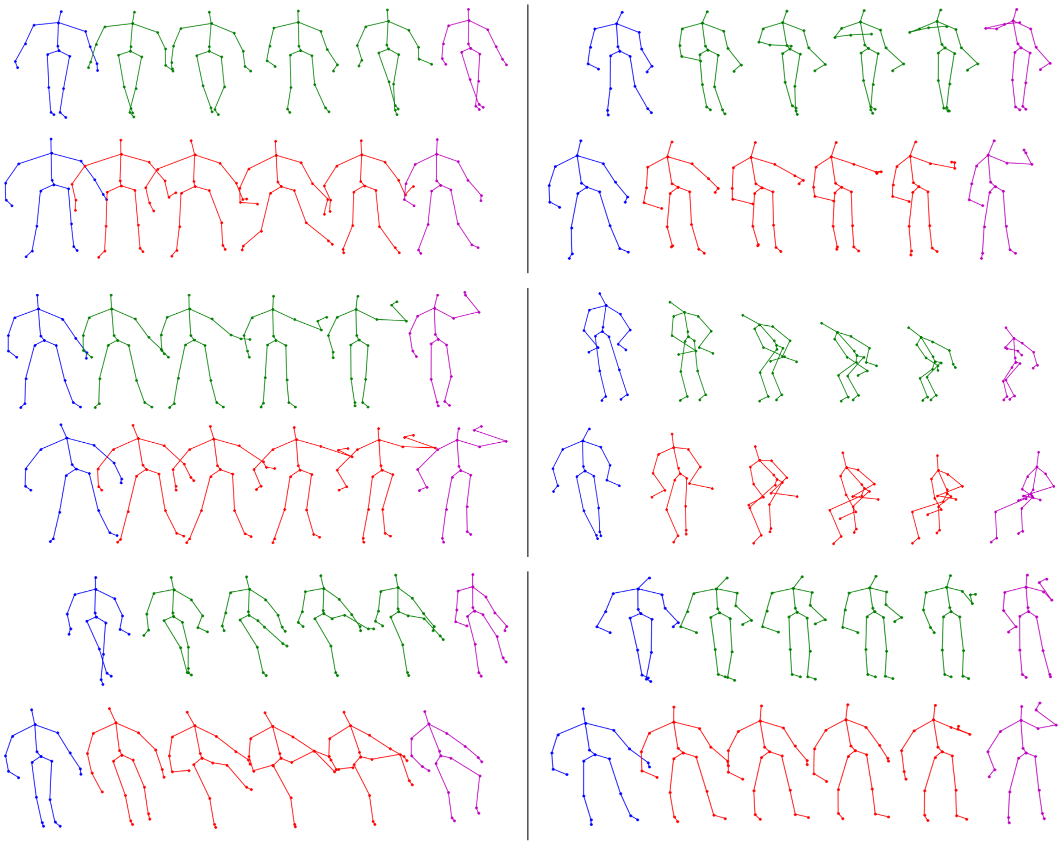


Figure 4.20: Deformation transfer from source (in green) to target (in red) with initial (in blue) and final (in magenta) poses for different actions *walk*, *watch clock*, *scratch head*, *phone*, *kick*, *seat* (in sequence top to bottom and left to right).

disturb the planarity, but this disturbance isn't very high see Figure 4.18 (see explanation in Appendix A.3).

Experiments on Skeleton:

To show the applicability of the proposed method on skeleton sequences, we have experimented on the UPCV action dataset [68]. The dataset consists of skeletal sequences corresponding to ten actions performed by twenty individuals. Each action is performed twice by an individual. Each skeleton has twenty joints. Six such action sequences generated by proposed method using source skeleton sequences for the target skeleton are shown in Figure 4.20.

The proposed Guided Deformation Transfer method adjusts for the temporal properties while deformation transfer without changing the shape of the target skeleton. For example, in Figure 4.20, the source action (watch clock sequence) is transferred to target.

Shape Deformation Vs Pose Deformation

We have also experimented on large mesh deformation FAUST dataset [70]. Results are shown in Figure 4.21. In Figure 4.21 (d), we show the results of Deformation Transfer using shape deformation. The shape deformation captures the shape difference between the reference source and the target poses, so it causes undesired artifacts for large pose deformation in the source mesh. With the proposed method, we have been able to address this limitation by employing the pose deformation for the Deformation Transfer. However, the shape deformation is computationally less burdensome than the pose deformation as the shape deformation is computed once from reference source and target meshes.

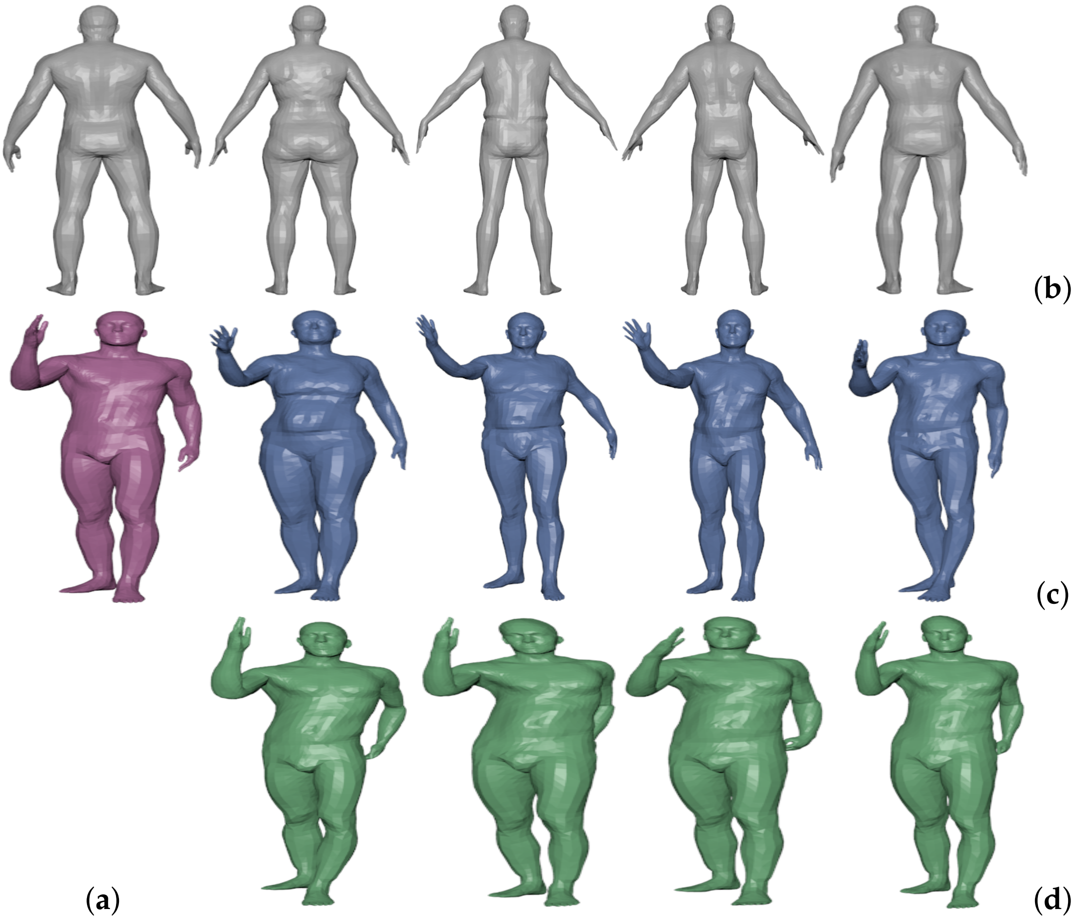


Figure 4.21: Deformation Transfer for large deformation. (**a**) Initial pose of the source(top) and its deformed pose (bottom). (**b**) Initial poses of different targets. Estimated poses of the target by (**c**) Deformation Transfer using pose deformation with *similar boundary condition* and (**d**) Deformation Transfer using shape deformation.

CHAPTER 5

Enveloping

The enveloping process establishes the relationship between deformation of the mesh and deformation of corresponding control structure. The control structure such as the skeleton, the point handles, the cages and the region handles are common in graphics applications. After establishing the relationship, an animator controls the deformation of a mesh via the control structure. Most techniques approach this problem by first identifying control structure and then establishing relation between the control structure and mesh vertex deformation.

The control structure is either constructed manually by an animator or constructed automatically. Assigning the control structure manually requires prior experience and skills. Most of the 3D modelling softwares (Blender, Maya, Autodesk) have a provision to assign the control structure to the given model. Apart from manual assignment, there are several methods proposed in [41, 71] which assign the skeleton to the model mesh automatically.

Due to its simplicity, real-time performance and compatibility with high performance computing hardware, Linear Blend Skinning (LBS) [6] is a widely used enveloping technique. In this, the deformation of each vertex of the mesh is linearly related to the deformation of the control structure. The weights are either manually assigned (based on animator's experience) or are derived from available poses (data driven approach). After computing the weights, the enveloped mesh is deformed by deforming the control structure. In the editing process, each vertex of the enveloped mesh is deformed independent to its neighbours by deforming the skeleton bones. Such individualistic vertex deformation may not preserve the mesh properties such as edge length. This relationship, if not maintained, results in inconsistency artifacts after deformation [6, 7].

In literature some non-linear methods are also proposed which produce high quality deformation of the mesh. For most of these methods, a mesh is represented by its differential representation [13]. The mesh deformation is then computed from deformation of the control structure. In order to ensure smooth and natural deformation of the mesh, the mesh deformation is constrained so as to preserve the properties of inter-vertex relationship such as edge length. Due to added constraint, the mesh editing becomes computationally expensive and sometimes incompatible with high performance computing hardware. Hence, these methods are not suitable for real-time applications such as gaming. To achieve real-time performance, methods based on clustering are suggested where the mesh surface is segmented into patches and each vertex of the mesh undergoes the similar deformation. The patches are generated based on geometrical similarities using clustering techniques such as K-means [8] and mean-shift [19]. The qualitative performance of this approach depends upon the accuracy of quality of the segmentation. Moreover, the segmentation also depends on the number of segments. In essence, all the enveloping methods are trying to deal with trade off between deformation quality and computational performance.

In this work, we show the applicability of the Vector Graph (VG) representation [72] for data-driven enveloping. The VG is a unifying representation that represents various types of meshes such as triangular, quad or hybrid as a collection of vectors. The mesh is reconstructed from its VG representation by solving a system of linear equations. In this work, we have considered the skeleton of the given mesh as its control structure. Both, the skeleton deformation and the mesh deformation groups are smooth manifolds. We establish a map between both the manifolds using a set of deformed poses of the mesh available in the dataset. The image of the map belongs to the vector deformation group. We show that the vector deformation which belongs to the image of the map preserves the properties of inter-vertex relationship without any explicit constraint on the vector deformation. In turn, the proposed method produce smooth and meaningful deformation of the mesh. Moreover, the simplified relationship between the skeleton and the mesh deformations results in the computationally efficient mesh editing process.

Similar to other data-driven enveloping approaches, the proposed enveloping process comprises of two consecutive steps; (i) assigning a skeleton to the mesh and (ii) establishing a map between the mesh deformation and the skeleton deformation. The skeleton is assigned to a reference mesh by the user using an add-on developed in the Blender. The add-on assigns the skeleton to other deformed poses of the mesh automatically and computes the deformations corresponding skeletons. Next, a map is established between the skeleton and vector deformations by estimating the map parameters using example poses of the dataset. The enveloped mesh is then deformed by deforming the skeleton through the established map.

Organization of Chapter: In section 5.1, we describe the proposed approach for control structure (skeleton) assignment to the reference mesh and computation of the bone deformation. We then define the vector and skeleton deformation groups in section 5.2. In section 5.3, we formulate describe proposed enveloping process and establish a map between vector and skeleton deformation groups. After establishing this map, we formulate the mesh deformation associated with skeleton deformation in section 5.4. Here, we show that the relationship between skeleton and mesh deformations using the VG is similar to that of the LBS. Finally, in section 5.5 we compare proposed method with the four state-of-the-art methods in terms of quality of deformation and computational time.

5.1 Skeleton Assignment and Deformation

For the mesh enveloping and editing, the skeleton is used as a control structure for the given mesh. The skeleton consists of the *bones* and the *joints*. First, a user forms a skeleton which has preferred number of bones and joints. Then the user selects a set of four vertices of the reference mesh for each skeleton joint as shown in Figure 5.1. The position of the joint is the mean of four selected vertices. The proposed Blender add-on automatically fits the skeleton to the reference mesh by shifting the skeleton joints to new joints. Next, an orthonormal frame is assigned to each skeleton bone at its one of the bone joints as shown in Figure 5.1. The frame assigned to the j^{th} bone is defined as,

$$H_j = [u_j, c_j, e_j] \tag{5.1}$$

where, the unit vector u_j is parallel to the j^{th} bone, $c_j = (\hat{e}_j \times u_j)/\|\hat{e}_j \times u_j\|$ and $e_j = u_j \times c_j$. The vector $\hat{e}_j$ originates from the joint j and points towards one of the selected vertices for that joint. Note that, the orthonormal frame is formed at only one of the two joints of a skeleton bone. Similarly, orthonormal frames are assigned to all of the m skeleton bones. The add-on then assigns the skeleton to other available deformed meshes.

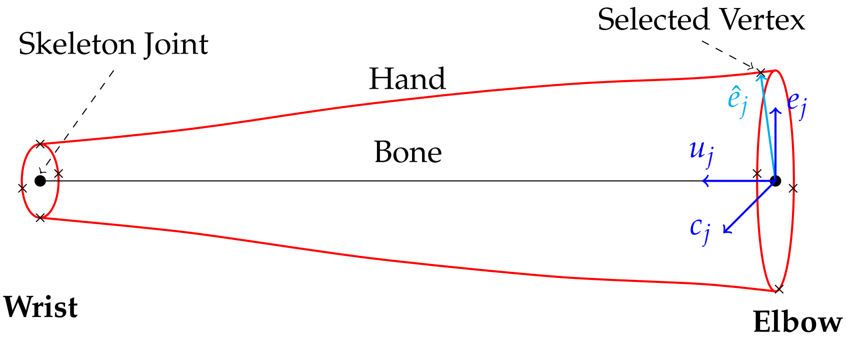


Figure 5.1: Assigning the skeleton and the orthonormal frames to j^{th} bone

5.1.1 Skeleton Deformation

The deformation of a skeleton bone is modelled as the transformation between the corresponding orthonormal frames of the reference and the deformed skeleton. Let's denote the reference and the corresponding deformed frames assigned to j^{th} bone as,

$$H_{rj} = [u_{rj}, c_{rj}, e_{rj}] \text{ and } H_{dj} = [u_{dj}, c_{dj}, e_{dj}]. \tag{5.2}$$

The transformation R_j between the j^{th} reference frame and deformed frames is defined as,

$$R_j H_{rj} = H_{dj} \rightarrow R_j = H_{dj} H_{rj}^{'}. \tag{5.3}$$

Since both the frames are orthonormal and their determinants are $+1$, the transformation R_j is a rotation matrix (see Appendix A.4 for proof). In the next section, we define the relationship between the skeleton deformation and corresponding mesh deformation.

5.2 The Map Between the Skeleton and the Mesh Deformations

In many mesh editing processes, the deformation of a face (triangular) of a given mesh is modelled via some or other linear transformation. The cumulative effect of the localized deformations of the faces then contributes to the desired deformation of the mesh. As noted in [14], the linear transformations capturing the face deformations form a group. The transformations which don't form a group may produce non-physical deformations of the mesh. After enveloping, the deformation of a mesh is being controlled by the corresponding deformation of the skeleton (control structure). Hence, it is expected that both the skeleton and mesh deformations should have the similar group structures. The skeleton deformation which is modelled as the rotations of the bones is defined as tuples $g_s = (R_1, R_2, \ldots, R_m) \in G_s$, where $G_s \in (SO(3))^m$. When using VG representation, the mesh deformation is modelled as a collection of composition of rotation and scaling of each vector. Hence, the mesh deformation using VG also forms a group.

Definition 2. The vector deformation group $g_v \in G_v$), is a tuple,

$$g_v = (a_1 J_1, a_2 J_2, \ldots, a_n J_n) \tag{5.4}$$

where, $a_i \in R^+$ and $J_i \in SO(3)$. The composition map,

$$G_v \times G_v \mapsto G_v, \tag{5.5}$$

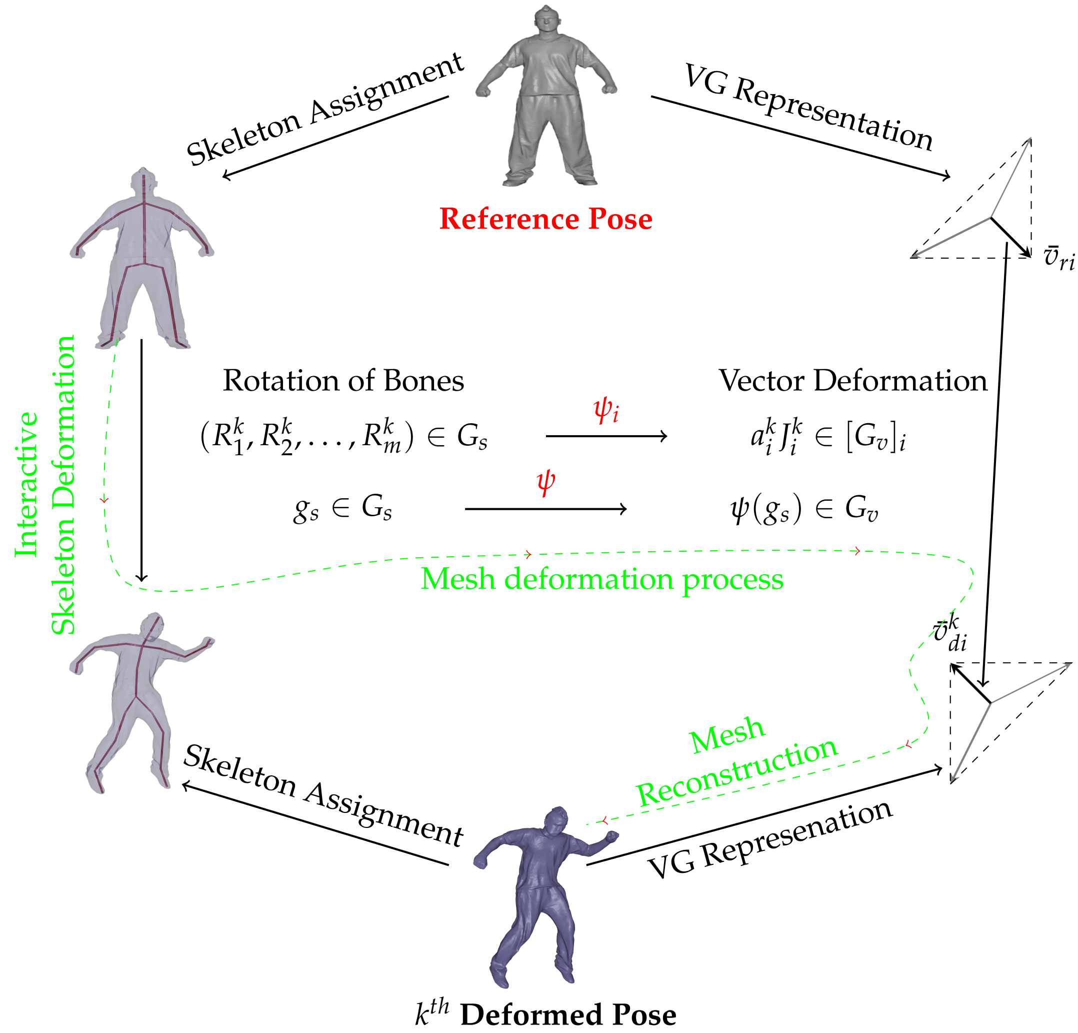


Figure 5.2: Illustration of proposed enveloping process. First, skeletons are assigned to the reference mesh and its deformed poses. The deformation of skeleton is computed as a set of rotations of bones. The meshes are represented by the VG to compute their deformation. The deformation of a vector i is computed as a composition of scaling and rotation. The map, ψ_i, maps skeleton deformation to the deformation of the vector i. The parameters of the map ψ_i are estimated using p available deformed poses and their corresponding skeletons. The dashed curve represent the mesh deformation process after establishing the all the map ψ. The detailed description of the mesh deformation process is shown in Figure 5.3.

$$\begin{aligned}(a_1^1 J_1^1,\, a_2^1 J_2^1,\, \ldots,\, a_n^1 J_n^1) \times (a_1^2 J_1^2,\, a_2^2 J_2^2,\, \ldots,\, a_n^2 J_n^2)\\ \mapsto (a_1^1 a_1^2 J_1^1 J_1^2,\, a_2^1 a_2^2 J_2^1 J_2^2,\, \ldots,\, a_n^1 a_n^2 J_n^1 J_n^2).\end{aligned} \tag{5.6}$$

The group G_v is a sub group of general linear group $GL(3, \mathbb{R})^n$ of invertible matrices which is a Lie group. The map which maps the skeleton deformation to deformation of vector i is defined as,

$$\psi_i : G_s \mapsto [G_v]_i. \tag{5.7}$$

where, the deformation of the vector i belongs to $[G_v]_i$. The collection of all such maps can be defined as,

$$\psi = (\psi_1, \psi_2, \dots, \psi_n) : G_s \mapsto G_v \tag{5.8}$$

where, m and n are the number of skeleton bones and vectors respectively. In proposed data-driven enveloping process, the map ψ is established using a set of available poses of a mesh. Generally, the number of vectors in the VG representation of a mesh depends on resolution of the mesh. The dimension of the manifold G_v is $4n$ but the desired deformation of the mesh belongs to a subset of G_v. A constraint on the deformation will restrict the vector deformation but at the cost of the additional computational time. As shown in Figure 5.3, The map ψ maps the skeleton deformation to the mesh deformation. The skeleton deformation is constrained to be the rigid transformation in the proposed method, so the image of the map belongs to vector deformation group G_v. As a result, the desired and meaningful deformation of the mesh is ensured without the compromising computational performance. In the next section, the construction of the map ψ is explained in detail.

5.3 Enveloping

In our enveloping process, all the maps ψ_i, $\forall i \in \{1, 2, \dots, n\}$ are established independently for all the vectors of the VG. The map ψ_i characterizes the deformation of i^{th} vector deformation in terms of the skeleton deformation. The map ψ_i can be approximated as shown in Figure 5.2. The parameters of the map ψ_i are estimated using the p example poses available in the database.

5.3.1 Bone-Vector Relationship

We define the relationship between the rotation of a vector i and the rotation of the skeleton bone j as,

$$R_j^k W_{ji} = J_i^k,\ k \in \{1, 2, \ldots, p\} \tag{5.9}$$

where J_i^k is the rotation of the vector i as defined in equation (3.13) and R_j^k is the rotation of the bone j as defined in equation (5.3) corresponding to the k^{th} pose. $W_{ji} \in SO(3)$ is the rotational component of the map ψ_i. Since W_{ji} is the same for all available p poses, it can be estimated by solving following,

$$\begin{aligned} &\underset{W_{ji}}{\text{argmin}} && \sum_{k=1}^{p} \|R_j^k W_{ji} - J_i^k\|_{\mathcal{F}} \\ &\text{subject to} && W_{ji}' W_{ji} = I_3,\ det(W_{ji}) = +1 \end{aligned} \tag{5.10}$$

where, $\mathcal{F}$ indicates Frobenius norm and I_3 is the identity matrix. The problem in equation (5.10) can be reformulated as (see Appendix A.5 for derivation),

$$\begin{aligned} &\underset{W_{ji}}{\text{argmax}} && Tr(W_{ji} \mathcal{J}_i' \mathcal{R}_j) \\ &\text{subject to} && W_{ji}' W_{ji} = I_3,\ det(W_{ji}) = +1 \end{aligned} \tag{5.11}$$

where, $\mathcal{R}_j = [R_j'^1,\ R_j'^2,\ \ldots,\ R_j'^p]'$ and $\mathcal{J}_i = [J_i'^1,\ J_i'^2,\ \ldots,\ J_i'^p]'$. As suggested in [73], the solution to above is,

$$W_{ji} = VU' \tag{5.12}$$

where, V and U are orthonormal matrices appearing in the SVD of $\mathcal{J}_i' \mathcal{R}_j = U \Sigma V'$.

5.3.2 Bone Assignment

After approximating W_{ji}'s, their cumulative effect on a vector rotation is characterized by an underlying sparsity assumption. Here, the sparsity indicates that the rotation of a vector depends upon the rotations of only some of the bones. For example, the rotation of a vector associated with the hand region doesn't depend upon the leg bone rotation. Hence, we propose to identify only one such bone with the lowest cost as defined in equation (5.10). Let's denote the index of the identified bone corresponding to vector i as z_i. We say that z_i^{th} bone is actively involved in deforming the vector i. After identifying the active bone, the rotation of the vector i is computed as,

$$J_i^k \approx R_{z_i}^k W_{z_i i}. \tag{5.13}$$

Further using equation (3.13), we get,

$$F_{di}^k = [\hat{v}_{di}^k, b_{di}^k, N_{di}^k] \approx R_{z_i}^k W_{z_i i} F_{ri} = R_{z_i}^k W_{z_i i} [\hat{v}_{ri}, b_{ri}, N_{ri}] \tag{5.14}$$

where, F_{di}^k and F_{ri} are the orthonormal frames associated with the vector i of pose k and the reference pose respectively. These frames are defined as indicated in equation (3.12). Since, we are interested in estimating the deformed vector, removal of the binormal and the normal vectors from above equation leads to following simplified form,

$$\hat{v}_{di}^k \approx R_{z_i}^k W_{z_i i} \hat{v}_{ri}. \tag{5.15}$$

Substituting, $\hat{v}_{di}^k = \bar{v}_{di}^k / \|\bar{v}_{di}^k\|$ and $\hat{v}_{ri} = \bar{v}_{ri} / \|\bar{v}_{ri}\|$ in above equation result in,

$$\bar{v}_{di}^k \approx a_i^k R_{z_i}^k W_{z_i i} \bar{v}_{ri} \tag{5.16}$$

where, $a_i^k = \|\bar{v}_{di}^k\| / \|\bar{v}_{ri}\|$ is the scaling factor as defined in the equation (3.13). The scaling component of the vector deformation should be established here to map the bone deformation to the vector deformation. Let $g_{z_i i}$ be the scaling parameter of the map ψ_i. We rewrite the equation (5.17) after including $g_{z_i i}$ as,

$$\bar{v}_{di}^k \approx g_{z_i i} R_{z_i}^k W_{z_i i} \bar{v}_{ri},\ g_{z_i i} \in R^+. \tag{5.17}$$

The parameter $g_{z_i i}$ can be approximated as,

$$g_{z_i i} = \omega^1 a_i^1 + \omega^2 a_i^2 + \cdots + \omega^p a_i^p \tag{5.18}$$

where, weights ω^ks are computed from the cosine similarity as,

$$\omega^k = \frac{\langle \bar{v}_{di}^k, R_{z_i}^k W_{z_i i} \bar{v}_{ri} \rangle}{\sum_{r=1}^{p} \langle \bar{v}_{di}^r, R_{z_i}^r W_{z_i i} \bar{v}_{ri} \rangle}. \tag{5.19}$$

The above process estimates the parameters ($W_{z_i i}$ and $g_{z_i i}$) of the map ψ_i using the p example poses.

We have also experimented by taking appropriate convex combination of deformation of two skeleton bones to approximate a vector deformation. However, the approximated vector deformation may not belong to the vector deformation group G_v. As a result, approximating the vector deformation using one bone leads to better mesh deformation qualitatively compared to the use of two bones as shown in Table 5.1 (column 5 and 6 of error section) and Figure 5.5(f)-(g). It is important to note that the computational time remains almost the same for both the cases.

5.3.3 The Injective Map

The map ψ_i maps the rotation of bone z_i to corresponding deformation of vector i. When its domain is restricted to the rotation of the bone z_i, the restricted map is defined as,

$$\psi_i|_{R_{z_i}} : R_{z_i} \mapsto a_i J_i \tag{5.20}$$

where the scaling a_i is the scaling parameter of the map; $g_{z_i i}$. The rotation of the vector i corresponding to bone z_i is,

$$J_i = R_{z_i} W_{z_i i} \rightarrow R_{z_i} = J_i W'_{z_i i}. \tag{5.21}$$

To show that the map $\psi_i|_{R_{z_i}}$ is injective, let's consider $R_{z_i}^1$ and $R_{z_i}^2$ to be the two different rotations of the bone z_i. Assuming that if the map $\psi_i|_{R_{z_i}}$ is not injective then resulting vector rotations due to rotations $R_{z_i}^1$ and $R_{z_i}^2$ are same,

$$J_i^1 = J_i^2 \rightarrow R_{z_i}^1 W_{z_i i} = R_{z_i}^2 W_{z_i i}. \tag{5.22}$$

Since $W_{z_i i}$ is a rotation matrix, $R^1_{z_i} = R^2_{z_i}$. This proves by contradiction that the map $\psi_i|_{R_{z_i}}$ is an injective map. Moreover, this map is also continuous. Similarly, the map $\psi|_R = (\psi_1|_{R_{z_1}}, \psi_2|_{R_{z_2}}, \ldots, \psi_n|_{R_{z_n}})$ is also an injective map. However, the map $\psi_i|_{R_{z_i}}$ is not a homomorphism because,

$$\psi_i|_{R_{z_i}}(R^1_{z_i} R^2_{z_i}) = g_{z_i i} R^1_{z_i} R^2_{z_i} W_{z_i i} \neq g_{z_i i} R^1_{z_i} W_{z_i i} g_{z_i i} R^2_{z_i} W_{z_i i} \tag{5.23}$$

$$= \psi_i|_{R_{z_i}}(R^1_{z_i}) \psi_i|_{R_{z_i}}(R^2_{z_i}). \tag{5.24}$$

5.4 Mesh and Skeleton Relationship

Once the parameters of the map ψ are estimated, it is straight forward to edit the enveloped mesh by editing the associated skeleton. The mesh editing process is illustrated in Figure 5.3. The deformation of vector i is defined in (5.17) as,

$$\bar{v}_{di} = g_{z_i i} R_{z_i} W_{z_i i} \bar{v}_{ri} \tag{5.25}$$

where $g_{z_i i} \in \mathbb{R}^+$ and $R_{z_i} W_{z_i i} \in SO(3)$. By substituting the bone rotation defined in (5.3) into (5.25), we get,

$$\bar{v}_{di} = g_{z_i i} H_{dz_i} H'_{rz_i} W_{z_i i} \bar{v}_{ri} \tag{5.26}$$

where, H_{dz_i} and H_{rz_i} are the orthonormal frames associated with the reference and the deformed skeleton bone z_i. Similarly, the other deformed vectors, v_{di}, are determined from the deformation of the respective skeleton bones z_is. The deformed VG representation of the mesh is obtained as,

$$\bar{V}_d = [\bar{v}_{d1}, \bar{v}_{d2}, \ldots, \bar{v}_{dn}] = [H_{d1}, H_{d2}, \ldots, H_{dm}]W = H_d W \tag{5.27}$$

where, W is a sparse matrix since the deformation of the vector i depends on the deformation of the z_i^{th} bone only. The construction of matrix W is explained in Appendix A.6.

The deformed mesh is reconstructed from its VG representation as describe in

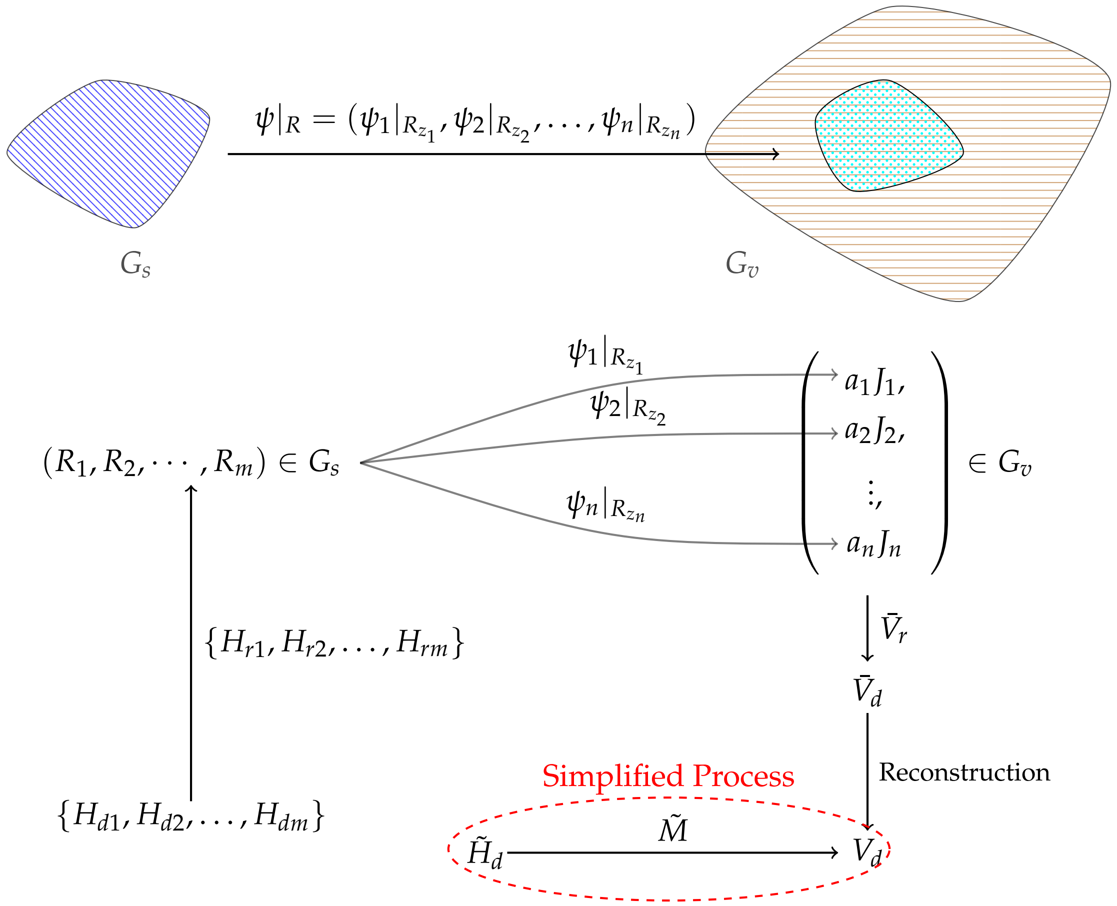


Figure 5.3: Illustration of mesh editing process. First, the orthogonal frames $H_d = [H_{d1}, H_{d2}, \ldots, H_{dm}]$ are obtained from skeleton deformed by user in order to compute bone rotations. The deformed VG, $\bar{V}_d$, is computed using established map $\psi|_R = \{\psi_1|_{R_{z_1}}, \psi_2|_{R_{z_2}}, \ldots, \psi_n|_{R_{z_n}}\}$ from skeleton deformation. The image of the map ψ belongs to the vector deformation manifold (G_v). The reconstruction process reconstruct the deformed mesh V_d from its VG representation. The entire process is simplified to get the deformed mesh V_d from $\tilde{H}_d = [H_{d1}, t_{d1}, H_{d2}, t_{d2}, \ldots, H_{dm}, t_{dm}]$ for computationally efficient mesh editing.

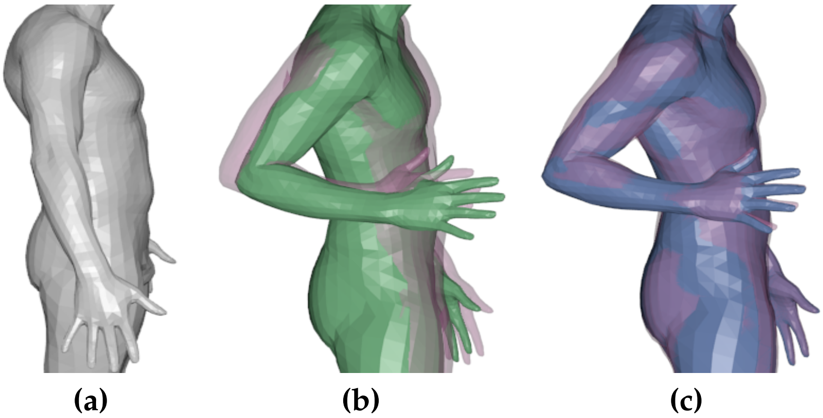


Figure 5.4: Demonstration of importance of positional constraints. The pose in pink color (transperent) is the ground truth pose. **(a)** Reference Pose. **(b)** Generated pose (in green) by proposed method with one positional constraint. **(c)** Generated pose (in blue) by proposed method by imposing positional constraint on few vertices. The deformation of constrained vertices are estimated using LBS [6].

section 3.1 using (3.9),

$$\hat{V}_d = \bar{V}_d \hat{A}^+ - V_L \hat{A}_L^+ = H_d W \hat{A}^+ - V_L \hat{A}_L^+ = H_d K - V_L \hat{A}_L^+. \tag{5.28}$$

where, matrices K and $\hat{A}_L^+$ are constructed and stored off-line. The matrix H_d consists of orthonormal frames (3×3) of m deformed skeleton bones. The matrix V_L consists of constraint vertices which are predicted by LBS ($V_L = \tilde{H}_d \Phi$) where, the matrix Φ is computed for the constrained vertices as described in [6]. The equation (5.28) can now be written as,

$$\hat{V}_d = H_d K - \tilde{H}_d \Phi \hat{A}_L^+ \tag{5.29}$$

where, $\tilde{H}_d = [H_{d1}, t_{d1}, H_{d2}, t_{d2}, \ldots, H_{dm}, t_{dm}]$ contains the orthonormal frames and the translations t_{di}, $i \in \{1, 2, \ldots, m\}$ associated with the m deformed bones and is computed as explained in [6]. Equation (5.29) can be rewritten in term of $\tilde{H}_d$ as,

$$\hat{V}_d = \tilde{H}_d \tilde{K} - \tilde{H}_d \Phi \hat{A}_L^+ = \tilde{H}_d (\tilde{K} - \Phi \hat{A}_L^+) = \tilde{H}_d M. \tag{5.30}$$

The matrix $\tilde{K}$ is define as,

$$\tilde{K} = [K_1', K_2', K_3', \mathbf{0}, K_4', K_5', K_6', \mathbf{0}, \ldots, K_{3m-2}', K_{3m-1}', K_{3m}', \mathbf{0}]' \tag{5.31}$$

where, K_i is the i^{th} row of the matrix K. The vector $\mathbf{0}$ (column) ensures that $\tilde{H}_d\tilde{K} = H_dK$. The vertices of the deformed mesh are obtained by appending V_L to $\hat{V}_d$,

$$V_d = [\hat{V}_d, V_L] = [\tilde{H}_dM, \tilde{H}_d\Phi] = \tilde{H}_d\tilde{M}. \tag{5.32}$$

Above relationship simplifies the entire mesh editing process as illustrated in Figure 5.3. Moreover, this equation is similar to LBS [6]. The computational complexity of proposed method is $O(mq)$ where, m and q are number of bones and vertices respectively. Table 5.2 shows a comparison of the proposed method with the four state of the art methods in terms of computation.

The selection of constraint vertices affect the quality of the estimated deformed mesh. In the process of vector deformation, the vector translation isn't involved explicitly but rather it is approximated during mesh reconstruction process. This may result in the reconstructed deformed mesh which may not be aligned with the ground truth pose as shown in Figure 5.4(b). In Figure 5.4(b), we have selected only one vertex as the constraint vertex. The LBS [6] involves the translation in the computation of vertex deformation. This ensures that the vertices which are attached firmly with bones are predicted accurately. Selecting those vertices as fixed positional constraint during mesh reconstruction process, we are able to accurately envelope the mesh as shown in Figure 5.4(c). We have selected 0.5% to 3.9% of the total mesh vertices as constraint vertices during experiments.

5.5 Results and Experiments

In this section, we present the results of the experiments conducted. The experiments include the process of enveloping followed by the mesh deformation. We have developed a Blender 2.82 add-on in python 3.7 for interactive user experience. All the experiments have been performed on intel-i7-2.8GHz, 8 core system with 8 GB memory. All the state-of-the-art methods have been implemented as *single threaded* python programs excluding DeepLBS [7]. The DeepLBS [7], has been implemented using CPU version of tensorflow [74].

We have compared our enveloping method with the widely used LBS [6], its recent variant DeepLBS [7], LBS+ARAP [8] and Rotational Regression (RR) [9] in terms of the quality and the computation time. We have enveloped 16 different models of various data sets: 10 (500xx) from [4], 3 (Horse, Camel and Flamingo) from [1], 1 (Man) from [5], 1 (Hand) from [75] and 1 (Kid) from [76] using the proposed method, for details refer to the Table 5.1 and Table 5.2.

Model	Verts	Train. Poses	Bones	Error											
				LBS		DeepLBS		LBS+ARAP		RR		Proposed(1)		Proposed(2)	
				μ	σ	μ	σ	μ	σ	μ	σ	μ	σ	μ	σ
50002	6890	480	33	4.65	15.85	4.64	15.81	4.51	14.95	4.34	14.99	4.35	14.99	4.80	14.99
50004	6890	480	33	4.98	20.87	4.98	20.85	4.99	20.87	4.71	17.58	4.41	18.07	4.85	18.77
50007	6890	457	33	4.41	21.69	3.36	21.56	4.19	15.69	4.09	14.97	3.93	14.97	4.34	14.97
50009	6890	312	33	3.92	14.07	3.97	14.04	3.83	10.74	3.83	10.62	3.74	8.14	4.13	10.28
50020	6890	325	33	3.63	14.39	3.60	14.36	3.64	13.31	3.49	9.22	3.41	8.85	3.89	9.47
50021	6890	382	33	4.04	18.21	4.07	18.17	4.01	13.50	3.92	11.27	3.81	11.13	4.22	12.57
50022	6890	499	33	4.63	23.77	4.63	23.64	6.43	21.76	4.48	15.27	4.37	15.27	4.94	15.27
50025	6890	515	33	3.96	21.86	3.98	21.86	4.06	16.60	3.93	12.17	3.77	12.17	4.17	12.17
50026	6890	488	33	5.54	20.77	5.45	20.72	5.06	10.98	5.04	10.84	4.82	10.66	5.30	10.66
50027	6890	409	33	3.86	16.48	3.62	12.26	3.97	12.29	3.78	12.08	3.72	12.08	4.18	12.08
Horse	8431	11	40	3.74	15.77	3.39	15.51	5.46	16.35	4.82	18.51	4.03	10.90	5.30	14.14
Camel	21887	11	29	6.57	36.50	6.57	81.26	7.14	24.43	8.10	19.60	6.69	18.78	8.09	18.78
Flamingo	26394	11	31	4.40	11.49	4.41	70.27	4.40	12.27	3.94	10.72	3.77	12.15	4.01	10.72
Man	10002	175	20	22.51	27.68	22.38	27.79	22.50	25.17	22.65	22.39	19.42	21.27	21.86	21.72
Hand	7997	44	20	2.27	12.33	2.26	12.31	2.67	7.56	4.45	12.72	2.29	8.85	2.72	8.85
Kid	59727	16	35	5.84	17.90	5.48	16.94	6.58	14.10	5.88	13.91	5.61	14.12	6.12	14.32

Table 5.1: Quantitative comparison of various methods in terms of RSS error. The value in bracket for proposed method indicates number of bones assigned to approximate vector deformation. μ and σ represents mean and standard deviation of the enveloping error respectively.

5.5.1 Quantitative comparison

In Table 5.1, we have compare proposed method with four state-of-the-art methods. We have first computed the Root of Sum of Squared (RSS) error for each vertex. The RSS error for the vertex i of pose k of an enveloped mesh (say E_i^k) is computed as,

$$E_i^k = \sqrt{\sum_{z=1}^{3} (v_{iz}^k - \tilde{v}_{iz}^k)^2} \tag{5.33}$$

where $v_i^k = [v_{i1}^k, v_{i2}^k, v_{i3}^k]$ and $\tilde{v}_i^k = [\tilde{v}_{i1}^k, \tilde{v}_{i2}^k, \tilde{v}_{i3}^k]$ are i^{th} vertices of k^{th} regenerated and ground truth poses respectively. Two statistical measures; mean and standard deviation have been computed to quantify the deformation quality as,

$$\mu = \frac{1}{q}\sum_{i=1}^{q}\frac{1}{p}\sum_{k=1}^{p} E_i^k \tag{5.34}$$

$$\sigma_i = \sqrt{\frac{1}{p}\sum_{k=1}^{p}(E_i^k - \mu_i)^2} \text{ where, } \mu_i = \frac{1}{p}\sum_{k=1}^{p} E_i^k \tag{5.35}$$

where, q and p are the number of vertices and poses of the enveloped mesh respectively. σ_i is the standard deviation of error computed for the vertex i. We compute the maximum standard deviation as,

$$\sigma = max\{\sigma_1, \sigma_2, \ldots, \sigma_q\}. \tag{5.36}$$

σ represents the standard deviation of the vertex where there is a maximum variation in the error.

In data driven enveloping approaches, the map which relates the skeleton and the mesh deformations, is approximated by solving either a linear or a non-linear optimization problem using available poses in the database. The over-fitting is one of the common issues observed for such methods. During optimization, the deformation of a large number of vertices are estimated accurately balancing out the badly estimated deformations for a few vertices. This inconsistency in the estimation leads to an erratic distribution of enveloping error which causes inconsistency artifacts in deformed mesh specially at the joints. The standard deviation defined in equation (5.35) is an appropriate measure of inconsistency. The mean quantifies the overall enveloping error of an enveloped mesh.

In Table 5.1, both the measures μ and σ are shown for various meshes enveloped by LBS [6], DeepLBS [7], ARAP+LBS [8], RR [9] and proposed method. In LBS [6], a vertex deformation is independently associated with deformation of skeleton bones ignoring inter-vertex relationship. Such an individualistic association yields over-fitting during the enveloping process and results in the inconsistency artifacts. Relatively higher μ and σ for results obtained by the LBS endorse the same inference. A slight decrease in μ and σ in DeepLBS [7] evinces the moderate refinement in inconsistency artifacts.

Model	Verts	Train. Poses	Bones	Computation Time									
				LBS		DeepLBS		LBS+ARAP		RR		Proposed	
				Env (s)	Edt (ms)	Env (s)	Edt (ms)	Env (s)	Edt (ms)	Env (s)	Edt (ms)	Env (s)	Edt (ms)
50002	6890	480	33	253.29	7.69	538.87	2262.3	315.42	313.8	114.09	22.23	552.10	6.92
50004	6890	480	33	240.02	7.43	658.04	2337.8	274.47	339.3	104.17	26.12	524.68	6.94
50007	6890	457	33	225.81	7.15	605.97	2284.5	269.82	315.9	103.10	25.15	517.67	6.82
50009	6890	312	33	152.75	7.54	457.63	1589.3	207.74	325.5	71.43	22.56	365.81	6.87
50020	6890	325	33	166.01	7.06	469.75	2269.3	219.96	314.3	71.52	24.89	371.13	6.91
50021	6890	382	33	203.93	7.49	583.91	2305.2	257.56	312.97	90.21	22.76	429.06	7.24
50022	6890	499	33	241.96	7.24	633.08	2208.4	296.58	325.6	112.80	24.34	528.76	6.99
50025	6890	515	33	256.80	6.97	381.35	2304.3	308.75	325.3	122.09	24.82	557.22	7.13
50026	6890	488	33	233.11	7.33	460.91	2258.7	283.03	327.1	110.60	22.54	544.74	6.90
50027	6890	409	33	202.32	7.28	418.90	2287.8	257.41	318.23	92.19	22.31	470.38	7.26
Horse	8431	11	40	12.10	9.07	251.09	2799.9	157.48	430.32	58.09	29.30	89.70	8.61
Camel	21887	11	29	19.23	8.15	242.01	1720.6	106.43	1067.5	116.61	30.64	156.14	8.20
Flamingo	26394	11	31	25.65	9.64	243.96	2160.1	131.89	1376.6	145.59	48.07	208.59	9.01
Man	10002	175	20	78.67	5.21	257.08	1178.5	653.86	533.1	581.46	16.91	296.80	5.10
Hand	7997	44	20	15.87	5.95	97.21	1186.1	114.91	381.4	111.20	17.50	88.88	4.72
Kid	59727	16	35	83.36	15.06	540.99	2617.0	446.18	3386.3	535.13	86.37	594.89	14.95

Table 5.2: Quantitative comparison of various methods in terms of computation time. *s* and *ms* indicate time duration in seconds and milliseconds respectively. *Env* and *Edt* stand for enveloping and editing respectively.

In ARAP+LBS [8] and RR [9], the mesh deformation is computed by representing mesh vertices or faces with a differential representation. In RR, the face deformation is approximated by skeleton bone deformation based on established non-linear relationship whereas it is constrained to be As-Rigid-As-possible in ARAP+LBS. Similar to ARAP and RR, the VG represents faces using vector graph as discussed in the chapter 3. The image of the map belongs to the vector deformation manifold (G_v). Hence, in these methods, smooth deformation is achieved and inconsistency artifacts are prevented during enveloped mesh editing. As shown in Table 5.1, the minuscule change is observed in μ for these methods compared to LBS and DeepLBS, although, the significant reduction in σ supports our claim.

To understand the effect of number of skeleton bones to be used to compute the vector deformation in the proposed method, we have experimented by approximating the vector deformation using convex combination of deformations of two are more skeleton bones. Since the convex combination of rotation matrices is not a rotation matrix, the approximated vector deformation doesn't belong to vector deformation group G_v. As shown in Table 5.1 (columns 5 and 6 in er-

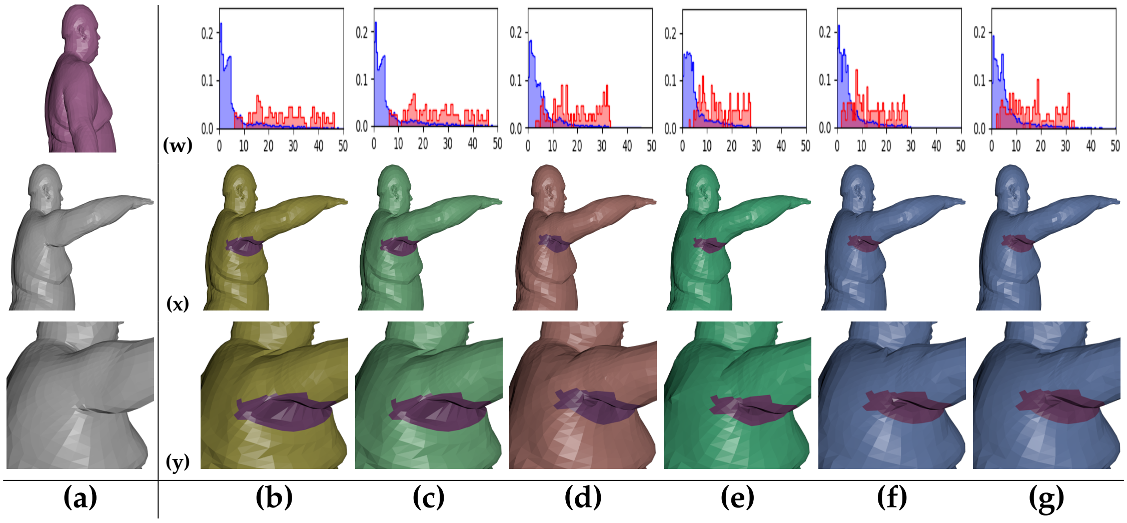


Figure 5.5: **(a)** Reference pose, ground truth and zoomed in region of the ground truth (top to bottom). The experimental results for **(b)** LBS [6] **(c)** DeepLBS [7] **(d)** LBS+ARAP[8] **(e)** RR [9], **(f)** proposed method with 1 bone/vector **(g)** proposed method with 2 bones/vector. **(w)** Normalized histogram of the error of all vertices (blue) and of vertices in highlighted region (red). (**x**) The deformed poses and (**y**) corresponding zoomed in region of interest.

ror), the significant increment in μ and negligible change in σ are observed for two bone assignment compared to one. The increment is μ shows that the error increases for two bone assignment compared to one bone assignment. The same can be observed in Figure 5.5(f) and (g). Hence, the quantitative analysis suggests the effectiveness of one bone assignment for the proposed method.

The application like gaming not only demands quality deformation but also real-time performance while editing enveloped mesh. In Table 5.2, the comparison have been conducted based on enveloping and editing time. The enveloping time is the time required to establish the mesh-skeleton deformation relationship whereas the editing time represents time the required to deform the enveloped mesh. The mesh editing should be in real-time for an enveloping method. The LBS [6] is computationally the most efficient. This makes it suitable for many applications despite its limitations. The LBS+ARAP [8] and RR [9] ensure the smooth and realistic mesh deformation during editing but both suffer from computational overload due to the presence of non-linearity. For proposed method, we have simplified the relationship between skeleton and mesh deformations. the proposed method turn out to be similar to the LBS [6] (see equation (5.28)) in terms of the computational time. Moreover, it also ensures the smooth and realistic deformation comparable with nonlinear methods such as ARAP+LBS [8] and RR [9].

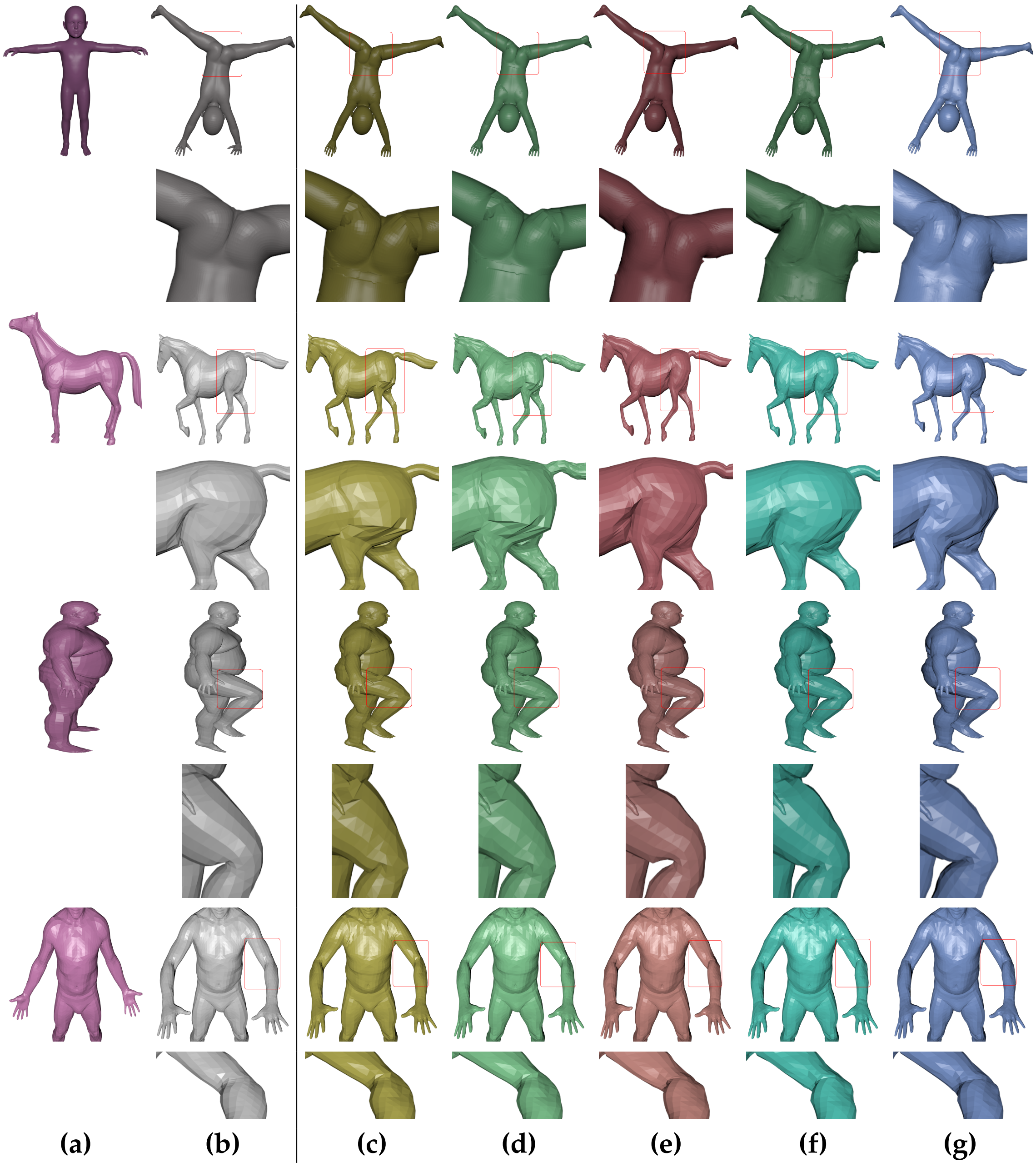


Figure 5.6: Qualitative evaluation on training data. **(a)** Reference poses of enveloped meshes. **(b)** Ground truth poses available in corresponding datasets. Estimated poses by **(c)** LBS [6] **(d)** DeepLBS [7] **(e)** LBS+ARAP[8] **(f)** RR [9] and **(g)** proposed method. Magnified portion of the poses are shown below the full pose.

5.5.2 Qualitative comparison

For qualitative assessment of the deformed mesh, all the training poses are regenerated using the proposed enveloping method. The distribution of error of the highlighted region are shown as the normalized histogram in Figure 5.5(w). The normalized histogram in red represents the error distribution in a set of vertices (in purple) near armpit. The enveloping error is computed as defined in (5.33) for each vertex using respective ground truth pose available in the data set [4]. The higher standard deviation of error (histogram in blue) corresponding to LBS [6] and DeepLBS [7] as shown in Figure 5.5(b) and (c) are indicative of the inconsistency artifact due to over-fitting. The large number of vertices are estimated perfectly at the cost of poorly estimated fewer vertices. The inconsistency artifacts around armpit regions (highlighted in purple) of meshes are shown in Figure 5.5(b) and (c). Comparatively compact error distribution in histogram corresponding to ARAP+LBS [8], RR [9] and proposed method. We observe that the standard deviation of error in the set of vertices near armpit is notably smaller compared to those of LBS and DeepLBS. The histogram representing the error of highlighted segment (red) shown in Figure 5.5(w) corresponding to ARAP+LBS, RR and proposed method is closer to zero compared to those of LBS and DeepLBS and hence endorses our inference.

In Figure 5.6 and Figure 5.7, we show a qualitative comparison between various enveloping methods. The qualitative assessment has been carried out by comparing regenerated training meshes with corresponding ground truth mesh in Figure 5.6. The Figure 5.7 shows comparison based on smoothness and naturalness of deformation in edited enveloped meshes. As shown in Figure 5.6(c) and Figure 5.7(b), the inconsistency artifacts are observed near joints of meshes enveloped by LBS [6]. DeepLBS [7] improvises on the LBS and reduces the error near the joint regions but it fails to correct large errors as shown in Figure 5.6(d) and 5.7(c). In ARAP+LBS [8], the inconsistency artifacts are overcome by imposing ARAP [36] constraint to LBS [6]. This results in smooth deformations as shown in Figure 5.6(e) and 5.7(d). The non-linear method RR [9] also ensures the smooth and natural deformation of enveloped meshes as shown in Figure 5.6(f) and Fig-

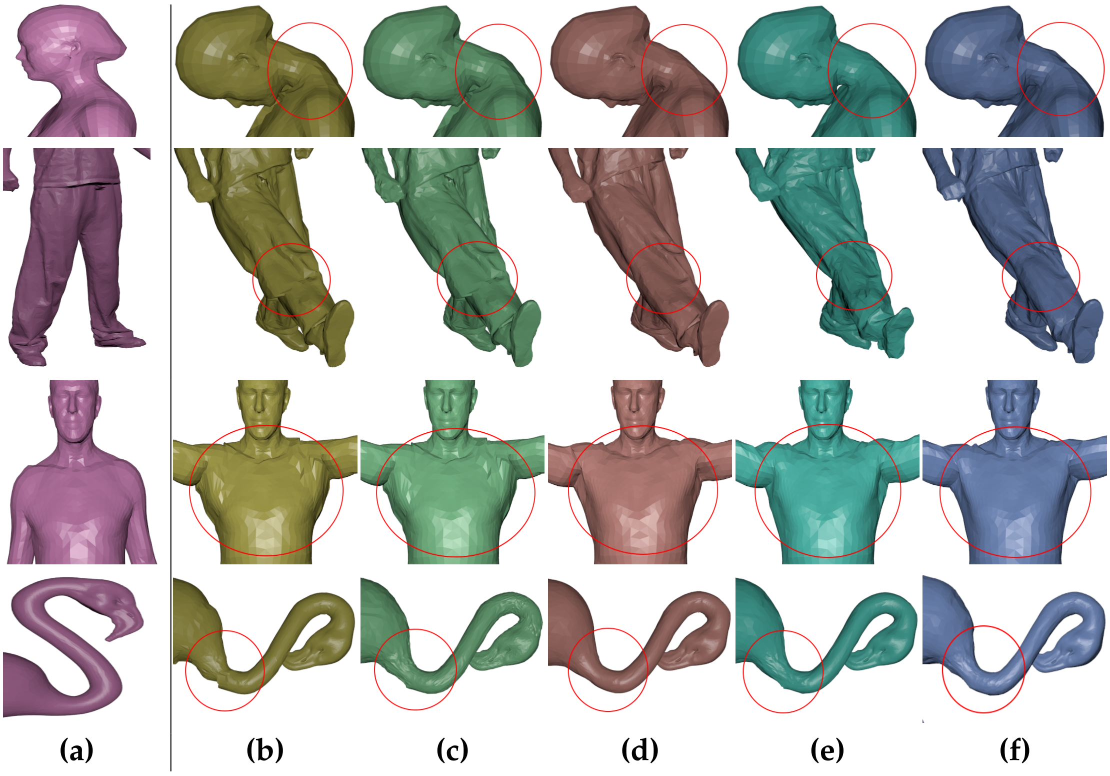


Figure 5.7: Qualitative evaluation on edited meshes. **(a)** Reference poses of enveloped meshes. Edited poses by **(b)** LBS [6] **(c)** DeepLBS [7] **(d)** LBS+ARAP[8] **(e)** RR [9] and **(f)** proposed method.

ure 5.7(e). The proposed method is qualitatively comparable with LBS+ARAP [8] and RR [9] as shown in Figure 5.6(g) and Figure 5.7(f).

CHAPTER 6

Patch Based Interpolation and Morphing

An animation sequence is a set of temporally smooth consecutive poses of a mesh (an animation character). Creating such a set of poses is a challenging task. Typically, an animator first creates a small set of poses (key-frames) by deforming the reference mesh. The intermediate poses between two key-frames are generated next by one of the shape interpolation methods. The shape interpolation method should yield realistic and smoothly deformed interpolated poses. At the same time, the method should be computationally efficient. It is in same cases redundant to generate similar animation sequence for some other mesh (character) by following the same process. This tedious task can be performed efficiently by transferring vital information (deformation) of the available animation of a character (source) to the other character (target). This process is known as Deformation Transfer (DT) in the literature. The deformation transfer process preserves the target's geometric details. We have discussed about challenges and the proposed framework for Deformation Transfer in chapter 4.

In the shape interpolation, the meshes are embedded into a higher dimensional shape space. The trajectory approximation between a pair of points is then carried out in the shape space. A point on the trajectory is an interpolated pose. In this approach, the quality of the deformation highly depends upon the trajectory approximation process and the embedding . A linear interpolation scheme is the simplest and fastest in which the trajectory between two points in the shape space is a line segment. However, it suffers from a shrinkage problem [10]. Such a problem can be addressed by introducing additional constraints into the trajectory approximation process. However, the added constraints make the trajectory approximation process computationally inefficient. Moreover, this approach can be directly be adopted for multi-pose interpolation and shape morphing with small modifications.

In another approach, the shape interpolation is defined as the interpolation of the deformations of the mesh. In this case, a mesh is first represented by the differential representation to compute the element (vertex or face) deformation. The deformation based approach can be easily adopted for the multi-pose interpolation and the morphing. It generates realistic and smooth deformations between the interpolated poses.

We explore the interpolation and the morphing by adopting the traditional approach using VG and learning base approach. The traditional approach is computationally inefficient due to element-wise operations compared to learning based approach. In learning based approach, the mesh is first segmented into a set of patches using the K-means clustering. The mesh segmented into patches serves as a low resolution structure. We then establish a map between the patches of segmented mesh and vectors of VG representation of mesh. We also show that the established map for a mesh can be used for interpolation and deformation transfer. Both interpolation and deformation transfer can also be performed on the morphed mesh by modifying the established map. We compare both the approaches with other methods qualitatively and quantitatively.

6.1 The Deformation Based Interpolation and Morphing Using the VG

In this section, we describe the deformation based interpolation and the morphing tasks using the VG representation. We present the comparison of the VG representation with the other differential representations for both the applications as show in Figure 6.1 and 6.2. We have interpolated deformation of vectors of VG representation. The deformation is computed between the vector of reference mesh and the VG vectors of the deformed mesh in terms of rotation and scaling as explained in section 3.3. For interpolation, we assume that both the meshes have

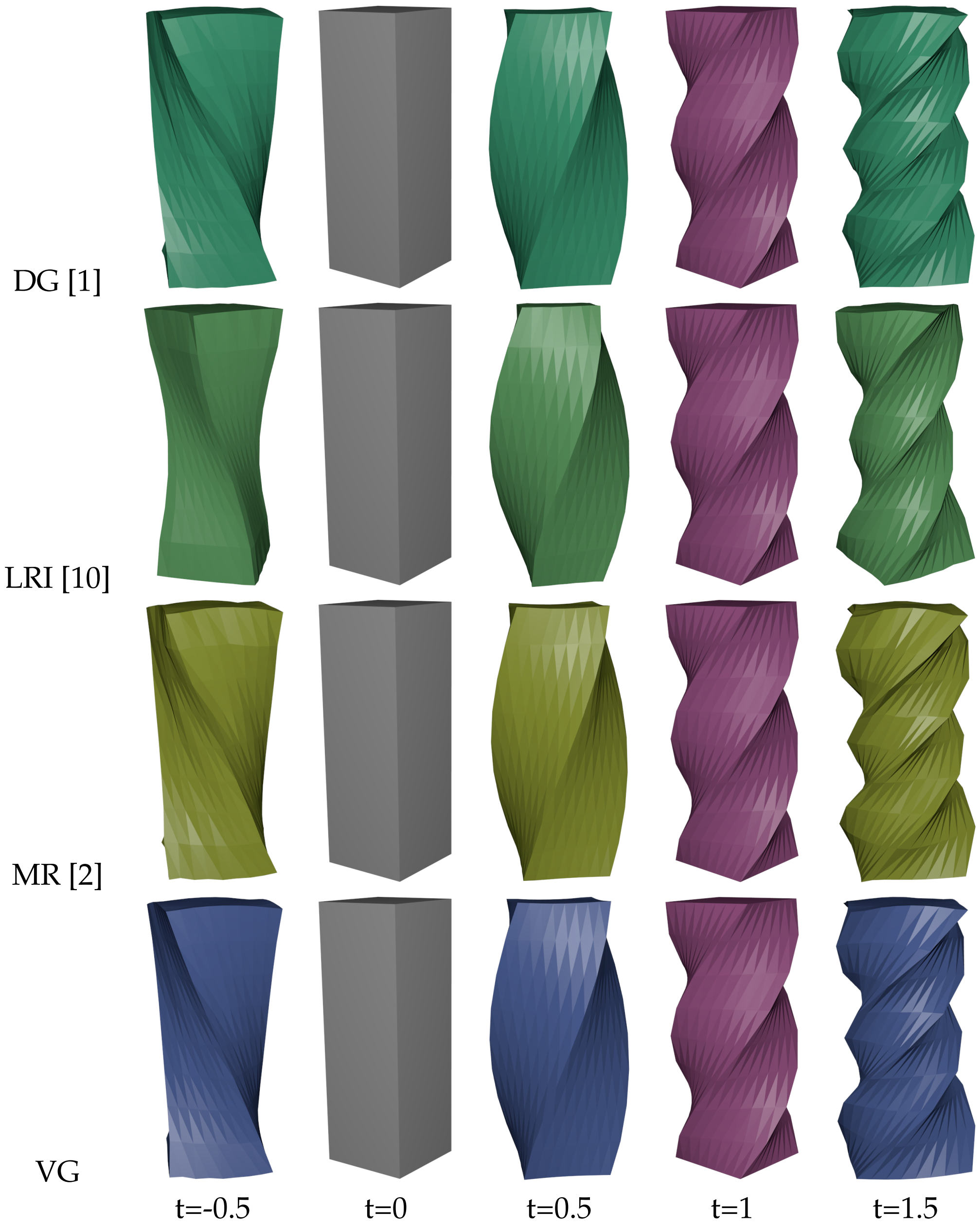


Figure 6.1: Demonstration of interpolation. We compare VG with three different representation namely Deformation Gradient (DG) [1], Linear Rotation Invariant coordinates (LRI) [10] and Manifold Representation (MR) [2]. $t = -0.5$ and $t = 1.5$ represent the extrapolation.

the same resolution and the face wise correspondence is available whereas both meshes may have different resolution for morphing. The correspondence, if not available, can be established by following the registration process as suggested in [1]. As explained in section 3.4.2, the deformation between the reference mesh and deformed mesh is a points on the manifold $\mathcal{M}$. Our aim is to find a suitable path

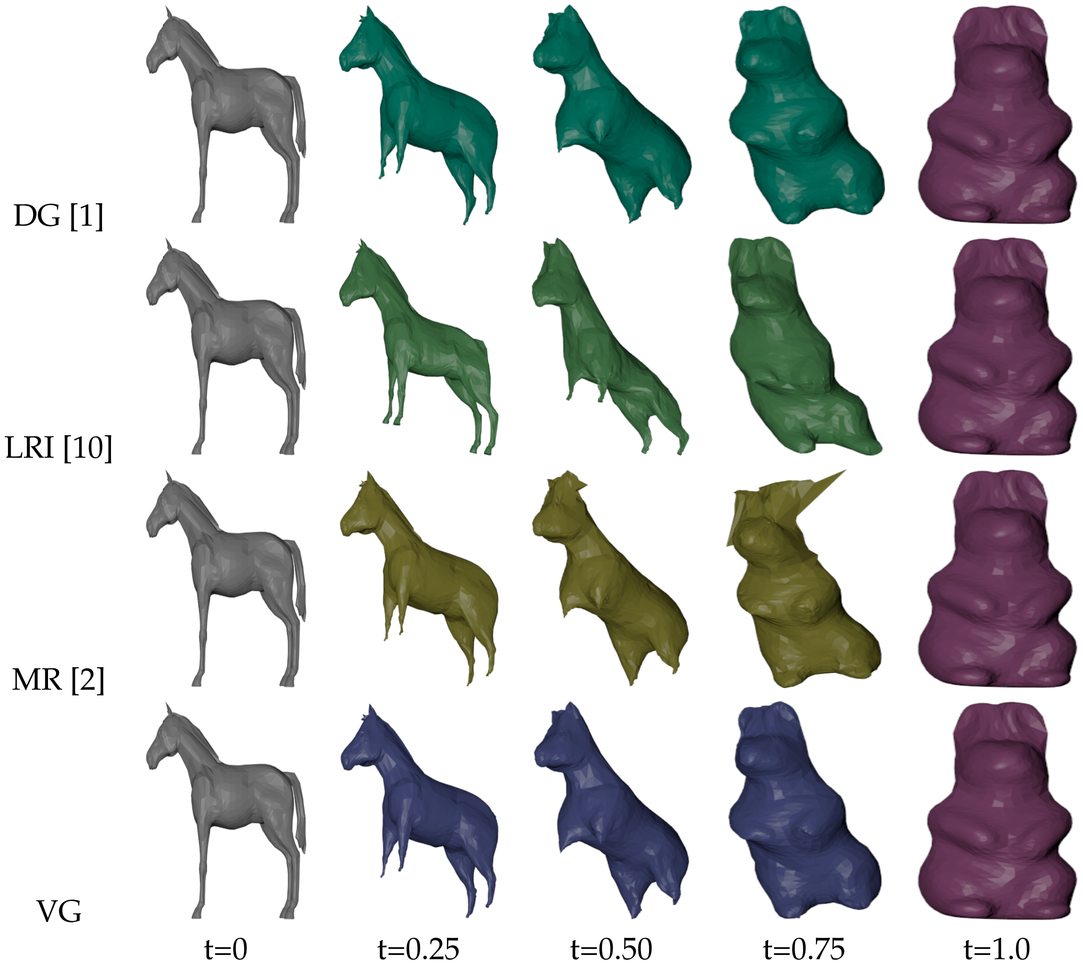


Figure 6.2: Demonstration of morphing. We compare VG with three different representation namely Deformation Gradient (DG) [1], Linear Rotation Invariant coordinates (LRI) [10] and Manifold Representation (MR) [2]. $t = -0.5$ and $t = 1.5$ represent the extrapolation.

between two points on the manifold $\mathcal{M}$. Let's consider rotation J_i and scaling a_i that deform a reference vector i to its corresponding deformed vector. To generate path on the manifold, we interpolate the rotational part from unit rotation I to J_i and scaling part from 1 to a_i as,

$$J_i^t = exp(t * log(J_i)) \text{ and } a_i^t = exp(t * log(a_i)),\ t \in [0, 1]. \tag{6.1}$$

The intermediate position of vector i can be computed as,

$$\bar{v}_{di}^t = a_i^t J_i^t \bar{v}_{ri}. \tag{6.2}$$

The intermediate vectors of VG representation are computed similarly. The VG representation of intermediate pose is,

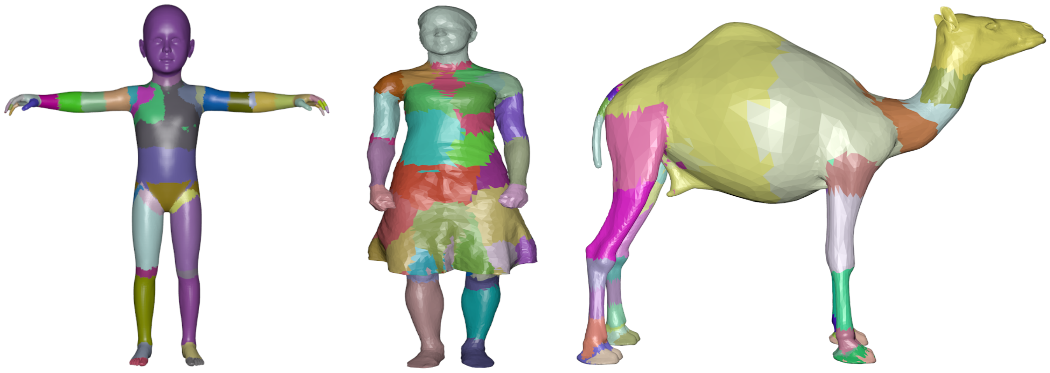
$$\bar{V}_d^t = [\bar{v}_{d1}^t, \bar{v}_{d2}^t, \ldots, \bar{v}_{dn}^t] = [a_1^t J_1^t \bar{v}_{r1}, a_2^t J_2^t \bar{v}_{r2}, \ldots, a_n^t J_n^t \bar{v}_{rn}] = V_d^t A. \tag{6.3}$$

The mesh V_d^t can be reconstructed from its VG representation $\bar{V}_d^t$ as discussed in section 3.1. In Figure 6.1 and Figure 6.2, we compare VG representation with Deformation Gradient (DG) [1], Linear Rotation Invariant coordinates (LRI) [10] and Manifold Representation (MR) [2] for interpolation and morphing applications. The performance of VG is comparable with all the three representations for interpolation. For morphing, VG and Deformation Gradient performs better than Linear Rotational Invariant and Manifold Representation. Since the deformation of each mesh element is computed independently the approach becomes computationally inefficient for real-time applications.

During the smooth mesh deformation, the deformations of neighboring faces of the mesh are correlated and such a relationship can help in reducing the computational complexity. Assuming such a dependency, we segment the mesh into patches by the K-means algorithm where each patch contains a set of faces whose deformations are correlated with each other. The segmented mesh serves as a low-resolution control structure. The patch deformations if applied directly to the corresponding set of faces to deform the mesh, it produces discontinuity at the boundary of the patches because of small incompatibility between the deformation of the patch and the deformation of mesh elements. In order to prevent the discontinuity artifacts, we establish a map between the patch deformation and the mesh deformation using poses available in the dataset. The established map takes care of small incompatibility. Further, the mesh deformation is computed by representing it with the Vector Graph representation [72, 3], which is a collection of vectors. The vector deformation and the patch deformation are calculated as the composition of rotation and scaling, which are commutative transformations.

Hence, it favours us to simplify the mesh deformation process. It is possible to deform the low-resolution structure in real-time, followed by the mesh deformation through the established map. We explore three applications; interpolation, deformation transfer, and morphing using the established map. We achieve qualitatively similar and computationally efficient results for all three applications compared to other methods.

6.2 Mesh Segmentation



(a) # Patches=50 **(b)** # Patches=50 **(c)** # Patches=40

Figure 6.3: Three meshes with color coded segmented patches. The segmentation is performed using K-means algorithm.

Let's consider $\mathcal{Y} = \{y_1, y_2, \ldots, y_l\}$ is a set of rotations of l faces where, $y_j \in [-1,1]^{9p}$ and p is number of poses. The faces are segmented into m patches using K-means classifier based on the similarity of y_j's between the faces. The deformation of a patch is then computed as describe below. The faces in a patch is represented by the VG representation and h_j is the set of indices of vectors for the patch j. Assuming that the patch vertices undergo similar deformation, the rotation of the patch j of pose k is computed by solving following constraint optimization,

$$\begin{aligned} \underset{R_j^k}{\text{argmin}} \quad & \|R_j^k \mathcal{V}_{rj} - \mathcal{V}_{kj}\|_{\mathcal{F}} \\ \text{subject to} \quad & (R_j^k)' R_j^k = I_3, \text{ and, } det(R_j^k) = +1 \end{aligned} \tag{6.4}$$

where $\mathcal{V}_{rj} \in \mathbb{R}^{3\times|h_j|}$ and $\mathcal{V}_{kj} \in \mathbb{R}^{3\times|h_j|}$ consist of vectors corresponding to patch j of reference and k^{th} poses respectively. By solving above equation, we get the rotation of the patch j of pose k as,

$$R_j^k = XZ' \tag{6.5}$$

where, X and Z are orthonormal matrices derived by computing SVD of $\mathcal{V}_{rj}\mathcal{V}'_{kj} = X\Sigma Z'$. The scaling of the patch j is computed as,

$$b_j = \frac{1}{|h_j|}\sum_{t\in h_j} a_t \tag{6.6}$$

where, $a_t = \|\bar{v}_{rt}\|/\|\bar{v}_{kt}\|$ is the scaling of vector t. Similar to the vector deformation group, the deformations of patches $\{R_j^k, b_j^k\}$ also form a group denoted as G_p which is also a smooth manifold. The dimension of this manifold is $4m$ where, m is number of patches. Since the dimension of the manifold G_p is small compared to manifold G_v, manipulating deformation of patches is computationally efficient. However, applying the patch deformation directly to affiliated faces may cause discontinuity artifacts at the patch boundaries. To nullify such artifacts, we establish a map between both the manifolds G_p and G_v. Figure 6.3 shows the segmented patches on various meshes. Observe the woman mesh, she wears highly deformable cloths. The patches are segmented nicely by proposed approach for such a complex case too.

6.3 Manifold Mapping

After identifying the patches, our aim is to establish a map between patch deformation manifold G_p and vector deformation manifold G_v. The map $\chi = (\chi_{z_1 1}, \chi_{z_2 2}, \ldots, \chi_{z_n n})$ is defined as,

$$\chi : G_p \mapsto G_v. \tag{6.7}$$

where, $\chi_{z_i i}$ maps the deformation of patch z_i to the deformation of corresponding vector i. We have already knew that the vector i belongs to the patch z_i based on

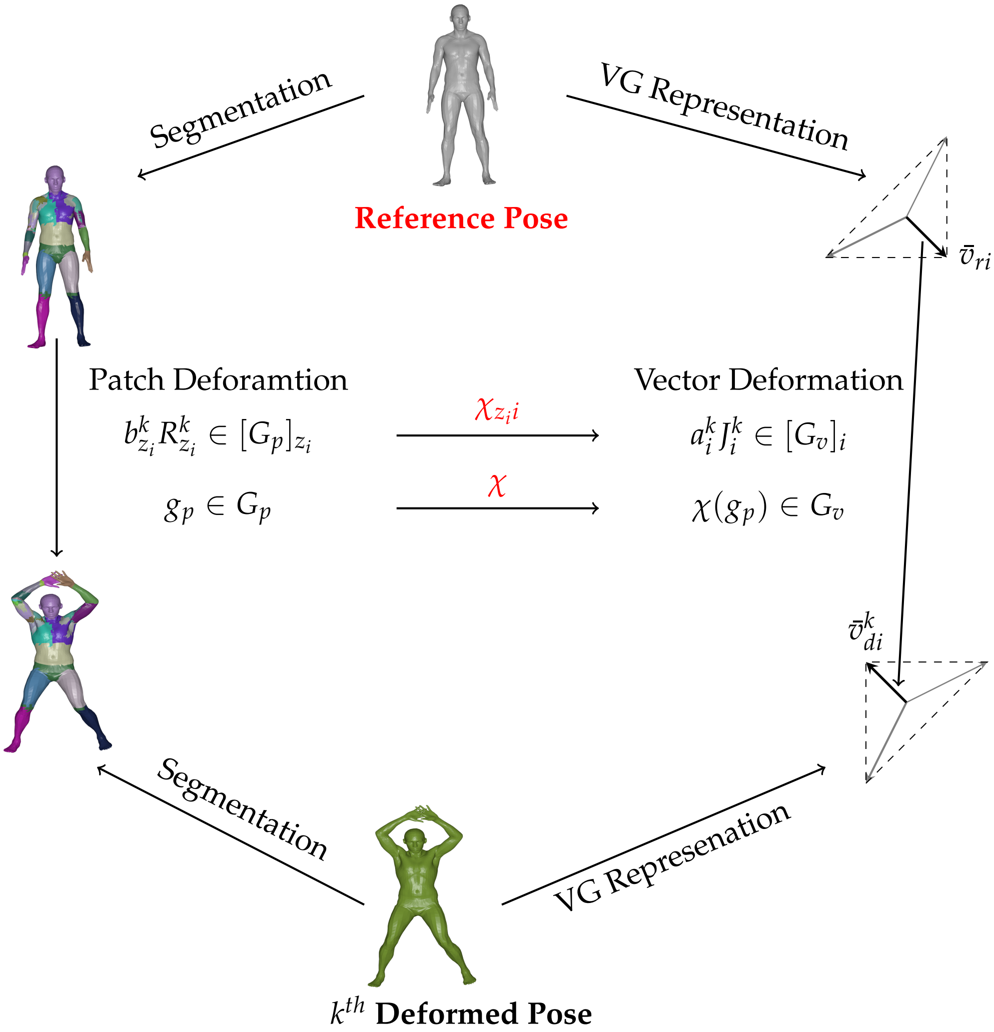


Figure 6.4: The illustration of map establishment processes. The map χ between patch deformation manifold G_p and vector deformation manifold G_v is established using p example poses of a mesh.

segmentation information. The map $\chi_{z_i i}$ is defined as,

$$\chi_{z_i i} : b_{z_i} R_{z_i} \mapsto a_i J_i. \tag{6.8}$$

The map $\chi_{z_i i}$ is modelled as; rotation and scaling. The parameters of $\chi_{z_i i}$ are computed using example poses available in the database. The rotation of the patch z_i and the rotation of vector i are,

$$R_{z_i} H_{z_i i} = J_i \tag{6.9}$$

where, $H_{z_i i}$ is a rotational component of the map $\chi_{z_i i}$. The relationship between scaling of patch z_i and vector i is defined as,

$$b_{z_i} s_{z_i i} = a_i \tag{6.10}$$

where $s_{z_i i} \in \mathbb{R}^+$ is the scaling component of the map $\chi_{z_i i}$. These components ($H_{z_i i}$ and $s_{z_i i}$) are estimated using example poses available in the database. All the maps $\chi_{z_i i}$ are injective maps so, the map χ is an injective map as explained in section 5.3.3.

6.3.1 Learning $\chi := (H, s)$

The map $\chi_{z_i i}$ is established by estimating its parameters using p example poses. The rotational component is estimated by solving following optimization problem,

$$\begin{aligned} &\underset{H_{z_i i}}{\text{argmin}} && \sum_{k=1}^{p} \| R^k_{z_i} H_{z_i i} - J^k_i \|_{\mathcal{F}} \\ &\text{subject to} && H'_{z_i i} H_{z_i i} = I_3, \text{ and, } det(H_{z_i i}) = +1 \end{aligned} \tag{6.11}$$

where, $\mathcal{F}$ is Frobenius norm. Above equation is simplified as explained in [73] as,

$$\begin{aligned} &\underset{H_{z_i i}}{\text{argmax}} && Tr(H_{z_i i} \mathcal{J}'_i \mathcal{R}_{z_i}) \\ &\text{subject to} && H'_{z_i i} H_{z_i i} = I_3, \text{ and, } det(H_{z_i i}) = +1 \end{aligned} \tag{6.12}$$

where, $\mathcal{R}_{z_i} = [R'^1_{z_i}, R'^2_{z_i}, \ldots, R'^p_{z_i}]'$ and $\mathcal{J}_i = [J'^1_i, J'^2_i, \ldots, J'^p_i]'$. The rotation $H_{z_i i}$ is computed as suggested in [73] as,

$$H_{z_i i} = XZ' \tag{6.13}$$

where, X and Z are orthonormal matrices derived by computing SVD of $\mathcal{J}'_i \mathcal{R}_{z_i} = X \Sigma Z'$. After estimating the rotational parameter, the scaling parameters $s_{z_i i}$ are computed as,

$$s_{z_i i} = \gamma^1 \frac{a^1_i}{b^1_{z_i}} + \gamma^2 \frac{a^2_i}{b^2_{z_i}} + \cdots + \gamma^p \frac{a^p_i}{b^p_{z_i}}. \tag{6.14}$$

where $a_i^k/b_{z_i}^k$ is the ratio of the scaling of vector i and patch z_i corresponding to pose k. The convex weights γ^k $k \in \{1, 2, \dots, p\}$ are computed as,

$$\gamma^k = \frac{e^k}{\sum_{z=1}^{p} e^z}, \text{ where, } e^k = \frac{1}{\|R_{z_i}^k H_{z_i i} - J_i^k\|_{\mathcal{F}}}. \tag{6.15}$$

Similarly, we establish all other maps $\chi = \{\chi_{z_1 1}, \chi_{z_2 2}, \dots, \chi_{z_n n}\}$ by approximating their parameters. The deformation of a vector is not related to the deformations of all the patches. For a humanoid mesh example, deformation of a vector belonging to head doesn't have any relationship with a patch belonging to the leg. Hence, we relate deformation of a vector i and deformation of patch z_i by $\chi_{z_i i}$. Such relationship leads to a sparse map.

6.3.2 Relating Patch and Mesh Deformations

After approximating parameters of the map, the vector $\bar{v}_{ri}$ is deformed using rotation J_i and scaling a_i defined in equation 3.13 as,

$$\bar{v}_{di} = a_i J_i \bar{v}_{ri}. \tag{6.16}$$

The rotation (J_i) and scaling (a_i) are computed from a patch deformation using estimated parameters. Substituting $a_i = b_{z_i} s_{z_i i}$ and $J_i = R_{z_i} H_{z_i i}$, in above, we get,

$$\bar{v}_{di} = b_{z_i} s_{z_i i} R_{z_i} H_{z_i i} \bar{v}_{ri} = \chi_{z_i i} \bar{v}_{ri}. \tag{6.17}$$

where $s_{z_i i}$ and $H_{z_i i}$ are the parameters of the map $\chi_{z_i i}$. Since scaling and rotation commute (as explained in section 3.4.3), above equation can be rewritten as,

$$\bar{v}_{di} = b_{z_i} R_{z_i} (s_{z_i i} H_{z_i i} \bar{v}_{ri}). \tag{6.18}$$

The deformation of vectors can be computed from corresponding patch deformation. The deformed VG representation is,

$$\begin{aligned}\bar{V}_d =&[\bar{v}_{d1}, \bar{v}_{d2}, \dots, \bar{v}_{dn}] &(6.19)\\ =&[b_{z_1} R_{z_1}(s_{z_1 1} H_{z_1 1} \bar{v}_{r1}), b_{z_2} R_{z_2}(s_{z_2 2} H_{z_2 2} \bar{v}_{r2}), \dots, b_{z_n} R_{z_n}(s_{z_n n} H_{z_n n} \bar{v}_{rn})]. &(6.20)\end{aligned}$$

where, $z_i \in \{1, 2, \dots, m\}$ is the index of the patch in which vector i belongs to. Above equation further be simplified as,

$$\bar{V}_d = [b_1 R_1, b_2 R_2, \dots, b_m R_m] K = BK. \tag{6.21}$$

where the matrix K is constructed as shown in Appendix A.7. The deformed mesh is reconstructed using reconstruction process from the deform VG representation $\bar{V}_d$ as explained in section 3.9,

$$\hat{V}_d = \tilde{V} \hat{A}^+ = (\bar{V} - v_c A_l) \hat{A}^+. \tag{6.22}$$

where, v_c is a constraint vertex which is set at origin of $0_3 \in \mathbb{R}^3$. $\hat{V}_d$ contains vertices of deformed mesh, the constraint vertex v_c is appended later to get the full deform mesh V_d. We rewrite equation 6.22 as,

$$\hat{V}_d = \tilde{V} \hat{A}^+ = \bar{V} \hat{A}^+ = BK \hat{A}^+ = BM. \tag{6.23}$$

where, matrix $M = K \hat{A}^+$ can be precomputed. Appending constraint vertex to $\hat{V}_d$, we get the deformed mesh as

$$V_d = [BM, v_c] = [BM, 0_3] = [BM, B0_{3m}] = B\tilde{M} \tag{6.24}$$

where, $\tilde{M}$ is formed by appending 0_{3m} (origin of $\mathbb{R}^{3m}$) to M at the column corresponding to the constraint vertex v_c. The matrix $\tilde{M}$ is precomputed using mesh connectivity, map parameters and VG representation of the reference mesh. Above equation is a simplified relation between deformed mesh V_d and patch deformation (B).

6.4 Mesh Sequence Interpolation

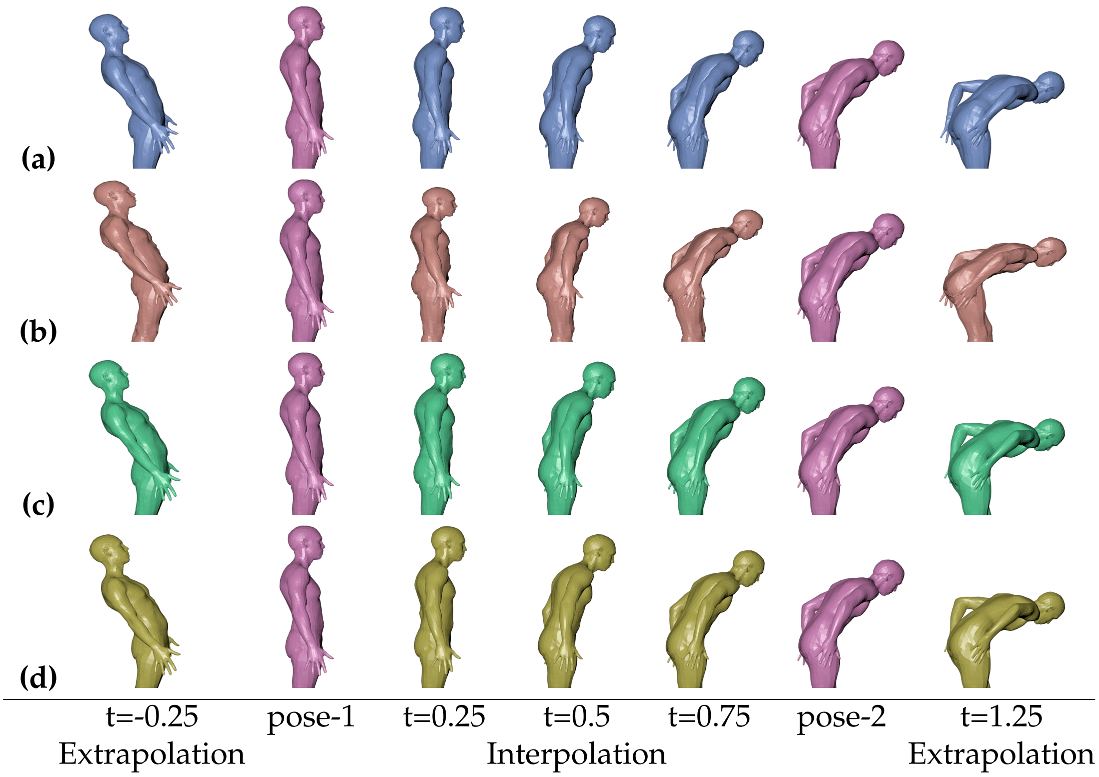


Figure 6.5: Comparison of proposed interpolation method with other methods. Interpolated and extrapolated poses by (**a**) proposed method, (**b**) Linear Rotation Invariant (LRI) coordinates [10], (**c**) Poisson interpolation [11] and (**d**) Lie body representation [12].

Primary objective of the interpolation is to generate intermediate poses between two meshes. It typically manifests transition from one mesh to the other. In the proposed interpolation method, the patch deformations for two meshes represent two points on the patch deformation manifold G_p. For the interpolation, a geodesic (say β_b) between these two points is approximated on the manifold G_p. Each point on the approximated geodesic are mapped to a point on the vector deformation manifold (G_v) using established map $\chi = \{\chi_{z_1 1}, \chi_{z_2 2}, \dots, \chi_{z_n n}\}$. Hence, the map χ maps the geodesic β_b to $\chi(\beta_p) = \beta_v$, where, β_v is a curve on the manifold G_v. Note that the curve β_v is not a geodesic. Performing interpolation on the patch deformation manifold G_p is computationally demanding then on G_v. Moreover, the deformed mesh is approximate using the simplified equation (6.24).

Hence, the proposed interpolation method is computationally real-time even for high resolution meshes (see Table 6.1 and Table 6.2). The interpolation process is illustrated graphically in Figure 6.6. Let's consider two mesh poses V_1 and V_2. The deformations of patch j of V_1 and V_2 w.r.t. reference mesh are denoted as $b_{1j}R_{1j}$ and $b_{2j}R_{2j}$ respectively. The interpolation between both deformations is defined as,

$$R_j(t) = R_{1j}exp(t * log(R'_{1j}R_{2j})) \tag{6.25}$$

$$b_j(t) = b_{1j}exp(t * log(b_{1j}^{-1}b_{2j})) \tag{6.26}$$

where, $t \in [0, 1]$ ($t \in R$) is the interpolation (extrapolation) factor. A geodesic on the patch deformation manifold is generated by interpolating all other patch deformations similarly. From the approximated geodesic using patch deformation manifold (G_p), the curve β_v on the vector deformation manifold can be generated using on established map χ. The patch deformation matrix B which is defined in equation (6.24) is computed for interpolation (extrapolation) factor t as,

$$B(t) = [b_1(t)R_1(t), b_2(t)R_2(t), \ldots, b_m(t)R_m(t)]. \tag{6.27}$$

Substituting the patch deformation matrix $B(t)$ into equation (6.24), the interpolated pose approximated for the factor t is,

$$V_d(t) = B(t)M. \tag{6.28}$$

The intermediate poses between V_1 and V_2 are generated by varying $t \in [0, 1]$ ($t \in R$ for extrapolation).

6.5 Deformation Transfer

In deformation transfer, the deformation of a mesh (source) is transferred to another mesh (target) to generate its deformed pose through established correspondence. As shown in the Figure 6.6, the map χ between patch deformation mani-

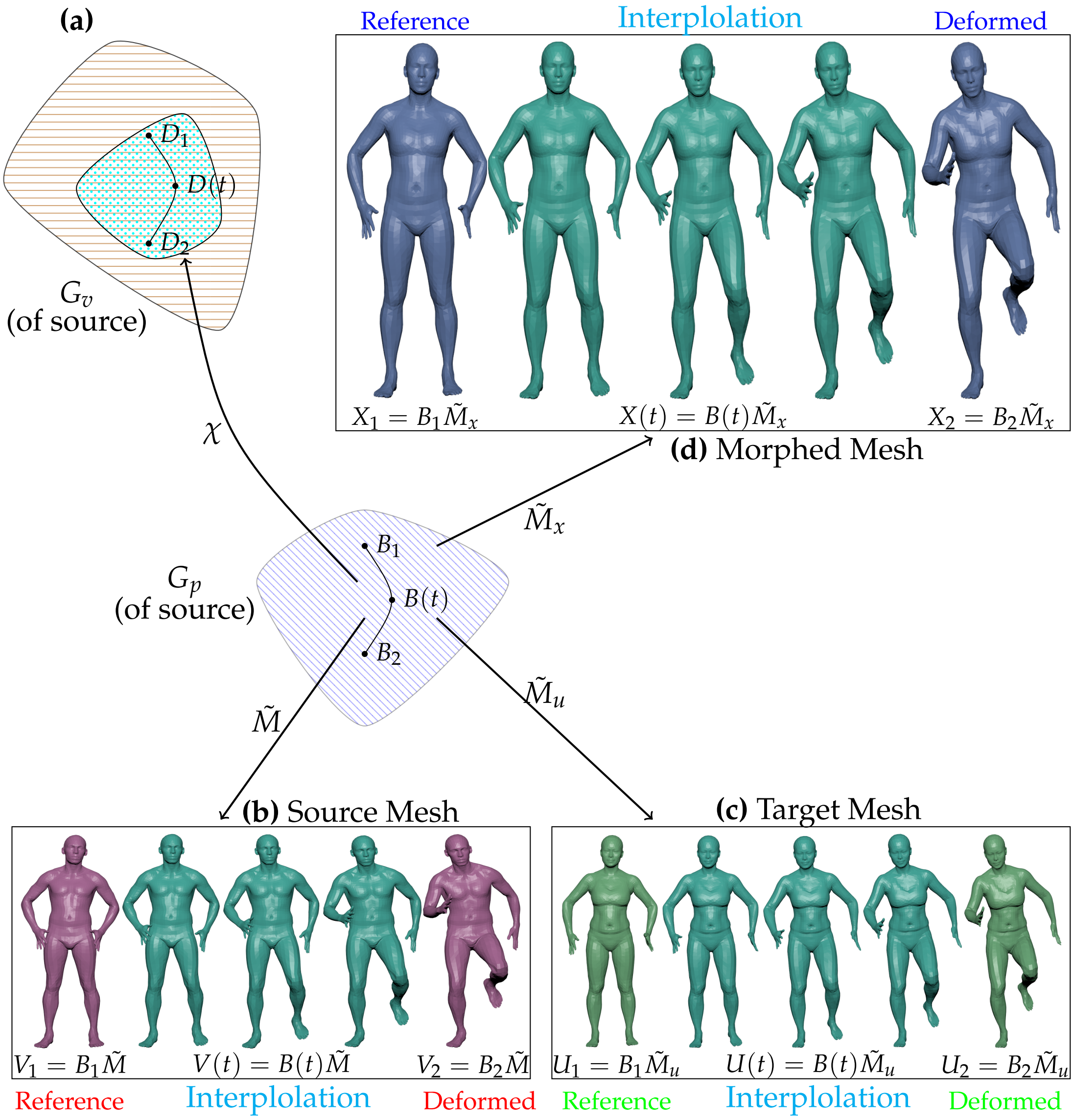


Figure 6.6: The illustration of map establishment, interpolation and deformation transfer processes. **(a)** The map χ between patch deformation manifold G_p and vector deformation manifold G_v is established using p example poses of a mesh (man). **(b)** The source meshes are interpolated by interpolating patch deformation (B) and then multiplying it to M (see section 6.4 for derivation). **(c)** The established map χ for the source (man) mesh can be transferred to the target mesh (woman) in order to perform interpolation and deformation transfer on the target mesh (refer section 6.5 for detailed explanation). **(d)** Using the patch deformations (B), the interpolation and Deformation Transfer can also be performed on morphed mesh (refer section 6.6 for detailed explanation)

fold G_p and vector deformation manifold G_v of the source reference mesh (V_r) is established using p available poses. Next, we transfer the established map χ to the target reference mesh (U_r). Here, we assume that the face wise correspondence is available between V_r and U_r. The correspondence can be established using mesh registration methods available in literature such as [1]. Let's consider i^{th} vector of the target mesh U_r as $\bar{u}_{ri}$. For Deformation Transfer, the corresponding source vector $\bar{v}_{ri}$ is replaced by the vector $\bar{u}_{ri}$ in equation (6.17). The target deformed vector is given by,

$$\bar{u}_{di} = b_{z_i} R_{z_i} (s_{z_i i} H_{z_i i} \bar{u}_{ri}) \tag{6.29}$$

where, $\bar{u}_{di}$ is the deformed vector corresponding to the deformed target mesh. We compute other deformed target vectors similarly. The deformed target VG is given by,

$$\bar{U}_d = [\bar{u}_{d1}, \bar{u}_{d2}, \ldots, \bar{u}_{dn}] \tag{6.30}$$

$$= [b_{z_1} R_{z_1} (s_{z_1 1} H_{z_1 1} \bar{u}_{r1}), b_{z_2} R_{z_2} (s_{z_2 2} H_{z_2 2} \bar{u}_{r2}), \ldots, b_{z_n} R_{z_n} (s_{z_n n} H_{z_n n} \bar{u}_{rn})] \tag{6.31}$$

where $z_i \in \{1, 2, \ldots, m\}$ is the index of the patch. Above equation can be simplified as,

$$\bar{U}_d = [b_1 R_1, b_2 R_2, \ldots, b_m R_m] K_u = B K_u. \tag{6.32}$$

where matrix K_u can be precomputed from the map χ and target mesh U_r. It can be formed similar to matrix K (see Appendix A.7). The matrix B is obtained from the patch deformations of the source mesh. The deformed target mesh can be reconstructed from its VG representation as,

$$\hat{U}_d = \bar{U} \hat{A}_u^+ = B K_u \hat{A}_u^+ = B M_u \tag{6.33}$$

where, the connection matrix A_u^+ is formed from the connectivity of the target mesh. Appending constraint vertex to $\hat{U}_d$, we get the deformed mesh as,

$$U_d = [BM, 0_3] = [BM, B0_{3m}] = B\tilde{M}_u \tag{6.34}$$

where, $\tilde{M}_u$ is formed from the established map χ, reference target mesh and the matrix A_u. As explained in section 6.4, replacing B with $B(t)$, the interpolation can also be performed on target meshes (see Figure 6.11).

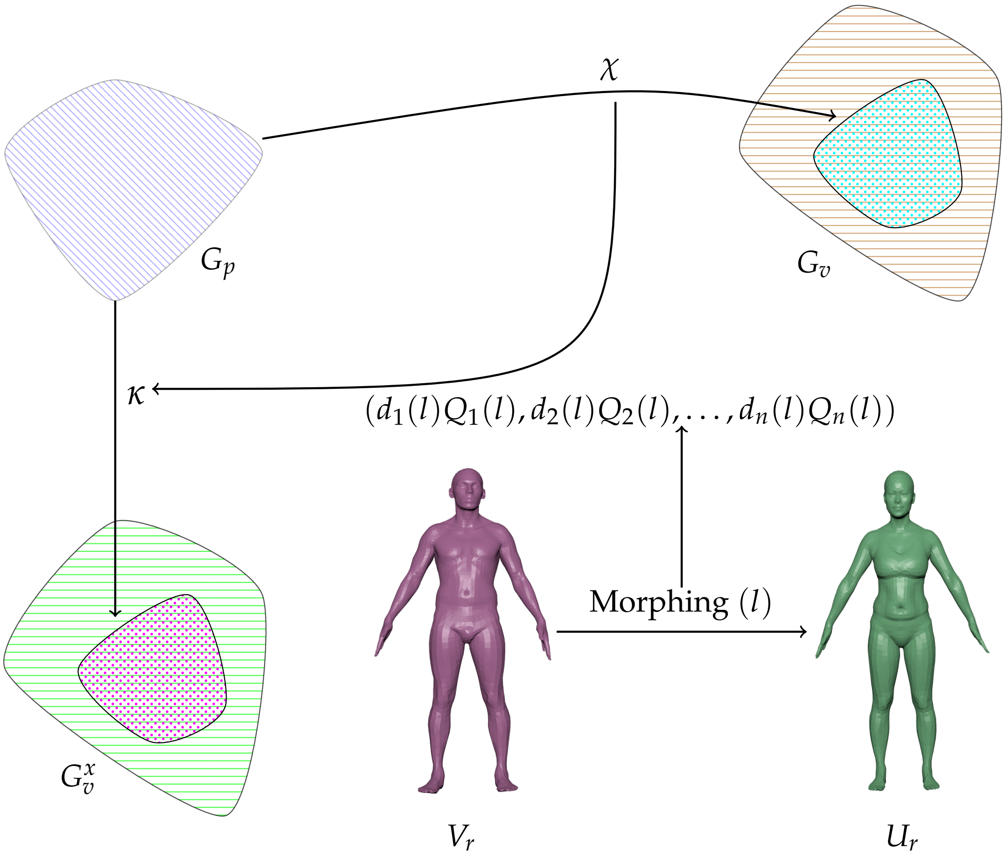


Figure 6.7: The map χ is established for the source mesh (a men) using p example poses as explained in section 6.3. The morphed mesh is generated between reference source mesh V_r and the reference target mesh U_r. To interpolate the morphed mesh, the map χ is modified to κ using the deformation of morphed mesh $\{d_1(l)Q_1(l), d_2(l)Q_2(l), \ldots, d_n(l)Q_n(l)\}$ which is obtained by interpolating deformation between V_r and U_r. l is the morphing factor $l \in [0,1]$. The G_v^x is the vector deformation manifold of the morphed mesh $X(l)$.

6.6 Interpolation for Morphed Model

In the shape morphing, the objective is to perform interpolation between a mesh of an object (source) and a mesh of another object (target). Let's consider the morphed object mesh $X_r(l)$ which represents the transition from the object V_r to another object U_r where, l is the morphing factor. Since both the meshes (V_r and U_r) may have different resolutions, the face wise correspondences should be computed using mesh registration method such as [1]. We assume here that such a correspondence is available between V_r and U_r. Such a correspondence is represented in the form of pairs of indices of vectors. Let's consider any such pair ($\bar{v}_{ri}$ and $\bar{u}_{ri}$) corresponding to the i^{th} vectors of meshes V_r and U_r. The relation between $\bar{v}_{ri}$ into $\bar{u}_{ri}$ is defined as,

$$\bar{u}_{ri} = d_i Q_i \bar{v}_{ri} \tag{6.35}$$

where, scaling d_i and rotation Q_i can be computed from $\bar{v}_{ri}$ and $\bar{u}_{ri}$ as explained in section 3.3. The morphed mesh $X_r(l)$ is generated by interpolating the deformations d_i and Q_i using logarithm and exponential maps as,

$$Q_i(l) = exp(l * log(Q_i)),\ d_i(l) = exp(l * log(d_i)) \tag{6.36}$$

$$\bar{x}_{ri} = Q_i(l) d_i(l) \bar{v}_{ri} \tag{6.37}$$

where, $l \in [0, 1]$ is a morphing factor and $\bar{x}_{ri}$ is the vector i of the morphed mesh ($X_r(l)$). To perform the Deformation Transfer and the interpolation on the morphed model $X_r(l)$, the map $\chi = (\chi_{z_1 1}, \chi_{z_2 2}, \ldots, \chi_{z_n n})$ established for the mesh V_r is modified to the map $\kappa = (\kappa_{z_1 1}, \kappa_{z_2 2}, \ldots, \kappa_{z_n n})$ for the morphed mesh $X_r(l)$. The deformed vector $\bar{x}_{di}$ of the deformed morphed model $X_r(l)$ is computed as,

$$\bar{x}_{di} = \chi_{z_i i} \bar{x}_{ri} = \chi_{z_i i} d_i(l) Q_i(l) \bar{v}_{ri} = \kappa_{z_i i} \bar{v}_{ri} \tag{6.38}$$

where, $\kappa_{z_i i}$ is the map which maps deformation of the patch z_i of the mesh V_r to deformation of vector i of the morphed mesh. After establishing the map κ, the

VG representation of a deformed morphed mesh is computed as,

$$\begin{aligned} \bar{X}_d = [\bar{x}_{d1}, \bar{x}_{d2}, \ldots, \bar{x}_{dn}] = [b_{z_1} R_{z_1} (s_{z_1 1} H_{z_1 1} d_1(l) Q_1(l) \bar{v}_{r1}), \\ b_{z_2} R_{z_2} (s_{z_2 2} H_{z_2 2} d_2(l) Q_2(l) \bar{v}_{r2}), \ldots, b_{z_n} R_{z_n} (s_{z_n n} H_{z_n n} d_n(l) Q_n(l) \bar{v}_{rn})]. \end{aligned} \tag{6.39}$$

$$\bar{X}_d(l) = [b_1 R_1, b_2 R_2, \ldots, b_m R_m] K_x \tag{6.40}$$

where $z_i \in \{1, 2, \ldots, m\}$ is index of the patch of the mesh V_r. The matrix K_x is formed similarly as matrix K (see Appendix A.7). As explained in the section 6.4, the deformed mesh can be obtained by simplifying above equation as,

$$X_d(l) = [b_1 R_1, b_2 R_2, \ldots, b_m R_m] \tilde{M}_x = B \tilde{M}_x \tag{6.41}$$

where, matrix $\tilde{M}_x$ is approximated from the established map κ, the VG representation of reference mesh V_r and connection matrix A of the mesh V_r. The matrix B is obtained from the deformed source mesh V_d to generate the corresponding deformed mesh $X_d(l)$. We substitute $B(t)$ in the equation (6.41) to generate interpolated poses of the morphed mesh $X_r(l)$ as explained in section 6.4. This process is illustrated in Figure 6.7.

6.7 Results and Experiments

To compare the proposed approach for interpolation, morphing and Deformation Transfer, we have carried out experiments on various datasets. In the process, we have created an add-on for Blender to perform these tasks. All the experiments have been carried out using intel-i7-2.8GHz, 8 core system with 8 GB memory. All the methods have been implemented as *single threaded* python programs.

We have compared our method with various interpolation [11, 10, 12] and deformation transfer [1, 2, 3] methods on both aspects: quality and computation. Table 6.1 and Table 6.2, describe 23 different models used from various datasets: 10 (500xx) from [4], 6 (Horse, Camel, Flamingo, Elephant, Lion and Cat) from [1], 4 models from [5], 1 (Hand) from [75] and 2 (Kid1 and Kid2) from [76].

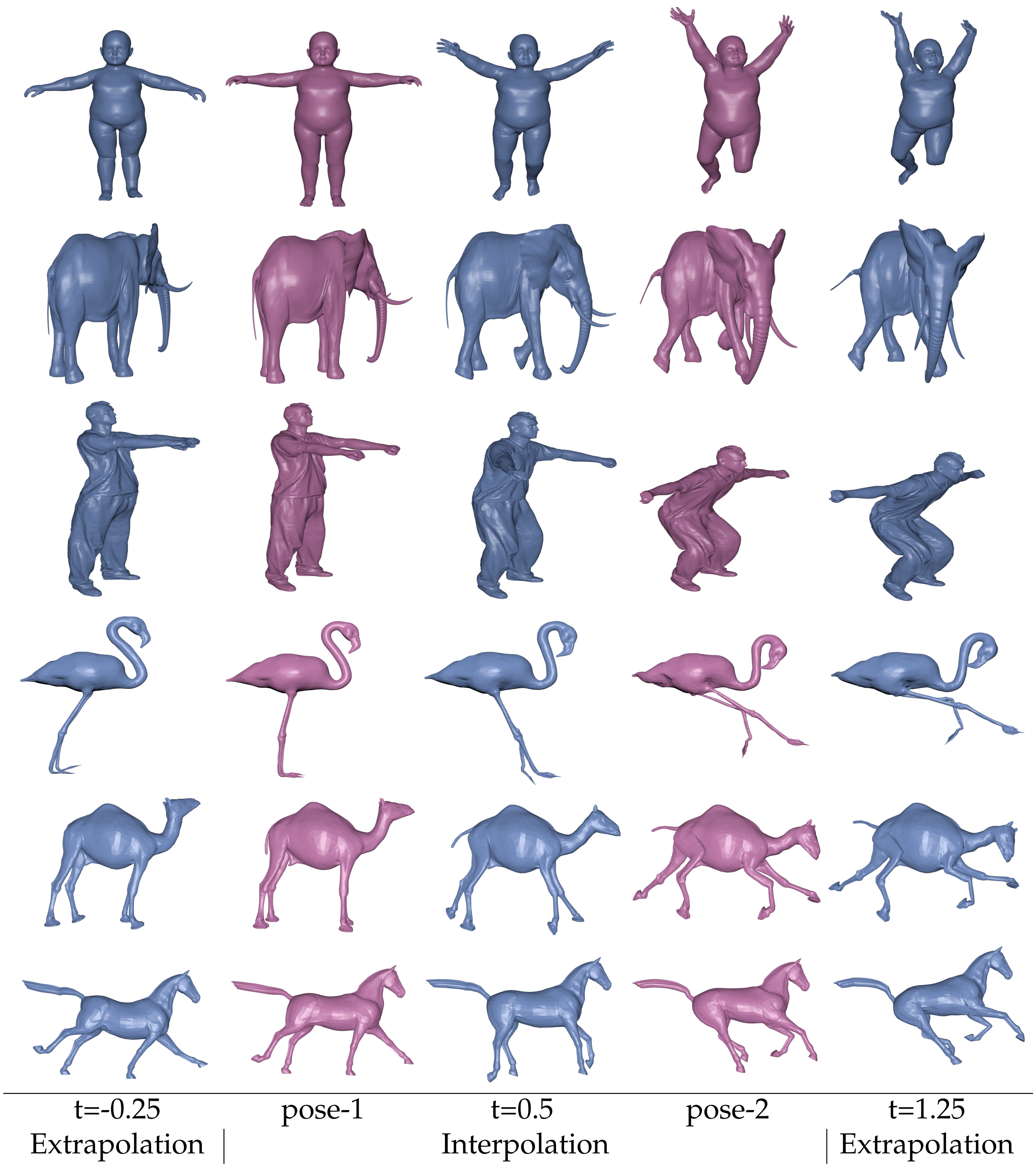


Figure 6.8: Interpolation and extrapolation using proposed method on various meshes.

6.7.1 Training Error

For the proposed data-driven approach, the map between the patch deformation manifold and the vector deformation manifold is established by solving a least square constraint optimization problem to estimate its parameters. Hence, it may not reproduce the example poses with 100% precision. To measure the inaccuracy in the training process, we compute two statistical measures; orientation (E_{oi}) and

Model	Verts	Training Poses	Segments	Training Error			
				Orientation		Area	
				Mean $\mu_o(10^{-3})$	Std $\sigma_o(10^{-2})$	Mean $\mu_a(10^{-2})$	Std σ_a
50002	6890	480	50	3.72	2.301	3.85	1.26
50004	6890	480	50	4.45	2.60	4.24	0.14
50007	6890	457	50	3.95	2.47	3.72	0.15
50009	6890	312	50	3.00	1.65	3.18	0.09
50020	6890	325	50	2.88	1.49	3.56	0.12
50021	6890	382	50	3.03	1.57	3.66	0.09
50022	6890	499	50	3.90	1.98	3.83	0.12
50025	6890	515	50	3.17	1.67	3.66	0.14
50026	6890	488	50	4.65	2.36	3.97	0.14
50027	6890	409	50	3.45	1.58	3.57	0.11
Horse	8431	11	40	13.23	4.83	14.81	0.27
Camel	21887	11	40	4.85	2.05	8.12	0.17
Flamingo	26394	11	40	4.49	1.82	11.53	0.15
Ben	10002	175	50	15.87	3.13	14.59	0.28
Ben2	9971	150	50	15.38	4.43	19.38	0.22
Ben3	10002	150	50	54.73	28.93	96.54	0.19
Ben4	10002	250	50	13.42	3.91	19.84	0.51
Hand	7929	9	30	7.49	5.10	8.25	0.19
Kid1	59727	16	50	9.29	4.27	9.88	0.47
Kid2	59727	16	50	13.52	5.76	12.05	0.71
Elephant	42321	10	60	4.29	2.24	4.24	0.09
Lion	5000	10	40	25.21	8.56	24.23	1.59
Cat	7207	10	40	27.28	8.98	25.25	1.12

Table 6.1: The table shows configuration of various meshes and the pose estimation error. *std* indicates standard deviation.

area (E_{ai}) errors for the face i of the mesh as,

$$E_{oi} = |1 - cos\theta_i| = |1 - N_i'\tilde{N}_i| \in [0,2], \text{ and } E_{ai} = |1 - \frac{\mathcal{A}_i}{\tilde{\mathcal{A}}_i}| \in \mathbb{R}^+ \qquad (6.42)$$

where, N_i and $\tilde{N}_i$ are unit normals of i^{th} faces of the estimated and the ground truth poses respectively. θ_i is the angle between N_i and $\tilde{N}_i$. $\mathcal{A}_i$ and $\tilde{\mathcal{A}}_i$ are areas of i^{th} faces of the estimated and the ground truth poses respectively. Both the errors E_{oi} and E_{ai} quantify the inaccuracy in the orientation and the area of the estimated face i. Since the map is established using p available poses of the mesh, both the errors are computed for every face of all available poses. We compute the mean (μ_o and μ_a) and standard deviation (σ_o and σ_a) of the both the errors (E_{oi} and E_{ai})

Model	Interpolation							
	Proposed		DG [11]		Lie [12]		LRI [10]	
	Prepr. (s)	itr (ms)	Prepr. (s)	itr (ms)	Prepr. (s)	itr (ms)	Prepr. (s)	itr (ms)
50002	2473.68	1.10	6.3	508.52	5.16	406.7	4.72	539.76
50004	2524.92	1.04	6.44	528.00	5.26	406.34	4.76	538.47
50007	2352.46	1.06	6.38	523.48	5.14	401.70	4.76	544.54
50009	2352.46	1.06	6.38	523.48	5.14	401.7	4.77	543.78
50020	1673.24	1.12	6.42	547.66	5.04	413.38	4.68	543.11
50021	1979.06	1.08	6.42	538.12	5.22	401.58	4.79	535.20
50022	2597.2	1.06	6.36	534.74	5.2	406.60	4.72	538.74
50025	2648.54	1.05	6.30	526.08	5.04	391.44	4.65	539.47
50026	2547.58	1.10	6.48	528.32	5.24	398.88	4.74	544.13
50027	2524.22	1.03	6.32	532.16	5.16	398.38	4.79	550.49
Horse	82.98	1.06	7.76	656.68	6.28	493.18	5.82	666.27
Camel	223.94	1.33	20.44	1683.5	15.98	1249.94	15.15	1762.93
Flamingo	282.54	1.81	24.22	2021.00	19.64	1536.44	17.94	2058.73
Ben	1358.98	1.28	9.14	762.12	7.52	588.74	6.82	776.36
Ben2	1152.48	1.32	9.06	747.68	7.52	590.32	7.01	774.47
Ben3	1219.46	1.32	9.28	759.36	7.68	611.14	6.88	779.96
Ben4	2036.6	1.3	9.18	762.42	7.56	597.44	6.83	778.60
Hand	56.84	0.92	7.2	594.64	5.88	469.26	4.43	428.15
Kid1	937.5	3.31	54.94	4520.12	44.44	3344.7	40.61	4658.69
Kid2	927.12	2.56	55.24	4514.42	44.14	3337.06	40.73	4618.69
Elephant	528.12	3.46	38.92	3202.84	31.48	2369.96	28.77	3300.33
Lion	46.2	1.10	4.76	387.92	3.92	299.74	3.51	404.56
Cat	66.34	1.14	6.68	559.64	5.56	436.46	5.00	571.19

Table 6.2: The table shows configuration of various meshes and comparison of various interpolation methods in term of computational complexity. Note that the *ms* and *s* indicates the time in mili second and second respectively. *prepr.* and *itr* indicate time require for preprocessing and interpolation.

in order to represent overall inaccuracy in the proposed method as,

$$\mu_{oi} = \frac{1}{p}\sum_{k=1}^{p} E_{oi}^{k},\ \mu_{ai} = \frac{1}{p}\sum_{k=1}^{p} E_{ai}^{k} \tag{6.43}$$

$$\mu_{o} = \frac{1}{l}\sum_{i=1}^{l} \mu_{oi},\ \mu_{a} = \frac{1}{l}\sum_{i=1}^{l} \mu_{ai} \tag{6.44}$$

$$\sigma_o = \sqrt{\frac{1}{l*p}\sum_{i=1}^{l}\sum_{k=1}^{p}(E_{oi}^{k}-\mu_o)^2},\ \sigma_a = \sqrt{\frac{1}{l*p}\sum_{i=1}^{l}\sum_{k=1}^{p}(E_{ai}^{k}-\mu_a)^2} \tag{6.45}$$

where, l and p are number of faces and poses of the mesh respectively. In the Table 6.1, the mean and the standard deviation are shown for all 23 objects. The small mean and standard deviation of both the errors shows that the proposed method can accurately regenerate the training poses. We have also shown the distribution of mean error (i.e. μ_{ai} and μ_{oi}) computed for every face of the mesh and as the histogram in Figure 6.9. The error distribution on the mesh is smooth and so is the deformation of the mesh. The highest mean error is observed at the joints of the mesh because of the low correlation between the patch deformation and the deformation of the vector near joints.

Methods Model Pair		DG [1]	Lie body [2]	GDT[3]	Proposed
Lion-Cat	Preproc. (s)	1.08	1.01	1.24	8.56
	DT (ms)	1.67	5.71	6.10	0.014
Hand-Devil	Preproc. (s)	1.22	1.09	1.35	7.68
	DT (ms)	1.87	6.49	6.54	0.010
Human-Devil	Preproc. (s)	1.53	1.35	1.72	14.96
	DT (ms)	2.40	7.91	8.50	0.015
50022-50007	Preproc. (s)	1.05	0.98	1.20	9.05
	DT (ms)	1.61	5.84	6.02	0.013

Table 6.3: Comparison of proposed Deformation Transfer method with Deformation Gradient (DG) [1], Lie body [2] and Guided Deformation Transfer (GDT) [3] in terms of preprocessing time and Deformation Transfer time. Note that *s* and *ms* indicates the time in seconds and miliseconds.

6.7.2 Interpolation

Experiments for the proposed interpolation and extrapolation methods have been carried out on various meshes. The Figure 6.5, shows the qualitative comparison of the proposed interpolation method with Poisson interpolation [11], Linear Rotation invariant (LRI) [10] and Lie Body [12]. As shown in the Figure 6.5(b), the shrinkage artifacts can be observed in the interpolated pose generated by Linear Rotation invariant [10] coordinate. The intermediate pose has shrunk to smaller size during the interpolation. Using Poissson interpolation [11] and Lie body [12], the transition between two consecutive poses is smooth and natural during interpolation and extrapolation as shown in the Figure 6.5(c) and (d). The proposed

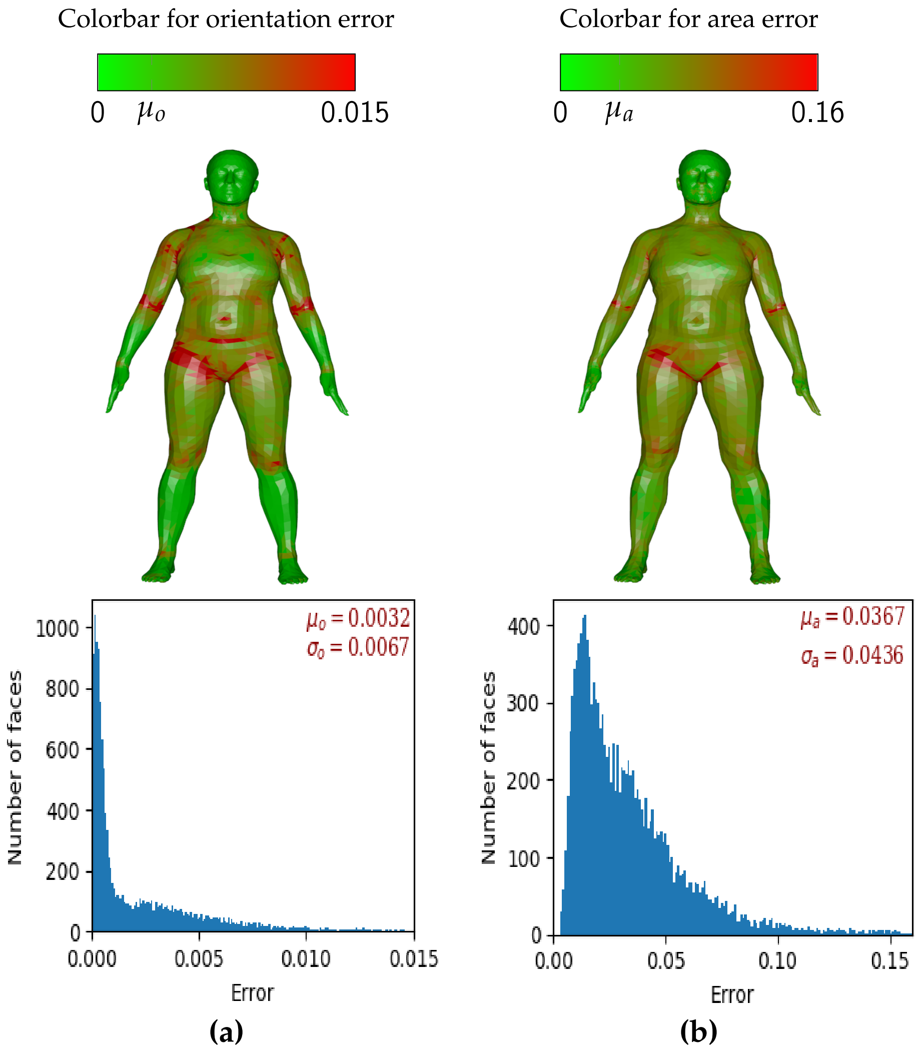


Figure 6.9: (**a**) Distribution of orientation error on mesh (top) and corresponding histogram of error (bottom). (**b**) Distribution of area error on mesh (top) and corresponding histogram of error (bottom). Note that ranges of the orientation and area errors are $[0, 2]$ and $\mathbb{R}^+$ respectively. The face i with $\mu_{oi} \geq 0.015$ and $\mu_{ai} \geq 0.15$ is assigned the red color.

method is qualitatively similar to Poisson interpolation [11] and Lie body [12]. We have also performed interpolation and extrapolation on various meshes using proposed interpolation approach as shown in Figure 6.8. Note that the poses shown in the Figure 6.8 are not used during the training process. The smooth and realistic deformation in the interpolated poses for all the cases endorse our inference. As shown in Figure 6.8 (row three), the proposed method can interpolate the complex folds in the cloth smoothly and naturally. We have also performed the interpolation on the morphed meshes as shown in Figure 6.11.

Table 6.2, describes the configuration of the various meshes and the associated computational cost to perform interpolation. The number of patches affects not only quality of deformation but also the computational performance in the proposed method. We have identified number of patches experimentally for all 23 meshes. In Table 6.2, we have compared the propose interpolation method with other methods [11, 10, 12] in terms of preprocessing time and interpolation time. The preprocessing is required only once to acquire prerequisite information for interpolation. The preprocessing time for proposed method depends on the number of poses, number of patches and resolution of the mesh whereas it depends only on resolution of the mesh for other methods. The interpolation time shown in the Table 6.2 specifies the time required to generate an interpolated pose between two key frames. Hence, the interpolation time is an appropriate choice to quantify computational performance of an interpolation method. The proposed method is able to generate approximately 289 to 1075 interpolated poses per second depending on resolution of the mesh and number of patches in the low resolution structure. Hence, the proposed method is computationally real-time. Whereas, the other interpolation methods are computationally sluggish due to large number of element-wise computations. Both qualitative and computational performances endorse our claim that the proposed method is qualitatively similar and computationally efficient than other methods.

The established map for a mesh can be transferred to approximate the map for morphed mesh as explained in the section 6.6. Once the map is approximated, the interpolation and deformation transfer are performed on various morphed meshes as shown in the Figure 6.11. Here, we have experimented with various source-target pairs by performing morphing followed by deformation transfer and interpolation on the morphed mesh. From Figure it is evident that 6.11, the geometric details of the morphed model are preserved during interpolation and deformation transfer. The computation time for interpolation remains the same as the source model interpolation as shown in the Table 6.2 because the morphed model has the same resolution and number of patches as the source mesh.

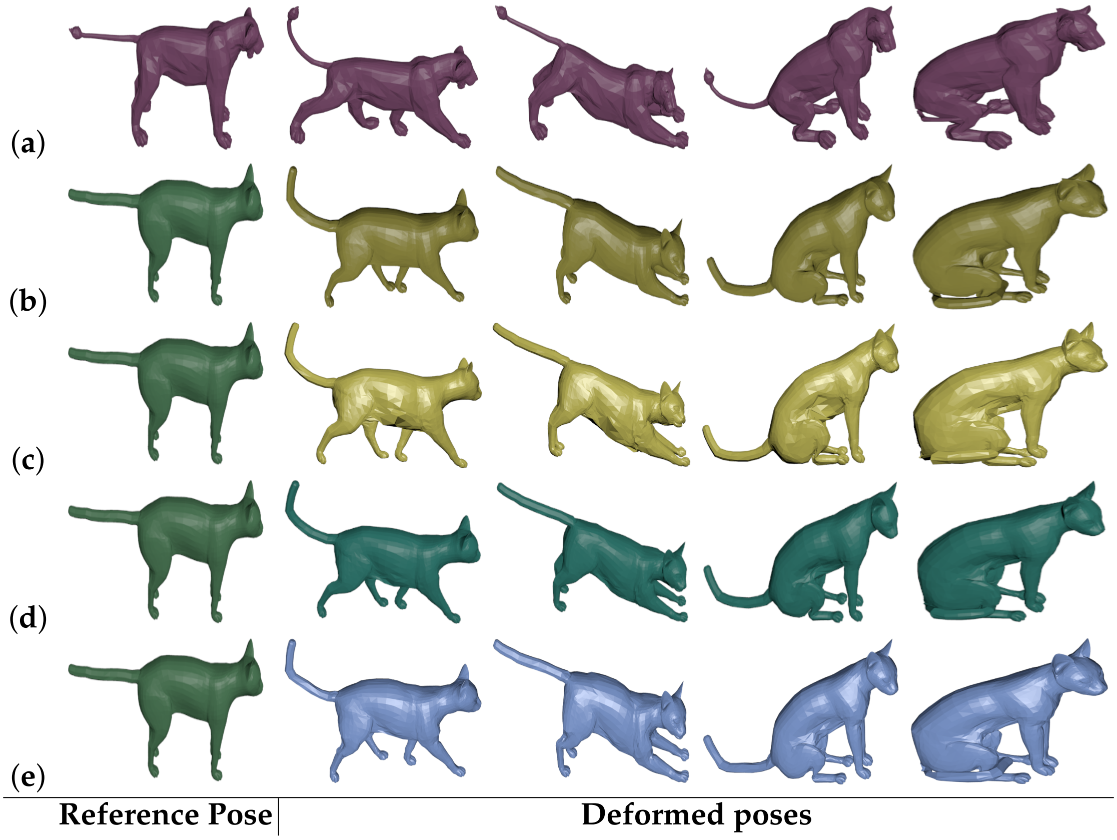


Figure 6.10: Comparing the proposed deformation transfer method with other methods. The deformation from source (**a**) to the target (cat) using Deformation Gradient (DG) [1] (**b**), Lie body [2] (**c**), Guided Deformation Transfer (GDT) [3] (**d**) and proposed method (**e**).

6.7.3 Deformation Transfer

In traditional Deformation Transfer [1, 2, 3], the deformation of each triangular face of the source mesh is transferred to corresponding triangular face of target mesh. The geometric details of the target mesh are preserved by such methods. As explained in the section 6.5, the established map for the source can be transferred to the target through the established correspondence. Unlike traditional approach, the patch deformation acquired from the deform source pose is employed to deform target mesh through transferred map. We have experimented on various source-target meshes as shown in Figure 6.10 and 6.11. The proposed method is compared with other Deformation Transfer methods such as Deformation Gradient [1], Lie Body [2] and Guided Deformation Transfer [3] qualitatively. In Figure 6.10, the deformation transfer is performed on different resolution source-target meshes such as lion-cat pair. The results of the proposed method are qualitatively similar to other methods. Apart from the Deformation Transfer, the real-time interpolation can also be performed between two target poses using transferred map from the source as shown in the Figure 6.11. The geometric details of the target mesh is preserved during both the interpolation and the deformation transfer.

We have presented the computational performance of various Deformation Transfer methods in Table 6.3. The preprocessing and the Deformation Transfer times are considered for comparing the computational requirements of these methods. The traditional methods [1, 2, 3] are computationally demanding due to element vise computation. On other hand in the proposed method since the patch deformation of the deform source pose is transferred to generate the deformed target pose is computationally efficient in comparison with other Deformation Transfer methods.

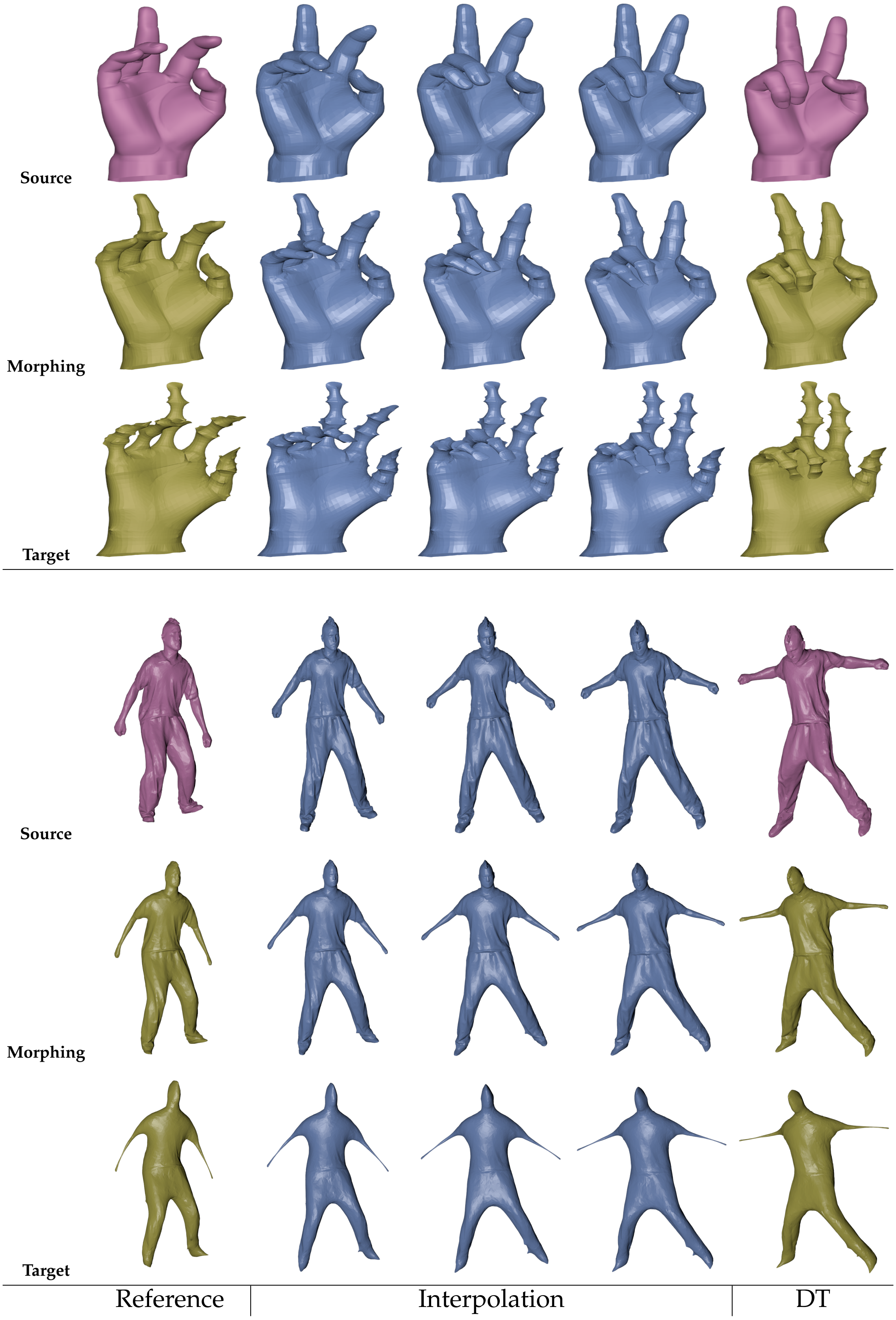


Figure 6.11: The interpolation is performed between two source meshes using established map. The interpolation and Deformation Transfer (DT) can also be performed on corresponding target meshes and morphed meshes. The morphing factor is 0.5 for both the cases.

Chapter 7

Blender Implementation

In this work, we first, propose computationally efficient algorithms that ensure the smooth and realistic deformation of the mesh for various applications. And then, implement these algorithms as the prototypes for an interactive experience. In this work, all the tasks are defined in a unified framework that involves three necessary steps. (i) The Vector Graph representation represents the mesh as a collection of vectors. (ii) The VG of the mesh is deformed by deforming vectors. (iii) From the deformed VG, the deformed mesh is reconstructed by solving a simple over-determined system of linear equations. The process of deformation of the VG (step (ii)) is different depending on the application, and it involves fundamental geometric transformations, rotation and scaling. The other two steps remain the same for all the tasks.

An open source software that allows us to visualize a mesh, access its data-structure, and create an interactive tool panel in its User Interaction (UI) is needed to implement the proposed unified framework. The Blender is the best fit. The Blender is an open source 3D creation software developed by various professionals such as artists, programmers, and graphic designers. It has provisions for rigging, animation, simulation, rendering, motion tracking, video editing, and 2D animation. The more details are found at https://www.blender.org/. Apart from its rich in-built functionality, it provides a platform where a user can create a UI panel in python. We have implemented all the proposed algorithms for different applications in Blender as an Add-ons. All other methods used for comparisons are also implemented in Blender as Add-ons. In this chapter, we document all the created Blender add-ons. This chapter will serve as a user manual for users.

Our implementation may not be an optimal one, but it indeed a prototype developed as proof of concept. All the Add-ons and other supporting python codes are available on *my GitHub page*.

7.1 Blender Setup

A user will require the following system configurations and basic setup to run all the Add-ons.

OS:

All the Add-ons are tested on Ubuntu 18.04.

System:

Add-ons can be run on any system with Blender 2.81 (or 2.82) and following python libraries must be installed.

Python Libraries:

Numpy, Scipy, Scikit-sparse, Scikit-learn

The main challenge, we face during implementation, is the compatibility of the above python libraries in the Blender environment. Here, we found two possible ways to access these libraries in the Blender.

1. Install these libraries directly into the system root. The version of the library should be compatible with the inbuilt Python version of the Blender.
2. Access these libraries from Anaconda by below steps,
    (a) Install Anaconda 3 (any version) from `https://docs.anaconda.com/anaconda/install/`.
    (b) Create a Python environment “Blender" with the version the same as Blender’s Python.

(c) Install all the libraries mentioned above. The Anaconda takes care of compatibility.

(d) For this option, uninstall these libraries from the system root; otherwise, there will be a clash between them during import.

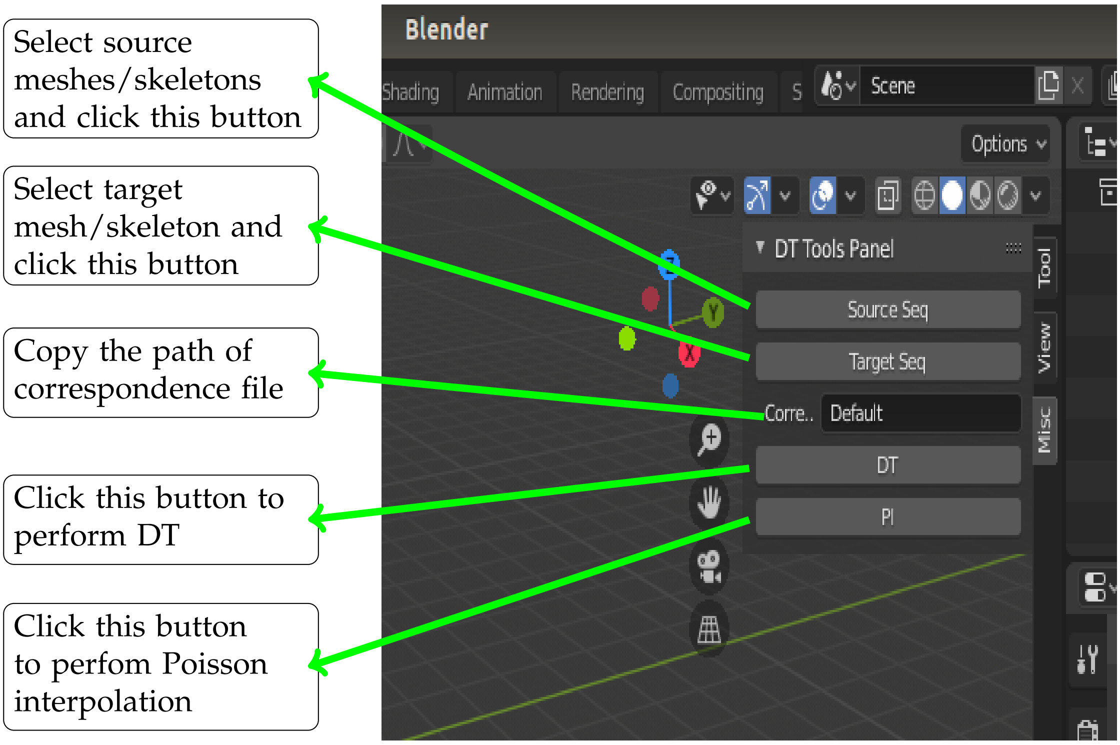
After installing python libraries, the user will have to make a few changes in all the Add-ons. User should first change the two variables; *LibPath* and *FilePath*. If the libraries are installed in the root then the *LibPath* variable must be an empty string. Otherwise, the *LibPath* must be,

LibPath='PathWhereAnacondaInstalled/anaconda3/envs/Blender/lib/python3.7/site-packages/'

where, the '*PathWhereAnacondaInstalled*' is the path where user install the Anaconda. The files generated by an Add-on are stored at *FilePath*, and it may require extensive storage space sometimes. Hence, the storage directory should be created in the hard drive. Now, everything is set to run a Blender Add-on. The toolbox is visible in the Blender UI by activating an Add-on (Edit->Preferences->add-ons->install). The user manuals for all the Add-ons are provided in the next section.

7.2 Deformation Transfer Add-on

This add-on is the implementation of the Deformation Transfer using the VG as explained in section 4.2. When the add-on is activated by the user, the tool panel shown in Figure 7.1 is visible in the Blender UI in *Misc* bar. The add-on is designed to process multiple meshes at once, so the source can be a set of meshes or skeleton poses. Note that the pose imported first in the Blender is considered as the reference pose. The user should also provide the reference target pose along with an optional pose for Poisson interpolation (for *non-similar boundary condition*). The user also has to provide the path of the correspondence file (".txt"), replacing the "Default" option. If both the source and the target meshes have a similar configuration, then the user must not change the "Default" in the correspondence box. The correspondence between two meshes can be established by a mesh registration method such as [1]. We use the implementation



Select source meshes/skeletons and click this button

Select target mesh/skeleton and click this button

Copy the path of correspondence file

Click this button to perform DT

Click this button to perfom Poisson interpolation

Figure 7.1: User interaction panel to perform Deformation Transfer in Blender 2.82.

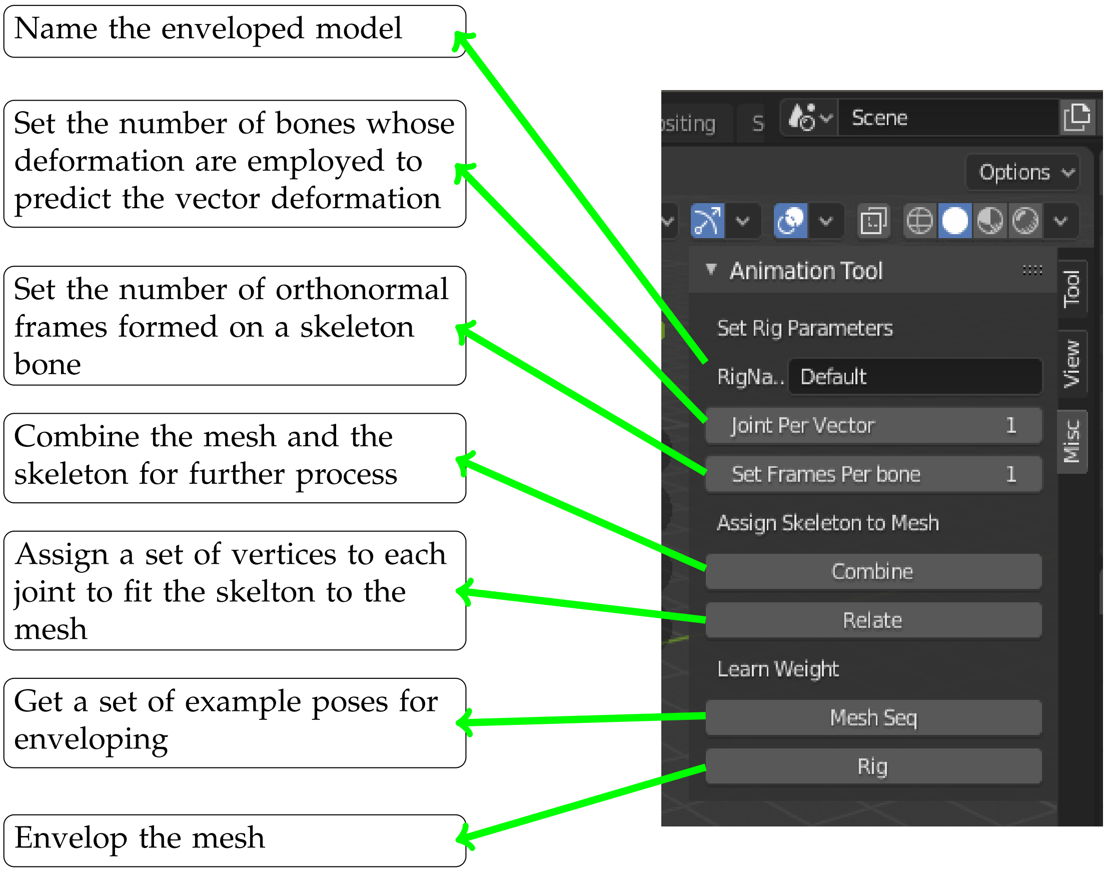
found at `https://github.com/Golevka/deformation-transfer` to find the correspondences. The output ".tricorr" file is further processed to get a correspondence file compatible with our implementation by running a python file *"Convert_tricorrs_To_oneTOone.py"* which is found at *my GitHub page*. The interactive use of this Add-on is available as a demo video at *Deformation Transfer Guide*.

7.3 Enveloping

As explained in chapter 5, a mesh is first enveloped by approximating the map using a set of example poses. The enveloped mesh is then deformed by deforming the corresponding skeleton (control structure). A user can perform the enveloping and the mesh editing processes interactively in the Blender using two separate Add-ons.

7.3.1 Enveloping Add-on

The user interface panel to perform the enveloping is shown in Figure 7.2. A user should compute the necessary setup steps as described in section 7.1. A skeleton and a set of example poses of a mesh are required to envelop the mesh model. The skeleton should consist of edges as bones and vertices as joints. Such a skeleton can be created in the Blender using its in-built tools (see the explanatory video at *Enveloping Guide*). The user should follow the following steps,


Name the enveloped model
Set the number of bones whose deformation are employed to predict the vector deformation
Set the number of orthonormal frames formed on a skeleton bone
Combine the mesh and the skeleton for further process
Assign a set of vertices to each joint to fit the skelton to the mesh
Get a set of example poses for enveloping
Envelop the mesh
siting
S
Scene
Options
Animation Tool
Tool
View
Misc
Set Rig Parameters
Assign Skeleton to Mesh
Combine
Relate
Learn Weight
Mesh Seq
Rig

RigNa.. Default

Joint Per Vector 1

Set Frames Per bone 1

Figure 7.2: User interaction panel for the enveloping in Blender 2.82.

Step-1: Name the enveloped model. The name is added to the file called *Riglist* which is stored in a directory. The path of the directory is assigned in the *FilePath* variable.

Step-2: Assign the number of bones in *Joint Per Vector* box. The deformations of the assigned number of bones are employed to predict the vector deformation.

Step-3: Assign the number of orthonormal frames formed on a skeleton bone in *Set Frames Per bone* box.

Step-4: Select both the reference mesh and the skeleton and then click on the *Combine* button to combine them in one Blender object. The user should join each skeleton joint with four vertices of a mesh by four edges (see the explanatory video at *Enveloping Guide* for guidance).

Step-5: Fit the skeleton to the mesh by clicking *Relate* button. The selection is stored, and it is further used to fit the skeleton to other example poses.

Step-6: Import a set of example poses in the Blender UI and select all of them. The skeleton is assigned automatically to all the example meshes by clicking the *Mesh seq* button using information stored in the previous step.

Step-7: Click *Rig* button to start enveloping process.

7.3.2 Editing

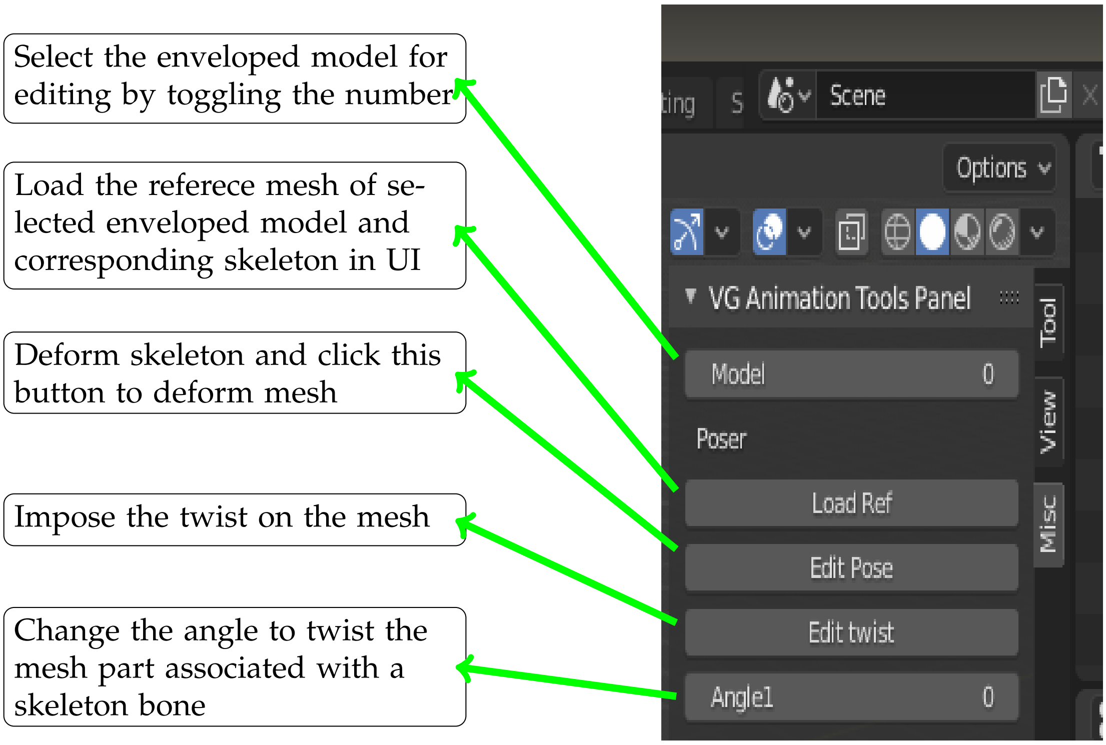
After completing the enveloping process, the user can now deform the enveloped mesh by deforming the corresponding skeleton. We have created a demo of the mesh editing process as a video at *Editing Guide*. To edit the mesh, follow the process given below,

Step- 1: Select the enveloped mesh by toggling the numbers in *Model* box. The name is printed in the terminal as the user toggle the number.

Step- 2: Load the reference mesh and the corresponding skeleton in the Blender UI by clicking the *Load Ref* button.

Step- 3: Deform the skeleton and then click on the *Edit Pose* button to deform the mesh accordingly. The twist of the skeleton bone is not accounted in this step.

Step- 4: A part of the mesh can be twisted by rotating the orthonormal frame around the associated skeleton bone. For this task, the user should first



Select the enveloped model for editing by toggling the number

Load the referece mesh of selected enveloped model and corresponding skeleton in UI

Deform skeleton and click this button to deform mesh

Impose the twist on the mesh

Change the angle to twist the mesh part associated with a skeleton bone

Figure 7.3: User interaction panel for mesh editing in Blender 2.82.

select two joints of the skeleton bone and change the twist angle value (-180° to 180°) in *Angle1* box to observe the twist on the mesh part in real-time. The Blender UI should be in the **edit mode** while performing this task. However, the twist is not imposed on the mesh yet.

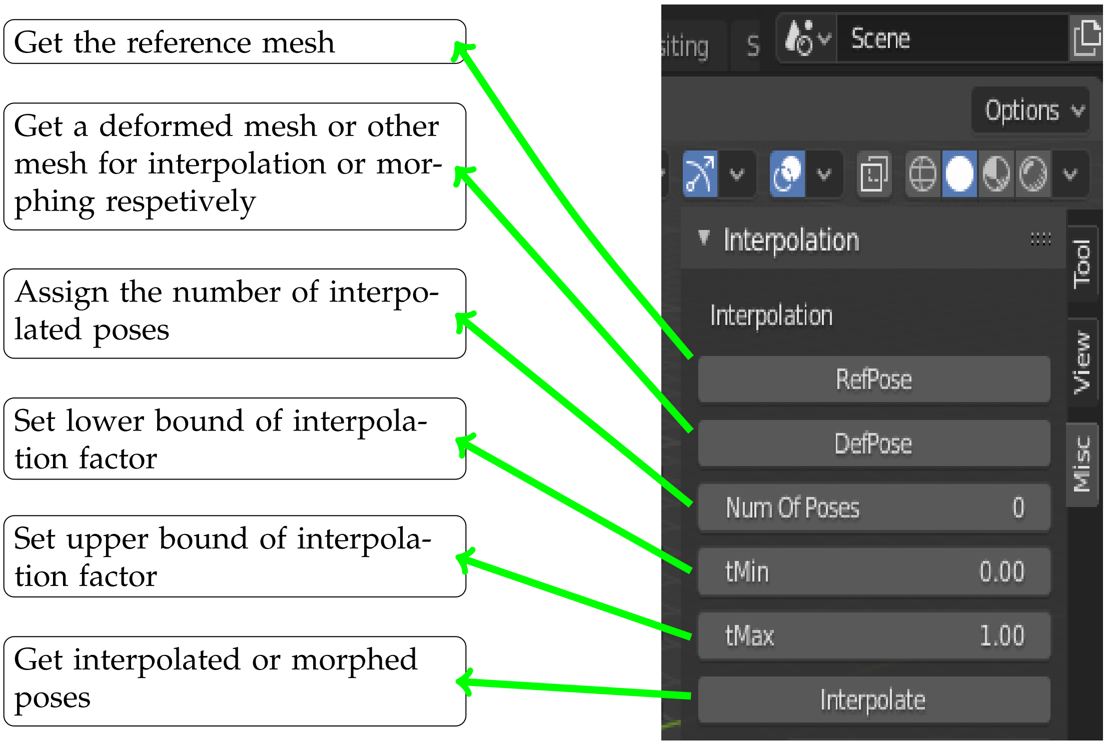
Step- 5: The user can impose the desired twist on the mesh part by clicking the *Edit twist* button. The Blender UI should be in the **object mode** while performing this task.

7.4 Interpolation and Morphing

As explained in chapter 6, we have explored two approaches for the interpolation and morphing. The user must complete the necessary setup steps as explained in section 7.1, before using the Add-ons created for these applications.

7.4.1 Basic Interpolation and Morphing

Here, we have documented the usage of the Add-on for the basic interpolation and morphing methods. The UI panel for this method is shown in Figure 7.4. Follow the steps given below to use this Add-on,



Get the reference mesh

Get a deformed mesh or other mesh for interpolation or morphing respetively

Assign the number of interpolated poses

Set lower bound of interpolation factor

Set upper bound of interpolation factor

Get interpolated or morphed poses

Num Of Poses 0

tMin 0.00

tMax 1.00

Figure 7.4: User interaction panel in Blender 2.82 for basic Interpolation and Morphing.

Step- 1: Import the reference mesh (for example, man) in the Blender UI and click the *Ref Mesh* button.

Step- 2: Import, a deformed mesh, corresponds to the reference mesh (Man) for interpolation or another mesh (Woman) for morphing. The user should then select the mesh and click *Def mesh* button. Here, we assume that the configuration of both meshes is the same.

Step- 3: Set the desired number of interpolated poses in the *Num of Poses* box.

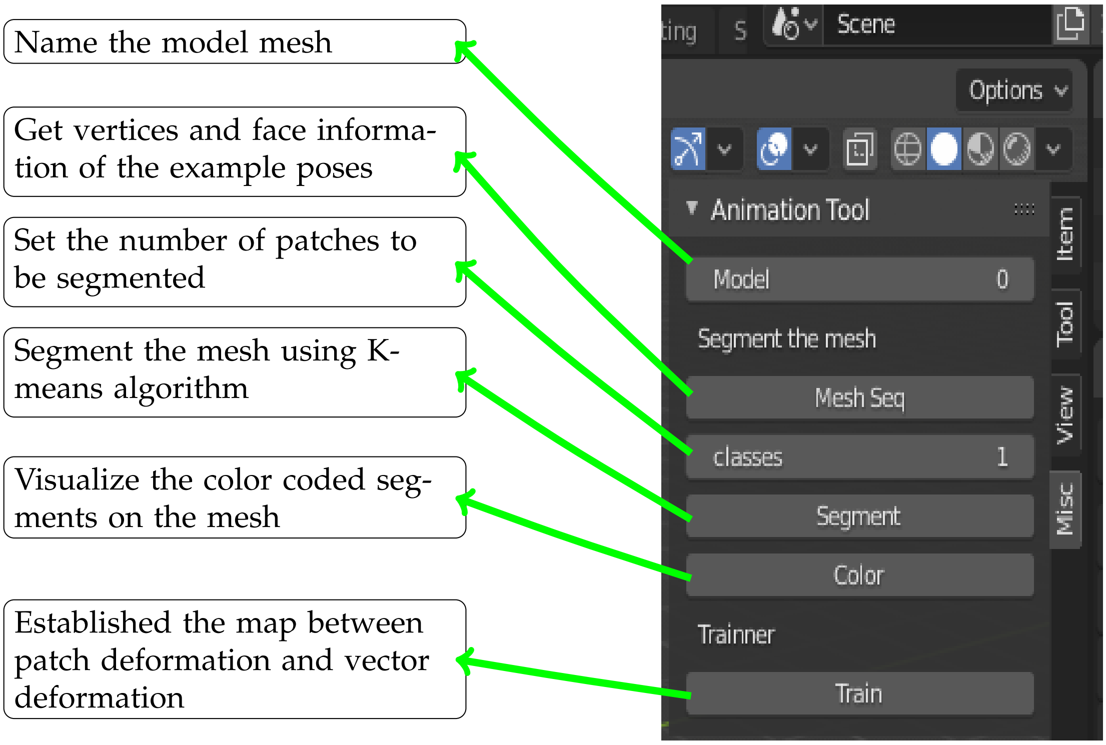
Step- 4: Set the lower bound for interpolation factor in the *tMin* box. This value should be 0 or less than 0 for interpolation or extrapolation, respectively.

Step- 5: Set the upper bound for interpolation factor in the *tMax* box. This value should be 1 or greater than 1 for interpolation or extrapolation, respectively.

Step- 6: Click the *Interpolate* button to perform interpolation or morphing.

7.4.2 Establishing Map

In section 6.4 and 6.5, we have explored another approach for real-time interpolation and deformation transfer. In this approach, a mesh is segmented into patches which serve as a low-resolution structure. We then establish a map between the patch deformation and the vector deformation using a set of example poses. The established map is used for real-time interpolation and deformation transfer for various meshes and the morphed meshes. The Blender Add-on to establish the map, and the user interaction panel is shown in Figure 7.5. The video is available here at *Interpolation Training Guide* for this process. The user manual is given below.



Name the model mesh

Get vertices and face information of the example poses

Set the number of patches to be segmented

Segment the mesh using K-means algorithm

Visualize the color coded segments on the mesh

Established the map between patch deformation and vector deformation

Figure 7.5: User interaction panel in Blender 2.82 to establish the map.

Step- 1: Give a name to the mesh in *Model* box.

Step- 2: Import example poses of the mesh from the database in Blender UI. The user should select all the poses and then click *Mesh seq* button to get their vertex and face information. Note that the first imported pose is considered as the reference mesh.

Step- 3: Enter the desired number of patches in the *classes* box. Click the *Segment* button to segment out patches using the K-means algorithm.

Step- 4: Select the reference mesh and click on the *Color* button to visualize color-coded patches on the reference mesh.

Step- 5: Finally, click on the *Train* button to establish the map between the patch deformation and the vector deformation.

7.4.3 Real-time Interpolation and Deformation Transfer

After establishing the map, the user can perform real-time interpolation and deformation transfer. The user can also perform both the operations on the morphed model. The usage of this Add-on is compiled in the video at *Interpolation Guide*. In Figure 7.6, we have shown user interaction panel for these tasks. The user guide is given below,

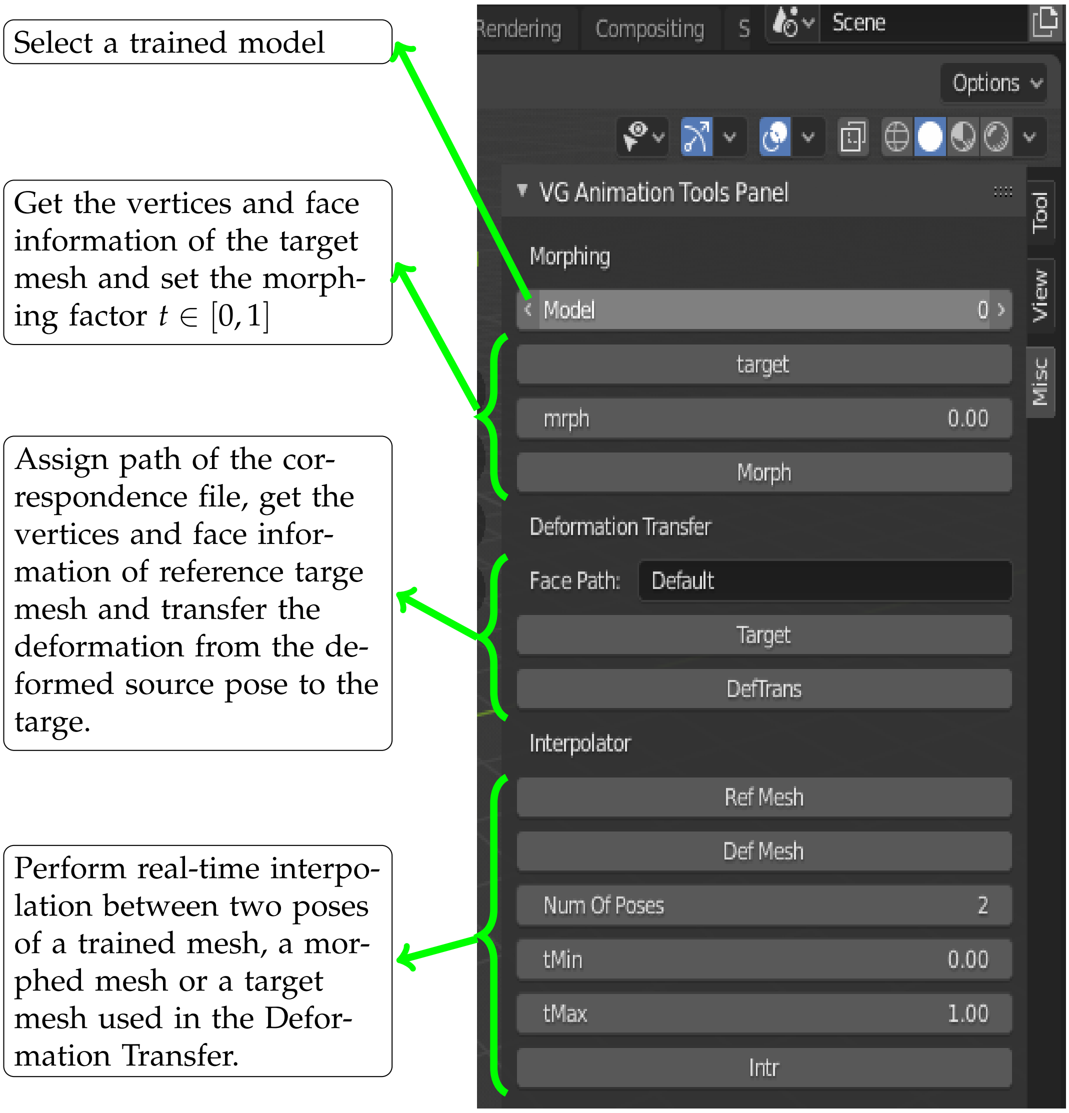
Real-time interpolation:

The user should first select the trained mesh by toggling the number in the *Model* box. The real-time interpolation can be performed between a pair of poses using the *Interpolator* section. The interpolation steps are the same as explained in section 7.4.1.

Deformation transfer:

Here, the user can also perform the Deformation Transfer using the established map. Follow the below steps.

1. Select the trained source mesh by toggling the number in the *Model* box and then import the reference target mesh in the Blender UI.



Select a trained model

Get the vertices and face information of the target mesh and set the morphing factor $t \in [0, 1]$

Assign path of the correspondence file, get the vertices and face information of reference targe mesh and transfer the deformation from the deformed source pose to the targe.

Perform real-time interpolation between two poses of a trained mesh, a morphed mesh or a target mesh used in the Deformation Transfer.

Model 0

mrph 0.00

Face Path: Default

Num Of Poses 2

tMin 0.00

tMax 1.00

Figure 7.6: User interaction panel in Blender 2.82 to perform real-time interpolation on trained models and deformation transfer on the target mesh. User can also perform both processes for a morphed mesh.

2. The user can establish the correspondence between the source and the target by following the process explained in section 7.2. The path to the correspondence file is then copied in *Face path* box.

3. The map established for the source (trained model) is transferred to the selected target reference mesh by clicking on the *Target* button.

4. Now, the user can perform Deformation Transfer by selecting deform source poses followed by clicking *DefTrans* button. The real-time interpolation on the target meshes can also be performed using the *Interpolator* section.

Interpolation and Deformation Transfer on Morphed Mesh. The user first selects the trained source mesh by toggling the number in *Model* box. The vertices and face information of the target mesh can be acquired by clicking on the *Target* button. The parameters of the established map for the trained mesh are adjusted for the target mesh according to the morphing factor assigned in *mrph* box by clicking the *Morph* button. The user can perform the Deformation Transfer and the real-time interpolation on the morphed model using the *Deformation Transfer* and *Interpolator* section of the tool panel.

CHAPTER 8

Conclusion and Future Work

This work demonstrates the universality of the Vector Graph (VG) representation and its applicability for various mesh processing tasks such as Deformation Transfer, Enveloping, Interpolation, and Morphing. We show that representing a mesh as the VG and reconstructing the mesh back from its VG representation are straightforward processes. We also show experimentally that the VG is a consistent representation by reconstructing the original mesh back from its VG with a negligible error. With just the composition of rotation and the scaling associated with vectors of the VG, it is possible to perform various mesh processing tasks efficiently compared to other state-of-the-art methods qualitatively and computationally. The proposed face selection algorithm reduces redundant computation for a mesh processing task without compromising on the deformation quality. Moreover, it is applicable to other face-based representations as well.

Deformation Transfer: In this work, we have explored two Deformation Transfer approaches using the VG representation by computing the shape deformation and the pose deformation. The shape deformation characterizes the shape variation between the source and the target meshes. The proposed Deformation Transfer method using the shape deformation, qualitatively, and quantitatively, is compared with other state-of-the-art methods. Extensive experiments are carried out on temporal sequences of human skeletons and triangular meshes to show the effectiveness of the proposed method. The Deformation Transfer using shape deformation causes artifacts on the target mesh when the deformation is large. It also fails to preserve the planarity of the quad faces. These limitations are addressed in the Deformation Transfer using the pose deformation which characterizes the pose variation in the source mesh. Here, we also bring out the differences between this approach and the other Deformation Transfer methods proposed in [1] and [2]. It is interesting to notice how the dot product property affects the process of deformation transfer and helps in preserving the shape and the geometric details of the target for a wide range of deformations. While deforming the target according to the source, the proposed method preserves the shape of the target. The results of the experiments endorse the claim that the imposed planarity constraint improves the planarity of the quad face during the reconstruction process.

In guided Deformation Transfer, interpolating target poses using the Poisson interpolation, adjusts the temporal position of the target poses and motion trajectory of the target. The experiments carried out on temporal sequences of human skeletons with triangular, and hybrid meshes show the effectiveness of guided Deformation Transfer.

Enveloping: Using the VG, we have proposed a data-driven enveloping method with the skeleton as a control structure. In this work, the mesh is represented by the VG representation to compute its deformation. As noted in the section 5.2, both the skeleton and the mesh deformations form two groups. We establish a injective map between both the groups using available poses of a mesh in the database. After establishing the map, we formulate the relationship between the skeleton deformation and the mesh deformation. The experiments shown in the section 5.5 suggest that the proposed enveloping method leads to smooth deformation on various enveloped meshes similar to non-linear methods and computationally as efficient as the Linear Blend Skinning.

Interpolation and Morphing: We have explored the interpolation and the morphing using the VG representation. We show that the interpolation and the morphing can be performed on various meshes adopting the traditional approaches. However, the traditional approaches are computationally inefficient due to element-wise operations. To address this issue, the mesh is first segmented into a set of patches using the K-means clustering . The mesh segmented into patches serves as a low resolution structure. A patch deformation is then computed for each patch. A map is established between the patch deformation and vector deformation. The parameters of the map are computed using examples poses available in the database. We also show that the established map for a mesh can be used for interpolation and deformation transfer. Both interpolation and deformation transfer can also be performed on the morphed mesh by modifying the established map. We have carried out extensive experiments to show the effectiveness of the proposed method on 23 different meshes. We have compared the proposed interpolation, deformation transfer method with other methods in terms of quality and computational complexity as shown in section 6.7. The experiments show that the proposed method performs qualitatively similar to and computationally efficient than other methods.

Future Work

To envelop the geometric models such as clothes, face mesh, etc., it is difficult to arrive at the skeleton-like control structure. We want to work towards developing an appropriate control structure for such types of meshes to expand the limited boundaries of the proposed enveloping method. In the case of facial expressions, the deformation heavily depends on the muscle movements, and it needs to be dealt with differently. Introducing the non-linearity during the map establishment process using Deep Learning approaches can clear up the discontinuation.

There are various limitations of the proposed interpolation method. It causes artifacts on the interpolated poses when the deformation is large (>180°). This problem can be solved by adopting the approach proposed in [47] to compute the patch deformation. Though it is a computationally demanding approach, it does not affect the real-time performance of the proposed interpolation due to the low dimensions of the patch deformation manifold. The proposed interpolation method also fails to interpolate the transition from one facial expression to another because the facial deformation can not be modelled by segmented patches accurately. This problem can be handled by replacing the patch-based structure with other suitable structures, or it can be solved by introducing a deep learning approach to established the map. We want to expand this work by keeping these issues in mind.

We would also like to explore Geometric Algebras [77, 78, 79, 80, 81] for the Vector Graph representation because the scaling and the rotation are well defined there. We have also neglected the vector translational which is import for the motion data. Inserting the translation to the deformation composition can expand the reach of the VG toward motion processing applications. It can also open up the possibility of the use of the VG in computer vision applications [82, 83, 84]. Moreover, the Deep Learning for mesh deformation [85, 86] can be a handy solution for the enveloping and interpolation and other applications. As a part of the future work, we would also like to explore other mesh editing tasks such as mesh registration [87, 88, 67], style transfer [55, 56, 54] and motion editing [60, 63, 64] using the VG representation.

References

[1] Robert W. Sumner and Jovan Popoviundefined. Deformation transfer for triangle meshes. *ACM Trans. Graph.*, 23(3):399–405, August 2004.

[2] A. El Rahman Shabayek, D. Aouada, A. Saint, and B. Ottersten. Deformation transfer of 3d human shapes and poses on manifolds. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 220–224, Sep. 2017.

[3] Prashant M Domadiya, Dr.Pratik Shah, and Suman Mitra. Guided deformation transfer. In *European Conference on Visual Media Production*, CVMP '19, New York, NY, USA, 2019. Association for Computing Machinery.

[4] Gerard Pons-Moll, Javier Romero, Naureen Mahmood, and Michael J. Black. Dyna: A model of dynamic human shape in motion. *ACM Transactions on Graphics, (Proc. SIGGRAPH)*, 34(4):120:1–120:14, August 2015.

[5] Ilya Baran, Daniel Vlasic, Eitan Grinspun, and Jovan Popović. Semantic deformation transfer. *ACM Trans. Graph.*, 28(3):36:1–36:6, July 2009.

[6] J. P. Lewis, Matt Cordner, and Nickson Fong. Pose space deformation: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[7] Stephen W. Bailey, Dave Otte, Paul Dilorenzo, and James F. O'Brien. Fast and deep deformation approximations. *ACM Trans. Graph.*, 37(4):119:1–119:12, July 2018.

[8] Alec Jacobson, Ilya Baran, Ladislav Kavan, Jovan Popović, and Olga Sorkine. Fast automatic skinning transformations. *ACM Trans. Graph.*, 31(4):77:1–77:10, July 2012.

[9] Robert Y. Wang, Kari Pulli, and Jovan Popović. Real-time enveloping with rotational regression. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[10] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.*, 24(3):479–487, July 2005.

[11] Dong Xu, Hongxin Zhang, Qing Wang, and Hujun Bao. Poisson shape interpolation. In *Proceedings of the 2005 ACM Symposium on Solid and Physical Modeling*, SPM '05, pages 267–274, New York, NY, USA, 2005. ACM.

[12] Sumukh Bansal and Aditya Tatu. Lie bodies based 3d shape morphing and interpolation. In *Proceedings of the 15th ACM SIGGRAPH European Conference on Visual Media Production*, CVMP '18, pages 5:1–5:10, New York, NY, USA, 2018. ACM.

[13] Olga Sorkine. Differential representations for mesh processing. *Computer Graphics Forum*, 25(4):789–807, 2006.

[14] Oren Freifeld and Michael J. Black. *Lie Bodies: A Manifold Representation of 3D Human Shape*, pages 1–14. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[15] Scott Kircher and Michael Garland. Free-form motion processing. *ACM Trans. Graph.*, 27(2):12:1–12:13, May 2008.

[16] Seung-Yeob Baek, Jeonghun Lim, and Kunwoo Lee. Isometric shape interpolation. *Comput. Graph.*, 46(C):257–263, February 2015.

[17] A. Sheffer and V. Kraevoy. Pyramid coordinates for morphing and deformation. In *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004.*, pages 68–75, Sep. 2004.

[18] Marc Alexa. Differential coordinates for local mesh morphing and deformation. *The Visual Computer*, 19(2):105–114, May 2003.

[19] Doug L. James and Christopher D. Twigg. Skinning mesh animations. *ACM Trans. Graph.*, 24(3):399–407, July 2005.

[20] Stefan Frohlich and Mario Botsch. Example-driven deformations based on discrete shells. *Computer Graphics Forum*, 30(8):2246–2257, 2011.

[21] Kun Zhou, Weiwei Xu, Yiying Tong, and Mathieu Desbrun. Deformation transfer to multi-component objects. *Computer Graphics Forum*, 2010.

[22] Martin Kilian, Niloy J. Mitra, and Helmut Pottmann. Geometric modeling in shape space. *ACM Trans. Graph.*, 26(3), July 2007.

[23] Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. Scape: Shape completion and animation of people. *ACM Trans. Graph.*, 24(3):408–416, July 2005.

[24] Y. Zhao, B. Pan, and Q. Peng. Robust deformation transfer via dual domain. In *2011 12th International Conference on Computer-Aided Design and Computer Graphics*, pages 302–305, Sept 2011.

[25] O. K. C. Au, C. L. Tai, L. Liu, and H. Fu. Dual laplacian editing for meshes. *IEEE Transactions on Visualization and Computer Graphics*, 12(3):386–395, May 2006.

[26] Mirela Ben-Chen, Ofir Weber, and Craig Gotsman. Spatial deformation transfer. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA, pages 67–74, New York, NY, USA, 2009. Association for Computing Machinery.

[27] Chin-chia Tung, Tsung-Hua Li, Hong-Shiang Lin, and Ming Ouhyoung. Cage-based deformation transfer using mass spring system. In *ACM SIGGRAPH*, SIGGRAPH, New York, NY, USA, 2014. Association for Computing Machinery.

[28] Yaron Lipman, David Levin, and Daniel Cohen-Or. Green coordinates. *ACM Trans. Graph.*, 27(3):78:1–78:10, August 2008.

[29] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics, (Proc. SIGGRAPH)*, 2(3):644–651, August 2004.

[30] Patrick Pérez, Michel Gangnet, and Andrew Blake. Poisson image editing. *ACM Trans. Graph.*, 22(3):313–318, July 2003.

[31] Xiaohuan Corina Wang and Cary Phillips. Multi-weight enveloping: Least-squares approximation techniques for skin animation. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '02, pages 129–138, New York, NY, USA, 2002. ACM.

[32] Alex Mohr and Michael Gleicher. Building efficient, accurate character skins from examples. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 562–568, New York, NY, USA, 2003. ACM.

[33] Ladislav Kavan and Jiří Žára. Spherical blend skinning: A real-time deformation of articulated models. In *Proceedings of the 2005 Symposium on Interactive 3D Graphics and Games*, I3D '05, pages 9–16, New York, NY, USA, 2005. ACM.

[34] Ladislav Kavan, Steven Collins, Jiří Žára, and Carol O'Sullivan. Skinning with dual quaternions. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 39–46, New York, NY, USA, 2007. ACM.

[35] Tomohiko Mukai and Shigeru Kuriyama. Efficient dynamic skinning with low-rank helper bone controllers. *ACM Trans. Graph.*, 35(4):36:1–36:11, July 2016.

[36] Olga Sorkine and Marc Alexa. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, pages 109–116, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[37] Isaac Chao, Ulrich Pinkall, Patrick Sanan, and Peter Schröder. A simple geometric model for elastic deformations. *ACM Trans. Graph.*, 29(4):38:1–38:6, July 2010.

[38] Ligang Liu, Lei Zhang, Yin Xu, Craig Gotsman, and Steven J. Gortler. A local/global approach to mesh parameterization. In *Proceedings of the Symposium on Geometry Processing*, SGP '08, pages 1495–1504, Aire-la-Ville, Switzerland, Switzerland, 2008. Eurographics Association.

[39] Yizong Cheng. Mean shift, mode seeking, and clustering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(8):790–799, Aug 1995.

[40] D. Comaniciu and P. Meer. Mean shift: a robust approach toward feature space analysis. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(5):603–619, May 2002.

[41] Nils Hasler, Thorsten Thormählen, Bodo Rosenhahn, and Hans-Peter Seidel. Learning skeletons for shape and pose. In *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D, pages 23–30, New York, NY, USA, 2010. ACM.

[42] Binh Huy Le and Zhigang Deng. Smooth skinning decomposition with rigid bones. *ACM Trans. Graph.*, 31(6):199:1–199:10, November 2012.

[43] Martin Ralph R. Yan Han-Bing, Hu Shi-Min. 3d morphing using strain field interpolation. *Journal of Computer Science and Technology*, 22(1):147–155, January 2007.

[44] B. Heeren, M. Rumpf, M. Wardetzky, and B. Wirth. Time-discrete geodesics in the space of shells. *Computer Graphics Forum*, 31(5):1755–1764, 2012.

[45] Robert W. Sumner, Matthias Zwicker, Craig Gotsman, and Jovan Popović. Mesh-based inverse kinematics. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, pages 488–495, New York, NY, USA, 2005. ACM.

[46] Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 157–164, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[47] Lin Gao, Yu-Kun Lai, Dun Liang, Shu-Yu Chen, and Shihong Xia. Efficient and flexible deformation representation for data-driven surface modeling. *ACM Trans. Graph.*, 35(5):158:1–158:17, July 2016.

[48] Christoph Von-Tycowicz, Christian Schulz, Hans-Peter Seidel, and Klaus Hildebrandt. Real-time nonlinear shape interpolation. *ACM Trans. Graph.*, 34(3):34:1–34:10, May 2015.

[49] Michael S. Floater. Mean value coordinates. *Computer Aided Geometric Design*, 20(1):19 – 27, 2003.

[50] Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. In *ACM SIGGRAPH 2005 Papers*, SIGGRAPH '05, page 561–566, New York, NY, USA, 2005. Association for Computing Machinery.

[51] Ofir Weber, Mirela Ben-Chen, and Craig Gotsman. Complex barycentric coordinates with applications to planar shape deformation. *Comput. Graph. Forum*, 28:587–597, 04 2009.

[52] T. Winkler, J. Drieseberg, M. Alexa, and K. Hormann. Multi-scale geometry interpolation. *Computer Graphics Forum*, 29(2):309–318, 2010.

[53] H. Chu and T. Lee. Multiresolution mean shift clustering algorithm for shape interpolation. *IEEE Transactions on Visualization and Computer Graphics*, 15(5):853–866, Sep. 2009.

[54] M. Ersin Yumer and Niloy J. Mitra. Spectral style transfer for human motion between independent actions. *ACM Trans. Graph.*, 35(4):137:1–137:8, July 2016.

[55] Eugene Hsu, Kari Pulli, and Jovan Popović. Style translation for human motion. *ACM Trans. Graph.*, 24(3):1082–1089, July 2005.

[56] Shihong Xia, Congyi Wang, Jinxiang Chai, and Jessica Hodgins. Realtime style transfer for unlabeled heterogeneous human motion. *ACM Trans. Graph.*, 34(4):119:1–119:10, July 2015.

[57] Michael Gleicher. Motion path editing. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, pages 195–202, New York, NY, USA, 2001. ACM.

[58] Loïc Ciccone, Martin Guay, Maurizio Nitti, and Robert W. Sumner. Authoring motion cycles. In *Proceedings of the ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '17, pages 8:1–8:9, New York, NY, USA, 2017. ACM.

[59] Matthew Brand and Aaron Hertzmann. Style machines. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 183–192, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[60] Byungkuk Choi, Roger Blanco i Ribera, J. P. Lewis, Yeongho Seol, Seokpyo Hong, Haegwang Eom, Sunjin Jung, and Junyong Noh. Sketchimo: Sketch-based motion editing for articulated characters. *ACM Trans. Graph.*, 35(4):146:1–146:12, July 2016.

[61] I. J. Schoenberg. *Cardinal Spline Interpolation*. Society for Industrial and Applied Mathematics, 1973.

[62] Andrew Witkin and Zoran Popovic. Motion warping. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 105–108, New York, NY, USA, 1995. ACM.

[63] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '99, pages 39–48, New York, NY, USA, 1999. ACM Press/Addison-Wesley Publishing Co.

[64] S. Lee, G. Wolberg, and S. Y. Shin. Scattered data interpolation with multilevel b-splines. *IEEE Transactions on Visualization and Computer Graphics*, 3(3):228–244, Jul 1997.

[65] Lucas Kovar, Michael Gleicher, and Frédéric Pighin. Motion graphs. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '02, pages 473–482, New York, NY, USA, 2002. ACM.

[66] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 33–42, New York, NY, USA, 1998. ACM.

[67] Oliver van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. A survey on shape correspondence. *Computer Graphics Forum*, 30(6):1681–1707, 2011.

[68] I. Theodorakopoulos, D. Kastaniotis, G. Economou, and S. Fotopoulos. Pose-based human action recognition via sparse representation in dissimilarity space. *Visual Communication and Image Representation*, 25:12–23, 2014.

[69] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3):77:1–77:10, July 2009.

[70] F. Bogo, J. Romero, M. Loper, and M. J. Black. Faust: Dataset and evaluation for 3d mesh registration. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3794–3801, June 2014.

[71] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, New York, NY, USA, 2007. ACM.

[72] P. Domadiya, P. Shah, and S. K. Mitra. Vector graph representation for deformation transfer using poisson interpolation. In *2018 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 876–884, March 2018.

[73] D. W. Eggert, A. Lorusso, and R. B. Fisher. Estimating 3-d rigid body transformations: A comparison of four major algorithms. *Mach. Vision Appl.*, 9(5-6):272–290, March 1997.

[74] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[75] Utah. Utah 3d model repository.

[76] Emanuele Rodolà, Samuel Rota Bulò, Thomas Windheuser, Matthias Vestner, and Daniel Cremers. Dense non-rigid shape correspondence using random forests. In *Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '14, pages 4177–4184, Washington, DC, USA, 2014. IEEE Computer Society.

[77] John Vince. *Geometric Algebra for Computer Graphics*. Springer London, 2008.

[78] H. Zhang, C. Zhu, Q. Peng, and J. X. Chen. Using geometric algebra for 3d linear transformations. *Computing in Science Engineering*, 8(3):68–75, 2006.

[79] RICHARD WAREHAM and Joan Lasenby. Rigid-body pose and position interpolation using geometric algebra. 01 2004.

[80] Dietmar Hildenbrand, Christian Perwass, Leo Dorst, and Daniel Fontijne. Geometric algebra and its application to computer graphics. In *Eurographics*, 2004.

[81] D. Fontijne and L. Dorst. Modeling 3d euclidean geometry. *IEEE Computer Graphics and Applications*, 23(2):68–78, 2003.

[82] Z. Huang, C. Wan, T. Probst, and L. Van Gool. Deep learning on lie groups for skeleton-based action recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1243–1252, 2017.

[83] Olivier Faugeras. Three-dimensional computer vision: a geometric viewpoint. *MIT press*, 01 1993.

[84] R. Liu, C. Xu, T. Zhang, W. Zhao, Z. Cui, and J. Yang. Si-gcn: Structure-induced graph convolution network for skeleton-based action recognition. In *2019 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2019.

[85] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. Geometric deep learning: Going beyond euclidean data. *IEEE Signal Processing Magazine*, 34(4):18–42, 2017.

[86] Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G. Kim, and Yaron Lipman. Convolutional neural networks on surfaces via seamless toric covers. *ACM Trans. Graph.*, 36(4), July 2017.

[87] D. Hirshberg, M. Loper, E. Rachlin, and M.J. Black. Coregistration: Simultaneous alignment and modeling of articulated 3D shape. In *European Conf. on Computer Vision (ECCV)*, LNCS 7577, Part IV, pages 242–255. Springer-Verlag, October 2012.

[88] Haggai Maron, Nadav Dym, Itay Kezurer, Shahar Kovalsky, and Yaron Lipman. Point registration via efficient convex relaxation. *ACM Trans. Graph.*, 35(4), July 2016.

CHAPTER A

Appendices

A.1 Derivation of Reconstruction Optimization Equation

Rewriting equation (4.29),

$$\underset{\hat{T}_d}{\operatorname{argmin}} \quad w_1 \|\hat{B}\hat{T}_d - b_d\|_2^2 + w_2 \|\hat{E}_d \hat{T}_d - e_d\|_2^2 \tag{A.1}$$

where, the matrix $\hat{E}_d$ is in form of, given by,

$$E_d = \begin{bmatrix} M'_{d1} & 0 & 0 & 0 & \dots & 0 \\ 0 & M'_{d2} & 0 & 0 & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & M'_{dn} & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots\ddots & \vdots & \end{bmatrix}_{n \times 3n} \begin{bmatrix} \bar{t}_{d1} \\ \bar{t}_{d2} \\ \vdots \\ \bar{t}_{dn} \end{bmatrix} = M_d \bar{T}_d = M_d B \tilde{T}_d \tag{A.2}$$

where, n is number of vectors. Taking differentiation of above equation with respect to $\hat{T}_d$ and equate with zero,

$$2w_i \hat{B}'(\hat{B}\hat{T}_d - b_d) + 2w_2 \hat{E}'_d(\hat{E}'_d \hat{T}_d - e_d) = 0 \tag{A.3}$$

Above equation can be rearranged as,

$$(w_1 \hat{B}'\hat{B} + w_2 \hat{E}'_d \hat{E}_d)\hat{T}_d = w_1 \hat{B}' b_d + w_2 \hat{E}'_d e_d. \tag{A.4}$$

A.2 Effect of Boundary conditions on Poisson interpolation

For simplicity, let's consider an available action sequence $\{U_1, U_2, U_3, U_4\}$ and sequence $\{V_1, V_2, V_3, V_4\}$ to be estimated. Here V_1 and V_4 are boundary poses which are available. Our aim is to derive Poisson equation in form of boundary poses and gradient field. Poisson equation for V_2 and V_3 is represented as,

$$2V_2 - V_1 - V_3 = 2U_2 - U_1 - U_3 \tag{A.5}$$

$$2V_3 - V_2 - V_4 = 2U_3 - U_2 - U_4 \tag{A.6}$$

Above equations can be rearrange to,

$$V_2 = U_2 - \frac{1}{2}(U_1 + U_3) + \frac{1}{2}(V_1 + V_3) \tag{A.7}$$

$$V_3 = U_3 - \frac{1}{2}(U_2 + U_4) + \frac{1}{2}(V_2 + V_4) \tag{A.8}$$

Both equations depend upon each other, so one can simplify these equations as,

$$V_2 = U_2 - \frac{2}{3}U_1 - \frac{1}{3}U_4 + \frac{2}{3}V_1 + \frac{1}{3}V_4 \tag{A.9}$$

$$V_3 = U_3 - \frac{1}{3}U_1 - \frac{2}{3}U_4 + \frac{1}{3}V_1 + \frac{2}{3}V_4 \tag{A.10}$$

From above equation, we can generalize relation of estimated pose with boundary conditions. Let's consider the available sequence $\{U_1, U_2, \ldots, U_m\}$ and sequence $\{V_1, V_2, \ldots, V_m\}$ to be estimated. The generalization of above equations can be written as,

$$
\begin{aligned}
V_2 &= U_2 - \frac{m-2}{m-1}U_1 - \frac{1}{m-1}U_m + \frac{m-2}{m-1}V_1 + \frac{1}{m-1}V_m \\
V_3 &= U_3 - \frac{m-3}{m-1}U_1 - \frac{2}{m-1}U_m + \frac{m-3}{m-1}V_1 + \frac{2}{m-1}V_m \\
&\vdots \\
V_i &= U_i - \frac{m-i}{m-1}U_1 - \frac{i-1}{m-1}U_m + \frac{m-i}{m-1}V_1 + \frac{i-1}{m-1}V_m \\
&\vdots \\
V_{m-1} &= U_{m-1} - \frac{1}{m-1}U_1 - \frac{m-2}{m-1}U_m + \frac{1}{m-1}V_1 + \frac{m-2}{m-1}V_m
\end{aligned}
$$

A.3 Poisson interpolation and Face Planarity

Let's consider the quad faces $u_i(i = 1,2\ldots,m)$ of available meshes $\{U_1, U_2, \ldots, U_m\}$ and $v_i(i = 1,2\ldots,m)$ of estimated sequence $\{V_1, V_2, \ldots, V_m\}$. From above relation, we can write relation among quad faces as,

$$
v_i = u_i - \frac{m-i}{m-1}u_1 - \frac{i-1}{m-1}u_m + \frac{m-i}{m-1}v_1 + \frac{i-1}{m-1}v_m. \tag{A.11}
$$

In our case, $U_1 = V_1 + T$ so as $u_1 = v_1 + T$ as the result above equation reduces to,

$$
v_i = u_i + \frac{i-1}{m-1}(v_m - u_m) + \frac{m-i}{m-1}T. \tag{A.12}
$$

Above equation indicate that Poisson interpolation has very little effect on planarity of re-estimated face for two reasons: First, as $i \to 1$ effect of the difference $(v_m - u_m)$ is very less. Second, as $i \to m0$, effect of the difference $(v_m - u_m)$ increases but at the same time, the shape similarity between u_i and u_m also increases. Translation T translates entire pose so it doesn't affect the planarity of faces.

A.4 J_i is a rotation matrix

Let F is a matrix which contains three orthonormal vectors defined similarly as in equation (3.12). The matrix F cab be written as,

$$F = [\hat{v}, b, N] = \begin{bmatrix} \hat{v}_x & b_x & N_x \\ \hat{v}_y & b_y & N_y \\ \hat{v}_z & b_z & N_z \end{bmatrix} = \begin{bmatrix} \hat{v}_x & N_y\hat{v}_z - N_z\hat{v}_y & N_x \\ \hat{v}_y & N_z\hat{v}_x - N_x\hat{v}_z & N_y \\ \hat{v}_z & N_x\hat{v}_y - N_y\hat{v}_x & N_z \end{bmatrix} \tag{A.13}$$

where N is a normal vector perpendicular to vector $\hat{v}$ and $b = (N \times \hat{v})/\|N \times \hat{v}\|_2 = N \times \hat{v}$ is a unit binormal vector. The matrix $J_i(= F'_{di}F_{ri})$ defined in equation (3.13) is computed by multiplying two such matrices. We must first prove that these (F'_{di} and F_{ri}) are rotation matrices to prove J_i as a rotation matrix. If a matrix is the rotation matrix then it must be orthogonal and its determinant must be $+1$. The matrix F is an orthogonal ($F'F = I_3$) matrix because it contains three orthonormal vectors. The determinant of $F'F$ is,

$$det(F'F) = det(F')det(F) = (det(F))^2 = 1. \tag{A.14}$$

The determinant of the matrix F is,

$$\begin{aligned} det(F) =& \hat{v}_x[N_z(N_z\hat{v}_x - N_x\hat{v}_z) - N_y(N_x\hat{v}_y - N_y\hat{v}_x)] \\ & - (N_y\hat{v}_z - N_z\hat{v}_y)[N_z\hat{v}_y - N_y\hat{v}_z] \\ & + N_x[\hat{v}_y(N_x\hat{v}_y - N_y\hat{v}_x) - \hat{v}_z(N_z\hat{v}_x - N_x\hat{v}_z)]. \end{aligned}$$

Further simplifying the above equation, we get,

$$det(F) = (N_z\hat{v}_x - N_x\hat{v}_z)^2 + (N_y\hat{v}_x - N_x\hat{v}_y)^2 + (N_y\hat{v}_z - N_z\hat{v}_y)^2 \geq 0.$$

Since the determinant of matrix F is positive, the equation (A.14) can be further simplified to,

$$det(F) = +1. \tag{A.15}$$

Hence, we can conclude that the matrix $F \in SO(3)$ is a rotation matrix. Both matrices F_{di} and F_{ri} are also defined the same as matrix F, therefore both are rotation matrices. The matrix $J_i = F_{di}F'_{ri}$ also satisfies both the conditions,

$$J'_i J_i = F_{ri}F'_{di}F_{di}F'_{ri} = F_{ri}F'_{ri} = I_3 \tag{A.16}$$

$$det(J_i) = det(F_{di}F'_{ri}) = det(F_{di})det(F'_{ri}) = det(F_{di})det(F_{ri}) = +1. \tag{A.17}$$

Hence, the matrix J_i is a rotation matrix. The skeleton bone deformation matrix R_j is also defined as a multiplication of orthonormal frames H_{rj} and H_{dj}. These frames are also formed similarly as F. Hence, the matrix R_j is also a rotation matrix.

A.5 Simplification of Equation (5.10)

Equation (5.10) can be rearranged as,

$$\underset{W_{ji}}{\text{argmin}} \sum_{k=1}^{p} Tr\{(R_j^k W_{ji} - J_i^k)(R_j^k W_{ji} - J_i^k)'\}. \tag{A.18}$$

Since the trace is a linear transformation, above equation is rewritten as,

$$\underset{W_{ji}}{\text{argmin}} \sum_{k=1}^{p} \{Tr(2I_3) - Tr(R_j^k W_{ji}(J_i^k)') - Tr((R_j^k W_{ji}(J_i^k)')')\}. \tag{A.19}$$

The traces of a matrix and its transpose are the same so, above equation is further simplified to,

$$\underset{W_{ji}}{\text{argmin}} \sum_{k=1}^{p} \{Tr(2I_3) - 2Tr(R_j^k W_{ji}(J_i^k)')\}. \tag{A.20}$$

The first term doesn't contain W_{ji} so it doesn't play any role in minimization. In order to minimize above cost function, we need to maximize second term. Hence, above minimization problem can be turned into maximization as,

$$\underset{W_{ji}}{\operatorname{argmax}} \sum_{k=1}^{p} Tr(R_j^k W_{ji} (J_i^k)'). \tag{A.21}$$

The trace is linear transformation and it remains unchanged with circular shift of multiplication of matrices. So, above equation can be rewritten as,

$$\underset{W_{ji}}{\operatorname{argmax}}\ Tr(W_{ji}(\mathcal{J}_i)'\mathcal{R}_j) \tag{A.22}$$

where, $(J_i)'\mathcal{R}_j = \sum_{k=1}^{p}(J_i^k)'R_j^k$. The matrices $\mathcal{R}_j = [(R_j^1)',\ (R_j^2)', \dots\ \dots, (R_{jp}^p)']'$ and $\mathcal{J}_i = [(J_i^1)',\ (J_i^2)', \dots,\ (J_i^p)']'$.

A.6 Formation of Matrix W

Let's consider that the deformation of the vector i is approximated by the deformation of skeleton bone z_i. The i^{th} column of the matrix W is shown in equation (A.23).

$$W = \begin{array}{c} \begin{array}{ccc} 1 & i & n \end{array} \\ \left(\begin{array}{ccc} \vdots & \vdots & \vdots \\ \vdots & 0 & \vdots \\ \vdots & g_{z_i i} H'_{r z_i} W_{z_i i} \bar{v}_{ri} & \vdots \\ \vdots & 0 & \vdots \\ \vdots & \vdots & \vdots \end{array}\right) \end{array} \begin{array}{l} \\ \\ \\ 3z_i + 1 : 3z_i + 3 \\ \\ \\ \end{array} \tag{A.23}$$

Similarly, the other columns of matrix W are filled up based on the bone-vector relationship sets $z_i,\ i \in \{1, \dots, n\}$.

A.7 Formation of Matrix K

For example, the deformation of the vector i is approximated by the deformation of skeleton bone z_i. Hence, entries in column i of the matrix K are,

$$K = \begin{array}{cc} \begin{array}{ccc} 1 & i & n \end{array} & \\ \begin{pmatrix} \vdots & \vdots & \vdots \\ \vdots & 0 & \vdots \\ \vdots & (s_{z_i i} H_{z_i i} \bar{v}_{ri}) & \vdots \\ \vdots & 0 & \vdots \\ \vdots & \vdots & \vdots \end{pmatrix} & \begin{array}{c} \\ \\ 3z_i + 1 : 3z_i + 3 \\ \\ \\ \end{array} \end{array} . \tag{A.24}$$

Similarly, we can fill the other columns of matrix K. The matrices K_u and K_x can also be formed similar to the matrix K.