FAST REALISTIC SKINNING FOR ANIMATING HIGHLY DEFORMABLE OBJECTS

Η
ΜΕΤΑΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ ΕΞΕΙΔΙΚΕΥΣΗΣ

Υποβάλλεται στην

ορισθείσα από την Γενική Συνέλευση Ειδικής Σύνθεσης
του Τμήματος Πληροφορικής
Εξεταστική Επιτροπή

από τον

Γεώργιο Αντωνόπουλο

ως μέρος των Υποχρεώσεων

για τη λήψη

του

ΜΕΤΑΠΤΥΧΙΑΚΟΥ ΔΙΠΛΩΜΑΤΟΣ ΣΤΗΝ ΠΛΗΡΟΦΟΡΙΚΗ

ΜΕ ΕΞΕΙΔΙΚΕΥΣΗ ΣΤΟ ΛΟΓΙΣΜΙΚΟ

Νοέμβριος 2010

DEDICATION

To Aristoula and to my family

ACKNOWLEDGEMENTS

First of all I want to thank my supervisor Professor Ioannis Fudos for his trust, tolerance, guidance and help, both in academic and personal level, over all these years of our cooperation.

I also want to thank Ph.D candidate Andreas A. Vasilakis for his insight, his time, his papers and above all his friendship.

I would also like to thank Ph.D Vasiliki Stamati for her help, guidance and soothing presence.

Finally, even though no words can contain my gratitude, I want to thank Dimitra Alexiou for setting my soul at ease and putting my train back on the rails.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ALGORITHMS

ABSTRACT

Antonopoulos, Georgios, N.
MSc, Computer Science Department, University of Ioannina, Greece. November, 2010.
Fast Realistic Skinning For Animating Highly Deformable Objects.
Thesis Supervisor: Ioannis Foudos

In 3D animation, key-frame compression is essential for the efficient storing and processing of the animation sequence. Compression is usually performed by producing an approximation of the animation. In the case of animating articulated objects, there exists an abundance of methods for skinning the object by using the bones of the model to establish bone-vertex influences, determine the movement of vertices as a function of the movements of the bones and achieve high quality results with satisfactory compression. In the case of highly deformable objects however there is no appropriate skeletal hierarchy to facilitate the skinning methods. A set of proxy-joints has to be introduced and distributed across the model so as to offer the best possible coverage. Influence weights also need to be established so as to introduce as many degrees of freedom as possible. We present a method that distributes the proxy-joints based on the deformation caused during the animation. Areas of similar deformation are identified and proxy-joints are distributed in these areas according to the degree of deformation. The goal is to position more proxy-joints in areas of high deformation to provide for the need of multiple degrees of freedom. We also accelerate the fitting process by applying it on decimated versions of the model. We show that although the simplification radically accelerates the fitting process, it barely affects the quality of the approximation.

ΕΚΤΕΝΗΣ ΠΕΡΙΛΗΨΗ ΣΤΑ ΕΛΛΗΝΙΚΑ

Γεώργιος Αντωνόπουλος του Νικολάου και της Ευγενίας

MSc. Τμήμα Πληροφορικής, Πανεπιστήμιο Ιωαννίνων, Νοέμβριος, 2010.

Μέθοδοι Ανακατασκευής Περιβλήματος για την Αποδοτική Αναπαραγωγή της Κίνησης Αντικειμένων Υψηλής Παραμόρφωσης

Επιβλέπων: Ιωάννης Φούντος

Η συμπίεση των στιγμιότυπων (key-frames) από τα οποία αποτελείται μια τριδιάστατη ακολουθία κίνησης (animation sequence) είναι απαραίτητη για την αποδοτική αποθήκευση και επεξεργασία του. Στην περίπτωση των αρθρωτών σκελετικών αντικειμένων είναι δυνατό να χρησιμοποιήσουμε μεθόδους προσδιορισμού του περιβλήματος (skinning) με πολύ καλά αποτελέσματα σχετικά με την συμπίεση που επιτυγχάνουμε και το σφάλμα που προκύπτει. Στην περίπτωση όμως των αντικειμένων υψηλής παραμόρφωσης δεν υπάρχει σκελετική ιεραρχία αλλά μπορούμε να εισάγουμε ένα σύνολο από ψευδό-αρθρώσεις που πρέπει να κατανεμηθούν πάνω στο μοντέλο ώστε να παρέχεται η καλύτερη δυνατή προσέγγισή του. Μια παράμετρος που επηρεάζει σημαντικά τα χαρακτηριστικά της μεθόδου αυτής είναι ο προσδιορισμός των βαρών επιρροής των αρθρώσεων στα σημεία του αντικειμένου. Στην εργασία αυτή παρουσιάζουμε μια μέθοδο προσδιορισμού των αρθρώσεων σε αντικείμενα υψηλής παραμόρφωσης βασιζόμενοι στην ποσότητα παραμόρφωσης που υφίσταται κάθε περιοχή του μοντέλου κατά την διάρκεια του animation. Η επιφάνεια του αντικειμένου διαμερίζεται σε περιοχές συναφούς παραμόρφωσης, ενώ οι αρθρώσεις κατανέμονται στις περιοχές αυτές με βάση την ποσότητα παραμόρφωσης. Σκοπός είναι η κατά το δυνατό καλύτερη προσέγγιση των περιοχών με υψηλή παραμόρφωση ώστε να μειώσουμε το σφάλμα της συμπίεσης σε αυτές. Επιπλέον για να επιταχύνουμε την διαδικασία της συμπίεσης εφαρμόζουμε μεθόδους μείωσης της ανάλυσης του αντικειμένου και δείχνουμε ότι είναι εφικτό να μειώσουμε δραστικά τον χρόνο χωρίς μεγάλη αύξηση του σφάλματος. Αρχικά παραθέτουμε το μαθηματικό υπόβαθρο που απαιτείται για την κατανόηση των μεθόδων που παρουσιάζουμε. Στην συνέχεια παρουσιάζουμε την επικρατούσα μέθοδο συμπίεσης για αρθρωτά, σκελετικά αντικείμενα και κάνουμε αναγωγή της μεθόδου αυτής σε αντικείμενα υψηλής παραμόρφωσης. Υποδεικνύουμε αρχές για αποδοτική διάδοση βαρών επιρροής, παρουσιάζουμε την έννοια της ποσότητας παραμόρφωσης και παρουσιάζουμε

μεθόδους κατανομής των ψευδο-αρθρώσεων. Στην συνέχεια περιγράφουμε κανόνες και μεθόδους μείωσης της ανάλυσης των αντικειμένων και πώς γίνεται η εφαρμογή των μεθόδων συμπίεσης σε αντικείμενα που έχουν υποστεί την μείωση αυτή. Ακολουθούν υπάρχουσες καθώς και δικές μας μέθοδοι για εκ' των υστέρων βελτίωση της συμπίεσης. Τέλος υπάρχει μια σύντομη περιγραφή της εφαρμογής μας, αποτελέσματα των μετρήσεών μας, συμπεράσματα και επεκτάσεις.

CHAPTER 1

INTRODUCTION

1.1 Related Work

Computer *animation* plays a major role in 3D visualization process. Animation is used in a broad spectrum of applications such as heavy industry, medicine, clothing, fashion design industry or entertainment industry (cinema, computer games). Although *articulated* (*skeletal*) animations currently dominate the field, with the advance of computer graphics hardware the demand for *highly deformable* animations has also been made feasible. Highly deformable animations are used to describe objects that deform under no skeletal influence, or act as a complement to existing skeletal animations. Highly deformable objects can be used to model clothes, both independently and in conjunction with skeletal animation. 3D animations of soft body internal organs are also used in medicine in the process of treatment, research and virtual reality medical training. Heavy industry also employs 3D animations of deformable objects in the process of studying material behavior under certain conditions such as pressure or temperature of variable intensity. Finally, highly deformable objects, in the form of clothing, are being employed by the clothing and fashion, entertainment and computer game industry. 3D animated movies production has increased in recent years and so has the use of animated models as substitute to real actors in the representation of crowds or to generate realistic scenes.

A 3D animation consists of a sequence of *poses* or *key-frames* of the same model. The production of such a sequence can be done, on certain occasions, by advanced scanning machinery. However in most cases it is the result of strenuous labor from the part of artists, who create the animation pose by pose. Specialized computer software provides several automated techniques for the generation of deformations but adding fine details always requires human intervention.

Apart from creation, real time animation sequence rendering and processing also raises challenges. The animation processing can occur key-frame by key-frame. However this technique introduces the problem of storage. Animation sequences consist of hundreds, sometimes thousands of frames, with each frame enclosing the same amount of information slightly changed. When animating large models, each sequence can sum up to hundreds of megabytes. The ramifications of size manifest not only in terms of the space used on the disk but also in terms of time required by an application to load the sequence as well as the amount of space to used in RAM. Furthermore the processing of the sequence is affected by size. In a scene a sequence may have to be replicated at various levels of details or required to interact with other objects (e.g. collide). Additionally to save time, for animating similar objects the animator may require the deformation of an object to be transferred to another. The amount of space and processing time increases dramatically considering that a scene may contain more than one animation sequences.

Matrix palette skinning (also referenced as *Skeletal Subspace Deformation*, or simply *Skinning*), is an alternative real-time rendering technique. It operates on the observation that on articulated models, in accordance to the skin, the deformation of each vertex is influenced by the skeleton. It assigns each vertex of the animation an influencing *bone* and the amount (*weight*) of this influence. The most popular algorithm employing this technique is *Linear Blend Skinning.* Instead of having a key-frame by key-frame representation, the animation sequence is diminished in a single reference key-frame and a collection of bone transformations and vertex weights. However, specifying the bones and their transformations and finding the vertex weights requires a processing of the sequence as a whole so, although model reproduction is hitherto simplified, the size of the animation sequence is still a drawback. So data reduction is imperative.

Although matrix palette skinning assumes the existence of some underlying skeletal hierarchy, it can still be applied to highly deformable objects which contain no skeleton. In this paper we present extensions of matrix palette skinning in this direction. We attempt to efficiently specify bones and areas of bone influences using clustering techniques based on the amount of deformation of the object. In addition we study and employ mesh decimation techniques to reduce the amount of data processed by the linear blend skinning algorithm and speed up the weight specification procedure. Weight acquisition is performed using both affine and rigid body transformations.

The rest of the document is structured as follows. In Chapter 2 the required mathematical background for this thesis is given. It contains elements of transformation theory describing the use of transformation matrices and dual quaternions. Certain aspects of plane theory are also discussed as well as the Barycentric coordinate system. Chapter 3 explains Matrix palette skinning and elaborates on two existing methods of fitting that where the motivation for this thesis. It distinguishes three stages of the fitting process. That of proxy joint specification, transformation fitting and weight fitting and each one is analyzed in depth. Chapter 4 describes how the fitting process can be made more efficient using a simplified version of a model. Chapter 5 describes refinement techniques that can improve the visual fidelity of the approximation. Chapter 6 contains implementation specifics and results. Finally, Chapter 7 offers conclusions and future research directions.

1.1 **Related Work**

Although a lot of work has been done on matrix palette skinning for articulated objects and quasi-articulated not much literature is concerned with this class of highly deformable objects.

In the case of quasi-articulated objects, which are more relevant, [8] constructs a skinning approximation by computing the transformations of near rigid components on the model. These are identified using facet deformation gradient and mean shift analysis. The results are very good but the algorithm presents with poor quality in highly deformable objects.

For highly deformable objects, Kavan et. al in [9] present a fitting method using dual quaternions which manages to reduce the execution time of the fitting process in the expense of quality. A refinement technique similar to Eigen-Skin [18] is also presented which improves the results with the cost of extra complexity and reduction in compression since more information must be stored. In [20] a fast approach in of the fitting process is presented, based on an iterative global optimization process. However no topology information is preserved and the location of proxy joints is occluded once the optimization process begins and can no longer be used.

In the field of mesh segmentation, [21] presents a method for identifying regions of similar movement by also employing mean shift analysis on the deformation gradient of vertices which is extracted from the weighted transformation matrices of each vertex. However only one proxy joint with a set of parameters is associated with each area and the goal is to produce a reduced representation of the model for animation editing. [22] also employs deformation gradient to identify near rigid sections onto the model. Using region growing and a geodesic distance metric, it identifies regions of low deformation and sets as segmentation boundaries the areas of high deformation. These near rigid areas are destined for use in decimation and deformation transfer. In this method the use of geodesic distances causes areas with different deformation to be group together due to proximity. What we need is to identify areas of similar deformation, whether low or high. Another technique for identifying near-rigid components is presented in [23]. Dihedral angle is used as deformation quantity and the method performs region partitioning based on minimum spanning tree expansion. However this method produces lots of fragmented sets and a merging algorithm must be employed. Furthermore dihedral angle greatly depends on the triangulation density of the model and may not produce accurate results.

CHAPTER 2

PRELIMINARIES

2.1 **Matrix Transformations**

In this section we will describe certain theoretical aspects of matrix theory and how matrices are related to 3D transformation

2.1.1 Linear Transformations

In linear algebra, a linear transformation from a linear vector space $\mathbb{R}^n$ to $\mathbb{R}^m$ is a map $L:\mathbb{R}^n \to \mathbb{R}^m$ that preserves the linear properties of $\mathbb{R}^n$:

$$\begin{aligned} &L(x+y) = L(x) + L(y),\ \forall x, y \in \mathbb{R}^n \\ &L(ax) = aL(x),\ \forall x \in R^n, a \in \mathbb{R} \end{aligned} \tag{2. 1}$$

This map can be represented by a matrix $\mathrm{A} \in \mathbb{R}^{m\times n}$ such that:

$$L(x) = Ax,\ \forall x \in \mathbb{R}^n \tag{2. 2}$$

If $e_1 = \underbrace{(1,0,...,0)}_{n-elements}, e_2 = \underbrace{(0,1,...,0)}_{n-elements}, ..., e_n = \underbrace{(0,0,...,1)}_{n-elements}$ is the standard basis of $\mathbb{R}^n$, the i-th column of matrix A is the image of the standard basis vector $e_i \in \mathbb{R}^n$ under the map L, i.e.:

$$A = [L(e_1), L(e_2), ..., L(e_n)] \in \mathbb{R}^{m\times n} \tag{2. 3}$$

This is an important property of linear transformations since it allows for sequences of transformations to be expressed as a sequence of matrix multiplications. Matrix multiplication can be efficiently implemented in the hardware and thus allow for very fast transformation operations.

In computer graphics, all rendering transformations take place in 3 dimensions, i.e. $n, m = 3$ so we narrow our approach to this dimension. When applied, linear transformations have the property of maintaining the angles of the transformed object, i.e. parallel lines remain parallel after the transformation. Depending on the effect of the transformation upon the object, linear transformations can represent rotations, scaling, shearing or reflections, with rotations, scaling and reflections being the most commonly used.

Rotation

In 3-D, rotation transformations have the effect of rotating the transformed data around an axis for a certain angle. For arbitrary axis described by $\mathbf{n} = [\mathbf{n}_\mathrm{x}\ \mathbf{n}_\mathrm{y}\ \mathbf{n}_\mathrm{z}]^T$, the matrix that performs rotation around it by θ degrees is given by:

$$R(\mathbf{n},\theta) = \begin{bmatrix} \mathbf{n}_\mathrm{x}^2(1-\cos\theta)+\cos\theta & \mathbf{n}_\mathrm{x}\mathbf{n}_\mathrm{y}(1-\cos\theta)-\mathbf{n}_\mathrm{z}\sin(\theta) & \mathbf{n}_\mathrm{x}\mathbf{n}_\mathrm{z}(1-\cos\theta)+\mathbf{n}_\mathrm{y}\sin\theta \\ \mathbf{n}_\mathrm{x}\mathbf{n}_\mathrm{y}(1-\cos\theta)+\mathbf{n}_\mathrm{z}\sin\theta & \mathbf{n}_\mathrm{y}^2(1-\cos\theta)+\cos\theta & \mathbf{n}_\mathrm{y}\mathbf{n}_\mathrm{z}(1-\cos\theta)-\mathbf{n}_\mathrm{x}\sin\theta \\ \mathbf{n}_\mathrm{x}\mathbf{n}_\mathrm{z}(1-\cos\theta)-\mathbf{n}_\mathrm{y}\sin\theta & \mathbf{n}_\mathrm{y}\mathbf{n}_\mathrm{z}(1-\cos\theta)+\mathbf{n}_\mathrm{x}\sin\theta & \mathbf{n}_\mathrm{z}^2(1-\cos\theta)+\cos\theta \end{bmatrix} \qquad \textbf{(2. 4)}$$

For example, for rotation around X-axis in a right-handed (the rotation direction is that of the fingers of the right hand, when the thumb shows down the positive X axis) coordinates system, where $\mathbf{n} = [1\ 0\ 0]^T$:

$$R(\mathbf{n},\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix} \qquad \textbf{(2. 5)}$$

Figure 2.1 shows the effect of applying such transformation for $\theta = 90^o$.

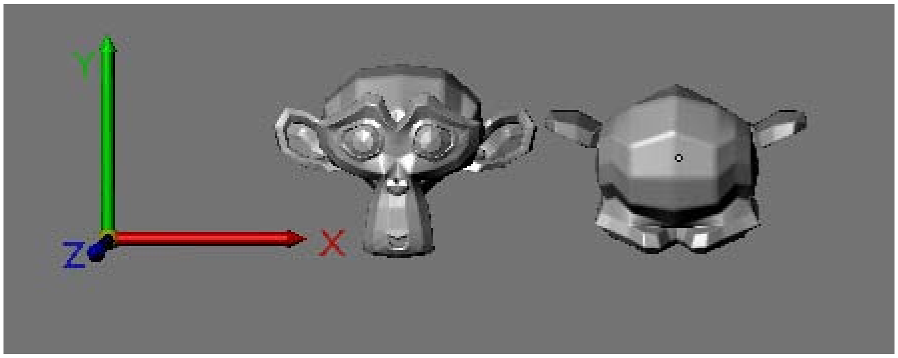


Figure 2.1: Rotation about X-Axis by 90 degrees

Scaling

Scaling transformations affect the size of an object towards an axis by a factor s. For arbitrary axis described by $\mathbf{n} = [\mathbf{n_x}\ \mathbf{n_y}\ \mathbf{n_z}]^T$, the transformation matrix is given by:

$$S(\mathbf{n}, s) = \begin{bmatrix} 1+(s-1)\mathbf{n}_x^2 & (s-1)\mathbf{n}_x\mathbf{n}_y & (s-1)\mathbf{n}_x\mathbf{n}_z \\ (s-1)\mathbf{n}_x\mathbf{n}_y & 1+(s-1)\mathbf{n}_y^2 & (s-1)\mathbf{n}_y\mathbf{n}_z \\ (s-1)\mathbf{n}_x\mathbf{n}_z & (s-1)\mathbf{n}_y\mathbf{n}_z & 1+(s-1)\mathbf{n}_z^2 \end{bmatrix} \quad \textbf{(2. 6)}$$

To double the size of an object towards the Y-Axis, that is $\mathbf{n} = [0\ 1\ 0]^T$ and $s = 2$, the resulting transformation matrix would be:

$$S(\mathbf{n}, s) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \textbf{(2. 7)}$$

Figure 2.2 shows the application of such transformation on the object to the left.

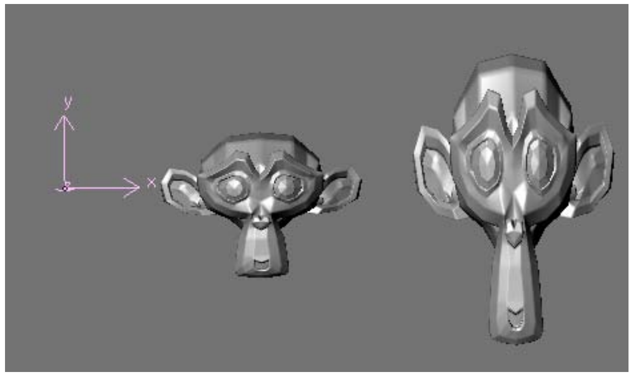


Figure 2.2: Scaling an object towards the Y-axis by factor 2.

Shearing

In 3-D, shearing is a transformation that "skews" the coordinate space and stretches it non-uniformly. It operates on the coordinate of one axis by adding to it a scalar multiple of the other. Depending on which axis is affected the transformation matrices are as follows:

$$H_{xy}(s,t)=\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ s & t & 1\end{bmatrix},\ H_{xz}(s,t)=\begin{bmatrix}1 & 0 & 0\\ s & 1 & t\\ 0 & 0 & 1\end{bmatrix},\ H_{yz}(s,t)=\begin{bmatrix}1 & s & t\\ 0 & 1 & 0\\ 0 & 0 & 1\end{bmatrix} \quad \textbf{(2. 8)}$$

Figure 2.3 depicts the effect of shearing upon a model.

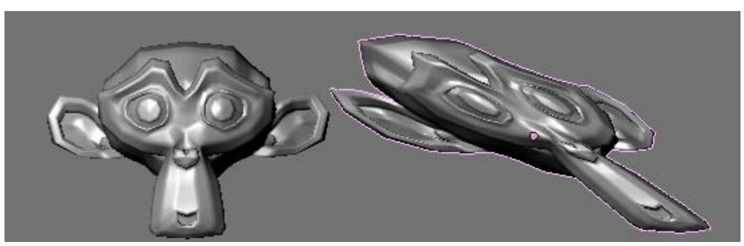


Figure 2.3: Shearing an object towards X-Axis

Reflection

In 3-D, reflection, or mirroring is a transformation that "flips" the object about a plane. Reflection is achieved if a scaling of factor $s=-1$ is applied upon the object. Thus, if $\mathbf{n}=[\mathbf{n}_x,\mathbf{n}_y,\mathbf{n}_z]^T$ is the normal (perpendicular) vector to the plane then the reflection matrix is by:

$$R(\mathbf{n}) = S(\mathbf{n},-1) = \begin{bmatrix} 1-2\mathbf{n}_x^2 & -2\mathbf{n}_x\mathbf{n}_y & -2\mathbf{n}_x\mathbf{n}_z \\ -2\mathbf{n}_x\mathbf{n}_y & 1-2\mathbf{n}_y^2 & -2\mathbf{n}_y\mathbf{n}_z \\ -2\mathbf{n}_x\mathbf{n}_z & -2\mathbf{n}_y\mathbf{n}_z & 1-2\mathbf{n}_z^2 \end{bmatrix} \tag{2. 9}$$

Figure 2.4 shows a reflection of an object about the XZ-plane

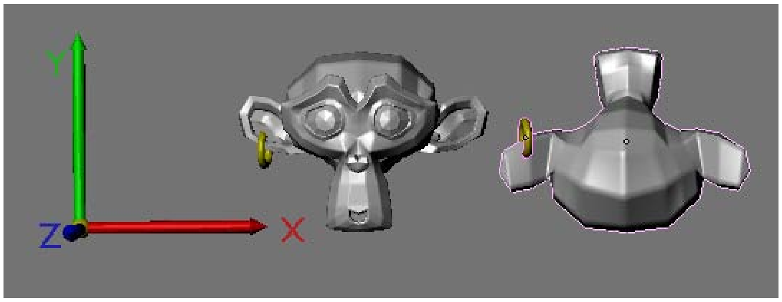


Figure 2.4: Reflection about the XZ -plane

2.1.2 *Composition of Transformations*

Several consecutive transformations can be expressed via a single matrix. The process of composing individual transformations is carried out by multiplying the corresponding matrices in an order, reversed to the one the transformations occur. For a rotation R followed by a scaling transformation S, the composition matrix is:

$$M = SR \tag{2. 10}$$

2.1.3 *Affine Transformations*

An affine transformation L from $\mathbb{R}^n$ to $\mathbb{R}^n$ is defined by a non-singular, linear transformation matrix $A \in \mathbb{R}^{n\times n}$, and a vector $b \in \mathbb{R}^n$ such that:

$$L : \mathbb{R}^n \to \mathbb{R}^n ; x \mapsto Ax + b \tag{2. 11}$$

It should be noted that such a map is not linear unless $b = 0$. To alleviate this and allow for affine transformations also to be performed as matrix multiplications L is raised by one dimension. Expressing $x \in R^n$ as $\begin{bmatrix} x \\ 1 \end{bmatrix} \in R^{n+1}$, L becomes a map from R^{n+1} to R^{n+1} and **Error! Reference source not found.**(2. 11) becomes:

$$L : \mathbb{R}^{n+1} \to \mathbb{R}^{n+1};\ \begin{bmatrix} x \\ 1 \end{bmatrix} \mapsto \begin{bmatrix} A & b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ 1 \end{bmatrix} \tag{2. 12}$$

This representation of x is called homogenous representation and it is used extensively in computer graphics since it allows for A, b to be expressed with a single matrix while preserving linearity of transformations.

Translation

The addition of vector $b \in \mathbb{R}^n$ to a linear transformation (2. 12) denotes another commonly used transformation called translation and has the effect of changing the transformed vectors location. The matrix representation of a translation by a vector $\mathbf{t} = [\mathbf{t}_x\ \mathbf{t}_y\ \mathbf{t}_z]^T$, in homogenous representation is given by:

$$T(\mathbf{t}) = \begin{bmatrix} 1 & 0 & 0 & \mathbf{t}_x \\ 0 & 1 & 0 & \mathbf{t}_y \\ 0 & 0 & 1 & \mathbf{t}_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2. 13}$$

2.1.4 Euclidean Transformations

A Euclidean transformation L from $\mathbb{R}^n$ to $\mathbb{R}^n$ is defined by a non-singular, linear transformation matrix $R \in \mathbb{R}^{n \times n}$ which has the property of being orthogonal (i.e. $R^T R = RR^T = I$), and a vector $T \in \mathbb{R}^n$ such that:

$$L : \mathbb{R}^n \to \mathbb{R}^n; x \mapsto Rx + T \tag{2. 14}$$

The notation choice of R is not random. Of all linear transformations, only rotation has the property of being orthogonal. Thus a Euclidian transformation is an affine transformation, which in homogenous representation is expressed by :

$$\begin{bmatrix} R & T \\ 0 & 1 \end{bmatrix} \in \mathbb{R}^{(n+1)\times(n+1)} \tag{2. 15}$$

and represents a rotation and a translation. Such transformations are also called rigid body Transformations since, upon application, the shape of the transformed object remains intact and what changes is the orientation and the location of the transformed object.

2.1.5 Inverse Transformations

A transformation is invertible if an opposite transformation exists that can restore the transformed object to its former condition. Since transformations are expressed as matrices, finding the inverse transformation is equivalent to finding the inverse matrix of the transformation. As far as the affine transformations are concerned, inverting the translational part is only a matter of negating it, i.e. translating the object to the opposite direction. However not all linear transformations are invertible thus A's singularity has to be checked. This is not the case with Euclidean transformations which are always invertible since R is orthogonal and $R^{-1} = R^T$.

Knowing that the composing transformations of $M = M_1 M_2 ... M_n$ are invertible, the inverse transformation of M is given by:

$$M^{-1} = M_n^{-1} ... M_2^{-1} M_1^{-1} \tag{2. 16}$$

2.2 **Quaternion Transformations**

In this section we present the theory behind using quaternions and dual quaternions to substitute matrix transformations.

2.2.1 Quaternions

A vector $\mathbf{p} = [x \ \ y]^T$ can be expressed as a complex number as follows:

$$\mathbf{p} = (x + yi) \tag{2. 17}$$

Then it can be rotated around the origin (Figure 2.5) by an angle θ if it gets multiplied by another complex number $q = (\cos\theta + (\sin\theta)i)$.

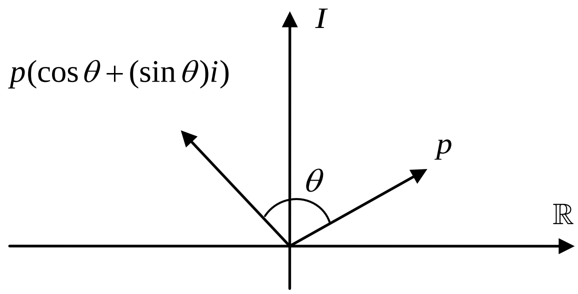


Figure 2.5: Complex number rotation around the origin by angle θ

Quaternions where introduced by William Hamilton [1] in an attempt to extend this notion in 3-D. As the name denotes, a quaternion is a quad of numbers $q = [w \ \ \mathbf{n}] = [w, x, y, z]$ that can be expressed a complex number

$$q = w + xi + yj + zk \tag{2. 18}$$

where $i^2 = j^2 = k^2 = -1,\ ij = k,\ ji = -k,\ jk = i,\ kj = -i,\ ki = j,\ ik = -j$. A quaternion can be used to represent a rotation (angular displacement) by θ degrees about an arbitrary vector $\mathbf{n}$ in 3-D. Figure 2.5 shows the relation between the quaternion elements, the angle θ and the axis of rotation $\mathbf{n}$.

$$q = [\cos\frac{\theta}{2} \;\; \sin\frac{\theta}{2}\mathbf{n}]$$
$$q = [\cos\frac{\theta}{2} \; \sin\frac{\theta}{2}\mathbf{n_x} \; \sin\frac{\theta}{2}\mathbf{n_y} \; \sin\frac{\theta}{2}\mathbf{n_z}] \tag{2. 19}$$

Note that only those quaternions that can be described by this equation can be used for angular displacement representations.

Quaternion Magnitude

The magnitude of a quaternion is given by:

$$\|q\| = \|w \;\; \mathbf{p}\| = \|w \;\; (\mathbf{p_x} \;\; \mathbf{p_y} \;\; \mathbf{p_z})\| = \sqrt{w^2 + \mathbf{p_x}^2 + \mathbf{p_y}^2 + \mathbf{p_z}^2} = \sqrt{w^2 + \|\mathbf{p}\|} \tag{2. 20}$$

If $\|q\| = 1$, a quaternion is called *Unit Quaternion*. Only unit quaternions can be used to describe angular displacement. Unit quaternions are described by (2. 24).

3-D Point as Quaternion

3-D points can also be represented by a quaternion. A point can be interpreted a quaternion that inflicts zero angular displacement around it, i.e. for point $\mathbf{p} = [\mathbf{p_x} \; \mathbf{p_y} \; \mathbf{p_z}]^T$ the following equation holds:

$$\mathbf{p} = q_{\mathbf{p}} = [0 \;\; \mathbf{p}] = [0 \;\; \mathbf{p_x} \;\; \mathbf{p_y} \;\; \mathbf{p_z}] \tag{2. 21}$$

Quaternion Negation

Denoted by $-q$, negating a quaternion is performed by negating each of its elements:

$$-q = [-w \;\; -\mathbf{n}] = [-w \;\; -xi \;\; -yj \;\; -zk] \tag{2. 22}$$

q and $-q$ represent the same angular displacement, executed from the opposite direction.

Identity Quaternion

Identity quaternions inflict no angular displacement upon a quaternion. The quaternion that achieves this is:

$$q = [1 \;\; \mathbf{0}] = [1 \;\; 0 \;\; 0 \;\; 0] = 1 + 0i + 0j + 0k \tag{2. 23}$$

$-q = [-1 \;\; \mathbf{0}]$ results in the same angular displacement but, mathematically, only $q = [1 \;\; \mathbf{0}]$ is to be considered as a identity quaternion.

Quaternion Normalization

As with vectors, transforming quaternions into unit ones is carried out by dividing them by their length:

$$q_{normal} = \frac{q}{\|q\|} \tag{2. 24}$$

Quaternion Conjugate and Inverse

Following the complex number conjugate, the conjugate of a quaternion is given by:

$$q^* = [w \;\; (x \;\; y \;\; z)]^* = [w \;\; -(x \;\; y \;\; z)] = w - xi - yj - zk \tag{2. 25}$$

The inverse quaternion is given by:

$$q^{-1} = \frac{q^*}{\|q\|} \tag{2. 26}$$

In the case of unit quaternions, the inverse is equal to the conjugate.

Quaternion Product (Cross Product)

According to their complex number representation, multiplication among two quaternions is defined as:

$$q_1 q_2 = \\ [w_1 \ \ (\mathbf{v_{1x}} \ \ \mathbf{v_{1y}} \ \ \mathbf{v_{1z}})][w_2 \ \ (\mathbf{v_{2x}} \ \ \mathbf{v_{2y}} \ \ \mathbf{v_{2z}})] = [w_1 w_2 - \mathbf{v_1} \bullet \mathbf{v_2} \ \ (w_1 \mathbf{v_2} + w_2 \mathbf{v_1} + \mathbf{v_2} \times \mathbf{v_1})] \tag{2. 27}$$

Note that, as with matrices, quaternion multiplication is associative but not commutative:

$$\begin{gathered} (q_1 q_2) q_3 = q_1 (q_2 q_3) \\ q_1 q_2 \neq q_2 q_1 \end{gathered} \tag{2. 28}$$

Angular Displacement via Quaternions

To rotate a 3-D point $\mathbf{p} = [\mathbf{p_x} \ \ \mathbf{p_y} \ \ \mathbf{p_z}]^T$ by angle θ about an axis defined by a vector $\mathbf{n} = [\mathbf{n_x} \ \ \mathbf{n_y} \ \ \mathbf{n_z}]^T$, the point $\mathbf{p}$ and the angular displacement are expressed as a quaternions p and q respectively (2. 21). Rotation is performed as follows:

$$p' = qpq^{-1} \tag{2. 29}$$

and then p' is expressed back as 3-D point. For multiple rotations equation becomes:

$$p' = q_2 (q_1 p q_1^{-1}) q_2^{-1} = (q_2 q_1) p (q_2 q_1)^{-1} \tag{2. 30}$$

showing that as series of rotations, as with matrices, it can be performed as a series of quaternion multiplications.

Quaternion Dot Product

Dot product among quaternions is defined as follows:

$$q_1 \bullet q_2 = [w_1 \ \ \mathbf{v_1}] \bullet [w_2 \ \ \mathbf{v}_2] = w_1 w_2 + v_1 \bullet v_\mathbf{2} = w_1 w_2 + \mathbf{v_{1x}} \mathbf{v_{2x}} + \mathbf{v_{1y}} \mathbf{v_{2y}} + \mathbf{v_{1z}} \mathbf{v_{2z}} \tag{2. 31}$$

Quaternion Scalar Multiplication

Scalar multiplication by a scalar k is performed by multiplying each of quaternion elements with k:

$$aq = [aw \;\; a\mathbf{n}] = [aw \;\; axi \;\; ayj \;\; azk] \tag{2. 32}$$

2.2.2 Dual Quaternions

Dual numbers

Similar to complex numbers, a dual number $\hat{a}$ is written as $\hat{a} = a_0 + \varepsilon a_\varepsilon$, where ε has the property $\varepsilon^2 = 0$. a_0 is considered the non-dual part, a_ε is the dual part and ε is the dual unit. Dual number operations follow complex numbers operations. For example dual number multiplication is carried out as follows:

$$(a_0 + \varepsilon a_\varepsilon)(b_0 + \varepsilon b_\varepsilon) = a_0 b_0 + \varepsilon(a_0 b_\varepsilon + a_\varepsilon b_0) \tag{2. 33}$$

The inverse of a dual number is given by:

$$\hat{a}^{-1} = \frac{1}{a_0 + \varepsilon a_\varepsilon} = \frac{1}{a_0} + \varepsilon \frac{a_\varepsilon}{a_0^2} \tag{2. 34}$$

Quaternions and dual numbers

Dual quaternions are dual numbers with their dual and non-dual parts being quaternions:

$$\begin{aligned} \hat{q} &= q_0 + \varepsilon q_\varepsilon = [w_0 \;\; x_0 \;\; y_0 \;\; z_0] + \varepsilon[w_\varepsilon \;\; x_\varepsilon \;\; y_\varepsilon \;\; z_\varepsilon] \\ &= w_0 + x_0 i + y_0 j + z_0 k + \varepsilon w_\varepsilon + \varepsilon x_\varepsilon i + \varepsilon y_\varepsilon j + \varepsilon z_\varepsilon k \end{aligned} \tag{2. 35}$$

where

$$\begin{aligned} \varepsilon i &= i\varepsilon \\ \varepsilon j &= j\varepsilon \\ \varepsilon k &= k\varepsilon \end{aligned} \tag{2. 36}$$

3-D Point as Dual Quaternion

A 3-D point $\mathbf{p} = [\mathbf{p_x} \;\; \mathbf{p_y} \;\; \mathbf{p_z}]^T$ can be expressed as a dual quaternion by:

$$\hat{\mathbf{p}} = 1 + \varepsilon(\mathbf{p_x} i + \mathbf{p_x} j + \mathbf{p_z} k) \qquad \textbf{(2. 37)}$$

where $q_\varepsilon = \mathbf{p_x} i + \mathbf{p_x} j + \mathbf{p_z} k$ is given by (2. 21).

Dual Quaternion Conjugate

There are three types of dual quaternion conjugates depending upon which component the operant is applied:

a) **Dual conjugate**: Conjugation is applied only upon the dual unit and the result is given by:

$$\bar{\hat{q}} = q_0 - \varepsilon q_\varepsilon \qquad \textbf{(2. 38)}$$

b) **Quaternion Conjugate**: Conjugation is applied only upon quaternion parts and the result is given by:

$$\widehat{q^*} = q_0^* + \varepsilon q_\varepsilon^* \qquad \textbf{(2. 39)}$$

c) **Dual Quaternion Conjugate**: Conjugation is applied both on quaternion parts and the dual unit and the result is given by:

$$\overline{\widehat{q^*}} = q_0^* - \varepsilon q_\varepsilon^* \qquad \textbf{(2. 40)}$$

Dual Quaternion Magnitude

The magnitude of a dual quaternion is given by :

$$\|q\| = \sqrt{\widehat{q^*}\hat{q}} = \|q_0\| + \varepsilon \frac{\langle q_0, q_\varepsilon \rangle}{\|q_0\|} \tag{2. 41}$$

If $\|\hat{q}\| = 1$, i.e. $\|q_0\| = 1, \langle q_0, q_\varepsilon \rangle = 0$, then $\hat{q}$ is a unit dual quaternion and has the property of always being invertible.

Dual Quaternion Inverse

The inverse of a dual quaternion exists if $q_0 \neq 0$ and is given by:

$$\hat{q}^{-1} = \frac{\widehat{q^*}}{\|\hat{q}\|^2} \tag{2. 42}$$

In the case of a unit dual quaternion it holds that $\hat{q}^{-1} = \widehat{q^*}$.

Dual Quaternion Normalization

Transforming a dual quaternion into a unit one is carried out by dividing it by its magnitude:

$$\hat{q}_n = \frac{\hat{q}}{\|\hat{q}\|} \tag{2. 43}$$

Using (2. 41), this equation becomes:

$$\hat{q}_n = \frac{q_0}{\|q_0\|} + \varepsilon \left(\frac{q_\varepsilon}{\|q_0\|} - \frac{q_0 \langle q_0, q_\varepsilon \rangle}{\|q_0\|^3} \right) \tag{2. 44}$$

Rigid Body Transformations via Dual Quaternions

Dual quaternions have the property of representing both rotation and translation transformations in 3-D. The non-dual is the rotational part and the dual is the translational part of a transformation. Similar to simple quaternions, only unit dual quaternions can be used to represent transformations.

Rotation

As with simple quaternions, for a 3-D point to be rotated, it must be represented as a dual quaternion (2. 37). Rotation is then carried out by multiplying the vector with the dual quaternion in an analogous fashion as in (2. 29):

$$\hat{p}' = \hat{q}\hat{p}\widehat{q^{-1}} = \hat{q}\hat{p}\overline{\widehat{q^{*}}} \qquad \textbf{(2. 45)}$$

The second part of the equation holds due to use of unit dual quaternions. Since $\hat{q}$ is a rotation only dual quaternion, $q_\varepsilon = 0$. So the above is simplified into:

$$q_0(1+\varepsilon(\mathbf{p_x}i+\mathbf{p_y}j+\mathbf{p_z}k))q_0^* = 1+\varepsilon q_0(\mathbf{p_x}i+\mathbf{p_y}j+\mathbf{p_z}k)q_0^* \qquad \textbf{(2. 46)}$$

Notice that the dual part is the quaternion rotation equation described by (2. 29). Also notice that after the multiplication operation is completed, what remains is the transformed 3-D point expressed as a dual quaternion.

Translation

A unit dual quaternion $\hat{t}$ defined as $\hat{t} = 1 + \varepsilon(\frac{\mathbf{t_x}}{2}i + \frac{\mathbf{t_y}}{2}j + \frac{\mathbf{t_z}}{2}k)$ corresponds to a translation by the vector $\mathbf{t} = [\mathbf{t_x} \;\; \mathbf{t_y} \;\; \mathbf{t_z}]^T$. Notice that the rotational part is the identity quaternion, thus there is no angular displacement.

Translation of a 3-D vector is carried out in the same way described in (2. 45):

$$\hat{t}p\overline{\hat{t}^*} = 1 + \varepsilon((\mathbf{p_x} + \mathbf{t_x})i + (\mathbf{p_y} + \mathbf{t_y})j + (\mathbf{p}_z + \mathbf{t}_z)k) \qquad \textbf{(2. 47)}$$

and what remains is the 3-D vector, expressed as a dual quaternion, translated by $\mathbf{t}$.

Rigid Body Motion

A unit dual quaternion $\hat{q}$ which describes rotation can be combined with a unit dual quaternion $\hat{t}$ which describes translation, to jointly describe a rigid body motion. The combination of these two quaternions, as in matrix representation, is their product. Again, the order of multiplication is important:

$$\hat{q} = (1 + \varepsilon(\frac{\mathbf{t_x}}{2}i + \frac{\mathbf{t_y}}{2}j + \frac{\mathbf{t_z}}{2}k))q_0 = q_0 + \varepsilon(\frac{\mathbf{t_x}}{2}i + \frac{\mathbf{t_y}}{2}j + \frac{\mathbf{t_z}}{2}k)q_0 = q_0 + \varepsilon q_\varepsilon \qquad \textbf{(2. 48)}$$

Applying the combined transformation is done in the way described by (2. 45) and in the form of quaternion transformations the result is:

$$\begin{aligned} p' &= (q_0 + \varepsilon q_\varepsilon)p(q_0^* - \varepsilon q_\varepsilon^*) = (q_0 + \varepsilon q_\varepsilon)(pq_0^* - \varepsilon pq_\varepsilon^*) = \\ & q_0 pq_0^* - \varepsilon q_0 pq_\varepsilon^* + \varepsilon q_\varepsilon pq_0^* = 1 + \varepsilon\left(\frac{q_\varepsilon pq_0^* - q_0 pq_\varepsilon^*}{q_0 pq_0^*}\right) \end{aligned} \qquad \textbf{(2. 49)}$$

2.2.3 Quaternion to Matrix Transformation

On several occasions it is required to change representation from quaternion to matrix. For a dual quaternion:

$$\hat{q} = (w_0 + x_0 i + y_0 j + z_0 k) + \varepsilon(w_\varepsilon + x_\varepsilon i + y_\varepsilon j + z_\varepsilon k) \qquad \textbf{(2. 50)}$$

its matrix representation is given by:

$$\begin{bmatrix} 1-2y_0^2-2z_0^2 & 2(x_0y_0-w_0z_0) & 2(x_0z_0+w_0y_0) & -2(w_\varepsilon x_0 - x_\varepsilon w_0 + y_\varepsilon z_0 - z_\varepsilon y_0) \\ 2(x_0y_0+w_0z_0) & 1-2x_0^2-2z_0^2 & 2(y_0z_0-w_0x_0) & -2(w_\varepsilon y_0 - x_\varepsilon z_0 - y_\varepsilon w_0 + z_\varepsilon x_0) \\ 2(x_0z_0-w_0y_0) & 2(y_0z_0+w_0x_0) & 1-2x_0^2-2y_0^2 & -2(w_\varepsilon z_0 + x_\varepsilon y_0 - y_\varepsilon x_0 - z_\varepsilon w_0) \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad \textbf{(2. 51)}$$

The representation for a simple quaternion is acquired by setting :

$$q_\varepsilon = 0 : w_\varepsilon = 0,\ x_\varepsilon = 0,\ y_\varepsilon = 0,\ z_\varepsilon = 0$$

2.2.4 Matrix VS Quaternion

Quaternions and dual quaternions offer an alternative representation for rotation and rigid body transformations respectively. When compared to matrices, quaternions appear to be advantageous in terms of required storage and in composition of transformations. However, they require considerably more operations to apply the transformation. Dual quaternions too require less storage than matrices, but the number of operations both for transformation composition and 3-D vector transformation is very high.

Table 2.1 shows a comparison between the quaternions and matrices in terms of storage and speed. Speed is measured in terms of the number of operations, namely Multiplications (M) and Additions (A).

A few notes on how these numbers emerged are necessary. First of all, the quaternion product operation and thus the composition of transformations, in the general case require 16M+12A operations. However, if the product is between a quaternion and a 3-D point representation as quaternion, then the number of operations is reduced to 12M+8A, since the real part of the 3-D point is 0. Dual Quaternions require 3(16M+12A)+4A=48M+40A operations for composition. However only a rotation or only a translation, requires the same amount (i.e. 16M+12A) with quaternions since in the first case $q_\varepsilon = 0$ and in the second $q_0 = 1$. Thus the operation is reduced in a quaternion product instead of a dual quaternion product. Additionally, the product of a dual quaternion with a 3-D point requires 2(12M+8A) = 24M+16A operations.

Finally, although quaternions require more operations than matrices to apply a transformation, if this transformation is a composition of several transformations then, as seen in **Error! Reference source not found.**, the operations gained by composition speed, earn quaternions enough time to be expressed as matrices and then apply the transformation. This is not the case with dual quaternions since even in composition time are outperformed by matrices.

Table 2.1: Comparison between matrices and quaternions in terms of storage and operations

	Matrix representation		Quaternion Representation		Dual Quaternion Representation	
Rotation Storage (in numbers)	9		4		4	
Rigid Body Storage (in numbers)	12		-		7	
Quaternion To Matrix Operations	**Mult.**	**Add.**	**Mult.**	**Add.**	**Mult.**	**Add.**
	-	-	18	21	30	33
Composition Operations			16	12	48	40
Rotation Only Composition Operations	27	18	16	12	16	12
Rigid Body Composition Operations	36	24	-	-	48	40

Rotation Only Transformation Operations	**Mult.**	**Add.**	**Mult.**	**Add.**	**Mult.**	**Add.**
	9	6	32	20	32	20
Rigid Body Transformation Operations	**Mult.**	**Add.**	**Mult.**	**Add.**	**Mult.**	**Add.**
	12	9	-	-	100	64
n Rotation Compositions followed Transformation	**Mult.**	**Add.**	**Mult.***	**Add.***	**Mult.**	**Add.**
	27n+12	18n+9	16n+18+12	12n+21+9	-	-
N Rigid Body Compositions followed Transformation	**Mult.**	**Add.**	**Mult.**	**Add.**	**Mult.***	**Add.***
	36n+12	24n+9	-	-	48n+30+12	40n+33+9
*Quaternions and dual quaternions transformed to matrices before transformation occurred						

2.3 Plane Theory Elements

2.3.1 Quadratic Plane Representation

A plane can be represented by a point $\mathbf{v} = [\mathbf{v_x} \ \ \mathbf{v_y} \ \ \mathbf{v_z}]^\mathbf{T}$, and a vector $\mathbf{n} = [a \ \ b \ \ c]^\mathbf{T}$, which is perpendicular to that plane and defines it (Figure 2.7). Any point $\mathbf{x} = [\mathbf{x_x} \ \ \mathbf{x_y} \ \ \mathbf{x_z}]^\mathbf{T}$ lies on that plane if the following equation holds [2]:

$$\begin{aligned}&(\mathbf{x}-\mathbf{v})^T \bullet \mathbf{n} = 0 \Leftrightarrow \\ &a(\mathbf{x_x}-\mathbf{v_x}) + b(\mathbf{x_y}-\mathbf{v_y}) + c(\mathbf{x_z}-\mathbf{v_z}) = 0 \Leftrightarrow \\ &a\mathbf{x_x} + b\mathbf{x_y} + c\mathbf{x_z} - (a\mathbf{v_x} + b\mathbf{v_y} + c\mathbf{v_z}) = 0\end{aligned} \tag{2. 52}$$

which is an equation of the form $ax + by + cz + d = 0$.

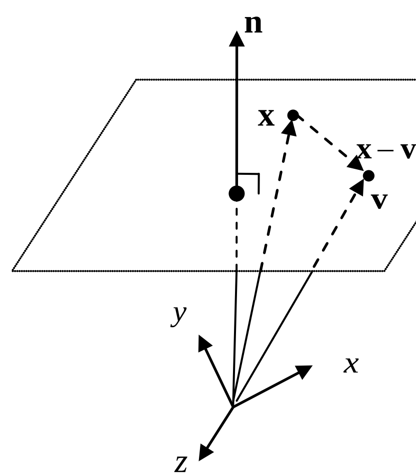


Figure 2.6: A plane defined by a point and a normal vector

In this paper, planes are specialized in the context of triangles, the building blocks of triangulated meshes. In the following sections all plane equations will be in reference to a triangle whose vertices are $T = [t_1\ t_2\ t_3],\ t_i \in \mathbb{R}^3$. Point $\mathbf{v}$ can be any of the triangles vertices. Under this context, in the plane equation (2. 52) part d is the triple product of the coordinates of the edges of the triangle [3]:

$$d = -[t_1, t_2, t_3] = -((t_1 \times t_2) \bullet t_3) \tag{2. 53}$$

Equation (2. 52) states that any point $\mathbf{x}$ lies on the triangle if the vector connecting $\mathbf{x}$ and $\mathbf{v}$ is perpendicular to the normal vector $\mathbf{n}$ (Figure 2.7). Also, since the distance from a plane is given by:

$$D = \frac{|ax + by + cz + d|}{\sqrt{a^2 + b^2 + c^2}} \tag{2. 54}$$

for a unit normal vector $\mathbf{n}$ (i.e. $\sqrt{a^2 + b^2 + c^2} = 1$), $(\mathbf{x} - \mathbf{v})^T \bullet \mathbf{n}$ is the distance of any point $\mathbf{x}$ from the triangle's plane. Intuitively, a point $\mathbf{x}$ lies on the triangle if its distance from it, is zero.

Combining (2. 52) and (2. 53) we can deduce that a plane can be represented by a vector $\mathbf{p} = [a \;\; b \;\; c \;\; d]^T$ so :

$$(\mathbf{p}^T \bullet \mathbf{x}) = [a \;\; b \;\; c \;\; d] \cdot [x \;\; y \;\; z \;\; 1]^T \qquad \textbf{(2. 55)}$$

is the distance of any vector $\mathbf{x}$ from plane $\mathbf{p}$.

The squared distance then, denoted by $Q_{\mathbf{p}}(x)$, is given by:

$$\begin{aligned} \mathbf{Q_p(x)} &= \mathbf{(p^Tx)^2} \\ &= \mathbf{(p^Tx)(p^Tx)} \\ &= \mathbf{(x^Tp)(p^Tx)} \\ &= \mathbf{x^T(pp^T)x} \\ &= \mathbf{x^TK_px} \end{aligned} \qquad \textbf{(2. 56)}$$

where $\mathbf{K_p}$ is the matrix:

$$\mathbf{K_p} = \begin{bmatrix} a^2 & ab & ac & ad \\ ab & b^2 & bc & bd \\ ac & bc & c^2 & cd \\ ad & bd & cd & d^2 \end{bmatrix} \qquad \textbf{(2. 57)}$$

$\mathbf{K_p}$ is called *fundamental error quadric* and can be used to find the squared distance of any vertex from a plane $\mathbf{p} = [a \;\; b \;\; c \;\; d]$.

2.3.2 Plane Deformation Quantities

When triangulation is used for 3D Mesh segmentation, triangles become the building blocks of the mesh. If a triangulated mesh is animated, almost all of its triangles undergo series of affine transformations throughout the animation sequence. Figure 2.7 shows how a facet of a triangulated mesh can deform within an animation sequence.

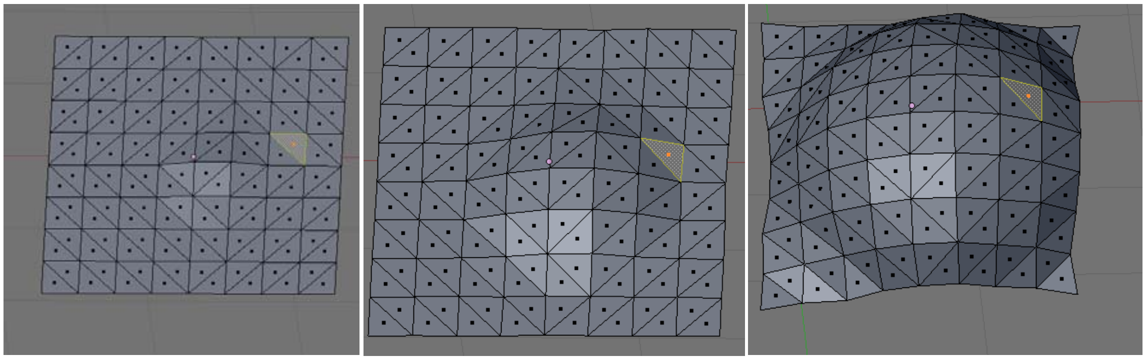


Figure 2.7: Facet Deformations in a triangulated mesh animation sequence

There are various means to quantify this deformation. In the following sections we present three of them. In each method we are referring to a mesh animation P consisting of n poses $P_0,...,P_n$, one of these poses is chosen as a reference pose (say P_0).

Triangle Area Deformation

One way is to measure how the area of the triangle changes between the triangle at some reference pose P_0 and the examining pose P_i. The area of a triangle $T=[t_1\ \ t_2\ \ t_3],\ t_i \in \mathbb{R}^3$ in 3D is given by:

$$Area=\frac{1}{2}\sqrt{\left(\det\begin{pmatrix} x_1 & x_2 & x_3 \\ y_1 & y_2 & y_3 \\ 1 & 1 & 1 \end{pmatrix}\right)^2+\left(\det\begin{pmatrix} y_1 & y_2 & y_3 \\ z_1 & z_2 & z_3 \\ 1 & 1 & 1 \end{pmatrix}\right)^2+\left(\det\begin{pmatrix} z_1 & z_2 & z_3 \\ x_1 & x_2 & x_3 \\ 1 & 1 & 1 \end{pmatrix}\right)^2} \quad \textbf{(2. 58)}$$

This way the shape deformation of the triangle can be measured.

Dihedral Angle

To measure the change in orientation, dihedral angle can be used, i.e. the angle between the triangle at some reference pose P_0 and the examining pose P_i. To measure this angle the normal vectors of both faces are used. The angle between these vectors, acquired by the dot product, is the Dihedral Angle of the planes:

$$(\mathbf{n_0} \bullet \mathbf{n_i}) = \|\mathbf{n_0}\|\|\mathbf{n_i}\|\cos(\theta) \Leftrightarrow$$
$$\theta = a\cos\left(\frac{(\mathbf{n_0} \bullet \mathbf{n_i})}{\|\mathbf{n_0}\|\|\mathbf{n_i}\|}\right) \quad \textbf{(2. 59)}$$

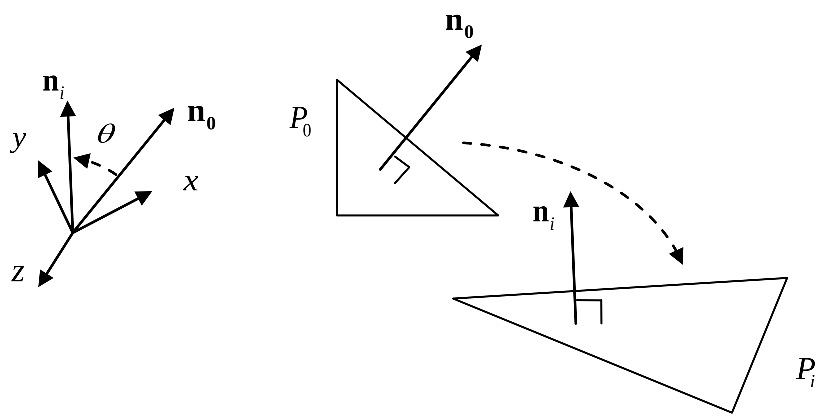


Figure 2.8: Dihedral angle between the same triangle at two different poses

Deformation Gradient

Deformation Gradient is a quantity that encloses both shape and orientation of the deformation. Assume that we have an animation sequence of a triangulated model with each triangle $T = \{v_1, v_2, v_3\}$ performing series of affine transformations throughout the animation. The affine transformation Φ_j of the j -th triangle that contains some vertex $\mathbf{v}$ is given by [5]:

$$\Phi_j(\mathbf{v}) = C_j\mathbf{v} + \mathbf{t}_j \quad \textbf{(2. 60)}$$

where C_j is a 3x3 transformation matrix which contains the rotation, scaling and skew components of the deformation and $\mathbf{t}_j$ the translation component. The deformation gradient of the triangle between its status in a pose P_i and a reference pose P_0 is enclosed in the Jacobian matrix:

$$D_p\Phi_j(\mathbf{v}) = C_j \quad \textbf{(2. 61)}$$

Note that the three vertices of the triangle are not enough to describe the deformation towards the direction perpendicular to the triangle. To alleviate this, a fourth vertex is introduced in the direction of the triangle's normal vector, with length proportional to the edges of the triangle [4]:

$$v_4 = v_1 + \frac{(v_2 - v_1)\times(v_3 - v_1)}{\sqrt{\left|(v_2 - v_1)\times(v_3 - v_1)\right|}} \tag{2. 62}$$

The affine transformation of each vertex is then described by:

$$C_j v_k^i + \mathbf{t}_j = v_k^0,\ 1 \le i \le n,\ 1 \le k \le 4 \tag{2. 63}$$

and in matrix form it is given by:

$$\begin{gathered} C[v_2^i - v_1^i \ \ v_3^i - v_1^i \ \ v_4^i - v_1^i] = [v_2^0 - v_1^0 \ \ v_3^0 - v_1^0 \ \ v_4^0 - v_1^0] \Leftrightarrow \\ C = [v_2^0 - v_1^0 \ \ v_3^0 - v_1^0 \ \ v_4^0 - v_1^0][v_2^i - v_1^i \ \ v_3^i - v_1^i \ \ v_4^i - v_1^i]^{-1} \end{gathered} \tag{2. 64}$$

Both rotation and stretch information can be extracted from C by applying *polar decomposition* [6] [7]. Performing thin SVD upon C we get:

$$C = U\Sigma V^T = (UV^T)(V\Sigma V^T) = RS \tag{2. 65}$$

where $R \in \mathbb{R}^{3\times 3}$ is the orthogonal matrix representing the rotational component, and $S \in \mathbb{R}^{3\times 3}$ is a symmetric matrix that applies stretching to the triangle before the rotation [8]. S can be further dissected into a scale (Sc) and a shear (Sh) matrix by extracting the associated matrix elements:

$$\begin{gathered} Sc = diag(S) \\ Sh = S - diag(S) \end{gathered} \tag{2. 66}$$

2.4 Barycentric Coordinate System

A Barycentric coordinate system is a system in which the coordinates of a point are defined with reference to the centre of mass of an object. The point may be located within the area of the object, including its boundary. Due to this notion Barycentric Coordinates are also called Area Coordinates. Barycentric coordinates where introduced by August Ferdinand Möbius in 1827.

2.4.1 Triangle Barycentric Coordinates

Given a triangle T with vertices $[\mathbf{t}_1 \;\; \mathbf{t}_2 \;\; \mathbf{t}_3],\;\; \mathbf{t}_i \in \mathbb{R}^{3\times 3}$, any point $\mathbf{t}$ within the area of this triangle can be described by the weighted sum of these vertices:

$$\mathbf{t} = \lambda_1 \mathbf{t}_1 + \lambda_2 \mathbf{t}_2 + \lambda_3 \mathbf{t}_3 \tag{2. 67}$$

The λ coefficients have the property $\lambda_3 = 1 - \lambda_1 - \lambda_2$ and are called Barycentric coordinates. Figure 2.9 shows how triangle areas can be described using this coordinate system.

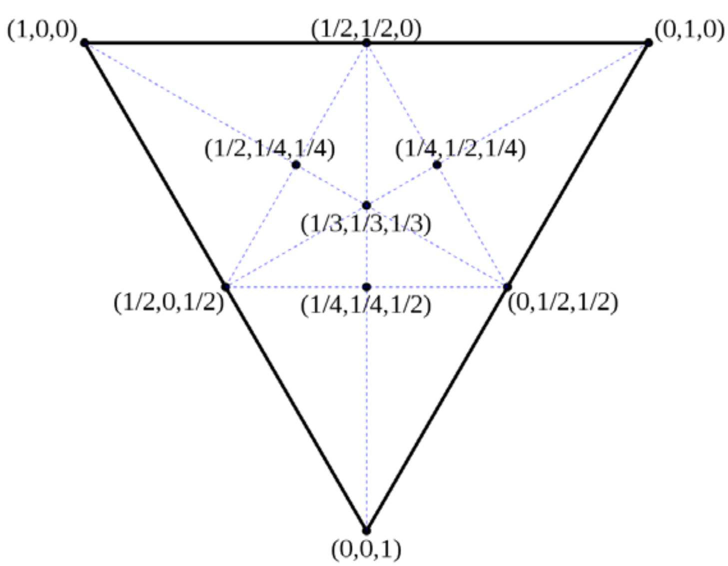


Figure 2.9: Barycentric Coordinate representation in the context of a triangle

Having the Cartesian coordinates of point $\mathbf{t}$, specifying the corresponding Barycentric one is a matter of solving the following system:

$$\begin{aligned} x &= \lambda_1 x_1 + \lambda_2 x_2 + (1-\lambda_1-\lambda_2)x_3 \\ y &= \lambda_1 y_1 + \lambda_2 y_2 + (1-\lambda_1-\lambda_2)y_3 \end{aligned} \tag{2. 68}$$

What (2. 68) describes is a linear transformation that in matrix notation is given by :

$$\begin{aligned} &\begin{pmatrix} x_1 - x_3 & x_2 - x_3 \\ y_1 - y_3 & y_2 - y_3 \end{pmatrix} \begin{pmatrix} \lambda_1 \\ \lambda_2 \end{pmatrix} = \begin{pmatrix} x - x_3 \\ y - y_3 \end{pmatrix} \Leftrightarrow \\ &\mathbf{T\lambda} = \mathbf{t} - \mathbf{t_3} \Leftrightarrow \boldsymbol{\lambda} = \mathbf{T}^{-1}(\mathbf{t} - \mathbf{t_3}) \end{aligned} \tag{2. 69}$$

and shows that specifying the Barycentric coordinates is a matter of inverting the 2x2 matrix $\mathbf{T}$.

2.4.2 Tetrahedron Barycentric Coordinates

Barycentric coordinates can be extended to 3D allowing to specify areas within a 3D simplex (i.e. a tetrahedron). In this context four coordinates $\lambda_1, \lambda_2, \lambda_3, \lambda_4$ must be specified. Extending (2. 69) the resulting system becomes:

$$\mathbf{T} \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = \mathbf{t} - \mathbf{t_4} \tag{2. 70}$$

whose solution is a matter of inverting the 3x3 matrix:

$$\mathbf{T} = \begin{pmatrix} x_1 - x_4 & x_2 - x_4 & x_3 - x_4 \\ y_1 - y_4 & y_2 - y_4 & y_3 - y_4 \\ z_1 - z_4 & z_2 - z_4 & z_3 - z_4 \end{pmatrix} \tag{2. 71}$$

CHAPTER 3

LINEAR BLEND SKINNING

In the following section we will present methods for the acquisition of the bone transformation matrices and then elaborate on methods of weight-influences computation.

3.1 **Skinning with Skeletal Hierarchy**

Approximating an animation sequence to produce a more succinct representation is common in the case of articulated models and is carried out through a process called Skinning. The idea is to approximate the trajectories of vertices based on the trajectories of the bones that influence them. This means that bone-vertex relations need to be established, meaning, which bones affect each vertex and what is the amount of their influence. Figure 3.1 shows an example of a vertex v_i being affected by 2 bones b_1, b_2 on an articulated model part in two different poses $\{p, p+k\}$.

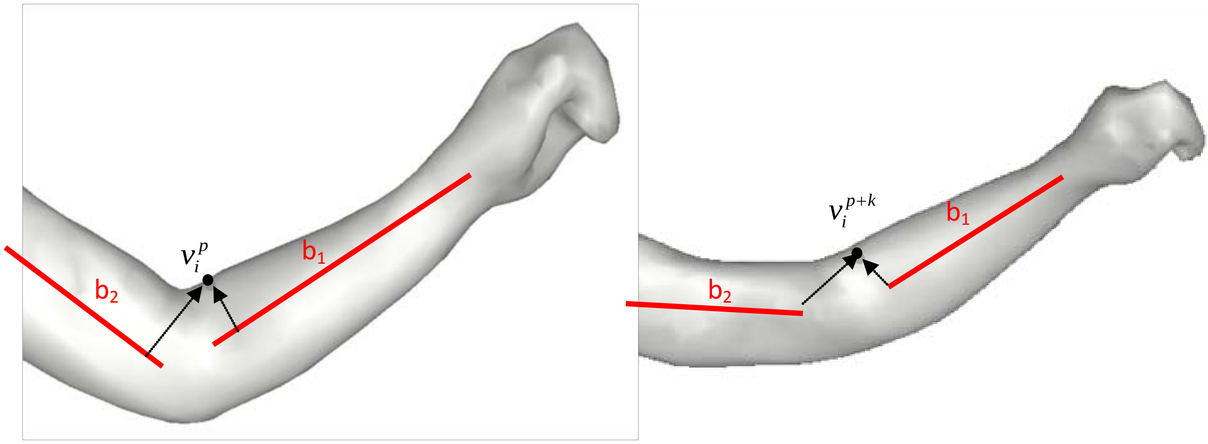


Figure 3. 1: Bone-Vertex influence on articulated model part

Using the linear blend skinning technique, the location of v_i can be approximated by the transformation matrices T_1 and T_2 that describe the movement of b_1 and b_2 weighted by the amount of influence each transformation should have on the vertex. These transformation matrices represent the movement of the bones compared to their location in a reference pose (*rest pose*) of the animation sequence. Usually this rest pose is chosen thus, that the model is depicted in a neutral state. In the case of the elbow example in Figure 3.1 a neutral position suitable for rest pose would be with the elbow straightened. In general, knowing the transformation that each bone undergoes in each pose and the amount of influence of each bone to the vertices of the model, we can approximate the whole animation sequence using only the rest pose.

More formally, we assume that we have an animation sequence of P poses of an articulated model consisting of B bones. The model has N vertices. It is important to state that the connectivity of the vertices throughout the animation sequence is not affected. Neither vertices nor edges are added or removed. We also assume that a skeletal hierarchy has been established on a selected rest pose of the animation sequence and each vertex has been assigned to one or more bones. We denote by T_b^p the transformation matrix that describes the transformation that bone $b \in \{1,...,B\}$ undergoes from the rest pose to pose $p \in \{1,...,P\}$. With w_{ib} we denote the amount of influence of bone b to the vertex with index $i \in \{1,...,N\}$. In the rest of this thesis we shall refer to this quantity as *weight-influence* or simply *weight*, of a bone to a vertex. Finally we denote by $v_i'^p \in \mathbb{R}^3$ the approximation of vertex i in pose p and by $\mathbf{v}_i \in \mathbb{R}^3$ the coordinates of vertex i in the rest pose. Using linear blend skinning the approximation of v_i^p is given by:

$$v_i'^p = \left(\sum_{b=1}^{B} w_{ib} T_b^p \right) \mathbf{v}_i \tag{3. 1}$$

In the formula above if a bone does not affect a vertex, w_{ib} is zero.

Weight influences are established in the rest pose and remain the same throughout the animation regardless of the pose we are approximating. Additionally, for each vertex the sum of the weights of the influencing bones is 1 and each weight-influence is non-negative, i.e.:

$$\sum_{b=1}^{B} w_{ib} = 1 \tag{3. 2}$$

and

$$w_{ib} \geq 0 \tag{3. 3}$$

Finally, it has been proved in practice that 4 bone influences per vertex are adequate for a good approximation. More than 4 weight-influences definitely add more detail but not enough to compensate for the increased complexity of weight acquisition.

In its entirety the approximation procedure is described by the following algorithm:

Algorithm 3.1 Approximate_SequencePerPose(AnimationSequence[])

1. Bone_L := GetBoneList(AnimationSequence[]);
2.WeightInfluences_L :=GetWeightInfluenceList(AnimationSequence[],Bone_L);
3.**for each** pose **in** AnimationSequence **do**
4. T[pose]:=ComputeTMatrices(AnimationSequence[Pose], Bone_L, WeightInfluences_L);
5. ApproximatedSequence[pose]=PerformLBS(T[Pose],AnimationSequence[Pose],Bone_L, WeightInfluences_L);

Algorithm 3. 1: Animation Sequence Approximation Procedure

Lines 1, 2 and 4 comprise the part of the process called *fitting*. Determining the bone number and location is referred to as *bone-fitting*, while determining the weight-influences is referred to as *weight-fitting*. Line 5 is the application of (3. 1) to produce the approximation.

3.2 Skinning Highly Deformable Models

The existence of a skeleton defines degrees of freedom on the vertex movement. For example in the case of the arm in Figure 3.1, although high deformation is expected in the area of the elbow, the majority of the vertices between the elbow and the wrist will follow the movement of the bone. Normally a bone is not supposed to change its shape and thus its movement is that of rigid body and so is the movement of the affected bone. This is not the case in highly deformable objects, i.e. models with no skeletal hierarchy established. There are no directives under which a vertex may move. Figure 3.2 shows selected frames from an animation sequence of a highly deformable object. Notice that all areas, except from the rightmost which is the area of the flag that adheres to the pole, are subject to random deformation.

Figure 3. 2: Highly deformable animation of a flag under the influence of wind

Nevertheless, following certain adaptations, the concept of skinning can be extended to highly deformable models. In fact in the following sections, when talking about highly deformable models, we shall regularly transform the problem to articulated model simply to reinforce our intuition. One of the adaptations mentioned, is to consider affine transformation matrices instead of rigid, to facilitate capturing of possible scaling and shearing deformations. Adaptations must also be made to the methods of the weight-influences specification, a task quite elaborate and less well established than transformation theory.

3.3 **Transformation Matrix Fitting**

We will assume for now that bone structure and weight-influences have been established in some way. The next step in the LBS algorithm is to compute the transformation matrices that describe the transformations that bones undergo throughout the animation. As stated before, describing the movement of highly deformable objects requires the use of affine transformation matrices, to capture deformations other than rotation and translation. Rigid body motion can be used but not without penalty in the quality of the approximation, since there is no guarantee that the deformation of a vertex is purely rigid.

3.3.1 Fitting with Affine Transformation Matrices

To formulate the problem of approximation, if v_i^p is the actual vertex coordinates in pose p (i.e. the value we are trying to approximate) and $v_i'^p$ is the approximation produced by implementing the LBS algorithm, we are trying to minimize the quantity:

$$\sum_{i=1}^{N} \left\| v_i'^p - v_i^p \right\|^2 \tag{3. 4}$$

namely the error of each vertex approximation.

As Algorithm 3. 1 states this operation is performed in a per pose basis. With reference to [9], this is equivalent to the least squares solution of the system:

$$v_i'^p = \sum_{b=1}^{B} w_{ib} T_b \mathbf{v}_i \tag{3. 5}$$

for each vertex of the model.

The above is a linear system of 3N equations, the unknowns of which are the (3x4) elements of the transformation matrices of each bone. This sums to 12B unknowns. The system can be expressed in a $Ax = \mathbf{b}$ form, where A is a 3Nx12B known matrix constructed by combining the rest-pose vertex positions and the corresponding vertex weights. More specifically the first 3 rows of matrix A which refer to the first vertex to be approximated are as follows:

$$A(1:3,:) = \begin{bmatrix} \overbrace{w_{11}\mathbf{v}_{1x} \quad w_{11}\mathbf{v}_{1y} \quad w_{11}\mathbf{v}_{1z} \quad w_{11}}^{b=1} & & \overbrace{w_{1B}\mathbf{v}_{1x} \quad w_{1B}\mathbf{v}_{1y} \quad w_{1B}\mathbf{v}_{1z} \quad w_{1B}}^{b=B} \\ w_{11}\mathbf{v}_{1x} \quad w_{11}\mathbf{v}_{1y} \quad w_{11}\mathbf{v}_{1z} \quad w_{11} & \ldots & w_{1B}\mathbf{v}_{1x} \quad w_{1B}\mathbf{v}_{1y} \quad w_{1B}\mathbf{v}_{1z} \quad w_{1B} \\ w_{11}\mathbf{v}_{1x} \quad w_{11}\mathbf{v}_{1y} \quad w_{11}\mathbf{v}_{1z} \quad w_{11} & & w_{1B}\mathbf{v}_{1x} \quad w_{1B}\mathbf{v}_{1y} \quad w_{1B}\mathbf{v}_{1z} \quad w_{1B} \end{bmatrix} \tag{3. 6}$$

$\mathbf{b}$ is a 3N known vector where the actual vertex coordinates are stacked:

$$\mathbf{b} = \begin{bmatrix} v_{1x}^{p} \\ v_{1y}^{p} \\ v_{1z}^{p} \\ \vdots \\ v_{Nx}^{p} \\ v_{Ny}^{p} \\ v_{Nz}^{p} \end{bmatrix} \tag{3. 7}$$

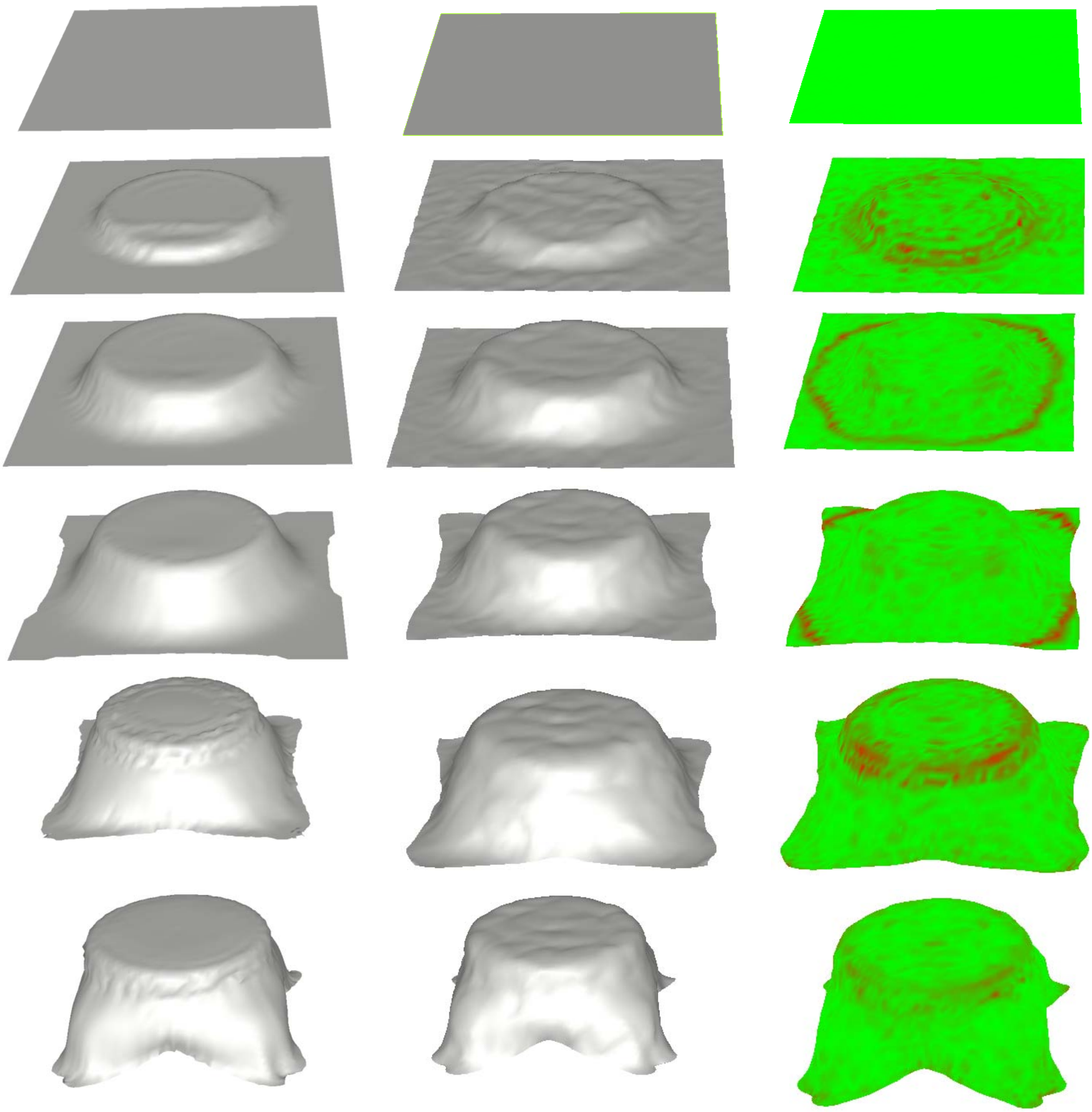
Transformation matrices are of the form:

$$T_b = \begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3. 8}$$

Since the last row is not needed in solving the system, x is a 12B vector of unknowns of the form:

$$x = \begin{bmatrix} \left.\begin{matrix} a_{11}^{1} \\ a_{12}^{1} \\ \vdots \\ a_{34}^{1} \end{matrix}\right\} b = 1 \\ \left.\begin{matrix} a_{11}^{B} \\ a_{12}^{B} \\ \vdots \\ a_{34}^{B} \end{matrix}\right\} b = B \end{bmatrix} \tag{3. 9}$$

Recall that each vertex is not influenced by all bones, in which case the weight is 0 along with the corresponding 12 elements of matrix A. Also recall that to each vertex usually correspond no more than 4 bones. This means matrix A can be highly sparse. To exploit this sparsity for the solution of the system we utilize the LSQR [10] algorithm. Figure 3. 3 depicts the results of the method.



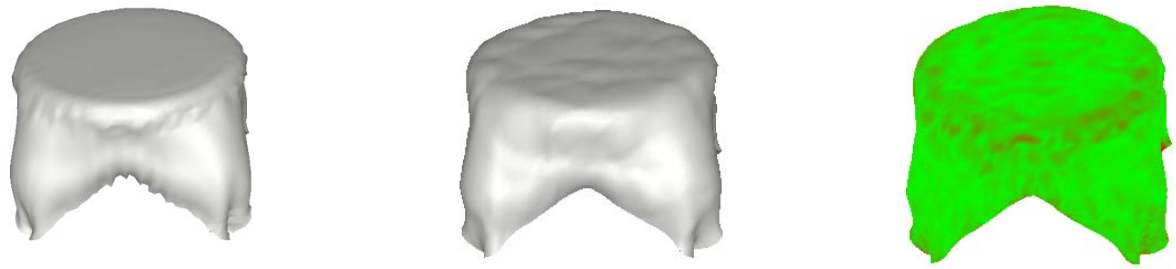


Figure 3. 3 : Results of Linear Blend Skinning using Affine Fitting, PF=1.6, a) Original animation, b) Approximation using Affine, P-Center based fitting, c) Approximation error distribution. Red areas denote high error of approximation

3.3.2 Fitting with Dual Quaternions

Even with the boost gained by LSQR algorithm, the transformation matrix computation is very time consuming and acts as a bottleneck to the algorithm of LBS as a whole. As shown in Figure 3. 4, meddling with the convergence tolerance error or the number of iterations LSQR is to perform, has significant effect upon the quality of the approximation. Solving time could be improved if there was a way to reduce the number of unknowns. Kavan et.al [9] suggested the use of dual quaternions to describe the motion of bones. Recall from (2. 37) that a 3D point can be represented by a dual quaternion. To the existing formulation we add the use of "^" to denote a quantity expressed as a dual quaternion. Thus $\hat{\mathbf{v}}_i$ is a rest pose vertex expressed as a dual quaternion, $\hat{v}_i^p$ is the actual coordinates of a vertex in a pose and $\hat{v}_i'^p$ is the approximation coordinates of that vertex. We denote by $\hat{q}_b$ the dual quaternion that represents the transformation of bone b. Recall from (2. 48) that given a transformation described by a dual quaternion $\hat{q}$, applying this transformation on a vertex, also expressed as a dual quaternion $\hat{p}$, is done by multiplying $\hat{p}$ from the right $\hat{q}$ and from the left with the conjugate inverse of $\hat{q}$. Using this notation (3.5) becomes:

$$\hat{v}_i'^p = \left(\sum_{i=1}^{N} w_{ib} \hat{q}_b \right) \hat{\mathbf{v}}_i \left(\overline{\sum_{i=1}^{N} w_{ib} \hat{q}_b} \right)^{-1} \tag{3. 10}$$

Multiplying with $\left(\overline{\sum_{i=1}^{N} w_{ib}\hat{q}_b}\right)$ from the left to remove the inverse we obtain:

$$\hat{v}_i'^{\,p}\left(\overline{\sum_{i=1}^{N} w_{ib}\hat{q}_b}\right)-\left(\sum_{i=1}^{N} w_{ib}\hat{q}_b\right)\hat{\mathbf{v}}_i = 0 \qquad \textbf{(3. 11)}$$

The resulting dual quaternions however must be invertible for $\left(\overline{\sum_{i=1}^{N} w_{ib}\hat{q}_b}\right)^{-1}$ to exist which means that at least one element must be non-zero, otherwise (2. 42) does not hold. To ensure this we force the real component of the non-dual part q_w of the quaternion to 1. This rule does not affect the range of transformations this method can describe since any dual quaternion represents exactly the same transformation with its real multiple

Thus we transform the problem of finding 12 element transformation matrices to finding 7 (8 minus the one set to 1) element Dual quaternions. Following a series of substitutions and calculations, (3. 11) can be simplified in the form of $\hat{A}\hat{x} = \hat{\mathbf{b}}$. Matrix $\hat{A}$ is a 4Nx7B one whose first 4 rows and 7 columns (i.e. the coefficients of the influence of the first vertex by the first bone) are described as follows:

$$A(1:4,1:7)=\left[\overbrace{\begin{matrix} w_{11}(v_{1x}^{p}-\mathbf{v}_{1x}) & w_{11}(v_{1y}^{p}-\mathbf{v}_{1y}) & w_{11}(v_{1z}^{p}-\mathbf{v}_{1z}) & 2w_{11} & 0 & 0 & 0 \\ 0 & w_{11}(v_{1z}^{p}+\mathbf{v}_{1z}) & -w_{11}(v_{1y}^{p}+\mathbf{v}_{1y}) & 0 & 2w_{11} & 0 & 0 \\ -w_{11}(v_{1z}^{p}+\mathbf{v}_{1z}) & 0 & w_{11}(v_{1x}^{p}+\mathbf{v}_{1x}) & 0 & 0 & 2w_{11} & 0 \\ w_{11}(v_{1y}^{p}+\mathbf{v}_{1y}) & -w_{11}(v_{1x}^{p}+\mathbf{v}_{1x}) & 0 & 0 & 0 & 0 & 2w_{11} \end{matrix}}^{b=1}\right] \qquad \textbf{(3. 12)}$$

The rest of the matrix is filled in a similar fashion. Vector $\hat{\mathbf{b}}$ is a 4Nx1 comprising of the rest pose vertices, the actual vertex positions and the sum of all the weights applied on each vertex. The first 4 rows of $\hat{\mathbf{b}}$, i.e the right part of the system for the first vertex, are:

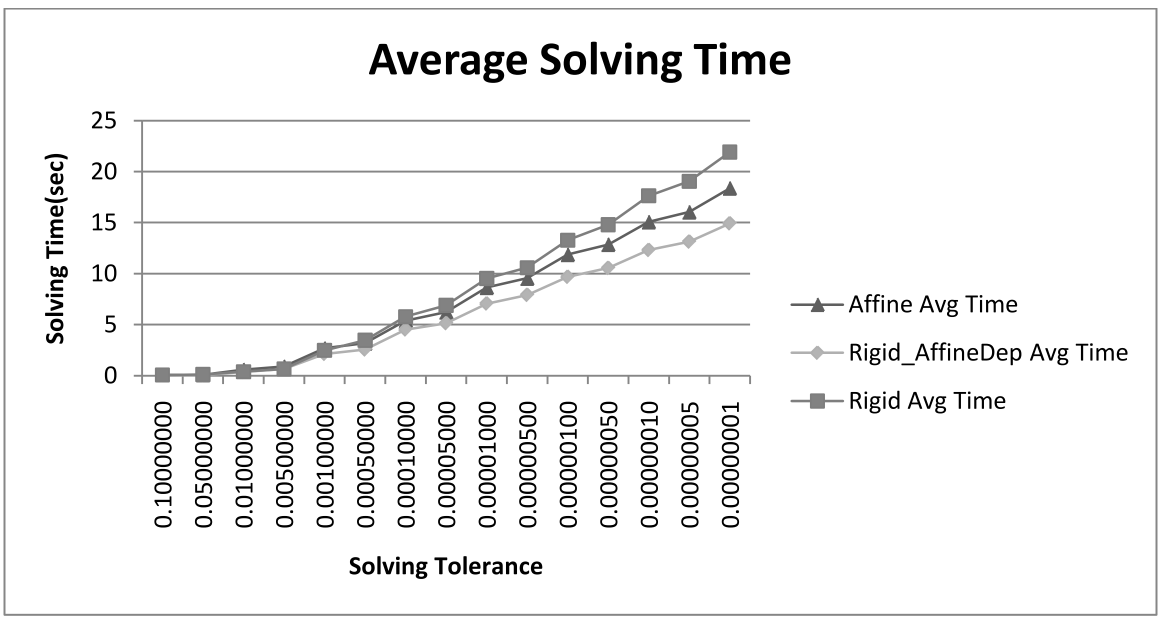
$$\hat{\mathbf{b}} = \left[0 \quad \sum_{b=1}^{B} w_{1b}(v_{1x}^{p}-\mathbf{v}_{1x}) \quad \sum_{b=1}^{B} w_{1b}(v_{1y}^{p}-\mathbf{v}_{1y}) \quad \sum_{b=1}^{B} w_{1b}(v_{1z}^{p}-\mathbf{v}_{1z})\cdots\right]^{T} \qquad \textbf{(3. 13)}$$

and the rest of the matrix is filled in the same fashion.

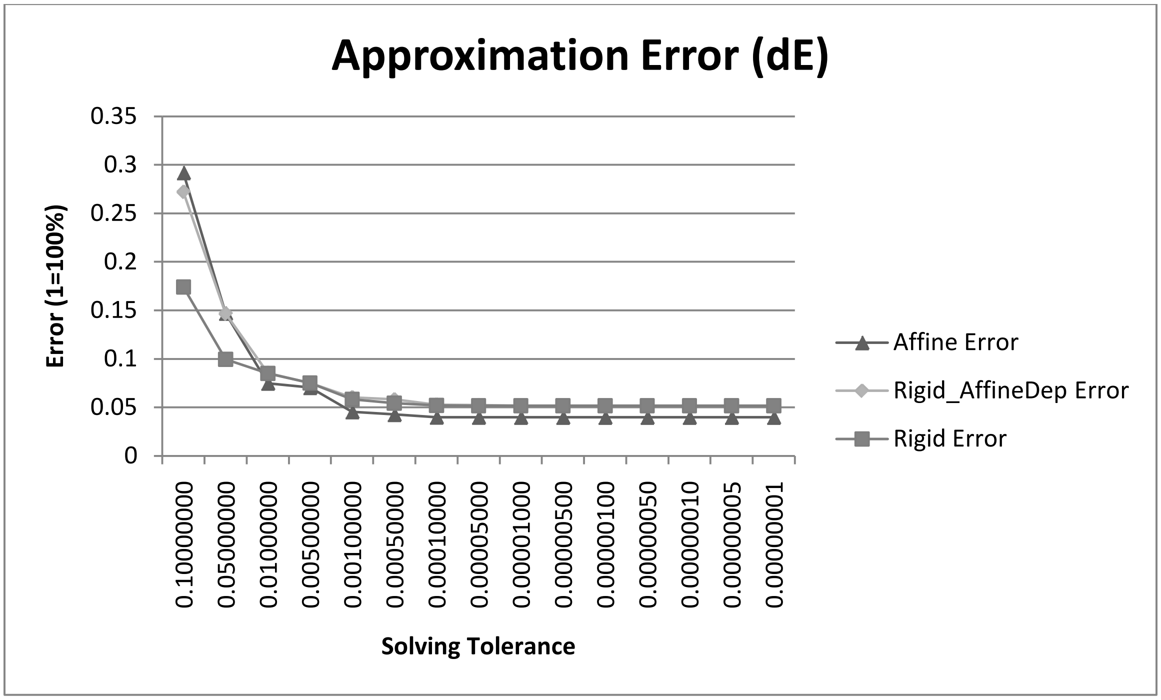
Finally matrix x is a 7Bx1 matrix of unknown dual quaternion elements stacked:

$$x = \left[\overbrace{q_{0x}^{1} \quad q_{0y}^{1} \quad q_{0z}^{1} \quad q_{\varepsilon w}^{1} \quad q_{\varepsilon x}^{1} \quad q_{\varepsilon y}^{1} \quad q_{\varepsilon z}^{1}}^{b=1} \cdots \overbrace{q_{0x}^{B} \quad q_{0y}^{B} \quad q_{0z}^{B} \quad q_{\varepsilon w}^{B} \quad q_{\varepsilon x}^{B} \quad q_{\varepsilon y}^{B} \quad q_{\varepsilon z}^{B}}^{b=B} \right]^{T} \quad \textbf{(3. 14)}$$

Dual Quaternions are capable of describing rigid body motion. This means that an error in the approximation is to be expected. However as shown in Figure 3. 4 the reduction in the time required for transformation matrix calculation, due to the reduction of unknowns in the system, is significant. We also see that dual quaternion fitting can be slower than affine if we don’t set a limit to the number of iterations the LSQR must perform in the process of achieving the required tolerance.

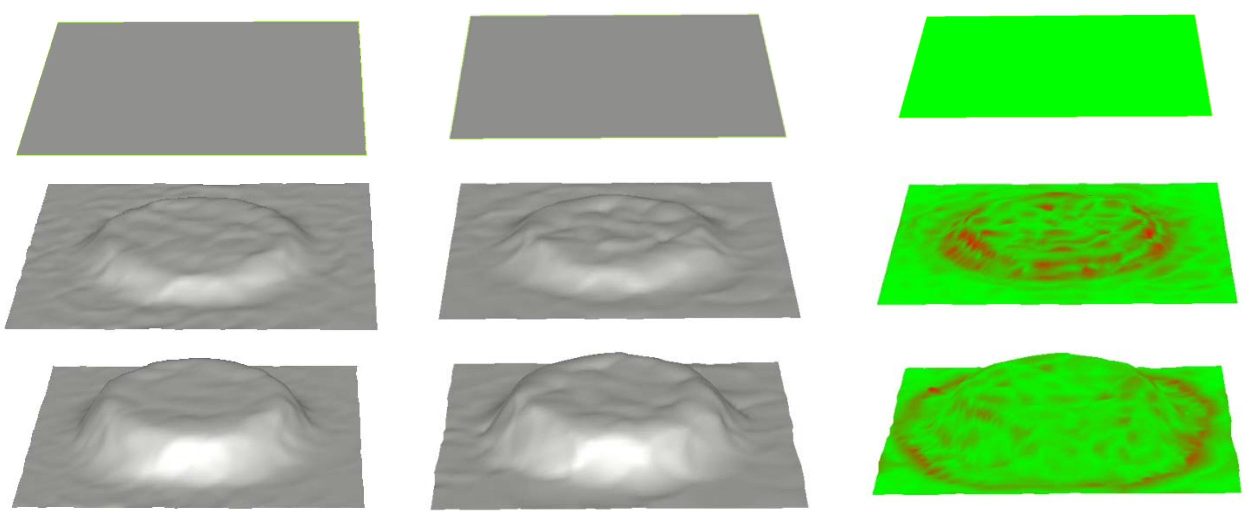


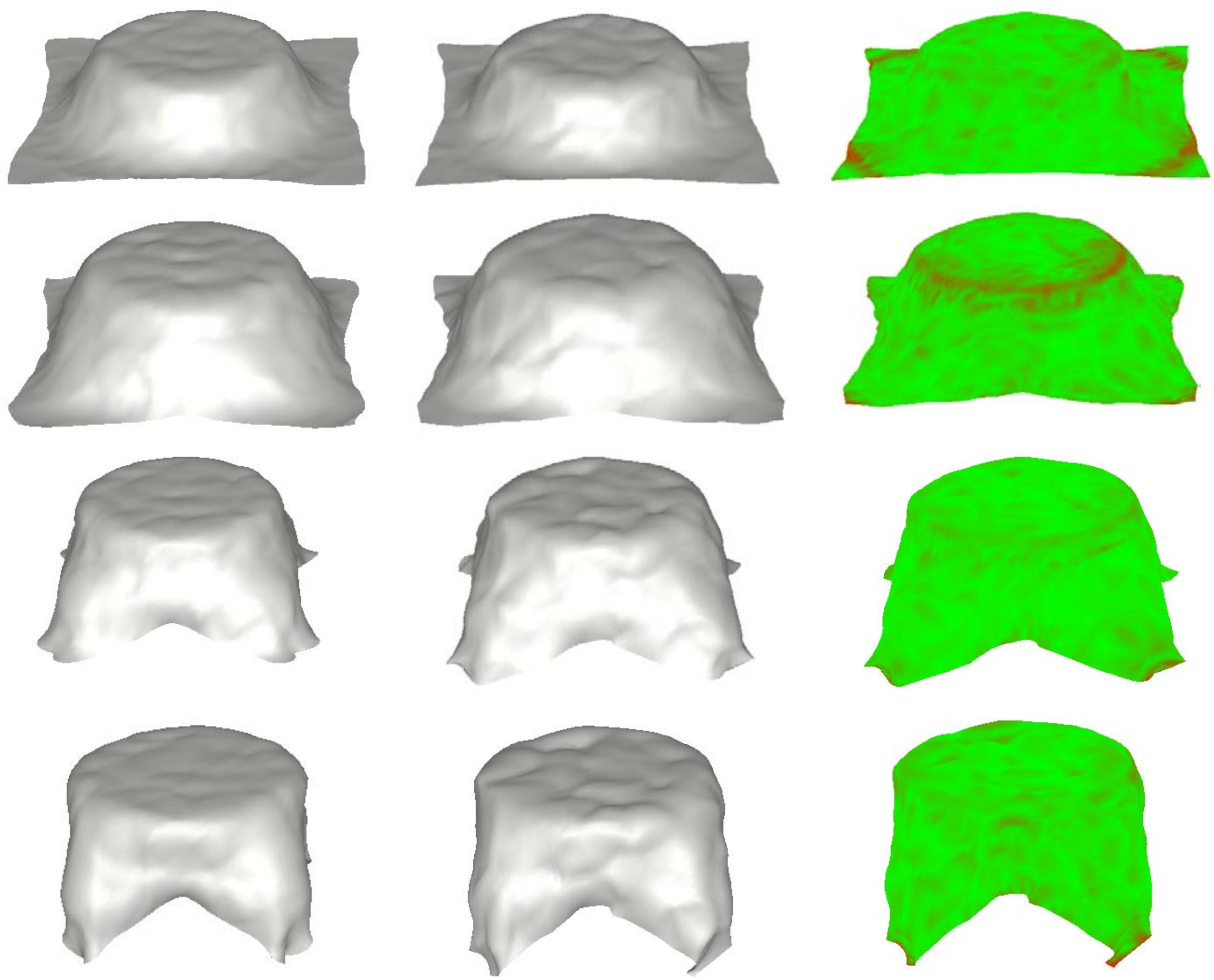
(a)



(b)

Figure 3. 4: a) The Average solving time and b) Approximation error of i) Affine Fitting, ii) Rigid Fitting with the number of LSQR Iterations Equal to that of Affine, iii) Rigid fitting with unlimited number of iterations. Horizontal axis is LSQR error Tolerance.





(a) (b) (c)

Figure 3. 5: a) Affine Fitting, b) Rigid Fitting, c) Error Distribution

3.4 Bone and Weight Fitting

Bone and weight-influences specification is fundamental to the skinning process. The choices made in this stage will greatly define the final result. We describe this process first by defining the transition from skeleton bones for articulated models to proxy bones for highly deformable ones. Then we elaborate on the principles of efficient bone and weight-influences fitting. Finally we mention the existing policy and the one we propose for bone and weight fitting.

3.4.1 Moving from Bones to Proxy Joints

One major difference when comparing articulated and highly deformable objects is that in a highly deformable object, not only it is difficult to specify a skeletal hierarchy but doing so may also impose undesired limitations. Consider for example the case of clothing an articulated model. It is desirable for the cloth to present with a deformable behavior and not be constrained by the rigidity of the movement of the bone it covers. Kavan et.al [9] suggested that selected vertices on the rest pose can act as bones (*proxy joints*). Each proxy joint will create an area of influence and each vertex within this area is assigned a weight according to some criterion. This criterion is usually based on the distance of the vertex from the influencing proxy joint. Figure 3.3 depicts this idea. The red dots underline the vertices that act as proxy joints and the contrast of the color denotes areas of different influence.

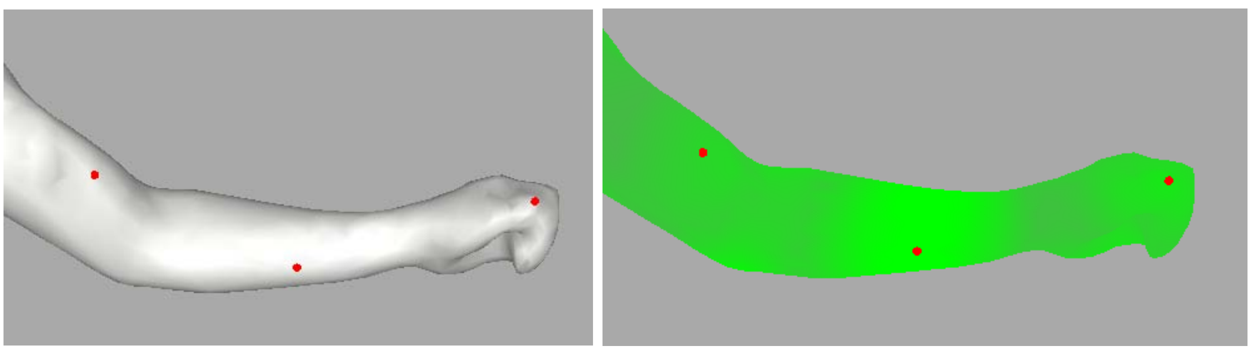


Figure 3. 6: Proxy joints instead of bones, and weight influences per proxy joint. The brightness of the color denotes the intensity of the influence.

The replacement of a bone structure by proxy joint vertices, distributed according to some pattern upon the model, appears to offer the flexibility and the control over the surface required for describing a highly deformable animation. The distribution pattern, along with a weight assignment policy will define the areas of influence and ultimately the quality of the approximation, thus both must be chosen carefully.

3.4.2 Principles of Efficient Proxy Joint and Weight Specification

Understanding how proxy bones and weight influences affect the approximation process is essential for establishing a specification that will lead to a quality result. Intuitively, proxy bones act as attractors and the weights represent the intensity of their attraction. Each vertex can be attracted by different bones and each bone attracts each vertex in its area of influence with different intensity. This presents several caveats when distributing proxy bones and assigning weight-influences.

Misfitting

The first is the possibility of assigning a vertex to a proxy bone whose movement is different. Figure 3.4 demonstrates this scenario. Vertex v_i is assigned to proxy bones whose movement is not representative to that of the vertex. The proxy bone movement affects (by some weight) the movement of the vertex and the result is a spike where it shouldn't be.

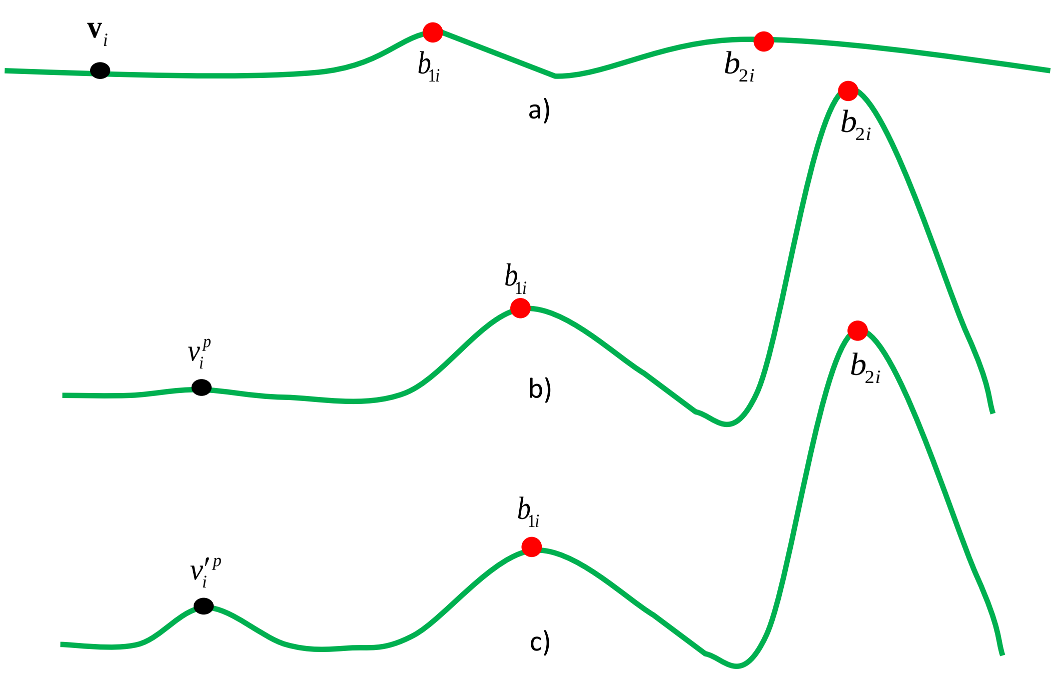


Figure 3. 7: Approximation error by assigning a vertex to irrelevant proxy bones. a) Rest pose, b) Actual pose p, c) Approximated pose p. Vertex $v_i'^p$ is elevated due to elevation of its proxy bones.

The above scenario exhibits the importance both of the proxy joint placement and weight assignment policy. Placing another proxy bone in the vicinity of the vertex and enforcing a proximity criterion for the intensity of the other two bone influences could alleviate the error. It would introduce a stronger attractor whose deformation would also be more representative of the deformation around the vertex. That would diminish the influence of the other joints (recall that weight influences are convex).

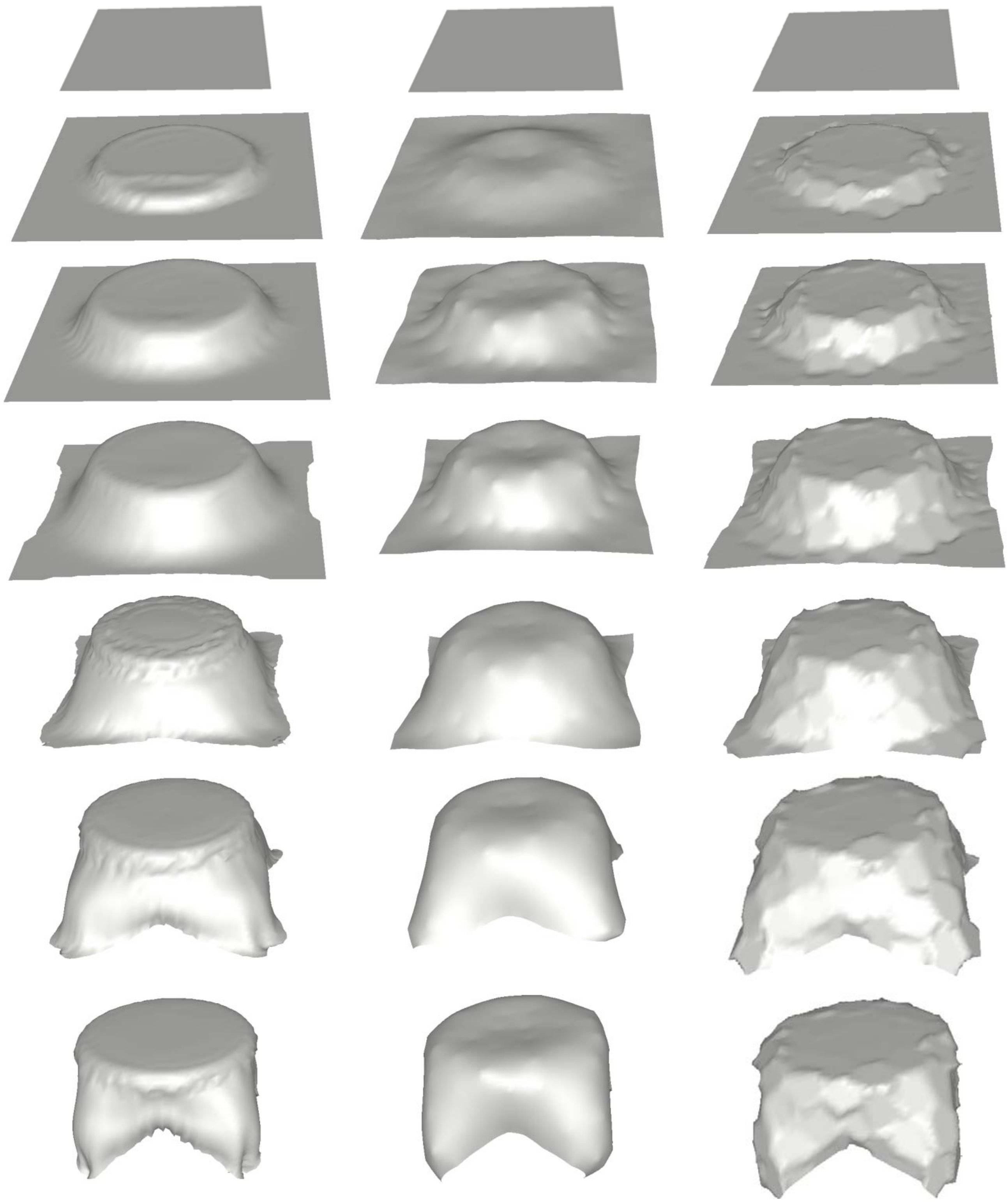
Over-Fitting

Excess of influences, may lead to the second caveat, that of *over-fitting*. Too many attractions will result in area averaging and ultimately to extreme loss of detail. Figure 3. 8(b) shows the effects of over-fitting, where the model appears rounded. Notice the bumps on the surface that imply the location of proxy joints. The influence each joint inflicts upon its immediate area is stronger and it forces this area to follow its movement. Another implication of over-fitting is the increase of the solving time for the transformation matrices, since more non –zero entries are introduced in the A matrix (3. 6).

Under-fitting

Lack of influence is the third caveat with the worst scenario being a vertex to be influenced by one proxy joint only. This will force the vertices to follow the movement of the proxy joint as if performing rigid body motion. The result is indeed rigidity in the areas of influence of each proxy joint as is demonstrated in Figure 3. 8(c).

For and efficient bone and weight fitting, these limitations need to be compromised. Even so there are no exact rules on how fitting is to be carried out. It mostly depends on the nature of the deformations throughout the animation sequence. Intuitively the more proxy joints present, with limited areas of influence to avoid over-fitting, the better the representation of deformations and thus the approximation. However the number of proxy bones directly affects the size of the system producing the transformation matrices (3. 6). Adding more bones means increase in the approximation time.



a) b) c)

Figure 3. 8: Tablecloth animation sequence. Bone and weight fitting caveats. a) Actual animation, b) Over-fitted approximation, c) Under-fitted (rigid) approximation.

3.4.3 Uniform Proxy Joint Distribution using P-Center Clustering

One way to ensure that the surface gets enough coverage from proxy joints is to distribute them uniformly and adjust the influence areas accordingly. This can be done utilizing the p-center algorithm [12]. In brief, this algorithm assigns a new proxy joint on the model by creating a new cluster in each step whose center is the vertex with the largest among the distances, of the vertices furthest from the center of their clusters. Upon creation each cluster claims from the existing clusters all vertices closest to its center. The algorithm repeats until the required number of proxy joints has been assigned. The distance metric used is the Euclidian.

Algorithm 3.2 P-CenterClustering(AnimationSequence[0], k)

```
1.VertexIndex:= PickRandomVertexIndex(AnimationSequence[0]);
2.while ProxyJointsSet != k
3.      NewClusterCenter = FindMostDistantVertex(ClusterList);
4.      NewCluster = AssignVerticesToNewCluster(NewClusterCenter);
5.      ClusterList.Add(NewCluster);
6.      ProxyJointsSet= ProxyJointsSet+1;
```

Algorithm 3. 2: P-Center Clustering

Note that, since in each step the furthest vertex is extracted, in the very first steps the algorithm tends to assign centers to the borders of the model and gradually moves to the interior. This behavior will come of use in a later section during the initialization of the Variation Region Growing algorithm.

The influence of each cluster is not limited only to the vertices it contains. That would lead to under-fitting. Instead each cluster defines a cyclic area of influence with radius r_{pj} equal to the distance D_{pj} of its furthest from the joint vertex. These influence areas are further enhanced by a factor $P \geq 1$. Figure 3.9 depicts a scenario of proxy joint placement using P-Center clustering. The dotted circle around each proxy joint is its area of influence without the P-Factor and the green dashed line is the area of influence including the P-Factor.

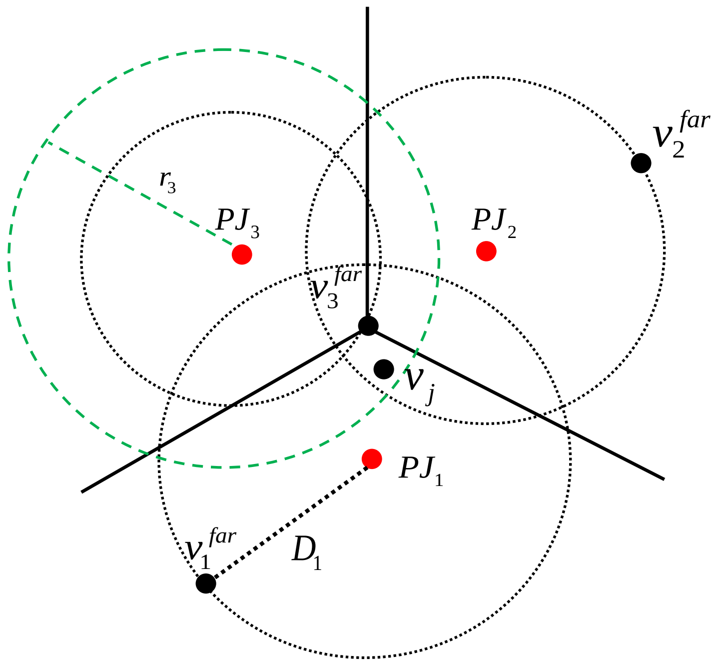


Figure 3. 9: Proxy joint influence areas and the effect of P factor in P-Center clustering

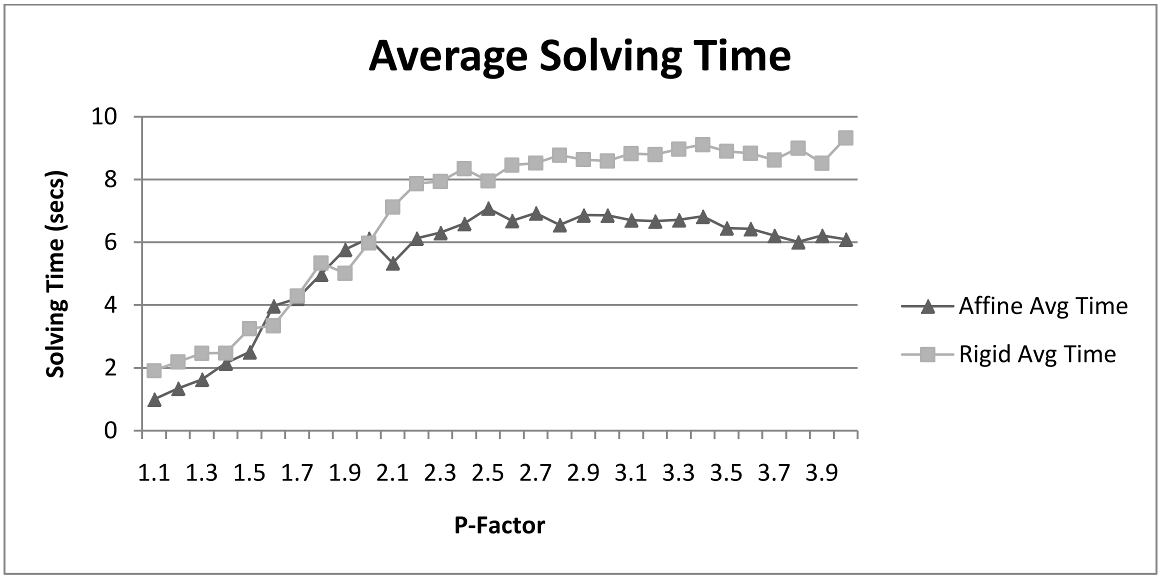
Recall that each time a new cluster center (proxy joint) is defined it assimilates all vertices around it which are closer than they are from the other cluster centers. Intuitively a line is drawn perpendicular to the middle of the line that connects the new center with each old one. Vertices are assigned according to which side of that line they are. Thus each cluster ends up being a polygon with a number of edges equal to the number of surrounding clusters. The weight influence on a vertex depends on its distance d_{ib} from the proxy joint b that influences it:

$$w_{ib} = 1.0 - \frac{d_{ib}}{r_b}, \quad r_b = \max_i(d_{ib}) \cdot PFactor \tag{3. 15}$$

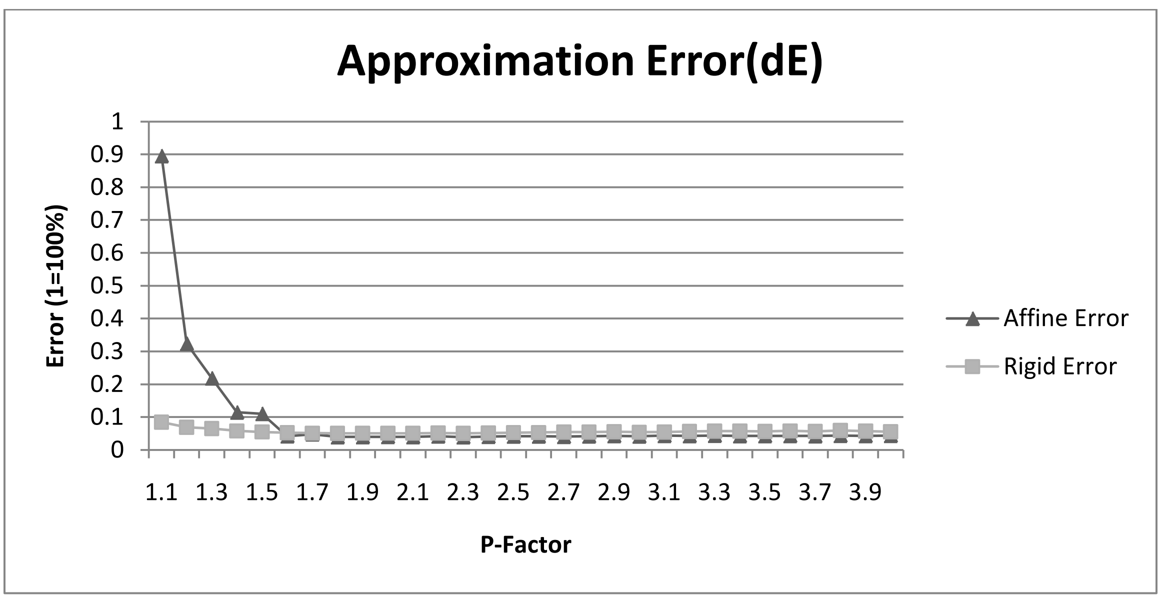
The result needs to be convex so we normalize it by dividing by the sum of all weight influences on this vertex. If only one proxy joint affects the vertex, then its weight becomes 1. In the case of vertex v_j in Figure 3.9 which is affected by three proxy joints the weights are:

$$w_{j1} = \frac{1.0 - \frac{d_{j1}}{r_1}}{\sum_b w_{jb}}, \quad w_{j2} = \frac{1.0 - \frac{d_{j2}}{r_2}}{\sum_b w_{jb}}, \quad w_{j3} = \frac{1.0 - \frac{d_{j3}}{r_3}}{\sum_b w_{jb}} \tag{3. 16}$$

The penetration of the area of influence of one cluster to another depends on where its most distance vertex is. The addition of P-Factor increases this penetration so P-Factor must be chosen carefully to avoid overfitting. Figure 3. 10 shows that, as the P-Factor increases so does the execution time. The approximation error reduces but from a certain point and after it is stabilized or even increases. Thus there is no point in setting very high P-Factor values.

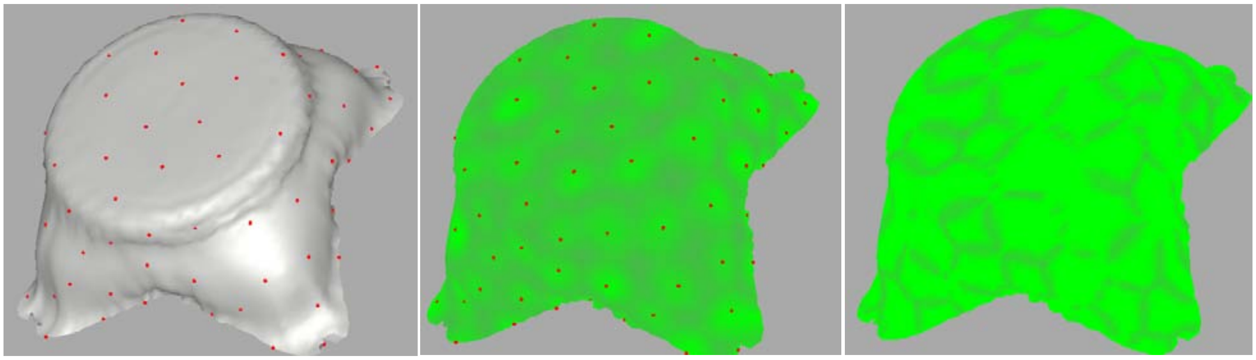


(a)



(b)

Figure 3. 10: a) How transformation fitting execution time progresses as P-Factor changes, b) How the change in P-Factor affects the approximation error. Tests were run on Tablecloth sequence with TOL=0.00005.



a) b) c)

Figure 3. 11: Proxy joint and weight fitting using P-Center clustering. a) Proxy joints uniformly distributed over the area of the model, b) Areas of influence with P-Factor=1.6, c) Areas of influence with P-Factor=1.0

P-Center based distribution yields satisfactory results with low approximation error. However the effect (bumps on the surface) of the spherical influence area upon the model is evident. Furthermore this distribution is bound to assign proxy joints to areas with low deformation where they will be of no actual use and could possibly be approximated by smaller amount of proxy bones. On the other hand it may assign less proxy joints than necessary in areas with high deformation. For example in our tablecloth model we would like less proxy joints on the area that adheres to the table and has low deformation and more proxy joints on the rest of the areas that present with higher deformation.

3.4.4 Deformation Gradient Based Bone and Weight Fitting

We propose a method of bone and weight fitting based on the deformation of the model throughout the animation. Our goal is to identify areas of characteristic deformation and distribute the proxy joints accordingly. A mean is required to quantify the deformation and we have chosen that mean to be *Deformation Gradient*. Deformation gradient is computed per facet and encloses the quantity of the facets rotation, scaling and shearing. In section 2.3.2 we saw that deformation gradient is embedded in three matrices, each describing a different type of deformation. To extract the exact magnitude of each deformation type from their containing matrices, we use the Frobenius norm, i.e. we compute the magnitude of each matrix.

Deformation Quantification and Decomposition

In an animation sequence of a triangulated mesh the deformation gradient at each pose can be computed by comparing the deformation of a pose with that of its preceding or with that of a specified reference pose. Then the deformation gradient of each pose is averaged to produce the deformation gradient of the sequence. We have chosen to perform comparison with a reference pose, and in particular with the rest pose of the animation sequence. The reason is that we want to capture all deformations throughout the animation. In per pose basis the majority of the deformations have very small values and averaging them at the end results in loss of information. For example a facet at some point may have presented with a high deformation and remained idle for the rest of the sequence. That would dominate over continuous but smaller deformations of other facets.

More formally, we denote by $d^p_{R,i}$ the rotational deformation of facet i in pose p of the mesh, by $d^p_{Sc,i}$ the scaling deformation, by $d^p_{Sh,i}$ the shearing deformation and by $d^p_{\Sigma,i}$ the sum of these deformation values.

Rotational component is extracted from the rotation matrix R by first computing the rotation axis l as the solution to the following system:

$$Rl = l \tag{3. 17}$$

and then computing the angle of rotation around this axis:

$$d^p_{R,i} = \angle(u, Ru) \tag{3. 18}$$

where u is the vector perpendicular to l.

Scaling and shearing deformation components are given by:

$$d^p_{Sc,i} = \left\| Sc^p_i \right\|_F, \; d^p_{Sh,i} = \left\| Sh^p_i \right\|_F \tag{3. 19}$$

The sum of the deformation is then given by:

$$d^p_{\Sigma,i} = d^p_{R,i} + d^p_{Sc,i} + d^p_{Sh,i} \tag{3. 20}$$

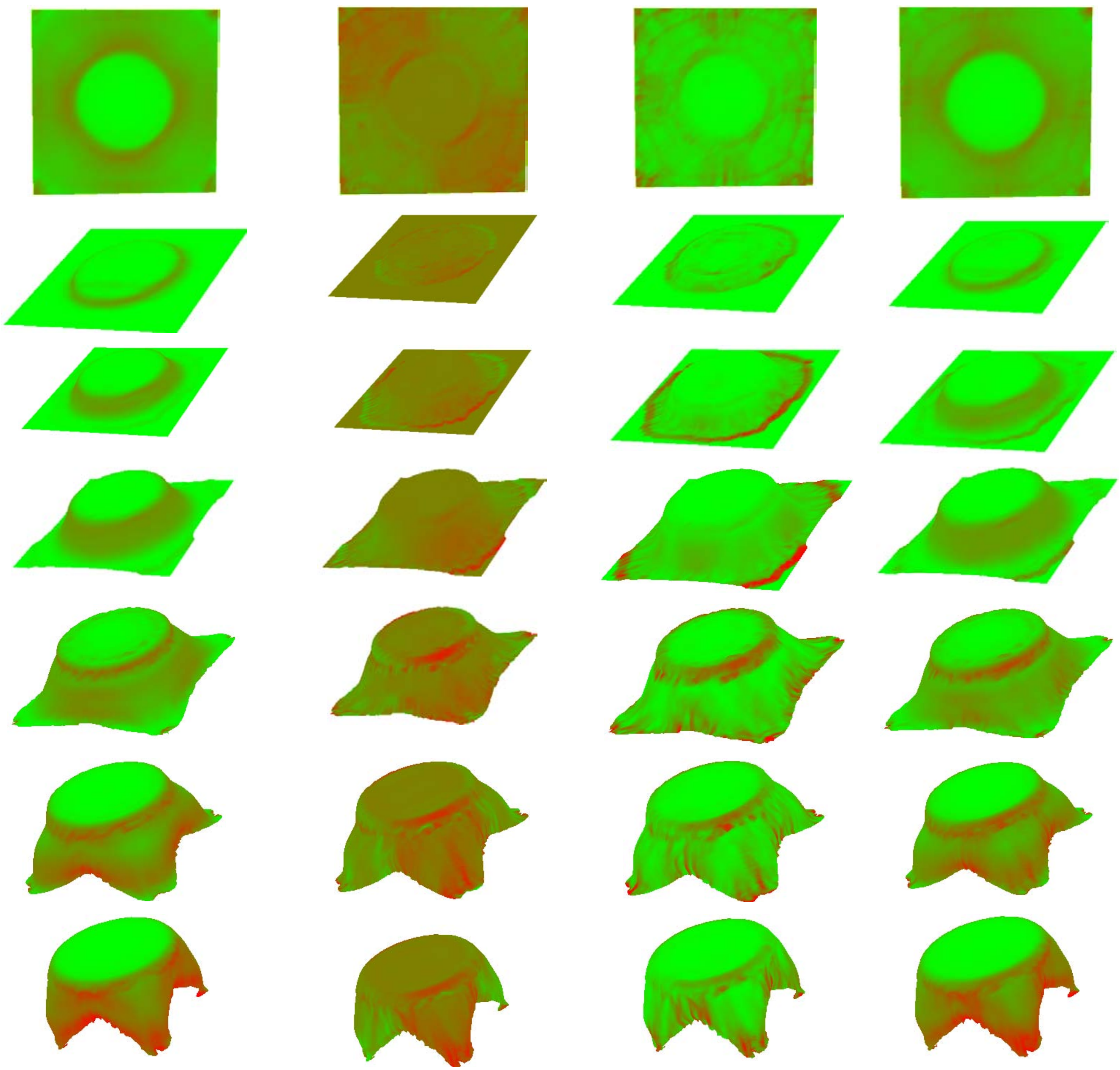
This quantity contains all three deformation types and actually is the Deformation Gradient of the facet

By $D_{\{R,Sc,Sh,\Sigma\},i}$ we denote the average deformation of facet i, for each type, throughout the sequence

$$D_{R,i} = \underset{p}{avg}(d^p_{R,i}),\ D_{Sc,i} = \underset{p}{avg}(d^p_{Sc,i}),\ D_{Sh,i} = \underset{p}{avg}(d^p_{Sh,i}),\ D_{\Sigma,i} = \underset{p}{avg}(d^p_{\Sigma,i}) \tag{3. 21}$$

Figure 3. 12 depicts each deformation component the tablecloth animation sequence. The rest pose contains no deformation data since it is the reference pose (deformation is zero). Thus rest pose is used to contain the mean of each facets deformation throughout the animation. The visualization shows which areas are close the global maximum value of the according deformation component. Red areas are those with deformation close to the maximum for that type and green are those close to minimum. There is no common maximum and minimum since each deformation component has its own domain of values and even normalization would make no difference.

In the following sections deformation gradient will always refer to $D_{\Sigma,i}$. This will be considered the deformation gradient of a facet for all the animation sequence. All deformation gradient calculations we use as input this quantity



a) b) c) d)

Figure 3. 12: a) Rotational deformation component, b) Scaling deformation component, c) Shearing deformation component, d) Deformation Gradient(sum of a),b),c)).

Deformation Based Clustering

It is now possible divide the surface into areas with similar deformation gradient. To carry out this division, a clustering method must be utilized. 3 properties must be satisfied:

1) The clusters must present with deformational coherency, to be possible to distribute proxy joints based on how much deformation an area presents.
2) The clusters must be connected (spatial coherency), to be possible to apply a weight fitting scheme on the cluster as a whole.
3) The method must be able to return at most the requested by the user number of clusters, to be possible to specify the number of distributed proxy bones.
4)

We have tested three possible clustering techniques:

a) K-Means
    a. Regular
    b. Hierarchical
b) Variational Region Growing

of which, Variational Region Growing yielded the best results. It should be noted that deformation gradient is a quantity closely related to the mesh's facets. For that, all clustering techniques are applied with reference to facets and not vertices and the distances used are the Euclidean among the centroids of the facets.

K-Means Clustering

K-Means is a widely used heuristic algorithm for partitioning a data set into k subsets. More formally, the algorithm attempts to partition a set of observations $(x_1, x_2, ..., x_n)$ into $k \leq n$ sets $S = \{S_1, S_2, ..., S_k\}$ where sum of squared distances of each point of the partition from the center of the partition is minimized:

$$\arg\min \sum_{i=1}^{k} \sum_{x_j \in S_i} \left\| x_j - \boldsymbol{\mu}_i \right\|^2 \quad \textbf{(3. 22)}$$

where $\boldsymbol{\mu}_i$ is the center produced by the mean of values (centroid) of S_i

$$\boldsymbol{\mu}_i = \frac{\sum_{x_j \in S_i} x_j}{|S_i|} \tag{3. 23}$$

The k-means algorithm is outlined in the following pseudo code:

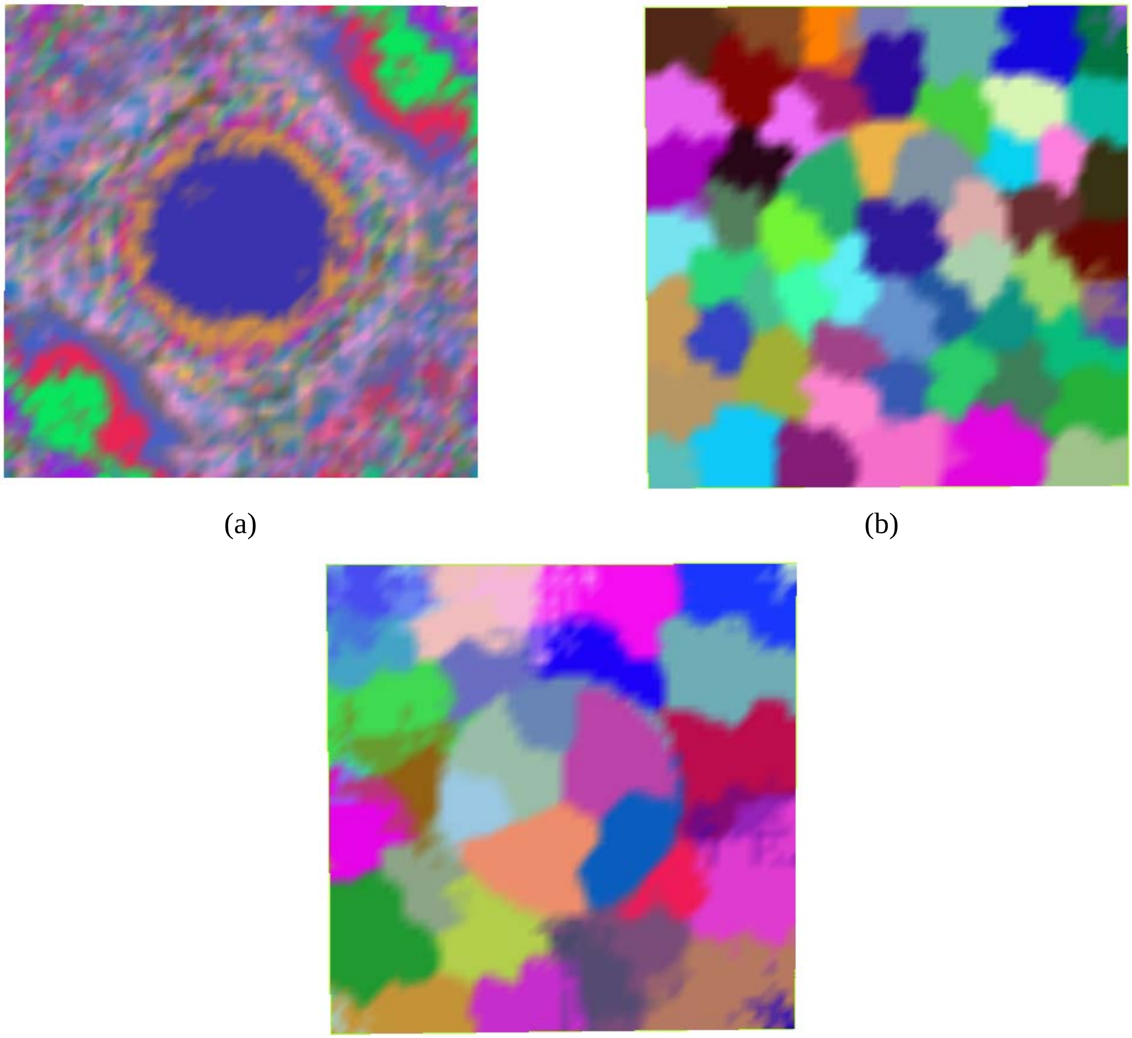
Algorithm 3.3 Perform_K_Means(MeanDeformationSequence[0])

```
1. Prev_K_CenterList:=∅;
2. K_CenterList := Pick_K_RandomFacets(MeanDeformationSequence[0]);
3. while Prev_K_CenterList != ∅ and objfun() > TOL do
4.         Prev_K_CenterList:= K_CenterList ;
5.         [KCenterMapping,K_CenterList]:=AttractNearestPoints(Prev_K_CenterList,
eanDeformationSequence[0]);
```

Algorithm 3. 3: K_Means Clustering

We start by picking k-random facets as initial cluster centers. Then each cluster attracts the facets closer to it. Upon insertion of new vertices, the centroids, i.e. the new cluster centers are being and the process repeats until some stopping criterion is reached. The algorithm returns the coordinates of the cluster centers and a mapping to these centers for each vertex.

Since k-means is a heuristic, there is no guarantee that it will converge and even if it does, the solution is not a global minimum. The result of the method will always depend on the initial choice of centers. Even so, in the context of our research, it can adequately recognize areas of similar deformation. However, if fed only with deformation data it presents with a serious disadvantage. The areas yielded are disjoint. Figure 3. 13 (a) depicts the result of applying K-Means clustering on purely deformation data. Apparently it is not possible to enforce some weighting behavior if the proxy joint of the same cluster are scattered all over the model.



(a) (b)

(c)

Figure 3. 13: a) K-Means clustering based purely on deformation data (17 clusters), b) Hierarchical K-Means clustering (59 clusters) with equally weight deformation and spatial data c) Hierarchical K-Means clustering with highly weight deformation data (37 cluster, 947 disjoint clusters). Colors are randomly chosen and depict no information other than spatial.

To insert topology in the clustering result and reduce disjoint sets, we implemented Hierarchical K-Means clustering, a two stage application of k-means. At the first level regular k-means is applied on a dataset that contains the deformation data of n facets and a first partitioning of $k', k' \leq n$ clusters is created. Then, at the second level, we apply k-means to the centers of the clusters created by the first level, based on their coordinates and a second partitioning of $k,\ k \le k' \le n$ clusters. Finally based on the Level2 cluster mapping we map the Level1 results to those of Level2. Algorithm 3. 4 describes this procedure.

Algorithm 3.4 Hierarchical_K_Means(DatasetWeight, MeanDeformationSequence[0])

1. Dataset = $\bigcup_{i=1}^{DatasetWeight}$ MeanDeformationSequence[0];
2. [L1_K_ClusterMapping, L1_K_ClusterCenters] := Perform_K_Means(Dataset);
3. [L2_K_ClusterMapping,L2_K_ClusterCenters]= Perform_K_Means(L1_K_ClusterCenters);
4. L2_K_Clusters:= MapL1ToL2(L1_K_ClusterMapping, L1_K_ClusterCenters,L2_K_ClusterMapping);

Algorithm 3. 4: Hierarchical K-Means Clustering

The mapping notion is depicted in Figure 3. 14

L1 Clustering	
f_1	c_1^1
f_2	c_4^1
f_3	$c_{k'}^1$
	⋮ ⋮ ⋮
f_n	c_7^1

L2 Clustering	
c_1^1	c_3^2
c_2^1	c_k^2
$c_{k'}^1$	c_5^2

Figure 3. 14: Hierarchical k-mean clustering mapping. Facets are assigned to the clusters of Level 2 via the mapping to the clusters of Level 1

In terms of functions if $C^1(x)$ maps facets to the clusters of the first level of clustering and $C^2(x)$ maps first level cluster centers to second level clusters then each facet's cluster is given by:

$$C(f_i) = C^2(C^1(f_i)) \quad \textbf{(3. 24)}$$

As shown in Algorithm 3. 4, to avoid clustering being dominated by spatial (center coordinates) information in the second level, the deformation domain can be enhanced by repetitions of the deformation data set. Figure 3. 13 (b) contains the result of hierarchical clustering with no deformation enhancement. It is apparent that clustering is dominated by spatial data. Results of enhancing the clustering domain with the deformation data appear in Figure 3. 13 (c).

Hierarchical k-means achieves better capturing of deformation data but does not avoid disjoint sets. Figure 3. 13 (c) only 37 clusters are returned, however if we apply an algorithm to identify disjoint clusters parts the number soars to 947. It appears that the number of returned clusters cannot be controlled if spatial coherency is enforced. Furthermore if each cluster is to be assigned at least one proxy bone, it is not practical to hold on to this partitioning scheme. In general k-means appears inadequate to satisfy the 3 properties mentioned in (0) and another partitioning method must be used to identify the required areas.

Variational Region Growing Clustering

A region growing algorithm appears to be a suitable solution for identifying areas of spatial coherency. Setting the upper limit of returned clusters is also a matter of specifying the required number of initial seeds and start the region growing algorithm from each.

A naive region growing scheme would be to grow seed facets until a certain criterion is met. Specifying this criterion is not a trivial task. It must ensure that facets with the right deformation are added to the appropriate region. If the threshold is not carefully chosen, a region may claim a facet that shouldn't, simply because it reached at it first. Also a special care must be taken for the possibility of orphaned facets. Apparently it is cumbersome to satisfy all these limitations and cannot be done without compromises that will render the result suboptimal.

Steiner et.al [13] presented a clustering technique which attempts to perform a partitioning of a mesh in a way that preserves anisotropy by globally minimizing the following distance for each region R_j :

$$L^{2,1}(T_i, P) = \left\| \mathbf{n}_i - \mathbf{N}_j \right\|^2 \left| T_i \right| \tag{3. 25}$$

where T_i is the triangle of facet i which belongs in R_j , $\left|T_i\right|$ is the triangle area, P is the centroid of the area , $\mathbf{n}_i$ is the normal vector of that facet, $\mathbf{N}_j$ is the area weighted average normal vector of region R_j :

$$\mathbf{N}_j = \sum_{T_i \in R_j} \left|T_i\right| \mathbf{n}_i \tag{3. 26}$$

This method attempts to identify areas based on the similarity of each facet normal vectors with a normal vector that is considered characteristic for each area. There is an analogy with the objective of our research since we have to identify areas based on the similarity of the deformation gradient of a facet and a deformation gradient quantity, characteristic of the area. To the existing nomenclature we add $R_j(f_i)$ which is the facet of the region with index i, $R_j(v_i)$ which is the vertex of the region with index i , $\left|R_j\right|_f$ and $\left|R_j\right|_v$ which are the number of facets and vertices in the region respectively.

We employ the same clustering algorithm as in [13], attempting to minimize the following distance for each region R_j :

$$D(T_i, F_j) = \left\| d_{\Sigma,i} - \mathbf{D}_j \right\|^2 \tag{3. 27}$$

where $\mathbf{D}$ is the mean deformation gradient of region R_j :

$$\mathbf{D}_j = \frac{\sum_{f_i \in R_j} D_{\Sigma,i}}{\left|R_j\right|} \tag{3. 28}$$

Note that all these operations take place on the rest pose. That is why the inter-pose mean deformation gradient $D_{\Sigma,i}$ of facet f_i is employed.

Steiner et.al name (3. 25) *distortion error*, so to preserve the analogy we name $D(T_i, F_j)$ *deformation error*. Notice that deformation error is not area weighted. There is no need for inserting the area in the metric since it is deformation we seek to capture upon the surface. Any area differentiation in the course of the animation sequence is captured by scaling and shearing deformation quantities, embedded in deformation gradient. The clustering algorithm is divided in three steps

1. **Initialization**: A number of k facets are picked at random, equal to the maximum number of clusters we require. These facets will serve as the first seeds for the growing algorithm. In a sense each seed is a region in this phase, and its deformation gradient is the $\mathbf{D}_j$ quantity of the deformation error. It must be noted here that this algorithm too is heuristic and greatly dependent on the initial seed choice. In an animation sequence the parts of a highly deformable model that are subject to most deformation are usually the boundaries. Recall that the P-Center algorithm has the tendency to assign centers at the boundaries in its first steps. Since k is small, this appears as a useful property and P-Center is used for initial seed selection instead of random selection.
2. **Deformation-Minimizing Flooding:** Each seed facet registers its immediately adjacent facets (one edge in common) to a global priority queue, sorted by their deformation error against $\mathbf{D}_j$ of the region they belong. Along with the facet, an index is stored indicating the region it has been tested against. So a facet may appear in the priority queue at most 3 times. Once all seeds have registered their adjacent facets, the first facet is exported from the priority queue and a check is made if it has already been assigned to some region. If not, it is assigned to the region that the corresponding index indicates and the facet is marked as claimed by that region. If the facet has been assigned to some region it is ignored and the next facet is exported from the priority queue. Prior to its assignment to the indicated region, each facet registers it's immediately (up to two) adjacent facets into the queue. The procedure continues in the same fashion until the priority queue is empty upon which point each facet has been assigned to a region. Note that seed facets don't enter the priority queue. This way we ensure that the requested number of regions is returned even if a region consists of one facet.
3. **Seed Fitting:** Upon emptying of the priority queue, the value $\mathbf{D}_j$ of each new region is calculated and a new seed is selected in each region. The seed is the facet whose deformation gradient is closer to $\mathbf{D}_j$. Notice that we do not replace $\mathbf{D}_j$ with $D_{\Sigma,seed}$. The seed facet is simply used to initiate the growing process.

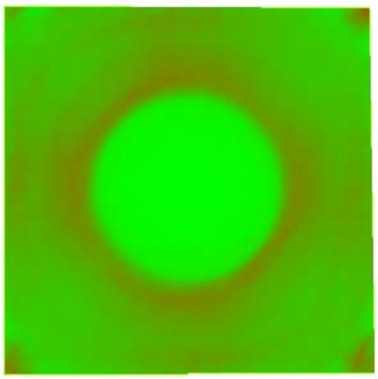
The process repeats from 2 to 3 until sum criterion is met or until a specified number of iterations is performed. The criterion we have used was the deformation error, at each step, of the newly appointed seed facet $D_{\Sigma,seed}$ as described in Seed Fitting part of the algorithm. Algorithm 3.5 gives the partitioning algorithm in pseudo code. We denote as $R_{j,seed}$ the seed facet of region R_j, and as $D_{j,Seed}$ the deformation gradient of the seed facet.

Algorithm 3.5 VariationalRegionGrowing(AnimationSequence[0], k ,maxIterations,TOL)

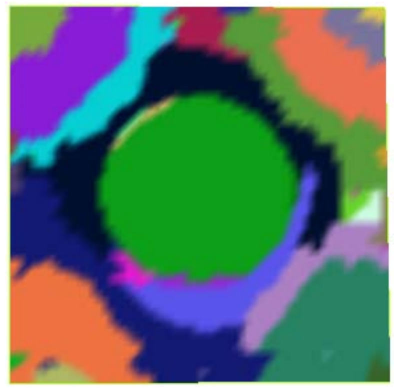
newC:=P-CenterClustering(AnimationSequence[0], k);
PriorityQueue:= $\varnothing$;
AssignedFacets:= $\varnothing$;
C :=newC;
while $\underset{|C|}{avg}\left(\sum_{R_j \in C} \left\| D_{j,Seed} - \mathbf{D}_j \right\|^2\right)$ <TOL **or** maxIterations>iterations **do**
    for each R_j **in** C **do**
        PriorityQueue := PriorityQueue $\cup$ ImmediatelyAdjacentFacets($R_{j,seed}$);
    while PriorityQueue != $\varnothing$ **do**
        Facet:= PopFromQueue();
        if !AssignedFacets.Contains(Facet.ID)
            PriorityQueue:= PriorityQueue $\cup$ ImmediatelyAdjacentFacets(Facet);
            AssignedFacets.Add(Facet.ID);
            newC[Facet.RegionID].Add(Facet);
    for each R_j in C **do**
        $\mathbf{D}_j$:=ComputeMeanDeformationgradient(R_j);
        $R_{j,seed}$:=GetNearestFacetToMean($\mathbf{D}_j$);
    C := newC;
    newC:= $\varnothing$;
    iterations:= iterations+1;

Algorithm 3.5: Variational region growing

Variational region growing ensures spatial and deformation coherency returns the required number of clusters. Initialization process is performed in P-Centers $O(kn)$ complexity. Deformation-minimizing flooding is performed rapidly ($n\log(n)$ complexity), while seed fitting is performed in $O(n)$ since all facet deformation gradients are to be compared with the mean deformation gradient of the region. Figure 3. 15 depicts the results when applying this algorithm.



a)



b)

Figure 3. 15: a) Mean deformation gradient distribution of animation sequence, b) Variational region growing

Deformation Based Proxy Joint Distribution

With the model partitioned according to the distribution of deformation gradient, it is now possible to distribute the proxy joints using this information. The idea is to distribute more proxy joints in region that enclose high deformation, while also considering the size of each area. It is pointless to assign a big number of proxy joints in a very small cluster because it presents with very high deformation and undermine larger clusters with slightly lower deformation. The area calculations are with respect to the rest pose of the animation. No assumptions or calculations are made about variations in the model's area through the animation sequence. The procedure is completed in two steps:

1) Specification of the number of proxy joints per region.
2) Distribution of proxy joints over the region.

Specification of the number of proxy joints per region

By default, all regions will be assigned with at least one proxy joint. We then need to distribute the rest of the proxy joints based on the participation of each region to the overall deformation and the overall area. One way to compute this participation would be to compute the deformation gradient of an area compared to that of the rest pose. However the deformation of a region has the area information embedded. As a result, large areas have high percentage of participation even if they present with small deformation and vice versa. It is essential that the deformation gradient amount of a region be separated by that of the area.

We achieve that by sorting each region according to the maximum value of deformation present in it. To avoid be misguided by outliers, we average a very small portion (e.g. 1%) of its maximum deformation gradients. The result is then linearly interpolated between the minimum and maximum deformation gradient values of the rest pose as a whole, to produce a value in [0,1]. We call the resulting quantity, *Deformation Indicator.*

$$dI_j = lerp(avg\_\max_j) \tag{3. 29}$$

where

$$lerp(x):[\min_j(\mathbf{D}_j), \max_j(\mathbf{D}_j)] \to [0,1] \tag{3. 30}$$

To maintain balance between region size and deformation we then compute the percentage of participation of each region to the area of the model. We denote by dI_j the deformation indicator of region R_j and a_j the area percentage of the region with respect to the area of the model. Their product gives the *Participation Factor* of the region:

$$FP_j = dI_j \cdot a_j \tag{3. 31}$$

To transform this factor into a percentage we sum up all participation factors and divide it by this sum. If n proxy joints are to be distributed then the share of each region is given by:

$$\left\lfloor n \cdot \frac{FP_j}{\sum_j FP_j} \right\rfloor \tag{3. 32}$$

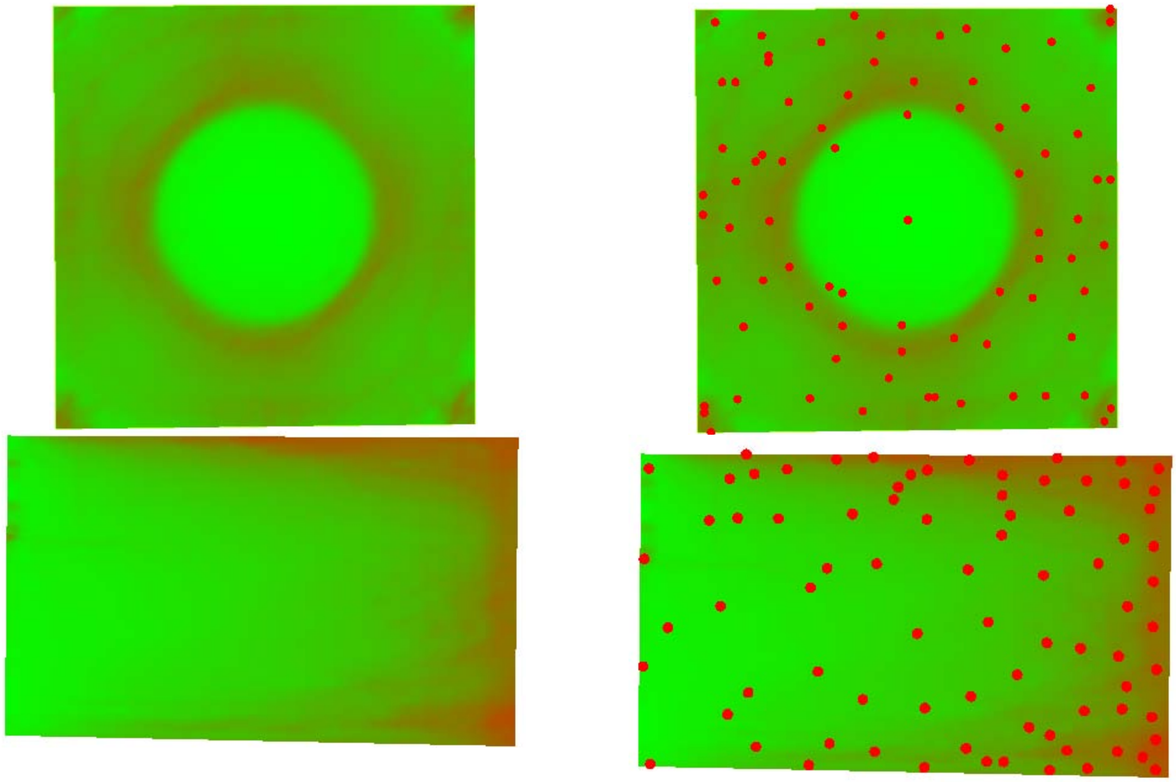
Assignment is performed with regions sorted by the deformation indicator. Thus regions with high deformation have higher priority. This formulation results in residuals due to truncation. The remaining proxy joints are sequentially assigned one per region, according to the priority bestowed by their deformation indicator. Only in this case the area percentage is not taken into account.

Distribution of proxy joints over the region

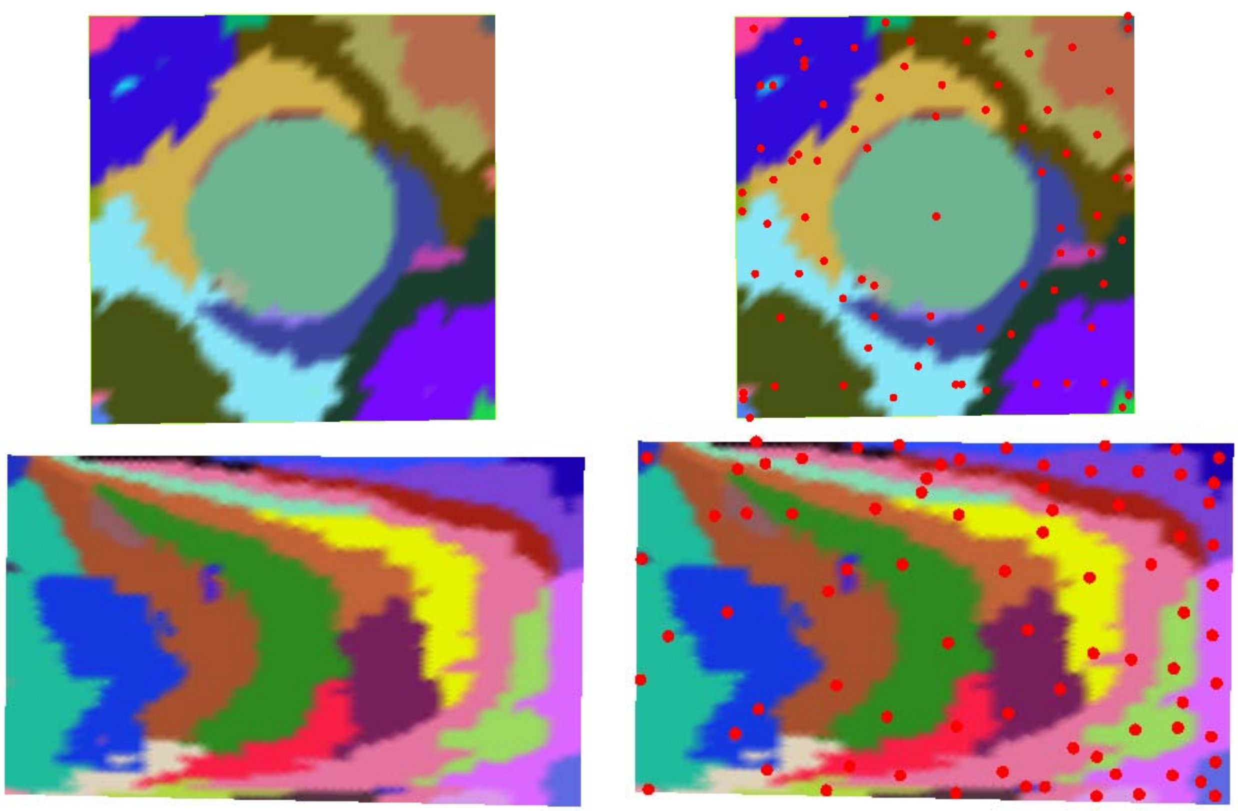
Knowing how many proxy joints each region has, what is left is to distribute the proxy joints upon this region. To facilitate our weight fitting policy we need the proxy joints uniformly distributed over each area. For this reason we employ the regular k-means clustering algorithm. Recall that proxy joints are vertices. So this time clustering is done using vertex coordinates. In case a region has only one proxy joint we assign it to the vertex closer to the centroid of the region. The centroid of the region is trivially computed by:

$$C_j = \frac{\sum_i \mathbf{v}_i}{\left|R_j\right|_v} \tag{3. 33}$$

Figure 3. 16 shows the results of our method on the tablecloth animation sequence and on a flapping flag. Notice how red areas are overpopulated with proxy joints.



(a) (b)



(c) (d)

Figure 3. 16: a) Deformation gradient blueprint, b) Proxy joints distribution against deformation gradient, c) Variational region growing partition, d) Proxy joints distribution against region partitioning

Deformation Based Weight Fitting

Deformation based partitioning and proxy joint distribution can now be used to facilitate the weight fitting process. It must be noted that, as with proxy joint fitting, weight fitting is vertex and not facet oriented. In the course of our research two weighting schemes were tested:

a. Distance based influence
b. Convolution propagated influence

on which we elaborate in the following sections. Weight initialization in both schemes is based on the distance of a vertex from the proxy joint. The further a vertex is from a proxy joint, the less influence it receives from it.

Distance based influence

This scheme employs purely distance based inter-cluster influence from proxy joints. Each proxy joint creates a spherical influence area around it affecting all vertices in it. Recall that k-means clustering has been used to distribute the proxy joints upon each region. The results of this clustering can also be used to define the influence regions of proxy joints. The radius of each influence area is set equal to the distance of the proxy joint from the furthest vertex of the cluster k-means returned and had as center that proxy joint. This scheme is similar to the one used of uniform proxy joint distribution (section 3.4.3). Figure 3. 17 depicts this notion.

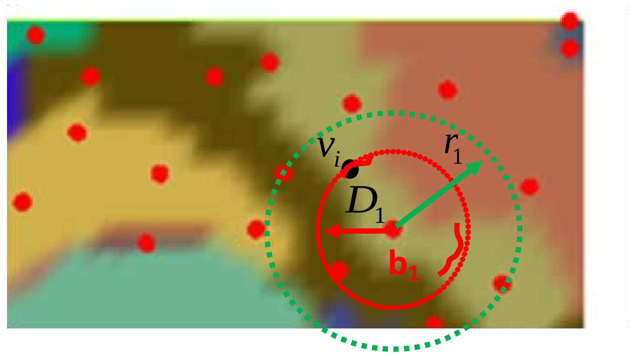
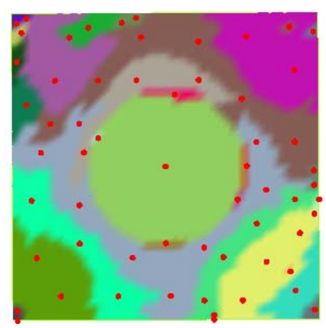


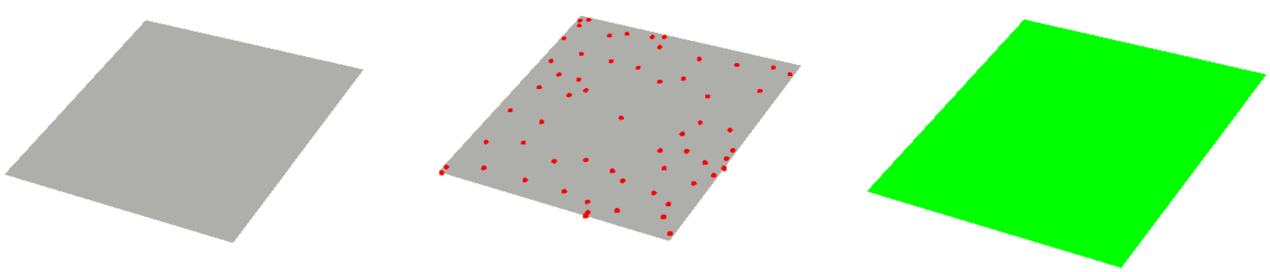
Figure 3. 17: Distance based inter-cluster influence

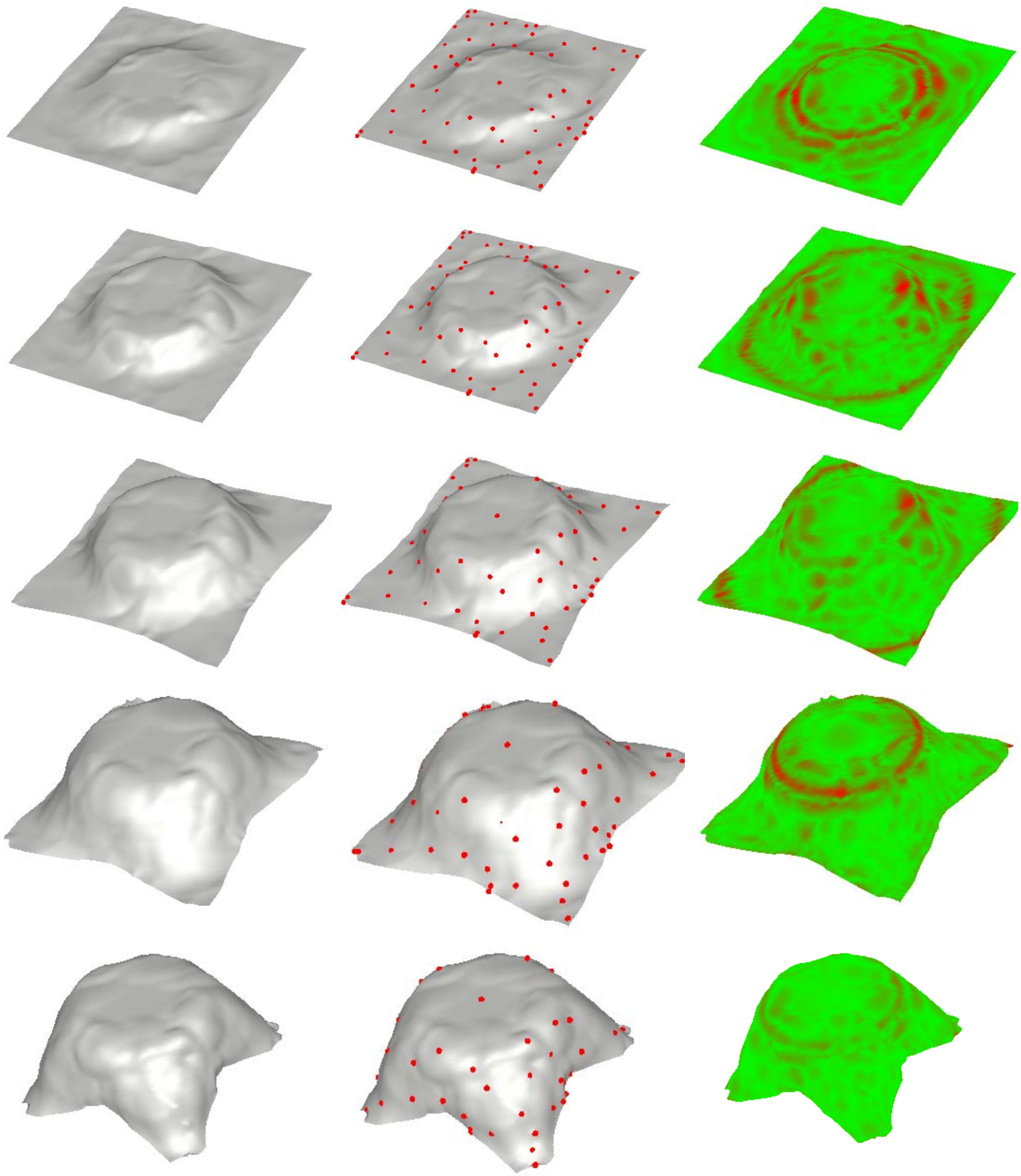
Proxy joint b_1 has v_i as its furthest vertex in its cluster. This produces a region of radius equal to their distance and outlined by the red dashed circle. The continuous red line marks the within-region borders between the clusters returned by applying k-means. There is also a P-Factor present to facilitate the enhancement of the influence area. Equations (3. 15) and (3. 16) give how weight influence is computed. Figure 3. 18 shows the results of the approximation process when using this scheme for weight fitting.

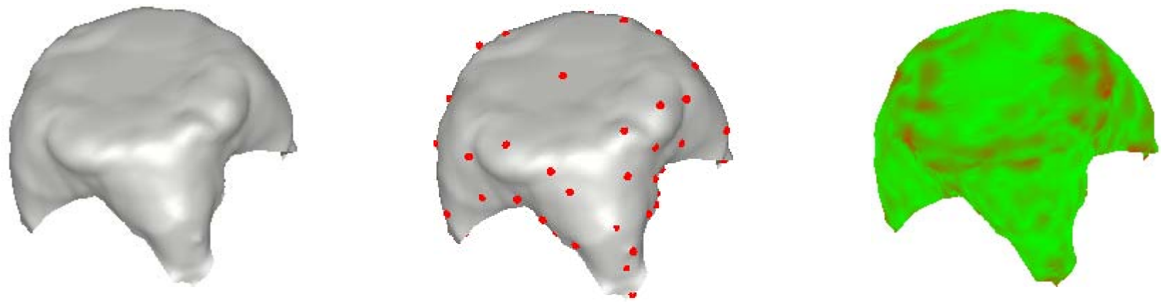
(a)



(b)







(c) (d) (e)

Figure 3. 18: Approximation with distance based weight influence. a) Deformation based region specification, b) Deformation based proxy joint distribution, c) Approximation d) Approximation with proxy joints, e) Approximation error distribution.

As in uniform proxy joint distribution, over-fitting effect is obvious. The result is the presence of bumps in the approximation. Especially in the event of small isolated regions with one proxy joint (Figure 3. 19).

Figure 3. 19: Bumps on isolated regions due to over-fitting

It appears that in some way the amount of inter-cluster influence must be regulated. We also need to take advantage of our deformation based region specification. For example in areas where there is little deformation (such as the middle of the tablecloth), a certain degree of rigidity may be acceptable and more inter-cluster influence should be added as we approach the borders of the regions.

Convolution Propagated Influence

In this scheme, during the initialization stage, we use the borders of the region to block inter-cluster penetration. All vertices within a region are influenced only be its proxy joints. Additionally each proxy joint affects all vertices within a region. Only the vertices on the borders, receive inter-cluster influence. Weight assignment is still distance based. Recall that only Euclidian distances are being used.

Kim et.al [14], proposed to perform Laplacian smoothing to smooth the weight influences from various bones of quasi articulated animation sequences. We adapt this idea to highly deformable animations sequences and change Laplacian smoothing, which is liable to produce negative weights, to mean smoothing. The process is analogous to the convolution process, applied for smoothing 2D images. Instead of pixels we have vertices and instead of color components (i.e. RGBA) we have influences from bones.

We denote by:

$$Star(v_i) = \{Sv_1, Sv_2, ..., Sv_n\} \tag{3. 34}$$

the set of vertices neighboring to v_i.

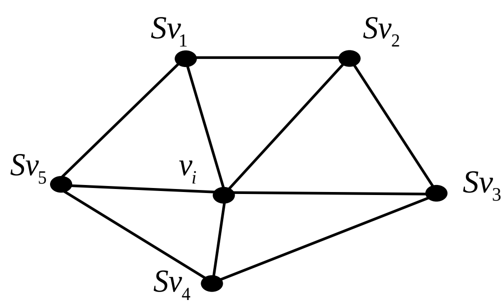
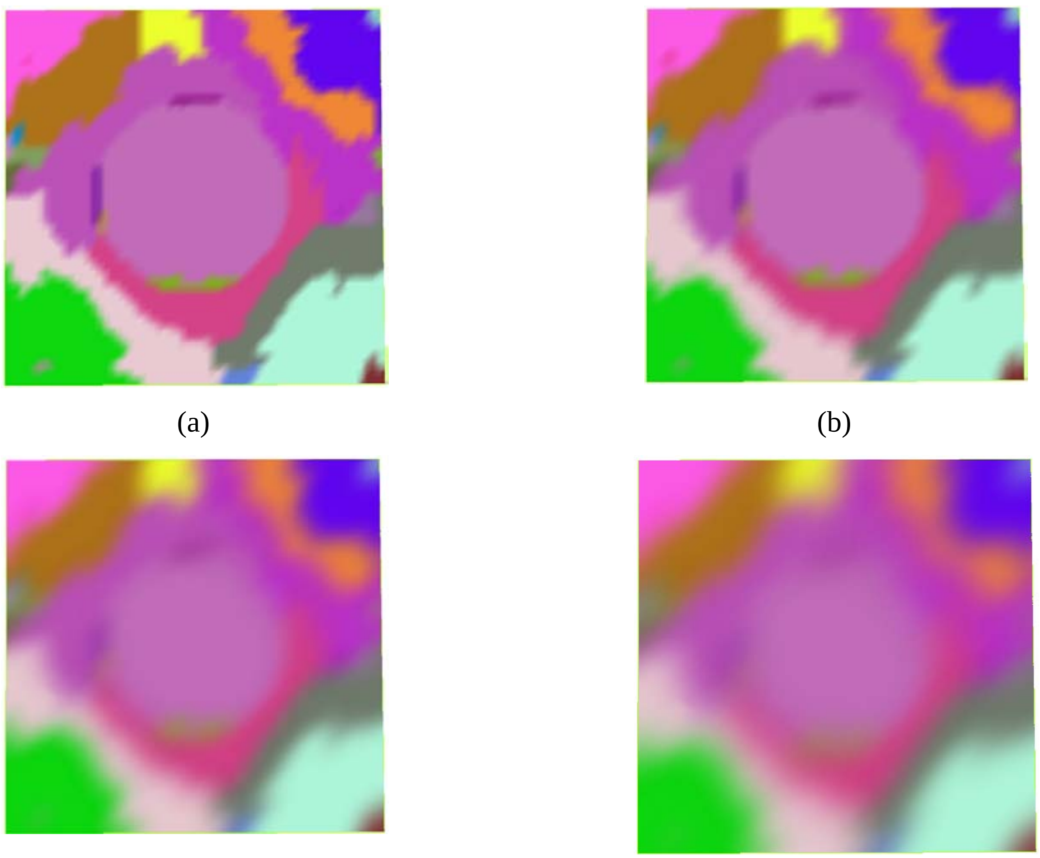


Figure 3. 20: Star of a vertex

The convolution kernel is not based on the pixels neighborhood but of the vertex's star. Thus each weight component (influence) is given by:

$$w_{bi} = \frac{1}{|Star(v_i)|}\left(w_{bi} + \sum_{j=1}^{n} w_b^{Sv_j} \right) \tag{3. 35}$$

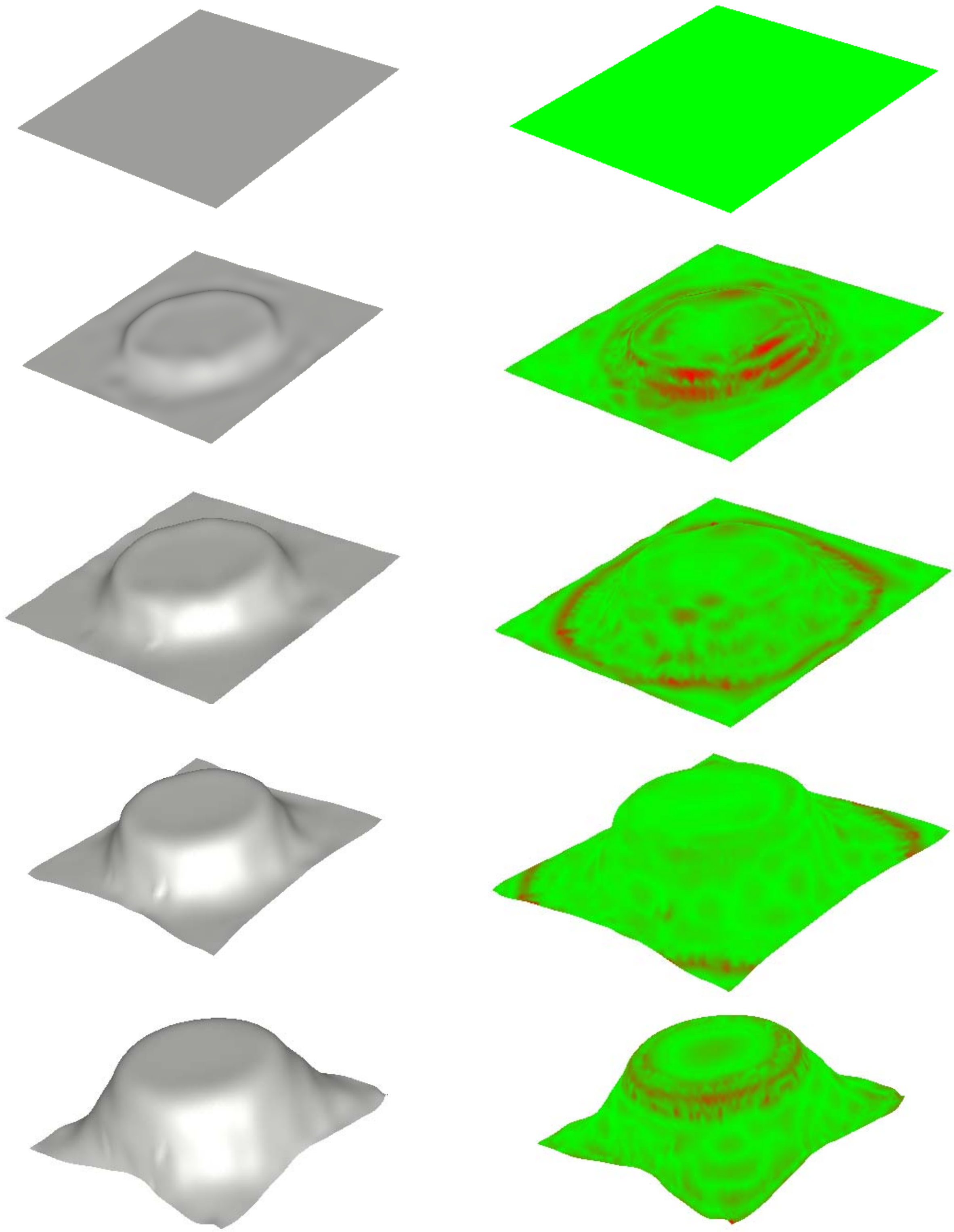
Upon completion of the initialization process, the convolution phase initiates, during which each vertex convolves its weight influences with the influences of its neighboring vertices. Figure 3.21 visualizes the effects of various levels of smoothing while Figure 3.22 depicts the approximation results when Convolution Propagated Weight fitting is used.



(a) (b)

(c) (d)

Figure 3. 21: a) Initial region specification, b) 1-pass Smoothing, c) 5-pass Smoothing, d) 10-pass Smoothing



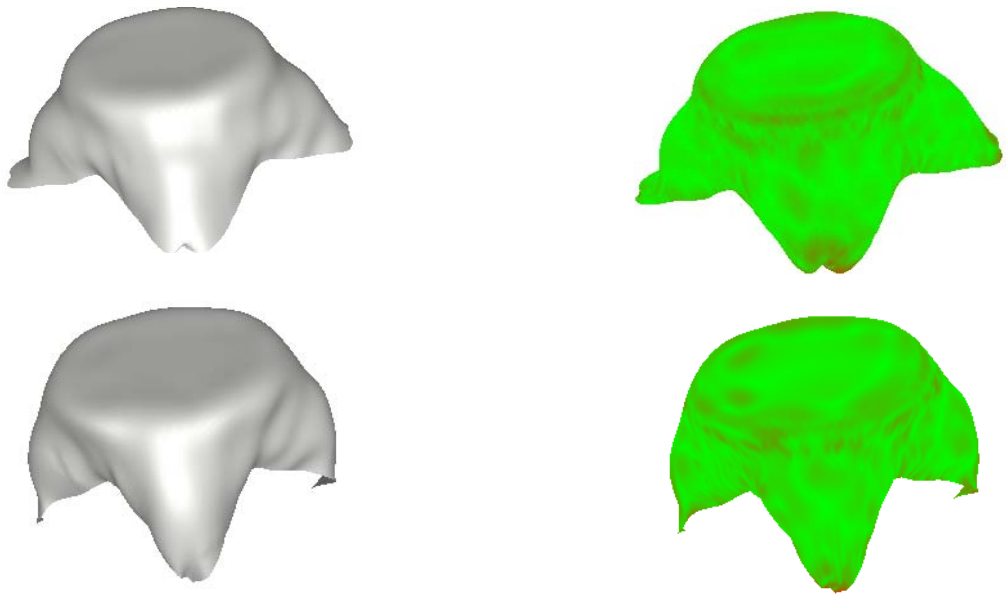


Figure 3. 22: Approximation with Convolution propagated weight fitting.

We see that the bumps on the surface have disappeared. This is a result of uniformly distributed weight influences over the surface due to weight smoothing. However the error is still close to that of the Uniform proxy joint distribution. This is because our method also suffers from over-fitting. We do not have a limitation over the number of influences a vertex may have. So even with regulated influences the result is an overall rounding of the model. Execution time has also increased due to over-fitting as expected.

CHAPTER 4

IMPROVING EFFICIENCY BY DECIMATION

4.1 **Introduction**

Transformation matrix fitting is the most time consuming phase of the approximation process. Using Dual quaternions instead of affine matrices can boost execution but with a tradeoff in quality. Another way would be to reduce the number of proxy joints but rationally this would also lead to increase of error. Another way to speedup execution of transformation fitting is to reduce the number of vertices of the model by applying some *decimation* (*simplification*) method on it. Depending on the level of simplification we expect some increase in the approximation error due to reduction of available samples for the approximation of transformations and to possible misplacements of the contracting vertices. However we show that the reduction in execution time is significant enough to render the approximation error acceptable.

The idea is to simplify the animation sequence to any level of detail we desire and then use the remaining vertices to specify the transformation matrices of the proxy joints.

In the following sections we present methods of achieving simplifications of animated sequences in a manner that produces quality approximations. We present an error metric that is used to decide the order in which the edge contractions must occur and then present various methods of deciding the vertex position of the contracting edge.

4.2 Simplification Process Overview

Equation (3.34) and Figure 3.20 present the definition of the star of a vertex. Simplification is the process of iteratively merging the stars of adjacent vertices. This merging is called contraction. Figure 4. 1 gives an example of a single iteration of simplification process.

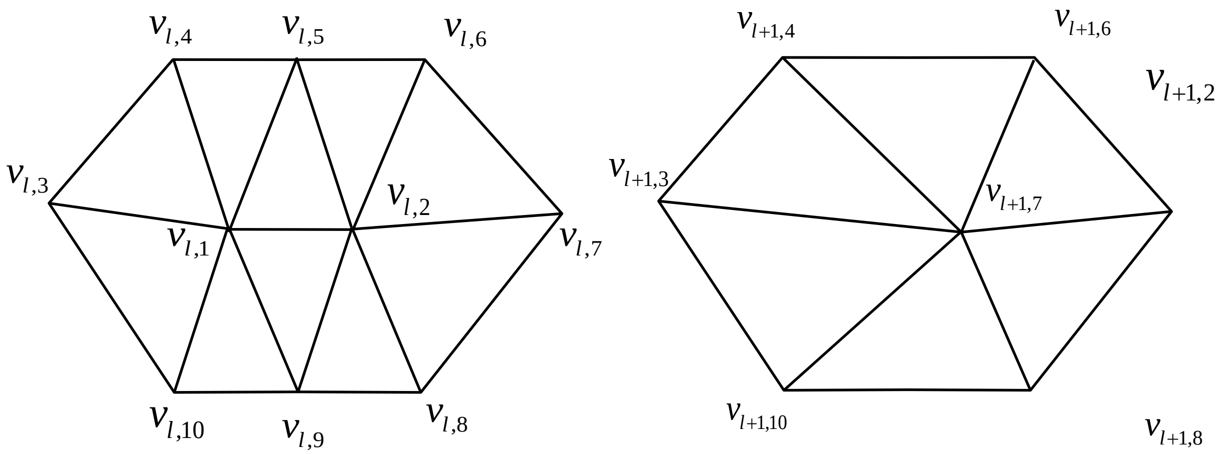


Figure 4. 1: Vertex contraction in simplification process

Index l indicates the level of detail the mesh currently is.

Generally for two vertices v_1, v_2 to contract, any of the following must hold:

1. (v_1, v_2) is an edge
2. $\|v_1 - v_2\| < t$, where t is some threshold.

In our context however it is not desirable for unconnected vertices to contract due to proximity, because mesh connectivity preservation is a major prerequisite. Additionally we make no distinction between edges (v_i, v_j) and (v_j, v_i) since the methods we use do not require such. Also, as seen in Figure 4. 1, when two vertices contract the resulting one takes the index of the second vertex of the edge. Finally the triangles that contain both contracting vertices disappear after contraction and with them any unused vertex indices.

It is also desirable to delay simplification of areas that undergo high deformation because we need as much vertices as possible to better approximate them. This means that a priority must be kept in3 the order of vertex contractions. Finally we must make sure that simplification does not leave a proxy joint without vertices dependent on it because it will not be possible to compute its transformation matrix and thus approximate the area it covers in the original model.

4.3 Contraction Priority Specification

To enforce a priority in the order of contractions it is necessary to introduce a metric that describes their cost. All contractions can then be sorted and executed according to this cost. Upon each contraction an algorithm is employed to decide the optimal position of the vertex produced by merging the contracting ones. No matter how accurate this positioning is, an error is always propagated with respect to how the shape of the model is preserved after each contraction. The amount of this error can be the cost criterion of the contraction. For now we assume that a method which decides on the merge vertex location after the contraction already exists. We shall present such methods later.

To quantify this error [15] introduce a method of approximating it using quadrics (2. 56) (2. 57). Each vertex in a triangulated mesh is the solution of the system of equations that describe the plane of the facets that contain it. Thus each facet has an error quadric (2. 57) associated with it. If we moved this vertex to some other location $\mathbf{v}'$ then the sum of the squared distances from all its former planes would be:

$$\begin{aligned} \Delta(\mathbf{v}') &= \sum_{\mathbf{p}\in planes(\mathbf{v})} (\mathbf{v}'^{\mathbf{T}}\mathbf{p})(\mathbf{p}^{\mathbf{T}}\mathbf{v}') \\ &= \sum_{\mathbf{p}\in planes(\mathbf{v})} \mathbf{v}'^{\mathbf{T}}(\mathbf{p}\mathbf{p}^{\mathbf{T}})\mathbf{v}' \\ &= \mathbf{v}'^{\mathbf{T}}\left(\sum_{\mathbf{p}\in planes(\mathbf{v})} \mathbf{p}\mathbf{p}^{\mathbf{T}}\right)\mathbf{v}' \\ &= \mathbf{v}'^{\mathbf{T}}\left(\sum_{\mathbf{p}\in planes(\mathbf{v})} \mathbf{K}_{\mathbf{p}}\right)\mathbf{v}' \end{aligned} \qquad \textbf{(4. 1)}$$

This gives the distance of any vertex from a set of facets that belong to the star of a vertex $\mathbf{v}$. Potentially what happens when a contraction takes place is that two vertices are moved from their original position and, merged, are positioned somewhere else, in a distance from the set of facets (planes) they belonged. This means that both vertices propagate an error dictated by their fundamental error quadrics. For two contracting vertices v_1 and v_2, we denote by Q_1 and Q_2 the error quadrics of each one respectively and the accumulated error quadric of their contraction (movement) is:

$$Q_{(1,2)} = Q_1 + Q_2 \tag{4. 2}$$

Depending on the coordinates of the merge vertex after the contraction, $Q_{(1,2)}$ gives the sum of squared distances of these coordinates from all of the facets from the stars of the contracting vertices.

The formulation above applies to static meshes, but it can be extended to animated sequences [17]. Assume that we have defined the new merge vertex coordinates for all contractions throughout the animation sequence. We denote by $\tilde{v}^p_{(i,j)}$ the vertex coordinates of the new vertex after contracting edges i, j in pose p. For each contraction pair (i, j) on an animated sequence of P poses the contraction error of an edge is:

$$\sum_{p=1}^{P} \tilde{v}^{p\;T}_{(i,j)} \left(Q^p_i + Q^p_j\right) \tilde{v}^p_{(i,j)} \tag{4. 3}$$

This formulation presents with an interesting property. The higher the deformation a facet undergoes the higher the contraction cost it produces for its vertices. This information can be used to form a priority queue for contractions and perform the contractions with high quadric error as late in the process as possible.

4.3.1 Proxy Joint Validity Preservation

Recall that proxy joint locations in the original model are associated with vertex locations. During the decimation process proxy joints maintain their original coordinates. That means that their base vertices may freely contract. We ensure that these contractions will not leave a proxy joint without vertices dependent on it by allowing contractions only between vertices that share exactly the same proxy joint influences, in the case of uniform proxy joint distribution (P-Center). In the case of deformation based region specification the rule is augmented by prohibiting contractions which contain vertices that are positioned on the border of a region. The reason is that we want to preserve the locality offered by this method even in the simplification phase.

4.3.2 Contraction Validity

One caveat of the simplification process is the possibility of change in the orientation of a facet. Figure 4. 2 demonstrates this behavior. If we assume that all facets are co-planar then contraction of vertices v_2 and v_3 from the mesh to the left results in what was facet $[v_3 v_1 v_6]$ to change its orientation (flip) and also overlap with $[v_6 v_5 v_2]$. This leads to not valid tessellation and potentially degenerate mesh. To avoid this effect, upon contraction cost computation we check the normal vectors of all facets in the star of contracting vertices. If the contraction results in a facet rotating its normal by 180^o then this contraction is canceled and ignored.

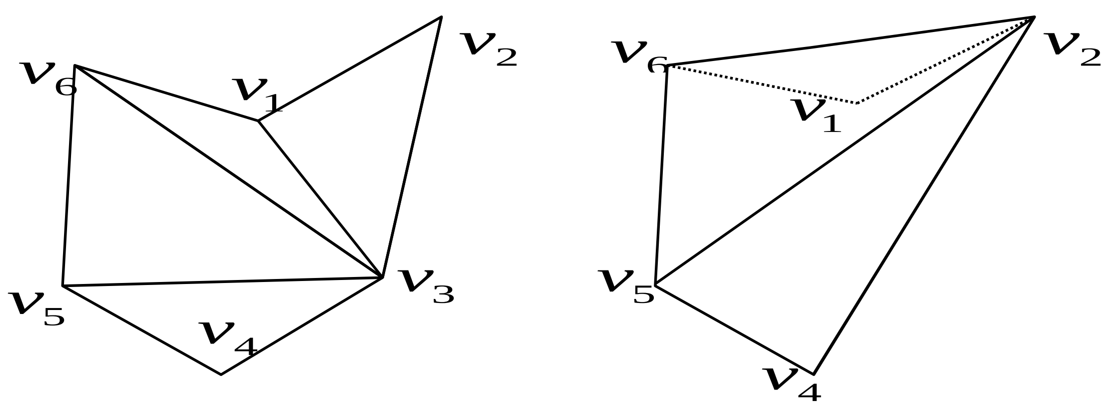
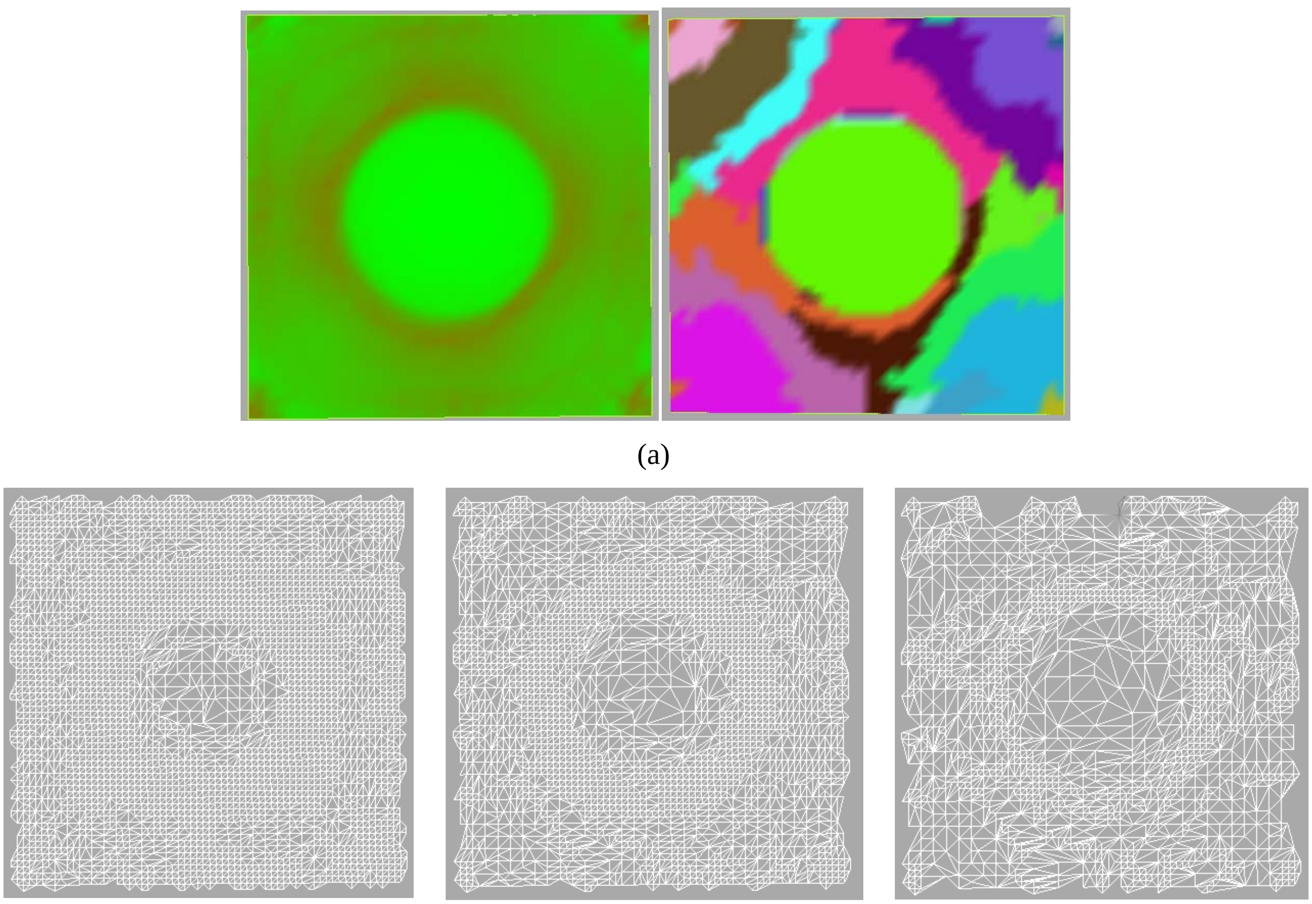


Figure 4. 2: Facet flipping after contraction

4.3.3 Priority Queue Schemes

The two different proxy joint distributions we presented employ different strategies in enforcing priority in the order of contractions.

In the case of proxy joint uniform distribution the priority queue is global. Contractions are viable to occur on any location upon the model, subject to the rules we have mentioned. Deformation based proxy joint fitting can also implement decimation using global priority queue. Figure 4. 3 shows the results of applying global priority queue decimation. Notice how dense the areas with high deformation remain and how the boundaries of the clusters are beginning to show after a certain level of decimation.



(a)

(b) (c) (d)

Figure 4. 3: Global priority queue decimation. a) Deformation footprint and clustering, b) 20% Decimation, c)40% decimation, d) 60% decimation. The boundaries of the clusters clearly visible.

In the case of Deformation based proxy joint fitting however, it is possible to deploy a more convoluted strategy. The reason is that our method has the notion of deformation embedded. We do make use of the quadric error but merely to order the contractions within each cluster. Globally, we want to use the information of deformation we have associated with the various regions, and allow regions with less deformation to be decimated to a higher degree while preserving regions with more deformation as much as possible. Another advantage that deformation clustering offers is the possibility to have multiple priority queues, each associated with a region. With the isolation provided by the proxy joint validity scheme we have the ability to perform the decimation process in parallel.

Deformation gradient data is utilized to decide how many contractions each region will perform on its vertices, compared the overall amount requested. This is done in a manner similar to the proxy joint distribution scheme. The size of the regions which in this context is interpreted as the size of the priority queue of each region (i.e. the vertex pairs viable for contractions) is also taken into account. The less deformation gradient enclosed within it and the largest its priority queue, the more contractions a region performs. In general we want all regions to participate in the simplification process and not have any exhaustively decimated.

To represent the deformation amount of a region, a quantity similar to deformation indicator (see (3. 29)) dI_j is utilized. However its use is slightly different. In the case of proxy joint distribution the higher the indicator was, the bigger the amount of data (proxy joints) to be distributed was. In the context of decimation the opposite holds. Big deformation indication means small amount of distributed data (contractions). Again we average only of a small portion of the maximum deformation values of each region (say 1%). Only this time the deformation returned is inverted and interpolated between the inverse maximum and minimum deformation values of the rest pose:

$$idI_j = inv\_lerp\left(\frac{1}{avg\_\max_j}\right) \tag{4. 4}$$

where

$$inv\_lerp(x) : \left[\frac{1}{\max_j(\mathbf{D}_j)}, \frac{1}{\min_j(\mathbf{D}_j)}\right] \to [0,1] \tag{4. 5}$$

Each time the simplification process is performed, each region is obliged to perform a fixed percent of contractions from its priority queue. We call this percentage *shuffling ratio*. idI_j (*inverse deformation indicator*) is used to regulate shuffling ratio. If we denote by PQ_j the set of contractions currently available in the priority queue of region R_j, the amount of contractions each region contributes is given by:

$$idI_j \cdot shuffling\_ratio \cdot |PQ| \quad \textbf{(4. 6)}$$

If a region manages to cover the amount of requested contractions, the process stops. This means that the larger the shuffling ratio, the less regions contribute in the decimation process. In general a large shuffling ratio is not preferable because it causes regions with small deformation to be exhaustively decimated while others remain untouched.

Algorithm 4. 1 provides a description of the decimation algorithm when using a global priority queue. For the priority queue to be populated, virtually all contractions in all poses must be calculated (function ComputeContractionCosts). This is done once during initialization and after each contraction only for those edges that where affected (function ContractGlobal, line5).

Algorithm 4.1 DecimationGlobalPQ(AnimationSequence,amount)

---

1. DummySequence:= AnimationSequence.Clone();
2. EdgeList := GetConnectivity(AnimationSequence);
3. GPQ := ComputeContractionCosts(EdgeList);
4. ContractionsList:= $\varnothing$;
5. **while** ContractionsList.Count != amount && GPQ != $\varnothing$ **do**
6.   Contraction := Pop(GPQ);
7.   **if** IsValid(Contraction) **then**
8.   ContractGlobal (Contraction);
9.   ContractionsList.Add(Contraction);

1. **function** ComputeContractionCosts (EdgeList)
2. **foreach** Edge **in** EdgeList **do**

SumCost :=0;
foreach Pose **in** AnimationSequence
MergeV:=NewVertexCoordinates(Edge,Pose);
SumCost := SumCost + $MergeV^T \cdot Edge.Q_1 \cdot Edge.Q_2 \cdot MergeV$
PQ.Push(SumCost,Edge);

function ContractGlobal(Contraction);
DummySequence.Delete(Contraction.V1);
GPQ.DeleteRelated(Contraction.V1);
GPQ.UpdateRelated(Contraction.V2);

Algorithm 4. 1: Description of the decimation process using a global priority queue

Algorithm 4. 2 describes the decimation process using one priority queue per deformation region.

Algorithm 4.2 DecimationLocalPQ(AnimationSequence,amount)

DummySequence := AnimationSequence.Clone();
ContractionsPerRegion := $\varnothing$;
ContractionsList := $\varnothing$;
foreach Region **in** AnimationSequence.DeformationRegions **do**
EdgeList := GetConnectivity(Region);
LPQ[Region] := ComputeContractionCosts(EdgeList);
ContractionsPerRegion := $idI_j \cdot shuffling\_ratio \cdot |LPQ[Region]|$
while ContractionsList.Count != amount && LPQ != $\varnothing$ **do**
foreach Region **in** AnimationSequence.DeformationRegions **do**
ContractionsDone:=0;
while ContractionsPerRegion[Region] > ContractionsDone
Contraction := Pop(LPQ[Region]);
if IsValid(Contraction) **then**
ContractLocal(Contraction, LPQ[Region]);

ContractionsDone:= ContractionsDone +1;

1. **function** ContractLocal(Contraction,PQ)
2. DummySequence.Delete(Contraction.V1);
3. PQ.DeleteRelated(Contraction.V1);
4. PQ.UpdateRelated(Contraction.V2);

Algorithm 4. 2: Description of the decimation process using Local priority queues

4.4 Merge Vertex coordinates Specification Methods

To perform a contraction, the coordinates of the merge vertex must be specified. Various schemes can be implemented for this decision. However it must be taken into account that this operation takes place at the heart of the simplification process and is repeated frequently. This means that the more complex the scheme the more time consuming the simplification process will be.

Second Edge Selection

Simple schemes can be used to for this selection such as selecting the location of the second vertex of the contracting edge. Figure 4.1 gives an example of such a contraction. This is the simplest scheme, the fastest and presents with good approximation results. The reason is that it retains the existing coordinates of the meshes vertices and thus the transformations of the real mesh. Some more sophisticated positioning might have been optimal with respect to the visual quality, but it would introduce deviations from the original geometry and thus error in the approximation. However a strong representation from the whole surface is still necessary and this method does not guarantee this prerequisite.

Optimal-Error Minimizing Selection

From equations (4. 1) and (4. 2) comes that merging the two contracting vertices to a new location $\bar{\mathbf{v}}$ results in an error of:

$$\Delta(\overline{\mathbf{v}}) = \overline{\mathbf{v}}^T Q \overline{\mathbf{v}} \tag{4. 7}$$

If we denote the unknown quadric matrix by :

$$Q = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4. 8}$$

then it holds that :

$$\begin{aligned} \Delta(\overline{\mathbf{v}}) \quad &= \overline{\mathbf{v}}^T Q \overline{\mathbf{v}} \\ &= q_{11}x^2 + 2q_{12}xy + 2q_{13}xz + 2q_{14}x + q_{22}y^2 + 2q_{23}yz \\ &+ 2q_{24}y + q_{33}z^2 + 2q_{34}z + q_{44} \end{aligned} \tag{4. 9}$$

which is a quadratic equation and to find the vertex $\overline{\mathbf{v}} = [x \quad y \quad z]^T$ that minimizes it is simply am matter of finding the roots of partial derivatives:

$$\frac{\partial \Delta}{\partial x} = \frac{\partial \Delta}{\partial y} = \frac{\partial \Delta}{\partial z} = 0 \tag{4. 10}$$

This is equivalent to solving :

$$\begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix} \overline{\mathbf{v}} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4. 11}$$

which is a matter of inverting Q :

$$\overline{\mathbf{v}} = \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} \\ q_{21} & q_{22} & q_{23} & q_{24} \\ q_{31} & q_{32} & q_{33} & q_{34} \\ 0 & 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{4. 12}$$

It is possible that matrix Q is not invertible. For each vertex, before the contraction, the 3x3 upper left part of its error quadric matrix is symmetric and positive semi definite:

$$A = \begin{bmatrix} a^2 & ab & ac \\ ab & b^2 & bc \\ ac & bc & c^2 \end{bmatrix} \tag{4. 13}$$

The eigen-values and eigen-vectors of this matrix define the principal axes of an ellipsoid whose centroid is the vertex. That ellipsoid is an iso-surface that contains all the possible locations around the vertex that have error $\Delta(\overline{\mathbf{v}}) = \varepsilon$. During contraction, the optimal location of the merge vertex is searched in the union of the two iso-surfaces of the contracting vertices.
In the event that the stars of these vertices are completely co-planar, then the iso-surfaces are planes, and the possible optimal locations are infinite. The same holds when the contracting vertices lie on a sharp edge, where the iso-surfaces are cylinders extending to infinite.
In these two cases, matrix Q is singular, an optimal location cannot be specified using (4. 12) and as such we rollback to selecting the second vertex of the contracting pair.

4.5 Weight Influences Propagation

The main purpose of decimating the object is to perform transformation fitting using smaller versions of the original model. As the decimation process removes and relocates vertices, vertex weights need to be updated. Depending on the fitting scheme different strategies may be followed.

In the case of Uniform Fitting, because weight fitting is purely distance based, the weights of the remaining vertices can be recalculated upon completion of the decimation process.
In deformation based fitting due to convolution propagated influences the weights need to be assigned during the contraction process. This is done in the same way that weights propagate during the weight fitting procedure.

CHAPTER 5

REFINEMENTS

5.1 **Introduction**

Matrix palette skinning is an approximation technique and has not enough information to approximate sufficiently all the fine details of an animation sequence. It is simply not possible to contain all the degrees of freedom required around a bone within a transformation matrix. There exist techniques that can be used to improve the visual fidelity of the model, however each one presents with a cost in complexity. In the following section we shall describe one such popular technique along with two optimization techniques that we introduce and add to the overall result of our methods.

5.2 **Eigen-Skin**

In [9] a variation of the Eigen Skin corrections presented in [18], is suggested. The basic idea of Eigen Skin is to compute the error produced by the approximation process at each vertex and after adding this error to the rest pose, repeat the Linear Blend Skinning Process. This results in correction of the approximation errors. To specify the correction vector the difference between the approximation and the actual pose is used:

$$e_i^p = v_i' - T_b^p \mathbf{v}_i^p \tag{5. 1}$$

Correction vector is then transformed back to the rest pose using the inverse of the transformation matrix:

$$\mathbf{e}_i^p = \left(T_b^p\right)^{-1} e_i^p = \left(T_b^p\right)^{-1} v_i' - \mathbf{v}_i^p \tag{5. 2}$$

Figure 5. 1 describes this notion.

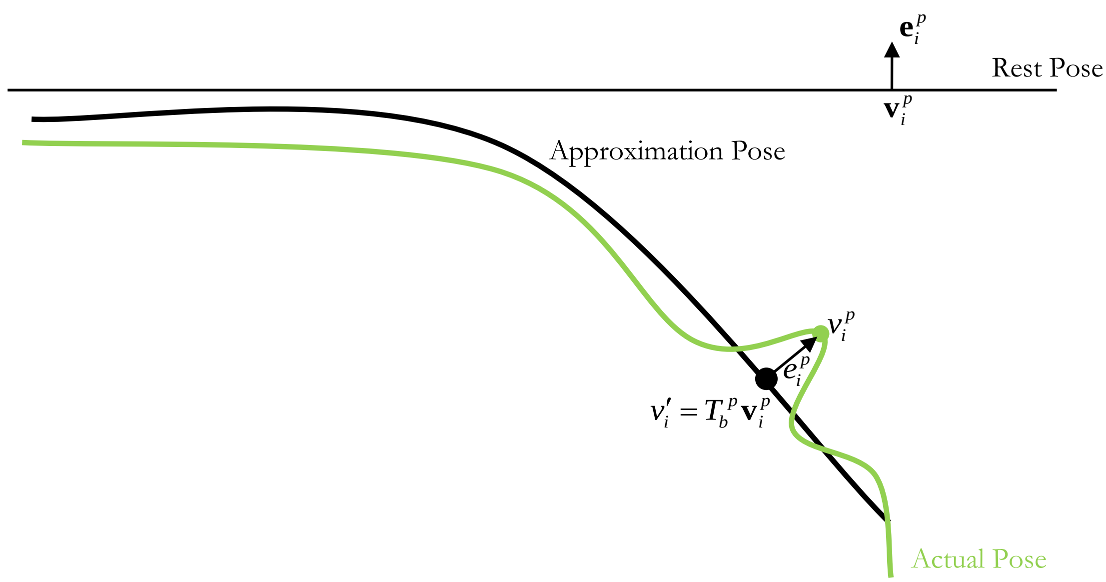


Figure 5. 1: Eigen Skin correction vector e_i^p

All correction vectors for each pose are computed and stacked in a 3NxP matrix E. Thin SVD is then performed upon E to decompose it in:

$$E = DK \tag{5. 3}$$

where D is a 3NxP matrix containing the so called eigen-displacement vectors while $K = SV^T$ is a PxP containing the eigen-displacement coefficients. To store these matrices so to have all the range of the corrections available would cancel the compression effect of matrix palette skinning. After all, not all of matrix D is required but a small number of its first columns which institute the eigenvectors that best describe the correction that must be applied per vertex so to be corrected throughout the animation. Thus instead of storing P columns only some $f \ll P$ requires to be stored. The rest of the unused columns are set to zero.

The resulting matrices are:

$$E' = D'K' \tag{5. 4}$$

where

$$D' = \begin{array}{c} \overbrace{\hphantom{d_{11} \quad \cdots \quad d_{1f}}}^{f} \; \overbrace{\hphantom{0 \;\; \cdots \;\; 0}}^{P-f} \\ \begin{bmatrix} d_{11} & \cdots & d_{1f} & 0 & \cdots & 0 \\ d_{21} & & d_{2f} & 0 & & 0 \\ \vdots & & & \vdots & & \vdots \\ d_{(3N)1} & \cdots & d_{(3N)f} & 0 & \cdots & 0 \end{bmatrix} \end{array} \tag{5. 5}$$

and

$$K' = \begin{array}{c} \overbrace{\hphantom{k_{11} \quad \cdots \quad k_{f1}}}^{f} \; \overbrace{\hphantom{0 \;\; \cdots \;\; 0}}^{P-f} \\ \begin{bmatrix} k_{11} & \cdots & k_{f1} & 0 & \cdots & 0 \\ k_{12} & & k_{f2} & 0 & & 0 \\ \vdots & & \vdots & & & \vdots \\ k_{1P} & \cdots & k_{fP} & 0 & \cdots & 0 \end{bmatrix}^{T} \end{array} \tag{5. 6}$$

Correcting the approximation is then only a matter of adding the corresponding eigen-correction to the approximation:

$$v_i^{p,correct} = T_b^p(\mathbf{v}_i^t + (\mathbf{e}_i^p)) \tag{5. 7}$$

Note that Eigen-Corrections need to be stored along with the rest of the matrices of the approximation. This reduces the compression achieved.

5.3 Rest Pose and Weight Corrections

Following the same philosophy we present two methods of applying corrections to the rest pose and the weights of the fitting process. These corrections are embedded in the final result and need not be stored separately as is the case with Eigen-Skin.

Rest Pose Corrections

Having computed transformation matrices T for all poses and proxy joints, we seek a vector $\mathbf{e}_i$ that is added to each vertex of the rest pose and minimizes the sum of the squared errors of the approximation in each pose:

$$\min\left(\sum_p \left\| \left(\sum_{b=1}^{B} w_{bi} T_b^p\right)(\mathbf{v}_i + \mathbf{e}_i) - v_i^p \right\|^2\right) \tag{5. 8}$$

The solution of this problem is equal to finding the least squares solution to:

$$\begin{aligned} &\sum_{b=1}^{B} w_{bi} T_b^p \mathbf{e}_i = v_i^p - \sum_{b=1}^{B} w_{bi} T_b^p \mathbf{v}_i \Leftrightarrow \\ &\sum_{b=1}^{B} w_{bi} T_b^p \mathbf{e}_i = v_i^p - v_i'^p \end{aligned} \tag{5. 9}$$

for each vertex i and pose p. The above can be expressed as a linear system of the form $Ax = b$ where A is a block vector of N blocks. Each of these blocks of size 3Px3 contains the weighted transformation matrices, without the translation component, of the proxy joints that affects each vertex for all poses. The 4th column, that of the translation component, is removed because we want $\mathbf{e}_i$ to lie on the plane W=0 in homogenous coordinates. This is because it is added to $\mathbf{v}_i$ which already is on plane W=1 of the homogenous coordinates. If we denote by:

$$L_b^p = T(:,1:3)_b^p \tag{5. 10}$$

to be the linear part of the affine transformation matrix T then:

$$A = \begin{bmatrix} A^1 \\ A^2 \\ \vdots \\ A^N \end{bmatrix}, A^i = \begin{bmatrix} \sum_b w_{ib} L_b^1 \\ \sum_b w_{ib} L_b^2 \\ \vdots \\ \sum_b w_{ib} L_b^P \end{bmatrix} \tag{5. 11}$$

From the right, b is a block vector of N blocks. Each block is of size 3Px1 that contains the difference of the actual pose from the approximation pose for all poses:

$$b = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^N \end{bmatrix}, b^i = \begin{bmatrix} v_i^1 - v_i'^1 \\ v_i^2 - v_i'^2 \\ \vdots \\ v_i^P - v_i'^P \end{bmatrix} \tag{5. 12}$$

Solving this set on N systems $A^i x = b^i$ results in one correction vector for each vertex in the rest pose. The rest pose is thus altered and all subsequent operations must take place on this rest pose. This is also the rest pose that is stored by the algorithm as a result of the compression process.

Weight Corrections

Another correction process can be applied on the weight influences. Again having performed the fitting process we are searching for a weight w'_{bi} that is added to each corresponding weight influence to a vertex i, such the sum of the squared errors of the approximation in each pose is minimized:

$$\min\left(\sum_p \left\| \sum_i (w_{bi} + w'_{bi}) T^p \mathbf{v}_i - v_i^p \right\|^2 \right) \tag{5. 13}$$

Again this is equivalent to the least squares solution of :

$$\begin{aligned} &\left(\sum_{b=1}^{B} (w_{bi} + w'_{bi}) T_b^p \right) \mathbf{v}_i - v_i^p = 0 \Leftrightarrow \\ &\sum_{b=1}^{B} w_{bi} T_b^p \mathbf{v}_i + \sum_{b=1}^{B} w'_{bi} T_b^p \mathbf{v}_i - v_i^p = 0 \Leftrightarrow \\ &\sum_{b=1}^{B} w'_{bi} T_b^p \mathbf{v}_i = v_i^p - v_i'^p \end{aligned} \tag{5. 14}$$

for each vertex i and pose p. This can also be expressed as a system of linear equations of the form $Ax = b$ where A is a vector of blocks. Each block is of size 3PxB and contains the transformed rest pose vertex for each of the influencing proxy joints. Overall $A \in R^{3NPxB}$:

$$A = \begin{bmatrix} A^1 \\ A^2 \\ \vdots \\ A^N \end{bmatrix}, A^i = \begin{bmatrix} T_1^1\mathbf{v}_i & T_2^1\mathbf{v}_i & \cdots & T_B^1\mathbf{v}_i \\ T_1^2\mathbf{v}_i & T_2^2\mathbf{v}_i & & T_B^2\mathbf{v}_i \\ \vdots & \vdots & & \vdots \\ T_1^p\mathbf{v}_i & T_2^p\mathbf{v}_i & \cdots & T_B^p\mathbf{v}_i \end{bmatrix} \tag{5. 15}$$

Matrix b is constructed from the differences of the actual from the approximation location of each vertex in each pose and is the same as (5. 12):

$$b = \begin{bmatrix} b^1 \\ b^2 \\ \vdots \\ b^N \end{bmatrix}, b^i = \begin{bmatrix} v_i^1 - v_i'^1 \\ v_i^2 - v_i'^2 \\ \vdots \\ v_i^P - v_i'^P \end{bmatrix} \tag{5. 16}$$

Recall from (3. 2) and (3. 3) that weight need to satisfy certain properties. The addition of the correction weight must not disturb these conditions. Thus certain constrains need to be added in the optimization process.

Convex Weight Requirement

Constraining all weights, after the addition of the correction weight, to sum up to 1 can be achieved if the sum of the first B-1 weights of the correction set is constrained to be equal to the negative of the Bth weight. This is because the sum of the fitted weights is already 1. The following equation describes this notion:

$$\begin{aligned}
&\sum_{b=1}^{B}(w_{bi}+w'_{bi})=1 \Leftrightarrow \\
&\sum_{b=1}^{B}w_{bi}+\sum_{b=1}^{B}w'_{bi} \overset{\sum_{b=1}^{B}w_{bi}=1}{=1} \Leftrightarrow \\
&\sum_{b=1}^{B}w'_{bi}=0 \\
&\sum_{b=1}^{B-1}w'_{bi}=-w'_{Bi}
\end{aligned} \tag{5. 17}$$

Substituting w'_{Bi} in (5. 14) we get:

$$\begin{aligned}
&w_{1i}T_1^{\,p}\mathbf{v}_i+w_{2i}T_2^{\,p}\mathbf{v}_i+\ldots+w_{Bi}T_B^{\,p}\mathbf{v}_i= \\
&w_{1i}T_1^{\,p}\mathbf{v}_i+w_{2i}T_2^{\,p}\mathbf{v}_i+\ldots+\left(-\sum_{b=1}^{B-1}w_{bi}\right)T_B^{\,p}\mathbf{v}_i= \\
&w_{1i}(T_1^{\,p}-T_B^{\,p})\mathbf{v}_i+w_{2i}(T_2^{\,p}-T_B^{\,p})\mathbf{v}_i+\ldots+w_{(B-1)i}(T_{B-1}^{\,p}-T_B^{\,p})\mathbf{v}_i=v_i^{p}-v_i'^{\,p}
\end{aligned} \tag{5. 18}$$

From the above holds that this constraint can be added implicitly by subtracting the last column of matrix A from each of the other. Thus A becomes:

$$\mathfrak{A}=\begin{bmatrix}\mathfrak{A}^1\\ \mathfrak{A}^2\\ \vdots\\ \mathfrak{A}^N\end{bmatrix},\mathfrak{A}^i=\begin{bmatrix}\overbrace{\begin{matrix}T_1^1\mathbf{v}_i & T_2^1\mathbf{v}_i & \cdots & T_{B-1}^1\mathbf{v}_i\end{matrix}}^{B-1}\\ \begin{matrix}T_1^2\mathbf{v}_i & T_2^2\mathbf{v}_i & & T_{B-1}^2\mathbf{v}_i\\ \vdots & \vdots & & \vdots\\ T_1^{\,p}\mathbf{v}_i & T_2^{\,p}\mathbf{v}_i & \cdots & T_{B-1}^{\,p}\mathbf{v}_i\end{matrix}\end{bmatrix} \tag{5. 19}$$

and weight w'_{Bi} is removed from the unknowns vector since it can be computed once all other weights have been specified.

Non-negativity Constraint

Each weight must be non negative. For that we must ensure that after the addition of the corrections the corrected weight remains positive. This means:

$$\begin{aligned} & w_{bi} - w'_{bi} \geq 0 \Leftrightarrow \\ & w'_{bi} \leq -w_{bi} \end{aligned} \tag{5. 20}$$

This produces a problem because we have removed the last correction weight from the list of unknowns to satisfy the convexity requirement. Thus we cannot add the $w'_{Bi} \geq -w_{Bi}$ constraint implicitly. The walk-around for this was to request explicitly that:

$$\begin{aligned} & w'_{Bi} \leq -w_{Bi} \Leftrightarrow \\ & -\sum_{b=1}^{B-1} w_{bi} \leq -w_{Bi} \Leftrightarrow \\ & \sum_{b=1}^{B-1} w_{bi} \geq w_{Bi} \end{aligned} \tag{5. 21}$$

Solving this set on N systems $A^i x = b^i$ results in a correction for each weight influence of each vertex.

Both correction processes can be applied independently or in conjunction with the order of application presenting with no actual difference.

CHAPTER 6

IMPLEMENTATION AND RESULTS

6.1 **Implementation Details**

Our implementation was build using C# programming language. OpenTK, an OpenGL and GLSL shading language wrapper for C#, and was used for 3D visualization. Mircosoft .NET libraries where utilized for the creation of user interface and Matlab for complex scientific calculations. Interoperability between matlab and .NET was established via Matlab.NET builder which creates dlls containing matlab code that can be executed by .NET applications. The whole application was developed using Microsoft Visual Studio Development Environment, versions 2008 and 2010. Matlab Development environment was also utilized. For the dockable windows, the open source DockPanel Suite [24] was used.

The application has the ability to load multiple mesh files of Wavefront .obj format, to render an animation sequence. Multiple animation sequences can be loaded and processed simultaneously. It utilizes all the techniques that were mentioned in this report to approximate the animation and can render the result of the approximation, as well as visualize error distribution upon the surface. We also use GLSL shading language to create visualizations for clustering and proxy joints. The approximation results can be saved in xml files and again be reloaded. Our application can also execute decimation of a mesh at variable levels and apply approximation techniques one the decimated model. Figure 6. 1 shows a snapshot of our application.

All experiments were run on a Intel Core i7 880 3.06Ghz 6GB RAM, NVidia GForece 480 GTX, under Windows7 64-bit operating System. All animation sequences where created using Blender.

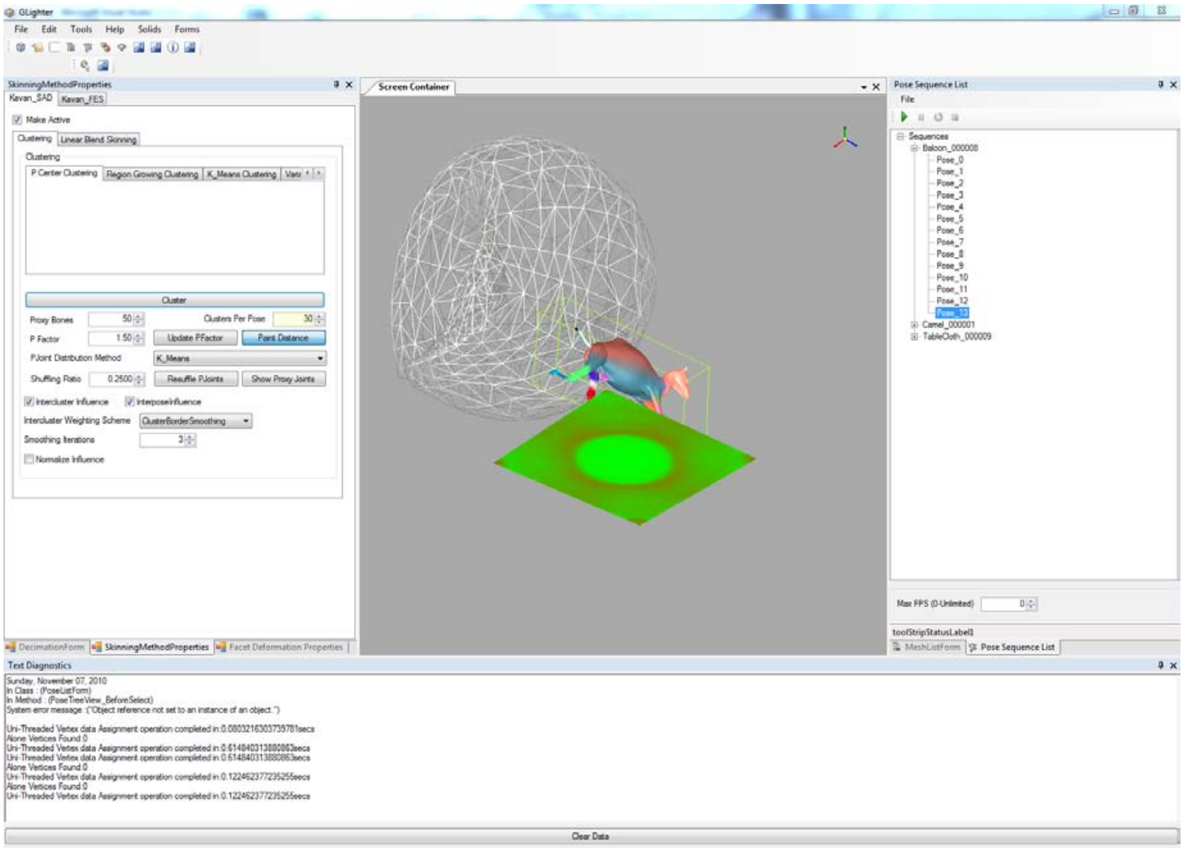


Figure 6. 1: Application User Interface screenshot

6.2 Test-bed Animation Sequences

We present four of the models that were used to test the results of our approximation techniques. Each model was picked to represent a specific type of deformation.

Tablecloth

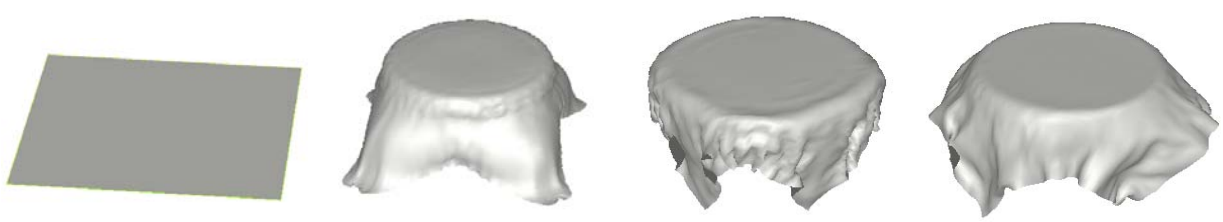


Figure 6. 2: Tablecloth animation sequence snapshots

Tablecloth is an animation sequence of a piece of cloth falling upon a table and starting to deform. Tablecloth is a characteristic example of moderately highly deformable animation in which large areas remain virtually un-deformed. Of the tablecloth animation, 70 poses where used.

Flapping Flag

Figure 6. 3: Flapping Flag animation sequence snapshot

Flapping flag is a model of a flag deforming against a wind field. Flag presents with extreme deformation to the right part and moderate to the left, which is the part adjacent to the pole. 70 poses of the flag animation were also used

Collapsing Camel

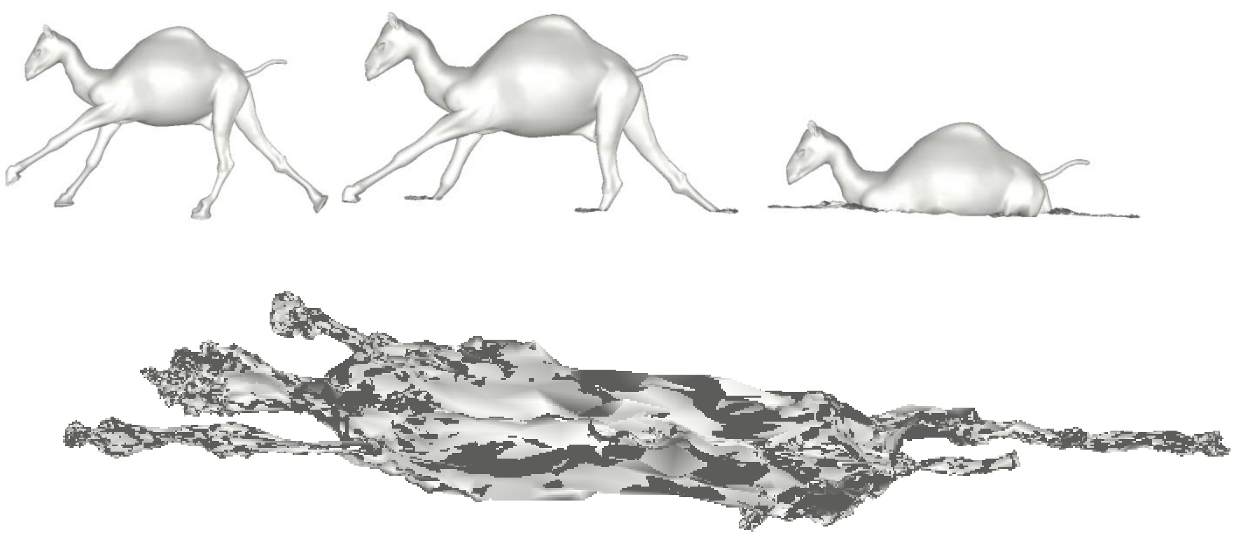


Figure 6. 4: Collapsing Camel animation sequence snapshot

Collapsing camel is an animation sequence of a camel cloth model that collapses under is own weight. This animation sequence is an example of extreme deformation. Collapsing camel is a 35 pose sequence.

Balloon

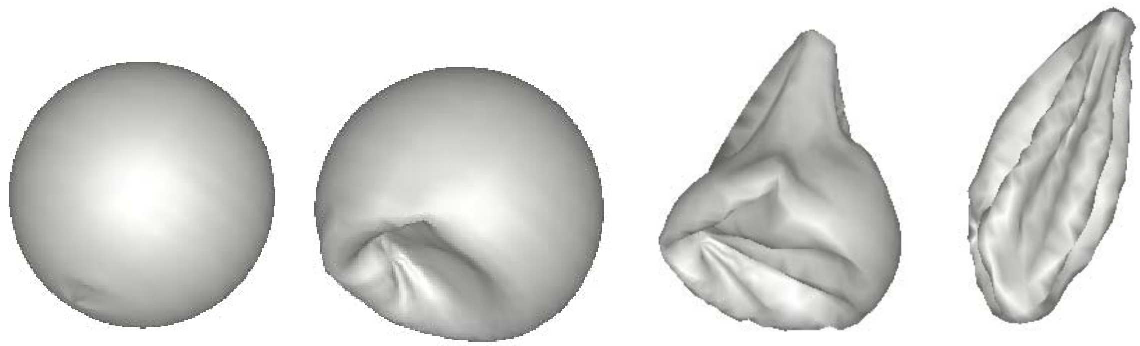


Figure 6. 5: Deforming Balloon

Balloon animation sequence depicts a cloth sphere that falls upon a pole and is deformed. Balloon generally presents with overall moderate deformation. 60 poses of the balloon sequence where used.

6.3 **Error Metrics**

For the evaluation of the approximation process three metrics where used. Two of them provide a percentage of deviation. They have been used in [9] and [20], so we used them for comparison. The first is called *Distortion Percentage* [8] and is given by:

$$dE = 100 * \frac{\left\| A_{orig} - A_{appr} \right\|_F}{\left\| A_{orig} - A_{avg} \right\|_F} \tag{6. 1}$$

where A_{orig} is a 3PxN matrix that contains the original coordinates of each vertex throughout the animation sequence. A_{appr} is structured similar to A_{appr} and contains the approximation coordinates. A_{avg} contains in its first 3 rows the average coordinates of each vertex and repeats for each pose. Because this metric is sensitive to global motion applied to the entire mesh, another metric which is translation invariant is used. It is similar to the first, only it divides by the scalar $\sqrt{3NP}$ to get the average deformation throughout the sequence:

$$E_{RMS} = 100 * \frac{\left\| A_{orig} - A_{appr} \right\|_F}{\sqrt{3NP}} \tag{6. 2}$$

The third metric is a variation of Hausdorff metric, only we average the average of the minimum distances:

$$d_{avg}(X,Y) = avg\left\{ \underset{x\in X}{avg}\left(\inf_{y\in Y} d(x,y) \right), \underset{y\in Y}{avg}\left(\inf_{x\in X} d(x,y) \right) \right\} \tag{6. 3}$$

6.4 **Decimated Approximation Results**

We have conducted experiments with the purpose of presenting the results of Deformation Based Fitting and the implications of approximating a decimated animation sequence. Uniform Fitting (P-Center based) is used as a benchmark for our method. The two fitting methods are applied and compared on various levels of decimation from 0% (no decimation) to 60%. The legend of the following figures is read as follows:

i) OS(PC): Optimal-Error Minimizing Decimation on P-Center based fitting

ii) OS(Deform):Optimal-Error Minimizing Decimation on Deformation Based Fitting

iii) SE(PC): Second Edge Decimation on P-Center based Fitting

iv) SE(Deform): Second Edge Decimation on Deformation based Fitting

v) OS(Corr): Optimal-Error Minimizing Decimation on P-Center based fitting with corrections applied

vi) OS(Corr_Deform): Optimal-Error Minimizing Decimation on Deformation Based Fitting with corrections applied

vii) SE(Corr): Second Edge Decimation on P-Center based Fitting with corrections applied

viii) SE(Corr_Deform): Second Edge Decimation on Deformation Based Fitting with corrections applied

Tablecloth Results

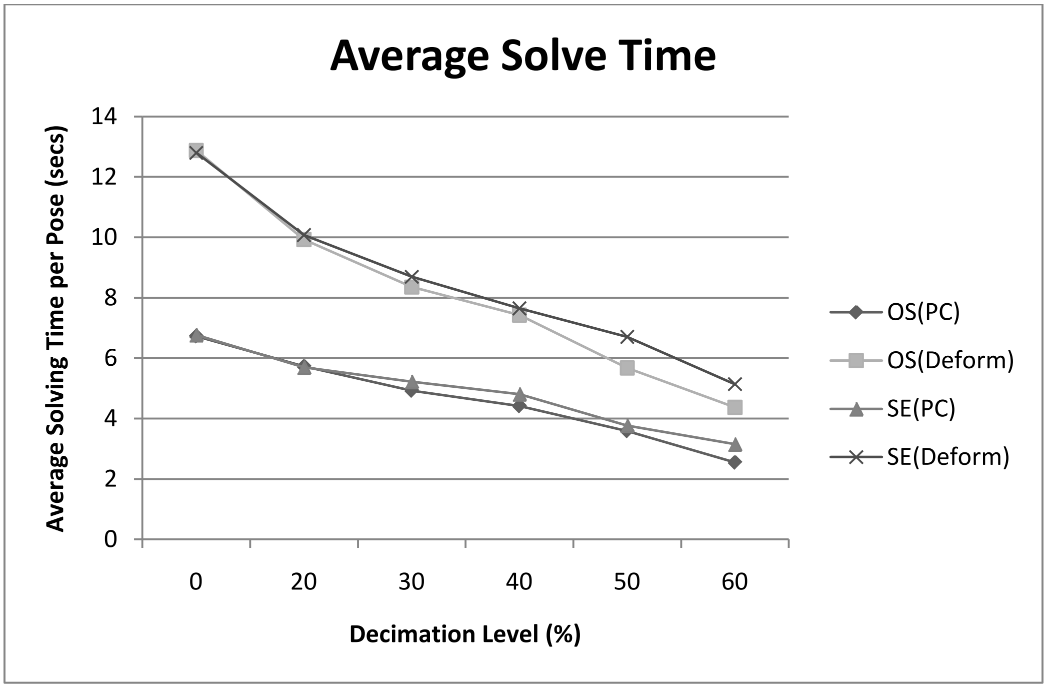


Figure 6. 6: Tablecloth decimation average solving times on various decimation levels

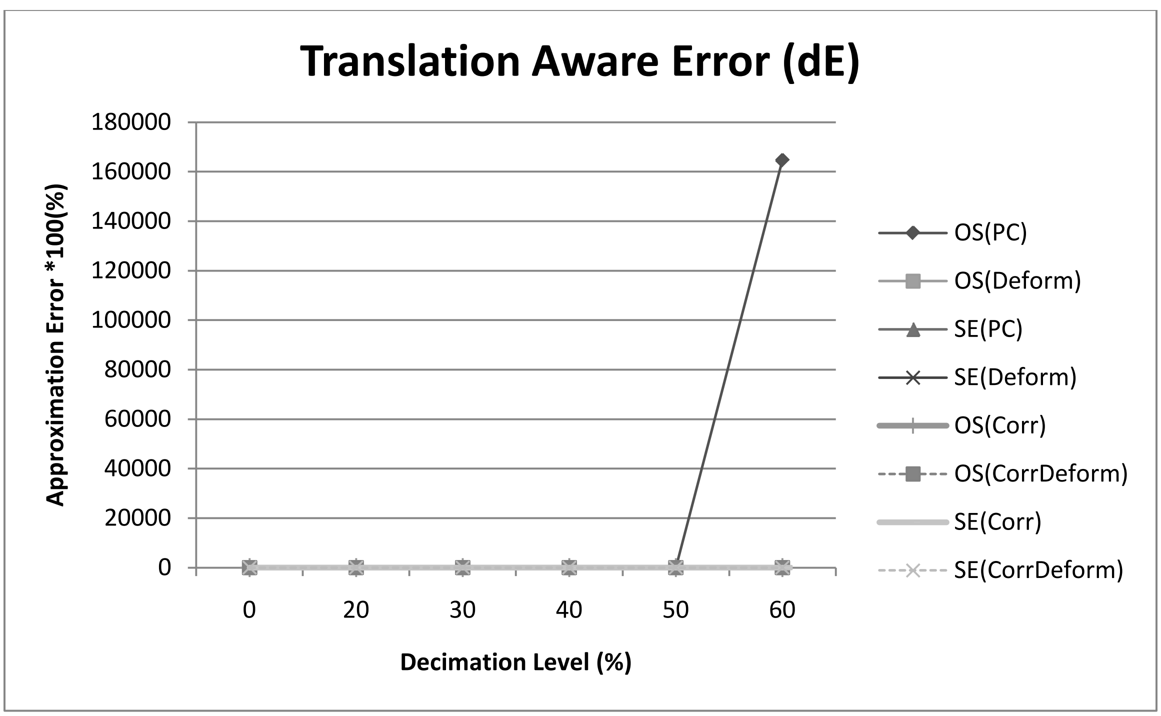


Figure 6. 7: Tablecloth approximation error dE progression due to decimation.

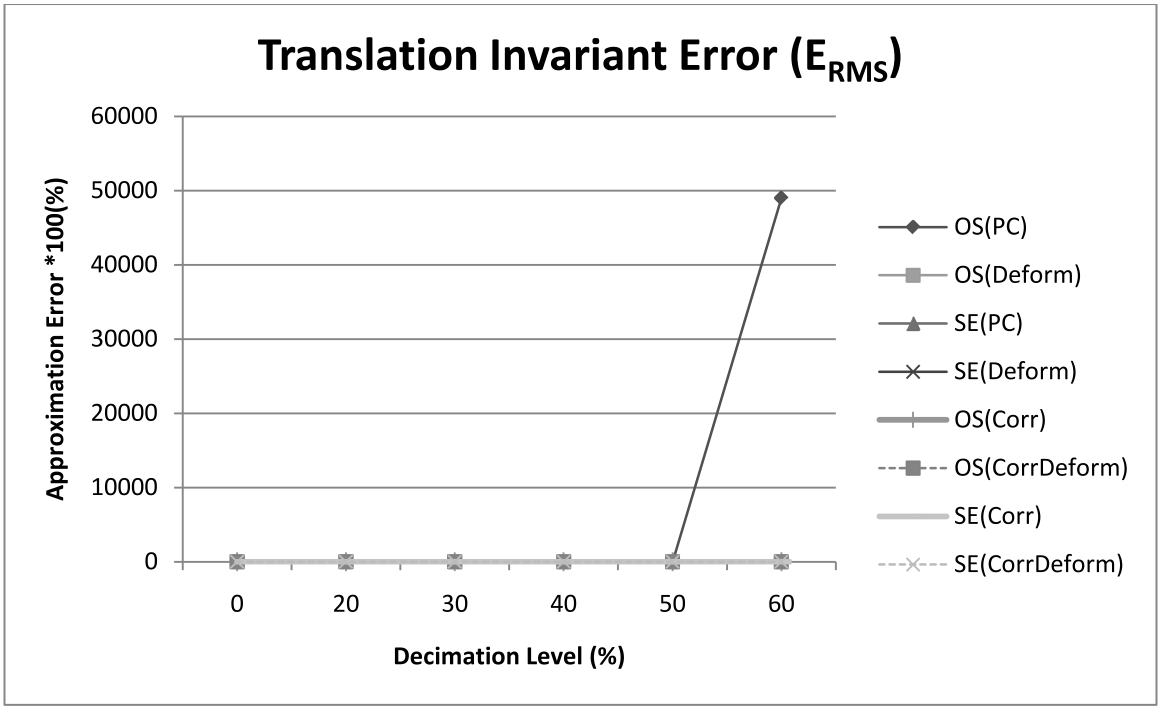


Figure 6. 8: Tablecloth approximation error E_{RMS} progression due to decimation.

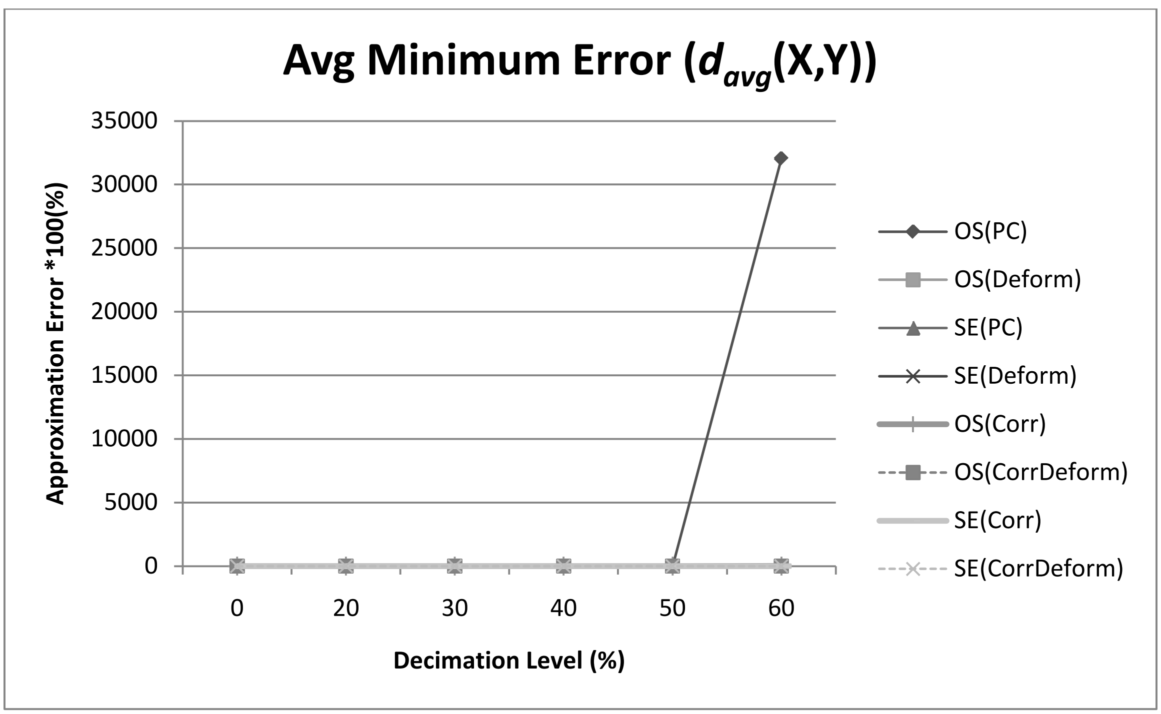


Figure 6. 9: Tablecloth approximation error $d_{avg}(X,Y)$ progression due to decimation.

Flag Results

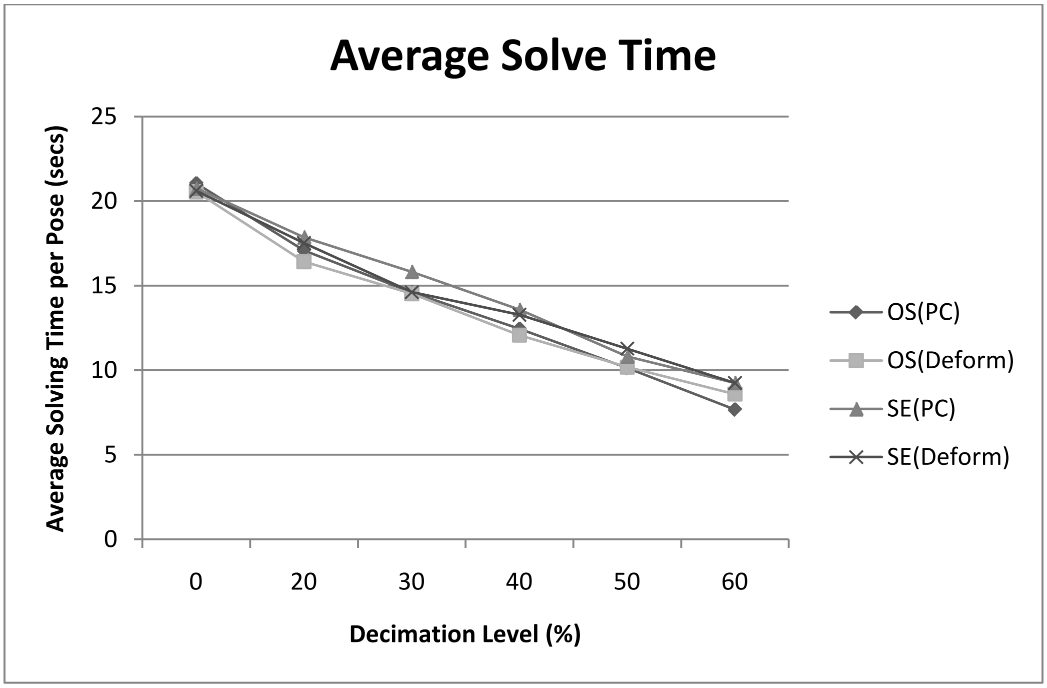


Figure 6. 10: Flag decimation average solving times on various decimation levels

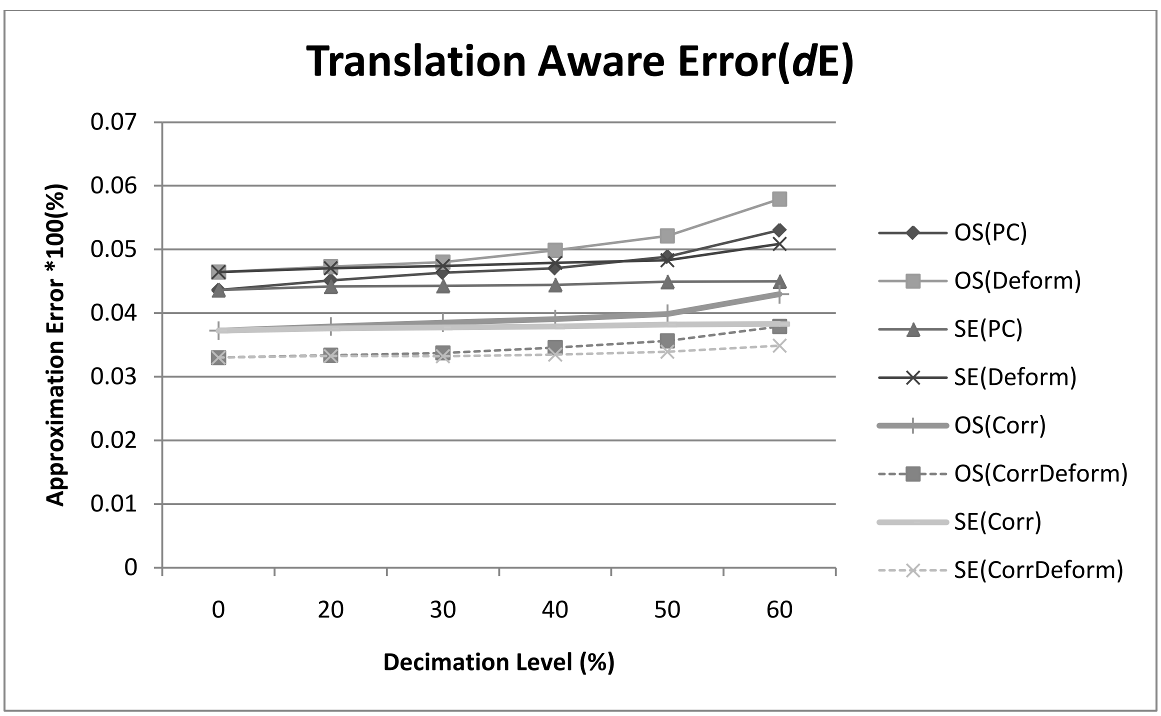


Figure 6. 11: Flag approximation error dE progression due to decimation.

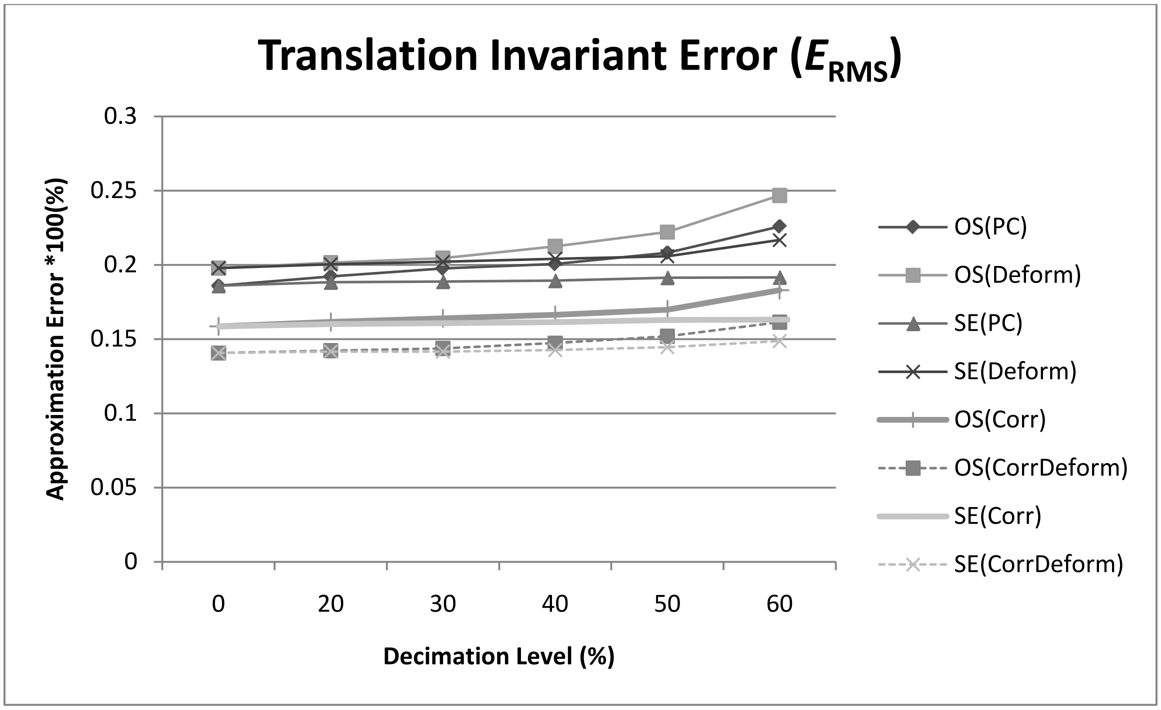


Figure 6. 12: Flag approximation error E_{RMS} progression due to decimation.

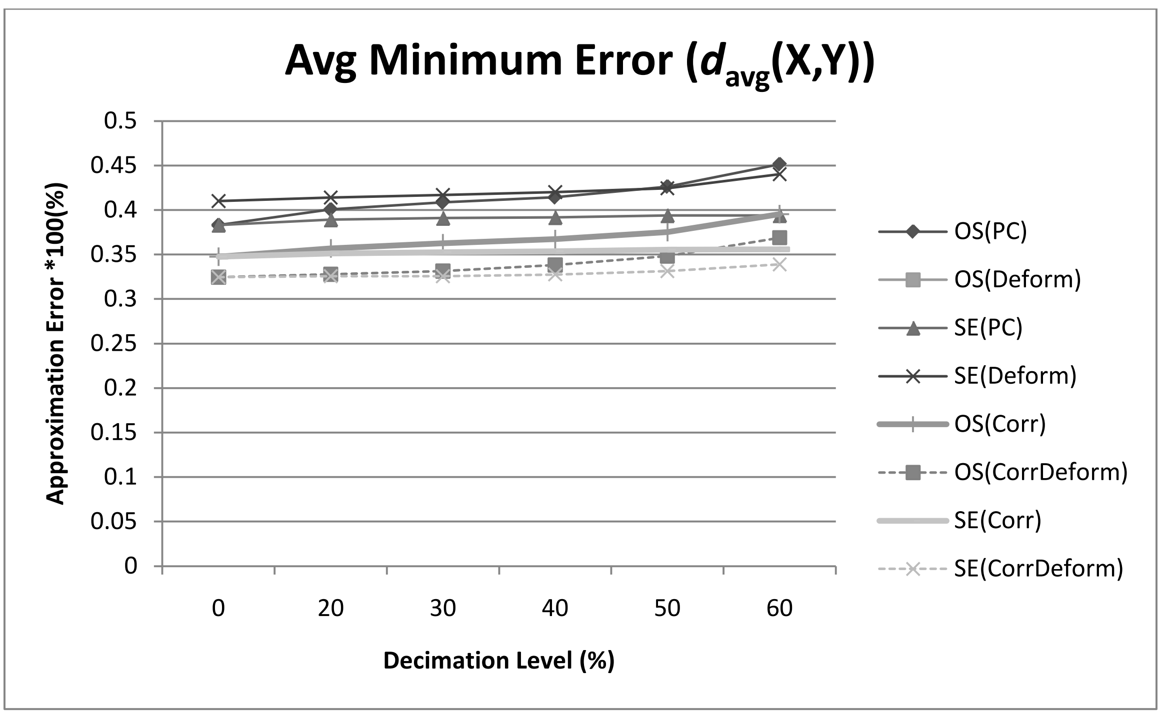


Figure 6. 13: Flag approximation error $d_{avg}(X,Y)$ progression due to decimation.

Camel Collapse Results

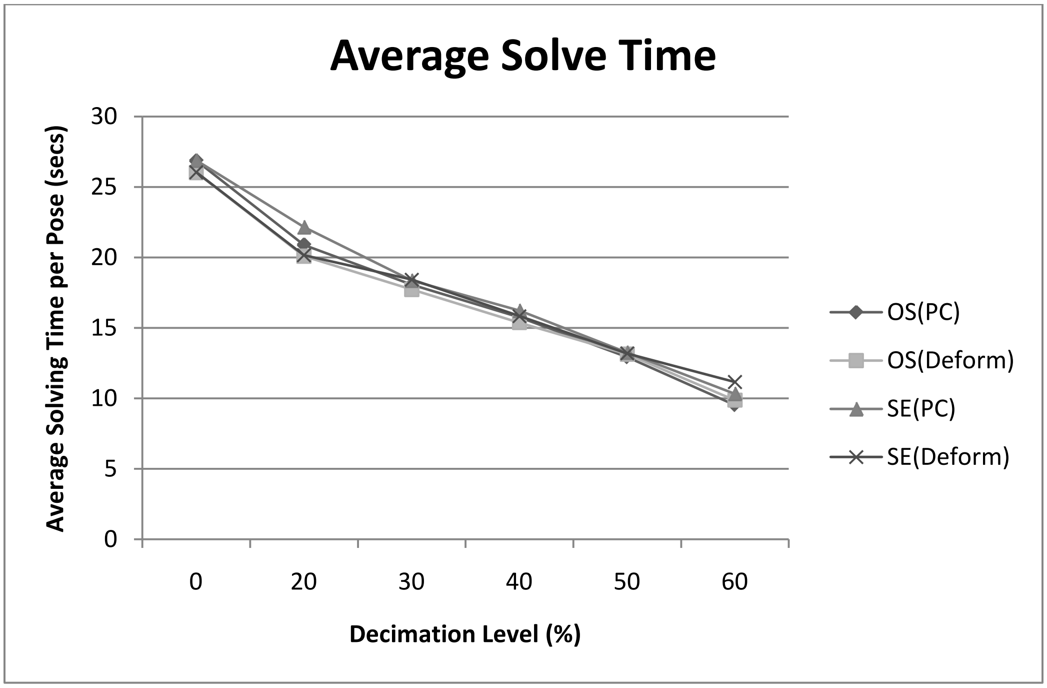


Figure 6. 14: Camel collapse decimation average solving times on various decimation levels

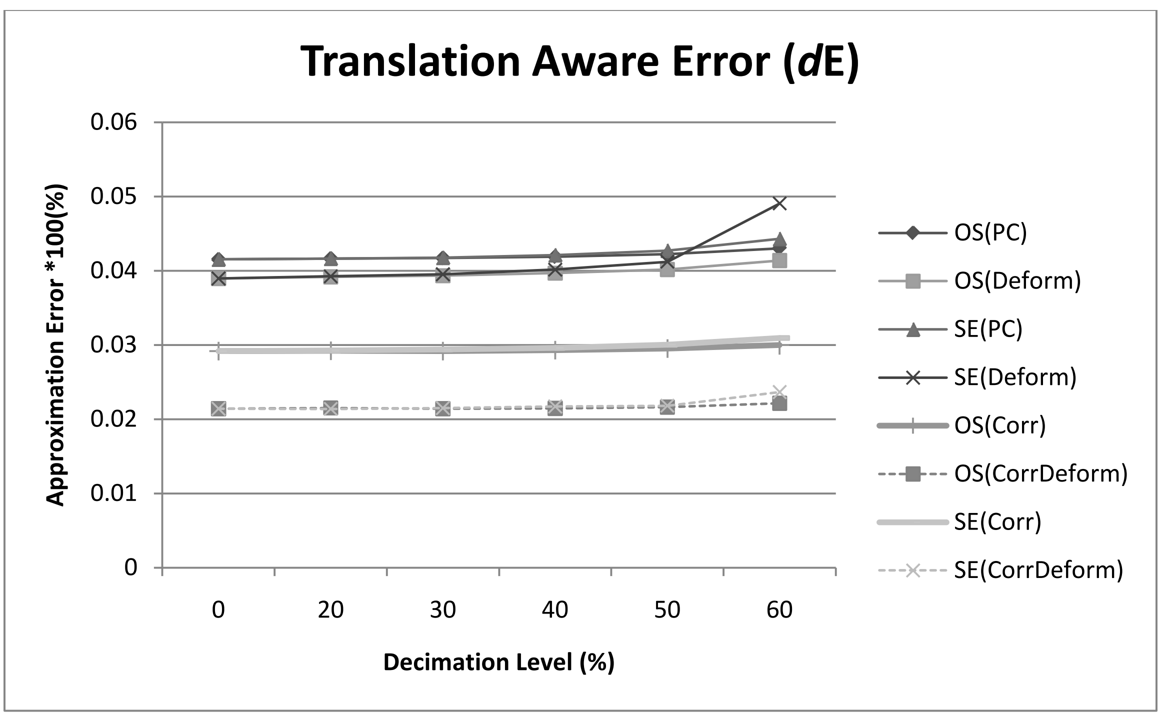


Figure 6. 15: Camel collapse approximation error dE progression due to decimation.

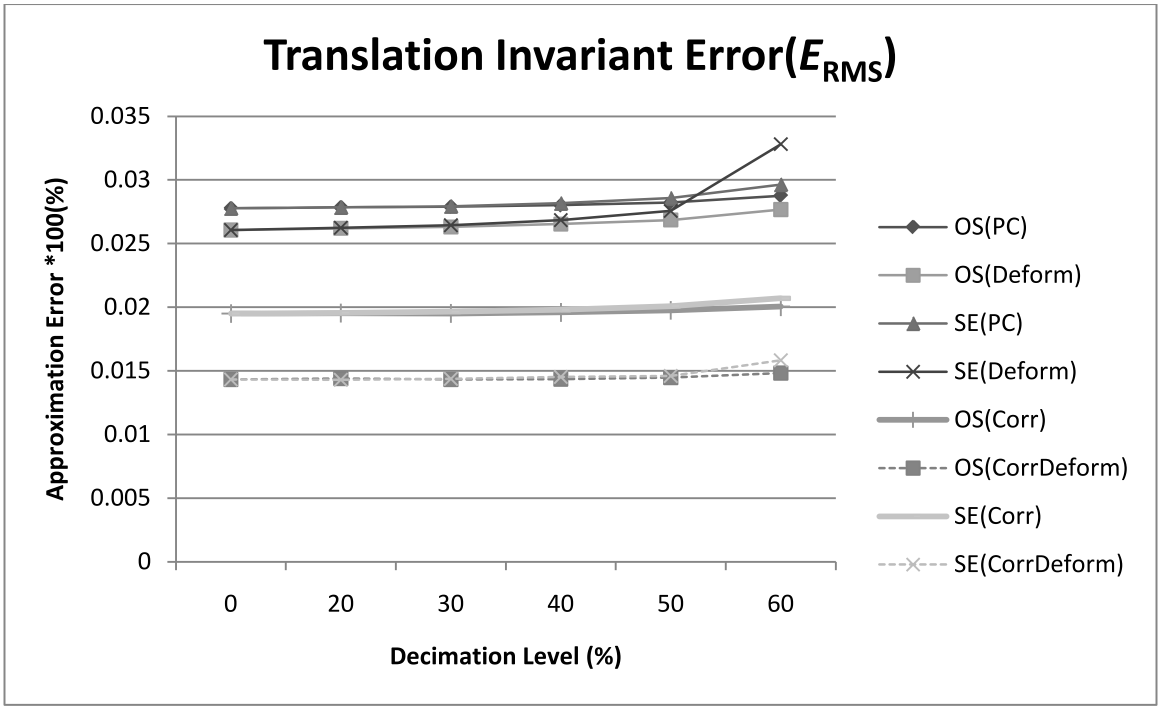


Figure 6. 16: Camel collapse approximation error E_{RMS} progression due to decimation.

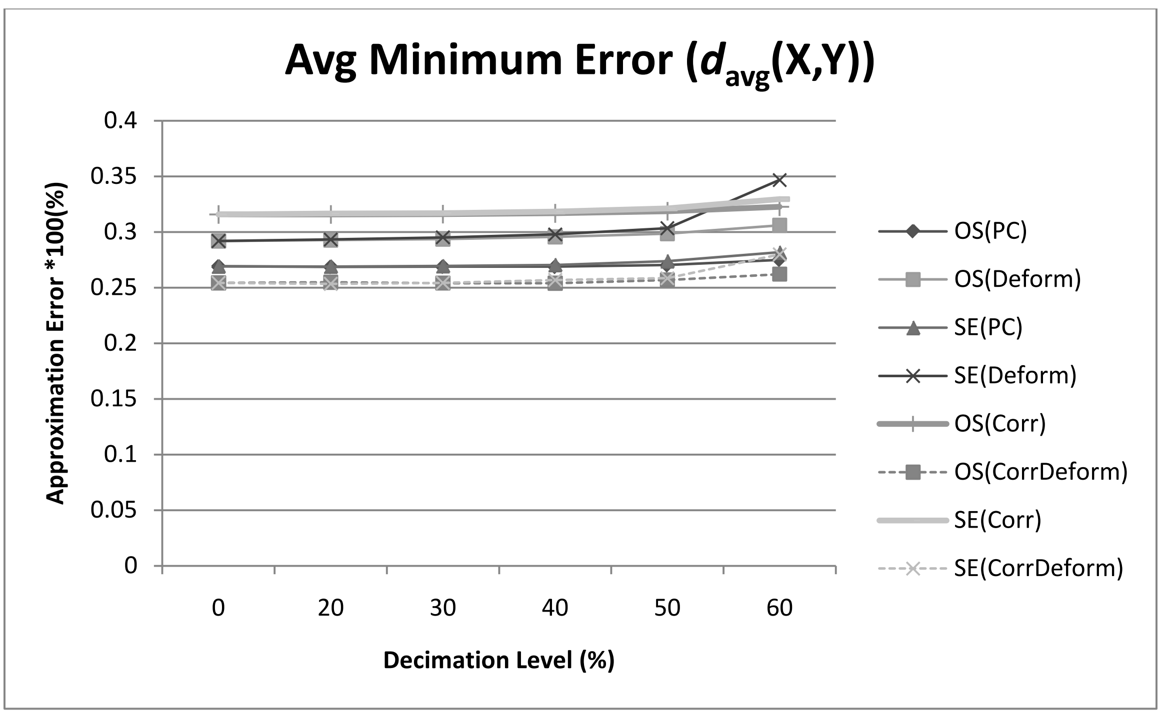


Figure 6. 17: Camel collapse approximation error $d_{avg}(X,Y)$ progression due to decimation.

Balloon Results

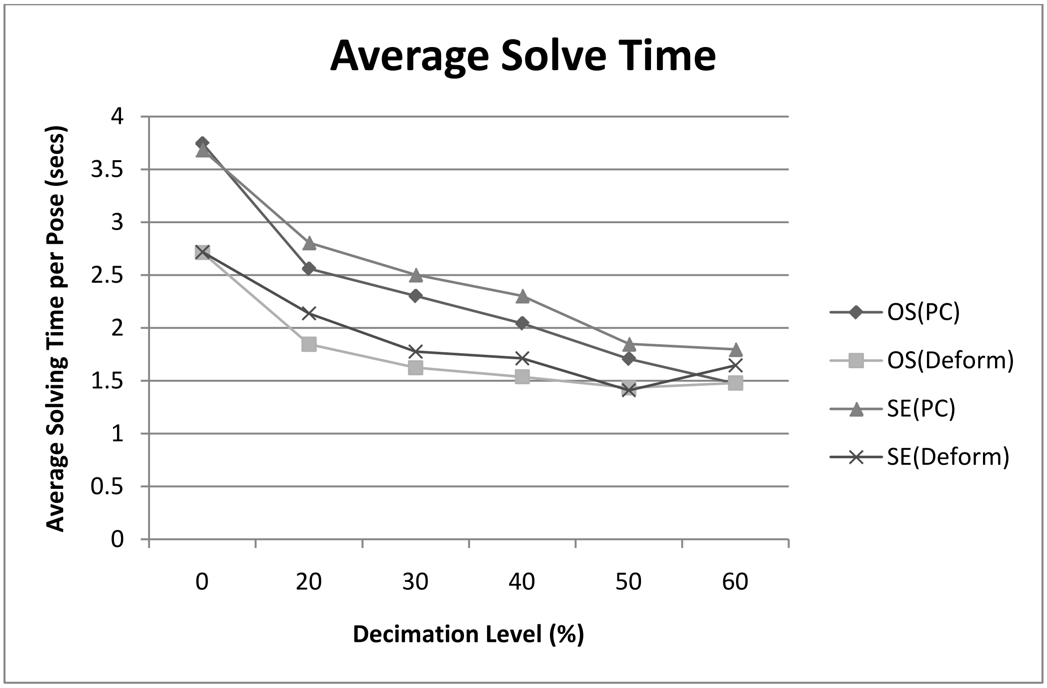


Figure 6. 18: Balloon decimation average solving times on various decimation levels

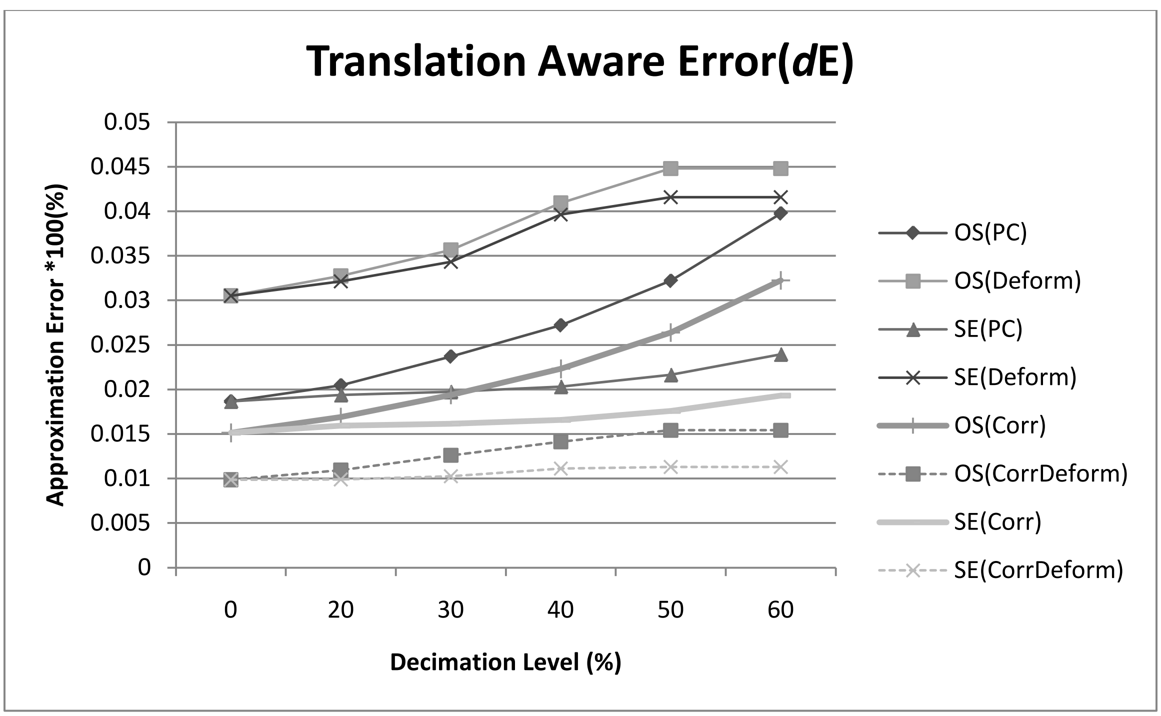


Figure 6. 19: Balloon approximation error dE progression due to decimation.

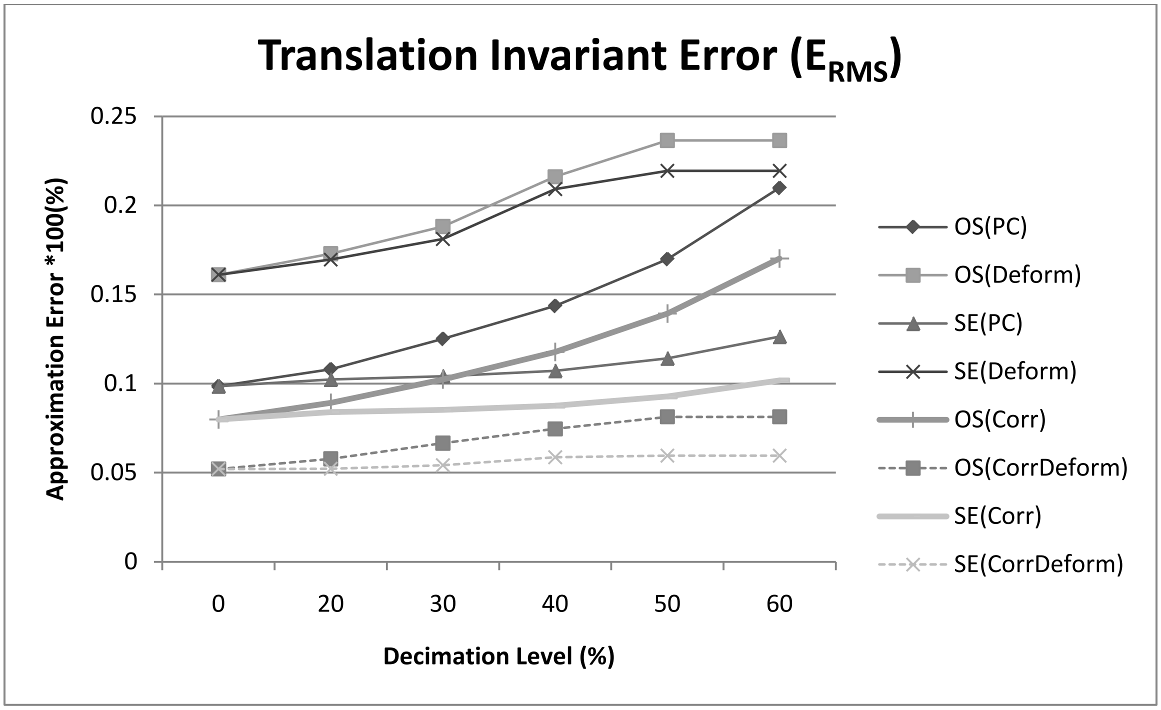


Figure 6. 20: Balloon approximation error E_{RMS} progression due to decimation.

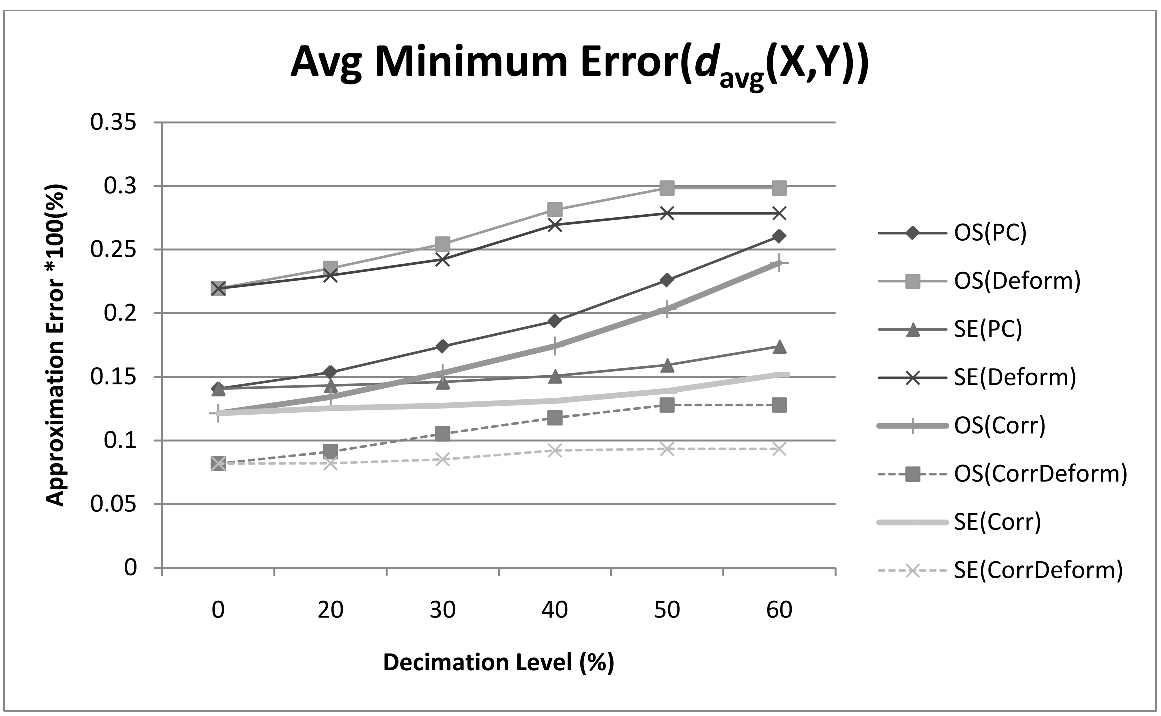


Figure 6. 21: Balloon approximation error $d_{avg}(X,Y)$ progression due to decimation.

Results show that P-Center based fitting and deformation based fitting are close to each other. Execution times may vary according to the initial clustering which is random in both methods, and according to different values for P-Factor and weight smoothing for P-Center and Deformation based fitting respectively.

Also, experiments show that the improvement in execution time is not followed by an analogous increase in the approximation error. Reduction can reach up to 4 times the original execution time while error increase is rarely more than 10%. These results also depict that our method is very close the uniform distribution results.

Finally it should be noted that when applying the corrections we proposed to the Uniform Fitting the approximation quality tents to be lower of that, using Deformation Based Fitting. The difference may seem small but at this low level of approximation error, it is notable.

CHAPTER 7

CONCLUSIONS AND FUTURE WORK

We have presented a method that uses the deformation gradient information of an animated sequence to partition the model into areas of similar deformation. These areas can be used to distribute the proxy bones of the matrix palette skinning process in a deformation based manner that results in more proxy bones being positioned at areas of higher deformation. We evaluated certain possible methods that can be used to identify these areas and presented a region growing variation of k-means clustering algorithm that accomplishes this task. A weight fitting technique has also been suggested. It is based on the notion of convolution and used to smoothly propagate inter-cluster influence. To accelerate the matrix palette skinning process we performed mesh decimation.

Results show that deformation based fitting produces slightly higher approximation error than the P-Center based one. Visual results however show that the approximation is free of the bumps that denote the presence of cyclic regions around proxy joints. Although a certain amount of rounding due to over-fitting is evident it is also possible that the model is scaled to a certain degree throughout the process and thus this error is propagated. Recall that we allow more than four weight influences to be applied to a vertex. This increases considerably the preprocessing time. Multiple weight influences have been efficiently substituted by less in the case of articulated objects but whether it is suitable to perform this on highly deformable ones has yet to be studied. Uniform propagation of influences may also be sub-optimal. Distance is probably not the only criterion for weight assignment and another scheme, probably deformation based, may need to be considered. Our decimation scheme with our deformation based clustering can also be extended. The potential of storing only the decimated pose along with the weights and the clustering and reconstruct the model using this information only is worth investigating.

BIBLIOGRAPHY

[1] W. R. Hamilton. “On quaternions, or on a new system of imaginaries in algebra”, *Philosophical Magazine. Vol. 25, n 3. p. 489-495. 1844*.

[2] M. Garland, Y. Zhou. “Quadric-Based Simplification in Any Dimension”, *ACM Transactions on Graphics, Vol. 24, No. 2, April 2005, p. 209–239.*

[3] P. Lindstorm. “Out-of-core simplification of large polygonal models”, *Proceedings of SIGGRAPH 2000, p. 259–262.*

[4] R. W. Sumner, J. Popovic. “Deformation Transfer for triangle meshes”, *ACM Transactions on Graphics 23,3 (Aug. 2004), p. 399-405*

[5] R. W. Sumner, M. Zwicker, C. Gotsman, J. Popovic. “Mesh-Based Inverse Kinematics”, *Proceedings of ACM SIGGRAPH 2005, p.488-495.*

[6] G. H. Golub, C. F. Loan. “Matrix Computations*”, third ed. Johns Hopkins University Press, Baltimore, 1996.*

[7] K. Shoemake, T. Duff. “Matrix Animation and Polar Decomposition”. *Proceedings of the Conference on Graphics Interface ’92, p.258-264.*

[8] D. L. James, C. D. Twigg. “Skinning Mesh Animations*”, ACM Trans. Graph. 24,3, p.399-407*

[9] L. Kavan, R. McDonnell, S. Dobbyn, J. Žára, C. O’ Sullivan. “Skinning Arbitrary Deformations”. *Proceedings of the 2007 symposium on Interactive 3D graphics and games -I3D ’07*, p53-60.

[10] C. Paige, M. Saunders. “LSQR: An Algorithm for Sparse Linear Equations and Sparse Least Squares”. *ACM Transactions on Mathematical Software, Vol 8,No. 1, March 1982, p.43-71.*

[11] J. M. McCarthy. ”Introduction to Theoretical Kinematics”. *MIT Press, Cambridge, MA, USA*

[12] T. F. Gonzales. “Clustering to Minimize Maximum Intercluster Distance”. *Theoretical Computer Science, 38, (1985), p.293-306*

[13] D. Cohen-Steiner, P. Allienz, M.Desbrun. “Variational Shape Approximation”, *ACM Transactions on Graphics (2004), 23,3, p.905-914.*

[14] B. U. Kim, W. W. Feng, Y. Yu. “Real-time data driven deformation with affine bones”. *The Visual Computer, 26,6-8, June 2010,p.487-495*

[15] M. Garland, P. Heckbert. “Surface Simplification Using Quadric Error Metrics”. *In Proceedings of SIGRAPH ’97 (1997), p.209-218*

[16] R. Ronfart, J. Rossignac. “Full-range approximation of triangulated polyhedra.”*Computer Graphics Forum, 15(3), Aug. 1996. Proc. Eurographics 1996.*

[17] A. Mohr, M. Gleicher. “Deformation sensitive decimation.”. *Technical Report, University of Wisconsin, 2003.*

[18] P. G. Kry, D. L. James, D. K. Pai. “EigenSkin: Real Time Large Deformation Character Skinning in Hardware”. *In the Proceedings of the 2002 ACM SIGGRAPH /Eurographics Symposium on Computer Animation, ACM Press, p.153-159*

[19] L. Kavan, S. Collins, J. Žára, C. O’Sullivan.”Skinning with Dual Quaternions”. *In Proceedings of symposium on Interactive 3D graphics and games, p.39-46.*

[20] L. Kavan, P. Sloan, C. O’Sullivan. ”Fast and Efficient Skinning of Animated Meshes”. *Eurographics 2010.*

[21] K. G. Der, R. W. Sumner, J. Popovic. “Inverse Kinematics for Reduced Deformable Models”. *Inverse kinematics for reduced deformable models, ACM Transactions on Graphics (TOG), v.25 n.3, July 2006*

[22] T. Y. Lee, Y. S. Wang, T.G. Chen.”Segmenting a deforming mesh into near-rigid components”. *The Visual Computer 2006, 22, p.729-739*

[23] S. Wuhrer, A. Brunton. “Segmenting Animated Objects into Near-Rigid Components”. *Visual Computer 2010, 26, p.147-155*

[24] http://dockpanelsuite.sourceforge.net/

SHORT VITA

George Antonopoulos was born on June 16, 1980 in Xanthi. He graduated from the 2nd High School of Chios and received his B.Sc. degree in 2007 from the Department of Computer Science of the University of Ioannina. He is currently pursuing his M.Sc degree from the same department. His research interests include 3D animation and software engineering.