RICE UNIVERSITY

Intuitive Methods for 3D Shape Deformation

by

Scott Schaefer

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

Doctor of Philosophy

Approved, Thesis Committee:

Joe Warren, Professor
Computer Science

Ron Goldman, Professor
Computer Science

Richard Baraniuk, Victor E. Cameron
Professor
Electrical and Computer Engineering

Houston, Texas

April, 2006
INFORMATION TO USERS

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleed-through, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.
Intuitive Methods for 3D Shape Deformation

Scott Schaefer

Abstract

Deformation is a key component in many applications including virtual surgery and the animation of digital characters in the movie industry. Previous deformation methods either require non-intuitive ways of specifying the deformation or have been too expensive to compute in real-time. We focus on three methods for creating intuitive deformations of 3D shapes. The first method is a new, smooth volumetric subdivision scheme that allows the user to specify deformations using conforming collections of tetrahedra, which generalizes the widely used Free-Form Deformation method. The next technique extends a fundamental interpolant in Computer Graphics called Barycentric Coordinates and lets the user manipulate low-resolution polygon meshes to control deformations of high-resolution shapes. Finally, we conclude with our work on creating deformations described by collections of points using Moving Least Squares.
Acknowledgements

Joe Warren, my advisor, has my gratitude for all of the help/guidance he's given me over the past six years. Without Joe, much of this work would never have been possible. He has taught me much during my graduate career and, hopefully, I've taught him something as well.

I would also like to thank my committee members for the comments and suggestions throughout this process.

To my wife, Amber, I would like to express how much she has meant to me. Without her support, much of this work would not have been possible.
Contents

Abstract i
List of Illustrations v
List of Tables ix

1 Introduction 1
  1.1 Previous Work ................................................. 5
  1.2 A Simple 2D Deformation ...................................... 7

2 Tetrahedral Subdivision 9
  2.1 Previous work .................................................. 12
  2.2 A tetrahedral subdivision scheme .......................... 14
    2.2.1 Linear subdivision .................................... 14
    2.2.2 Smoothing ............................................... 17
  2.3 Application to deformations ................................ 19
  2.4 Smoothness analysis ........................................... 21
    2.4.1 Interior of a base tetrahedron ....................... 22
    2.4.2 Faces of the base mesh ................................ 24
    2.4.3 Edges of the base mesh .............................. 27
    2.4.4 Vertices of the base mesh .......................... 28
  2.5 Conclusions .................................................... 29

3 Mean Value Coordinates 31
  3.1 Mean value interpolation .................................... 35
  3.2 Coordinates for piecewise linear shapes ............... 37
3.2.1 Mean value coordinates for closed polygons .......................... 37
3.2.2 Mean value coordinates for closed meshes ......................... 38
3.2.3 Robust mean value interpolation .................................. 40
3.3 Applications and results .............................................. 42
  3.3.1 Boundary value interpolation .................................. 42
  3.3.2 Volumetric textures ............................................. 44
  3.3.3 Surface Deformation ............................................. 44
  3.3.4 Implementation ................................................ 46
3.4 Conclusions and Future Work ....................................... 48

4 Moving Least Squares Deformation ..................................... 50
  4.1 Moving Least Squares deformation .................................. 52
    4.1.1 Affine deformations ........................................ 55
    4.1.2 Similarity deformations .................................... 56
    4.1.3 Rigid deformations ......................................... 57
  4.2 Deformation with line segments ..................................... 60
    4.2.1 Affine lines ................................................ 62
    4.2.2 Similarity lines ............................................. 64
    4.2.3 Rigid lines ................................................ 66
  4.3 Blended family .................................................... 67
  4.4 Applications and implementation ................................... 68
    4.4.1 Image morphing .............................................. 68
    4.4.2 Implementation ............................................. 70
  4.5 Conclusions ....................................................... 71

5 Conclusions and Future Work ........................................... 73

Bibliography ............................................................... 75
Illustrations

1.1 An example of a high-resolution model of David’s head with 4 million polygons (left). A close-up of the highlighted region of David’s eye illustrating the number of polygons (right). ........................................ 2

1.2 Rest pose of an armadillo man (left). Deformed poses created with the technique in Chapter 3. ................................................................. 3

1.3 The left shows the rest position of the armadillo man with a polygon mesh used as a deformation handle shown in wireframe. On the right, the deformation handle is modified, which induces a smooth deformation on the high-resolution figure of the armadillo man. ..... 4

1.4 Original image of a wooden gingerbread man (left) with a triangle as the deformation handle having vertices $p_i$. In the middle, we represent each point $v$ of the image as a combination of the $p_i$ using the areas $A_i$ of the wedges. Deforming the vertices $p_i$ to $q_i$ induces a deformation on the image (right). ................................. 7

2.1 Gingerbread man with a polygon approximating its shape (left). We triangulate the interior of the polygon and apply the simple deformation method from Chapter 1 to each triangle individually (middle two images). The resulting deformation (right) is not smooth across triangle boundaries. ......................................................... 9

2.2 From left to right: initial deformation, linear subdivision, smoothing and the final deformation after one round of subdivision. .............. 10
2.3 A sequence of deformations produced by subdivision with 0, 1 and 2 rounds of subdivision. ................................................. 11

2.4 Close-up of the cerebellum on a cross-section of a mouse brain. Initial embedding in a uniform grid and its deformation using free-form deformations (left). Subdivision surface that models the boundaries of the different anatomical regions and the explicit deformation of those boundaries (right). ................................................. 12

2.5 Splitting a tetrahedron generates an octahedron in the middle. Splitting the octahedron into tetrahedra requires choosing a diagonal. 15

2.6 Linear subdivision splits a tetrahedron into four tetrahedra and an octahedron (top). An octahedron is split into six octahedra and eight tetrahedra (bottom). .................................................. 16

2.7 Centroid masks for tetrahedra/octahedra. The highlighted vertex is the vertex being repositioned by smoothing. 18

2.8 Smoothing pass for tet/oct subdivision ..................................... 19

2.9 Subdivision defines a map \( f \) between points in a rest configuration to a deformed configuration. ........................................... 20

2.10 Surface deformation via tetrahedral subdivision. Initial shape shown (left) and three deformed poses (right). ................................. 21

2.11 Initial set of tetrahedra, subdivided surface, deformed surface and cut interior. Parameter lines are smooth after deformation. ............... 22

2.12 The subdivision mask for the linear subdivision on the uniform grid \( M \). 24

2.13 Two face-adjacent tetrahedra subdivided and opened at the shared face. Subdivision generates tet/tet and oct/oct pairs along that face. 25

2.14 Characteristic map for edges of valence 3 through 10. .................. 27

2.15 Characteristic maps for several low valence configurations of tetrahedra around a vertex. ................................................. 29
3.1 Comparison of deformations produced via Subdivision using two
different triangulations. On the left are the initial triangulations of
the control polygon. On the right are the deformation produced with
the same vertex locations, but different triangulations. The
deformations are similar but not identical. .......................... 31

3.2 Undeformed gingerbread man with control polygon (left) and the
deformation created using mean value coordinates on the right. .... 32

3.3 Interpolating hue values at polygon vertices using Wachspress
coordinates (a, b) versus mean value coordinates (c, d) on a convex
and a concave polygon. .................................................. 33

3.4 Mean vector $m$ on a circular arc $E$ with edge normals $n_i$ (a) and on a
spherical triangle $T$ with arc lengths $\theta_i$ and face normals $n_i$. .... 40

3.5 Mean value coordinates on a triangular mesh .......................... 43

3.6 Original model of a cow (top-left) with hue values specified at the
vertices. The planar cuts illustrate the interior of the function
generated by 3D mean value coordinates. ............................ 44

3.7 Textured bunny (top-left). Cuts of the bunny to expose the
volumetric texture constructed from the surface texture. ............ 45

3.8 Original horse model with enclosing triangle control mesh shown in
black (a). Several deformations generated using our 3D mean value
coordinates applied to a modified control mesh (b,c,d). ............... 46

3.9 Original model and surrounding control mesh shown in black
(top-left). Deforming the control mesh generates smooth
deformations of the underlying model. ............................... 47

3.10 Number of evaluations per second for various models. ............... 48
4.1 Deformation of the test shape from figure 4.2 using thin-plate splines (left). The deformation is smooth but lacks realism. On the right we use the method by Igarashi et al. shown with triangulation (right). The lack of smoothness is clearly visible in the wood grain.  

4.2 Deformation using Moving Least Squares. Original image with control points shown in blue (a). Moving Least Squares deformations using affine transformations (b), similarity transformations (c) and rigid transformations (d).  

4.3 Original image (left) and its deformation using the rigid MLS method (right). After deformation, the face is thinner and she is smiling.  

4.4 Original image (left) and its deformation using the rigid MLS method (right).  

4.5 Deformation of the Leaning Tower of Pisa. From left to right: original image, Affine MLS, Similarity MLS and Rigid MLS deformations.  

4.6 Comparison of the line deformation method of Beier et al. (left) with the Rigid MLS deformation (right).  

4.7 From left to right: original image, rigid MLS deformation, similarity MLS deformation, blended deformation with eyes set to rigid and nose to similarity.  

4.8 Morph between two people found on the left and right sides of the middle row. Deformation of the first image towards the second (top) and the second image towards the first (bottom). Blending these images results in the morph (middle).
Tables

4.1  Deformation times for the various methods.  . . . . . . . . . . . . . .  71
Chapter 1

Introduction

The field of geometric modeling now provides many useful tools that allow artists to construct complicated shapes [7, 36, 1]. Previously such applications were restricted to high-end, special purpose workstations such as those built by Silicon Graphics. However, in recent years, commodity level processors have made substantial performance gains and memory/storage has become relatively inexpensive. Moreover, special purpose graphics cards are extremely fast and capable of rendering very complex models in real-time on the average user's computer. For instance, NVidia's latest 7900 series graphics card can process 1.4 billion vertices per second. These developments have helped move high-end graphics to general purpose desktop machines. Furthermore, this movement has helped fuel an explosion of very high-resolution models, previously only the domain of off-line processes, in everyday graphics.

Even these advances have not been enough to keep up with the truly massive meshes generated by the Digital Michelangelo project at Stanford [31]. This project is attempting to create a digital repository of Michelangelo's works of art to sub-millimeter accuracy. The size of the models produced by this project are staggering; some are on the order of hundreds of millions of polygons. Figure 1.1 shows a picture of the head of David scanned to a resolution of one millimeter. This model contains approximately four million polygons and is only a fraction of the total model and half the desired resolution.

Though these shapes may be very detailed, static models are not very interesting. Instead, we would like to take these high-resolution models and animate them. In other words, we would like to change the physical characteristics of the model to pose
the model into different positions. For example, Figure 1.2 shows an example of a model of an armadillo man that has been deformed into various positions.

One of the challenges in creating such deformations is that we have very large models in Computer Graphics. These models are not relegated to some offline computing facility, because personal computers have advanced to the point that such high-resolution shapes are found on commodity desktop machines in everyday applications. Since, these models are so large, we need efficient techniques for deforming these shapes, which could be on the order of millions of polygons. For shapes this large, we cannot use offline processes designed to work on models of the order of ten thousand polygons and apply them to models on the order of ten million polygons. Therefore, computational efficiency is a concern.

Furthermore, we need simple techniques for specifying these deformations. Large models have far too many vertices for users to manipulate by hand. Instead, we need some small number of handles that users can control the shape of the deformation with. To make this concept more formal, we define a deformation handle to be a low-resolution shape that the user manipulates to control the shape of a high-resolution surface.
In particular, we take a space-warping approach to deformation where we construct a deformation function $f$ that maps points from undeformed space to deformed space. Given a set of deformation handles $p_i$ in rest position, the modified deformation handles $q_i$ and a point $v$ in space, we find the deformed position of the point by applying $f$ to $v$.

For $f$ to be useful for deformation, the function must satisfy three properties:

- **Interpolation:** The handles $p_i$ should map directly to $q_i$ under the deformation ($f(p_i) = q_i$).

- **Smoothness:** The deformations should produce smooth deformations ($\frac{\partial f(v)}{\partial v}$ is continuous).

- **Identity:** Applying the deformation function $f$ to the undeformed handles $p_i$ yields the identity transformation (i.e; $q_i = p_i \Rightarrow f(v) = v$).

These properties are very similar to those used in scattered data interpolation. The first two properties simply state that the function $f$ interpolates the scattered data values and is smooth. The last property is sometimes referred to as linear precision.
in approximation theory. This property states that if data is sampled from a linear function, then the interpolant reproduces that linear function. Given these similarities, it comes as no surprise that many deformation methods borrow techniques from scattered data interpolation.

Figure 1.3 shows an example using handles to specify a deformation. Given a shape like the armadillo man, we embed him in a low-resolution polygon mesh (shown here in wireframe, left) that approximates his shape. This low-resolution mesh is the deformation handle. Now, given a modified deformation handle such as in Figure 1.3 (right), we apply a deformation function to each vertex of the armadillo man, which results in the deformed surface on the right.

Since there are such large models present in Graphics today, we need efficient methods for computing the deformations specified by the function $f$. Furthermore, we need the ability to specify handles that are intuitive for the user to manipulate in order to control the deformation. In this dissertation, we will explore three different constructions of deformation functions using collections of tetrahedra, closed polygon meshes and points as deformation handles. Each of these methods constructs defor-
mations satisfying all three properties listed in this section and can be computed in real-time even for relatively large meshes.

1.1 Previous Work

Deformation is not a new field and there has been much previous work on this topic. Previous approaches have varied in terms of how the user specifies the deformation and the computational complexity required to compute the deformation.

Skeletal deformation is an intuitive method for specifying deformations of humanoid or animal-like objects, which was developed mostly for the game industry because of its computational efficiency. Today skeletal animation systems can be found in games, professional modeling systems such as Maya and the movie industry. Conceptually, the user specifies a skeleton structure inside of the object, which they will use to control the deformation. In reality, there are no bones at all but, instead, a hierarchical collection of coordinate frames. Each point on the surface of the model is represented in the coordinate frame of a single bone or a set of bones. In the case of multiple bones, a set of weights are associated with each coordinate frame and the vertex is then a weighted combination of its position in each frame. To create a deformation, the user moves the coordinate frames and the deformed vertex is found using the weighted combination of its position in each individual coordinate frame. This deformation method has many advantages: skeletal deformation creates an intuitive method for specifying deformations (very much like our own bones create deformations of our bodies) and these deformations can be computed very quickly. However, there are many problems with skeletal deformations, particularly at the joints of the model. One of the biggest disadvantages of this method is that the deformations are not smooth.

Free-form deformations [44, 22] are another popular method for specifying deformations, which actually create smooth deformations. These methods embed the object to be deformed in a uniform grid of cubes, which will be used as a deformation
handle. The space inside of this grid is then parameterized by trivariate, tensor-product splines (typically B-splines) to create a smooth parametrization of space. Each point in this volume may then be represented as a weighted combination of the vertices of the grid so that moving these control vertices induces a smooth deformation on the interior of the grid and, hence, the model within. While these methods produce fast, smooth deformations, they are not necessarily intuitive to use because the grids do not conform to the shape of the object.

Laplacian surface editing techniques \cite{48, 59, 33} include a family of methods that take a surface as input and try to represent every vertex on the surface as a linear combination of the vertices of its one-ring (i.e; the vertices edge-adjacent to a particular vertex). To deform the surface, the user drags individual vertices to desired positions. These vertices form constraints and the system then solves for the deformation to minimize the difference in the $L_2$ norm between the rest of the vertex locations and the vertex positions specified by the linear combinations computed earlier. The resulting minimum of this quadratic error functional can be found by solving a linear system of equations. The resulting deformations can look quite good though many such methods do not produce rotationally invariant coordinates \cite{48, 59}. However, the size of the system of equations can be as large as the number of vertices in the model. For large models on the order of hundreds of thousands to millions of polygons, this method is not computationally feasible.

Recently Radial Basis Functions (RBFs) \cite{6} have also been used quite successfully in the context of skeletal deformation \cite{32}. These RBF techniques are based on scattered data interpolation methods and can create deformations satisfying a set of constraints while minimizing the amount that the deformation must bend to satisfy such constraints. In terms of computational efficiency, these methods also involve solving a system of equations similar to the Laplacian methods and involve transcendental functions such as log.

Finally, there are methods that use finite element techniques for creating de-
formations [25, 24]. These techniques create very realistic looking deformations and, unlike the previous techniques, are based off of real physical equations for how objects deform. While these methods create realistic deformations, they are also the most expensive to compute. These methods discretize space into small cells (typically called elements) and numerically solve a partial differential equation that governs how the object should deform. These techniques are very expensive and are typically designed to be computed offline even for relatively small shapes of a few thousand polygons. Applying these methods to extremely large shapes is currently not computationally feasible on commodity hardware.

Figure 1.4 : Original image of a wooden gingerbread man (left) with a triangle as the deformation handle having vertices $p_i$. In the middle, we represent each point $v$ of the image as a combination of the $p_i$ using the areas $A_i$ of the wedges. Deforming the vertices $p_i$ to $q_i$ induces a deformation on the image (right).

1.2 A Simple 2D Deformation

To illustrate constructing deformations using handles, we first consider building a very simple deformation for two-dimensional shapes. Typically in graphics we construct our algorithms in 3D and operate on 3D shapes. However, it is usually easier to explain/visualize these algorithms in 2D. Therefore, we will first develop a deformation scheme for images (i.e; a two-dimensional shape).
To specify the deformation, we will use one of the simplest deformation handles available in 2D: a triangle. In fact, a triangle is so simple that it’s called a simplex in 2D. Therefore, we begin by taking a image and surrounding the shape with a triangle as shown in Figure 1.4 (left). Now our goal is to represent every point \( v \) in the image as a function of the deformation handle. For a triangle, this operation is simple. We represent \( v \) as a weighted combination of the vertices \( p_i \) of the triangle by partitioning the triangle into a set of wedges using \( v \) as shown in Figure 1.4 (middle). Now \( v \) is given as

\[
v = \sum_{i=1}^{3} \alpha_i p_i
\]

where \( \alpha_i = \frac{A_i}{A_1 + A_2 + A_3} \).

As the user moves the vertices of the triangle to their modified location \( q_i \), we create the deformed image by applying the weights \( \alpha_i \) not to the vertices \( p_i \) but to the deformed vertices \( q_i \). We apply this deformation to every point \( v \) in the image and the result is the deformed image in Figure 1.4 (right).

While this deformation is easy to compute, it is far from ideal. The deformations with this method are not very expressive: we can create only affine transformations of the original image. Furthermore, the deformation handle is not very intuitive, since most shapes are not approximated very well by a single triangle. Nevertheless, this technique forms the basis for many of the deformations we create in subsequent chapters.
Chapter 2

Tetrahedral Subdivision

In the previous chapter we considered building a simple deformation method using a single triangle. Unfortunately, the deformations produced by that method were not very expressive; we could only produce affine transformations of the original image. However, we can easily extend this deformation to create more interesting deformations.

Figure 2.1: Gingerbread man with a polygon approximating its shape (left). We triangulate the interior of the polygon and apply the simple deformation method from Chapter 1 to each triangle individually (middle two images). The resulting deformation (right) is not smooth across triangle boundaries.

Continuing with our previous example, we can create a deformation of our image by approximating the shape we wish to deform using a closed-polygon (see Figure 2.1, left). Next, we triangulate the interior of the polygon as shown in the second image of Figure 2.1. Given a deformed version of this triangle mesh, for each triangle in the mesh and its deformed position, we apply the simple deformation method from the previous chapter to that triangle and the portion of the image inside the triangle.
The resulting deformation is far more expressive and does not consist of only a single, affine transformation. Unfortunately, this deformation is still too simple. The deformations produced are not smooth across the triangle boundaries, which can be clearly seen in the silhouette of the gingerbread man as well as in the grains of wood on the interior. Introducing these smoothness discontinuities is certainly undesirable. To combat this problem, we need some way of smoothing out the deformations produced by this method.

Fortunately, subdivision provides a method for creating smooth deformations of images from these triangulated polygons. Traditionally subdivision schemes have been used for 3D surface representations and are attractive because they create smooth surfaces from arbitrary topology polygonal meshes [7, 34]. Nonetheless, subdivision schemes may be used to create smooth deformations of images.

![Figure 2.2](image.png)

(a) (b) (c) (d)

Figure 2.2: From left to right: initial deformation, linear subdivision, smoothing and the final deformation after one round of subdivision.

To illustrate the process, we describe a simple subdivision method for creating smooth deformations of the gingerbread man in Figure 2.2. Subdivision schemes are typically defined as recursive rules that refine polygons. For our example, we will use Loop subdivision [34], which can be described by two separate passes: refinement and smoothing [55]. In the refinement phase, we insert new triangles in the mesh by splitting each triangle into four new triangles (see Figure 2.2 (b)). We then take
the output of the refinement phase and apply the smoothing mask described by Warren and Schaefer [55], which results in the triangle mesh in Figure 2.2 (c). The deformation after one round of subdivision is then shown in Figure 2.2 (d).

Figure 2.3: A sequence of deformations produced by subdivision with 0, 1 and 2 rounds of subdivision.

Notice that the resulting deformation is smoother than before but is still noticeably not smooth. However, subdivision is a recursive process so we can repeat the refinement and smoothing rules on this new, refined triangle mesh. This iteration generates a sequence of deformations that, in the limit, converge to a smooth deformation of the image (see Figure 2.3).

While most work on subdivision has focused on surface meshes (appropriate for creating deformations of images), we consider the problem of subdividing volumetric meshes. Typically, volumetric subdivision schemes have been proposed as a means to define deformations. However, the existence of simple schemes for tensor product volumetric meshes (i.e.; uniform grids) such as free-form deformations [44] reduces this question to why subdivision schemes for unstructured meshes are important.

Figure 2.4 shows an application of subdivision to the problem of image deformation that illustrates the superiority of unstructured methods. In this case, the image being deformed is a cross-section of a mouse brain, where the pixel intensities represent the cell density in different anatomical (colored) regions of the brain. On the left, the image has been covered by a uniform base mesh. Subdividing this quad mesh using
bi-cubic subdivision yields a C^2 mesh that defines a smooth parameterization of the image. Perturbing the vertices of the base mesh in the region of the cerebellum (dark folds) induces a corresponding deformation of the underlying image. On the right, the image has been covered by an unstructured quadrilateral mesh. A subset of the edges in this mesh have been creased to define a network of crease curves that partition the base mesh into anatomical regions. This quadrilateral base mesh is subdivided using Catmull-Clark subdivision [7] to define a smooth parameterization of the underlying image. Perturbing the vertices of the quadrilateral base mesh induces deformations that are restricted to a single anatomical region. Thus, the use of an unstructured mesh allows the construction of deformations with much finer control than those built using tensor product methods.

2.1 Previous work

While previous work on subdivision of unstructured volumetric meshes has been limited, there are a few papers that have addressed this problem. MacCracken and Joy [37] developed one of the first volumetric subdivision schemes. This scheme was developed primarily to define deformations based on unstructured hexahedral meshes. Unfortunately, the subdivision rules proposed in the paper were developed in a some-
what ad-hoc manner making any type of proof of smoothness for the scheme very
difficult. Later, Bajaj et al [3] developed different subdivision rules for hexahedral
meshes, which generate deformations that are provably smooth everywhere except at
vertices of the hexahedral base mesh.

Both of these schemes use hexahedra (topological cubes) as their volumetric el-
ements. Unfortunately, building unstructured meshes of hexahedra that conform to
specific boundary shapes can be difficult. Traditionally, mesh generation methods
generate unstructured meshes of tetrahedra instead. The most relevant piece of pre-
vious work is a subdivision scheme for unstructured tetrahedral meshes proposed
by Chang et al [8]. In that paper, the authors build subdivision rules for unstruc-
tured tetrahedral meshes by generalizing the subdivision rules for a particular class
of trivariate box-splines.

While this approach was successfully used by Loop [34] to generalize the subdivi-
sion rules for the $C^2$ three-direction quartic box splines to unstructured triangular
meshes, using trivariate box splines to generate subdivision rules for unstructured
tetrahedral grids is much more difficult. The drawback of the subdivision rules pro-
posed in Chang et al is that these rules encode a preferred direction in each tetrahe-
dron of the base mesh. (Section 2.2.1 will elaborate on this point.) This directional
preference makes implementing the scheme of Chang et al tricky, and proving any
results concerning the smoothness of the scheme is extremely difficult.

Contributions

In contrast to Chang et al, we develop volumetric subdivision rules for unstructured
tetrahedral meshes that avoid the assumption of any preferred direction in the base
mesh. This construction also generalizes the bivariate case and leads to a trivariate
scheme with two important properties:

- The scheme is simple to implement in terms of linear subdivision and smoothing.
- The deformations induced by the scheme are provably $C^2$ everywhere except
  along edges of the base mesh. Along edges shared by four or more tetrahedra,
we present strong evidence that the resulting deformations are $C^1$.

We first present the rules of our subdivision scheme and provide pseudo-code for implementing the method. Next, we discuss applications of this method to 3D surface deformation. We conclude this chapter with a mathematical analysis of the smoothness of the scheme using a combination of regularity analysis (Reif [42]) and spectral analysis (Levin/Levin [29]).

2.2 A tetrahedral subdivision scheme

Our proposed scheme is a combination of linear subdivision followed by a smoothing pass. This structure is similar to that of the several schemes proposed for subdividing surface meshes [3, 49, 60]. As in the bivariate case, implementing our scheme is quite simple and does not require neighbor finding or mesh traversal algorithms. To illustrate the ease of implementation, we provide pseudo-code for the smoothing pass at the end of this section.

2.2.1 Linear subdivision

To perform linear subdivision on a mesh of tetrahedra, we define a split on a single tetrahedron, which is then applied to all tetrahedra in the mesh. Given a tetrahedron, we insert new vertices at the midpoints of each edge and connect the vertices together to form four new tetrahedra at the corners of the original tetrahedron. Chopping these four children off the corners of the parent tetrahedron leaves an octahedron (see Figure 2.6 top). (Note that performing this corner chopping on a triangle yields a triangle making linear subdivision for triangular meshes much easier.)

At this point, we are faced with a dilemma. We can either split the octahedron into four tetrahedra by choosing a diagonal for the octahedron (see Figure 2.5) or leave the octahedron alone and develop an analog of linear subdivision for octahedra. At first glance, splitting the octahedron along a diagonal might seem like the simpler
Figure 2.5: Splitting a tetrahedron generates an octahedron in the middle. Splitting the octahedron into tetrahedra requires choosing a diagonal.

choice because the split produces only tetrahedra. In reality, this choice leads to substantial complications during any attempt to analyze the smoothness of the associated subdivision scheme. This choice of diagonal causes the resulting tetrahedral mesh to contain a preferred direction associated with the choice of diagonal. To generate a provably smooth subdivision scheme, this diagonal must be inherited during linear subdivision. More crucially, each tetrahedron in the base mesh must be assigned such a diagonal. Any type of smoothness analysis that considers the interface between two tetrahedra in the base mesh must enumerate all possible choices for this diagonal.

Given that our goal is to create a scheme that contains no preferred direction and is simple enough to prove smoothness, we do not choose a diagonal for the middle octahedron and split a tetrahedron into four new tetrahedra and an octahedron (see Figure 2.6). Since we have introduced an octahedron into the volumetric mesh, our subdivision scheme is not simply a tetrahedral subdivision scheme, but a tetrahedral/octahedral subdivision scheme. Therefore, we must define a refinement rule for octahedra as well.

To refine an octahedron, we insert vertices at the midpoints of each edge on the octahedron and at the centroid of the octahedron, formed by averaging all of the vertices of the octahedron. Next, we connect the vertices to form six new octahedra (corresponding to the six vertices of the original octahedron) and eight new tetrahedra
Figure 2.6: Linear subdivision splits a tetrahedron into four tetrahedra and an octahedron (top). An octahedron is split into six octahedra and eight tetrahedra (bottom).

(corresponding to the eight faces of the original octahedron). The entire refinement process is illustrated in Figure 2.6.

While Chang et al's tetrahedral scheme is similar to ours in that it does not topologically split the octahedron, their scheme generates subdivision rules that encode a preferred diagonal along the octahedron. This preferred diagonal is a natural result of their use of trivariate box splines to generate their subdivision rules. Due to the existence of a preferred diagonal, Chang et al's scheme is guaranteed to be smooth only on the interior of each tetrahedron in the base mesh. In particular, Chang et al make no attempt to analyze the smoothness of their scheme across the face shared by two tetrahedra in the base mesh.

The need for such face/face analysis is somewhat surprising and was not even recognized by Chang et al. This failure is understandable, since a subdivided triangular
mesh is uniform along the interior of edges in the base mesh. Similarly, a subdivided
hexahedral mesh is uniform along the interior of quad faces of the base mesh. Un-
fortunately, a subdivided, unstructured tetrahedral mesh is not uniform across the
interior of triangular faces of the base mesh. Thus, substantial care must be used in
designing the subdivision rules if one hopes to construct a scheme that is provably
smooth across these faces. In section 2.4, we use the joint spectral radius techniques
of Levin/Levin to prove that our scheme is $C^2$ across the interior of these faces.

2.2.2 Smoothing

After linear subdivision, we perform a smoothing pass over the tetrahedral/octahedral
mesh to reposition each vertex. For each vertex in the mesh after linear subdivision,
we find each volumetric cell (tetrahedron or octahedron) containing that vertex. Then
we compute the weighted centroids shown in Figure 2.7 for each cell. For tetrahedra,
this centroid computes $-\frac{1}{16}$ of the vertex being repositioned and $\frac{17}{48}$ of the edge adja-
cent vertices. To generate the centroid for octahedra we take $\frac{3}{8}$ of the vertex being
repositioned, $\frac{1}{12}$ of the edge adjacent vertices and $\frac{7}{24}$ of the cell-adjacent vertex. We
then average all of these centroids together to obtain the new location of the reposition-
ted vertex. Despite the fact that there is a negative weight in the centroid mask
for tetrahedra, the subdivision rules produced by combining linear subdivision and
smoothing use only convex combinations.

To show this convex property, there are three cases to consider: the ordinary
case, an extraordinary vertex and an extraordinary edge (extraordinary faces follow
the same argument as the ordinary case). In the ordinary setting, our averaging
masks reproduce the rules generated by convolution (see Section 2.4), which have
only positive weights. For the extraordinary vertex, the only point with a negative
weight is the central vertex. However, the edge adjacent vertices are at the midpoints
of those edges and applying the tetrahedral smooth mask to those vertices results in
a positive weight for the central vertex ($\frac{-1}{16} + 3\frac{17}{48}$. Extraordinary edges follow a
Figure 2.7: Centroid masks for tetrahedra/octahedra. The highlighted vertex is the vertex being repositioned by smoothing.

similar argument as vertices.

Since this smoothing pass is described only in terms of centroid masks, it yields a very simple implementation. Given an unstructured tetrahedra/octahedra mesh, we first apply linear subdivision. This operation can be implemented on a cell by cell basis. For smoothing, we initialize the vertices of a mesh with the same topology as the input to be identically 0. Then, for each cell in the mesh, we compute the centroid mask of Figure 2.7 in all possible orientations and add that quantity to the vertex to be repositioned for that orientation. Finally, we divide each vertex by its valence (the number of cells containing that vertex). Figure 2.8 illustrates pseudo-code for the smoothing pass. This description requires no neighbor finding in the mesh or external data structures to traverse the mesh and is quite easy to implement.

We can also incorporate sharp features where we alter the continuity of the volume to be $C^0$ using the method described by Hoppe et al [18]. Figure 2.11 shows a cylindrical volume with a crease surface defined by Loop subdivision and crease edges, which form B-splines, around the top and bottom of the cylinder.
2.3 Application to deformations

As alluded to in the introduction, volumetric subdivision schemes find their main use in generating smooth volumetric deformations. Given a volume $R$, a volumetric deformation $f$ maps points $v$ in $R$ to new points $f(v)$ in $f(R)$. The deformation $f$ is $C^k$ continuous if each coordinate function comprising $f$ can be expressed locally as the graph of a function with $k$ continuous derivatives.

To construct volumetric deformations, we use the basic technique described in MacCracken and Joy [37]. Given a base mesh $p^0$, we define $R$ to be the volume spanned by the limit mesh $p^\infty$. If the mesh $p^\infty$ forms a one-to-one covering of $R$, we
can express each point \( v \) in \( R \) as a unique point on the limit mesh \( p^\infty \) of the form \( \sum_i \alpha_i p_i^0 \) where the \( p_i^0 \) are vertices of the base mesh \( p^0 \) (see Figure 2.9). Perturbing the vertices of the base mesh to form a new mesh \( q^0 \) defines an associated deformation \( f \) of the form

\[
f(v) = \sum_i \alpha_i q_i^0.
\]  

In section 2.4 we show that the deformations induced by our tetrahedral subdivision scheme are provably \( C^2 \) everywhere except along edges of the base mesh. Along edges shared by four or more tetrahedra, we hypothesize the scheme is \( C^1 \) and provide strong evidence to back this claim.

Figure 2.10 shows an application of our method to the problem of deforming a dinosaur skeleton. First, we embed the skeleton in a tetrahedral base mesh \( p^0 \). Next, we perform several rounds of subdivision on the base mesh to form a refined mesh \( p^k \). Each new vertex inserted in the tetrahedral/octahedral mesh \( p^k \) is represented as a convex combination of the vertices of the base mesh \( p^0 \). Then, for each vertex \( v \) of the skeleton, we find the tetrahedra or octahedra in the refined mesh \( p^k \) that contains that vertex and compute the barycentric coordinates of that vertex with respect to its enclosing cell. Since all vertices of the cell are convex combinations of vertices of \( p^0 \), the skeleton vertex \( v \) can then be represented as a convex combination of the vertices of \( p^0 \). Perturbing the vertices of the base mesh forms a new base mesh \( q^0 \).
that defines a deformation $f(v)$ of the vertices of the skeleton as given in equation 2.1.

Figure 2.10 illustrates several different deformations of the dinosaur model. Since we use an unstructured grid of tetrahedra, we can encase the surfaces to be deformed in far fewer volumetric elements than would be required by free-form deformations using structured grids. Therefore, this sparse embedding yields deformations that require relatively few tetrahedral vertices and can be performed in real-time.

Figure 2.11 depicts another example in which a 3D test pattern is deformed using our scheme. The base mesh $p^0$ is a tetrahedral mesh approximating the shape of a cylinder. To generate the sharp circular edges along the top and bottom of the cylinder, we have creased the appropriate edges of the cylinder. The left part of the figure shows the tetrahedral mesh in wireframe. The middle and right portions of the figure show the 3D test pattern before and after perturbation of the base mesh.

### 2.4 Smoothness analysis

Given a tetrahedral base mesh $p^0$, we consider the smoothness of a deformation $f(v)$ induced by a perturbation of the base mesh. In particular, our smoothness analysis considers four cases:
Figure 2.11: Initial set of tetrahedra, subdivided surface, deformed surface and cut interior. Parameter lines are smooth after deformation.

- $v$ lies in the interior of a tetrahedron of the base mesh,
- $v$ lies on the interior of a face shared by two tetrahedra of the base mesh,
- $v$ lies on the interior of an edges shared by several tetrahedra of the base mesh,
- $v$ lies at a vertex of the base mesh.

2.4.1 Interior of a base tetrahedron

To begin our analysis, we first consider the structure of the uniform mesh generated by linearly subdividing a single tetrahedron repeatedly. If this base tetrahedron has vertices of the form $(1,0,0,0), (0,1,0,0), (0,0,1,0) , (0,0,0,1)$, $k$ rounds of linear subdivision generate a uniform 3D mesh whose vertices have barycentric coordinates of the form $\frac{1}{2^k}(i_0, i_1, i_2, i_3)$ where the $i_j$ are non-negative integers that sum to $2^k$. (The embedding of the base tetrahedron in the hyperplane $x_0 + x_1 + x_2 + x_3 = 1$ allows the coordinates to be treated symmetrically and avoids the use of any preferred direction in our construction.)

Relaxing the restriction that the coordinates of the mesh vertices are non-negative yields a sequence of infinite uniform meshes $M^k$ associated with the subdivision process. Unfortunately, the scaling relation between consecutive meshes $M^k$ and $M^{k+1}$ is subtle due to the use of barycentric coordinates. However, if we translate the base
mesh $M^0$ to interpolate the origin, the resulting mesh $M_c$ lies on the hyperplane
$x_0 + x_1 + x_2 + x_3 = 0$ and has vertices of the form $(i_0, i_1, i_2, i_3)$, where the $i_j$ are
integers whose sum is 0. After translation, linearly subdividing the mesh $M_c$ yields a
dilated mesh of the form $\frac{1}{2}M_c$.

For this scale-invariant mesh, we now consider the hat function $n(v)$ centered at
the origin that is generated by linear subdivision with no smoothing. Based on the
splitting rules for linear subdivision, $n(v)$ satisfies a scaling relation of the form

$$n(v) = n(2v) + \frac{1}{2} \sum_{j=1}^{12} n(2v - \delta_j) + \frac{1}{6} \sum_{j=1}^{6} n(2v - \gamma_j)$$  \hspace{1cm} (2.2)

where the vectors $\delta_j$ and $\gamma_j$ define integer offsets in the uniform mesh $M$. These
offsets correspond to vertices of $M_c$ that lie in the one-ring of the origin (see the left
portion of figure 2.12) and correspond to

$$\delta = \text{Permutations} \ (1, -1, 0, 0)$$
$$\gamma = \text{Permutations} \ (1, 1, -1, -1)$$

where Permutations yields a set without duplication ($\delta$ contains 12 elements while $\gamma$
has 6). The right portion of Figure 2.12 shows the 3D subdivision mask formed by
the coefficients of equation 2.2.

We next consider the effect of the smoothing pass on the mesh formed by linear
subdivision. If we apply the weighted-centroid averaging rules for this pass to the
one-ring of the origin, the smoothing mask that results is also supported over the
one-ring of origin and is exactly $\frac{1}{8}$ of the subdivision mask for linear subdivision.
(The agreement is no coincidence as we chose the weights used in the smoothing pass
to ensure this agreement.)

Now, the subdivision mask for the composite scheme formed by linear subdivision
and smoothing is simply $\frac{1}{8}$ of the discrete convolution of the mask of figure 2.12 with
itself. As shown in [52], the discrete convolution of two subdivision masks yields a
new subdivision mask whose associated basis function is the continuous convolution
of the basis functions associated with each original mask. In our case, the convolution
of the hat function $n(v)$ with itself is the basis function for our composite scheme. Since the continuous convolution of two $C^0$ functions ($n(v)$ with itself) is always a $C^2$ function [57], our scheme generates $C^2$ deformations on uniform meshes.

2.4.2 Faces of the base mesh

While the smoothness of our subdivision scheme on the interior of a base tetrahedron follows by appealing to convolution, we must verify smoothness of our scheme in the other cases using spectral methods suitable for analysis of non-uniform schemes because the mesh formed by applying linear subdivision to an unstructured tetrahedral mesh is uniform only on the interior of base tetrahedra and is no longer uniform across the faces of the base mesh. In particular, the uniform mesh $M_c$ generated on the interior of a base tetrahedron has the property that every face in the grid is shared by a tet/oct pair. On the other hand, the infinite mesh $M_f$ generated by applying linear subdivision to two face-adjacent tetrahedra consists of two copies of $M_c$ joined along a triangular interface formed by tet/tet pairs and oct/oct pairs. (see figure 2.13.)
Figure 2.13: Two face-adjacent tetrahedra subdivided and opened at the shared face. Subdivision generates tet/tet and oct/oct pairs along that face.

To prove that our scheme is $C^2$ on the mesh $M_f$, we use the joint spectral radius test originally developed by Levin/Levin [29] to analyze the smoothness of triangle/quad subdivision along the interface between triangles and quads. Note that the mesh structure in their triangle/quad analysis is similar to the mesh structure of $M_f$: two uniform meshes separated by planar interface.

If $S$ is the subdivision matrix associated with $M_f$, we first compute the eigenvalues $\lambda_j$ and eigenvectors $z_j$ of the form $Sz_j = \lambda_j z_j$. The Levin/Levin $C^2$ smoothness test involves checking three conditions:

- First, check whether the eigenvalues $\lambda_j$ (ordered in descending value) have the form

  $$1, \frac{1}{2}, \frac{1}{2}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \ldots$$

  (2.3)

  Note that the subdominant eigenvectors $(z_1, z_2, z_3)$ reproduce the grid $M_f$. As a result, the eigenfunctions associated with these eigenvectors define a characteristic map that produces a one-to-one covering of space.

- Next, we check whether the eigenfunctions associated with eigenvectors $z_4, \ldots, z_9$ are quadratic functions when plotted over the characteristic map.
Finally, the joint spectral radius of the subdivision scheme must be less than $\frac{1}{4}$.

The first test is simple to perform and involves only extraction of the eigenvalues and eigenvectors of the subdivision matrix. The second condition can be checked using quasi-interpolants as in Levin/Levin [29]. However, the joint spectral radius test requires more work.

Figure 2.13 (left) shows a portion of $M_f$ with the interface between a tet/tet and oct/oct pair highlighted. This pair forms a patch on the face between two tetrahedra. To perform the joint spectral radius test, we must construct 4 subdivision matrices $S_i$ that map the support of the patch on Figure 2.13 (left) to the support of each of the four sub-quads formed after one round of subdivision (Figure 2.13 right). Each sub-quad is a scaled and translated version of the original quad and yields a square subdivision matrix $S_i$.

After building the $S_i$, we then construct a diagonalizing matrix $W$ using $S_1$ such that

$$W^{-1}S_1W = \begin{pmatrix} \Lambda & C_1 \\ 0 & Y_1 \end{pmatrix}$$

$$W^{-1}S_iW = \begin{pmatrix} \theta_i & C_i \\ 0 & Y_i \end{pmatrix} \quad \text{for } i \neq 1$$

where $\Lambda$ is a diagonal matrix whose entries are the specified eigenvalues in equation 2.3 and $\theta_i$ is an upper triangular matrix that shares the same diagonal entries as $\Lambda$. $W$ can be constructed using the eigenvectors in $S_1$ corresponding to the eigenvalues in $\Lambda$ and the null space of those vectors.

Finally, to perform the joint spectral radius test, we compute

$$\rho^{[k]}(Y_1, \ldots, Y_4) = (\max \{ \| Y_{\epsilon_k} Y_{\epsilon_{k-1}} \cdots Y_{\epsilon_1} \|_\infty \})^\frac{1}{k}$$

where $\epsilon_i \in \{1, \ldots, 4\}$. If $\rho^{[k]} < \frac{1}{4}$ for some $k$, then the scheme is $C^2$ over that extraordinary complex. For our four subdivision matrices we computed $\rho^{[9]} = 0.238$ and conclude that our scheme is $C^2$ along the face shared by two tetrahedra.
Figure 2.14: Characteristic map for edges of valence 3 through 10.

2.4.3 Edges of the base mesh

To analyze the smoothness of our scheme along edges of the base mesh is more difficult than the face case, since the structure of the infinite mesh $M_\infty$ formed by subdividing $n$ tetrahedra sharing a common edge depends on $n$. In practice, we know of no analysis technique capable of establishing the smoothness of our scheme along this edge. However, we hypothesize that a combination of the analysis methods of Levin/Levin [29] and Reif [42] can be used to analyze the smoothness of volumetric scheme in configurations of this type.

Given the subdivision matrix $S$ for the mesh $M_\infty$, we hypothesize that the scheme is $C^1$ if:

- Its eigenvalues are of the form $1 > \lambda_1 \geq \lambda_2 \geq \lambda_3 > \ldots$

- The characteristic map formed by the eigenvectors $z_1, z_2, z_3$ is regular and injective.

- Finally, the joint-spectral radius of the scheme $(\rho^{[k]}(Y_1, Y_2))$ must be less than
\[ \lambda_3. \]

The core of our hypothesis is that the second condition allows the smoothness analysis to be reduced to the functional case used in the joint spectral radius test.

We have checked these three criteria for our scheme. Since the mesh \( M_c \) is parameterized by the number of tetrahedra sharing the edge of the base mesh, the subdivision matrices \( S \) will contain a block circulant structure that makes extraction of the eigenvectors \( z_1, z_2, z_3 \) as a symbolic function of \( n \) possible. However, it is unclear how the block circulant structure interacts with the joint spectral radius test. Therefore, we have numerically computed the characteristic map for edge valences 3 through 10 and visually inspected their shape (see figure 2.14).

To apply the joint spectral radius test, we construct subdivision matrices \( S_1, S_2 \) corresponding to one of two shifts along the edge. For our scheme, these matrices have eigenvalues of the form \( 1, \lambda_1, \lambda_1, \frac{1}{2}, ... \) and satisfy the joint spectral radius test for valences \( 10 \geq n > 3 \). For \( n = 3 \), we could not find a \( k \) that satisfied the joint spectral radius condition. While this failure does not mean the scheme is \( C^0 \), we visually inspected the smoothness of the volumes using the 3D test patterns shown in Figure 2.11 and the deformations produced are not smooth. For other valences \( 10 \geq n > 3 \), the 3D test patterns undergoes visually smooth deformation.

### 2.4.4 Vertices of the base mesh

As for the edge case, we know of no analysis method for proving that our scheme is smooth at a vertex of the base mesh. However, we again hypothesize that the conditions for the edge case suffice to establish smoothness at a vertex. In the vertex case, the third test is redundant, since there is only a single matrix \( S_1 = S \) used in computing the joint spectral radius.

Similar to the edge case, we consider only arbitrary packing of tetrahedra around a base vertex. Unfortunately, the simple parameterization by valence used in analyzing surface subdivision schemes is unavailable for volume schemes and no block circulant
structures can be exploited to analyze the smoothness of the scheme. Therefore, we enumerated through all configurations of tetrahedra around a vertex for valences 4 through 10 and tested random configurations of tetrahedra for higher valences. Each configuration of tetrahedra passed the eigenvalue and characteristic map test. Figure 2.15 shows several characteristic maps produced for different configurations and number of tetrahedra around a vertex.

2.5 Conclusions

We have presented a simple subdivision scheme for unstructured tetrahedral meshes. This scheme consists of linear subdivision followed a smoothing pass. Implementing the pass of the scheme requires only a standard topological mesh representation without the need for any auxiliary adjacency information. The key to the simplicity of the scheme is the symmetric treatment of the octahedron generated by chopping off
the corners of a tetrahedron. As we show in section 2.4, this choice makes smoothness analysis possible.
Chapter 3

Mean Value Coordinates

Returning to our simple example, subdivision allows us to create a smooth deformation of images by triangulating a polygon that approximates the shape of the image. However, when the user creates the deformation, the triangulation on the interior is typically superfluous as the user only manipulates the vertices on the exterior of the polygon. Unfortunately the triangulation chosen in the interior of the polygon affects the deformation. Figure 3.1 shows a deformation of our test shape using two different triangulations. The deformations are similar to one another; however, this dependence on the triangulation creates a deformation that is not unique.

![Image of four figures showing different deformations](image)

Figure 3.1: Comparison of deformations produced via Subdivision using two different triangulations. On the left are the initial triangulations of the control polygon. On the right are the deformation produced with the same vertex locations, but different triangulations. The deformations are similar but not identical.

Therefore, for some applications, it may be desirable to remove the triangulation on the interior of the polygon and create a deformation using just the polygon itself (see figure 3.2). In other words, we would like to represent every point $v$ of the image
as a function of the vertices of a closed polygon. Notice that this problem is actually very similar to the simplest deformation method in Chapter 1. There we represented a point as a function of the vertices of a triangle (also another type of polygon). In fact, these methods are intimately related through a type of interpolant called barycentric coordinates.

![Image of barycentric coordinates](image)

Figure 3.2: Undeformed gingerbread man with control polygon (left) and the deformation created using mean value coordinates on the right.

Given a closed polygon with vertices $p_i$, barycentric coordinates allow us to represent a point $v$ as a weighted combination of the $p_i$ expressed in the form

$$v = \frac{\sum_j w_j p_j}{\sum_j w_j}$$

In this formula, note that each weight $w_j$ is normalized by the sum of the weights, $\sum_j w_j$ to form an associated coordinate $\frac{w_j}{\sum_j w_j}$. These normalized coordinates are called barycentric coordinates.

Barycentric coordinates have the property that they are parametric and, hence, can be used to extend function values $g_i$ associated with the vertices of the polygon to its interior. The value at a point $v$ on the interior is then given as

$$f(v) = \frac{\sum_j w_j g_j}{\sum_j w_j}.$$ (3.1)

Since $f(v)$ has the property that it interpolates the values at the $p_i$ (i.e., $f(p_i) = g_i$), we call this equation a barycentric interpolant.
Figure 3.3: Interpolating hue values at polygon vertices using Wachspress coordinates (a, b) versus mean value coordinates (c, d) on a convex and a concave polygon.

Mesh parameterization methods [20, 10, 26, 43, 15] and freeform deformation methods [45, 9, 38, 27] also make heavy use of interpolants of this type. Both applications require that a point v be represented as an affine combination of the vertices on an enclosing shape.

For convex polygons in 2D, a sequence of papers, [51], [35] and [41], have proposed and refined an interpolant that is linear on its boundaries and involves only convex combinations of data values at the vertices of the polygons. This Wachspress interpolant has a simple, local definition as a rational function and reproduces linear functions. [53, 56] also generalized this interpolant to convex shapes in higher dimensions. Unfortunately, Wachspress’s interpolant does not generalize to non-convex polygons. Applying the construction to a concave polygon yields an interpolant that has poles (divisions by zero) on the interior of the polygon. The top portion of Figure
3.3 shows Wachspress's interpolant applied to two closed polygons. Note the poles on the outside of the convex polygon on the left as well as along the extensions of the two top edges of the non-convex polygon on the right.

More recently, several papers, [12, 13, 14], [39] and [19], have focused on building interpolants for non-convex 2D polygons. In particular, Floater proposed a new type of interpolant based on the mean value theorem [14] that generates smooth coordinates for star-shaped polygons. Given a polygon with vertices $p_j$ and associated values $g_j$, Floater's interpolant defines a set of weight functions $w_j$ of the form

$$w_j = \frac{\tan \left( \frac{\alpha_j-1}{2} \right) + \tan \left( \frac{\alpha_j}{2} \right)}{|p_j - v|}.$$  \hspace{1cm} (3.2)

where $\alpha_j$ is the angle formed by the vector $p_j - v$ and $p_{j+1} - v$. Normalizing each weight function $w_j$ by the sum of all weight functions yields the mean value coordinates of $v$ with respect to $p_j$.

In his original paper, Floater primarily intended this interpolant to be used for mesh parameterization and only explored the behavior of the interpolant on points in the kernel of a star-shaped polygon. In this region, mean value coordinates are always non-negative and reproduce linear functions. Subsequently, Hormann [19] showed that for any simple polygon (or nested set of simple polygons), the interpolant $f(v)$ generated by mean value coordinates is well-defined everywhere in the plane. By maintaining a consistent orientation for the polygon and treating the $\alpha_j$ as signed angles, Hormann also shows that mean value coordinates reproduce linear functions everywhere. The bottom portion of Figure 3.3 shows mean value coordinates applied to two closed polygons. Note that the interpolant generated by these coordinates possesses no poles anywhere even on non-convex polygons.

**Contributions**

Hormann's observation suggests that Floater's mean value construction could be used to generate a similar interpolant for a wider class of shapes. In this chapter, we provide such a generalization for arbitrary closed surfaces and show that the resulting
interpolants are well-behaved and have linear precision. Applied to closed polygons, our construction reproduces 2D mean value coordinates. We then apply our method to closed triangular meshes and construct 3D mean value coordinates. (In independent contemporaneous work, [16] have proposed an extension of mean value coordinates from 2D polygons to 3D triangular meshes identical to section 3.2.2.) Next, we derive an efficient, stable method for evaluating the resulting mean value interpolant in terms of the positions and associated values of vertices of the mesh. Finally, we consider several practical applications of such coordinates, including a simple method for generating classes of deformations useful in character animation.

3.1 Mean value interpolation

Given a closed surface $P$ in $\mathbb{R}^3$ and an auxiliary function $g(x)$ defined over $P$, let $p(x)$ be a parameterization of $P$ (where $x$ is two-dimensional). Our problem is to construct a function $f(v)$ where $v \in \mathbb{R}^3$ that interpolates $g(x)$ on $P$, i.e.; $f(p(x)) = g(x)$ for all $x$. Our basic construction extends an idea of Floater developed during the construction of 2D mean value coordinates.

To construct $f(v)$, we project each point $p(x)$ of $P$ onto the unit sphere $S_v$ centered at $v$. Next, we weight the point's associated value $g(x)$ by $\frac{1}{|p(x)-v|}$ and integrate this weighted function over $S_v$. To ensure affine invariance of the resulting interpolant, we divide the result by the integral of the weight function $\frac{1}{|p(x)-v|}$ taken over $S_v$. Putting the pieces together, the mean value interpolant has the form

$$f(v) = \frac{\int_x w(x, v)g(x)dS_v}{\int_x w(x, v)dS_v}$$

(3.3)

where the weight function $w(x, v)$ is exactly $\frac{1}{|p(x)-v|}$. Observe that this formula is essentially an integral version of the discrete formula of Equation 3.1. Likewise, the continuous weight function $w(x, v)$ and the discrete weights $w_j$ of Equation 3.2 differ only in their numerators. As we shall see, the $\tan\left(\frac{a}{2}\right)$ terms in the numerators of the $w_j$ are the result of taking the integrals in Equation 3.3 with respect to $dS_v$. 
The resulting mean value interpolant satisfies three important properties.

**Interpolation:** As \( v \) approaches a point \( p(x) \) on \( P \), \( f(v) \) converges to \( g(x) \).

**Smoothness:** The function \( f(v) \) is well-defined and smooth for all \( v \) not on \( P \).

**Linear precision:** If \( g(x) = p(x) \) for all \( x \), the interpolant \( f(v) \) is identically \( v \) for all \( v \).

Interpolation follows from the fact that the weight function \( w(x, v) \) approaches infinity as \( p(x) \to v \). Smoothness follows because the projection of \( g(x) \) onto \( S_v \) is continuous in the position of \( v \) and taking the integral of this continuous process yields a smooth function. The proof of linear precision relies on the fact that the integral of the unit normal over a sphere is exactly zero (due to symmetry). Specifically,

\[
\int \frac{p(x) - v}{|p(x) - v|} dS_v = 0
\]

since \( \frac{p(x) - v}{|p(x) - v|} \) is the unit normal to \( S_v \) at parameter value \( x \). Rewriting this equation yields the theorem.

\[
v = \frac{\int x \frac{p(x)}{|p(x) - v|} dS_v}{\int x \frac{1}{|p(x) - v|} dS_v}
\]

Notice that if the projection of \( P \) onto \( S_v \) is one-to-one (i.e.; \( v \) is in the kernel of \( P \)), then the orientation of \( dS_v \) is non-negative, which guarantees that the resulting coordinate functions are positive. Therefore, if \( P \) is a convex shape, then the coordinate functions are positive for all \( v \) inside \( P \). However, if \( v \) is not in the kernel of \( P \), then the orientation of \( dS_v \) is negative and the coordinates functions may be negative as well.
3.2 Coordinates for piecewise linear shapes

In practice, the integral form of Equation 3.3 can be complicated to evaluate symbolically. However, in this section, we derive a simple, closed form solution for piecewise linear shapes in terms of the vertex positions and their associated function values. As a simple example to illustrate our approach, we first re-derive mean value coordinates for closed polygons via mean value interpolation. Next, we apply the same derivation to construct mean value coordinates for closed triangular meshes.

3.2.1 Mean value coordinates for closed polygons

Consider an edge $E$ of a closed polygon $P$ with vertices $\{p_1, p_2\}$ and associated values $\{g_1, g_2\}$. Our first task is to convert this discrete data into a continuous form suitable for use in Equation 3.3. We can linearly parameterize the edge $E$ via

$$p(x) = \sum_i \phi_i(x)p_i$$

where $\phi_1(x) = (1 - x)$ and $\phi_2(x) = x$. We then use this same parameterization to extend the data values $g_1$ and $g_2$ linearly along $E$. Specifically, we let $g(x)$ have the form

$$g(x) = \sum_i \phi_i(x)g_i.$$

Now, our task is to evaluate the integrals in Equation 3.3 for $0 \leq x \leq 1$. Let $\overline{E}$ be the circular arc formed by projecting the edge $E$ onto the unit circle $S_v$, we can rewrite the integrals of Equation 3.3 restricted to $\overline{E}$ as

$$\frac{\int_x w(x, v)g(x)dE}{\int_x w(x, v)dE} = \frac{\sum_i w_i g_i}{\sum_i w_i} \tag{3.4}$$

To evaluate the integral of Equation 3.3, we can relate the differential $dS_v$ to $dx$ via

$$dS_v = \frac{p^+(x).(p(x) - v)}{|p(x) - v|^2}dx$$

where $p^+(x)$ is the cross product of the $n - 1$ tangent vectors $\frac{\partial p(x)}{\partial s_i}$ to $P$ at $p(x)$. Note that the sign of this expression correctly captures whether $P$ has folded back during its projection onto $S_v$. 
where weights \( u_i = \int_x \frac{\phi_i(x)}{p(x) - v} dE \).

Our next goal is to compute the corresponding weights \( u_i \) for edge \( E \) in Equation 3.4 without resorting to symbolic integration (since this will be difficult to generalize to 3D). Observe that the following identity relates \( u_i \) to a vector,

\[
\sum_i w_i (p_i - v) = m. \tag{3.5}
\]

where \( m = \int_x \frac{p(x) - v}{|p(x) - v|} dE \) is simply the integral of the outward unit normal over the circular arc \( E \). We call \( m \) the mean vector of \( E \), as scaling \( m \) by the length of the arc yields the centroid of the circular arc \( E \). Based on 2D trigonometry, \( m \) has a simple expression in terms of \( p_1 \) and \( p_2 \). Specifically,

\[
m = \int_{\theta_1}^{\theta_2} (\cos(x), \sin(x)) dx
= \tan \left( \frac{\theta_2 - \theta_1}{2} \right) \left( \cos(\theta_1) + \cos(\theta_2), \sin(\theta_1) + \sin(\theta_2) \right)
= \tan(\alpha/2) \left( \frac{p_1 - v}{|p_1 - v|} + \frac{p_2 - v}{|p_2 - v|} \right)
\]

where \( \alpha \) denotes the angle between \( p_1 - v \) and \( p_2 - v \). Hence we obtain \( u_i = \tan(\alpha/2)/|p_i - v| \) which agrees with the Floater’s weighting function defined in Equation 3.2 for 2D mean value coordinates when restricted to a single edge of a polygon.

Equation 3.4 allows us to formulate a closed form expression for the interpolant \( f(v) \) in Equation 3.3 by summing the integrals for all edges \( E_k \) in \( P \) (note that we add the index \( k \) for enumeration of edges):

\[
f(v) = \frac{\sum_k \sum_i w_i^k g_i^k}{\sum_k \sum_i w_i^k} \tag{3.6}
\]

where \( w_i^k \) and \( g_i^k \) are weights and values associated with edge \( E_k \).

### 3.2.2 Mean value coordinates for closed meshes

We now consider our primary application of mean value interpolation for this chapter; the derivation of mean value coordinates for triangular meshes. These coordinates are the natural generalization of 2D mean value coordinates.
Given triangle $T$ with vertices $\{p_{1}, p_{2}, p_{3}\}$ and associated values $\{g_{1}, g_{2}, g_{3}\}$, our first task is to define the functions $p(x)$ and $g(x)$ used in Equation 3.3 over $T$. To this end, we simply use the linear interpolation formula of Equation 3.1. The resulting function $g(x)$ is a linear combination of the values $g_{i}$ times basis functions $\phi_{i}(x)$.

As in 2D, the integral of Equation 3.3 reduces to the sum in Equation 3.6. In this case, the weights $w_{i}$ have the form

$$w_{i} = \int_{x} \frac{\phi_{i}(x)}{|p(x) - v|} d\overline{T}$$

where $\overline{T}$ is the projection of triangle $T$ onto $S_{v}$. To avoid computing this integral directly, we instead relate the weights $w_{i}$ to the mean vector $m$ for the spherical triangle $\overline{T}$ by inverting Equation 3.5. In matrix form,

$$\{w_{1}, w_{2}, w_{3}\} = m \{p_{1} - v, p_{2} - v, p_{3} - v\}^{-1}$$  \hspace{1cm} (3.7)

All that remains is to derive an explicit expression for the mean vector $m$ for a spherical triangle $\overline{T}$. The following theorem solves this problem.

**Theorem 3.2.1** Given a spherical triangle $\overline{T}$, let $\theta_{i}$ be the length of its $i^{th}$ edge (a circular arc) and $n_{i}$ be the inward unit normal to the face formed by the $i^{th}$ edge and the vertex $v$ (see Figure 3.4 (b)). Then,

$$m = \sum_{i} \frac{1}{2} \theta_{i} n_{i}$$  \hspace{1cm} (3.8)

where $m$, the mean vector, is the integral of the outward unit normals over $\overline{T}$.

**Proof:** Consider the solid triangular wedge of the unit sphere with cap $\overline{T}$. The integral of outward unit normals over a closed surface is always exactly zero [11, p.342]. Thus, we can partition the integral into three triangular faces whose outward normals are $-n_{i}$ with associated areas $\frac{1}{2} \theta_{i}$. The theorem follows since $m - \sum_{i} \frac{1}{2} \theta_{i} n_{i}$ is then zero. ⊥

Note that a similar result holds in 2D, where the mean vector $m$ defined by Equation 3.2.1 for a circular arc $\overline{E}$ on the unit circle can be interpreted as the sum
Figure 3.4: Mean vector \( m \) on a circular arc \( E \) with edge normals \( n_i \) (a) and on a spherical triangle \( T \) with arc lengths \( \theta_i \) and face normals \( n_i \).

of the two inward unit normals of the vectors \( p_i - v \) (see Figure 3.4 (a)). In 3D, the lengths \( \theta_i \) of the edges of the spherical triangle \( T \) are the angles between the vectors \( p_{i-1} - v \) and \( p_{i+1} - v \) while the unit normals \( n_i \) are formed by taking the cross product of \( p_{i-1} - v \) and \( p_{i+1} - v \). Given the mean vector \( m \), we now compute the weights \( w_i \) using Equation 3.7 (but without doing the matrix inversion) via

\[
w_i = \frac{n_i \cdot m}{n_i \cdot (p_i - v)} \tag{3.9}\]

At this point, we should note that projecting a triangle \( T \) onto \( S_v \) may reverse its orientation. To guarantee linear precision, these folded-back triangles should produce negative weights \( w_i \). If we maintain a positive orientation for the vertices of every triangle \( T \), the mean vector computed using Equation 3.8 points towards the projected spherical triangle \( \overline{T} \) when \( T \) has a positive orientation and away from \( \overline{T} \) when \( T \) has a negative orientation. Thus, the resulting weights have the appropriate sign.

### 3.2.3 Robust mean value interpolation

The discussion in the previous section yields a simple evaluation method for mean value interpolation on triangular meshes. Given a point \( v \) and a closed mesh, for each triangle \( T \) in the mesh with vertices \( \{p_1, p_2, p_3\} \) and associated values \( \{g_1, g_2, g_3\} \),

1. Compute the mean vector \( m \) via Equation 3.8
2. Compute the weights $w_i$ using Equation 3.9

3. Update the denominator and numerator of $f(v)$ defined in Equation 3.6 by adding $\sum_i w_i$ and $\sum_i w_i g_i$

To correctly compute $f(v)$ using the above procedure, however, we must overcome two obstacles. First, the weights $w_i$ computed by Equation 3.9 may have a zero denominator when the point $v$ lies on plane containing the face $T$. Our method must handle this degenerate case gracefully. Second, we must be careful to avoid numerical instability when computing $w_i$ for triangle $T$ with a small projected area. Such triangles are the dominant type when evaluating mean value coordinates on meshes with a large number of triangles. Next we discuss our solutions to these two problems and present the complete evaluation algorithm as pseudo-code in Figure 3.5.

- **Stability:**

  When the triangle $T$ has small projected area on the unit sphere centered at $v$, computing weights using Equation 3.8 and 3.9 becomes numerically unstable due to canceling of unit normals $n_i$ that are almost co-planar. To this end, we next derive a stable formula for computing weights $w_i$. First, we substitute Equation 3.8 into Equation 3.9. Using trigonometric identities we obtain

  $$w_i = \frac{\theta_i - \cos(\psi_{i+1})\theta_{i-1} - \cos(\psi_{i-1})\theta_{i+1}}{2\sin(\psi_{i+1})\sin(\theta_{i-1})|p_i - v|},$$

  (3.10)

  where $\psi_i (i = 1, 2, 3)$ denotes the angles in the spherical triangle $\overline{T}$. Note that the $\psi_i$ are the dihedral angles between the faces with normals $n_{i-1}$ and $n_{i+1}$. We illustrate the angles $\psi_i$ and $\theta_i$ in Figure 3.4 (b).

  To calculate the $\cos$ of the $\psi_i$ without computing unit normals, we apply the half-angle formula for spherical triangles [5],

  $$\cos(\psi_i) = \frac{2 \sin(h) \sin(h - \theta_i)}{\sin(\theta_{i+1})\sin(\theta_{i-1})} - 1,$$

  (3.11)
where \( h = (\theta_1 + \theta_2 + \theta_3)/2 \). Substituting Equation 3.11 into 3.10, we obtain a formula for computing \( w_i \) that only involves lengths \(|p_i - v|\) and angles \( \theta_i \). In the pseudo-code from Figure 3.5, angles \( \theta_i \) are computed using \( \arcsin \), which is stable for small angles.

- **Co-planar cases:** Observe that Equation 3.9 involves division by \( n_i \cdot (p_i - v) \), which becomes zero when the point \( v \) lies on plane containing the face \( T \). Here we need to consider two different cases. If \( v \) lies on the plane inside \( T \), the continuity of mean value interpolation implies that \( f(v) \) converges to the value \( g(x) \) defined by linear interpolation of the \( g_i \) on \( T \). On the other hand, if \( v \) lies on the plane outside \( T \), the weights \( w_i \) become zero as their integral definition \( \int \frac{\psi_i(x)}{|p(x) - v|} d\Gamma \) becomes zero. We can easily test for the first case because the sum \( \Sigma \theta_i = 2\pi \) for points inside of \( T \). To test for the second case, we use Equation 3.11 to generate a stable computation for \( \sin(\psi_i) \). Using this definition, \( v \) lies on the plane outside \( T \) if any of the dihedral angles \( \psi_i \) (or \( \sin(\psi_i) \)) are zero.

### 3.3 Applications and results

While mean value coordinates find their main use in boundary value interpolation, these coordinates can be applied to a variety of applications. In this section, we briefly discuss several of these applications including constructing volumetric textures and surface deformation. We conclude with a section on our implementation of these coordinates and provide evaluation times for various shapes.

#### 3.3.1 Boundary value interpolation

As mentioned in the introduction to this chapter, these coordinate functions may be used to perform boundary value interpolation for triangular meshes. In this case, function values are associated with the vertices of the mesh. The function constructed
by our method is smooth, interpolates those vertex values and is a linear function on the faces of the triangles. Figure 3.6 shows an example of interpolating hue specified on the surface of a cow. In the top-left is the original model that serves as input into our algorithm. The rest of the figure shows several slices of the cow model, which reveal the volumetric function produced by our coordinates. Notice that the function is smooth on the interior and interpolates the colors on the surface of the cow.
3.3.2 Volumetric textures

These coordinate functions have applications to volumetric texturing as well. Figure 3.7 (top-left) illustrates a model of a bunny with a 2D texture applied to the surface. Using the texture coordinates \((u, v)\) as the \(g_i\) for each vertex, we apply our coordinates and build a function that interpolates the texture coordinates specified at the vertices and along the polygons of the mesh. Our function extrapolates these surface values to the interior of the shape to construct a volumetric texture. Figure 3.7 shows several slices revealing the volumetric texture within.

3.3.3 Surface Deformation

Surface deformation is an application of mean value coordinates that depends on the linear precision property outlined in Section 3.1. In this application, we are given two shapes: a model and a control mesh. For each vertex \(v\) in the model, we first compute
Figure 3.7: Textured bunny (top-left). Cuts of the bunny to expose the volumetric texture constructed from the surface texture.

its mean value weight functions \( w_j \) with respect to each vertex \( p_j \) in the undeformed control mesh. To perform the deformation, we move the vertices of the control mesh to induce the deformation on the original surface. Let \( q_j \) be the positions of the vertices from the deformed control mesh. Then the new vertex position \( f(v) \) in the deformed model is computed as

\[
f(v) = \frac{\sum_j w_j q_j}{\sum_j w_j}.
\]

Notice that, due to linear precision, if \( q_j = p_j \), then \( f(v) = v \). Figures 3.8 and 3.9 show several examples of deformations generated with this process. Figure 3.8 (a) depicts a horse before deformation and the surrounding control mesh shown in black. Moving the vertices of the control mesh generates the smooth deformations of the horse shown in (b,c,d).

Previous deformation techniques such as freeform deformations \([45, 38]\) require volumetric cells to be specified on the interior of the control mesh. The deformations
Figure 3.8: Original horse model with enclosing triangle control mesh shown in black (a). Several deformations generated using our 3D mean value coordinates applied to a modified control mesh (b,c,d).

Produced by these methods depend on how the control mesh is decomposed into volumetric cells. Furthermore, many of these techniques restrict the user to creating control meshes with quadrilateral faces.

In contrast, this barycentric deformation technique allows the artist to specify an arbitrary closed triangular surface as the control mesh and does not require volumetric cells to span the interior. Our technique also generates smooth, realistic looking deformations even with a small number of control points and is quite fast. Generating the mean value coordinates for Figure 3.8 took 3.3s and 1.9s for Figure 3.9. However, each of the deformations only took 0.09s and 0.03s respectively, which is fast enough to apply these deformations in real-time.

3.3.4 Implementation

Our implementation follows the pseudo-code from Figure 3.5 very closely. However, to speed up computations, it is helpful to pre-compute as much information as possible.

Figure 3.10 contains the number of evaluations per second for various models sampled on a 3GHz Intel Pentium 4 computer. Previously, practical applications
involving barycentric coordinates have been restricted to 2D polygons containing a very small number of line segments. In this chapter, for the first time, barycentric coordinates have been applied to truly large models (on the order of 100,000 polygons). The coordinate computation is a global computation and all vertices of the surface must be used to evaluate the function at a single point. However, much of the time spent is determining whether or not a point lies on the plane of one of the triangles in the mesh and, if so, whether or not that point is inside that triangle. Though we have not done so, using various spatial partitioning data structures to reduce the number of triangles that must be checked for coplanarity could greatly enhance the speed of
<table>
<thead>
<tr>
<th>Model</th>
<th>Tris</th>
<th>Verts</th>
<th>Eval/s</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horse control mesh (fig 3.8)</td>
<td>98</td>
<td>51</td>
<td>16281</td>
</tr>
<tr>
<td>Armadillo control mesh (fig 3.9)</td>
<td>216</td>
<td>111</td>
<td>7644</td>
</tr>
<tr>
<td>Cow (fig 3.6)</td>
<td>5804</td>
<td>2903</td>
<td>328</td>
</tr>
<tr>
<td>Bunny (fig 3.7)</td>
<td>69630</td>
<td>34817</td>
<td>20</td>
</tr>
</tbody>
</table>

Figure 3.10: Number of evaluations per second for various models.

the evaluation.

3.4 Conclusions and Future Work

Mean value coordinates are a simple, but powerful method for creating functions that interpolate values assigned to the vertices of a closed mesh. Perhaps the most intriguing feature of mean value coordinates is that fact that they are well-defined on both the interior and the exterior of the mesh. In particular, mean value coordinates do a reasonable job of extrapolating values outside of the mesh. We intend to explore applications of this feature in future work.

Also, we only consider meshes that have triangular faces. One important generalization would be to derive mean value coordinates for piecewise linear mesh with arbitrary closed polygons as faces. On these faces, the coordinates would degenerate to standard 2D mean value coordinates. We plan to address this topic in a future paper.

Finally, in Section 3.3.4 we mentioned that the deformations are globally supported meaning that if one vertex of the control mesh is modified, the entire surface deforms. However, these deformations exhibit "almost local control" meaning that the influence of a control point drops dramatically with the distance from that control point. So when the armadillo man in Figure 3.9 waves his hand, his foot moves. However, the movement is so slight that it may not even be within numerical accuracy.
Still, a method that provides true local control may be desirable and is an area of future research.
Chapter 4

Moving Least Squares Deformation

Image deformation has a number of uses from animation, to morphing [47] and medical imaging [54]. So far we have considered several types of deformation handles that the user manipulates to control the deformation. These handles have taken the form of polygon grids [38] or lines [19]. However, we have not considered using the simplest deformation handle: a set of points. While points are certainly a simpler primitive than grids or lines, they can actually be the most difficult to construct deformations from due to the lack of topological information. In this chapter, we construct a family of deformations for images using only points as the deformation handles.

Previous work

Previous work on image deformation has focused on specifying deformations using different types of handles. Grid-based techniques such as free-form deformations [45, 28] parameterize the image using bivariate cubic splines to create $C^2$ deformations. Typically these methods require aligning grid lines corresponding to the control points of the spline with features of the image, which can be cumbersome for the user.

Beier et al. [4] improve upon these grid-based techniques and allow the user to specify the deformation using sets of lines. This method is based on Shepard’s interpolant [46] and creates smooth deformations. However, the authors note that their method produces complicated warps that can sometimes suffer from "ghosts", undesirable folding in the deformation. Koba et al. [27] later generalized this technique to surface deformations.

Very few deformation methods investigate the type of transformations that are desirable for performing deformation. One notable exception is worked based on thin-
plate splines [6] that attempts to minimize the amount of bending in the deformation. Bookstein presents a deformation algorithm using the simplest deformation handle, a point, that employs radial basis functions with thin-plate splines. Figure 4.1 (left) shows an example of the deformation created with thin-plate splines for our example in figure 4.2. The deformation appears very similar to the affine-method in figure 4.2. In both cases, the test shape undergoes local non-uniform scaling and shearing, which is undesirable in many applications.

This chapter builds primarily on a recent paper by Igarashi et al. [23] that proposes a point-based image deformation technique for cartoon-like images in which the resulting deformations are as “rigid-as-possible”. Such deformation have the property that the amount of local scaling and shearing is minimized. (The concept of rigid-as-possible transformations was itself first introduced in Alexa [2].) To produce rigid-as-possible deformations, Igarashi et al. triangulate the input image and set up an energy functional that measures the amount of local scaling and shearing introduced during deformations of the triangulation. Their method then minimizes this energy functional using a linear solver. This method has two principal drawbacks. First, the use of piecewise linear deformations results in deformations that are not smooth across edges of the triangulation. While suitable for deformations of flat-shaded images such as cartoons, such deformations may not be acceptable for textured objects (see figure 4.1, right). Second, the size of the linear system solved during the optimization process varies in the size of the triangulation. Thus, increasing the density of the triangulation to mitigate these smoothness defects comes at the cost of increased deformation time.

**Contributions**

In this chapter, we propose an image deformation method based on linear Moving Least Squares (MLS). To construct deformations that minimize the amount of local scaling and shear, we restrict the classes of transformations used in Moving Least Squares to similarity and rigid-body transformations. By using MLS, we avoid
the need to triangulate the input image (as done in Igarashi et al.) and produce deformations that are globally smooth.)

Next, we derive closed-form formulas for both similarity and rigid MLS deformations. These formula are simple, easy to implement and provide real-time deformations. This derivation relies on a surprising and little-known relationship between similarity transformations and rigid transformations that minimize a common least squares problem. As opposed to Igarashi et al., our formulas do not require the use of a general linear solver.

As a natural extension of our point-based method, we extend our MLS deformation method from sets of points to sets of line segments and again provide closed-form expressions for the resulting deformation method. Finally, we conclude with an application to image morphing.

4.1 Moving Least Squares deformation

Here we consider building image deformations based on collections of points with which the user controls the deformation. Let $p$ be a set of control points and $q$ the deformed positions of the control points $p$. We construct a deformation function $f$
Figure 4.2: Deformation using Moving Least Squares. Original image with control points shown in blue (a). Moving Least Squares deformations using affine transformations (b), similarity transformations (c) and rigid transformations (d).

satisfying the three properties outlined in chapter 1 using a technique called Moving Least Squares [30]. Given a point \( v \) in the image, we solve for the best affine transformation \( l_v(x) \) that minimizes

\[
\sum_i w_i |l_v(p_i) - q_i|^2
\]

where \( p_i \) and \( q_i \) are row vectors and the weights \( w_i \) have the form

\[
w_i = \frac{1}{|p_i - v|^{2\alpha}}.
\]

Because the weights \( w_i \) in this least squares problem depend on the point of evaluation \( v \), we call this a Moving Least Squares minimization. Therefore, we obtain a different transformation \( l_v(x) \) for each \( v \).

Now we define our deformation function \( f \) to be \( f(v) = l_v(v) \). Observe that as \( v \) approaches \( p_i \), \( w_i \) approaches infinity and the function \( f \) interpolates, (i.e; \( f(p_i) = q_i \)). Furthermore, if \( q_i = p_i \), then each \( l_v(x) = x \) for all \( x \) and, therefore, \( f \) is the identity transformation \( f(v) = v \). Finally, this deformation function \( f \) has the property that it is smooth everywhere (except at the control points \( p_i \) when \( \alpha \leq 1 \)).

Since \( l_v(x) \) is an affine transformation, \( l_v(x) \) consists of two parts: a linear trans-
formation matrix $M$ and a translation $T$.

$$l_v(x) = xM + T$$ (4.2)

We can actually remove the translation $T$ from this minimization problem further simplifying these equations. Equation 4.1 is quadratic in $T$. Since the minimizer is where the derivatives with respect to each of the free variables in $l_v(x)$ are zero, we can solve directly for $T$ in terms of the matrix $M$. Taking the partial derivatives with respect to the free variables in $T$ produces a linear system of equations. Solving for $T$ yields that

$$T = q_* - p_* M$$

where $p_*$ and $q_*$ are weighted centroids.

$$p_* = \frac{\sum_i w_i p_i}{\sum_i w_i}$$
$$q_* = \frac{\sum_i w_i q_i}{\sum_i w_i}$$

With this observation we can substitute $T$ into equation 4.2 and rewrite $l_v(x)$ in terms of the linear matrix $M$.

$$l_v(x) = (x - p_*)M + q_*$$ (4.3)

Based on this insight, the least squares problem of equation 4.1 can be rewritten as

$$\sum_i w_i |\hat{p}_i M - \hat{q}_i|^2$$ (4.4)

where $\hat{p}_i = p_i - p_*$ and $\hat{q}_i = q_i - q_*$. Notice that Moving Least Squares is very general in that the matrix $M$ does not have to be a fully affine transformation. In fact, this framework allows us to investigate different classes of transformation matrices $M$. In particular, we are interested in the case where $M$ is a rigid transformation. However, we first examine the case where $M$ is an affine transformation as the derivation is the simplest. Next we construct deformations with similarity transformations and show how these solutions can be used to find closed-form solutions to Moving Least Square deformations with rigid transformations.
4.1.1 Affine deformations

Finding an affine deformation that minimizes equation 4.4 is straightforward using the classic normal equations solution.

\[ M = \left( \sum_i \hat{p}_i^T w_i \hat{p}_i \right)^{-1} \sum_j \hat{p}_j^T \hat{q}_j. \]

Though this solution requires the inversion of a matrix, the matrix is a constant size \((2 \times 2)\) and is fast to invert. With this closed-form solution for \(M\) we can write a simple expression for the deformation function \(f_a(v)\).

\[ f_a(v) = (v - p_\ast) \left( \sum_i \hat{p}_i^T w_i \hat{p}_i \right)^{-1} \sum_j \hat{p}_j^T \hat{q}_j + q_\ast. \] (4.5)

Applying this deformation function to each point in the image creates a new, deformed image.

While the user creates these deformations by manipulating the points \(q\), the points \(p\) are fixed. Since the \(p\) do not change during deformation, much of equation 4.5 can be precomputed yielding very fast deformations. In particular, we can rewrite equation 4.5 in the form

\[ f_a(v) = \sum_j A_j \hat{q}_j + q_\ast. \]

where \(A_j\) is a single scalar given by

\[ A_j = (v - p_\ast) \left( \sum_i \hat{p}_i^T w_i \hat{p}_i \right)^{-1} \hat{p}_j^T. \]

Notice that, given a point \(v\), everything in \(A_j\) can be precomputed yielding a simple, weighted sum. Table 4.1 provides timing results for the examples in this chapter, which shows that these deformations may be performed over 500 times per second in our examples.

Figure 4.2 (b) illustrates this affine Moving Least Squares deformation applied to our test image. Unfortunately, the deformation does not appear very desirable due to the stretching in the arms and torso. These artifacts are created because
affine transformations include deformations such as non-uniform scaling and shear. To eliminate these undesirable deformations, we need to consider restricting the linear transformation \( l_v(x) \). In particular, we modify the class of deformations \( l_v(x) \) produces by restricting the transformation matrix \( M \) from being fully linear to similarity and rigid-body transformations.

### 4.1.2 Similarity deformations

While affine transformations include effects such as non-uniform scaling and shear, many objects in reality do not undergo even these simple transformations. Similarity transformations are a special subset of affine transformations that include only compositions of translation, rotation and uniform scaling.

To alter our deformation technique to only use similarity transformations, we constrain the matrix \( M \) to have the property that \( M^TM = \lambda^2I \) for some scalar \( \lambda \). If \( M \) is a block matrix of the form

\[
M = \begin{pmatrix} M_1 & M_2 \end{pmatrix}
\]

where \( M_1 \), \( M_2 \) are column vectors of length 2, then restricting \( M \) to be a similarity transform requires that \( M_1^TM_1 = M_2^TM_2 = \lambda^2 \) and \( M_1^TM_2 = 0 \). This constraint implies that \( M_2 = M_1^\perp \) where \( \perp \) is an operator on 2D vectors such that \((x,y)^\perp = (-y,x)\). Though restricted, the minimization problem from equation 4.4 is still quadratic in \( M_1 \) and can be rephrased as finding the column vector \( M_1 \) that minimizes

\[
\sum_i w_i \left\| \begin{pmatrix} \hat{p}_i \\ \hat{p}_i^\perp \end{pmatrix} \right\| M_1 - \hat{q}_i^T \right\|^2.
\]

This quadratic function has a unique minimizer, which yields the optimal transformation matrix \( M \)

\[
M = \frac{1}{\mu_s} \sum_i w_i \begin{pmatrix} \hat{p}_i \\ \hat{p}_i^\perp \end{pmatrix} \begin{pmatrix} \hat{q}_i^T & \hat{q}_i^{\perp T} \end{pmatrix}
\]  
(4.6)
where
\[ \mu_s = \sum_i w_i \hat{p}_i \hat{p}_i^T. \]

Similar to the affine deformations, the user manipulates the \( q \) to produce the deformation while the \( p \) remain fixed. Using this observation we write the deformation function \( f_s(v) \) in a form that allows us to precompute as much information as possible. \( f_s(v) \) is then
\[ f_s(v) = \sum_i \hat{q}_i \left( \frac{1}{\mu_s} A_i \right) + q_* \]
where \( \mu_s \) and \( A_i \) depend only on the \( p_i \) and \( v \) and can be precomputed and \( A_i \) is
\[
A_i = w_i \begin{pmatrix} \hat{p}_i & v - p_* \end{pmatrix}^T \begin{pmatrix} \hat{p}_i \ - (v - p_*)^\perp \end{pmatrix}. \tag{4.7}
\]

As expected, similarity MLS deformations preserves angles in the original image better than affine MLS deformations. (Transformations that strictly preserve angle are called conformal transformations and have been studied extensively in [17].) While approximate (or exact) angle preservation is a desirable property in many cases, allowing local scaling can often lead to undesirable deformations. Figure 4.2 (c) shows an example of applying the similarity Moving Least Squares deformation to our test image. The result is a much more realistic looking deformation than (b). However, this deformation scales the size of the upper arm as it is stretched. To remove this scaling, we consider building deformations using only rigid transformations.

### 4.1.3 Rigid deformations

Recently, several papers [2, 23] have shown that, for realistic shapes, deformations should be as rigid as possible; that is, the space of deformations should not even include uniform scaling. Traditionally researchers in deformation have been reluctant to approach this problem directly due to the non-linear constraint \( M^T M = I \). However, we note that closed-form solutions to this problem are known from the Iterated Closest Point community [21]. Horn shows that the optimal rigid transformation can
be found in terms of eigenvalues and eigenvectors of a covariance matrix involving the points \( p_i \) and \( q_i \). We show that these rigid deformations are related to the similarity deformations from section 4.1.2 via the following theorem.

**Theorem 4.1.1** Let \( C \) be the matrix that minimizes the following similarity functional

\[
\min_{M^T M = \lambda I} \sum_i w_i |\hat{p}_i M - \hat{q}_i|^2.
\]

If \( C \) is written in the form \( \lambda R \) where \( R \) is a rotation matrix and \( \lambda \) is a scalar, the rotation matrix \( R \) minimizes the rigid functional

\[
\min_{M^T M = I} \sum_i w_i |\hat{p}_i M - \hat{q}_i|^2.
\]

**Proof:** First, we expand both of the above error functions into their quadratic forms yielding

\[
\begin{align*}
\min_{R^T R = I, \lambda} & \sum_i w_i \left( \lambda^2 \hat{p}_i \hat{p}_i^T - 2\lambda \hat{p}_i R \hat{q}_i^T + \hat{q}_i \hat{q}_i^T \right) \\
\min_{R^T R = I} & \sum_i w_i \left( \hat{p}_i \hat{p}_i^T - 2\hat{p}_i R \hat{q}_i^T + \hat{q}_i \hat{q}_i^T \right)
\end{align*}
\]

These minimization problems are very similar. We find the matrices that minimize these error functions by differentiating the functions with respect to the free variables.
\[ \sum_i w_i \left( -2 \lambda \hat{q}_i \frac{\partial R}{\partial \theta_j} q_i^T \right) = 0 \]
\[ \sum_i w_i \left( -2 \hat{p}_i \frac{\partial R}{\partial \theta_j} q_i^T \right) = 0 \]

Now, unless \( \lambda = 0 \), which implies a degenerate transformation, these equations are equal. Since \( C = \lambda R \), this implies that \( \pm R \) minimizes the quadratic function using rigid transformations. The negative solution corresponds to a maximum while the positive solution is the minimum. QED

This theorem is valid in arbitrary dimension, however, it is very easy to apply in 2D. Using this theorem, we find that the rigid transformation is exactly the same as equation 4.6 except that we use a different constant \( \mu_r \) in the solution so that \( M^T M = I \) given by

\[ \mu_r = \sqrt{\left( \sum_i w_i \hat{q}_i \hat{p}_i^T \right)^2 + \left( \sum_i w_i \hat{q}_i \hat{p}_i^{+T} \right)^2} \]

Unlike the similarity deformation \( f_s(v) \), we cannot precompute as much information for the rigid deformation function \( f_r(v) \). However, the deformation process can still be made very efficient. Let

\[ \vec{f}_r(v) = \sum_i \hat{q}_i A_i \]

where \( A_i \) is defined in equation 4.7, which may be precomputed. This vector \( \vec{f}_r(v) \) is the vector generated from the best similarity transformation (minus the scaling from \( \mu_s \)) and is a rotated and scaled version of the vector \( v - p_* \). To compute \( f_r(v) \) we remove the scale by normalizing \( \vec{f}_r \) and scaling by the length of \( v - p_* \) (which also can be precomputed). Finally, we add the translation \( q_* \).

\[ f_r(v) = |v - p_*| \frac{\vec{f}_r(v)}{|\vec{f}_r(v)|} + q_* \]

This method is slower than the similarity deformation due to the normalization; however, these deformations are still very fast as shown in table 4.1.

Figure 4.2 (d) shows this rigid deform applied to the test image in (a). As opposed to the other methods, this deformation is quite realistic and almost feels as if the
user is manipulating a real object. Figures 4.3 and 4.4 show additional examples of this rigid deformation method. In the figure with the Mona Lisa, we deform the image to create a thinner facial profile and make her smile. In the figure with the horse, we stretch the horses legs and neck to create a giraffe. Due to the use of rigid transformations, the deformation maintains rigidity and scale locally so that the body and head of the horse retain their relative shape.

4.2 Deformation with line segments

So far we have considered creating deformations with Moving Least Squares using only sets of points to control the deformation. In applications where precise control over curves such as profiles in the image is needed, points may be insufficient for specifying these deformations. One solution that allows the user to control curves precisely is to convert these curves to dense sets of points and apply a point-based deformation [58]. The disadvantage of this approach is that the computation time of the deformation is proportional to the number of control points used and creating
large numbers of control points adversely affects performance.

Figure 4.5: Deformation of the Leaning Tower of Pisa. From left to right: original image, Affine MLS, Similarity MLS and Rigid MLS deformations.

Alternatively, we desire a generalization of these Moving Least Squares deformations from section 4.1 to arbitrary curves in the plane. First, assume \( p_i(t) \) is the \( i \)th control curve and \( q_i(t) \) is the deformed curve corresponding to \( p_i(t) \). We generalize the quadratic function in equation 4.1 by integrating over each control curve \( p_i(t) \) where we assume \( t \in [0, 1] \).

\[
\sum_i \int_0^1 w_i(t) \left| p_i(t) M + T - q_i(t) \right|^2
\]

where \( w_i(t) \) is

\[
w_i(t) = \frac{|p'_i(t)|}{|p_i(t) - v|^{2\alpha}}
\]

and \( p'_i(t) \) is the derivative of \( p_i(t) \). (This factor of \( |p'(t)| \) makes the integrals independent of the parameterization of the curve \( p_i(t) \).) Now notice that, despite the integral, equation 4.9 is still quadratic in \( T \), which can be solved for in terms of the matrix \( M \).

\[
T = q_* - p_* M
\]

where \( p_* \) and \( q_* \) are again weighted centroids.

\[
p_* = \frac{\sum_i \int_0^1 w_i(t) p_i(t) dt}{\sum_i \int_0^1 w_i(t) dt}
\]

\[
q_* = \frac{\sum_i \int_0^1 w_i(t) q_i(t) dt}{\sum_i \int_0^1 w_i(t) dt}
\]
Therefore, we rewrite equation 4.9 only in terms of \( M \) as

\[
\sum_i \int_0^1 w_i(t) |\hat{p}_i(t)M - \hat{q}_i(t)|^2
\]

\[\text{(4.11)}\]

where

\[
\hat{p}_i(t) = p_i(t) - p_c,
\]

\[
\hat{q}_i(t) = q_i(t) - q_c.
\]

Until now, \( p_i(t) \) and \( q_i(t) \) have been arbitrary curves. However, the integrals in equation 4.11 may be difficult to evaluate for arbitrary functions. Instead, we restrict these functions to be line segments and derive closed-form solutions for the deformations in terms of the end-points of these segments. Similar to section 4.1, we first consider affine transformations due to the relatively simple derivation and then move to similarity transformations, which we use to create closed-form solutions to the equivalent problem with rigid-body transformations.

### 4.2.1 Affine lines

Since \( \hat{p}_i(t), \hat{q}_i(t) \) are line segments, we can represent these curves as matrix products

\[
\hat{p}_i(t) = \begin{pmatrix} 1 - t & t \end{pmatrix} \begin{pmatrix} \hat{a}_i \\ \hat{b}_i \end{pmatrix}
\]

\[
\hat{q}_i(t) = \begin{pmatrix} 1 - t & t \end{pmatrix} \begin{pmatrix} \hat{c}_i \\ \hat{d}_i \end{pmatrix}
\]

where \( \hat{a}_i, \hat{b}_i \) are the end-points of \( \hat{p}_i(t) \) and \( \hat{c}_i, \hat{d}_i \) are the end-points of \( \hat{q}_i(t) \). Equation 4.11 is then written as

\[
\sum_i \int_0^1 \left| \begin{pmatrix} 1 - t & t \end{pmatrix} \begin{pmatrix} \hat{a}_i \\ \hat{b}_i \end{pmatrix} M - \begin{pmatrix} \hat{c}_i \\ \hat{d}_i \end{pmatrix} \right|^2
\]

\[\text{(4.12)}\]

whose minimizer is

\[
M = \left( \sum_i \begin{pmatrix} \hat{a}_i \\ \hat{b}_i \end{pmatrix}^T W_i \begin{pmatrix} \hat{a}_i \\ \hat{b}_i \end{pmatrix} \right)^{-1} \sum_j \begin{pmatrix} \hat{a}_j \\ \hat{b}_j \end{pmatrix}^T W_j \begin{pmatrix} \hat{c}_j \\ \hat{d}_j \end{pmatrix}
\]
where $W_i$ is a weight matrix given by

$$W_i = \begin{pmatrix} \delta_{i00} & \delta_{i01} \\ \delta_{i01} & \delta_{i11} \end{pmatrix}$$

and the $\delta_i$ are integrals of the weight function $w_i(t)$ multiplied by the different quadratic polynomials.

$$\delta_{i00} = \int_0^1 w_i(t)(1-t)^2 dt$$
$$\delta_{i01} = \int_0^1 w_i(t)(1-t)t dt$$
$$\delta_{i11} = \int_0^1 w_i(t)t^2 dt$$

In order to complete the derivation, we need closed-form solutions for integrals of three quadratic polynomials times the weight function $w_i(t)$ over the line segments. These integrals have closed-form solutions for various values of $\alpha$, and we provide a solution for $\alpha = 2$, though other solutions can be computed with the aid of a symbolic integration package. Let $a_i, b_i$ be the endpoints of the line segment described by $p_i(t)$ and let

$$\Delta_i = (a_i - v)^T(a_i - b_i)^T$$
$$\theta_i = \tan^{-1}\left(\frac{(b_i - v)(b_i - a_i)^T}{(b_i - v)(a_i - a_i)^T}\right) - \tan^{-1}\left(\frac{(a_i - v)(a_i - b_i)^T}{(a_i - v)(a_i - b_i)^T}\right)$$
$$\beta_{i00} = (a_i - v)(a_i - v)^T$$
$$\beta_{i01} = (a_i - v)(v - b_i)^T$$
$$\beta_{i11} = (v - b_i)(v - b_i)^T.$$

The integrals then have the closed-form solution

$$\delta_{i00} = \frac{|a_i - b_i|}{2\Delta_i} \left(\frac{\beta_{i01}}{\beta_i} - \frac{\beta_{i11} \theta_i}{\Delta_i}\right)$$
$$\delta_{i01} = \frac{|a_i - b_i|}{2\Delta_i} \left(1 - \frac{\beta_{i01} \theta_i}{\Delta_i}\right)$$
$$\delta_{i11} = \frac{|a_i - b_i|}{2\Delta_i} \left(\frac{\beta_{i01}}{\beta_i^T} - \frac{\beta_{i11} \theta_i}{\Delta_i}\right).$$

When $v$ is on the line segment defined by $a_i$ and $b_i$, these integrals do not need to be evaluated because the function $f(v)$ interpolates the line segments. However, if $v$ is
on the extension of one of these line segments, \( \Delta_i = 0 \) and these integrals reduce to

\[
\begin{align*}
\delta_i^{00} &= \frac{|a_i - b_i|^5}{3((v-b_i)(b_i-a_i)^2)((a_i-v)(b_i-a_i)^2)^{3/2}} \\
\delta_i^{01} &= \frac{|a_i - b_i|^5}{6((v-b_i)(b_i-a_i)^2)^2((a_i-v)(b_i-a_i)^2)^{1/2}} \\
\delta_i^{11} &= \frac{|a_i - b_i|^5}{3((v-b_i)(b_i-a_i)^2)^{3/2}((a_i-v)(b_i-a_i)^2)^{1/2}}.
\end{align*}
\]

Note that these integrals can also be used to evaluate \( p_* \) and \( q_* \) from equation 4.10.

\[
\begin{align*}
p_* &= \frac{\sum_i a_i (\delta_i^{00} + \delta_i^{01}) + b_i (\delta_i^{01} + \delta_i^{11})}{\sum_i \delta_i^{00} + 2\delta_i^{01} + \delta_i^{11}} \\
q_* &= \frac{\sum_i c_i (\delta_i^{01} + \delta_i^{11}) + d_i (\delta_i^{01} + \delta_i^{11})}{\sum_i \delta_i^{00} + 2\delta_i^{01} + \delta_i^{11}}
\end{align*}
\]

As before, we write the deformation function \( f_a(v) \) as

\[
f_a(v) = \sum_j A_j \begin{pmatrix} \hat{c}_j \\ \hat{d}_j \end{pmatrix} + q_*
\]

where \( A_j \) is a \( 1 \times 2 \) matrix of the form

\[
A_j = (v - p_*) \left( \sum_i \begin{pmatrix} \hat{a}_i \\ \hat{b}_i \end{pmatrix}^T W_i \begin{pmatrix} \hat{a}_i \\ \hat{b}_i \end{pmatrix} \right)^{-1} \begin{pmatrix} \hat{a}_j \\ \hat{b}_j \end{pmatrix}^T W_j.
\]

During the deformation, the end-points \( a_i \) and \( b_i \) of the line segment \( p_i(t) \) are fixed while the user manipulates the end-points \( c_i \) and \( d_i \) of the line segments \( q_i(t) \). Since \( A_j \) is independent of \( c_i \) and \( d_i \), \( A_j \) can be precomputed.

Figure 4.5 shows an example deformation performed with line segments, where we modify the Leaning Tower of Pisa to lean in the opposite direction and shrink the tower. The Affine MLS deformation shears the tower to the side instead rotating the tower and does not appear to be realistic. To remove this shear effect, we restrict the matrix in equation 4.11 to be a similarity or rigid-body transformation.

### 4.2.2 Similarity lines

Restricting equation 4.12 to similarity transforms requires that \( M^T M = \lambda^2 I \) for some scalar \( \lambda \). As noted in section 4.1.2, \( M \) can be parameterized using a single column
vector $M_1$ yielding

$$\sum_i \int_0^1 \left[ \begin{pmatrix} 1 - t & 0 & t & 0 \\ 0 & 1 - t & 0 & t \end{pmatrix} \begin{pmatrix} \hat{a}_i \\ -\hat{a}_i^\perp \\ \hat{b}_i \\ -\hat{b}_i^\perp \end{pmatrix} M_1 - \begin{pmatrix} \hat{c}_i^T \\ \hat{d}_i^T \end{pmatrix} \right]^2$$

This error function is quadratic in $M_1$. To find the minimizer, we differentiate with respect to the free variables in $M_1$ and solve the linear system of equations to obtain the matrix $M$.

$$M = \frac{1}{\mu_s} \sum_j \begin{pmatrix} \hat{a}_j \\ -\hat{a}_j^\perp \\ \hat{b}_j \\ -\hat{b}_j^\perp \end{pmatrix}^T W_j \begin{pmatrix} \hat{c}_j^T \\ \hat{d}_j^T \\ \hat{c}_j^\perp T \\ \hat{d}_j^\perp T \end{pmatrix}$$  (4.13)

where $W_j$ is a weight matrix

$$W_j = \begin{pmatrix} \delta_{j0}^0 & 0 & \delta_{j0}^1 & 0 \\ 0 & \delta_{j0}^0 & 0 & \delta_{j0}^1 \\ \delta_{j1}^1 & 0 & \delta_{j1}^{11} & 0 \\ 0 & \delta_{j1}^1 & 0 & \delta_{j1}^{11} \end{pmatrix}$$

and $\mu_s$ is again a scaling constant, which has the form

$$\mu_s = \sum_i \hat{a}_i \hat{a}_i^T \delta_{i0}^0 + 2\hat{a}_i \hat{b}_i^\perp \delta_{i0}^{11} + \hat{b}_i \hat{b}_i^\perp \delta_{i1}^{11}.$$ 

This deformation function has a very similar structure to the point-based similarity deformation. Using this matrix, we write $f_s(v)$ explicitly as

$$f_s(v) = \sum_j (\hat{c}_j \cdot \hat{d}_j \cdot \frac{1}{\mu_s} A_j) + q_s$$

where $A_j$ is a $4 \times 2$ matrix.

$$A_j = W_j \begin{pmatrix} \hat{a}_j \\ -\hat{a}_j^\perp \\ \hat{b}_j \\ -\hat{b}_j^\perp \end{pmatrix} \begin{pmatrix} v - p_s \\ -(v - p_s)^\perp \end{pmatrix}^T$$  (4.14)
Figure 4.6: Comparison of the line deformation method of Beier et al. (left) with the Rigid MLS deformation (right).

Figure 4.5 shows the tower deformed using this similarity-based method. In contrast to the affine method, the tower actually appears to be rotated, not sheared, to the left resulting in a more realistic deformation. Similarity transformations contain uniform scaling, which is apparent from the way in which the tower shrinks with the line segment. Rigid transformations remove this uniform scaling.

4.2.3 Rigid lines

Using the solution from section 4.2.2 and Theorem 4.1.1, we immediately have a closed form solution for rigid-body transformations. The transformation matrix is, therefore, the same as equation 4.13 except we choose a different scaling constant $\mu_r$ so that $M^TM = I$.

$$\mu_r = \left| \sum_j \begin{pmatrix} \hat{a}_j^T & -\hat{a}_j^T \hat{b}_j^T & -\hat{b}_j^T \\ \hat{c}_j^T \end{pmatrix} W_j \begin{pmatrix} \hat{c}_j^T \\ \hat{d}_j^T \end{pmatrix} \right|$$
Figure 4.7: From left to right: original image, rigid MLS deformation, similarity MLS deformation, blended deformation with eyes set to rigid and nose to similarity.

This deformation is non-linear, but we can compute it in a simple fashion using equation 4.8. This equation uses the rotated vector $\vec{f}_r(v)$, scales the vector so that its length is $|v - p_\ast|$ and translates by $q_\ast$. For this deformation using line segments, the rotated vector is given by

$$\vec{f}_r(v) = \sum_j (\hat{c}_j \hat{d}_j) A_j$$

where $A_j$ is from equation 4.14.

Figure 4.5 (right) shows a deformation of the tower using this rigid method. In this deformation, the tower is rotated but does not shrink as the similarity deformation does. Instead the effect is almost the same as non-uniform scaling along the direction of the line segment.

Figure 4.6 also shows a comparison of the rigid deformation technique (right) with the line deformation method of Beier et al. [4] (left). The warps created with Beier et al.'s method fold and pull in unrealistic ways whereas the rigid method does not suffer from these same defects.

### 4.3 Blended family

So far we have derived three different deformation methods corresponding to affine, similar and rigid transformations. However, in some situations, the user may want deformations such as uniform scaling not present in the more restrictive classes. Since
$f_s(v)$ and $f_r(v)$ are so closely related, we can construct a family of methods that blends between each of these solutions such that some regions of the image may be governed by rigid transformations whereas other regions are locally similarity transformations.

Given a blending parameter $\lambda_v$, we blend the scale of the solutions given by the similarity and rigid Moving Least Squares deformations together.

$$f(v) = (1 - \lambda_v)f_s(v) + \lambda_v f_r(v)$$

To find $\lambda_v$ for each point $v$ in the image, we associate a blending parameter $\lambda_i$ with each control point $p_i$. Then, given a point $v$, we use Shepard’s interpolant [46] to average the parameters $\lambda_i$ together. Figure 4.7 shows an example of a deformation of the Mona Lisa using rigid and similarity-based Moving Least Squares deformations as well as the blended deformation. In this example, we move the bottom control point so as to magnify the jaw. With rigid deformations, the face does not change scale and is simply stretched downwards. Similarity deformations allow the jaw to change size but also magnify the rest of the image even around the eyes whose control points did not move. On the right we use a blended version of these deformations with the eyes being rigid and the bottom control point a similarity transformation.

### 4.4 Applications and implementation

#### 4.4.1 Image morphing

Once we have a method for deforming an image, we also have a method for morphing between two different images [58]. For example, in Figure 4.8 we are given images of two different people from the AR Face Database [40] (middle row, left and right sides). The goal is to create an animation that morphs smoothly between these two images. Of course, we would also like the features of the images (such as the facial profile, eyes, nose, etc...) to align at every frame of animation to create a convincing morph.
Figure 4.8: Morph between two people found on the left and right sides of the middle row. Deformation of the first image towards the second (top) and the second image towards the first (bottom). Blending these images results in the morph (middle).

To construct this morph, the user simply selects a set of handles $p$ in the first image $I$ and $\overline{p}$ in the second image $\overline{I}$ at various features (in this example, the handles correspond to line segments). Furthermore, there is a correspondence between these handles such that the feature associated with $p_i$ in $I$ is the same feature at $\overline{p}_i$ in $\overline{I}$. Now we create the morph by moving $p$ to $\overline{p}$. There are many ways to interpolate between these handles [58]; we simply linearly interpolate the positions at the end-points of the line segments.

Using this intermediate set of handles, we construct two different deformed images. The first deformed image is created by applying $f(v)$ to $I$ with undeformed handles $p_i$ and deformed handles $(1 - \gamma)p_i + \gamma\overline{p}_i$ where $\gamma \in [0, 1]$ controls the morph. The second deformed image is created by deforming $\overline{I}$ using undeformed handles $\overline{p}_i$ and deformed
handles \((1 - \gamma)p_i + \gamma\overline{p}_i\). Now we cross-fade between the two deformed images with the fade controlled by the parameter \(\gamma\).

Clearly when \(\gamma = 0\), the morphed image is \(I\) and when \(\gamma = 1\) the morphed image is \(\overline{I}\). Furthermore, the features specified at the \(p_i\) and \(\overline{p}_i\) align at each parameter \(\gamma\). These morphs may be produced using any deformation technique \([6, 4]\); however, we have already seen that using the appropriate types of transformations can lead to better deformations using fewer deformation handles. Fewer deformation handles translates into faster deformations and less work for the user.

Figure 4.8 shows the morphing process. In this example, we use the rigid-line Moving Least Squares deformation to give precise control over the outline of the face and hair. The top row shows \(\overline{I}\) deformed for various values of \(\gamma\) while the bottom row depicts the same deformations for \(I\). Blending these two deformation sequences together results in the morph in the middle row.

### 4.4.2 Implementation

To implement these deformations, we precompute as much information as possible for the deformation functions \(f(v)\). When we apply the deformation to an image, we typically do not apply \(f(v)\) to every pixel in the image. Instead we approximate the image with a grid and apply the deformation function to each vertex in the grid. We then fill the resulting quads using bilinear interpolation.

In practice, this approximation technique produces deformations indistinguishable from the more expensive process of applying the deformation to every pixel in the image. For all of the examples in this chapter, the images were approximately 500 \(\times\) 500 pixels. To compute the deformations, we used grids on the order of 100 \(\times\) 100 vertices. If desired, more accurate deformations may be achieved with denser grids and the deformation time is linear in the number of vertices of these grids.

Table 4.1 shows the amount of time taken to deform each of the images using various methods on a 3 GHz Intel machine. Each deformation uses a grid of size
100 × 100. The rigid transformations take the longest due to the square root in the deformation function. However, the deformations are still very fast and, even with 50+ line segments, the rigid deformations can be performed at over 60 frames per second.

4.5 Conclusions

We have provided a method for creating smooth deformations of images using either points or lines as handles to control the deformation. Using Moving Least Squares, we created deformations using affine, similarity and rigid transformations while providing closed-form expressions for each of these techniques. Though the least squares minimization with rigid transformations led to a non-linear minimization, we showed how these solutions could be computed directly from the closed-form deformation using similarity transformations thereby bypassing the non-linear minimization. We also illustrated how these deformation could be used to produce morphs between different images.

In the future, we would like to explore generalizing these deformation methods to 3D to deform surfaces. Such a generalization has potential applications in motion capture, where animation data can take the form of points in space for each frame.

\[
\begin{array}{|c|c|c|c|c|}
\hline
\text{Method} & \text{Figure 4.2} & \text{Figure 4.4} & \text{Figure 4.5} & \text{Figure 4.8} \\
& \text{(7 points)} & \text{(11 points)} & \text{(7 lines)} & \text{(52 lines)} \\
\hline
\text{Affine MLS} & 1.5 \text{ ms} & 2.2 \text{ ms} & 1.5 \text{ ms} & 6.9 \text{ ms} \\
\text{Similarity MLS} & 2.3 \text{ ms} & 3.4 \text{ ms} & 1.6 \text{ ms} & 7.8 \text{ ms} \\
\text{Rigid MLS} & 2.6 \text{ ms} & 3.8 \text{ ms} & 3.3 \text{ ms} & 15.9 \text{ ms} \\
\text{[6]} & 2 \text{ ms} & 2.7 \text{ ms} & \text{N/A} & \text{N/A} \\
\text{[4]} & \text{N/A} & \text{N/A} & 1.6\text{ms} & 7.5 \text{ ms} \\
\hline
\end{array}
\]

Table 4.1: Deformation times for the various methods.
of animation. However, the similarity transformation in section 4.1.2 leads not to a quadratic minimization, but to an eigenvector problem and we are looking into methods to efficiently compute the solution to this minimization.
Chapter 5

Conclusions and Future Work

Throughout the last three chapters, we have developed three different methods for creating deformations of images as well as 3D shapes. Each method has its own advantages and disadvantages. For instance, subdivision schemes require an internal tetrahedralization to create deformations. While this may seem like a disadvantage because the deformation depends on what tetrahedralization is chosen, interior vertices may be added to the triangulation to create additional internal controls for the deformation. Furthermore, subdivision schemes create a natural, multi-resolution hierarchy of deformations, which may be advantageous in some applications.

On the other hand, barycentric coordinates can construct deformations using only a polygon that approximates the shape of the object. Furthermore, the deformations produced by barycentric coordinates are $C^{\infty}$ away from the control polygon as opposed to the $C^2$ almost everywhere deformations that subdivision produces.

Finally, the Moving Least Squares deformations can create deformations without using any of the topology information present in the other two methods. However, fine level controls with point deformations may require large numbers of points due to the lack of topological information. Nevertheless, these point deformation methods allow us to easily change the types of deformations used from rigid to similarity to affine deformations.

The Moving Least Squares deformations actually hint at future avenues of research as well. Most previous method focus on creating deformations using different types of handles. However, the actual types of deformations desirable for different applications have been largely unexplored. We saw examples in the Moving Least Squares work
where more restrictive classes of deformations were actually beneficial. Extending these methods to surfaces to create a type of as-rigid-as-possible surface deformation method is an interesting area for future work.

Furthermore, we may want to enforce properties of these deformations other than the class of deformations the method can produce. For example, we may want to guarantee that the object that we deform has constant volume under deformation since most realistic objects do not gain/lose volume as they move.

Another useful property is local control. When the user modifies a deformation handle, it is desirable from the users perspective that only a portion of the surface deforms rather than the entire surface. The subdivision scheme in the second chapter has this local property. The barycentric coordinate method and the Moving Least Squares deformations have what is called almost local control in that the amount that the deformation handles influence the rest of the surface drops dramatically as the distance from that portion of the handle increases. When the user moves the hand of the armadillo man in the barycentric coordinate chapter, the foot moves. However, the movement is very slight and may not even be within the numerical precision that the computer can represent.

Finally, another important property of deformations is that they should not cause the surface to fold back on itself. If the user specifies that an arm penetrates through the body of the armadillo man, there is not much one can do. However, it is important that there are no local fold-backs that occur during typical deformations. Restricting fold-backs has been very difficult to build in practice and few methods can boast this property. One possible fix is the technique of Tiddeman et al. [50]. Even with this paper, constructing deformations that do not fold back is a very lucrative area for further work on deformations.
Bibliography


[34] Loop, C. Smooth subdivision surfaces based on triangles, 1987.


