

Mesh Modification Using Deformation Gradients

by

Robert Walker Sumner

Submitted to the Department of Electrical Engineering and Computer Science
on 15 December 2005, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Electrical Engineering and Computer Science

Abstract

Computer-generated character animation, where human or anthropomorphic characters are animated to tell a story, holds tremendous potential to enrich education, human communication, perception, and entertainment. However, current animation procedures rely on a time consuming and difficult process that requires both artistic talent and technical expertise. Despite the tremendous amount of artistry, skill, and time dedicated to the animation process, there are few techniques to help with reuse. Although individual aspects of animation are well explored, there is little work that extends beyond the boundaries of any one area. As a consequence, the same procedure must be followed for each new character without the opportunity to generalize or reuse technical components. This dissertation describes techniques that ease the animation process by offering opportunities for reuse and a more intuitive animation formulation. A differential specification of arbitrary deformation provides a general representation for adapting deformation to different shapes, computing semantic correspondence between two shapes, and extrapolating natural deformation from a finite set of examples.

Deformation transfer adds a general-purpose reuse mechanism to the animation pipeline by transferring any deformation of a source triangle mesh onto a different target mesh. The transfer system uses a correspondence algorithm to build a discrete many-to-many mapping between the source and target triangles that permits transfer between meshes of different topology. Results demonstrate retargeting both kinematic poses and non-rigid deformations, as well as transfer between characters of different topological and anatomical structure. Mesh-based inverse kinematics extends the idea of traditional skeleton-based inverse kinematics to meshes by allowing the user to pose a mesh via direct manipulation. The user indicates the class of meaningful deformations by supplying examples that can be created automatically with deformation transfer, sculpted, scanned, or produced by any other means. This technique is distinguished from traditional animation methods since it avoids the expensive character setup stage. It is distinguished from existing mesh editing algorithms since the user retains the freedom to specify the class of meaningful deformations. Results demonstrate an intuitive interface for posing meshes that requires only a small amount of user effort.

Thesis Supervisor: Jovan Popović

Title: Assistant Professor of Electrical Engineering and Computer Science

Acknowledgments

I wish to express sincere gratitude to my advisor, Jovan Popović, for his guidance over the past four years. His teaching, his enthusiasm, and his graciousness have been invaluable.

I'm indebted to my co-authors Matthias Zwicker and Craig Gotsman for their help with mesh-based inverse kinematics. This project was my first experience with collaborative research and it was a tremendously positive one. Our goals were not premeditated but born from a collective effort of four people meeting once a week to talk about research. These meetings were my favorite part of the week and I'm thrilled by the end result.

I am grateful for the suggestions that my committee members, William Freeman, Frédo Durand, and Michael Garland, offered about this dissertation. I'd like to thank Daniel Vlasic, Charles Han and Shuang You for their help with deformation transfer. Daniel was instrumental in an earlier version of the system. Thanks also to Sivan Toledo for his assistance with the numerical solution used by mesh-based inverse kinematics. I'm grateful for discussions with Mario Botsch and Mark Pauly that helped me to better understand the context of my work and identify a missing component in the numerical formulation. Eric Chan and Jiawen Chen also deserve thanks for patiently helping me with the code to draw a square and pick some points which I secretly had trouble writing on my own.

Many people have influenced me in this journey. I would like to think Julie Dorsey for advising me during my first years at MIT and supporting my in-depth study of lichens. She also supported my petition to add a soda fountain machine to the graphics lab, for which I am also thankful.

The three summers I spent working at Pixar Animation Studios influenced my graduate research greatly. Thanks to Tony DeRose and Dirk Van Gelder for introducing me to the articulation system used at Pixar and helping me to appreciate the scope of this aspect of animation. Brad Andalman, Sudeep Rangaswamy, Susan Fisher, Wayne Wooten, John Alex, and the others I worked with made the job feel more like a vacation than anything else. My great friends Tasha Harris and Wendell Lee showed me how real animators work and I'm still in awe of what they can do.

Jessica Hodgins invited me to join her research group as undergraduate, started me along the animation path, and still looks after me today. I attribute much of my academic ethics to her and greatly value the support she has offered over the years. Jessica and James O'Brien advised me on my first academic research of simulating sand, mud, and snow, or, as we called it, the "dirt" project. David Brogan, Nancy Pollard, Deborah Carlson, Victor Zordan, Wayne Wooten, Ron Metoyer, and the others in Jessica's Animation Lab taught me what to expect from graduate school and treated me like I was already there.

Cynthia Allen recognized potential in me after just a short meeting when I was an undergraduate and convinced Ken Perlin to invite me to spend a summer with the graphics group at NYU. I'd like to thank Cynthia for making it happen and Ken for supporting me then and in subsequent years. Ken, Cynthia, Athomas Goldberg, Jon Meyer, Clilly Castiglia, and the others in the NYU Media Research Lab showed me how graphics is done in New York City.

One of the most valuable aspects of graduate school has been the friendships I've formed within the MIT Graphics Group. My office mates over the years—Aseem Agrawala, Aaron Isaksen, and Rob Jagnow in the beginning, Sara Su in the middle, and John Alex, Tilke Judd, and Jingyi Yu at the

end—made each day an adventure. Mok Oh and Frédo Durand imparted some of their finesse at handling tricky situations. In recent years, Daniel Vlasic, Tom Buehler, and I, with much help from Paul Green, Jonathan Regan-Kelley, Yeuhi Abe, Tilke Judd, Bennett Rogers, Robert Wang, and Sara Su, developed a social force that has left a permanent mark on Cambridge. Everyone, including Barb Cutler, Justin Legakis, Matt Peters, Max Chen, Jan Kautz, Sylvain Paris, Matthias Zwicker, Soonmin Bae, Jiawen Chen, Eugene Hsu, Olivier Koch, Addy Ngan, Peter Sand, Kevin Der, Howard Chan, Wojciech Matusik, and Kari Pulli has contributed to make the MIT CGG one of which I am proud to be an Alumnus. Finally, Bryt Bradley, Adel Hanna, and Tom Buehler deserve special mention since they are not only friends but provided valuable services that kept the lab running.

Of course, friendships extend well beyond MIT. Andrew Elliott provided emotional support during much of my time in Cambridge. Annie Choi always reminded us how hip computer science is. Ana Jaklenec protected me from Daniel when he got rowdy. Eitan Grinspun provided professional advice and, together with Victor Zordan and Paul Kry, we went on several unprofessional adventures. Wilmot Li and Mira Dontcheva kept me both grounded and immensely entertained when I was out of town.

Finally, I'd like to thank my family for supporting me throughout this process, before it, and after. My mom Mary, my father Evans, and my brother Billy provide a foundation in my life on which I know I can always depend.

Contents

1	Introduction	II
2	Character-Animation Pipeline	17
2.1	Modeling	17
2.1.1	Geometric Representations	18
2.1.2	Modeling Tools	22
2.2	Rigging	29
2.3	Animation	35
2.3.1	Keyframe Animation	35
2.3.2	Motion Capture	39
2.3.3	Physics-Based Character Animation	41
3	Deformation Transfer	45
3.1	Deformation Representation	47
3.1.1	Displacement Fields	47
3.1.2	Deformation Gradients	49
3.1.3	Summary	56
3.2	Correspondence	56
3.3	Transfer	57
3.3.1	Integration	58

3.3.2	Optimization	59
3.3.3	Reconstruction Error	61
3.4	Numerics	62
3.4.1	Linearity	62
3.4.2	Solution	67
3.5	Analytic Derivation	70
3.6	Results and Discussion	72
3.6.1	Kinematic Poses	73
3.6.2	Non-rigid Deformations	74
3.6.3	Animation Retargeting	76
3.6.4	Dissimilar Characters	77
3.6.5	Detail-Dependent Deformations	77
4	Correspondence	81
4.1	Template Fitting	85
4.2	Triangle Pairing	91
5	Mesh-Based Inverse Kinematics	95
5.1	Principles of MESH IK	98
5.1.1	Feature Vectors	98
5.1.2	Linear Feature Space	99
5.1.3	Nonlinear Feature Space	101
5.2	Numerics	104
5.2.1	Gauss-Newton Algorithm	104
5.2.2	Cholesky Factorization	106
5.3	Results and Discussion	107
6	Conclusion	111
6.1	Contributions	111
6.2	Future Directions	112
7	Bibliography	117

List of Figures

1-1	The contributions in this dissertation	16
2-1	Barr-style deformations	24
2-2	Free-form deformation	24
2-3	Skeleton-subspace deformation	32
3-1	Overview of deformation transfer	46
3-2	The transfer problem is demonstrated on a bending line	48
3-3	Deformation gradient mapping	49
3-4	Boundary-based deformation gradients versus isomorphic dissection	55
3-5	Source deformation gradients are used to deform the target mesh	58
3-6	Linear system construction for identical topology	65
3-7	Linear system construction for different topologies	66
3-8	The nonzero structure of $A^T A$ for the lion mesh	69
3-9	Horse poses transfered to a camel	73
3-10	Cat poses retargeted onto a lion	73
3-11	Mismatched reference poses	74
3-12	Collapsing horse transferred to the camel	75
3-13	Transferred facial expressions	75
3-14	Galloping horse animation retargeted to the camel	75

3-15	Horse poses mapped onto a flamingo	78
3-16	Horse poses mapped onto an elephant	78
3-17	Self-intersections are not prevented by deformation transfer	78
3-18	Inflating a 2D shape	79
3-19	Inflating a 3D shape	79
4-1	Overview of the correspondence system	84
4-2	Grid-based spatial binning algorithm	87
4-3	Results from template fitting	89
4-4	Template-fitting as a stand-alone application	90
4-5	Visualization of the triangle pairings	91
4-6	Correspondence comparison with Kraevoy and Sheffer [2004]	92
5-1	A simple demonstration of MESH IK	96
5-2	Rotation correction	102
5-3	Three-way interpolation	104
5-4	Using MESH IK to pose a bar	108
5-5	Posing the lion mesh	108
5-6	Posing a simulated flag	108
5-7	Galloping horse and elephant animations	109
5-8	MESH IK solve time versus number of examples	110
6-1	A horse/tree transfer is ambiguous	114

Introduction

1

Animators bring fictional characters to life through movement. Their central task is to make a character act with personality and style, and thereby tell a story. The animation process begins in the modeling stage with the construction of a three-dimensional (3D) digital representation of a character's shape. The 3D model can be created with software tools or sculpted out of clay and scanned. The end result of modeling is a static shape that must be instrumented with so-called rigging controls to change its posture, bulge muscles, change facial expressions, and generate other necessary deformations. The rigging stage is critical since these controls determine the full range of deformation that will be seen in the final result. During the animation stage, the animator uses the rigging controls to create continuous movement by reposing the character over the course of the animation.

Although this process is effective, all aspects of it are challenging and labor intensive. The rigging procedure is perhaps the most expensive stage in the animation pipeline since it requires both artistic talent and technical expertise. Crafting the deformations required for lifelike movement is an artistic endeavor, while building the rigging controls to achieve these deformations is inherently technical. Once a rigging control has been designed, specializing it for a particular character's shape often involves time consuming parameter tuning in order to ensure that the generated deformations are acceptable for all control settings. The rigging process is tolerated because it is an essential component of the animation pipeline. It allows the user to parameterize the space of meaningful deformations so that the character can be animated efficiently.

Despite the tremendous amount of artistry, skill, and time dedicated to the animation process, there are few techniques to help with reuse. In order to reuse a deformation created for one shape on another, the specific parameters that control the deformation must be adapted to the new shape. Hand-sculpted alterations made during modeling have no inherent parameterization and are not easily adapted. For most rigging controls, re-tuning the parameters is just as time consuming as starting from scratch. Special-purpose transfer algorithms can adapt some forms of rigging and associated animation, but may fail in the common case where a variety of rigging techniques are used in tandem. As a result, the work spent modeling, rigging, and animating a character cannot be reused after its planned application.

Current research in character animation addresses individual aspects of the animation pipeline, but does not extend beyond the boundaries of any single stage. As a consequence, the global procedure remains unchanged. The modeling, rigging, and animation phases must be followed, in order, to create a character and make it move. With no general method of reuse, the entire process must be repeated from scratch for each new character. The rigid procedure for character animation and the absence of reuse algorithms are significant problems in computer animation addressed in this dissertation.

Challenges

Shape deformation plays an essential role throughout the animation pipeline. Editing tools employ deformation algorithms to sculpt a character's shape, rigging controls parameterize meaningful deformations, and animators use the rigging controls to create continuous deformations over time. The deformations employed by animation range from those that are primarily skeletal in nature (e.g., bending at the elbow) to ones that are non-rigid (e.g. facial expressions, cloth). To accommodate this wide range, a generic reuse mechanism must provide some way to represent arbitrary deformations without making domain-specific assumptions that would limit its applicability.

In order to effectively reuse the motion of one shape to deform another, deformations must be applied to semantically similar components: the legs of one character should deform like the legs of the other, the head like the head, the tail like the tail, and so on. To resolve all ambiguities, this association should continue to the smallest geometric entities. For example, when triangle meshes represent a character's shape, reuse should ensure that individual triangles are deformed appropriately.

Differences in mesh topology must also be considered in order to accommodate characters that have a different number of triangles, number of vertices, connectivity, or genus.

In order to escape the traditional animation pipeline, alternate methods are required to specify the meaningful deformations of a character and create animation using this specification. Example data demonstrating prototypical deformations is a convenient specification, but requires some technique to generalize from the given examples.

Contributions

The research I present in this dissertation eases the animation process by adding a general-purpose reuse mechanism to the animation pipeline and breaking the traditional modeling-rigging-animation sequence so that animators can skip the time-consuming rigging stage and specify the class of meaningful deformations through examples. A differential specification of arbitrary deformation provides a representation to transfer deformation between disparate meshes, compute semantic correspondence, and extrapolate natural deformations from a collection of examples. These topics are discussed below and summarized in Figure I-I.

Deformation Transfer. Deformation transfer retargets any deformation of a source character onto a different target character. Since this algorithm makes no assumptions about the method used to deform the source shape, it adds a general-purpose reuse mechanism to the animation pipeline. Deformations from hand-sculpted alterations made during modeling or individual poses produced with rigging controls can be reapplied to new characters. An animation sequence can be retargeted by applying the transfer algorithm frame-by-frame. This functionality allows an entire database of animations to be compiled and retargeted onto new characters when needed.

In order to support a completely general specification of deformation, my algorithm uses a representation based on the deformation gradient tensor field. Used in continuum mechanics, this quantity is a differential specification designed to represent large deformations. Since change in shape is most naturally specified as a differential quantity, deformation gradients are an appropriate choice for the transfer problem. A deformation gradient is the 3×3 Jacobian matrix of a global function that maps points from \mathbb{R}^3 to \mathbb{R}^3 . Since this function is defined for volumes and characters used in animation are often represented as triangulated surfaces, I propose a boundary-based approximation

of the deformation gradient tensor field on a triangle mesh. Each per-triangle deformation gradient encodes the change in orientation and stretch induced by the deformation on the triangle. Taken as a whole, the per-triangle deformation gradients can represent any mesh deformation regardless of its origin or complexity.

Deformation transfer employs a discrete correspondence, discussed below, that associates triangles of a source mesh with those of a different target mesh in order to indicate which triangles should deform similarly. This mapping makes the transfer algorithm broadly applicable since the source and target need not have the same topology or even anatomical structure. Per-triangle source deformation gradients encode the change in shape of the source mesh from a reference pose to a deformed pose. The transfer algorithm constructs these source deformation gradients, relates them to the target mesh via the correspondence map, and reconstructs a deformed target from this differential specification of deformation. The reconstruction process is expressed as a quadratic optimization problem so that the reconstructed target mesh will mimic the source deformation as closely as possible in a least-squares sense. The resulting normal equations are efficiently solved using sparse Cholesky factorization. After factoring, transferring a new deformation to the target mesh only requires performing the backsubstitution step.

Correspondence. A correspondence algorithm relates semantically similar mesh components to one another using a user-guided template-fitting procedure to find a partial parameterization of one mesh with respect to the other. The user controls the process through the specification of marker points at matching features on each mesh. The fitting procedure deforms one mesh to match the shape of the other. This deformation is formulated as an optimization problem using the same deformation gradients that enable deformation transfer. The algorithm is applied iteratively until a close match in shape is achieved. Since only a partial parameterization is required, the procedure can handle situations where a strict parameterization does not exist: the two meshes need not be of the same genus, and the algorithm is robust to topological “errors” where connectivity is improperly specified.

Once a close match in shape is found, a discrete correspondence map is extracted by searching for pairs of triangles whose centroids are in close proximity. This pairing indicates the parts of the two shapes that should deform similarly and provides a versatile specification of correspondence since

it supports a general many-to-many mapping. The generality enables transfer between meshes of different topology or even gross anatomical structure.

Mesh-Based Inverse Kinematics. Mesh-based inverse kinematics (MESH IK) extends the idea of traditional skeleton-based inverse kinematics to meshes by allowing the user to pose a mesh via direct manipulation. With MESH IK, the user avoids the time-consuming rigging stage and instead indicates the class of meaningful deformations using examples. The examples can be scanned, hand-sculpted, or designed with deformation tools from any stage of the animation pipeline. Because of the versatility with which examples can be created, MESH IK simplifies posing tasks even when traditional animation or editing methods do not apply. Furthermore, this work builds upon deformation transfer since the required example deformations can be automatically transferred from another mesh.

Once these examples are given, the user can select and drag any subset of mesh vertices to produce a meaningful change in shape. Although the user retains complete freedom to precisely specify the position of any vertex, most tasks only require moving a few. As a result, MESH IK achieves meaningful mesh deformations and pose changes in an intuitive manner with only a small amount of work by the user. The animator can pose an object by moving only a few vertices or bring it to life by keyframing these vertex positions. Furthermore, the user always retains the freedom to choose the class of meaningful deformations for any mesh.

In MESH IK, a feature vector of deformation gradients is computed for each user-given example deformation. A nonlinear span of these feature vectors defines a feature space of appropriate mesh deformations. When the user displaces a few mesh vertices, MESH IK positions the remaining ones to produce a mesh whose feature vector is as close as possible to the feature space. This procedure ensures that the reconstructed mesh meets the user’s constraints exactly while it best reproduces the example deformations.

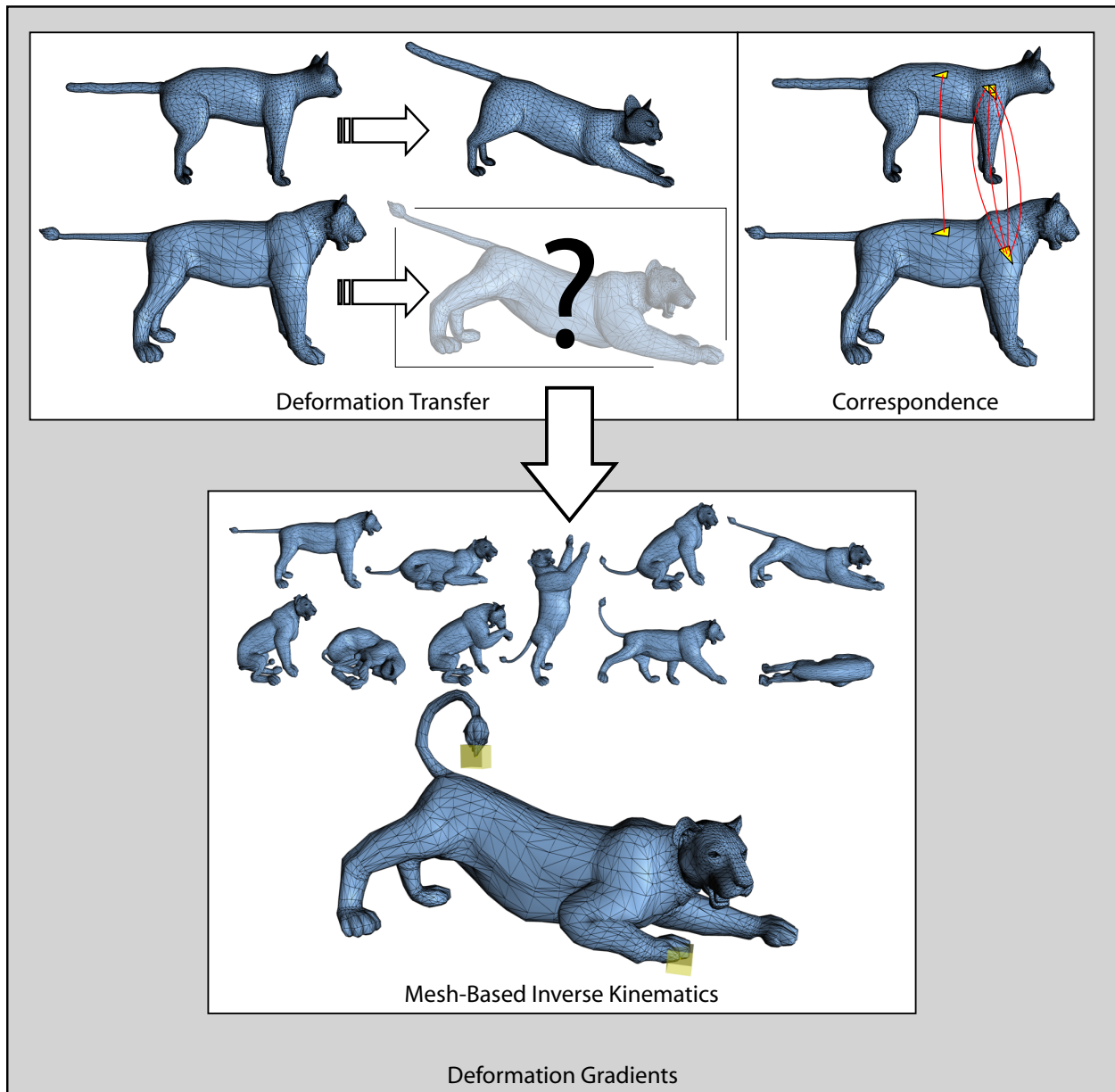


Figure 1-1: An overview of the contributions in this dissertation is depicted above. Deformation transfer adds a general-purpose reuse mechanism to the animation pipeline by transferring any deformation of a source triangle mesh onto a different target mesh. The transfer system uses a correspondence algorithm to build a discrete many-to-many mapping between the source and target triangles that permits transfer between meshes of different topology. Mesh-based inverse kinematics allows the animator to skip the time consuming rigging stage and instead specify the class of meaningful deformations through examples. The examples can be created automatically with deformation transfer, sculpted, scanned, or produced by any other means. Each of these algorithms employs the same differential specification of deformation based on the deformation gradient tensor field from continuum mechanics.

Character-Animation Pipeline

2

Character animation is divided into three stages—modeling, rigging, and animation—that must be followed, in order, to create a character and make it move. Animating a new character requires starting the process over at the beginning. The research presented in this dissertation aims to provide a degree of compatibility in the animation process that previously did not exist. My work allows disparate characters to be related to one another so that changes in one, whether hand sculpted alterations made during modeling, individual poses produced with rigging controls, or movement made by manipulating the controls over time, can be transferred to the other. I show how to break the modeling-rigging-animation sequence so that artists can leverage the ease with which a character’s shape is modeled and avoid the difficulties inherent in the rigging process.

Naturally, these contributions are best understood in the context of current animation procedures. This chapter presents the three primary stages in the character animation pipeline. Each stage entails a different set of goals and challenges. I describe the issues involved and discuss the substantial amount of related research in these areas.

2.1 Modeling

The first step in animation is generating a digital representation of the character to be animated. The shape creation process is variously called modeling, sculpting, or mesh editing, and a variety of

techniques exist to accomplish it. In computer animation, sculpting a character out of real clay or other materials is still a common technique since it gives artists a highly expressive medium with which they are well trained. The physical model of the character, called a “maquette,” is scanned to create a digital representation. While sculpting from real clay has been perfected over thousands of years, newer modeling techniques work directly with a digital representation. The format used to represent a character’s shape influences the type of modeling operations that can be performed. Thus, I first summarize common formats used to represent digital geometry and then describe modeling tools used to sculpt a character’s shape.

2.1.1 Geometric Representations

Digital representations of geometric objects are abundant. Since each representation has advantages and disadvantages, the choice of which one to use should be governed by the requirements of the application for which it is needed. **Triangle meshes** are one of the most common and widely used formats because of the ease with which triangle mesh data can be acquired using 3D scanners. A triangle mesh consists of a sequence of vertices $V = (\mathbf{v}_1, \mathbf{v}_2 \dots \mathbf{v}_n)$ and faces $F = (f_1, f_2 \dots f_m)$. Each vertex is a position in 3D space and each face is a sequence of three vertex indices indicating how the vertices are connected to form a triangle. The union of all triangles represents the surface of an object. The topology of the mesh refers to the mesh structure: the number of vertices and faces as well as their connectivity. In contrast, the shape of the mesh is determined by the positions of the vertices.

Meshes are often acquired through 3D scanning. While scanned meshes include fine details, they require a large amount of storage to do so. Since each face of a polygon mesh is planar, many faces are required to faithfully reproduce high-frequency surface detail. As a result, meshes with hundreds of thousands of triangles are commonplace. The large storage and complexity in terms of number of vertices and faces required to reproduce fine-scale features are disadvantages of triangle meshes.

One common topological consideration with respect to triangle meshes is whether or not a given mesh is a manifold. A manifold with boundary is a surface in which the neighborhood of every point is topologically equivalent to a disc, for interior points, or a half-disc, for boundary points. In order for a triangle mesh to be manifold, every interior edge must have exactly two incident triangles and every boundary edge must have exactly one. In addition, triangles which share a given vertex must form a

closed loop around that vertex, or a single fan for boundary vertices [Garland 1999]. Many algorithms require meshes that are manifold with boundary. Others require manifold meshes (with no boundary) which are called watertight. Still other algorithms restrict the genus, or, informally, the number of holes in the mesh. Unfortunately, triangle mesh data generated by scanning systems is notoriously “messy” and often non-manifold. Repairing such meshes is an active area of research (cf. [Ju 2004; Sharf et al. 2004; Nooruddin and Turk 2003; Liepa 2003]). However, the algorithms presented in this dissertation do not require manifold surfaces or place restrictions on the mesh genus.

Another consideration with meshes is parameterization. A parameterization is a mapping $P : \mathbb{R}^2 \rightarrow \mathbb{R}^3$ from 2D points in the plane to 3D points on the surface. Triangle meshes provide no explicit parameterization and computing one is non-trivial. Because of their importance in texture mapping and mesh processing, computing high-quality parameterizations has received much attention. For the purposes of my research, so-called cross-parameterization [Kraevoy and Sheffer 2004] or inter-surface mapping [Schreiner et al. 2004] is more important. These methods compute a parameterization of one triangle mesh with respect to another. This type of parameterization is discussed in Chapter 4 as it relates to mesh correspondence.

Although I consider triangle meshes to be the most appropriate representation for the presented research, it is important to consider other representations that are actively used in modeling and animation. Triangle meshes have two advantages over **polygon meshes**, in which each face is a planar polygon with an arbitrary number of vertices. First, every face in triangle mesh has the same number of vertices, which simplifies implementation. Second, each triangle is, by definition, planar, unlike in a general polygon mesh where faces may become non-planar due to programming or precision errors. On the other hand, some surfaces are better approximated by non-triangular faces [Dong et al. 2005; Cohen-Steiner et al. 2004].

A **parametric surface** is a mapping of the flat 2D plane to a curved 3D surface that takes the form:

$$r(u, v) = x(u, v)\hat{i} + y(u, v)\hat{j} + z(u, v)\hat{k}.$$

Thus, parametric surfaces admit a natural parameterization, which is one of their primary advantages. NURBS surfaces are a common parametric representation [Piegl 1991] that generalize B-spline curves to surfaces. NURBS have been widely used in computer-aided design applications since they can

succinctly represent many smooth shapes, are efficient to evaluate, and support a variety of geometric operations important to designers. Since a single NURBS surface, or “patch,” must be topologically equivalent to a sheet, cylinder, or torus, complex shapes are modeled by stitching together many patches. A collection of NURBS patches stitched together in this way is analogous to a polygon mesh where each patch represents a curved portion of the object. The primary drawback of NURBS surfaces is the difficulty in maintaining smoothness across patch boundaries. Creases where NURBS patches join are often unavoidable [DeRose et al. 1998]. Nonetheless, many animation tools employ NURBS patches to model the complex shapes of animated characters.

Subdivision surfaces provide the flexibility of polygon meshes yet still naturally represent smooth surfaces [DeRose et al. 1998]. A subdivision surface is comprised of a polygonal base mesh and a set of subdivision rules that indicate how the base mesh should be subdivided. Each step of subdivision divides the mesh polygons into finer ones by adding new vertices and displacing them. Iteratively applying the subdivision rules to the base mesh yields a mesh of increasing smoothness. Thus, a coarse base mesh with a small number of vertices can represent a smooth object in the limit of the subdivision procedure. Sharp features can be represented by adding special “crease” rules to the subdivision scheme that cause some mesh edges to be only partially subdivided.

Subdivision surfaces are flexible since the base mesh is a polygon mesh that can be created or modified using any mesh editing tools. The limit surface generated after repeated subdivision is smooth without the continuity problems that are common with NURBS representations. This flexibility and guaranteed smoothness makes them well suited for animation and has resulted in their widespread adoption in recent years [Zorin and Schröder 2000]. Like triangle meshes, subdivision surfaces give no explicit parameterization of the 2D plane. However, a parameterization of the limit surface to the base mesh is a natural consequence of the subdivision procedure. Since any triangle mesh can be used as a base mesh for subdivision, the algorithms in this dissertation are compatible with subdivision representations.

Point-based representations [Zwicker et al. 2002] use 3D points to represent a continuous surface. In essence, a point-based surface is a polygon mesh with the connectivity information removed and hence is sometimes called a “meshless” representation. Point-based surfaces simplify many geometric operations such as editing, deformation, surface completion, and resampling. However, they require more sophisticated algorithms for other tasks including normal estimation and surface extraction.

An **implicit surface** is defined as the isosurface of a 3D scalar function:

$$f(x, y, z) = \text{constant}.$$

One advantage of implicit surfaces is their ability to represent surfaces of complex and changing topology. Thus, they are a natural choice in situations such as fluid simulation where topological changes are common. When topological changes are not required, implicit surfaces may be a hindrance since preventing a change in topology can be difficult. Distance and inside/outside queries are efficient with implicit surfaces. Often, the implicit function represents the distance to the surface so that the distance from a point in space to the surface is computed by simply evaluating the function at that point. With other representations, distance queries are more difficult to implement efficiently.

I choose triangle meshes to represent geometric objects for the research presented in this dissertation over other options for several reasons. On the practical side, triangle meshes are, in my opinion, the simplest representation with which to work. They do not require complex data structures for storage, for display, or for other tasks such as normal estimation. Because of 3D scanners, mesh data is abundant. Triangle meshes provide the highest degree of compatibility since any other representation can be converted to a triangle mesh: polygon meshes and subdivision surfaces at any level of refinement can be triangulated [O'Rourke 2000], parametric surfaces can be sampled in parameter space to generate a triangle mesh at any resolution, implicit surfaces can be triangulated using efficient polygonization algorithms [Lorensen and Cline 1987; Bloomenthal 1994], and triangulated surfaces can be extracted from point-based representations [Curless and Levoy 1996]. Due to this high level of compatibility, they are well supported by commercial software which facilitates the generation of example meshes with which to test the presented algorithms.

From a technical standpoint, the algorithms in my dissertation deal with the deformation of geometric objects. Since each triangle is a piecewise linear approximation of the object's shape, it is possible to extract the effect of the deformation on a single triangle as an affine transformation. In contrast, a single affine transformation may not be able to capture the change that a general polygon or entire NURBS primitive undergoes during deformation. At the other extreme, point-based representations provide too little information to compute such a transformation for each point. With

implicit surfaces, the concept of deformation is ill-defined since there is no clear correspondence between surfaces extracted from different implicit functions.

On the other hand, some aspects of using triangle meshes are cumbersome. For example, the correspondence algorithm in Chapter 4 relies on a compatible-closest-point computation. Efficient implementation of this query with triangle meshes requires precomputing a spatial-binning data structure and performing a complicated breadth-first search for each query. Furthermore, the efficiency of the deformation transfer (Chapter 3), correspondence (Chapter 4), and mesh-based inverse kinematics (Chapter 5) algorithms depends on the number of vertices that comprise the mesh. Performance suffers with densely sampled meshes.

2.1.2 Modeling Tools

Once the choice of *how* to represent geometric objects has been made, the actual shape of the character must be sculpted. This process may involve creating the character’s shape from “scratch” (sometimes called *ab initio* design [Zorin et al. 1997; Kobbelt et al. 1999a]) or modifying an existing version created earlier or acquired by a scanner.

When building a shape from scratch, tools focus on the creation of smooth surfaces. For example, geometric primitives such as spheres and cylinders can be combined to make more complex shapes, NURBS patches can be stitched together to form a smooth surface, and variational techniques yield smooth surfaces that interpolate user-specified control points [Welch and Witkin 1992; Sorkine and Cohen-Or 2004]. Teddy, one of the most well known research efforts exploring novel interfaces for *ab initio* design, “inflates” hand-drawn sketches to create smooth 3D shapes [Igarashi et al. 1999]. Early work in modeling focuses on space deformations that generate large-scale changes by modifying only a few parameters. Space deformations merge well with the shape design process as they easily accomplish scales, twists, bends and other modifications that are required when sculpting from scratch.

Because of the increasing prevalence of mesh representations, recent research focuses on sculpting tools for meshes that provide more flexibility than space deformations. Editing a scanned mesh entails different requirements than *ab initio* design. Meshes acquired with 3D scanners are extremely detailed and may contain hundreds of thousands or even millions of vertices. When editing these meshes, the focus shifts from creating smooth shapes to detail preservation: low-frequency changes to the mesh should preserve the high-frequency details. Detail preservation is a necessary consequence of the

complexity of scanned meshes. Generating a broad change in shape by moving every vertex individually would be tedious and inefficient. Mesh editing tools allow edits at different frequencies to be created more efficiently. Two major categories of detail-preserving editing techniques are multiresolution methods and differential representations. In this section I first discuss space deformations and then these two mesh editing techniques.

Space Deformations

Space deformations deform solid or surface primitives by remapping the space in which the primitives are embedded. They can be applied to any of the geometric representations defined in Section 2.1.1. In 3D, a space deformation is defined by a global function $\mathbf{U} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ where

$$\mathbf{U}(\mathbf{p}) = \mathbf{U}(p_1, p_2, p_3) = \begin{bmatrix} U_1(p_1, p_2, p_3) \\ U_2(p_1, p_2, p_3) \\ U_3(p_1, p_2, p_3) \end{bmatrix}.$$

Barr [1984] is the first to use this type of function as a modeling tool. He refers to \mathbf{U} as a “globally specified deformation” and proposes several examples including functions for twisting, bending, and tapering. Barr demonstrates how to construct a chair using six primitives and seven bends. These deformations are still in use today and are incorporated into present-day modeling and animation software as so-called nonlinear deformers [Alias 2005]. Figure 2-1 shows several examples of these deformations.

Barr also defines a “locally specified deformation” to be the 3×3 Jacobian matrix of \mathbf{U} :

$$\mathbf{J} = \frac{\partial \mathbf{U}}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial U_1}{\partial p_1} & \frac{\partial U_1}{\partial p_2} & \frac{\partial U_1}{\partial p_3} \\ \frac{\partial U_2}{\partial p_1} & \frac{\partial U_2}{\partial p_2} & \frac{\partial U_2}{\partial p_3} \\ \frac{\partial U_3}{\partial p_1} & \frac{\partial U_3}{\partial p_2} & \frac{\partial U_3}{\partial p_3} \end{bmatrix}. \quad (2.1)$$

The matrix \mathbf{J} indicates how differential vectors are transformed by the function \mathbf{U} . If some surface is embedded in the space operated on by \mathbf{U} and the vector \mathbf{t} is tangent to the surface before deformation, then the vector $\mathbf{J}\mathbf{t}$ will be tangent to the surface after deformation. If the vector \mathbf{n} is normal to the surface before deformation then $|\mathbf{J}|\mathbf{J}^{-1\top}\mathbf{n}$ is normal to the surface after deformation, where $|\cdot|$

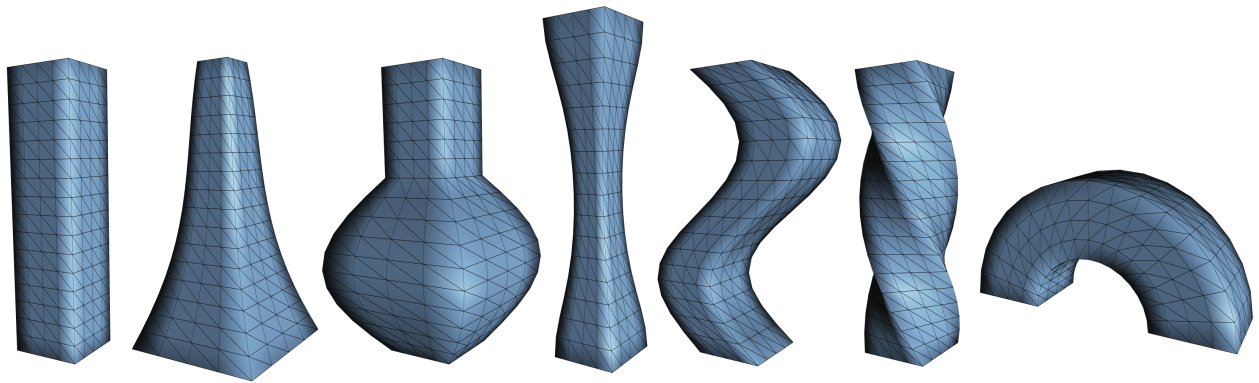


Figure 2-1: A box (*far left*) is deformed by several Barr-style deformations.

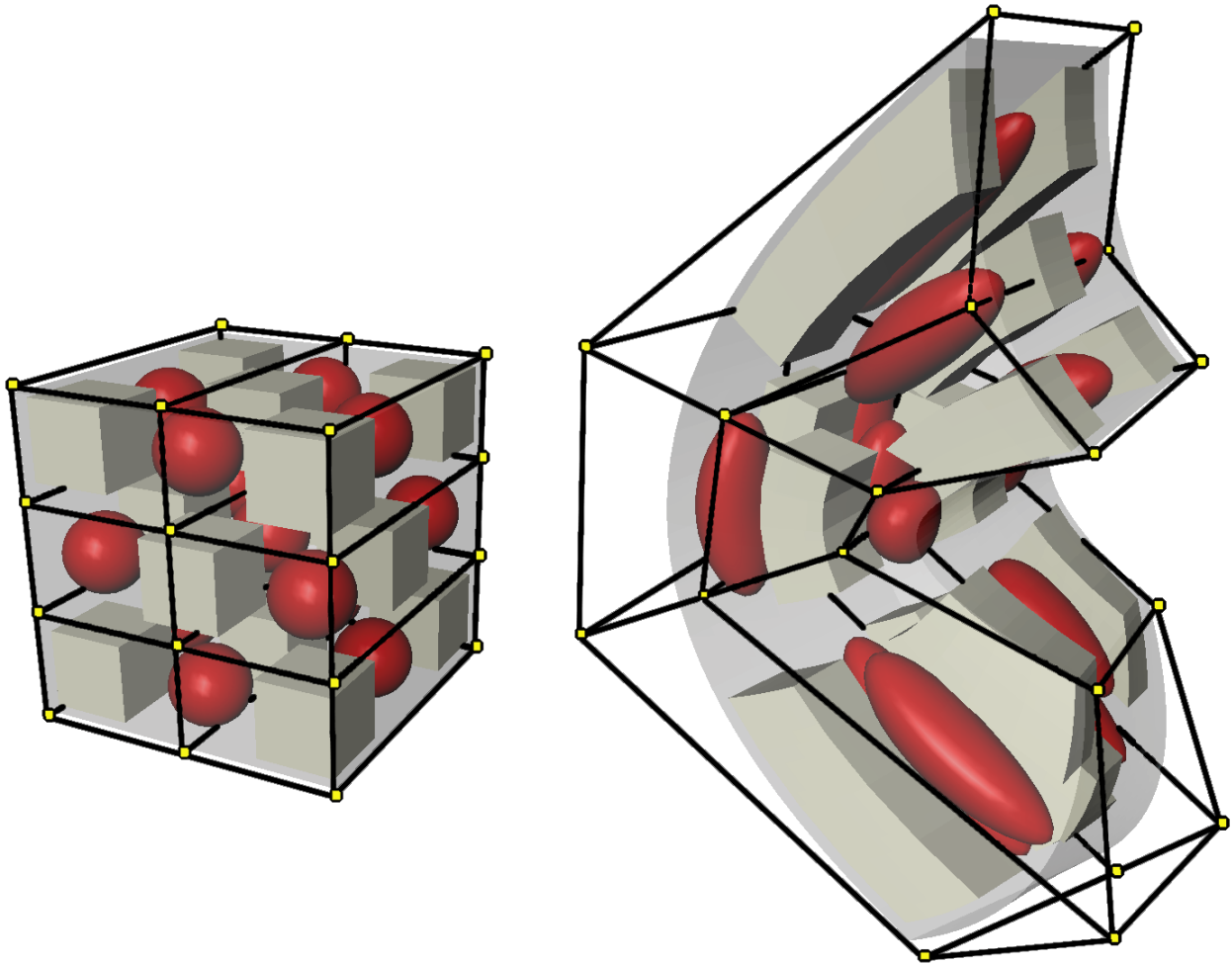


Figure 2-2: Manipulating the vertices of a free-form deformation lattice induces a deformation on the enclosed space.

indicates the determinant. Barr also defines a procedure to convert a locally specified deformation of a primitive back to a global specification via integration. Starting from some arbitrary origin (the constant of integration), the differential changes are integrated across the primitive to find the global deformed positions. This procedure requires J to actually be the Jacobian matrix of some space deformation. As Barr notes, his description of deformation is completely general. For this reason, it forms the basis of the deformation representation I use for the algorithms presented in this dissertation.

Although Barr's global and local deformations are well suited for algebraic operations such as bending, twisting, and tapering, his method does not provide an interface for more general sculpting operations. Free-form deformation (FFD) is an alternate space deformation technique with a long history in graphics. In the original formulation of Sederberg and Parry [1986], FFD is a mapping from \mathbb{R}^3 to \mathbb{R}^3 determined by a trivariate tensor product Bernstein polynomial. A 3D parallelepiped lattice forms the control points of the polynomial so that deforming the lattice points induces a deformation on the enclosed space. Figure 2-2 shows an example. An important property of FFD is that the generated deformations are independent of the complexity of the object being deformed. Sederberg and Parry easily sculpt a rod into the shape of a telephone handset using three FFD lattices.

FFD has been extended in many ways. Greissmair and Purgathofer [1989] extend the original formulation to trivariate B-Splines. Coquillart [1990] provide an extended set of lattices while MacCracken and Joy [1996] allow lattices of arbitrary topology. Hsu [1992] allows the user to manipulate the embedded object directly rather than indirectly via the lattice control points. The change in control point positions is solved for automatically to match the surface point change.

The WIRES framework of Singh and Fiume [1998] borrows the idea of deformations controlled by polynomial curves from the FFD setting (where the curves are encoded as trivariate tensor products) but removes the tensor setting. Instead, object geometry is bound to curves drawn on its surface. Subsequent manipulation of the curves induces a space deformation that is applied to the geometry according to a distance function.

Multiresolution Mesh Editing

Multiresolution editing techniques allow the user to edit highly detailed mesh representations. As opposed to space deformations, multiresolution methods address mesh detail directly: the user should be able to decide at which scale to alter a mesh. Details encoded in higher frequency bands should

be preserved. Multiresolution methods achieve detail-preserving edits at varying scales by generating a hierarchy of simplified meshes together with corresponding detail coefficients. When the user changes the mesh at a coarse resolution, finer scale details are preserved. The original formulation by Lounsbery, DeRose, and Warren [1997] extends multiresolution analysis based on wavelets to polygon meshes in order to achieve not only multiresolution editing but also compression, continuous level-of-detail, and progressive display/transmission. This representation consists of a base mesh together with a sequence of detail coefficients that indicate how to recover the input mesh from repeated subdivision of the base. This scheme suffers from two primary problems. First, since it is based on subdivision, the original mesh must have so-called subdivision connectivity or, equivalently, be semi-regular: each sub-region of the input mesh that corresponds to a single face of the base mesh must have the same connectivity that results from repeated subdivision of the face. In order to edit arbitrary meshes, researchers have proposed remeshing strategies to enforce the required connectivity (cf. [Eck et al. 1995; Lee et al. 1998; Kobbelt et al. 1999b; Boier-Martin et al. 2004]). However, remeshing is sometimes undesired as it yields only an approximation of the original geometry. Second, once the desired connectivity has been achieved, low frequency edits are restricted to the vertices of the simplified mesh at the proper resolution for that frequency. However, these vertices may not align with features that the user wants to change.

To address these problems, Kobbelt and colleagues [Kobbelt et al. 1998; Kobbelt et al. 1999a] propose geometric simplification (i.e., smoothing) as opposed to topological simplification. Discrete fairing [Kobbelt 1997] is used to create a smoothed version of the mesh with the same topological structure. Detail coefficients store vertex displacements between the smoothed and original surface. The authors propose an editing interface which has been used by many subsequent applications. The user selects an arbitrary support region on the mesh, or region of interest (ROI). The support region will be modified by the user while the rest of the mesh remains unchanged. Next, the user marks a handle region within the support. The handle can be manipulated by the user by applying translations, rotations, or other transformations. When the user selects the support region, it is smoothed subject to continuity constraints with the handle and the remainder of the mesh. Detail coefficients are then computed. As the handle is moved, the smoothed surface is updated and the details are added to reconstruct the edited shape. Botsch and Kobbelt [2004] extend this method to allow continuous control of the mesh continuity at the handle and support region borders.

The idea of using vertex displacements to store mesh details is well explored. Zorin and colleagues [1997] extend the original multiresolution method of Lounsbery, DeRose, and Warren [1997] to use detail coefficients based on displacements rather than wavelets for meshes with subdivision connectivity. Guskov, Sweldens, and Schröder [1999] remove this restriction in their work on multiresolution signal processing. Guskov and colleagues [2000] and Lee, Moreton, and Hoppe [2000] develop mesh representations based on displacement vectors. Kobbelt, Bareuther, and Seidel [2000] remove the restriction of any hierarchical structure linking different levels of detail in a multiresolution framework: each level can have an arbitrary vertex connectivity. Detail information is found by casting rays from vertices in one level to the next to compute normal offsets. By providing a parameterization [Kraevoy and Sheffer 2004; Schreiner et al. 2004] of one base mesh with respect to a different one (with a different geometry and/or connectivity), the details can be transferred to the new mesh. This form of transfer precipitates the expression transfer work of Noh and Neuhmann [Noh and Neumann 2001] which uses a parameterization to transfer vertex displacement vectors from one face mesh onto another. The length and orientation of the vectors are corrected using heuristics based on the local vertex neighborhood.

Differential Representations for Mesh Editing

The modeling tools discussed so far focus on the Cartesian representation of geometry: a shape is defined by the Cartesian positions of its vertices or control points. Since the goal of many mesh editing applications is detail preservation, it is advantageous to represent a shape in terms of these details. Differential representations store information about the local shape properties of a mesh, such as curvature, scale, and orientation. By representing a mesh in terms of these details, editing operators can be developed that strive to preserve them.

Perhaps the most widely used differential representation is that of Laplacian coordinates, which are also known as differential coordinates or δ -coordinates. (For detailed information, see Sorkine's recent survey [2005].) Laplacian coordinates were first used by Alexa for morphing [2001; 2002b; 2003b] and by Lipman and colleagues [2004] for mesh editing. In this framework, a mesh is represented as the difference of each vertex and a weighted sum of its neighbors. If the weights are chosen to be the so-called cotangent weights [Meyer et al. 2003] then each Laplacian coordinate approximates the surface normal and mean curvature. The linear operator L which extracts the Laplacian coordinates from the

Cartesian representation of a mesh is a square matrix with one row and column for each vertex that discretizes the Laplace-Beltrami operator for triangulated 2-manifolds [do Carmo 1976]. Extracting the Laplacian coordinates amounts to applying the linear operator (i.e., matrix multiplication); converting back to Cartesian coordinates involves inverting L . Mesh editing is achieved by fixing some vertices as constraints controlled by the user when reconstructing the Cartesian positions. This method is efficient since the resulting normal equations need to be factored only once, after which the factorization can be reused to reconstruct the edited surface via backsubstitution.

A persistent problem with the Laplacian representation is its lack of rotation invariance. The reconstruction procedure strives to preserve the global orientation of the Laplacian coordinates, and, therefore, to preserve the orientation of local features. As a result, local features experience shearing in an attempt to maintain their global orientation when the mesh is reconstructed. More natural deformations result from rotating the features. Indeed, the multiresolution methods discussed previously achieve natural deformations by computing detail coefficients in local frames that rotate as the surface deforms. In Laplacian editing, Lipman and colleagues [2004] solve the Laplacian system and then smooth the result to obtain estimates of the changes in surface orientation. Then, they explicitly rotate the Laplacian coordinates by these amounts and re-solve to obtain the final result. Sorkine and colleagues [Sorkine et al. 2004; Lipman et al. 2005a] find the optimal transformation of each Laplacian coordinate which requires linearization of the rotation constraint in order to maintain a linear reconstruction procedure. Yu and colleagues [2004] extend Poisson-based gradient field manipulation techniques which have proven successful for image manipulation [Fattal et al. 2002; Pérez et al. 2003; Agarwala et al. 2004] to mesh editing. As in the image manipulation algorithms, Yu and colleagues solve a problem of the form

$$\nabla^2 f = \nabla \cdot \mathbf{w}, \quad (2.2)$$

subject to Dirichlet boundary conditions. In this equation, $\nabla^2 f = \frac{\partial^2 f}{\partial x_1^2} + \frac{\partial^2 f}{\partial x_2^2} + \frac{\partial^2 f}{\partial x_3^2}$ is the Laplacian of f with respect to the three coordinates x_1 , x_2 , and x_3 , and $\nabla \cdot \mathbf{w} = \frac{\partial w_1}{\partial x_1} + \frac{\partial w_2}{\partial x_2} + \frac{\partial w_3}{\partial x_3}$ is the divergence of the vector field \mathbf{w} defined on the mesh. The unknown f represents one of the three scalar coordinate functions defined on the mesh. Discretizing this partial differential equation for triangle meshes leads to the same linear operator L , and solving it finds the mesh deformed according to the guidance field \mathbf{w} [Sorkine 2005]. The advantage of the Poisson framework is that the guidance field \mathbf{w} may be easier to specify than devising a way to transform the Laplacian coordinates directly.

Yu and colleagues [2004] compute the guidance field by using geodesics to propagate transformations induced by handle vertices, while Zayer and colleagues [2005] propose a harmonic field for computing the guidance field more naturally.

Other researchers have searched for rotation invariant mesh representations. The *pyramid coordinates* of Sheffer and Kraevoy [2004] are invariant under rigid transformations. In this representation, only lengths and angles are used to represent the mesh, giving it the desired invariance. However, the algorithm to reconstruct the Cartesian representation from the pyramid coordinates is nonlinear and, in their implementation, too slow for interactive mesh editing. Lipman and colleagues [2005b] present a mesh representation invariant to rigid transformations based on the first and second fundamental forms discretized for triangle meshes. This method also stores only lengths and angles. However, their reconstruction algorithm requires only linear solves. First, a linear system is solved to find the local frames at each vertex, and then a different system is solved to reconstruct the Cartesian coordinates of the vertices from the frames. The method of Lipman and colleagues [2005b] as well as the Poisson framework [Xu et al. 2005] also permit shape interpolation.

Other advancements in mesh editing address more sophisticated modeling metaphors. Nealen and colleagues [2005] provide a sketch-based interface. The user marks the support region and then draws a screen space curve that indicates the desired change in silhouette. Soft positional constraints are derived from this curve and used to solve a Laplacian system for the deformed mesh. Llamas and colleagues [2003] present a two-handed interface for deriving 3D space deformations, while Igarashi and colleagues [2005b] develop a technique for deforming 2D shapes using a multiple-point input device.

2.2 Rigging

After the shape of a character has been designed, it must be instrumented with a set of controls that allow the animator to deform the character into different poses. The modeling techniques discussed in the previous section are not appropriate for animation since the numerical criteria employed by modeling tools does not encode the animator’s high-level knowledge about how a character should deform. For example, when manipulating a character’s leg, it should bend only at the hip, knee, and ankle—not in an arbitrary location. The restricted set of *meaningful* deformations determines the mesh kinematics, or how the mesh vertices are allowed to move. The kinematics includes the

animator’s semantic knowledge about which deformations are appropriate for the mesh in question. Since modeling tools do not address mesh kinematics, a process called *rigging* is used instead.

During rigging, a character’s shape is augmented with a set of controls that approximate the character’s kinematics. These controls include both gross skeletal changes that determine the character’s posture as well as more subtle deformations such as muscle bulging and facial expressions. The rigging controls are like the strings of a marionette: they are used by the animator to make the character move. The rigging process is one of the most important but also expensive steps in the production pipeline. These controls determine the final shape that the viewer will see. Thus, the quality of the deformations generated by the character’s rigging has a tremendous influence on the quality of the final animation. Furthermore, since rigging is solely responsible for deforming the character’s vertices, every nuance of expression that the animator requires to convey the story—from the bending of joints to subtle facial motion such as furrowing the brow—must be captured by the rigging controls.

The most common form of rigging is skeleton-subspace deformation (SSD), which is sometimes referred to as “skinning” or “enveloping.” This algorithm, although unpublished, is included in almost all animation software packages and used, in some form, in most character animation generated for film, television, and video games. Weber [2000] gives an excellent practical overview of SSD. As the name implies, SSD addresses skeletal deformation and begins with the construction of a kinematic skeleton. A skeleton has the topological structure of a tree and consists of a collection of nodes (“joints”) connected by edges (“bones”) that approximates the character’s true anatomical skeleton. The skeleton is usually built manually by the artist, although some automatic techniques exist [Wade and Parent 2002; Katz and Tal 2003; Liu et al. 2003; Thorne et al. 2004]. The bones of the skeleton typically have a fixed length and the joints approximate real joints with either one (e.g., elbow, knee), two (e.g., wrist, ankle), or three (e.g., neck, shoulder, hip, waist) rotational degrees of freedom. The skeleton can be posed via forward kinematics in which values for the joint rotations are selected manually, or via inverse kinematics in which positions for the end effectors are specified and the joint angles are found automatically [Zhao and Badler 1994; Grochow et al. 2004].

In order to deform a character’s mesh using SSD, the skeleton is first placed in a *bind pose* which matches the kinematic configuration of the mesh. Then, the mesh vertices are associated with the joints of the skeleton through a collection of vertex weights. These weights indicate how the position of each vertex should be influenced by the posture of the skeleton. A posed vertex position is given by

a weighted sum of its position in the local frame of each joint:

$$\tilde{\mathbf{v}}_i = \sum_{j=1}^{|\mathcal{J}|} w_{i,j} \mathbf{M}_j \mathbf{B}_j \mathbf{v}_i, \quad (2.3)$$

where $\tilde{\mathbf{v}}_i$ is the posed position of vertex i , \mathbf{v}_i is its unposed position, \mathcal{J} represents the set of skeletal joint frames, \mathbf{M}_j is the concatenation of joint frame transformations for the posed skeleton from the root to joint j , and \mathbf{B}_j is a matrix that expresses the undeformed vertex in the frame of the joint’s bind pose. The parameter $w_{i,j}$ indicates how vertex i is influence by joint j . The weights for a given vertex are typically restricted to be positive and to sum to one. The initial weighting may be calculated using distance heuristics, and software packages often provide a “painting” interface for additional tuning [Alias 2005]. If these weights are carefully selected, a gradual bend will occur at the joints.

SSD formalizes the space deformations of Barr [I984] (Section 2.1.2) in a way that makes them appropriate for kinematic animation. Barr’s formulation includes global bends parameterized by bending angle and bending rate. SSD computes a bending transformation for each joint parameterized by angle (but not rate). The global nature of Barr’s deformations, which becomes unmanageable for kinematic animation, is traded for a hierarchical structure and explicit weighting. The hierarchical structure imposed by SSD allows changes in joints close to the root to automatically influence the extremities. The weighting allows the user to manage which parts of the mesh are influenced by which transformations.

The main advantage of SSD is speed. In most situations, a vertex will have nonzero weights for at most four joints. Thus, only a few matrix-vector multiplications are required to compute each posed vertex position. Furthermore, this algorithm can be implemented entirely on commodity graphics hardware [Lindholm et al. 2001; Fernando and Kilgard 2003]. While SSD is simple and fast, it is also well known for artifacts such as the “collapsing elbow” (Figure 2-3). Unfortunately, the user may struggle at length to remove these artifacts and never arrive at an acceptable result. No amount of weight tuning can solve these problems since, in many cases, the desired deformation does not lie in the space of deformations spanned by the SSD algorithm [Lewis et al. 2000]. This space is unclear and not easily explored since the user only has indirect control over the mesh shape via the weights. In response, Mohr, Tokheim, and Gleicher [2003] make the tuning process easier by visualizing the space of possible deformations and allowing the user to directly manipulate vertices within this space.

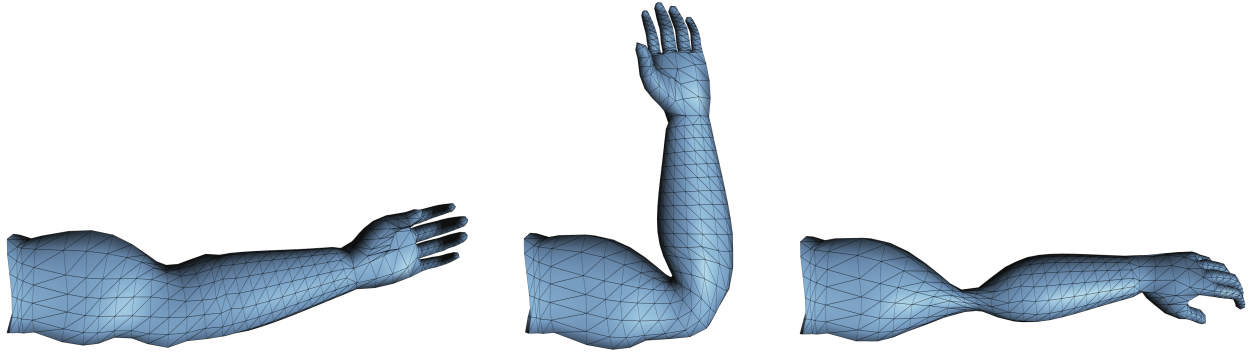


Figure 2-3: An arm mesh (*Left*) is posed using skeleton-subspace deformation. SSD performs the gross skeletal deformation but suffers from collapsing artifacts near the elbow when the arm is bent (*Middle*) which become more noticeable when the arm is twisted (*Right*).

In order to generate deformations outside the space of those spanned by SSD, Lewis and colleagues [2000] and Sloan and colleagues [2001] develop a hybrid method that combines traditional SSD with shape interpolation, allowing the user to sculpt corrections to the character’s shape at arbitrary kinematic poses. During character setup, the artist adjusts the joint parameters to pose the mesh using naïve SSD. Then the shape of the mesh can be altered using any mesh modeling tools to repair artifacts like the collapsing elbow or to add more details such as muscle bulges. These alterations are stored and associated with the character’s kinematic pose. During runtime, the hand-sculpted changes are interpolated using radial basis functions based on the kinematic configuration of the skeleton. EigenSkin [Kry et al. 2002] finds a reduced basis for the alterations that can be evaluated on graphics hardware. While not using SSD per se, Allen, Curless, and Popović [2002] augment a skeletal model with displacements derived from range scans of a human torso while Sand, McMillan, and Popović [2003] compute displacements from silhouette information taken from video of an actor. These hybrid techniques use the skeleton to encapsulate the mesh rotations and add linear offsets in the rotated frames. In this way, they address similar issues as mesh editing algorithms [Lipman et al. 2004; Sorkine et al. 2004; Lipman et al. 2005a; Sorkine 2005] which must find some way to compute rotations so that features are transformed in a natural fashion.

The quality of SSD can also be improved by adding additional parameters to the model which are automatically tuned to best reproduce user-supplied examples. With multi-weight enveloping, Wang and Phillips [2002] extend traditional SSD by allowing each vertex to weigh each entry in each joint frame matrix independently. Thus, every vertex has twelve weights per joint rather than one.

They show how to solve for these weights using a least-squares fitting procedure so that the resulting deformations approximate a user-provided training set. Mohr and Gleicher [2003] address a similar problem. They use traditional SSD as their articulation model but augment the skeleton with additional heuristically chosen joints, as suggested by Weber [2000]. Then, they use a user-provided training set to solve for both the vertex weights and the bind pose positions. Other methods provide different interpolation schemes in order to reduce artifacts [Kavan and Žára 2003; Kavan and Žára 2005].

As an alternative to SSD, researchers generalize free-form deformation (Section 2.1.2) to address rigging and animation problems. Chadwick, Haumann, and Parent [1989] use a skeleton to influence the control points of a FFD lattice in order to induce a deformation on a character’s mesh. This layered approach in which the FFD lattice loosely represents muscle and fatty tissue precipitates the more involved anatomical models discussed below. Singh and Kokkevis [2000] develop a hybrid deformation algorithm that bears resemblance to both FFD and multiresolution modeling. A deformer object is computed as a simplified approximation of a character’s mesh. The mesh vertices are bound to the deformer object based on Euclidean distance, after which a change in the shape of the deformer is propagated to the mesh.

Another important class of animation techniques extends FFD with physical simulation to generate dynamic deformations. Faloutsos, van de Panne, and Terzopoulos [1997] provide a dynamic generalization of traditional FFD [Sederberg and Parry 1986]. Capell and colleagues [2002b] apply the finite element method (FEM) to a control lattice defined using volumetric subdivision [MacCracken and Joy 1996]. The same authors [Capell et al. 2002a] apply this technique to skeletal-driven deformation by using a skeleton to infer constraints in the FEM simulation. This allows traditional skeletal animation to generate dynamic effects. Capell and colleagues [2005] extend their work beyond skeletal deformations to a complete physically-based rigging system.

While procedural techniques such as SSD compute approximate deformations very quickly, more realistic skin deformations can be generated by modeling the anatomical structures underneath it at a much higher computational cost. Anatomical models of the human head for facial animation have an extensive history. Kähler [2003] gives a historical review of this literature as part of his dissertation on the subject. For the human body, Chen and Zeltzer [1992] develop a physically-based model of muscles and compare their simulations to real muscle experiments. Wilhelms and Van Gelder [Wilhelms and Gelder 1997; Wilhelms 1997] present an anatomically-based model of animals that includes bones,

muscles, and tissue. Scheepers and colleagues [1997] develop a muscle model of the human arm and torso. Teran and colleagues [2003] focus on numerical aspects of muscle simulation in order to improve computational efficiency. The actual geometric shape of the muscles and other anatomical structures can be created by hand [Scheepers et al. 1997; Aubel and Thalmann 2001], constructed semi-automatically to conform to the 3D shape of a mesh [Wilhelms and Gelder 1997; Wilhelms 1997; Pratscher et al. 2005], or built from medical data such as MRI scans [Chen and Zeltzer 1992] or the visible-human data set [Hong Zhu et al. 1988; Hirota et al. 2001; Teran et al. 2005].

Some rigging methods rely completely on shape interpolation. In blend-shape animation, motion is generated by varying the blending weights in the linear combination of a collection of example meshes in topological correspondence. This technique is a standard approach for facial animation that has been used for twenty years [Lewis et al. 2005]. Its compatibility with modern graphics hardware helps to maintain its popularity today. The success of blend shapes for facial animation as well as methods that build linear [Blaž and Vetter 1999; Zhang et al. 2004] or multilinear [Vlasic et al. 2005] models of facial expressions suggests that facial deformations are primarily linear in nature. Linear models of full body shape variation are also successful [Allen et al. 2003] but require a more complicated representation in order to accommodate kinematic deformations [Seo et al. 2003; Anguelov et al. 2005]. One important property of blend shape animation is that it can be retargeted onto a new character by replacing each shape with a corresponding example of the new character in the same pose or making the same expression. For example, Bregler and colleagues [2002] develop a system to capture 2D cartoon sequences using blend shape animation. Retargeting the animation onto a different character requires replacing each pose of the blend shape model with a drawing of the new character. They also demonstrate that kinematic deformations of 2D graphics can be reproduced using blend shapes if the space is densely sampled. However, when applying their method to 3D characters, they opt for a skeletal representation.

Ngo and colleagues [2000] present a formalism of the concept of rigging. They note that the configuration space of a shape—the space spanned by its degrees of freedom (e.g., mesh vertices)—contains nonsense shapes in much higher proportion than meaningful ones. They propose to parameterize a shape by modeling precisely the portion of its configuration space that is meaningful. To represent this subspace, they use a cross product of simplicial complexes which can accommodate arbitrary topology. Each vertex in the complex is associated with a point in the shape’s configuration space. Points inside

a simplex indicate interpolation via Barycentric weights. While Noh and colleagues propose building the complex by hand, Kovar and Gleicher [2001] provide an easier mechanism for constructing it in which the user is asked to classify a series of drawings as valid or invalid. Once the complex is build, the user is free to select any position (or sequence of positions to create an animation) within it and is guaranteed to get a meaningful result. Similarly, a SSD skeleton or other rigging control described in this section is used to parameterize the space of meaningful mesh deformations—the mesh kinematics.

2.3 Animation

Once a character has been modeled and rigged, it is ready for animation. Animating a character is the process of setting the values of the rigging controls over time in order to create the illusion of movement. More stylized motion is usually created using keyframing in which the rigging controls are set at key moments in time. The values of the controls at in between frames are interpolated by the computer. Keyframing gives the artist full creative control over the resulting animation but, as a consequence, burdens the artist with generating every required nuance of motion. When more realistic motion is appropriate, the movement of a real actor can be digitized in a motion capture studio. Motion capture returns skeletal motion that can be used to deform a mesh via SSD or other skeleton-based rigging methods. Physics-based animation, which entails simulation of the physical laws that govern the character’s movement, can generate realistic motion from sparse constraints and generalize captured motion. However, the difficulty of controlling physical simulations can pose a problem when artistic requirements demand a particular effect. Indeed, in both motion capture and physics-based animation, the artist gives up control in order to simplify the animation process. Much of the research in these areas focuses on regaining the lost control without sacrificing the benefits.

2.3.1 Keyframe Animation

Keyframe animation extends the process of traditional hand-drawn animation to the computer-generated setting. In traditional animation, a character is drawn in key poses that capture the overall action of the scene and its timing. Once the key poses have been established, the ones in between are drawn in a process aptly named “inbetweening” [Johnston and Thomas 1995]. An analogous procedure is followed in computer-generated animation. The values for the rigging controls are set at key points

in time and the values in between are computed by the animation software. Spline interpolation is most often used for this purpose since it gives the animator control over continuity at the keyframes, which is necessary to achieve the proper timing of movement. When generating keyframe animation, an animator works much like a sculptor. First the gross motion is mapped out by setting keyframes, say, for the global translation of the character. Then, little by little, more details are added. The animator constantly “scrubs” through the motion generated so far to get an idea of what should be animated next. The end result is fluid movement with both exaggerations and subtleties that, if done properly, mask the deliberate intentions of the animator and make it seem as if the character is acting on its own volition [Johnston and Thomas 1995].

Because of the strong correspondence between hand-drawn and computer-generated animation, it is no surprise that the same fundamental principles that guide the art of hand-drawn 2D animation apply to its 3D counterpart. Lasseter [1987] enumerates the principles perfected by the Walt Disney Studio [Johnston and Thomas 1995]—squash and stretch, timing, anticipation, staging, follow through, overlap, slow in / slow out, arcs, exaggeration, secondary action, and appeal—and describes how to apply them in the computer generated setting. Although the roots of these principles are in art, they are surprisingly applicable to technical research in computer graphics. For example, the principle of squash and stretch says that objects in motion should be flattened or elongated to emphasize action and speed. However, the volume of the object must remain constant during these changes. Both the space deformations of Barr [1984] and free-form deformation [Sederberg and Parry 1986] described in Section 2.1.2 provide the facility to generate squash and stretch. However volume preservation has long been and continues to be an active area of research [Sederberg and Parry 1986; Lee et al. 1995; Rappoport et al. 1996; Hirota et al. 1999; Botsch and Kobbelt 2003; Angelidis et al. 2004].

Timing is a critical aspect of animation that conveys weight, size, speed, resistance, and force, permits anticipation, follow through, overlap, and reaction, and even suggests mood [Whitaker and Halas 2002]. In response to the importance of timing, researchers investigate intuitive interfaces to specifying it. Terra and Metoyer [2004] develop a mechanism to re-time keyframe animation by “acting out” the desired timing using a 2D sketching interface. Thorne, Burke, and van de Panne [2004] provide a complete authoring system for computer-generated animation based on sketching that includes a “cursive alphabet” for motion specification. Tokens in the alphabet are mapped to keyframe animation sequences which are re-timed to match the speed with which the curves are drawn.

Igarashi, Moscovich, and Hughes [2005a] present an alternative to temporal keyframing in order to accommodate performance-driven animation. In *spatial keyframing*, key poses are associated with positions in 3D space rather than positions in time. Poses at other points in space are computed using radial basis interpolation. An animation is generated interactively by drawing a path through the 3D domain. This method is quite similar to shape interpolation techniques that allow the user to place shapes at arbitrary locations in an abstract space [Lewis et al. 2000; Sloan et al. 2001]. The spatial keyframing method restricts the space to three dimensions and interpolates transformations rather than vertex positions. In cartoon capture [Bregler et al. 2002], a subset of n hand-selected examples from an existing animation form an n -dimensional space. A path through this space (analogous to the path drawn interactively in spatial keyframing) is found automatically in order to reproduce the rest of the animation. Replacing the n examples with n new ones of a different character in the same poses retargets the animation to the new character. Likewise, replacing the poses used in spatial keyframing with ones of a new character would retarget any recorded animations created with this method. James and Twigg [2005] solve a problem similar to that of cartoon capture where deformations are controlled by SSD rather than shape interpolation. In order to approximate an animation provided as a sequence of mesh poses, a collection of bones and SSD weights are found, as well as a set of bone transformations for each animation frame. This transformation sequence can be thought of as a “path” through the SSD parameter space analogous to the path found for cartoon capture. However, the similarities do not extend to transfer since there is no simple way to transfer a SSD animation onto a new character.

The animator’s overarching goal is to imbue a character with personality and style in order to tell a compelling story. The character should appear as if all of its actions are the result of its own decisions and thought processes [Lasseter 2001]. In the research community, two notable attempts at creating autonomous characters that act and react under their own volition and with their own personality and style (or appear to, at least) are the Improv system of Perlin and Goldberg [1996] and the Gertie system of Loyall and colleagues [2004]. These systems present authoring tools to generate believable characters that respond to each other and to user input according to their personality and mood while always staying true to the author’s goals and intentions. In essence, these tools are designed to create an autonomous computer-generated “Kermit the Frog” or “Bugs Bunny” that acts appropriately even though the puppeteer or animator is not in direct control. Both systems provide an animation component responsible for movement and a behavior component that encapsulates

a character's personality and decision making process. Improv uses a behavior scripting language suitable for non-programmers that allows the author to craft complex non-deterministic behavior and account for communication with other characters. Gertie provides a more full-featured language designed to process, evaluate, and react to concurrent action.

Neff and Fiume [2005] provide an alternative authoring environment for expressive animation that formalizes the separation between actions (such as waving) and style so that the two can be authored independently. The same action will be performed differently by characters authored with different styles. This system builds upon the authors' earlier work on modeling the expressiveness conveyed by muscle tension and relaxation [Neff and Fiume 2002], on interactive tools to edit the activation time, range of motion, and extent of joints in an animated sequence [Neff and Fiume 2003], and on exploring expressive variations in posture [Neff and Fiume 2004]. While some of these methods rely on dynamic simulation (discussed below in Section 2.3.3), they are included here to emphasize their focus on the aesthetic aspects of character animation.

Chi and colleagues [2000] develop tools to modify animated motion based on theoretical results in classifying human movement. Motions are modified based on traits in two categories: *effort* involves the level of tension and control, the sense of urgency, and the level of impact, while *shape* rates the motion on being horizontal, vertical, and forward/backward. The authors develop algorithms to change existing motions along these dimensions.

When skeletal rigging such as SSD is used, keyframe animation involves selecting angles for the joints of the skeleton at key points throughout the animation. Inverse kinematics (IK) allows the animator to pose the skeleton by directly positioning its extremities (e.g., hands and feet). The joint angles required to satisfy the positional constraints are found automatically [Zhao and Badler 1994]. However, IK is a classic example of an underdetermined problem: many different joint configurations are valid possibilities to meet the user's constraints. In any IK implementation, some mechanism is required to select one out of the many valid configurations. Grochow and colleagues [2004] propose to resolve this ambiguity in a way that respects a character's style. Their system uses a training set of skeletal poses to learn a probability distribution function (PDF) over the space of all poses. Based on this PDF, their algorithm selects the most likely pose that meets the user-supplied constraints. By using a training set with examples of a particular style, the IK solution will exhibit the same style when new poses are created.

2.3.2 Motion Capture

Motion capture is the process of recording the time-varying kinematic configuration of a real actor as he or she performs a task or action. Modern motion capture technology [Vicon 2004] uses reflective markers which are attached to the actor's body at various positions. Cameras placed around the workspace are used to recover the 3D positions of the markers over time. From these positions, the joint angles of a kinematic skeleton are estimated at each frame. The result is an animated skeleton performing the same action as the actor. Thus "motion," in the context of motion capture, refers to the recorded skeletal animation. The strength of motion capture is the relative ease with which realistic motion can be generated. The recorded data exhibits the same nuances and style as the performer which makes the motion appear natural and believable. The drawback of motion capture is the difficulty of modifying the data once it has been recorded. Motion capture data represents the exact action performed during the recording session and is appropriate only for a kinematic skeleton that matches the proportions of the actor. Research in this area focuses on generalizing the captured data so that it can be applied in a wider variety of situations. These efforts include editing the motion capture data to easily generate smooth kinematic changes over time, retargeting the data to fit the proportions of a different sized skeleton, combining small segments of motion to generate a wide array of complex actions, and modifying the style in which an action is performed. In short, just as with triangle meshes, as soon as the data is available, people want to edit it.

Early work applies signal processing techniques to filter, displace, and warp motion curves [Bruderlin and Williams 1995; Witkin and Popović 1995]. These methods permit simple kinematic changes, such as making a walking character duck through a doorway, by adding smooth displacements. However, excessive work is required for complex alterations. Gleicher uses an optimal trajectory method to edit motion capture data [Gleicher 1997] and retarget it to a new skeleton with identical structure but different limb lengths [Gleicher 1998a]. The optimal trajectory formulation (discussed also in Section 2.3.3) permits arbitrary constraints at any point in time and can accommodate more complex kinematic changes. However, the entire animation must be solved for in one global optimization problem so that the motion at earlier points in time can anticipate constraints which will become active later. Instead of optimal trajectory, Lee and Shin [1999] use a hierarchical curve fitting procedure that allows constraints to be enforced without discontinuities. Monzani and colleagues [2000] employ an intermediate skeleton and an ease-in/ ease-out procedure for constraints. Le Callennec and

Boulic [2004] demonstrate how to prioritize constraints to avoid problems that arise with competition. When constraints on end effectors are explicitly defined throughout time rather than being turned on and off at discrete moments, solutions based on inverse kinematics permit “online” motion retargeting where the output is computed frame-by-frame without global knowledge of future constraints [Choi and Ko 2000; Shin et al. 2001; Meredith and Maddock 2005]. These methods are especially useful in performance driven animation where future constraints are not known. Gleicher presents a comprehensive overview of constraint-based kinematic methods including much of the research described in this paragraph [2001].

The methods discussed so far are kinematic in nature. Superior results are achieved with algorithms that employ physics to guide motion adaptation, albeit at a much higher computational cost. The use of optimal trajectory techniques with constraints to enforce physics has been well explored. Most research concentrates on ways to manage the complexity of human motion [Popović and Witkin 1999; Liu and Popović 2002; Safonova et al. 2004; Abe et al. 2004; Sulejmanpašić and Popović 2005]. Alternatives to optimal trajectory formulations are dynamic tracking which employs forward dynamics to track recorded motion in a physically realistic way [Zordan and Hodgins 1999; Nancy S. Pollard 2001], dynamic filtering which modifies recorded motion to increase physical realism [Tak et al. 2000; Yamane and Nakamura 2003; Shin et al. 2003; Tak and Ko 2005], and force-based editing which modifies estimated forces to achieve new goals [Pollard and Behmaram-Mosavat 2000]. The general approach of physics-based animation is discussed in more detail in Section 2.3.3.

Motion editing and adaptation methods generalize specific motions to meet new constraints. In contrast, motion graph methods create new motions by resequencing existing ones [Kovar et al. 2002; Lee et al. 2002; Arikan and Forsyth 2002]. Given a database of motion capture data, a graph is formed by identifying segments of motion that can be plausibly blended. A walk through this graph generates continuous movement by transitioning between the different segments. The transitions can be computed using algorithms for motion blending [Perlin 1995; Rose et al. 1996; Park et al. 2002; Kovar and Gleicher 2003]. Given a motion graph, specific goals such as path following can be accomplished by searching for the proper graph traversal. Computing the traversal interactively may require precomputation [Lee and Lee 2004] or careful graph design [Gleicher et al. 2003]. The expected quality of the results from using a given motion graph for navigation in a particular environment can also be estimated [Reitsma and Pollard 2004].

One of the strengths of motion capture is its ability to reproduce the nuances of style that make human motion look natural. Since style strongly influences perception, it plays an important role in animation. Unuma, Anjyo, and Takeuchi [1995] demonstrate that adjusting frequency bands in recorded motion can alter its perceived emotional content. Rose, Cohen, and Bodenheimer [1998] explicitly model style by scoring captured motions according to their style content and then blending between them. Brand and Hertzmann [2000] and Hsu, Pulli, and Popović [2005] develop automatic ways to learn the style exhibited in motion capture data and then retarget it to novel inputs.

2.3.3 Physics-Based Character Animation

In physics-based animation, motion is computed by simulating the physical laws that govern movement in the real world. Research in this area strives to generate *realistic* motion such that a physical replica of the animated object or character would move in the same way as the computer-generated version. Indeed, many of the techniques discussed in this section have their roots in the study of robot locomotion [Raibert 1986; Raibert and Hodgins 1991]. In physics-based animation, physical parameters such as mass, force, velocity, and torque determine object motion indirectly via simulation. The difference between this scenario and keyframe animation is obvious when considering a simple example where a character drops a ball. In keyframe animation, the ball will hover motionless in space unless the animator explicitly keyframes its movement. In physics-based animation, the ball will fall to the ground under the force of gravity and bounce back up again. Simulation can be extremely advantageous when the resulting motion matches the user's expectations. However, it can be just as frustrating when even a small change is required. The mapping between the simulation inputs and the generated motion is complex, making any particular change difficult to achieve.

Although physics-based animation focuses on realism and keyframe animation often focuses on expressive and caricatured motion, the two are not opposites. In fact, much of the art of traditional animation is grounded in physics. When creating expressive animation by hand, some of the properties an animator must consider include an object's weight, how it will react when a force is applied, inertia, momentum, reaction forces, gravity, center of mass, friction, and drag [Whitaker and Halas 2002]. The expressiveness of traditional animation involves understanding the physical movement of an object and then deliberately exaggerating the essential properties of that movement to an extreme [Whitaker and Halas 2002].

This strong connection to physics is evidenced in optimal trajectory methods, dubbed “spacetime constraints” by Witkin and Kass [1988]. This formulation casts animation as constrained optimization where the trajectories of the character’s degrees of freedom throughout the animation are the unknowns. The user models the physical properties of the character such as its shape, mass, how it is articulated, and how it is actuated (e.g., by using its muscles to create torques at its joints). Constraints on the character’s position at different moments in time specify the goals of the animation and an objective function such as energy conservation indicates how the character should accomplish the goals. An optimization procedure automatically finds an animation that must obey the laws of physics, that must satisfy all constraints, and that minimizes the objective. Witkin and Kass [1988] demonstrate that, when optimal trajectory methods are applied to *ballistic* motion, principles from expressive animation such as squash-and-stretch, anticipation, and follow-through naturally emerge. During the flight phase of a jump, a character’s motion is determined solely by its momentum and the force of gravity. In order to land in a particular position, the character must anticipate that goal before leaving the ground and adjust the takeoff accordingly. Since optimal trajectory methods solve for the entire animation sequence at once, they are able to resolve this anticipation such that constraints at any point in time can have a global influence. The optimal way for an articulated “Luxo” lamp to jump from one spot to another is to crouch down in anticipation of the jump, propel itself into the air by stretching out its body completely, and then tuck in at landing to absorb the impact. Furthermore, by increasing the mass of the lamp’s base, an exaggerated version of the jump is created where the base appears excessively heavy [Witkin and Kass 1988].

The optimal trajectory formulation, while effective, has limitations with regard to complexity, interactivity, and goal specification. The optimization problem is highly nonlinear and rapidly grows in complexity with the number of degrees of freedom and the length of the animation. Since the entire animation is solved for at once, all constraints and objectives must be specified ahead of time with no chance for unexpected user interaction. Finally, while a simple task such as an energy-conserving jump is easily specified, more complicated movements or specific styles are difficult to encode mathematically.

In order to cope with the nonlinear nature of the problem, Ngo and Marks [1993] promote a global search that explores a wider range of trajectories when trying to find the best one. Cohen [1992] divides the large problem into small “windows” which cover smaller ranges of time as well as subsets of the character’s full degrees of freedom. This reduces the complexity and accommodates user input since

the entire motion is not solved for at once. Other researchers mitigate the complexity of human motion using a simplified character model [Popović and Witkin 1999], a hierarchical wavelet representation of character motion [Liu et al. 1994], an alternative constraint formulation [Liu and Popović 2002; Fang and Pollard 2003; Abe et al. 2004], or dimensionality reduction [Safonova et al. 2004]. Sulejmanpašić and Popović [2005] demonstrate that proper scaling of the constraint equations allows efficient simulation of human motion without simplification, reformulation, hierarchical representation, or reduction. In order to relieve the burden of inventing an appropriate objective function for every desired class of motion, many of these and other techniques [Rose et al. 1996; Popović and Witkin 1999; Abe et al. 2004; Sulejmanpašić and Popović 2005] address motion adaptation (see Section 2.3.2) rather than generating the motion from scratch. Liu, Hertzmann, and Popović [2005] describe how to learn the physical parameters of a character such as its muscle preferences from motion capture data in order to generate physical motions in different styles.

A different approach for physics-based animation combines forward dynamics with a general control procedure, or controller, that encapsulates a desired action or behavior such as running or balancing [van de Panne et al. 1990]. The controller examines the current state of the character (e.g., its kinematic configuration, external forces, etc.) and decides how to apply torques at the joints in order to reach a desired goal state or to satisfy criteria pertinent to the motion that the controller is designed to generate. For example, a balance controller may decide to apply torques to the character’s knees in order to adjust the center of mass [Wooten 1998]. A controller can be thought of as a model of the character’s instinctual reflexes that produce coordinated body movement for a particular task [Popović and Witkin 1999]. Controllers allow the character to react to unexpected forces that might be generated interactively by the user. Work in this area is derived from research in robot locomotion [Raibert 1986; Raibert and Hodgins 1991] where sensors on the robot provide input to the controller and torques are produced by actuators. Raibert and Hodgins [1991] present simulated versions of actual robots that they built. A parallel can be drawn between this setting and the “straight-ahead” technique from hand-drawn animation. Unlike keyframe animation where an entire sequence is planned ahead of time using key poses, straight-ahead animation involves drawing one frame right after the next in sequence [Blair 1994]. The animator must decide at each frame how the shape should evolve in order to produce the desired result. In controller-driven animation, the control procedure makes an analogous local choice about how to update the character’s state.

Controllers have been hand-designed for many specific tasks including, among others, running and vaulting [Hodgins et al. 1995], bicycling [Brogan et al. 1998], leaping, tumbling, landing, and balancing [Wooten 1998], diving [Wooten and Hodgins 1996], and swimming [Yang et al. 2004]. However, designing controllers by hand is difficult and requires a significant amount of tuning for each new behavior and for each new character. Hodgins and Pollard [1997] describe an algorithm to automatically adapt existing control systems to new characters. In order to develop a more varied repertoire of dynamic actions, research has focused on composing simple controllers to create more complex aggregate actions [van de Panne et al. 1990; Wooten 1998; Faloutsos et al. 2001a; Faloutsos et al. 2001b]. Other methods augment forward dynamic simulation with motion capture to handle impact, collision, and response [Zordan and Hodgins 2002; Komura et al. 2004; Zordan et al. 2005] and to transition between simulation and motion capture [Shapiro et al. 2003].

Algorithms for the control of physics-based animation are both challenging and especially important since, by the very nature of simulation, the animator has relinquished much of her control over the resulting motion. Popović and colleagues [2000] allow direct manipulation of rigid body simulations at any point during the animation timeline. A new motion is immediately found that meets the modified constraints. Subsequent work [Popović et al. 2003] provides a motion sketching interface to design rigid-body motion using a mouse or hand gestures. Kondo, Kanai, and Anjyo [2005] extend keyframe control to elastic objects, although the resulting deformations are only loosely based on physics. For actuated characters, simulations can be controlled by interactively selecting from a collection of predefined controllers or by mapping the degrees of freedom of an input device to the character's degrees of freedom [Laszlo et al. 2000; Zhao and van de Panne 2005]. Laszlo, Neff, and Singh [2005] provide predictive feedback that looks into the (simulated) future and displays the possible results of control decisions. Finally, a recent focus on control algorithms for fluid simulation allows fluids to be animated as if they were characters [Treuille et al. 2003; Fattal and Lischinski 2004; McNamara et al. 2004; Shi and Yu 2005].

Deformation Transfer

3

Shape deformation plays a critical role in the animation pipeline. Sculpting tools are used to deform a character’s shape to match the artistic vision of the designer. Rigging controls are created to parameterize the character’s meaningful deformations, and values for these controls are either keyframed, captured, or simulated in order to create continuous deformation over time. Creating these deformations requires a tremendous amount of artistic and technical expertise. The vast amount of research that aims to assist the user in each of these tasks attests to their difficulty. Despite the importance of deformations in the animation process and the amount of artistry, skill, and time required to create them, there are few techniques to help with reuse. The work spent designing a deformation typically cannot be reused after its planned application.

Deformation transfer—the contribution discussed in this chapter—reuses any deformation of a source mesh by transferring it onto a different target mesh [Sumner and Popović 2004]. Figure 3-1 shows an overview of the transfer process. The deformation of a source mesh is specified by a reference pose S and a deformed pose \tilde{S} where the mesh topology (number of vertices, number of triangles, and connectivity) remains the same but the vertex positions have been changed. Given a matching reference pose T of a target mesh with a different topology, the goal of deformation transfer is to apply the exhibited source deformation to the target in a natural way and produce a deformed target \tilde{T} .

This problem presents several challenges. When editing tools are used to sculpt a mesh’s shape, they leave behind no persistent history of the changes that have taken place. The many intermediate

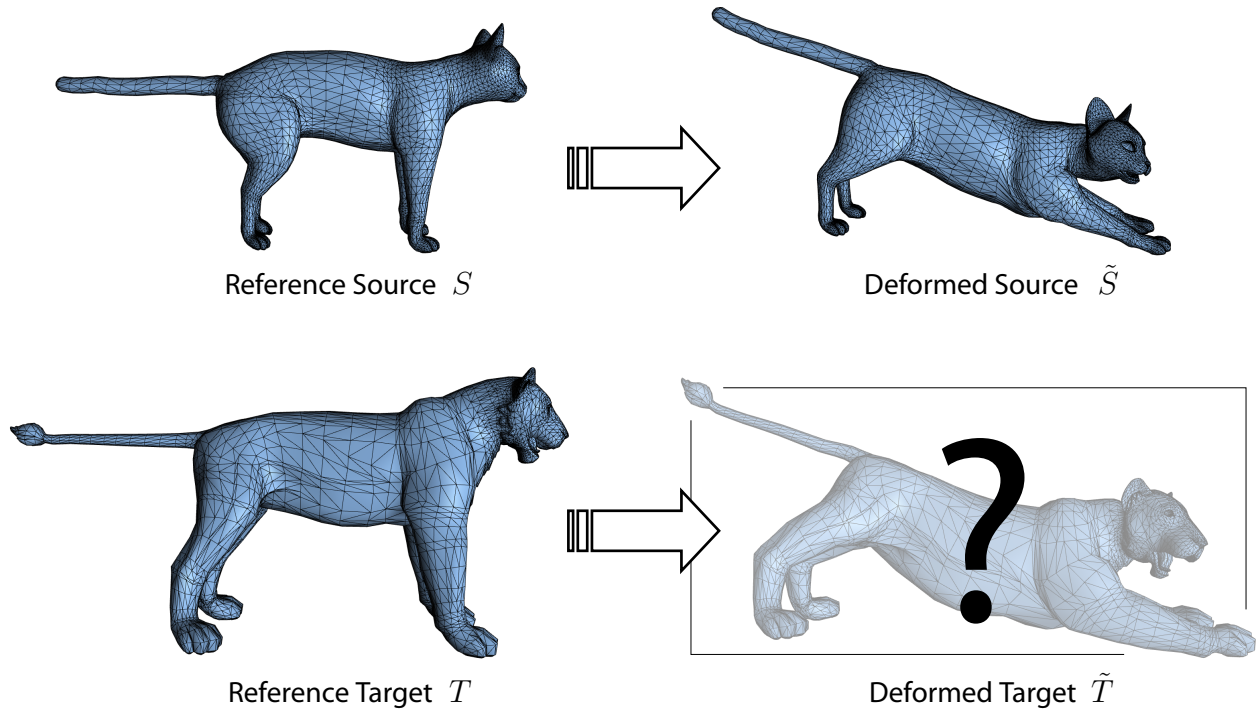


Figure 3-1: An overview of the deformation transfer problem. We are given a source mesh in both a reference pose S and a deformed pose \tilde{S} . The challenge of deformation transfer is to apply the exhibited deformation of the source onto a different target mesh T in order to create a deformed version of the target \tilde{T} .

operations used to arrive at the final version are usually discarded to conserve space. Although rigging controls often define a procedural equation for mesh deformation, the procedure may depend on many parameters. The control must be specialized for each character's shape by carefully tuning these parameters in order to achieve a desired result. The parameters selected for one character will, in general, not be appropriate for another. Re-tuning the rigging parameters is likely to be just as time consuming as starting from scratch. The problem is compounded in the common case where several different rigging controls are used in tandem. Designing automatic adaptation methods for all forms of rigging and their combinatorial combinations is impractical. Furthermore, other shape deformations are not procedural in nature and instead employ complex numerical simulations. In light of the breadth of deformation algorithms used in the animation process, the first challenge in the transfer problem is one of *deformation representation*. We require a representation of mesh deformation that is independent of the algorithm that produced the deformation in the first place. In order to be maximally general, it should not depend on any particular modeling procedure or rigging algorithm.

When discussing the idea of deformation transfer, intuitively we know that the legs of the target mesh should deform like the legs of the source, the head of the target like the head of the source, the tail like the tail, and so on. However, current geometric representations have no concept of legs, head, or tail. Since mesh topology almost always differs between characters, it is not clear how deformations of one mesh should be transferred to a different one. Thus, the second challenge is one of *correspondence*. We require some method to make precise the mapping between the source and target and define which parts of the two shapes should deform similarly.

The final challenge is *transfer*. Given our representation of deformation and of correspondence, we must develop an algorithm to transfer deformations of one character onto another via the correspondence. The transfer procedure should be efficient and broadly applicable to the variety of deformations and characters used in animation. Since there are many ways to represent deformation and to specify correspondence, it is important to choose ones that are amenable to transfer. My solutions to these three challenges are tightly coupled, with the single purpose of adding reuse to the animation process.

3.1 Deformation Representation

In order to reuse shape deformations we need some way to represent the deformation that will be transferred. This requirement is related to the issue of geometric representation discussed in Section 2.1.1. Here, however, we address the *change* in the shape's geometry, rather than the geometry itself. The representation should be general enough to accommodate the wide variety of methods used in animation to create deformations. A general representation will allow *any* deformation to be transferred, whereas assumptions made about the form of the deformations will necessarily limit the applicability of the overall method.

3.1.1 Displacement Fields

One possible representation is a displacement field that encodes the positional change of each source vertex. Noh and Neumann [2001] use this approach to transfer facial expressions from one face mesh to another. Each expression is encoded with vertex displacements that define the differences between the reference face and the expression face. Expressions are transferred by applying the source

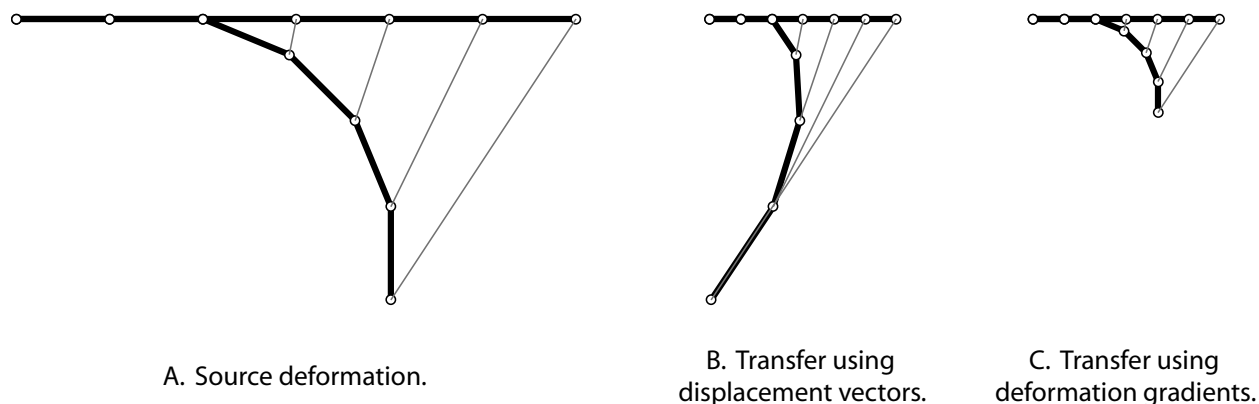


Figure 3-2: The transfer problem is demonstrated on a bending line. (A) A horizontal line is bent downward and represents the source deformation. Vertex displacements are shown in gray. (B) If the displacements are copied directly to a shorter target line, the result is distorted in an unnatural fashion. (C) Deformation transfer creates a more a natural result by transferring differential changes: the change in length and orientation of each segment of the source line is applied to the target. (The line shown was deformed using my system.)

displacements to the target mesh. While this method works for the relatively small deformations of facial expressions, it does not extend to the general case of full-body poses. Indeed, other displacement-based algorithms account for full-body deformation in a domain-specific way by, for example, using skeleton-subspace deformation [Lewis et al. 2000; Sloan et al. 2001; Kry et al. 2002].

The limitations of a displacement-based approach are best understood by considering the overall goal of deformation transfer: extract the change in shape exhibited by the source and apply it to the target. Change is most naturally represented as a differential quantity. Displacements encode the change in Cartesian coordinates but do not indicate how differential vectors have been modified. Consequentially, applying displacement vectors to another shape does not preserve differential changes. For example, if a character bends its arm at the elbow, displacement vectors do not discover the fact that the arm has rotated. Applying these vectors to another character does not result in rotation. Figure 3-2 illustrates the failure of a displacement-based transfer algorithm for the simple case of a bending line. Noh and Neumann [2001] employ heuristics to mitigate the artifacts seen in Figure 3-2 B. Their method adapts the length and orientation of displacement vectors based on differences in local shape properties of the reference source and reference target faces. However these corrections are limited. By considering differential changes, my algorithm permits the transfer of large-scale deformations without the need of heuristics. The line in Figure 3-2 C is deformed with my transfer implementation and faithfully reproduces the source deformation.

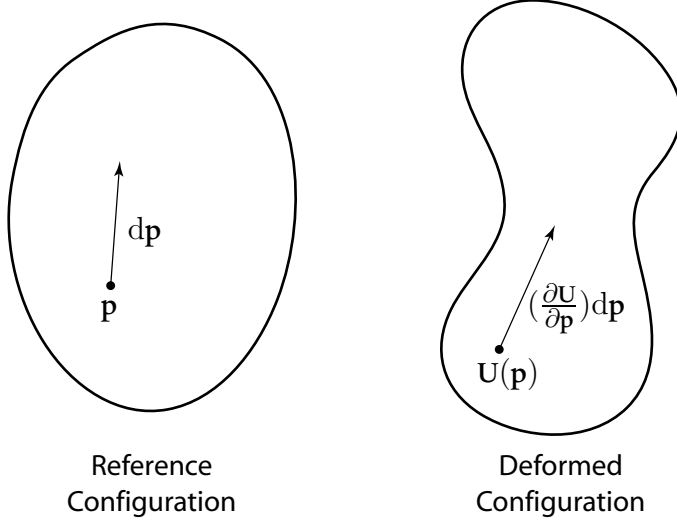


Figure 3-3: A point \mathbf{p} in the reference configuration is mapped to the deformed configuration by the deformation function \mathbf{U} . Likewise, a differential vector $d\mathbf{p}$ is mapped by the deformation gradient $\frac{\partial \mathbf{U}}{\partial \mathbf{p}}$.

3.1.2 Deformation Gradients

Our objective is to encode shape deformation using a differential specification so that we can develop a reuse algorithm that transfers differential changes. Continuum mechanics deals with the behavior of materials subjected to external forces [Lai et al. 1993]. Because solids deform when under load, the field of continuum mechanics provides tested methods for representing large deformations. The so-called deformation gradient provides precisely the representation we need. Consider a solid object in both a reference configuration and a deformed configuration (Figure 3-3). Points in the reference configuration are denoted by column vectors of their Cartesian coordinates $\mathbf{p} = [p_1 \ p_2 \ p_3]^\top$. The position of \mathbf{p} after deformation, denoted as $\tilde{\mathbf{p}} = [\tilde{p}_1 \ \tilde{p}_2 \ \tilde{p}_3]^\top$, is determined by the function $\mathbf{U} : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ so that

$$\tilde{\mathbf{p}} = \mathbf{U}(\mathbf{p}) = \begin{bmatrix} U_1(p_1, p_2, p_3) \\ U_2(p_1, p_2, p_3) \\ U_3(p_1, p_2, p_3) \end{bmatrix}. \quad (3.1)$$

The deformation of an infinitesimal vector $d\mathbf{p}$ within the solid is determined by the deformation gradient $\frac{\partial \mathbf{U}}{\partial \mathbf{p}}$. Since \mathbf{U} maps from \mathbb{R}^3 to \mathbb{R}^3 and varies with position, its gradient is a second-order tensor field:

$$\frac{\partial \mathbf{U}}{\partial \mathbf{p}} = \begin{bmatrix} \frac{\partial U_1}{\partial p_1} & \frac{\partial U_1}{\partial p_2} & \frac{\partial U_1}{\partial p_3} \\ \frac{\partial U_2}{\partial p_1} & \frac{\partial U_2}{\partial p_2} & \frac{\partial U_2}{\partial p_3} \\ \frac{\partial U_3}{\partial p_1} & \frac{\partial U_3}{\partial p_2} & \frac{\partial U_3}{\partial p_3} \end{bmatrix}. \quad (3.2)$$

The deformation gradient determines how $d\mathbf{p}$ changes as a result of the deformation \mathbf{U} , yielding $d\tilde{\mathbf{p}} = (\frac{\partial \mathbf{U}}{\partial \mathbf{p}})d\mathbf{p}$. It encodes the change in orientation, scale, and skew that differential vectors within a solid have undergone. Because the deformation gradient encodes both rigid rotation and stretching, it is an appropriate choice to represent the large deformations we wish to transfer.

Equation 3.2 is the Jacobian matrix of \mathbf{U} , making this formulation identical to Barr's [1984] local deformations discussed in Section 2.1.2. Barr uses the deformation gradient in the context of geometric modeling in order to update tangent and normal vectors after a deformation has been applied. In this setting, the function \mathbf{U} is known explicitly and the deformation gradient at a particular point is found by computing the partial derivatives analytically and then evaluating them at the point. Barr's setup limits the class of deformations to those that are easily expressed analytically such as bends, twists, and stretching along predefined axes.

Simplicial Dissection

In the more general case where there is no analytic formulation, \mathbf{U} can be discretized via a simplicial dissection (triangulation in 2D or tetrahedralization in 3D) of the shape. Each vertex i of the simplicial complex is mapped by \mathbf{U} from its initial position \mathbf{v}_i in the undeformed configuration to its deformed position $\tilde{\mathbf{v}}_i$. Since points within each simplex are determined by a linear interpolant, the deformation gradient tensor field is piecewise constant with one constant tensor \mathbf{J}_j for each simplex j . These tensors are referred to as the deformation gradients. Thus, each simplex acts like a differential volume whose edge vectors (as well as all other vectors within the simplex) are transformed by its deformation gradient. In 3D, every simplex is a tetrahedron (tet) with four vertices. If these four vertices have indices $\{i_1, i_2, i_3, i_4\}$, then the deformation gradient transforms the three independent edge vectors of the tet according to:

$$\begin{aligned} \mathbf{J}_j(\mathbf{v}_{i_2} - \mathbf{v}_{i_1}) &= \tilde{\mathbf{v}}_{i_2} - \tilde{\mathbf{v}}_{i_1} \\ \mathbf{J}_j(\mathbf{v}_{i_3} - \mathbf{v}_{i_1}) &= \tilde{\mathbf{v}}_{i_3} - \tilde{\mathbf{v}}_{i_1} \\ \mathbf{J}_j(\mathbf{v}_{i_4} - \mathbf{v}_{i_1}) &= \tilde{\mathbf{v}}_{i_4} - \tilde{\mathbf{v}}_{i_1} \end{aligned} \tag{3.3}$$

We can rewrite this expression in matrix form

$$\mathbf{J}_j \mathbf{V}_j = \tilde{\mathbf{V}}_j, \tag{3.4}$$

where the 3×3 matrices \mathbf{V}_j and $\tilde{\mathbf{V}}_j$ contain the undeformed and deformed edge vectors, respectively, as columns:

$$\begin{aligned}\mathbf{V}_j &= [\mathbf{v}_{i_2} - \mathbf{v}_{i_1} \quad \mathbf{v}_{i_3} - \mathbf{v}_{i_1} \quad \mathbf{v}_{i_4} - \mathbf{v}_{i_1}] \\ \tilde{\mathbf{V}}_j &= [\tilde{\mathbf{v}}_{i_2} - \tilde{\mathbf{v}}_{i_1} \quad \tilde{\mathbf{v}}_{i_3} - \tilde{\mathbf{v}}_{i_1} \quad \tilde{\mathbf{v}}_{i_4} - \tilde{\mathbf{v}}_{i_1}]\end{aligned}\tag{3.5}$$

By solving Equation 3.4 for \mathbf{J}_j we obtain a closed form expression for the deformation gradient:

$$\mathbf{J}_j = \tilde{\mathbf{V}}_j \mathbf{V}_j^{-1}.\tag{3.6}$$

Geometrically, the columns of \mathbf{J}_j are the basis vectors (i.e., edge vectors) of the deformed tet expressed in the undeformed tet's basis. If both the undeformed and deformed vertex positions are known, the deformation gradients can be computed by evaluating Equation 3.6.

Isomorphic Dissection

In continuum mechanics, the deformation of a solid is usually not known a priori but instead evolves with time according to a physical simulation. The problem formulation involves discretizing an object in its reference configuration and then performing some calculation to determine its deformation. Deformation gradients are used because they provide a way to measure physical quantities such as stress. Implicit in this setting is a correspondence between the undeformed and deformed configurations that allows deformation gradients to be computed for each simplex using Equation 3.6.

Our setting is different since we start with *two* shapes: a mesh in both a reference pose and a deformed pose. We wish to encode the known deformation of the boundary using deformation gradients. But, since deformation gradients are defined for solids, they do not immediately apply when only the boundary is known. Discretizing the interior is more complicated in this case. Although the boundaries of the two meshes are in one-to-one correspondence, independent tetrahedralization of each mesh will not result in correspondence within their interiors.

Alexa, Cohen-Or, and Levin [2000] address this problem in the context of shape interpolation. Given the boundary of two shapes in correspondence, they use deformation gradients to compute interpolation sequences that maximize the local rigidity within the interior. In order to employ deformation gradients in this context, the authors suggest an *isomorphic* simplicial dissection of the

two shapes. That is, their algorithm finds a dissection of the interiors of both shapes such that the resulting vertices and simplices are in one-to-one correspondence. Once the isomorphic dissection has been found, deformation gradients can be computed for each corresponding pair of simplices by evaluating Equation 3.6. In 2D, the two shapes are defined by polygons in one-to-one boundary correspondence. An isomorphic dissection is computed using the method proposed by Aronov, Seidel, and Souvaine [1993]. In this algorithm, each polygon is first triangulated independently, which can always be accomplished without adding any additional so-called Steiner vertices in the interior. Then, both shapes are mapped to a convex n -sided polygon (where n is the number of boundary edges) so that the two individual triangulations are overlaid on top of one another. Any intersections of the triangle edges are computed in this configuration. The intersections are then mapped back to each of the original polygons in order to yield the isomorphic dissection. Alexa and colleagues extend this algorithm to 3D for genus zero triangle meshes. Both meshes are individually tetrahedralized and mapped to a convex n -sided polyhedron. Intersections of the overlaid tetrahedralizations are computed and then mapped back to each mesh. In this case, the intersection computation may result in four-, five-, or six-sided shapes that must subsequently be tetrahedralized. Both 2D and 3D isomorphic dissection algorithms require iterative refinement in order to improve the quality of the dissection and prevent numerical problems in later computations. The refinement must be performed in such a way that the isomorphism is not invalidated.

While isomorphic dissection provides one method to extract deformation gradients from boundary representations, it has many disadvantages. We are primarily concerned with 3D meshes. Even without the requirement of isomorphism, generating *any* solid dissection of a 3D mesh is a nontrivial task and computing high-quality tetrahedralizations is an active area of research [Alliez et al. 2005]. Dissecting a mesh requires that it be manifold and watertight. This strict requirement is violated by many meshes used for animation in practice and limits the ultimate applicability of the transfer method. Many tetrahedralization methods only approximate the connectivity of the input mesh. This complicates the isomorphic dissection algorithm described above since it relies on one-to-one boundary correspondence. In fact, it is impossible to tetrahedralize some triangle meshes (even though they are manifold and watertight) without modifying the boundary [Shewchuk 1998]. Unlike triangulation, tetrahedralization usually requires adding Steiner vertices which makes it more difficult to map the shape to a convex polyhedron. The 2D isomorphic dissection algorithm of Aronov, Seidel, and Sou-

vaine [1993] adds $O(n^2)$ additional vertices for an n -sided polygon. While no bound is proven for the 3D extension, the complexity of tetrahedralization suggests that it will be greater. The performance of subsequent computations depends on the number of vertices, so adding extra ones is undesirable. Reusing an animation involves transferring the deformation exhibited in each frame. If an isomorphic dissection must be computed for every new source deformation, the performance of repeated transfers is greatly reduced. Isomorphic dissection does not scale to multi-way dissections because the number of Steiner vertices quickly becomes prohibitively large. While transfer only requires the pairwise extraction of deformation gradients, the mesh-based inverse kinematics algorithm described in Chapter 5 requires multi-way extraction.

Boundary-Based Approximation

In light of these difficulties, I propose a boundary-based algorithm to approximate the deformation gradient tensor field on a triangle mesh without the need to tetrahedralize its interior. This method computes one deformation gradient for each triangle that maps the triangle’s edge vectors from their reference configuration to their deformed configuration. Each deformation gradient is a 3×3 matrix and thus has nine degrees of freedom. When working with a tetrahedralization, the image of the three independent tet edge vectors under the deformation function fully determines these degrees of freedom: each vector provides three equations, giving nine in total. However, a 3D triangle has only two independent edge vectors, yielding an underdetermined system of equations. These two vectors alone do not establish how the space perpendicular to the triangle deforms. For the source mesh, the algorithm resolves this problem by explicitly modeling the transformation of the perpendicular space, while for the target mesh, it finds the least norm transformation.

Source Deformation. When representing the source deformation, it is important to have control over the transformation of the space perpendicular to each triangle. This ensures that portions of the target mesh which are not in perfect alignment will deform appropriately when the deformation is transferred. In order to extract the source deformation, we compute a fourth “auxiliary” vertex for every triangle, essentially promoting the triangles to tets. If a triangle’s vertices have indices

$\{i_1, i_2, i_3\}$, we assign the new vertex index i_4 and compute its position according to

$$\mathbf{v}_{i_4} = \mathbf{v}_{i_1} + \frac{(\mathbf{v}_{i_2} - \mathbf{v}_{i_1}) \times (\mathbf{v}_{i_3} - \mathbf{v}_{i_1})}{\sqrt{\|(\mathbf{v}_{i_2} - \mathbf{v}_{i_1}) \times (\mathbf{v}_{i_3} - \mathbf{v}_{i_1})\|}}, \quad (3.7)$$

where $\|\cdot\|$ is the l^2 norm. This formula places the new vertex in the perpendicular direction and ensures that the perpendicular space rotates with the triangle and scales in proportion to the change in triangle edge length. An analogous computation is performed to compute the position of the deformed vertex $\tilde{\mathbf{v}}_{i_4}$. With the auxiliary vertices, the per-triangle deformation gradients can be computed in closed form by applying Equation 3.6. Thus, the source deformation gradient tensor field is approximated on the surface of the mesh without requiring dissection of the interior. While the source poses must be in one-to-one correspondence, they need not be manifold, watertight, or of a particular genus. Since no dissection is required, this technique naturally extends to the case where many deformed source meshes are given.

Target Deformation. In the original formulation [Sumner and Popović 2004], the target deformation is represented using the same procedure of promoting each triangle to a tet through the addition of a fourth vertex. However, adding extra vertices to the target is disadvantageous since it increases the complexity of later computations in the same way that additional Steiner vertices do in an isomorphic dissection. However, since the target deformation is not being transferred, we only care about what actually happens to the target triangles themselves and not the perpendicular space. This allows us to derive an expression for the per-triangle target deformation gradients that does not require an extra vertex. If triangle j has vertices with indices $\{i_1, i_2, i_3\}$ then

$$\mathbf{J}_j \mathbf{W}_j = \tilde{\mathbf{W}}_j, \quad (3.8)$$

where \mathbf{W}_j and $\tilde{\mathbf{W}}_j$ are 3×2 matrices containing the two undeformed and deformed triangle edge vectors as columns, respectively:

$$\begin{aligned} \mathbf{W}_j &= [\mathbf{v}_{i_2} - \mathbf{v}_{i_1} \quad \mathbf{v}_{i_3} - \mathbf{v}_{i_1}] \\ \tilde{\mathbf{W}}_j &= [\tilde{\mathbf{v}}_{i_2} - \tilde{\mathbf{v}}_{i_1} \quad \tilde{\mathbf{v}}_{i_3} - \tilde{\mathbf{v}}_{i_1}] \end{aligned} \quad (3.9)$$

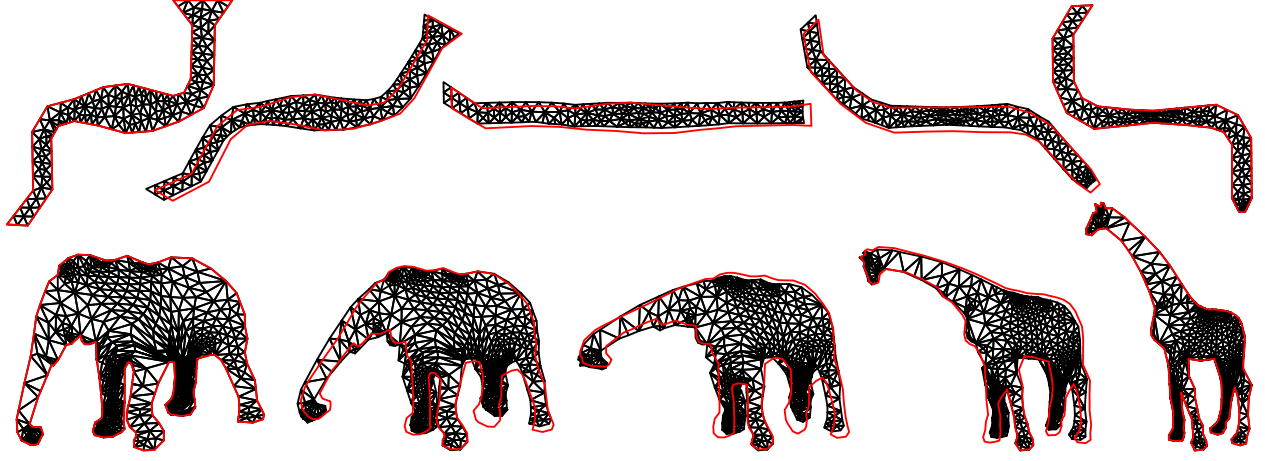


Figure 3-4: The result of the boundary-based method proposed in this dissertation is visualized by the red boundary line while the as-rigid-as-possible interpolation [Alexa et al. 2000] is visualized by the black triangulation. The body of the snake and the trunk of the elephant deform in a similar, locally rigid fashion for both methods. However, the boundary-based method is numerically much simpler as it only considers the boundary rather than the isomorphic dissection of the interior.

We wish to solve Equation 3.8 for J_j . This problem is underdetermined since J_j is a 3×3 matrix with 9 unknowns, and W_j and \tilde{W}_j are each 3×2 matrices, giving only 6 equations. However, the minimum norm solution to the underdetermined problem can be found by computing the QR factorization of W_j [Golub and Loan 1996]:

$$W_j = Q_j \begin{bmatrix} R_j \\ 0 \end{bmatrix} = \begin{bmatrix} Q_{j\alpha} & Q_{j\beta} \end{bmatrix} \begin{bmatrix} R_j \\ 0 \end{bmatrix} = Q_{j\alpha} R_j. \quad (3.10)$$

R_j is a 2×2 upper triangular matrix and Q_j is a 3×3 orthogonal matrix. The minimum norm J_j is given by

$$J_j = \tilde{W}_j R_j^{-1} Q_{j\alpha}^\top. \quad (3.11)$$

In Figure 3-4, I compare my boundary-based deformation gradient representation to the full isomorphic dissection proposed by Alexa, Cohen-Or, and Levin [2000] for 2D shape interpolation. The details of this interpolation algorithm are discussed in Chapter 5 in the context of mesh-based inverse kinematics. The interpolation sequence using my algorithm is generated with no additional vertices at a far lesser computational cost. It behaves reasonably despite ignoring the interior.

3.1.3 Summary

I propose deformation gradients as a representation of deformation because they naturally capture differential shape changes. To avoid the complexity of isomorphic dissection, I develop a boundary-based approximation of the deformation gradient tensor field where one deformation gradient is computed for each triangle. Each of the per-triangle deformation gradients encodes the change in orientation, scale, and skew induced by the deformation on that triangle. Taken as a whole, the deformation gradients can represent any mesh deformation regardless of its complexity or origin. For the source deformation where both the undeformed and deformed vertex positions are known, Equation 3.6 is used to compute the deformation gradients using an auxiliary fourth vertex. The deformation gradients that encode the unknown target deformation are related to the target vertices by Equation 3.11. This representation gives precise control over the deformation of the space perpendicular to the source triangles and ignores it when searching for a deformed target shape.

3.2 Correspondence

The goal of deformation transfer is to transfer the deformations exhibited by a source mesh onto a different target mesh. Conceptually, the desired result is clear: the head of the target should deform like the head of the source, the legs of the target like the legs of the source, the tail like the tail, and so on. A digital representation of a character as a triangle mesh has no concept of head, legs, or tail. Even if such semantic information were available, the relationship of finer scale details—down to individual triangles—would remain ambiguous. Thus, some means is required to indicate how the source triangles should be related to the target triangles.

If the mesh structure of the source and target is identical so that there is a one-to-one correspondence between the source and target triangles, it is immediately clear which parts of the two shapes should deform similarly: triangle j in the target mesh should deform like triangle j in the source. While this setting simplifies the transfer problem, it is unrealistic and overly restrictive. Few meshes are in one-to-one correspondence with one another. And, when they are, it is almost certainly the case that they were designed from the start with this correspondence as a requirement.

In order to make deformation transfer applicable in a wide range of settings, my transfer algorithm does not require one-to-one correspondence. The source and target are free to have different numbers

of vertices and triangles, as well as a different connectivity. They need not even have the same genus.

In order to cope with these differences, the user must indicate how the source and target are related to one another by supplying a mapping \mathcal{M} between the source and target triangles. This mapping is a discrete pairing of triangle indices:

$$\mathcal{M} = \{(s_1, t_1), (s_2, t_2), \dots, (s_{|\mathcal{M}|}, t_{|\mathcal{M}|})\}. \quad (3.12)$$

A pair (s_j, t_j) indicates that the target triangle with index t_j should deform like the source triangle with index s_j . Since there are no restrictions on \mathcal{M} , this mapping allows transferred deformations to originate from any region of the source mesh. In most cases, \mathcal{M} is a general many-to-many mapping, but it can also be bijective (one-to-one and onto), surjective (onto), or not-onto. Thus, one triangle of the target may be paired with several triangles of the source, and vice versa. In fact, this generality is required to enable transfer between meshes of different tessellations.

For the remainder of this chapter, we assume that \mathcal{M} has been supplied by the user. In the next chapter, I describe my correspondence algorithm which allows \mathcal{M} to be easily generated by identifying matching feature points on the source and target meshes.

3.3 Transfer

After adopting deformation gradients to represent mesh deformation and a discrete triangle pairing to identify correspondence, we have all the necessary ingredients for transfer. Our strategy is to extract the per-triangle deformation gradients from the source, map them through the correspondence \mathcal{M} , and use them to reconstruct a deformed target shape (Figure 3-5). To derive this procedure, we first consider the case where the source and target have identical mesh topology. Thus, there is a one-to-one triangle correspondence between the source and target which obviates the need for the explicit triangle associations given in \mathcal{M} with the understanding that each triangle j of the target should deform like triangle j of the source. After developing the algorithm in this setting, transfer between meshes with different structure only requires a minor change.

The deformation of the source vertices from their positions in the reference pose S to their positions in the deformed pose \tilde{S} is encoded with one deformation gradient for each triangle. If the

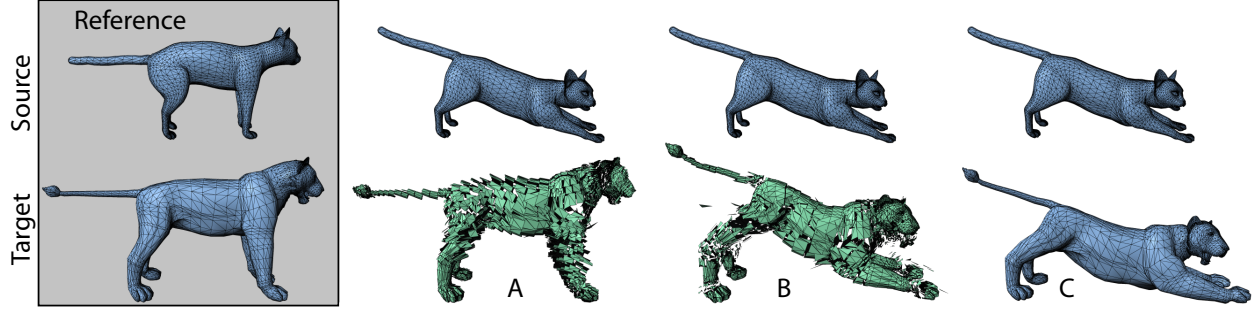


Figure 3-5: We encode a source deformation with one deformation gradient for each triangle and relate this specification to the target through a user supplied triangle correspondence. The deformation gradients indicate the ideal change in orientation and scale of the target triangles. This change is visualized in (A) by transforming the edge vectors of each target triangle by one corresponding source deformation gradient while fixing the triangle’s position in place. Since deformation gradients are a differential specification, they do not encode how the target triangles should be positioned relative to their neighbors. Deriving the new positions from the source, as shown in (B), gives a disconnected shape. In general, it is impossible to achieve a connected mesh by translating the deformed target triangles since consistency requirements are not enforced. Deformation transfer obtains the result in (C) by solving an optimization problem for a deformed target mesh whose per-triangle deformation gradients are as close as possible to the corresponding ones of the source.

source has n vertices and m triangles, Equation 3.6 is evaluated once per triangle, yielding an ordered set $\mathcal{S} = \{\mathbf{S}_1 \dots \mathbf{S}_{|\mathcal{S}|}\}$ of source deformation gradients such that $|\mathcal{S}|$ (the number of elements in \mathcal{S}) is equal to m .

Applying these deformation gradients to the target shape requires the complimentary operation: given the target reference pose T and the set \mathcal{S} , we wish to find a deformed target \tilde{T} where each triangle $j \in 1 \dots m$ has undergone precisely the change in orientation and stretch specified by \mathbf{S}_j . We refer to this process as reconstruction since we recover the Cartesian representation of the mesh from the differential specification of deformation. The reconstruction algorithm forms the core of deformation transfer.

3.3.1 Integration

Equation 3.8 relates the deformation gradient to the edge vectors of the target triangle before and after deformation: $\mathbf{J}_j \mathbf{W}_j = \tilde{\mathbf{W}}_j$. If we use the vertex positions of triangle j in the target reference pose T to build \mathbf{W}_j and evaluate $\mathbf{S}_j \mathbf{W}_j$, then the columns of $\tilde{\mathbf{W}}_j$ are the edge vectors of triangle j in \tilde{T} —the deformed target pose which we wish to recover. However, the edge vectors themselves do not provide the global positions of the vertices. In fact, because the deformation gradient representation

stores only differential properties, it is invariant to translation. Therefore, we must specify the position of one vertex arbitrarily. Then, the positions of the neighboring vertices can be found by evaluating Equation 3.8 and adding the deformed triangle edge vectors to the fixed position. By repeating this process incrementally, all vertex positions can be discovered. In essence, this procedure integrates the differential changes specified by each deformation gradient over the mesh and the fixed vertex serves as the constant of integration.

This procedure is a discrete version of the method proposed by Barr [1984] to recover global positions from a locally specified deformation. As Barr notes, it is valid only if the deformation gradients are the gradients of some global mapping of the domain. This condition is necessary for path independence when the differential changes are integrated over the mesh. If this condition is not met, the deformation gradients will be inconsistent and the recovered vertex positions will depend on the particular set of edges chosen in the integration procedure. The mesh will have artifacts where parts of it do not “meet up” because neighboring vertices were recovered using different paths. The deformation gradients in \mathcal{S} are a consistent representation of the source deformation because they are computed from a deformed source mesh. However, they are most likely inconsistent with respect to the target since the target triangles differ from those of the source in size and shape. For large meshes, numerical precision prevents an accurate reconstruction even when a consistent set of deformation gradients is supplied because small errors are compounded as the integration procedure works across the mesh. Thus, reconstruction artifacts are unavoidable with this algorithm.

3.3.2 Optimization

In order to accurately recover a deformed target mesh, all vertices must be solved for simultaneously rather than one at a time. Optimization provides a framework to perform this computation in a way that prevents precision error from accumulating and distributes error due to inconsistent deformation gradients throughout the mesh rather than concentrating it in one place. Since the deformation gradients in \mathcal{S} are inconsistent with respect to the target reference pose T , there is no deformed target \tilde{T} whose deformation gradients perfectly match those in \mathcal{S} . Thus, we treat \mathcal{S} as the set of *ideal* deformation gradients and define a new set $\mathcal{T} = \{T_1 \dots T_{|\mathcal{T}|}\}$ of *actual* ones. The optimization procedure finds the vertex positions of \tilde{T} that define the set of deformation gradients \mathcal{T} which are as close as possible to the ideal ones in \mathcal{S} . This computation is expressed by minimizing the difference

between the matrices specified in \mathcal{S} and \mathcal{T} :

$$\begin{aligned} \min_{\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n} \quad & \sum_{j=1}^m \|\mathbf{S}_j - \mathbf{T}_j\|_F^2. \\ \text{subject to} \quad & \tilde{\mathbf{v}}_k = \mathbf{c}. \end{aligned} \tag{3.13}$$

Since each $\mathbf{T}_j, j \in 1 \dots m$, can be expressed in terms of the deformed target vertices $\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n$ using Equation 3.11, the minimization is over these deformed vertex positions. The matrix norm $\|\cdot\|_F$ is the Frobenius norm, or the square root of the sum of the squared matrix elements. In the objective function, this norm evaluates to zero when \mathbf{T}_j is equal to \mathbf{S}_j . Thus, minimizing the sum over all j finds the mesh \tilde{T} whose deformation gradients are as close as possible to the ideal ones. Because, as discussed earlier, deformation gradients are invariant to translation, there are infinitely many solutions to this optimization which admit the same minimum: all translations of one optimal solution are also optimal. The constraint $\tilde{\mathbf{v}}_k = \mathbf{c}$ fixes the position of one vertex and makes the solution unique.

In the general case where the source and target have different topologies, the correspondence map \mathcal{M} indicates which triangles of the source and target should deform similarly. It allows us to properly relate the deformation gradients in \mathcal{S} to those in \mathcal{T} even though $|\mathcal{S}| \neq |\mathcal{T}|$:

$$\begin{aligned} \min_{\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n} \quad & \sum_{j=1}^{|\mathcal{M}|} \|\mathbf{S}_{s_j} - \mathbf{T}_{t_j}\|_F^2. \\ \text{subject to} \quad & \tilde{\mathbf{v}}_k = \mathbf{c}. \end{aligned} \tag{3.14}$$

This modified objective function sums over each pair of corresponding triangles specified in \mathcal{M} . Solving this optimization problem finds the deformed target vertices $\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n$ such that each target triangle has deformed as similarly as possible to the corresponding source triangle or triangles. This formulation allows transfer between meshes with different topology since the correspondence map can encode the proper triangle associations. If one target triangle corresponds to several source triangles, the minimized error will include the sum of the norm of the difference between the target deformation gradient and all corresponding source gradients. The result will always be the overall best set of vertices taking into account all correspondences.

3.3.3 Reconstruction Error

When the ideal deformation gradients provide a consistent representation of deformation with respect to T , the optimization procedure finds a deformed target shape where each triangle's change in orientation and stretch exactly matches the ideal specification. For example, if we transfer the source deformation gradients back onto the same source reference mesh so that $T = S$, then deformation transfer perfectly recovers the deformed source such that $\tilde{T} = \tilde{S}$. If the per-triangle gradients in \mathcal{S} are not consistent with respect to T , then no deformed target mesh exists whose deformation gradients in \mathcal{T} match those in \mathcal{S} exactly. In this case, the ideal deformation is only approximated. The minimum found will be a positive number that represents the error incurred when reconstructing the deformed mesh \tilde{T} from the set of source deformation gradients \mathcal{S} .

This value, called the reconstruction error, indicates how well each target deformation gradient matches each corresponding source deformation gradient. However, since the target deformation gradients are found via a least norm QR formulation and the source are computed by explicitly modeling the perpendicular space, the reconstruction error will always be high. The optimization correctly finds the vertex positions that determine the in-plane transformation of each target triangle, but it has no control over the transformation of the perpendicular space. As a result, the perpendicular-space component never matches. The found vertex positions of the deformed target shape are correct since, by definition, each target triangle forms a plane and the transformation of space perpendicular to this plane has no effect on the triangle's vertices. However, the value of the objective function at the minimum is difficult to interpret when it includes the perpendicular-space error and is not an accurate measure of the success or failure of the transfer procedure. We can eliminate the perpendicular-space error in order to obtain a more meaningful measure.

This error is eliminated by modifying the source deformation gradients to include only components that transform the plane of the corresponding target triangle. When the meshes are in one-to-one correspondence (Equation 3.13), the source deformation gradient \mathbf{S}_j for triangle j is applied to the edge vectors of the undeformed target triangle j via $\mathbf{S}_j \mathbf{W}_j$. The original undeformed target triangle and this new deformed one capture the portion of \mathbf{S}_j that acts on the target triangle's plane. Thus, computing the least norm deformation gradient for this undeformed/deformed pair using Equation 3.11 yields a new source deformation gradient \mathbf{S}'_j specialized for the plane of target triangle j . The complete equation is:

$$\mathbf{S}'_j = \mathbf{S}_j \mathbf{W}_j \mathbf{R}_j^{-1} \mathbf{Q}_{j\alpha}^\top = \mathbf{S}_j \mathbf{P}_j. \quad (3.15)$$

When the correspondence map is used (Equation 3.14), the indices s_j and t_j are substituted:

$$\mathbf{S}'_{s_j} = \mathbf{S}_{s_j} \mathbf{W}_{t_j} \mathbf{R}_{t_j}^{-1} \mathbf{Q}_{t_j\alpha}^\top = \mathbf{S}_{s_j} \mathbf{P}_{t_j}. \quad (3.16)$$

The matrices $\mathbf{P}_j = \mathbf{W}_j \mathbf{R}_j^{-1} \mathbf{Q}_{j\alpha}^\top$ and $\mathbf{P}_{t_j} = \mathbf{W}_{t_j} \mathbf{R}_{t_j}^{-1} \mathbf{Q}_{t_j\alpha}^\top$ select only the components of \mathbf{S}_j and \mathbf{S}_{s_j} , respectively, that transform the plane of the target triangle and find the minimum norm solution for the remainder of the matrix.

With these modifications, the reconstruction error can be interpreted as a measure of how well the source deformation is transferred to the target. Lower reconstruction errors indicate that the source deformation gradients, when mapped through the correspondence, provide a largely consistent specification of deformation. In this case, the deformed target is expected to mimic the source deformation faithfully. If the reconstruction error is high, the deformed target poorly approximates the source deformation and artifacts are expected.

3.4 Numerics

The optimization problem in Equation 3.14 is written in an intuitive way. The objective function sums the difference between the deformation gradients of each pair of corresponding source and target triangles. Minimizing this sum finds a deformed target shape whose deformation gradients match the corresponding ones of the source as closely as possible. However, while this formulation provides intuition about the problem, it does not lend itself to an efficient solution. Thus, we exploit the linearity of the deformation gradient representation to rewrite the objective function as a linear system. In this form, we can apply the manifold techniques from linear algebra to solve the system efficiently.

3.4.1 Linearity

The objective function minimized by deformation transfer is based on the per-triangle target deformation gradients. Equation 3.11 relates these gradients to the undeformed and deformed mesh vertices. The undeformed vertices $\mathbf{v}_1 \dots \mathbf{v}_n$ are used to build \mathbf{W}_j (which is then factored into $\mathbf{Q}_{j\alpha} \mathbf{R}_j$) and the

deformed vertices $\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n$ determine $\tilde{\mathbf{W}}_j$ for each triangle j . Each matrix element in the deformation gradient \mathbf{T}_j is a linear function of the deformed vertices $\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n$. Since the relationship between the deformation gradients and the deformed vertices is linear, it can be expressed as a linear system. This allows us to develop a linear operator for a target mesh with n vertices and m triangles that maps the deformed vertices to the per-triangle deformation gradients. Inverting this operator in a least squares sense provides an equivalent formulation of deformation transfer. To derive these formulas, we first pack all deformed vertices and all deformation gradients into two tall column vectors.

The deformed vertices $\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n$ are stacked into one $3n \times 1$ column vector \mathbf{x} :

$$\mathbf{x}_1 = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \quad \mathbf{x}_2 = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} \quad \mathbf{x}_3 = \begin{bmatrix} z_1 \\ z_2 \\ \vdots \\ z_n \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix}. \quad (3.17)$$

Thus, \mathbf{x}_1 , \mathbf{x}_2 , and \mathbf{x}_3 are each $n \times 1$ column vectors containing the x -, y -, and z -coordinates of the vertices, respectively. The $3n \times 1$ column vector \mathbf{x} stacks all three so that the x -coordinates come first, the y -coordinates second, and the z -coordinates third.

Similarly, all deformation gradients $\mathbf{T}_1 \dots \mathbf{T}_m$ are unrolled and stacked into one $9m \times 1$ column vector \mathbf{f} :

$$\mathbf{f}_1 = \begin{bmatrix} \mathbf{T}_1(1, :)^{\top} \\ \mathbf{T}_2(1, :)^{\top} \\ \vdots \\ \mathbf{T}_m(1, :)^{\top} \end{bmatrix} \quad \mathbf{f}_2 = \begin{bmatrix} \mathbf{T}_1(2, :)^{\top} \\ \mathbf{T}_2(2, :)^{\top} \\ \vdots \\ \mathbf{T}_m(2, :)^{\top} \end{bmatrix} \quad \mathbf{f}_3 = \begin{bmatrix} \mathbf{T}_1(3, :)^{\top} \\ \mathbf{T}_2(3, :)^{\top} \\ \vdots \\ \mathbf{T}_m(3, :)^{\top} \end{bmatrix} \quad \mathbf{f} = \begin{bmatrix} \mathbf{f}_1 \\ \mathbf{f}_2 \\ \mathbf{f}_3 \end{bmatrix}. \quad (3.18)$$

The notation $\mathbf{T}_j(k, :)$ indicates the k^{th} row of \mathbf{T}_j , which is a 1×3 row vector. Thus, \mathbf{f}_1 is a $3m \times 1$ column vector whose first three entries are the three elements in the first row of \mathbf{T}_1 , second three entries are the three elements in the first row of \mathbf{T}_2 , and so on. The vectors \mathbf{f}_2 and \mathbf{f}_3 are build in the same fashion and stack the second and third rows of the deformation gradients, respectively. Finally, \mathbf{f} is a $9m \times 1$ column vector containing all unrolled and concatenated deformation gradients in the prescribed order.

Given these definitions, the calculation $T_j = \tilde{W}_j R_j^{-1} Q_{j\alpha}^\top$ for $j \in 1 \dots m$ can be replaced by the linear system:

$$Gx = f. \quad (3.19)$$

The linear operator G is a $9m \times 3n$ matrix whose coefficients depend on the target reference pose vertices $v_1 \dots v_n$ and come from the $R_j^{-1} Q_{j\alpha}^\top$ term in Equation 3.11. G is sparse with a repeated block structure:

$$G = \begin{bmatrix} A & & \\ & A & \\ & & A \end{bmatrix}. \quad (3.20)$$

The $3m \times n$ submatrix A is sparse with three nonzero entries in each row. Expanding Equation 3.19 reveals that A multiplies x_1 , x_2 , and x_3 to give f_1 , f_2 , and f_3 :

$$\begin{bmatrix} A & & \\ & A & \\ & & A \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix}. \quad (3.21)$$

This structure indicates that the computation is separable in the spatial dimension. Individual equations (i.e., rows of G) involve only the x -coordinates, only the y -coordinates, or only the z -coordinates and never mix the coordinates together. Each application of A maps the x -, y -, or z -coordinates to the first, second, or third rows (in vectorized form) of the deformation gradients. The separability allows us to define the $n \times 3$ matrix $X = [x_1 \ x_2 \ x_3]$ and the $3m \times 3$ matrix $F = [f_1 \ f_2 \ f_3]$ and write Equation 3.19 in a more succinct form:

$$AX = F. \quad (3.22)$$

Deformation transfer deals with the case where the source deformation gradients are already known but the deformed target vertices are not. If we assume again that the source and target have the same topology, and use the set of source deformation gradients $\mathcal{S} = \{S_1 \dots S_{|\mathcal{S}|}\}$ to build F , then Equation 3.22 expresses deformation transfer as a system of linear equations. A is build from the target reference pose T and X contains the unknown vertices of the deformed target \tilde{T} . Solving this system for X finds \tilde{T} . Figure 3-6 shows how to setup the linear system $AX = F$ in this case. When the source and target meshes have different topologies, only bookkeeping changes are required to incorporate

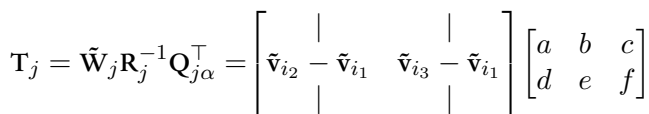


Figure 3-6: **Linear System Construction for Identical Topology.** If the source and target have identical topology with n vertices and m triangles, the linear system $\mathbf{A}\mathbf{X} = \mathbf{F}$ in Equation 3.22 has the structure shown above. Each target triangle $j \in 1 \dots m$ contributes 3 rows to \mathbf{A} . The entries in these rows are the values that multiply the deformed target vertices in $\mathbf{T}_j = \tilde{\mathbf{W}}_j \mathbf{R}_j^{-1} \mathbf{Q}_{j\alpha}^\top$ (Equation 3.11). The coefficients in the three rows for triangle j are shown, where the 6 elements of the 2×3 matrix product $\mathbf{R}_j^{-1} \mathbf{Q}_{j\alpha}^\top$ are represented by the symbols a, b, c, d, e , and f . Since this triangle has vertex indices $\{i_1, i_2, i_3\}$, the coefficients are entered into columns i_1, i_2 , and i_3 . Each group of three rows of the matrix \mathbf{F} contains the transposed source deformation gradient \mathbf{S}_j^\top that represents the ideal differential change of target triangle j .

the correspondence map \mathcal{M} into the construction of \mathbf{A} , \mathbf{X} , and \mathbf{F} . Figure 3-7 describes these changes. Subsequent equations do not differentiate between the construction for one-to-one correspondence and the construction using \mathcal{M} but are valid for both.

3.4.2 Solution

Since \mathbf{A} has more rows than columns, the linear system in Equation 3.22 is overdetermined. This fact reflects the consistency requirement discussed in Section 3.3: for an arbitrary (inconsistent) set of ideal deformation gradients, there is no deformed target mesh whose actual deformation gradients match them. However, we find the mesh that matches them as closely as possible by solving the linear system in a least-squares sense. Because the computation is separable, the least-squares formulation can be written as three independent optimization problems for the columns of \mathbf{X} :

$$\begin{aligned} \min_{\mathbf{x}_i} \quad & \|\mathbf{A}\mathbf{x}_i - \mathbf{f}_i\|_2^2, \quad i \in 1 \dots 3 \\ \text{subject to} \quad & \tilde{\mathbf{v}}_k^i = \mathbf{c}^i \end{aligned} \tag{3.23}$$

The notation $\tilde{\mathbf{v}}_k^i = \mathbf{c}^i$ indicates that the i^{th} component of $\tilde{\mathbf{v}}_k$ is constrained to equal the i^{th} component of \mathbf{c} . Equation 3.23 is equivalent to the summation formulation in Equation 3.13 if the construction for identical topology is used or in Equation 3.14 when using the construction for different topology. The multiplication $\mathbf{A}\mathbf{x}_i$ computes the target deformation gradients $\mathcal{T} = \{\mathbf{T}_1 \dots \mathbf{T}_{|\mathcal{T}|}\}$, the subtraction $\mathbf{A}\mathbf{x}_i - \mathbf{f}_i$ subtracts the actual target deformation gradients and the ideal source ones, and the l^2 norm $\|\cdot\|_2$ performs the function of both the Frobenius norm and the summation. Although the computation has been reordered, the objective function in Equation 3.23 (or, more precisely, the sum of the three objective functions) evaluates to exactly the same value as in the summation formulation.

Equation 3.23 describes three linearly constrained quadratic optimization problems that can be solved separately, one by one. Each solution is found in two steps. First, the constrained optimization problem is transformed into an equivalent unconstrained one. Then, the minimum is found by taking the derivative of the objective function with respect to the unknowns and setting it equal to zero.

The constraint $\tilde{\mathbf{v}}_k^i = \mathbf{c}^i$ is enforced by treating the constrained vertex as a constant rather than a free variable. If we assume, without loss of generality, that $k = n$ so that the constrained vertex has

index n , then we form the equivalent unconstrained optimization problems:

$$\min_{\hat{\mathbf{x}}_i} \|\hat{\mathbf{A}}\hat{\mathbf{x}}_i - \hat{\mathbf{f}}_i\|_2^2, \quad i \in 1 \dots 3. \quad (3.24)$$

The modified matrix $\hat{\mathbf{A}}$ is void of column n and $\hat{\mathbf{x}}_i$ is void of element n . The removed column is multiplied by the constrained value and subtracted from \mathbf{f}_i , yielding $\hat{\mathbf{f}}_i = \mathbf{f}_i - \mathbf{c}^i \hat{\mathbf{A}}(:, n)$. Although constraining one vertex is required in order to resolve the translation invariance, any number of vertices can be constrained in this manner. Each constraint removes one variable from \mathbf{x}_i and deletes the column in \mathbf{A} that multiplies it. The result of this multiplication is subtracted from \mathbf{f}_i . Vertex constraints give limited control over the transfer process. For example, in Figure 3-14 (Section 3.6), constraints on the hoofs of a galloping camel are used to enforce ground contact.

Once the constrained optimization problem has been transformed into an unconstrained one, it is solved by expanding the matrix norm in the objective function, taking the derivative with respect to \mathbf{x}_i , and setting the derivative equal to zero. For notational simplicity, the hats ($\hat{}$) are not included in the following formulas. All of the matrices and vectors should, however, be properly built to handle any constrained vertices.

Expanding the matrix norm gives

$$\begin{aligned} \|\mathbf{A}\mathbf{x}_i - \mathbf{f}_i\|_2^2 &= (\mathbf{A}\mathbf{x}_i - \mathbf{f}_i)^\top (\mathbf{A}\mathbf{x}_i - \mathbf{f}_i) \\ &= \mathbf{x}_i^\top \mathbf{A}^\top \mathbf{A} \mathbf{x}_i - \mathbf{x}_i^\top \mathbf{A}^\top \mathbf{f}_i - \mathbf{f}_i^\top \mathbf{A} \mathbf{x}_i - \mathbf{f}_i^\top \mathbf{f}_i \\ &= \mathbf{x}_i^\top \mathbf{A}^\top \mathbf{A} \mathbf{x}_i - \mathbf{x}_i^\top \mathbf{A}^\top \mathbf{f}_i - (\mathbf{f}_i^\top \mathbf{A} \mathbf{x}_i)^\top - \mathbf{f}_i^\top \mathbf{f}_i \\ &= \mathbf{x}_i^\top \mathbf{A}^\top \mathbf{A} \mathbf{x}_i - 2\mathbf{x}_i^\top \mathbf{A}^\top \mathbf{f}_i - \mathbf{f}_i^\top \mathbf{f}_i. \end{aligned} \quad (3.25)$$

Taking the derivative¹ of this expression with respect to \mathbf{x}_i yields

$$\frac{d}{d\mathbf{x}_i} \left(\mathbf{x}_i^\top \mathbf{A}^\top \mathbf{A} \mathbf{x}_i - 2\mathbf{x}_i^\top \mathbf{A}^\top \mathbf{f}_i - \mathbf{f}_i^\top \mathbf{f}_i \right) = 2\mathbf{A}^\top \mathbf{A} \mathbf{x}_i - 2\mathbf{A}^\top \mathbf{f}_i. \quad (3.26)$$

Finally, setting the derivative equal to zero gives the normal equations:

$$\mathbf{A}^\top \mathbf{A} \mathbf{x}_i = \mathbf{A}^\top \mathbf{f}_i. \quad (3.27)$$

¹Minka [2000] gives a thorough discussion of vector and matrix derivatives.

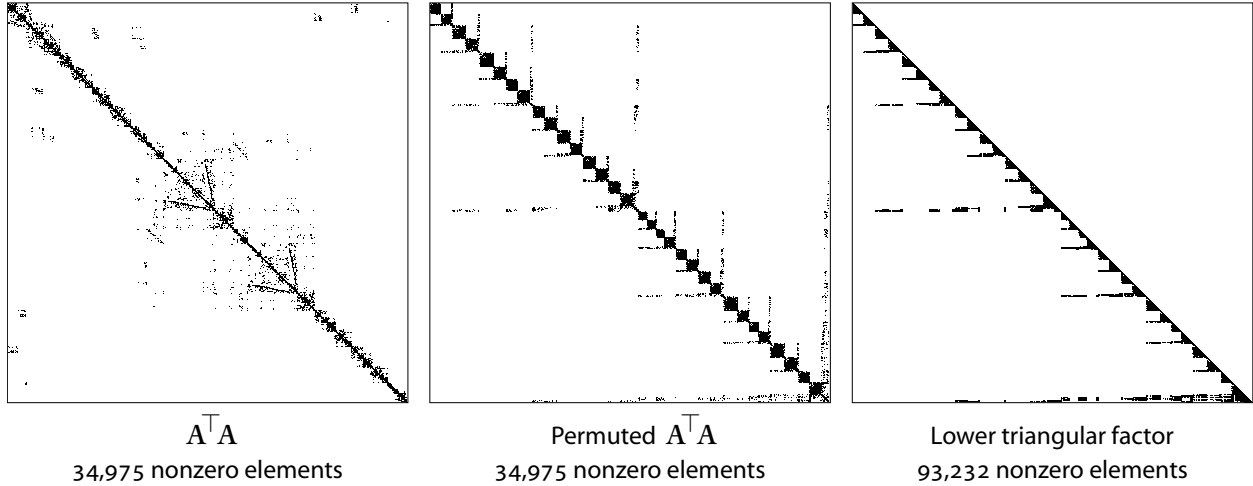


Figure 3-8: The nonzero structure of the matrix $A^T A$ for the lion mesh from Figure 3-1 is shown on the left. The mesh has 5,000 vertices and 9,996 triangles. One vertex constraint is used, so $A^T A$ has dimensions $4,999 \times 4,999$. An entry a_{pq} in $A^T A$ is nonzero if vertex p and vertex q are neighbors or if $p = q$. Prior to factorization, $A^T A$ is permuted using METIS [Karypis and Kumar 1998] to reduce nonzero fill-in. Both $A^T A$ and the permuted version, shown in the center, have 34,975 nonzero elements. The lower triangular factor of the Cholesky factorization, shown on the right, is computed using TAUCS [Toledo 2003] and has 93,232 nonzero elements. Transferring a source deformation to the lion involves performing backsubstitution with the factored matrix using a right-hand side derived from the source.

$A^T A$ is an $n \times n$ sparse matrix, where n is the number of unconstrained vertices. An element a_{pq} of $A^T A$ is nonzero only if target vertices p and q are neighbors or $p = q$.

Care must be taken in order to solve Equation 3.27 efficiently. Because $A^T A$ is symmetric and positive definite, we can apply Cholesky factorization and then solve for each right-hand side by reusing the factorization during backsubstitution. In fact, since the elements of A depend only on the target reference pose T and correspondence map \mathcal{M} , $A^T A$ can be factored once and reused for any new source deformation. This permits an entire animation sequence to be retargeted efficiently since the factored system is reused to transfer the deformation in every frame. However, naïve factorization will not preserve sparsity in the factored system. As a result, a permutation algorithm must be used in order to reduce nonzero fill-in. Botsch, Bommes, and Kobbelt [2005] discuss the performance of possible matrix reorderings as well as many practical issues related to solving sparse linear systems. The nonzero structure of $A^T A$ before and after permutation as well as its lower triangular factor for the lion mesh in Figure 3-1 is shown in Figure 3-8.

I have experimented with three direct sparse matrix solvers including UMFPACK v4.1 [Davis

2003], PARDISO included with Intel MKL v7.1 [Intel 2004], and TAUCS v2.2 [Toledo 2003]. UMFPACK performs LU-decomposition, while PARDISO and TAUCS perform Cholesky factorization. COLAMD [Davis et al. 2004] is used for fill-reducing reordering with UMFPACK and METIS [Karypis and Kumar 1998] is used for reordering with PARDISO and TAUCS. In my informal experience, all three packages show comparable performance. A recent technical report by Gould, Hu, and Scott [2005] provides a formal comparison of these and other sparse direct solvers.

3.5 Analytic Derivation

An appropriate specification of deformation is essential to the success of deformation transfer. The choice to use deformation gradients is motivated by their application in continuum mechanics to represent large deformations that include both rigid rotation and non-rigid stretching. An approximation of the deformation gradient tensor field on the surface of a mesh avoids the drawbacks of simplicial dissection. Since the source deformation gradients are inconsistent with respect to the target shape, reconstructing the Cartesian representation of a mesh from the modified deformation gradients is naturally expressed as a variational problem in order to cope with the inconsistencies as well as possible precision errors.

In contrast to a variational formulation, mesh representations based on equations in strong form produce successful results for mesh editing and interpolation. In these methods, the Laplace equation, the Poisson equation, or the first and second fundamental forms are discretized on the mesh and solved to generate an edited or interpolated result (Section 2.1.2). A closer connection between these methods and deformation gradients is discovered by considering the continuous analog of the reconstruction problem used in deformation transfer.

The function $U : \mathbb{R}^3 \rightarrow \mathbb{R}^3$ defined in Equation 3.1 maps points of an undeformed surface Ω to their deformed positions. The deformation gradient tensor field $\frac{\partial U}{\partial \mathbf{p}}$ is written as

$$\frac{\partial U}{\partial \mathbf{p}} = \begin{bmatrix} (\nabla U_1)^\top \\ (\nabla U_2)^\top \\ (\nabla U_3)^\top \end{bmatrix}, \quad (3.28)$$

where the gradients of the three coordinate functions U_1 , U_2 , and U_3 form the rows of the matrix.

The deformation transfer reconstruction problem for identical connectivity in Equation 3.13 can be rewritten in continuous form yielding:

$$\min_{\mathbf{U}} \iint_{\Omega} \left\| \begin{bmatrix} (\nabla U_1)^\top \\ (\nabla U_2)^\top \\ (\nabla U_3)^\top \end{bmatrix} - \begin{bmatrix} \mathbf{s}_1^\top \\ \mathbf{s}_2^\top \\ \mathbf{s}_3^\top \end{bmatrix} \right\|_F^2 = \min_{\mathbf{U}} \iint_{\Omega} \left\| \begin{bmatrix} (\nabla U_1)^\top - \mathbf{s}_1^\top \\ (\nabla U_2)^\top - \mathbf{s}_2^\top \\ (\nabla U_3)^\top - \mathbf{s}_3^\top \end{bmatrix} \right\|_F^2. \quad (3.29)$$

The sum over each triangle in the original equation is replaced by an integral over the surface. At each point on the surface, the difference between the deformation gradient and a “guidance function” expressed as a matrix composed of the three rows \mathbf{s}_1^\top , \mathbf{s}_2^\top , and \mathbf{s}_3^\top is computed. The guidance function corresponds to the source deformations. Both the deformation gradient and guidance function vary with position on the surface. Equation 3.29 is equivalent to

$$\min_{\mathbf{U}} \iint_{\Omega} \left(\|\nabla U_1 - \mathbf{s}_1\|_2^2 + \|\nabla U_2 - \mathbf{s}_2\|_2^2 + \|\nabla U_3 - \mathbf{s}_3\|_2^2 \right), \quad (3.30)$$

where the Frobenius norm of the matrix is replaced by the sum of the Euclidean norms of three vector expressions. Since each of the three terms in the sum must be positive, the entire expression achieves its minimum when the individual terms are minimized:

$$\min_{\mathbf{U}} \iint_{\Omega} \|\nabla U_1 - \mathbf{s}_1\|_2^2 \quad \min_{\mathbf{U}} \iint_{\Omega} \|\nabla U_2 - \mathbf{s}_2\|_2^2 \quad \min_{\mathbf{U}} \iint_{\Omega} \|\nabla U_3 - \mathbf{s}_3\|_2^2. \quad (3.31)$$

This decomposition is expected since we know the problem is separable in the three spatial dimensions. Each of these variational problems has a corresponding strong form [Weinstock 1974]:

$$\nabla^2 U_1 = \nabla \cdot \mathbf{s}_1 \quad \nabla^2 U_2 = \nabla \cdot \mathbf{s}_2 \quad \nabla^2 U_3 = \nabla \cdot \mathbf{s}_3, \quad (3.32)$$

where constraints in the original formulation (Equation 3.13) serve as Dirichlet boundary conditions on the partial differential equations.

This derivation demonstrates that the deformation gradient reconstruction problem can be reformulated as a Poisson equation. The Laplacian of the deformation coordinate functions makes up the left-hand side and the divergence of the guidance function comprises the right-hand side.

Mesh	Vertices	Triangles
Horse	8,431	16,843
Camel	21,887	43,814
Cat	7,200	14,410
Lion	5,000	9,996
Face	29,299	58,836
Head	15,941	31,620
Flamingo	26,907	52,895
Elephant	42,321	84,638

Table 3.1: The number of vertices and triangles for the meshes used in the examples.

Example	Permutation & factorization	Per-pose computation
Horse/Camel	0.435 s	0.088 s
Cat/Lion	0.072 s	0.024 s
Face/Head	0.693 s	0.102 s
Horse/Flamingo	1.781 s	0.111 s
Horse/Elephant	3.593 s	0.190 s

Table 3.2: Timing results on a 3.2GHz Pentium IV machine. The second column includes the precomputation time to permute and factor $A^T A$ and the third column includes the time to compute $A^T f_i$ and solve for $x_i, i \in 1 \dots 3$, using backsubstitution.

A comparison with Equation 2.2 suggests an equivalence between deformation gradients and the Poisson mesh editing method [Yu et al. 2004; Xu et al. 2005]. Understanding the precise relationship as well as any connections to other mesh algorithms may lead to improvements on both sides.

3.6 Results and Discussion

To demonstrate both the success and limitations of deformation transfer, I present results in a variety of situations. Examples include the transfer of key poses that are primarily kinematic in nature as well as the opposite extreme of completely non-rigid deformations. Pose transfer naturally extends to animation by transferring the deformation exhibited in each frame. I challenge the technique with characters that are more dissimilar where forming an adequate correspondence is difficult. Finally, I show that deformations that depend on fine-scale surface details are poor candidates for my method.

Table 3.1 lists geometric information about each model, and Table 3.2 gives timing results for each example. None of the source and target meshes in these examples share the same number of vertices, triangles, or connectivity. Deformation transfer is extremely fast. For example, in the horse/camel transfer, the camel mesh has 21,887 vertices yielding a $21,886 \times 21,886$ normal equations matrix (with one vertex constraint). In this case, reordering and Cholesky factorization requires 0.435 seconds and solving for each retargeted pose takes only 0.088 seconds on a 3.2GHz Pentium IV machine.

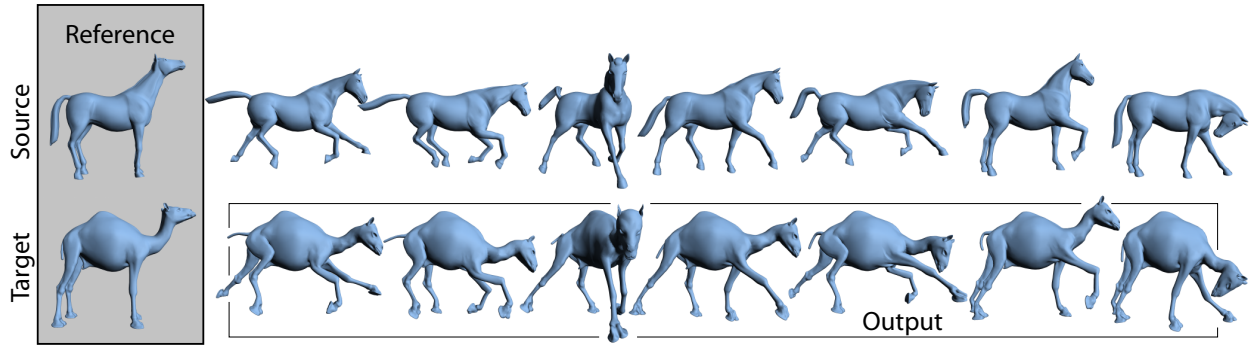


Figure 3-9: Deformation transfer copies the deformations exhibited by a source mesh onto a different target mesh. In this example, deformations of the reference horse mesh are transferred to the reference camel, generating seven new camel poses. Both gross skeletal changes as well as more subtle skin deformations are successfully reproduced.

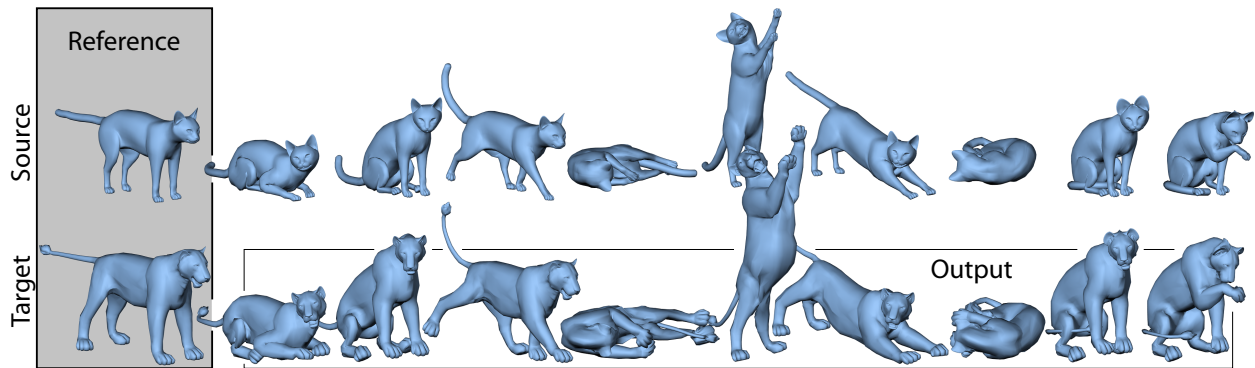


Figure 3-10: Cat poses retargeted onto a lion.

3.6.1 Kinematic Poses

Figure 3-9 shows deformations of a horse transferred onto a camel. The reference horse mesh, shown in the gray box, is deformed into seven key poses. The key poses are primarily kinematic in nature, although they also include more subtle skin deformations like stretching near the joints. The input to the algorithm is the reference horse mesh, the seven deformed horse poses, the reference camel mesh, and the correspondence between the two reference meshes. Given this data, deformation transfer generates seven new camel poses by transferring the source deformations onto the reference camel mesh. Both the gross skeletal changes as well as the more subtle skin deformations are faithfully reproduced. Figure 3-10 demonstrates a similar set of deformations. Here, key poses of a cat are retargeted onto a lion. Even extreme changes where the cat lies down and curls its body are perfectly matched by the lion.

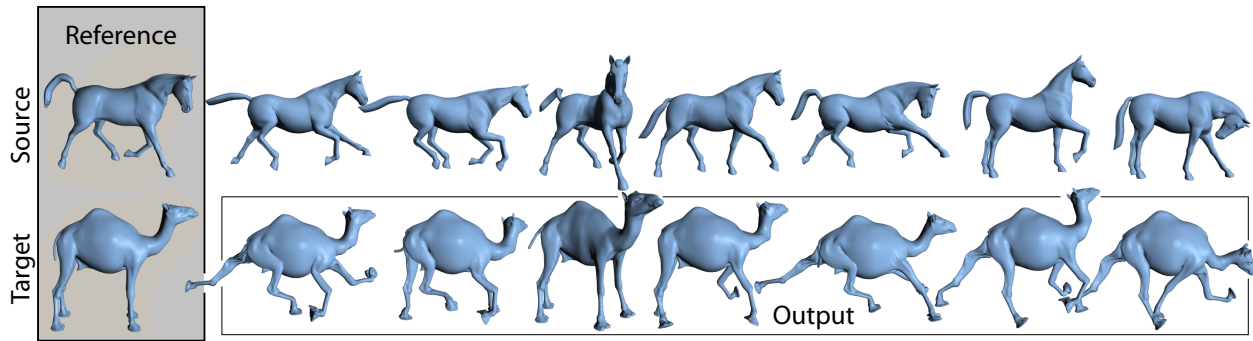


Figure 3-11: Deformation transfer copies the *change* in shape from the source to the target and thus requires matching reference poses. If the posture of the reference meshes does not match, the result is unnaturally distorted. The same transfer with matching meshes is shown in Figure 3-9.

Since deformation transfer copies the *change* in shape induced by the deformation, the source and target reference meshes should have the same kinematic pose when skeletal deformations are retargeted. If transfer is applied to reference poses that do not match, the results are undesirable and distorted, as demonstrated in Figure 3-11. This example includes the same seven source poses as shown in Figure 3-9 but replaces the source reference pose with one in which the horse's legs are bent and tail, body, and neck are arched. The output camel poses are unnaturally distorted. This restriction limits the applicability of deformation transfer to cases where matching kinematic poses of the source and target are available. However, in many cases, the source poses are generated via some form of rigging. The source mesh can be reposed using the rigging controls to match the posture of a given target shape. Source deformations can then be copied onto the target.

3.6.2 Non-rigid Deformations

While the deformations in Figures 3-9 and 3-10 are primarily skeletal in nature, Figure 3-12 demonstrates the effectiveness of deformation transfer on non-rigid deformations. Here, the horse collapses as if it were made of a rubber sheet. Its legs buckle and its entire body falls to the ground, folding on top of itself. The deformations are transferred to the camel, and its body buckles and collapses similarly.

In Figure 3-13, facial expressions of a real person, acquired with a 3D scanning system, are transferred onto a digital character. A great deal of expressiveness—especially around the eyes and nose—is captured and adapted to the target head. This type of transfer might be used when a digital stand-in must replace a real actor, or to map the facial expressions of a voice actor onto an animated character.

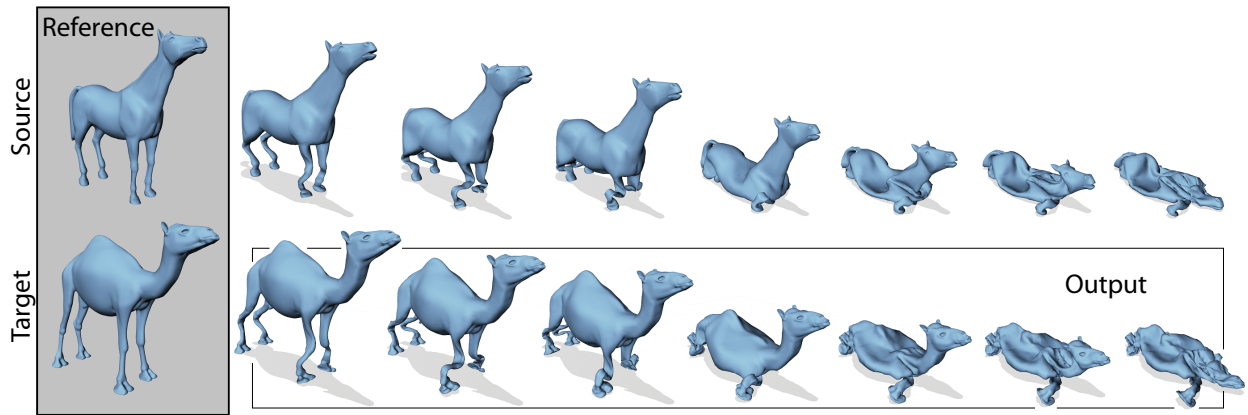


Figure 3-12: The horse deformation, collapsing as if it were made of a rubber sheet, is transferred to the camel.

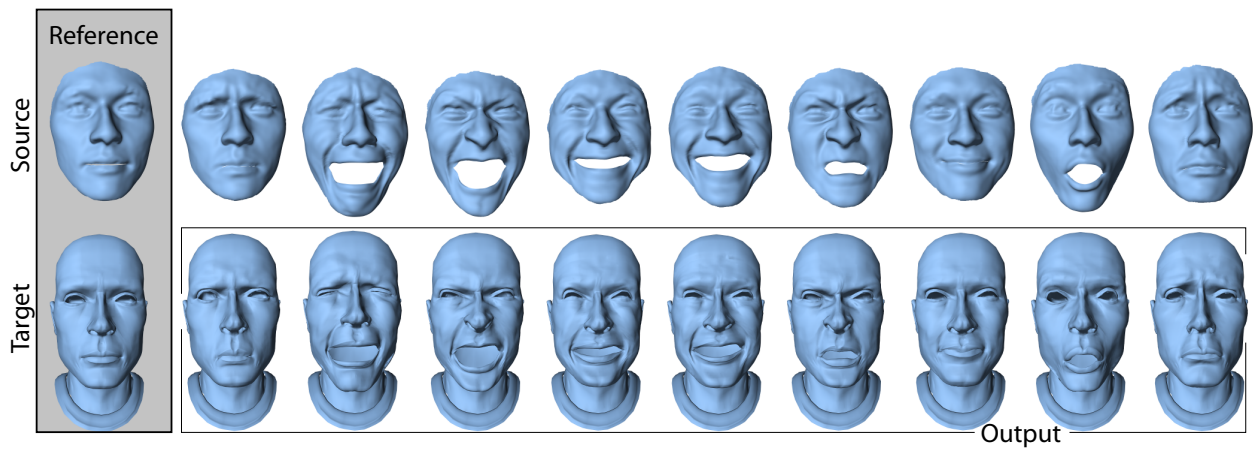


Figure 3-13: Scanned facial expressions are cloned onto a digital character.

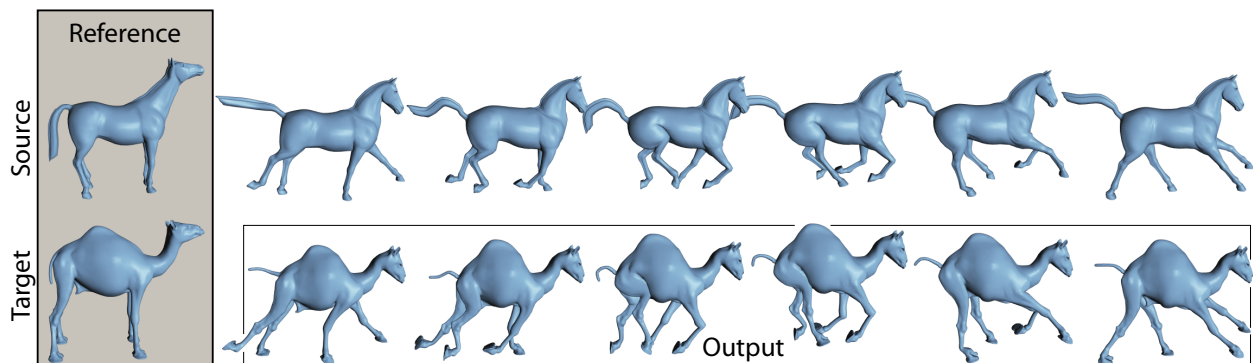


Figure 3-14: An entire animation can be retargeted by transferring the deformation in each frame to the target shape. In this example, six frames from a longer horse gallop animation are transferred to the camel. Because deformation transfer is linear, temporal consistency in the source deformation yields a temporally coherent output animation.

Since the scanned data is in the form of face masks and the target mesh consists of an entire head and neck, the mapping between the source and target triangles is not onto. Only a subset of the target triangles (those of the front of the face) are listed in the correspondence. The deformation of the remaining target triangles is minimized by mapping them to the identity matrix.

3.6.3 Animation Retargeting

Deformation transfer can also retarget deformations that vary continuously through time. Figure 3-14 shows a gallop gait transferred from the horse to the camel. An animation of the collapsing motion from Figure 3-12 is also transferred to the camel. In order to resolve the global positioning of the camel over time and to enforce foot/ground contact, I extracted the positions of one vertex on each foot of the horse over time, performed an overall scaling to better match the larger size of the camel, and added vertex constraints to match a vertex on each camel foot to these positions. Deformation transfer then copies the horse deformation onto the camel while simultaneously satisfying the vertex constraints. Because deformation transfer is a linear operation, temporal consistency of the source animation and vertex constraints results in a temporally coherent target animation.

The use of constraints in animation retargeting brings up another limitation of deformation transfer. The optimization problem is unique up to a global translation and requires at least one vertex position to be specified by the user. When retargeting only key poses, as in Figures 3-9 and 3-10, it is easy to resolve the global position by fixing one vertex in place. However, when retargeting an entire animation sequence, the position must be specified at each point in time. A positional constraint specified only during selected intervals (such as during ground contact) will result in “popping” artifacts when the constraint becomes active. Since the overall shape of the horse and camel is similar, it is easy to derive appropriate constraints for the camel by a global scaling of the horse mesh. But, if two meshes have very different proportions, it may be more difficult to formulate an acceptable constraint.

One approach to this problem is to directly address the temporal dimension in the retargeting process in order to enforce temporal coherence as well as reduce the global positioning problem to the specification of positional constraints only during key events such as foot/ground contact. Gleicher [1998b] uses this approach to retarget skeletal motion to new skeletons with different bone lengths. However, a formulation of deformation transfer as an optimal trajectory problem would increase the numerical complexity of the optimization considerably.

3.6.4 Dissimilar Characters

I designed deformation transfer for the case where there is a clear semantic correspondence between the source and target. Anatomically similar meshes have an obvious mapping (i.e., legs to legs, head to head, etc.). I challenge this assumption by mapping the horse poses onto more dissimilar characters. Figure 3-15 shows transfer onto a flamingo mesh. The correspondence is ambiguous as the flamingo has only two legs, no tail, and a beak. I map the flamingo’s legs to the horse’s front two legs, the flamingo’s body to the entire horse’s body including its tail, and its beak to the horse’s head. Building this mapping pushes the limits of the correspondence system described in Chapter 4. But, once the correspondence has been adequately specified, the flamingo faithfully deforms like the horse. However, a real flamingo’s hips bend in the opposite direction of a horse’s front hips, which demonstrates a reason why deformation transfer between anatomically different meshes may not be appropriate.

In Figure 3-16, I transfer the horse poses onto an elephant mesh. Although the horse and elephant are both quadrupeds, both the gross shape and the topology of the two meshes are quite different. The elephant has enormous ears, a long curved trunk, and five times as many vertices and triangles. For this example, a different method is used to compute the triangle associations [Kraevoy and Sheffer 2004]. As with the flamingo, once the proper correspondence is specified, the transfer algorithm works well. Although the generated elephant deformations look plausible, the horse/elephant transfer demonstrates that the formulation does not prevent self-intersections. In the fifth and seventh output poses (enlarged in Figure 3-17), the elephant’s trunk intersects its leg.

3.6.5 Detail-Dependent Deformations

When the nature of the source deformation is highly dependent on specific details of the source shape that are not present in the target, deformation transfer performs poorly. Consider the 2D example in Figure 3-18. The reference source shape is “inflated” to form a circle. When this deformation is transferred to a similar but different target shape, the result looks nothing like inflation. Figure 3-19 shows an analogous example in 3D. In the top row, the cat mesh is inflated using a nonlinear algorithm that attempts to make every portion of the mesh locally as flat as possible. The middle row shows the result of deformation transfer onto the lion mesh. The lion mesh increases in size and seems to inflate, but does not have the balloon-like characteristics of the source deformation. The lion’s feet become overly

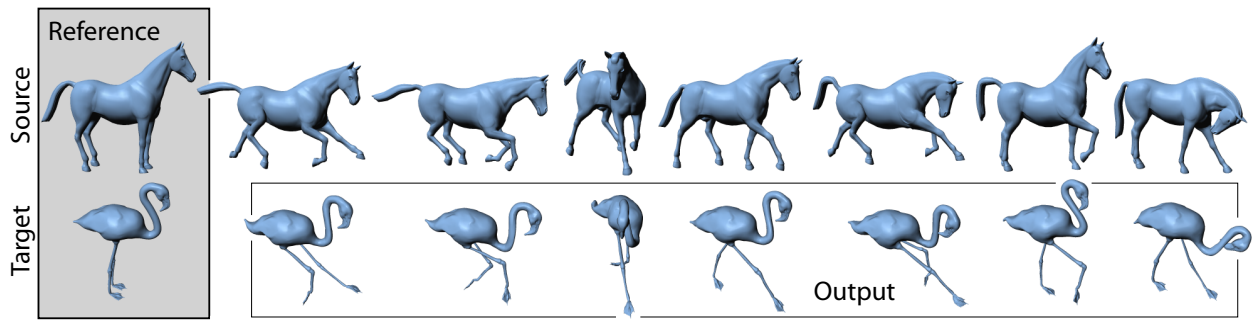


Figure 3-15: Horse poses mapped onto a flamingo. The correspondence is less obvious because the flamingo has only two legs, no tail, and a beak.

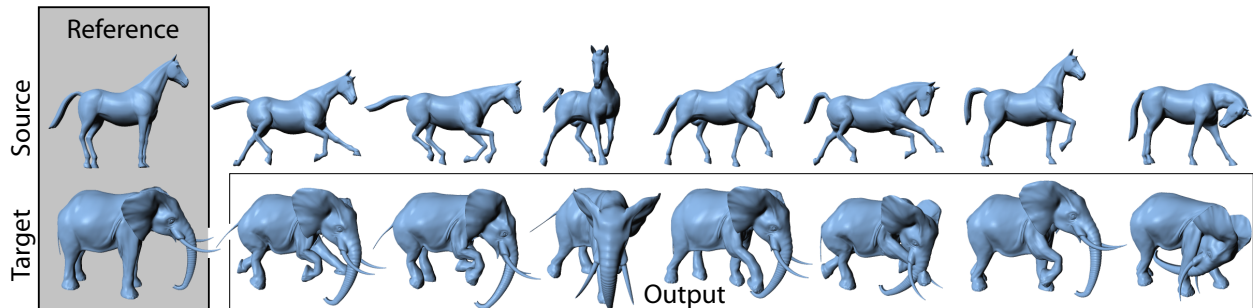


Figure 3-16: Horse poses mapped onto an elephant. The elephant's enormous ears and trunk make the correspondence difficult.

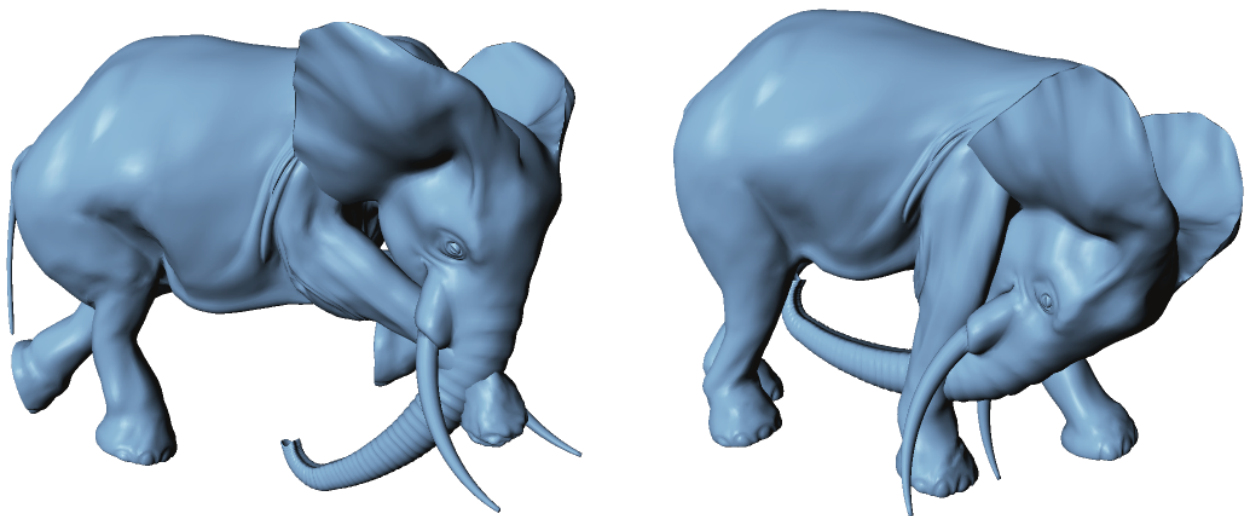


Figure 3-17: The fifth and seventh output poses from Figure 3-16 are enlarged to show that deformation transfer does not prevent self-intersections.

distorted, and other parts of the body have an exaggerated angular appearance. Running the nonlinear inflation deformer directly on the reference lion mesh yields a result truer to the source deformation. In both Figures 3-18 and 3-19, deformation transfer fails because the nature of the deformation depends on fine scale surface details of the mesh. Since these details differ between the source and target, directly transferring the change in source shape to the target does not produce the desired result.

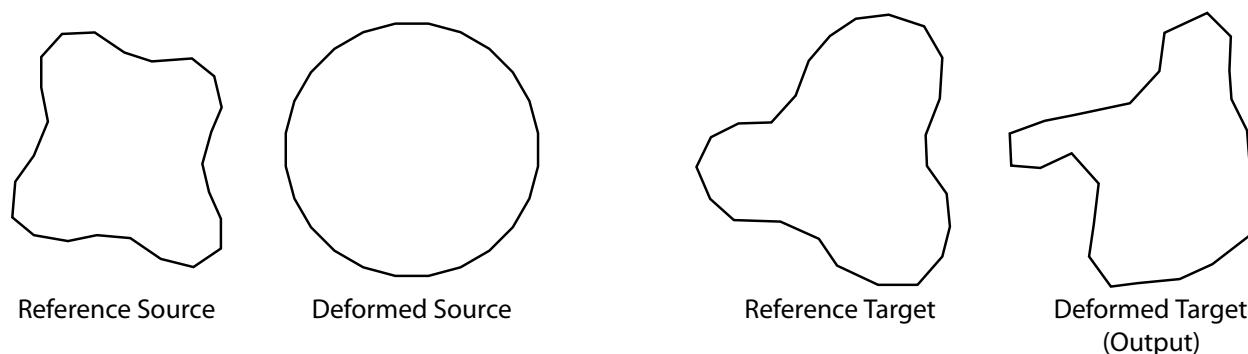


Figure 3-18: The 2D source shape is inflated to form a circle. When the deformation is transferred onto a different target shape, the deformed target does not appear inflated.

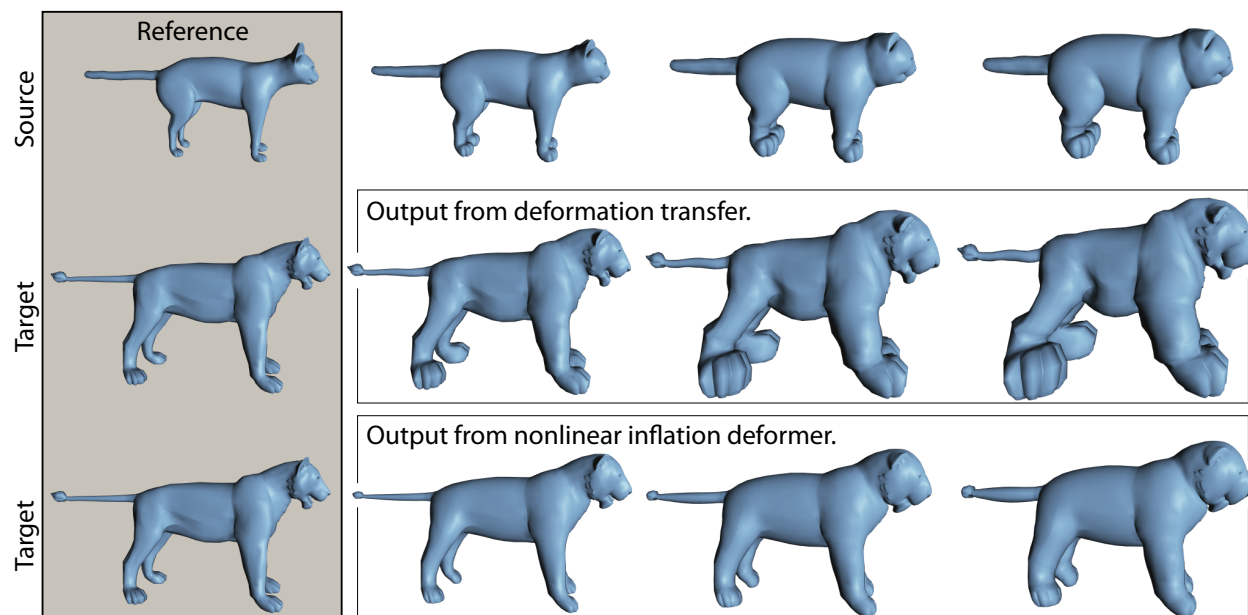


Figure 3-19: (Top) The cat mesh is inflated using a nonlinear inflation deformer. (Middle) When these deformations are transferred to the lion mesh, the output appears distorted and does not capture the balloon-like nature of the source deformation. (Bottom) Applying the nonlinear inflation algorithm directly to the lion mesh reproduces the characteristics of the source deformation more faithfully.

Correspondence

4

Different characters created for computer animation rarely share the same mesh topology. Some meshes require a fine tessellation in order to represent small geometric features, while others use triangles more sparingly to encode flatter regions. Furthermore, large scale differences may exist in the gross anatomical structure of two characters. Because of these differences, correspondence is an integral component of the deformation transfer algorithm described in the previous chapter. Reusing existing animations on new characters requires some way to relate the mesh structure of one character to the structure of another. In this chapter, I discuss my solution to the correspondence problem which enables deformation transfer between disparate meshes.

The need for correspondence is not unique to deformation transfer. In fact, it stretches across numerous problem domains in computer graphics and vision. Many applications require a one-to-one correspondence in mesh structure and resample all input meshes to achieve the same connectivity. These applications include, among others, statistical models of shape variation in human faces [Blanz and Vetter 1999; Vlasic et al. 2005] or body shape [Allen et al. 2003; Seo and Magnenat-Thalmann 2003; Seo et al. 2003; Allen et al. 2004; Anguelov et al. 2005], shape interpolation [Alexa et al. 2000], shape blending [Kraevoy and Sheffer 2004], and texture and detail transfer [Praun et al. 2001]. Nonrigid registration of 3D scans must find a correspondence between points on one scan and points on another [Anguelov et al. 2004]. Multiresolution employs correspondence to relate detail coefficients between different levels of resolution even when each level has an arbitrary vertex connectivity [Kobbelt et al.

2000]. Expression cloning [Noh and Neumann 2001] requires correspondence between a source and target face mesh. In computer vision, the correspondence problem has a long history ranging from stereo imaging [Scharstein and Szeliski 2002] to object recognition and retrieval [Berg et al. 2005].

Because correspondence is addressed in so many different domains, there are a variety of ways to approach it. Perhaps the most formal method is a parameterization of two meshes with respect to each other in the form of a bijective mapping between all points on one mesh and all points on the other. This function is referred to as a cross-parameterization [Kraevoy and Sheffer 2004] or inter-surface mapping [Schreiner et al. 2004]. Techniques to compute such a bijection are robust to large shape differences and require little user input. However, they can only be applied when a bijection actually exists. Thus, the input meshes must be manifold and of the same genus. This requirement restricts the candidates for deformation transfer. For example, the camel mesh (Figure 3-9) has pierced ears, making it genus two and therefore incompatible with the horse. Furthermore, both the horse (Figure 3-9) and cat meshes (Figure 3-10) have connectivity “errors” that make them non-manifold. Unfortunately, topological problems such as these are commonplace and limit the applicability of parameterization-based algorithms.

Template fitting is a common approach to bring meshes into one-to-one correspondence [Blanz and Vetter 1999; Allen et al. 2003; Seo and Magnenat-Thalmann 2003; Angelov et al. 2005]. A single template mesh is constructed that approximates the features of a collection of meshes not in correspondence. For example, if the collection consists of human faces, the template could be a “generic” face with gross facial features such as eyes, nose, and mouth but without details specific to a particular individual. During fitting, a copy of the template is repeatedly deformed to match the shape of each mesh in the collection. The meshes are discarded and replaced by the deformed copies of the template. Applying such a procedure in deformation transfer to bring the source and target into one-to-one correspondence simplifies transfer by obviating the need for a correspondence map. However, changing the mesh structure may not be an option in practical situations due to design requirements. In other cases, template fitting may not be appropriate because of large disparities in the source and target mesh shape. For example, in Figure 3-13, the source is a “face mask” comprising only the front part of the face, while the target represents the entire head. Finding a common topology for these two meshes would be difficult.

A different approach finds a set of discrete correspondence pairs by solving a combinatorial optimization problem. A probabilistic formulation of this approach maximizes the joint probability of all possible matching pairs of points using a model based on local geometry and the proximity of the matched pairs [Anguelov et al. 2004]. In computer vision, correspondences within images are found using integer quadratic programming [Berg et al. 2005], min-cut and max-flow algorithms [Boykov and Kolmogorov 2001], and belief propagation [Sun et al. 2003]. The strength of these formulations is their expressiveness. A discrete correspondence can express arbitrary matchings between two shapes regardless of topological differences. However, combinatorial optimization is difficult to perform efficiently. For example, integer quadratic programming is NP-Complete and domain-specific knowledge must be incorporated to find an approximate solution [Berg et al. 2005].

In order to be maximally general, deformation transfer employs a discrete triangle pairing for correspondence. The correspondence map \mathcal{M} , discussed in Section 3.2, associates triangles of the source and triangles of the target by storing pairs of source and target triangle indices:

$$\mathcal{M} = \{(s_1, t_1), (s_2, t_2), \dots, (s_{|\mathcal{M}|}, t_{|\mathcal{M}|})\}. \quad (4.1)$$

A pair (s_j, t_j) indicates that the target triangle with index t_j should deform like the source triangle with index s_j . Since only triangle indices are involved, differences in genus or errors in the vertex connectivity pose no problem. No restrictions are placed on the type of mapping that \mathcal{M} represents. It can be, and almost always is, a general many-to-many mapping. A single target triangle can be included in the mapping again and again, each time paired with a different source triangle, and vice versa. This flexibility allows meshes of different structure and tessellation to be related to each other. When the source contains features whose deformation should not be reproduced on the target, the source triangles can be left out of the correspondence altogether. For example, the map used to generate the horse/flamingo transfer in Figure 3-15 matches the horse's front legs with the flamingo's legs and simply ignores the horse's back legs. Special source index tags can be used to indicate that a target triangle should deform according to some canonical deformation gradient such as the identity transform. This technique is used in the map for the face/head transfer of Figure 3-13. The target's face is matched with the source face, while the remaining target triangles that comprise the rest of the head and neck are paired with the identity transform so that their deformation is minimal.

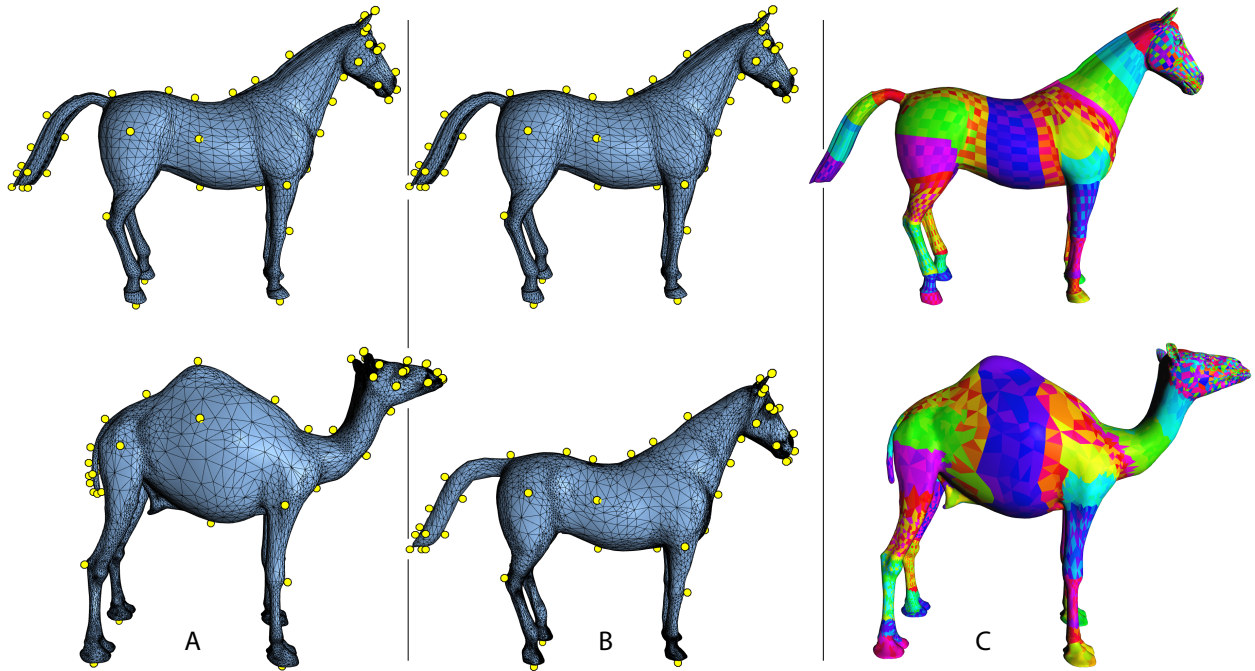


Figure 4-1: An overview of the correspondence system. (A) The user selects marker points, shown as yellow circles, on the source and target characters that indicate matching features. (B) Using a template fitting algorithm, the camel mesh is deformed to match the shape of the horse while preserving its original mesh structure. (C) Once the two meshes have a similar shape, a discrete correspondence is computed by searching for pairs of source and target triangles whose centroids are in close proximity. After selecting an arbitrary colormap for the horse mesh, the discrete mapping is visualized by assigning each camel triangle the color of its corresponding horse triangle. If a camel triangle corresponds to multiple horse triangles, only one is used for the visualization.

While the format of the source/target correspondence is discrete, the algorithm I propose to compute it uses a template fitting procedure to find a continuous deformation of one mesh into the other. The source and target meshes play the roles of template and destination during fitting. An iterated closest point algorithm with regularization controlled by user selected marker points fits the template mesh to the destination. In this process, the template retains its original mesh structure but its vertices are moved so that its shape resembles that of the destination. The deformed template mesh provides an approximate parameterization of the template with respect to the destination since each template vertex is mapped to a position in space near the destination shape. Once the deformation is computed, triangle pairings in \mathcal{M} are found by searching for template and destination triangles whose centroids are in close proximity. Thresholds on distance and surface properties give additional control over the final triangle pairing. The correspondence process is summarized in Figure 4-1. The template fitting and triangle pairing algorithms are described in more detail in the following two sections.

Finding the correspondence map for two new characters using this procedure is an interactive process that requires about one hour of a user’s time. Typically, the user begins by adding marker points at obvious features such as the feet, hands, tail, joints, and face. Once an initial set of markers is chosen, the correspondence system executes the fitting algorithm and computes a tentative triangle pairing. The user can judge the quality of the resulting correspondence map by visualizing the paired triangles as in Figure 4-1 C or by examining the results of deformation transfer. The correspondence of regions that exhibit artifacts can then be refined by adding additional markers.

4.1 Template Fitting

The first stage in deriving a correspondence between the source and target meshes is a template fitting procedure that deforms one mesh, called the template, to match the shape of the other, called the destination. The roles of the source and target as template and destination are interchangeable. Either the source mesh is deformed to match the shape of the target, or the target is deformed to match the shape of the source. The choice of which mesh serves as template and which as destination is left to the user. In many cases, the mesh with more vertices makes a more suitable template since it has more degrees of freedom with which to express deformation.

The template fitting algorithm solves a minimization problem similar to the one used for deformation transfer, but it is designed to deform one mesh *into* the other, rather than deforming it *like* the other deforms. The user controls the deformation by supplying a set of marker points specified as pairs of template and destination vertex indices. Each pair indicates that the template vertex, after deformation, should match the position of the destination vertex. These markers are enforced as constraints in the minimization. The objective function contains one term that enforces deformation smoothness, one that prevents over smoothing, and one that moves the template vertices to the surface of the destination mesh. These terms are similar to those used by Allen, Curless, and Popović [2003] for registering human body shapes. However, the optimization I present uses the same numerical framework based on deformation gradients employed by deformation transfer.

Similar to the procedure followed by deformation transfer in Section 3.3, we let $\mathcal{T} = \{T_1 \dots T_{|\mathcal{T}|}\}$ be the set of per-triangle deformation gradients that define the template mesh deformation. Deformation smoothness, E_S , indicates that the deformation gradients of adjacent triangles should be equal:

$$E_S = \sum_{j=1}^{|\mathcal{T}|} \left\| \mathbf{T}_j - \left(\sum_{k \in \text{adj}(j)} \frac{1}{|\text{adj}(j)|} \mathbf{T}_k \right) \right\|_F^2. \quad (4.2)$$

Here, $\text{adj}(j)$ is the set of triangles adjacent to triangle j and $|\text{adj}(j)|$ is the number of adjacencies. Note that this term is minimized when the change in deformation across the surface, and not the surface itself, is smooth. For example, regardless of the surface smoothness, any affine transformation applied to all triangles minimizes E_S .

Deformation identity, E_I , is minimized when all deformation gradients are equal to the identity matrix:

$$E_I = \sum_{j=1}^{|\mathcal{T}|} \|\mathbf{T}_j - \mathbf{I}\|_F^2. \quad (4.3)$$

This term prevents the optimization from generating a drastic change in shape in order to achieve optimal smoothness.

The closest valid point term, E_C , indicates that the position of each vertex of the template mesh should be equal to the closest valid point on the destination mesh. Assuming the template has n vertices, this term has the form:

$$E_C = \sum_{i=1}^n \|\tilde{\mathbf{v}}_i - \mathbf{c}_i\|_2^2. \quad (4.4)$$

In this equation, \mathbf{c}_i is the closest valid point on the destination mesh to template vertex i . When computing the closest valid point, vertex normals of the template mesh are compared with triangle normals of the destination mesh and a difference in orientation of less than 90° indicates a valid point. This validity test results in better matches in regions such as the lips or fingers where opposite facing surfaces are in close proximity.

Care must be taken to implement the closest valid point query efficiently since a naïve implementation that tests every triangle greatly retards the performance of the correspondence system. Because the validity requirement is binary, it cannot be incorporated into a distance metric. As a result, standard closest-point algorithms are not applicable. I use a grid-based spatial binning algorithm to accelerate the closest valid point computation. A 3D grid of fixed resolution is built that surrounds both the template and destination mesh. The triangles of the destination mesh are scan converted into the grid by storing in each cell the indices of all triangles whose bounding box overlaps the cell. A hash table allows efficient storage of the grid contents. To perform a query, a breadth-

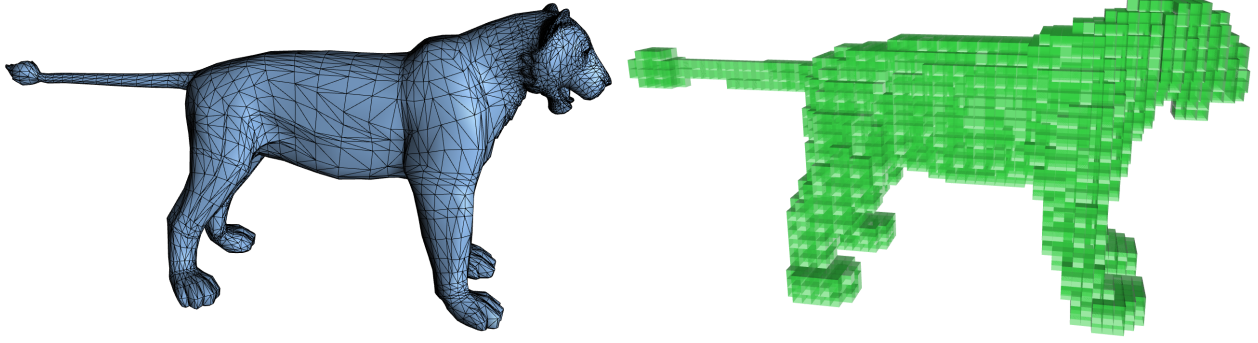


Figure 4-2: In order to perform the closest valid point query efficiently, the triangles of the destination mesh, shown on the left, are scan converted into a 3D grid. The occupied grid cells, shown on the right, are represented using a hash table. A closest valid point is found by performing a breadth first search of the grid data structure starting at the query location.

first search over the grid is initiated at the cell in which the query location resides. As each cell is encountered in the search, the closest point on every triangle whose index is stored in the cell to the query location is computed and the validity of this point is tested. If the validity test passes and the distance is smaller than that found in previous computations, this point is stored as the current closest valid point. The search continues until a valid point is found and all closer triangles have been tested. If no valid point exists within a distance threshold, the query is rejected and the associated term is removed from the summation in Equation 4.4. Figure 4-2 visualizes the spatial-binning data structure.

The deformed template vertices $\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n$ are found by minimizing the weighted sum of the three terms E_S , E_I , and E_C subject to the marker constraints:

$$\begin{aligned} \min_{\tilde{\mathbf{v}}_1 \dots \tilde{\mathbf{v}}_n} \quad & w_S E_S + w_I E_I + w_C E_C. \\ \text{subject to} \quad & \tilde{\mathbf{v}}_{t_k} = \mathbf{m}_k, \quad k \in 1 \dots m \end{aligned} \tag{4.5}$$

In this equation, w_S , w_I , and w_C are weights, t_k is the template vertex index for marker k , \mathbf{m}_k is the position of marker k on the destination mesh, and m is the number of markers. Equation 4.5 is a linearly constrained quadratic optimization problem. Like Equation 3.14 of deformation transfer, it results in a linear system of normal equations. Because the smoothness term compares the deformation gradients of adjacent triangles that may not lie in the same plane, deformation in the perpendicular direction is important and the fourth vertex formulation [Sumner and Popović 2004] should be used.

Example	Number of markers	Average target correspondences
Horse/Camel	62	1.155
Cat/Lion	77	1.829
Face/Head	42	2.062
Horse/Flamingo	73	1.131
Horse/Elephant	87	1.060

Table 4.1: This table includes statistics about the correspondence computation for the deformation transfer examples including the number of markers used for each source/target pair in the template fitting procedure and the average number of source triangles with which each target triangle is paired in the discrete correspondence map.

The final deformed template mesh is found by solving Equation 4.5 repeatedly in two phases. At the onset of the computation, the closest valid points are not meaningful since the template and destination are not aligned. Thus, in the first phase, we ignore the closet valid point term by using weights $w_S = 1.0$, $w_I = 0.001$, and $w_C = 0$ and solve the problem to find a deformed template mesh. The marker points of the deformed mesh will match exactly since they are specified as constraints, and the rest of the mesh will be carried along by the smoothness and identity terms. We use this initial estimation to compute a set of valid closest points. Then, in the second phase, we solve the same problem increasing w_C each time and updating the closest points after each iteration. Preserving $w_S = 1.0$ and $w_I = 0.001$ while increasing w_C in four steps from 1.0 to 5000.0 produces good results in all tests. Each time the minimization problem is solved, the template mesh is deformed from its original undeformed state. Since w_C increases, the template mesh more closely approximates the shape of the destination after each iteration. Figure 4-3 shows results from the template fitting procedure for examples used in deformation transfer and Table 4.1 lists the number of markers used in each example.

The template fitting algorithm works best when the template and destination are similar in shape so that fitting requires only small deformations. The horse/camel, cat/lion, and face/head examples in Figure 4-3 fall into this category and the deformed templates closely match the destination shapes. When substantial shape differences exist, more extreme deformations are required to fit the template to the destination. The deformation smoothness term (Equation 4.2) fights against these large deformations by favoring only gradual changes in shape. For example, in order to deform the flamingo model into the horse, the flamingo’s “S”-shaped neck must be unbent to match the horse’s

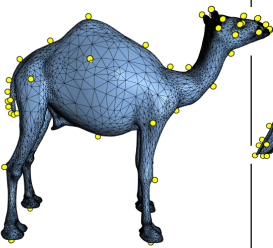
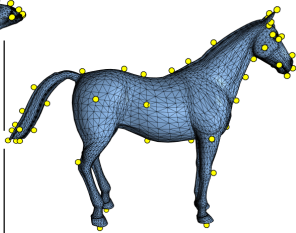
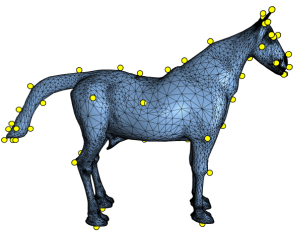
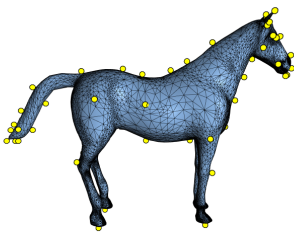
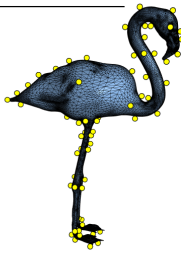
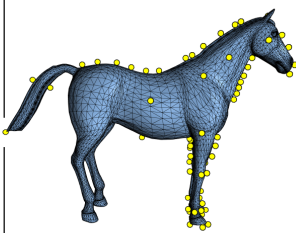
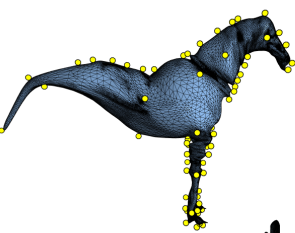
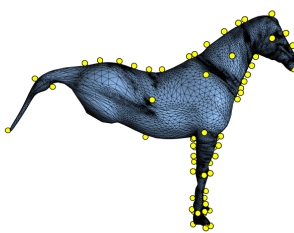
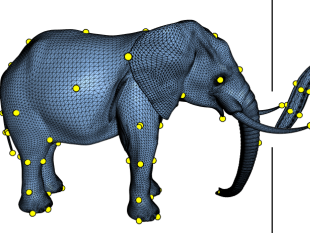
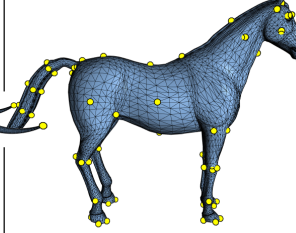
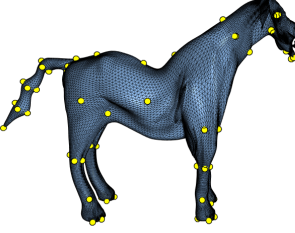
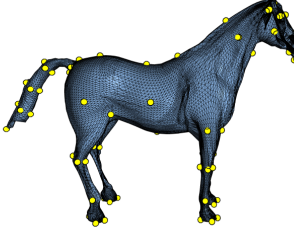
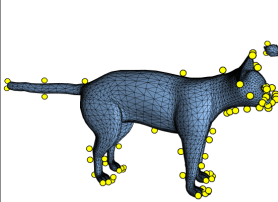
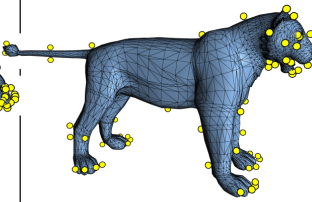
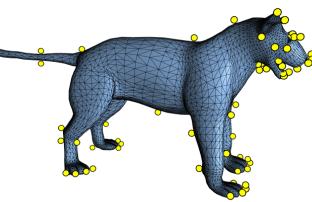
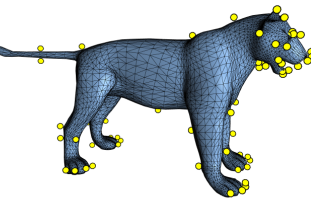
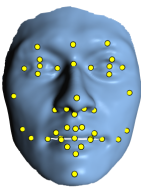
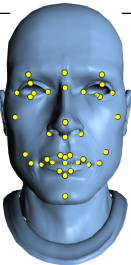
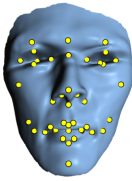
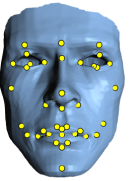
Template mesh	Destination mesh	Deformed template after first phase	Deformed template after second phase
			
			
			
			
			

Figure 4-3: The results of the template fitting procedure used to generate the correspondence for the horse/camel, horse/flamingo, horse/elephant, cat/lion, and face/head examples are shown above. The first and second columns indicate which mesh is used as the template and which as the destination. The yellow circles are the user-selected marker points. The third column shows the deformed template after the first phase where the smoothness and identity terms are used but the closest valid point term is ignored. The last column contains the final result after the second phase of fitting where the template approximates the shape of the destination mesh. Wireframe rendering is disabled in the last row because the tessellation of the face mesh is so dense that it completely obscures the surface.

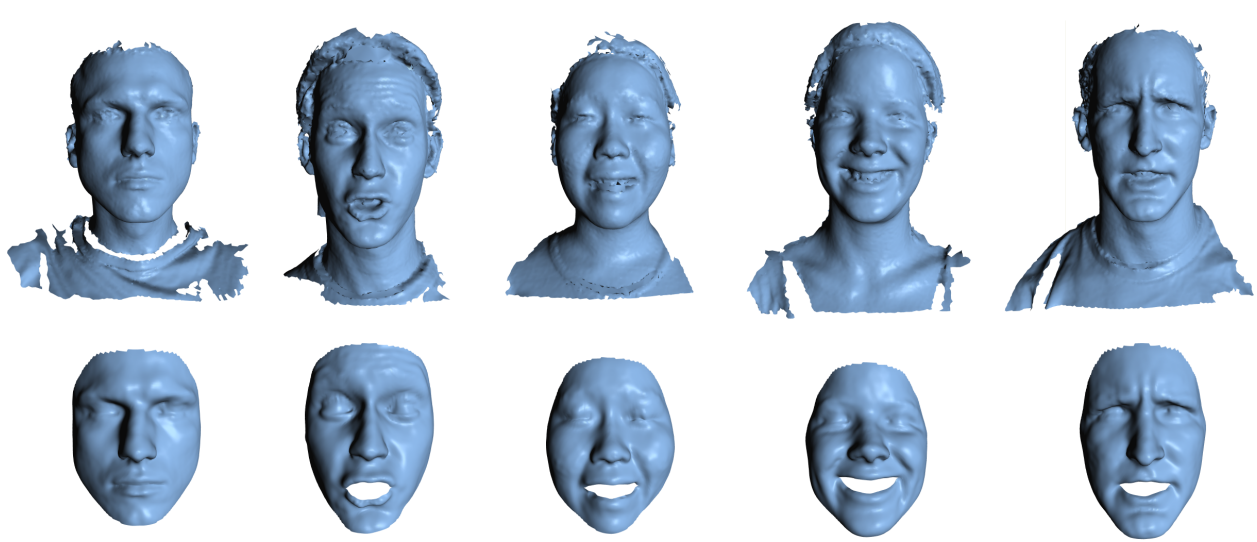


Figure 4-4: The template-fitting component of the correspondence system provides a self-contained application to bring meshes into one-to-one correspondence. Five scans from a database of 400 are shown above in the first row. The second row shows a “face mask” template fitted to each scan [Vlasic et al. 2005].

straight neck. Many markers along the neck are required to force this unbending. Likewise, in the horse/elephant example, the elephant’s long trunk and big ears must be compressed to conform to the horse’s head.

Although I designed the template fitting algorithm as one component of the correspondence system for deformation transfer, it can be used in other domains as a self-contained application to bring meshes into one-to-one correspondence. It was used in this way to build two databases of face meshes that include different individuals making different expressions [Vlasic et al. 2005]. A template “face mask” was deformed to match raw scans acquired with a 3D scanner. The first database contains 15 individuals each making 10 expressions, for 150 meshes in total. The expressions of one individual in the database comprise the source deformations in Figure 3-13. The second database contains 16 subjects performing 5 visemes each in 5 different expressions, yielding 400 meshes in total. The top row of Figure 4-4 shows five raw scans acquired for the second database, and the bottom row shows the template fitted to each scan.

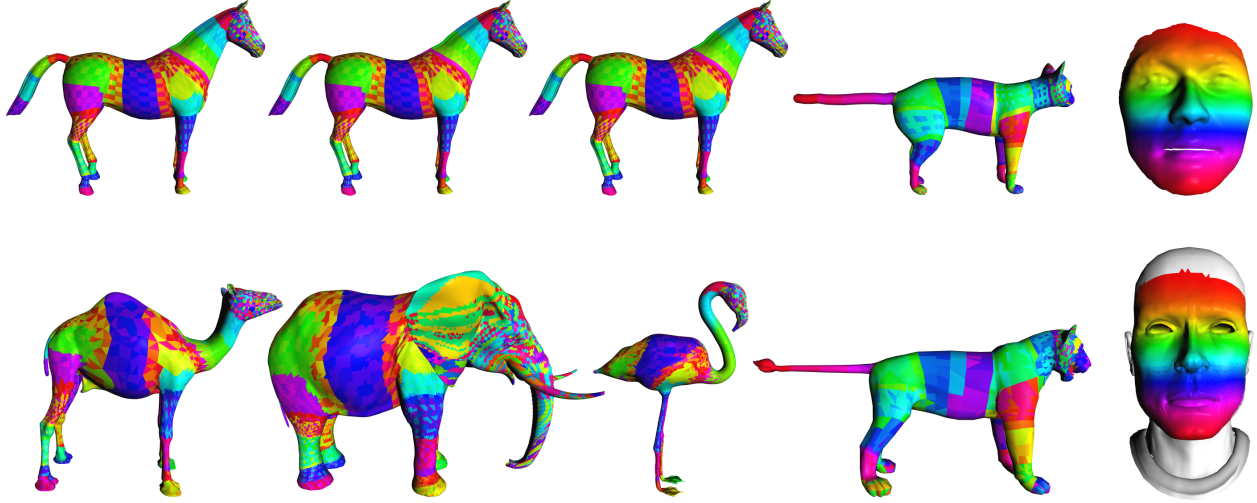


Figure 4-5: Triangle pairings are visualized by selecting arbitrary colormaps for the source meshes, shown in the top row, and assigning each triangle of the target meshes, shown in the bottom row, the color of one of its corresponding source triangle.

4.2 Triangle Pairing

Once the template mesh has been deformed to match the shape of the destination, the triangle pairing component of the correspondence system builds the map \mathcal{M} by comparing the proximity and orientation of triangles of both meshes. This method uses a compatibility criterion similar to the one in the closest valid point query of the previous section. Two triangles are compatible if their centroids are within a certain threshold of each other and the angle between their normals is less than 90° . This compatibility test prevents two nearby triangles with disparate orientation (e.g., triangles from the upper and lower lips of a face) from entering the correspondence. For each triangle of the deformed template, the pairing algorithm computes the closest compatible triangle (if any) of the destination mesh and adds the pair to the correspondence list. Likewise, for each triangle of the destination, the algorithm computes the closest compatible triangle of the deformed template and adds that pair. A spatial grid data structure similar to the one described in the previous section accelerates the distance queries. This process builds a many-to-many mapping that ensures all triangles of both meshes, subject to the compatibility restriction, are listed among the correspondences. Since both the source and target meshes can play the roles of template and destination, the actual order of the two indices in each pair may need to be reversed so that the source index comes first and the target second.

Figure 4-5 visualizes the computed triangle correspondence for the five examples in Figure 4-3. The

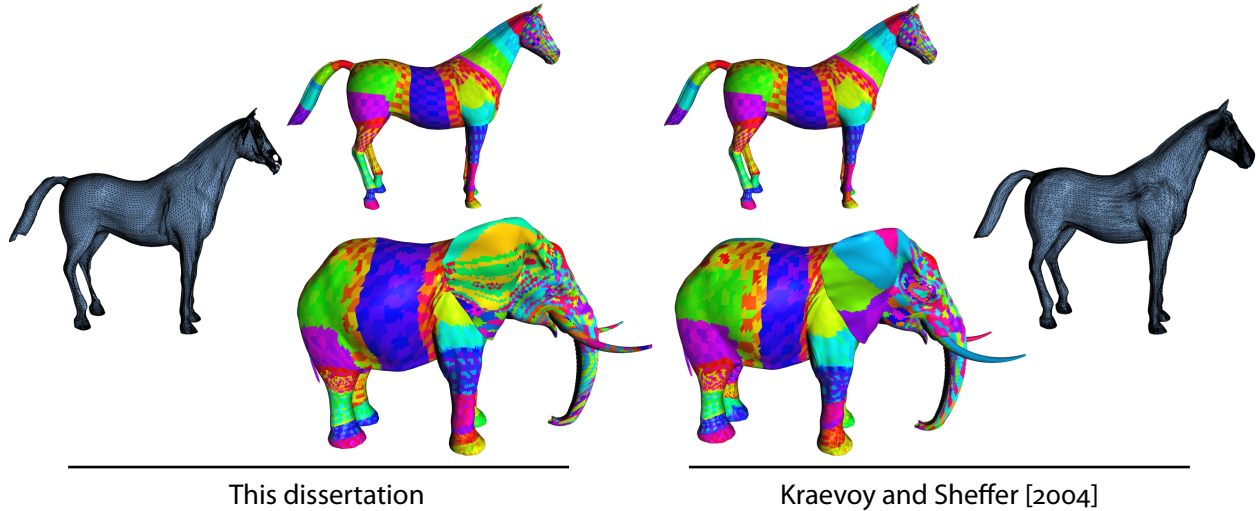


Figure 4-6: The per-triangle correspondences computed using the output from my template fitting method are compared with those computed using the output from the cross-parameterization algorithm of Kraevoy and Sheffer [2004].

source meshes are shown in the top row and the target meshes in the bottom. An arbitrary colormap is selected for the source meshes and each target triangle is assigned the color of its corresponding source triangle. If a target triangle corresponds to multiple source triangles, only one is used in the visualization. The white region of the head mesh in the bottom right indicates that those triangles correspond to nothing. A special tag is inserted into the correspondence map for each white triangle indicating its deformation should be minimal.

Table 4.1 indicates the average number of source triangles associated with each target triangle in the correspondence maps used by deformation transfer. Since the camel, flamingo, and elephant meshes are all more detailed than the horse mesh, this average is close to 1.0 indicating that the target triangles usually match only one source triangle. In the cat/lion and face/head examples, the target shape has fewer triangles than the source and the average is higher because each target triangle frequently matches several source triangles. The face/head example is an extreme case as the face mesh has roughly twice as many triangles in a much smaller area. The head mesh triangles far from the face have zero matches, while the ones that comprise the face have, on average, 3.762 matches. The overall average is 2.062.

Because template fitting is independent of triangle pairing, the triangle pairs can be computed using the output of a different fitting algorithm. Figure 4-6 compares results from my template fitting procedure to those from the cross-parameterization algorithm of Kraevoy and Sheffer [2004]. The

wireframe meshes show the elephant mesh deformed by each algorithm to match the shape of the horse. The colored meshes visualize the resulting per-triangle correspondence mappings. Before cross-parameterization could be applied, connectivity errors in the horse mesh had to be repaired by hand to achieve a manifold mesh. With these repairs, cross-parameterization produces a better deformed elephant mesh than my algorithm due to the significant shape differences between the elephant and horse. However, the triangle pairings appear similar in most parts of the body. The horse/elephant transfer in Figure 3-16 uses the correspondence map built from the output of Kraevoy and Sheffer's technique.

Mesh-Based Inverse Kinematics

5

Character rigging is a time consuming process that requires both artistic talent and technical expertise. In striving to make animation easier and more efficient, the rigging requirement is exactly what we want to abolish. This task is difficult, however, since rigging plays an essential role in the animation pipeline. It allows the user to vastly reduce the degrees of freedom of a mesh and carefully design a reduced space of *meaningful* deformations that incorporates the user’s semantic knowledge about how the character should move. Since this space encodes the movement of mesh vertices, I refer to it as the *mesh kinematics*. Once the mesh kinematics has been determined via rigging, animators can quickly generate motion that brings a character to life.

In contrast to rigging, mesh editing techniques allow the user to sculpt a mesh’s shape with no complex setup or parameter tuning. The intuitive interface employed by many editing algorithms lets the user make a broad change in mesh shape by repositioning only a few vertices. The remaining ones are positioned automatically to satisfy some general criterion such as detail preservation. However, detail preservation or other general criteria cannot capture mesh kinematics, since the class of meaningful deformations varies from mesh to mesh and may even depend on the context in which a character is manipulated. Thus, mesh editing methods are effective for sculpting details but do not create deformations that respect the mesh kinematics. For example, when manipulating a properly rigged character’s leg, it will bend only at the knee, hip, and ankle. These restrictions are difficult to enforce using existing editing techniques.

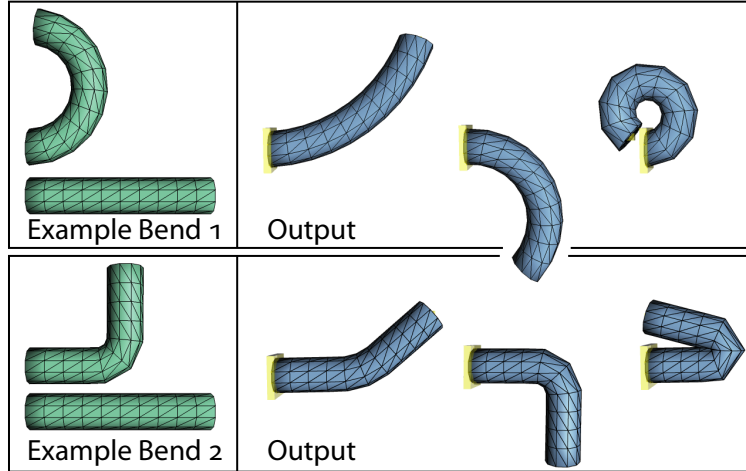


Figure 5-1: A simple demonstration of MESH IK. (*Top row*) Two examples are given, shown in green in the left column. By fixing one cap in place and manipulating the other end, the bar bends like the examples. (*Bottom row*) If a different example bend is provided, MESH IK generates the new type of bend when the mesh is manipulated.

I have developed an algorithm for posing meshes that requires no rigging controls or parameter tuning, provides direct manipulation of mesh shape using the intuitive interface employed by editing techniques, and generates deformations that respect the mesh kinematics [Sumner et al. 2005]. When the user selects and positions any subset of mesh vertices, my method produces a meaningful deformation automatically. Complex pose changes are accomplished intuitively by manipulating only a few vertices. Since kinematic changes are created via direct manipulation of the mesh analogously to traditional skeleton-based inverse kinematics for posing skeletons, I call this general problem *mesh-based inverse kinematics*, and my example solution MESH IK.

MESH IK relies on examples to indicate the space of meaningful deformations. As an alternative to character rigging, which is time consuming and difficult, and a general objective (e.g., detail preservation), which cannot encode semantic knowledge about an arbitrary object’s movement, MESH IK generalizes user-provided examples to learn the mesh kinematics. Using the learned space, it generates new shapes that respect the deformations exhibited by the examples, yet still satisfy vertex constraints imposed by the user. Although the user retains complete freedom to precisely position any vertices, most tasks only require manipulating a few. The animator can pose an object with minimal effort or bring it to life by keyframing vertex positions. Furthermore, the user always retains the freedom to choose the class of meaningful deformations for any mesh, as demonstrated by Figure 5-1. Since this class is specified by examples, MESH IK simplifies posing tasks even when traditional animation

or editing methods do not apply.

MESHIK examples can be scanned, hand-sculpted, designed with free-form modeling tools, or computed with arbitrarily complex procedural or simulation methods. The freedom to use *any* mesh editing or deformation algorithm greatly eases example creation. Skeleton-based deformation tools can create gross kinematic changes. Artifacts typical of these tools can be touched up with mesh editing algorithms. Additional details such as muscle bulges or wrinkles can be added, as well as any tweaks required to make each particular example match the vision of the artist. Creating a handful of examples in this way is much easier than character rigging since rigging must ensure that the mesh looks good in *every* pose. Rigging controls must be carefully tuned to generate the best possible deformations since tweaks and “fix ups” that efficiently repair particular problems of a single example are less effective when artifacts exist in a continuous range of poses. Furthermore, due to their generality, rigging controls cannot easily provide the rich class of deformations afforded by sculpting techniques. These reasons make MESHIK an attractive alternative to traditional character rigging.

MESHIK builds upon deformation transfer (Chapter 3) through the use of examples and extends it by allowing the user to create novel poses and animations. Since deformation transfer retargets any deformation of one mesh onto another, it provides an effective means of example creation. Constituent poses of a source character can be automatically transferred onto a target to provide the example collection required by MESHIK. Although deformation transfer can only transfer existing deformations, MESHIK uses the transferred target poses to create new deformations or animation.

MESHIK and deformation transfer also share the same numerical foundation of deformation gradients. Internally, MESHIK represents each example with a *feature vector* of deformation gradients that describes how the example has deformed relative to a reference mesh. The *feature space*, defined as a nonlinear span of the example feature vectors, describes the space of meaningful deformations. When the user displaces a few mesh vertices, MESHIK positions the remaining ones to produce a mesh whose feature vector is as close as possible to the feature space. This ensures that the reconstructed mesh meets the user’s constraints exactly while it best reproduces the example deformations. The calculations required to extract the example feature vectors are identical to those employed by deformation transfer, and the MESHIK reconstruction algorithm extends the one used by deformation transfer by finding the optimal nonlinear feature vector combination.

5.1 Principles of MESH IK

MESH IK uses the example meshes provided by the user to form a space of meaningful deformations. The definition of this space is critical as it must include deformations representative of those exhibited by the examples even far from the given data. The key to designing a good space is to extract, from each example, a vector of features that encodes important shape properties. MESH IK uses deformation gradient feature vectors that encode the change in shape exhibited by the examples on a triangle-by-triangle basis.

The simplest feature space is the linear span of the feature vectors of the example poses. Although this space is not what we will ultimately use, I describe it first because it is simple, fast, and may still be valuable in applications where linearity assumptions are sufficient [Blanz and Vetter 1999; Ngo et al. 2000] or where artifacts can be avoided by dense sampling [Bregler et al. 2002]. A more powerful nonlinear span is required in the general case when the natural interpolation of the example deformations is not linear (e.g., for rotations).

An edited mesh can be reconstructed from a feature vector by solving a least squares problem for the free vertices while enforcing constraints for each vertex that the user has positioned. The error incurred by fixing some vertices as constraints will propagate across the mesh, rather than being concentrated at the constrained vertices. MESH IK couples the constrained reconstruction process with a search within feature space so that it finds the position in feature space that has the minimal reconstruction error.

5.1.1 Feature Vectors

An obvious and explicit way to represent the geometry of a triangle mesh is with the coordinates of its vertices in the global frame. However, this representation, while simple and direct, is a poor choice for any mesh editing operation as the coordinates in the global frame do not capture the local shape properties and relationships between vertices [Sorkine et al. 2004].

For manipulating meshes, it is more useful to describe a mesh as a vector in a different feature space based on properties of the mesh. The components of the feature vector relate the geometry of nearby vertices and capture the short-range correlations present in the mesh. Since our goal is to encode the space of mesh deformations, deformation gradients (Section 3.1) are a natural choice and

comprise the feature vectors used by MESH IK.

Given a mesh in a reference pose S and a deformed pose \tilde{S} , the per-triangle deformation gradients are extracted using the same procedure employed by deformation transfer to compute the set of source deformation gradients: Equation 3.6 is applied once for each triangle after computing a fourth vertex using Equation 3.7. All per-triangle gradients are unrolled and concatenated into a single feature vector \mathbf{f} using the layout specified in Equation 3.18. If S and \tilde{S} both have n vertices and m triangles, \mathbf{f} is a $9m \times 1$ column vector.

Mapping a feature vector \mathbf{f} back to a Cartesian representation \mathbf{x} involves solving the optimization problem:

$$\begin{aligned} \min_{\mathbf{x}} \quad & \|\mathbf{G}\mathbf{x} - \mathbf{f}\|_2^2. \\ \text{subject to} \quad & \tilde{\mathbf{v}}_k = \mathbf{c}_k, \quad k \in 1 \dots p \end{aligned} \tag{5.1}$$

This formula mirrors Equation 3.23 but has not been separated in the spatial dimension. The unknown deformed vertices are stacked in \mathbf{x} following Equation 3.17. \mathbf{G} is a $9m \times 3n$ matrix that extracts deformation gradients using the QR formulation in Equation 3.11. As shown in Equation 3.20, \mathbf{G} is block diagonal with the $3m \times n$ matrix \mathbf{A} repeated three times. Since the meshes used by MESH IK have the same topology, \mathbf{A} is constructed according to Figure 3-6. The constraints $\tilde{\mathbf{v}}_k = \mathbf{c}_k, k \in 1 \dots p$, fix the position of p vertices so that they are treated as constants in the optimization.

5.1.2 Linear Feature Space

A feature space defines the space of desirable deformations. The simplest feature space is the linear span of the features extracted from the example meshes. A member \mathbf{f} of this space is parameterized by the coefficients in the weight vector \mathbf{w} :

$$\mathbf{f}(\mathbf{w}) = \mathbf{M}\mathbf{w}, \tag{5.2}$$

where \mathbf{M} is a matrix whose columns are the feature vectors $\mathbf{f}^i, i \in 1 \dots l$, corresponding to the l example meshes.

In practice, MESHK computes the mean $\bar{\mathbf{f}}$ and uses the mean centered feature vectors $\bar{\mathbf{f}}^i$:

$$\mathbf{M}\mathbf{w} = \bar{\mathbf{f}} + \sum_{i=1}^{l-1} w_i \bar{\mathbf{f}}^i. \quad (5.3)$$

Note that the linear dependence introduced by mean centering implies using $l - 1$ example features and weights instead of the l features and weights used in the non-centered linear combination.

Given only a few specified vertex positions, linear MESHK computes the pose \mathbf{x}^* whose features are most similar to the closest point $\mathbf{M}\mathbf{w}^*$ in the linear feature space:

$$\begin{aligned} \mathbf{x}^*, \mathbf{w}^* = \operatorname{argmin}_{\mathbf{x}, \mathbf{w}} \quad & \|\mathbf{G}\mathbf{x} - \mathbf{M}\mathbf{w}\|_2^2. \\ \text{subject to} \quad & \tilde{\mathbf{v}}_k = \mathbf{c}_k, \quad k \in 1 \dots p \end{aligned} \quad (5.4)$$

This equation replaces the feature vector \mathbf{f} in Equation 5.1 with a linear combination of example features $\mathbf{M}\mathbf{w}$. Because the linear space extrapolates poorly, this metric can be further augmented to penalize solutions that are far from the example meshes:

$$\begin{aligned} \mathbf{x}^*, \mathbf{w}^* = \operatorname{argmin}_{\mathbf{x}, \mathbf{w}} \quad & \|\mathbf{G}\mathbf{x} - \mathbf{M}\mathbf{w}\|_2^2 + \gamma \|\mathbf{w}\|_2^2. \\ \text{subject to} \quad & \tilde{\mathbf{v}}_k = \mathbf{c}_k, \quad k \in 1 \dots p \end{aligned} \quad (5.5)$$

The second term $\gamma \|\mathbf{w}\|_2^2$ favors examples close to the mean $\bar{\mathbf{f}}$ by penalizing large weights.

The value of γ , which weights the penalty term, can be chosen in a principled fashion by considering the Bayesian interpretation of the linear model: it maximizes the likelihood of the parameter vector \mathbf{w} with respect to the example poses. Accordingly, this method can be improved by compressing the matrix \mathbf{M} using its principal components and selecting an appropriate value for the weighting parameter γ as a function of the variance lost during PCA [Tipping and Bishop 1999].

An alternative to this linear Gaussian model is a nonlinear Gaussian-process-latent-variable model [Grochow et al. 2004] in which each component of the feature vector is an independent Gaussian process. This implies that one should carefully parameterize the feature space to match this independence assumption. For skeletons, exponential maps or Euler angles accomplish this task but introduce a

nonlinear mapping between the independent parameters and the user-specified handles. Applying a similar strategy on meshes will also produce nonlinear constraints and make it difficult to solve for thousands of vertices interactively.

5.1.3 Nonlinear Feature Space

Since the reconstructed mesh vertices are a linear function of the feature vectors, linear blending of feature vectors amounts to naïve linear blending of poses, which is well known to result in unnatural effects if the blended poses have undergone rotation. Thus, in our setting, linear blending will suffice only if the example set is dense enough that large rotations are not present. However, dense sampling is not the typical case and *generalizations* of the examples beyond small deformations are not possible with linear blending. To avoid artifacts due to large rotations, which are typical in most nontrivial settings, we require a “span” of the example features that combines rotations in a more natural way. Our approach is based on polar decomposition [Shoemake and Duff 1992] and the matrix exponential map.

First, we decompose the deformation gradient \mathbf{T}_{ij} for the j^{th} triangle ($j \in 1 \dots m$) in the i^{th} pose ($i \in 1 \dots l$) into rotational and scale/shear components using polar factorization:

$$\mathbf{T}_{ij} = \mathbf{R}_{ij}\mathbf{S}_{ij}. \quad (5.6)$$

We then use the exponential map to combine the individual rotations of the different poses. The scale and shear part can be combined linearly without further treatment.

We implement the exponential map using the matrix exponential and logarithm functions [Murray et al. 1994]. These provide a mapping between the group of 3D rotations $\mathbf{SO}(3)$ and the Lie algebra $so(3)$ of skew symmetric 3×3 matrices. A practical approach to interpolating rotations is to map them to $so(3)$ using the matrix logarithm, interpolate linearly in $so(3)$, and map back to $\mathbf{SO}(3)$ using the matrix exponential [Murray et al. 1994; Alexa 2002a]. This leads to the following expression for the nonlinear span of the deformation gradient of the j^{th} triangle:

$$\mathbf{T}_j(\mathbf{w}) = \exp \left(\sum_{i=1}^l w_i \log(\mathbf{R}_{ij}) \right) \left(\sum_{i=1}^l w_i \mathbf{S}_{ij} \right) \mathbf{P}_j. \quad (5.7)$$

The matrix exponential and logarithm are evaluated efficiently using Rodrigues’ formula [Murray et al.

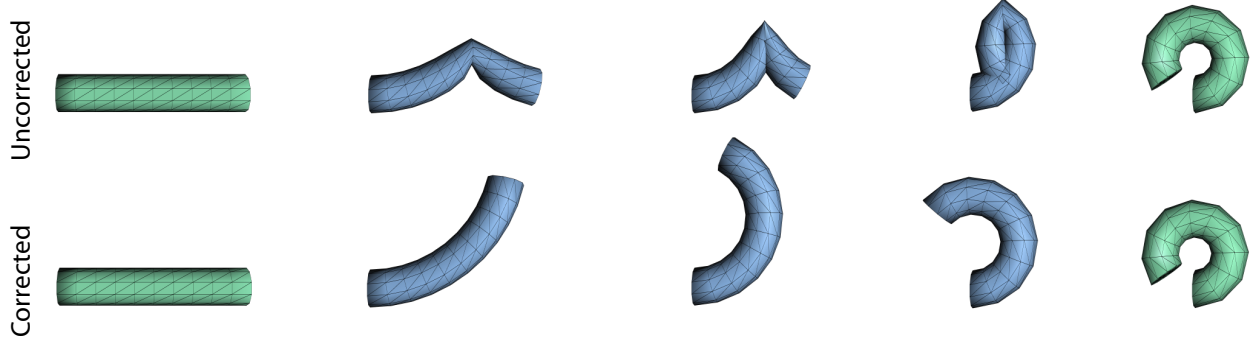


Figure 5-2: The ambiguity inherent in the matrix logarithm is demonstrated with this interpolation sequence. When all triangles use rotation angles in the same range, neighboring triangles may take different paths during interpolation. This problem is evident in the “Uncorrected” row where the bar pinches and folds on top of itself. By marking certain triangles and adding 2π , the correct interpolation sequence is generated, as shown in the “Corrected” row.

1994]. Experiments with exponential and logarithm functions for general matrices [Alexa 2002a] which do not require factorization into rotations and scales exhibit singularities that prevent a stable solution of the minimization problem. Because MESHK solves an optimization that minimizes reconstruction error, it is important that this error be a meaningful measure of the quality of the reconstructed mesh. For this reason, Equation 5.7 must include the matrix \mathbf{P}_j defined in Equation 3.15 to remove the perpendicular space component from the interpolated deformation gradient. Otherwise, the perpendicular-space error dominates the cost function and the weights are adjusted to lower this error even though the triangles match the in-plane transformation more poorly.

We chose to use the matrix exponential and logarithm because we can easily take derivatives of the resulting nonlinear model with respect to \mathbf{w} . For later use in Section 5.2.1, we note that the partial derivatives of $\mathbf{T}_j(\mathbf{w})$ are given by

$$\begin{aligned} D_{w_k}(\mathbf{T}_j(\mathbf{w})) &= \exp \left(\sum_{i=1}^l w_i \log(\mathbf{R}_{ij}) \right) \log(\mathbf{R}_{kj}) \sum_{i=1}^l w_i \mathbf{S}_{ij} \mathbf{P}_j \\ &+ \exp \left(\sum_{i=1}^l w_i \log(\mathbf{R}_{ij}) \right) \mathbf{S}_{kj} \mathbf{P}_j. \end{aligned} \quad (5.8)$$

The matrix logarithm is a multi-valued function: each rotation in $\mathbf{SO}(3)$ has infinitely many representations in $\mathfrak{so}(3)$. In some cases, interpolation may require equivalent rotations in a different range which can be computed by adding multiples of 2π . This problem is demonstrated with the

interpolation sequence shown in Figure 5-2. When all triangles use rotation angles in the range between $-\pi$ and π , some triangles take a different path than their neighbors during interpolation, as shown by the “Uncorrected” row. My implementation of the matrix logarithm returns rotation angles between $-\pi$ and π by default. However, the user can mark particular triangles to indicate that a corrective multiple of 2π should be added. The interpolation sequence using these corrections is shown in the “Corrected” row of Figure 5-2. The same ambiguity also exists when rotations are represented as Euler angles or quaternions. It is present in the interpolation algorithms of Alexa, Cohen-Or, and Levin [2000] and Xu and colleagues [2005]. Alexa utilizes a greedy approach to automatically adjust the angles of adjacent simplices [Alexa 2003a]. Whether proper corrections can be found for all cases is an open problem.

The nonlinear feature space can be thought of as an n -way boundary-based version of as-rigid-as-possible shape interpolation [Alexa et al. 2000]. Rather than performing a two-way interpolation based on the compatible dissection of the interior of two shapes, MESH IK interpolates the boundary of n shapes. As discussed in Section 3.1, the practical implication of this reformulation is significant. MESH IK interpolation is faster because it solves for fewer vertices and easier to apply because compatible dissection of n shape interiors is difficult without adding an extremely large number of Steiner vertices. An experimental comparison of the two methods is shown in Figure 3-4 and demonstrates that, for 2D polygonal shapes, MESH IK interpolation behaves reasonably despite ignoring the interior. The remaining results in this dissertation and my experience with MESH IK indicate that the same holds for 3D meshes.

By setting the weights directly, rather than solving for them within the IK framework, the nonlinear feature space can create multi-way blends. Figure 5-3 demonstrates three-way interpolation. Used in this way, the nonlinear feature space matches the concurrent work of Xu and colleagues on boundary-based mesh interpolation [2005]. This similarity is expected since the analytic derivation in Section 3.5 can be applied to show that reconstructing a mesh using fixed blending weights is equivalent to solving a Poisson problem. In this case, the guidance function corresponds to the interpolated feature vectors. However, while Xu and colleagues focus on interpolation with prescribed blending weights, my primary contribution is a formulation of mesh-based inverse kinematics that hides these weights from the user behind an intuitive interaction metaphor.

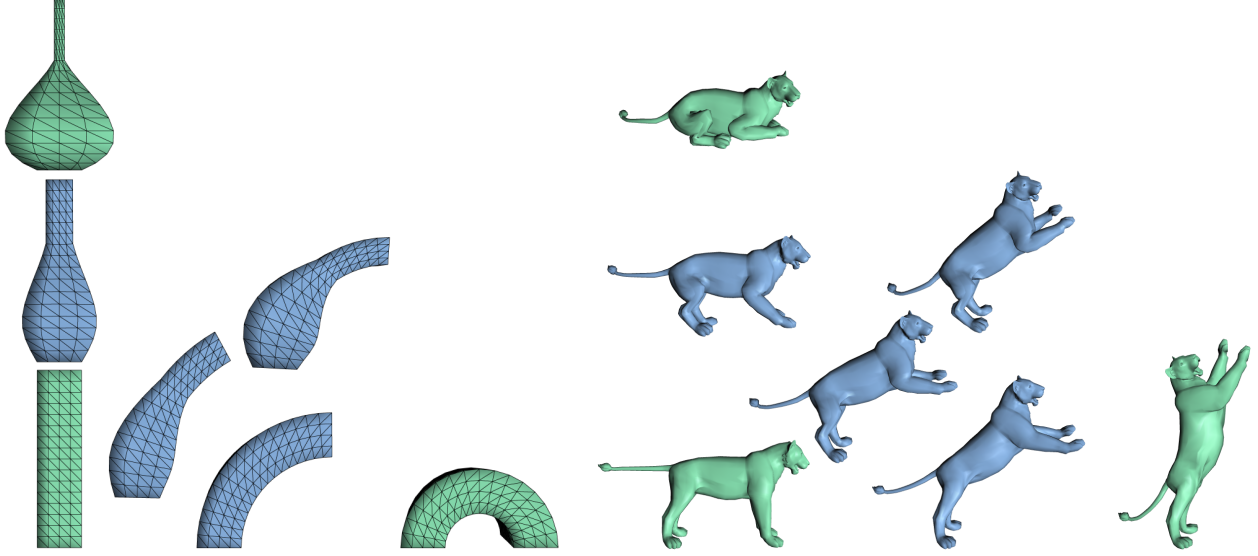


Figure 5-3: The nonlinear feature space can be used to perform multi-way interpolation. In these two examples, the green corner meshes are blended to produce the blue ones.

5.2 Numerics

In this section I show how to solve the following nonlinear analog of the linear inversion in Equation 5.4:

$$\begin{aligned} \mathbf{x}^*, \mathbf{w}^* = \operatorname{argmin}_{\mathbf{x}, \mathbf{w}} \quad & \|\mathbf{G}\mathbf{x} - \mathbf{M}(\mathbf{w})\|_2^2, \\ \text{subject to} \quad & \tilde{\mathbf{v}}_k = \mathbf{c}_k, \quad k \in 1 \dots p \end{aligned} \tag{5.9}$$

where \mathbf{M} is now a function that combines the feature vectors nonlinearly according to Equation 5.7. This is a nonlinear least-squares problem which can be solved using the iterative Gauss-Newton algorithm [Madsen et al. 2004]. At each iteration, a linear least-squares system is solved which involves solving the normal equations by Cholesky decomposition and backsubstitution. I now elaborate on the key stages of this procedure.

5.2.1 Gauss-Newton Algorithm

In MESHK, the Gauss-Newton algorithm linearizes the nonlinear function of the feature weights which defines the feature space:

$$\mathbf{M}(\mathbf{w} + \boldsymbol{\delta}) = \mathbf{M}(\mathbf{w}) + \mathbf{D}_{\mathbf{w}}(\mathbf{M}(\mathbf{w}))\boldsymbol{\delta}. \tag{5.10}$$

Then, each Gauss-Newton iteration solves a linearized problem to improve \mathbf{x}_k and \mathbf{w}_k —the estimates of the vertex positions and the weight vector at the k^{th} iteration:

$$\begin{aligned}\boldsymbol{\delta}_k, \mathbf{x}_{k+1} &= \underset{\boldsymbol{\delta}, \mathbf{x}}{\operatorname{argmin}} \|\mathbf{G}\mathbf{x} - \mathbf{D}_{\mathbf{w}}(\mathbf{M}(\mathbf{w}_k))\boldsymbol{\delta} - (\mathbf{M}(\mathbf{w}_k) + \mathbf{c})\|_2^2 \\ \mathbf{w}_{k+1} &= \mathbf{w}_k + \boldsymbol{\delta}_k.\end{aligned}\tag{5.11}$$

The constraints from Equation 5.9 have been enforced by treating the constrained vertices as constants, updating \mathbf{G} and \mathbf{x} , and computing the constraint vector \mathbf{c} as described in Section 3.4.2.

The process repeats until convergence, which we detect by monitoring the change in the objective function $f_k = f(\mathbf{w}_k)$, the gradient of the objective function, and the magnitude of the update vector $\boldsymbol{\delta}_k$ [Gill et al. 1989]:

$$\begin{aligned}\|f_k - f_{k-1}\|_\infty &< \epsilon(1 + f_k) \\ \|\mathbf{D}_{\mathbf{w}}(f(\mathbf{w}))\|_\infty &< \sqrt[3]{\epsilon}(1 + f_k) \\ \|\boldsymbol{\delta}_k\|_\infty &< \sqrt[2]{\epsilon}(1 + \|\mathbf{w}_k\|_\infty).\end{aligned}\tag{5.12}$$

In my experiments, the optimization converges after about six iterations with $\epsilon = 1.0 \times 10^{-6}$.

Solving the linear least-squares problem in Equation 5.11 leads to a system of normal equations:

$$\mathbf{B}^\top \mathbf{B} \begin{bmatrix} \mathbf{x} \\ \boldsymbol{\delta} \end{bmatrix} = \mathbf{B}^\top (\mathbf{M}(\mathbf{w}_k) + \mathbf{c}),\tag{5.13}$$

where \mathbf{B} is a sparse matrix of size $9m \times (3(n - p) + l)$ of the form

$$\mathbf{B} = \left[\begin{array}{cc|c} \mathbf{A} & & -\mathbf{J}_1 \\ & \mathbf{A} & -\mathbf{J}_2 \\ & & \mathbf{A} & -\mathbf{J}_3 \end{array} \right].\tag{5.14}$$

Recall that \mathbf{A} is also a very sparse matrix, having only three entries per row (Figure 3-6). As we will see in Section 5.2.2, this permits efficient numerical solution of the system despite its size. The three blocks \mathbf{J}_i are the blocks of the Jacobian matrix $\mathbf{D}_{\mathbf{w}}(\mathbf{M}(\mathbf{w}))$ partitioned according to the three vertex coordinates.

5.2.2 Cholesky Factorization

Without a special purpose solver, the normal equations in Equation 5.13 can take a minute or longer to solve during each step of the Gauss-Newton iteration. This is much too slow for an interactive system, which, in my experience, requires at least two solutions for every second of interaction. The key to accelerating the solver is to reuse computations between iterations. A direct solution with a general purpose method (e.g., Cholesky or QR factorization [Golub and Loan 1996]) will not be able to reuse the factorization from the previous iteration because \mathbf{B} continually changes. And, despite the sparsity of $\mathbf{B}^\top \mathbf{B}$, conjugate gradient converges too slowly even with a variety of preconditioners.

My solution uses a direct method with specialized Cholesky factorization that exploits the block structure of the system matrix:

$$\mathbf{B}^\top \mathbf{B} = \begin{bmatrix} \mathbf{A}^\top \mathbf{A} & & & -\mathbf{A}^\top \mathbf{J}_1 \\ & \mathbf{A}^\top \mathbf{A} & & -\mathbf{A}^\top \mathbf{J}_2 \\ & & \mathbf{A}^\top \mathbf{A} & -\mathbf{A}^\top \mathbf{J}_3 \\ -\mathbf{J}_1^\top \mathbf{A} & -\mathbf{J}_2^\top \mathbf{A} & -\mathbf{J}_3^\top \mathbf{A} & \sum_{i=1}^3 \mathbf{J}_i^\top \mathbf{J}_i \end{bmatrix}. \quad (5.15)$$

The three $\mathbf{A}^\top \mathbf{A}$ blocks, each sparse $(n-p) \times (n-p)$ matrices, are constant throughout the iterations. If these blocks are pre-factored, the remaining portion of the Cholesky factorization may be computed efficiently.

First, symbolic Cholesky factorization $\mathbf{U}^\top \mathbf{U} = \mathbf{B}^\top \mathbf{B}$ reveals the block structure of the upper-triangular Cholesky factor:

$$\mathbf{U} = \begin{bmatrix} \mathbf{R} & & & -\mathbf{R}_1 \\ & \mathbf{R} & & -\mathbf{R}_2 \\ & & \mathbf{R} & -\mathbf{R}_3 \\ & & & \mathbf{R}_s \end{bmatrix} \quad (5.16)$$

where

$$\mathbf{R}^\top \mathbf{R} = \mathbf{A}^\top \mathbf{A}. \quad (5.17)$$

We precompute \mathbf{R} by sparse Cholesky factorization [Toledo 2003] after re-ordering the columns to reduce the number of additional non-zero entries [Karypis and Kumar 1998]. The remaining blocks of \mathbf{U} are computed by solving the following equations in each iteration:

$$\mathbf{R}^\top \mathbf{R}_i = \mathbf{A}^\top \mathbf{J}_i, \quad i \in 1 \dots 3 \quad (5.18)$$

$$\mathbf{R}_s^\top \mathbf{R}_s = \sum_{i=1}^3 \mathbf{J}_i^\top \mathbf{J}_i - \mathbf{R}_i^\top \mathbf{R}_i. \quad (5.19)$$

In Equation 5.18, backsubstitution with the precomputed \mathbf{R} computes the blocks \mathbf{R}_1 , \mathbf{R}_2 , and \mathbf{R}_3 by solving three linear systems. These blocks are in turn used on the right-hand side of Equation 5.19 to compute the $l \times l$ matrix whose dense Cholesky factorization yields the last block \mathbf{R}_s . For a large number of examples, this factorization step will eventually become the bottleneck. In experiments, however, with $l=20$ or fewer examples, the solution of Equation 5.18 for the three dense $(n - p) \times l$ blocks and their use in the computation of $\mathbf{R}_i^\top \mathbf{R}_i$ dominates the cost.

5.3 Results and Discussion

I have implemented MESH IK both as an interactive mesh manipulation system as well as in an offline application that uses keyframed constraints to solve for mesh poses over time. In the interactive system, the user can select groups of vertices that become “handles” which can be interactively positioned. As the handles are moved the rest of the mesh is automatically deformed.

Figure 5-4 demonstrates the power of MESH IK. Given a cylindrical bar in two poses (5-4 A), one straight, and one smoothly bent, the user constrains the left cap to stay in place and manipulates one vertex on the right cap. Using the nonlinear feature space, my system is able to generalize to any other bend of the bar in the same plane (5-4 B). In contrast, the linear feature space (5-4 C) interpolates the two examples poorly (the tip of the bar collapses in between the examples) and extrapolates even more poorly. If the end of the bar is dragged perpendicular to the example bend (5-4 D), it deforms differently since no example has demonstrated how to deform in this direction. Given an additional example, the bar can bend in that plane (5-4 E) as well as the space in between (5-4 F). By supplying a different example, the bar bends differently (Figure 5-1). Thus, MESH IK does not prescribe one type of deformation but instead derives the appropriate class of deformations from the examples.

Figure 5-5 shows how MESH IK can be used to pose a character. Ten example poses, shown in green in the top row, are used for this demonstration. Two handle vertices are selected as constraints on the front and back foot of the reference pose (5-5 A). By dragging the front foot forward, the lion

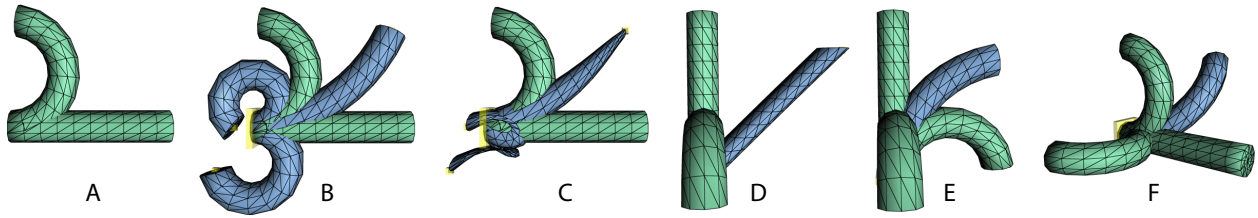


Figure 5-4: Using MESH IK to pose a bar. (A) Two example poses superimposed on top of each other. (B) The left cap of the unbent bar is constrained to stay in place while a single vertex on the right side is manipulated. Three edits using the nonlinear feature space are shown. Note that MESH IK generalizes beyond the two examples and can create arbitrary bends in the plane. (C) In contrast, the linear feature space interpolates and generalizes poorly. (D) In this top down view, moving the constrained vertex perpendicular to the bend causes a shear since no examples are provided in this direction. (E)–(F) Providing one additional example in the perpendicular direction allows MESH IK to generalize to bends in that direction as well as in the space in between.

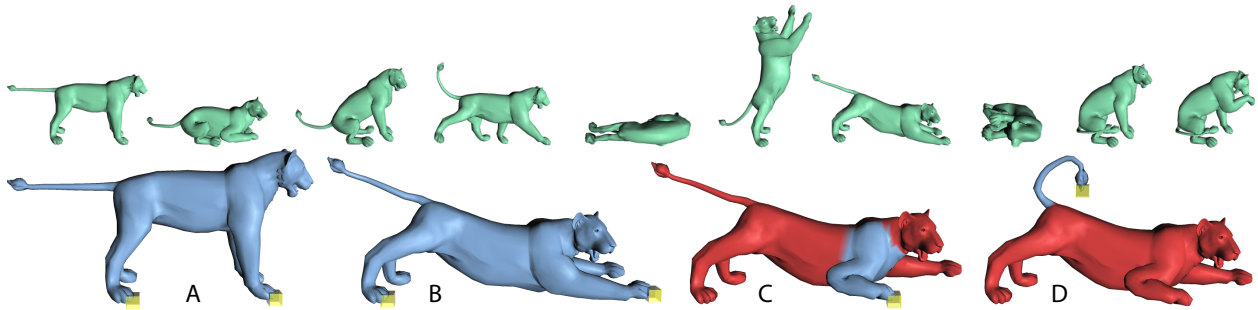


Figure 5-5: (Top row) Ten lion example poses. (Bottom row) A sequence of posing operations. (A) Two handle vertices are chosen. (B) The front leg is pulled forward and the lion continuously deforms as the constraint is moved. (C) The red region is selected and frozen so that the front leg can be edited in isolation. (D) A similar operation is performed to adjust the tail. The final pose is different from any individual example.

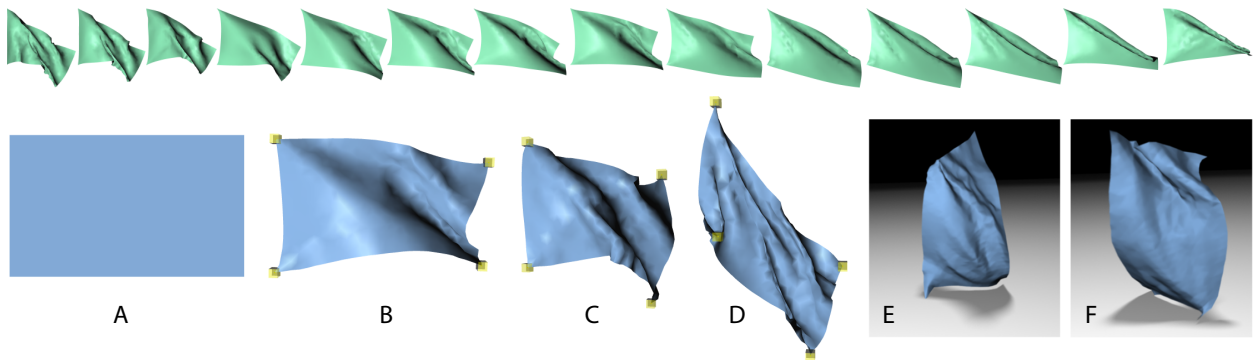


Figure 5-6: Posing a simulated flag. (Top row) Fourteen examples of a flag blowing in the wind created with a cloth simulation. (Bottom row) (A) An undeformed flag is used as the reference pose. (B)–(D) By positioning only the corners of the flag, the user creates realistic cloth deformations without requiring any dynamic simulation. (E)–(F) Two frames from an animation in which the constraints on the corners are keyframed to produce a walking motion.

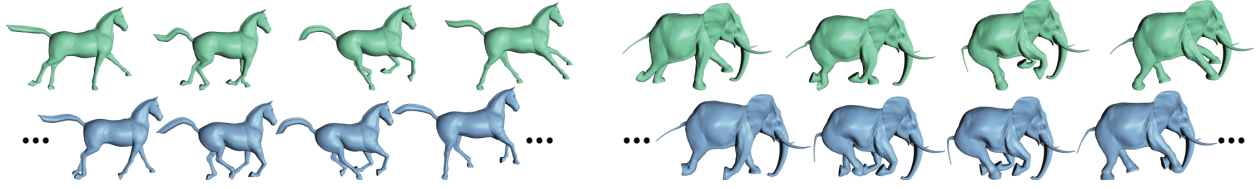


Figure 5-7: Galloping horse and elephant animations are created using only four examples of each along with the same keyframed motion of one vertex on each foot.

bends its front legs at the hip and stretches its body forward. The position of the lion’s paw can be precisely controlled by the user. In (5-5 B) the paw has been pulled farther forward than its position in any example. The body of the lion deforms realistically to meet the constraints so that there is no discernible distortion. In order to pose only the front right leg and keep the rest of the body fixed (5-5 C), the user selects the unwanted region (shown in red) and removes it from the objective function by building a feature space that ignores the deformation gradients of the selected triangles. This region remains fixed in place, but does not contribute to the error as the optimal weights are computed. This allows the user to pose the front leg independent of the rest of the body. After performing the same operation for the tail (5-5 D), the user has achieved a novel pose different from all those shown in the example set. This result also shows how MESH IK builds upon deformation transfer since the example lion poses are the result of transfer from a cat mesh (Figure 3-10).

Figure 5-6 demonstrates that MESH IK also applies when deformations have no obvious skeletal representation. The input for this demonstration is fourteen flag examples from a dynamic simulation, shown in the top row. Starting with an undeformed flag (5-6 A), the user arbitrarily positions the flag’s four corners (5-6 B–D). The interior deforms in a cloth-like fashion. By keyframing the position of the constraints over time, the user creates an animation of a walking flag (5-6 E–F).

Figure 5-7 shows the system used to produce a galloping animation. Four example poses of a horse are used as input, and one vertex on each foot of the horse is keyframed to follow a gallop gait. The positions of the remaining vertices of the horse are chosen by MESH IK for each frame, resulting in a galloping animation. If we replace the four horse poses with those of an elephant and use the same keyframed foot positions, MESH IK computes a galloping elephant.

Temporal coherence is important when generating offline animations. Since the deformation system is nonlinear, a small change in the constraints may result in a large change in the resulting deformation. In order to achieve temporal coherence, we add the additional term $\eta \|\mathbf{w} - \mathbf{w}_0\|_2^2$ to the

objective function in Equation 5.9. This term encourages the new blending weights w to be similar to the ones from the previous frame of animation w_0 . A value of 100 is used for η in all animations.

Table 5.1 gives statistics about the meshes used in these results including the number of vertices, the number of triangles, the number of examples, the number of fixed handle vertices, and the running times. The timing was measured on a 3.2GHz Pentium 4 PC with 2GB of RAM. The “Factor” column indicates the time required to permute $A^T A$ and compute the Cholesky factorization. This computation is a preprocess as the factorization does not change for a particular choice of handle vertices. The “Solve” column indicates the time required to perform one iteration of the Gauss-Newton algorithm described in Section 5.2.1. After each iteration, the user-interface is updated and new positions for the constrained handle vertices are queried by the solver. This allows the system to remain interactive during the nonlinear optimization. Figure 5-8 graphs the solve time as a function of the number of examples for the horse and elephant meshes.

Mesh	Vertices	Triangles	Examples	Fixed	Factor	Solve
Bar	132	260	2	22	0.000 s	0.000 s
Flag	516	932	14	3	0.016 s	0.016 s
Lion	5,000	9,996	10	2	0.109 s	0.156 s
Horse	8,425	16,846	4	2	0.203 s	0.125 s
Elephant	42,321	84,638	4	2	10.282 s	0.688 s

Table 5.1: Number of vertices, triangles, and example meshes, as well as the number of fixed handle vertices and resulting timing data for the demonstrated results.

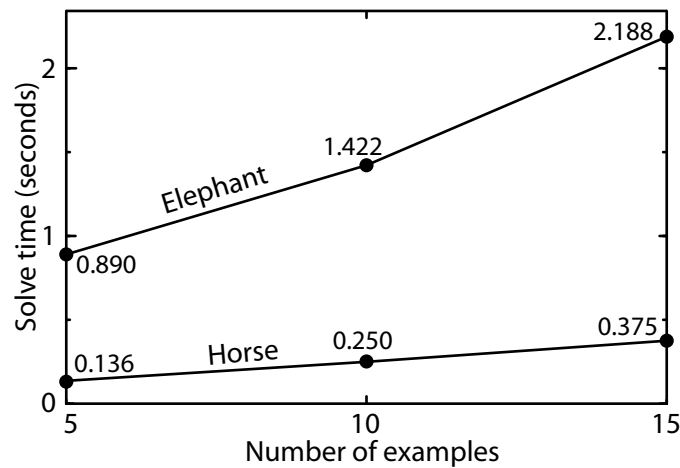


Figure 5-8: Solve time as a function of the number of examples for the horse and elephant meshes. (Timings are from the original publication [Sumner et al. 2005].)

Conclusion

6

Computer-generated character animation, where human or anthropomorphic characters are animated to tell a story, holds tremendous potential to enrich education, human communication, perception, and entertainment. However, current animation procedures rely on a time consuming and difficult process that requires both artistic talent and technical expertise. Although the stages of the animation pipeline—modeling, rigging, and animation—are well explored individually, there is little work that extends beyond the boundaries of any one area. As a consequence, the same procedure must be followed for each new character without the opportunity to generalize or reuse technical components. The work addressed in this dissertation eases the animation process by offering novel connections between the stages and opportunities for reuse.

6.1 Contributions

Deformation transfer allows deformations from any stage in the pipeline to be reused on another character. The mesh-based formulation does not depend on the mechanism used to create the deformation and can be applied to hand-sculpted alterations made during modeling, individual poses created with rigging controls, or continuous keyframed or simulated animation. Deformation transfer enables the compilation of a database of both skeletal and non-skeletal deformations such as running, walking, or gesturing that can be retargeted onto new characters of different shape when needed. Reusing this motion amortizes the human effort spent creating it in the first place.

The correspondence algorithm provides a means of communication between the myriad existing meshes used in computer animation. By employing a many-to-many mapping in the form of discrete triangle pairs, shapes of different topology, tessellation and even gross anatomical structure are related to one another. A deformation algorithm that computes a user-guided partial parameterization creates this mapping efficiently.

Mesh-based inverse kinematics allows the user to avoid the traditional rigging phase and animate a mesh via direct manipulation. The user can generate significant mesh deformations and pose changes intuitively with a minimal amount of work. This technique is distinguished from traditional animation methods since no formal rigging is required. It is distinguished from existing mesh editing algorithms since it gives the user the freedom to specify the class of meaningful deformations. Example creation can use the full spectrum of editing, sculpting, scanning, deformation, and animation techniques and is further facilitated by deformation transfer.

The deformation transfer, correspondence, and mesh-based inverse kinematics algorithms employ the same mathematical formulation which approximates the deformation gradient tensor field from continuum mechanics on a triangle mesh. In deformation transfer, deformation gradients are used to encode the source deformation and apply it to the target. For correspondence, the deformation smoothness term allows the deformation incurred by vertex constraints to be smoothly propagated across the mesh. In mesh-based inverse kinematics, deformation gradients make up the feature vectors and an appropriately chosen nonlinear interpolation scheme generates a feature space that captures the class of meaningful mesh deformations.

6.2 Future Directions

Generalized Skeletons. Without exception, existing systems that generate 3D character animation single out kinematic deformation and model it with a skeleton or a similar rigging device. Non-kinematic deformation such as muscle bulging or facial expressions is created differently. Even methods that support a generic specification of deformation for 2D characters defer to skeleton-based articulation when 3D kinematic deformation is involved [Ngo et al. 2000; Bregler et al. 2002]. The domain-specific nature of these solutions limits the degree to which they can be generalized and applied in new situations.

In contrast, my algorithms apply to any deformation, skeletal or non-skeletal. The underlying representation of deformation gradients accommodates kinematic and non-rigid deformation without distinction. As a consequence, the methods presented in this dissertation are broadly applicable. Deformation transfer can apply the kinematic poses of the horse shown in Figure 3-9 to the camel as successfully as the nonrigid buckling displayed in Figure 3-12. Likewise, MESHK can animate a crouching lion (Figure 5-5) as easily as a flapping flag (Figure 5-6).

One conceptual interpretation of my work is that it generalizes the idea of skeleton-driven animation to the point where each triangle acts analogously to a joint by contributing a transformation matrix to the ultimate deformed pose. Following this analogy, a mesh can be thought of as a very complex skeleton in which every joint corresponds to a triangle, the bones are not rigid, and are not arranged hierarchically in a tree. Thus, the algorithms I present lie at the opposite extreme of traditional skeleton-based animation where bones are chosen parsimoniously. The conceptual contribution of a mesh as a “generalized skeleton” may apply to other problems in computer graphics. Furthermore, just as traditional skeletons have benefited from years of research, this generalization will improve with time, particularly through the exploration of other parameterizations in between the most restricted *skeleton* representations and the most general *mesh* representations.

Copy/Paste Transfer. In the presented results of deformation transfer, whole body deformations of one character are transferred to another. Only the face/head (Figure 3-13) and horse/flamingo (Figure 3-15) examples deviate from the entire body. However, these cases demonstrate that the transfer algorithm can be executed on a more fine-grained level. Thus, one avenue of future work is developing tools that allow copy/paste operations for deformation transfer from multiple sources. Movement originating with the legs of one character, the arms of another, and the head of a third could all be transferred onto one target mesh to generate a novel animation different than any single source.

The challenge in copy/paste transfer is primarily one of developing tools to specify the proper correspondence. The algorithm I present in Chapter 4 works well for whole-body mappings between anatomically similar characters. However, it is not designed to map individual body parts and requires user-guidance in the form of matching marker locations. Ideally, a copy/paste application would incorporate segmentation, correspondence, and transfer into one intuitive interface.

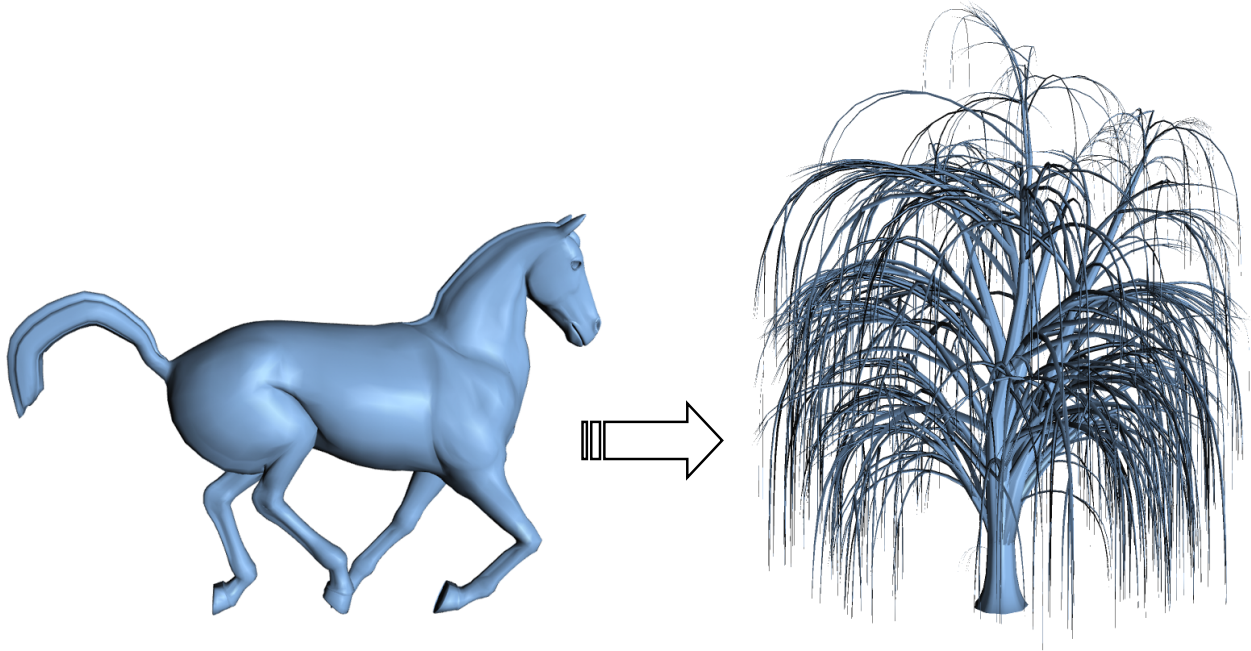


Figure 6-1: Deformation transfer between drastically different meshes is an open problem in computer graphics. Making the tree gallop like a horse requires a versatile method to relate the two meshes to one another and some way to adapt the source motion to the target shape that incorporates artistic direction.

Drastically Different Characters. Deformation transfer requires a gross similarity between the source and target. When two meshes are drastically different, their correspondence may be unclear and ambiguous even on a semantic level. Consider, for example, the horse and tree meshes shown in Figure 6-1. Transferring the galloping horse animation onto the tree would be difficult using my deformation transfer algorithm since it is not clear how to specify a correspondence between the two meshes. However, I am certain that talented animators, if asked to make the tree gallop like a horse, would succeed. In fact, this example is not unreasonable since many animated cartoons feature anthropomorphized characters.

Transferring deformation between drastically different meshes is an open problem in computer graphics that presents several challenges. It requires an extremely versatile technique to relate the source and target to one another that can accommodate ambiguous and arbitrary mappings. Second, it requires a method to appropriately *adapt* the deformation to the target, rather than simply transferring it directly without modification. Finally, this adaptation method must provide some way to incorporate artistic direction into the transfer process.

Rigging Reuse. Character rigging provides countless tested methods to model mesh kinematics. This dissertation characterizes the rigging process as time consuming and difficult and offers MESH IK as an alternative. However, the rigging controls themselves are not the enemy; it is the lengthy parameter tuning process that makes rigging so laborious. An alternate approach to mitigate the expense of character rigging is *rigging reuse*: transfer the rigging controls themselves (including the hard-to-tune parameters) from one character to another. Rigging reuse would allow animators to use familiar rigging tools and the same proven animation pipeline while avoiding the lengthy tuning process.

Deformation transfer provides a starting point for rigging reuse. The reconstruction problem can be thought of as a metric that measures how well one mesh’s deformation mimics that of another. This metric permits a reformulation to solve for other quantities that influence a character’s shape.

Localized Examples and Simplicial Modeling for MESH IK. A principle design goal of MESH IK is to allow the user to generate the most meaningful mesh deformations with the least amount of work. Thus, the presented framework focuses on selecting and moving a few vertices in order to generate a full-body deformation. For example, the lion mesh from Figure 5-5 can be made to stretch forward or sit down by fixing one vertex on its back paw and dragging a vertex on its front paw forward or backward. MESH IK finds the best pose in the nonlinear span of all examples.

In a production setting where animators require precise control over every nuance of shape, the number of examples required to express the character’s deformation may become unmanageable. Localizing the examples to individual parts of the character’s body can reduce the total example count. For example, the user may sculpt a multitude of facial expressions but only a few examples of the legs and arms. This approach avoids a combinatorial increase in the number of examples that might otherwise result when fine-grained control over the mesh kinematics is required.

Simplicial configuration modeling [Ngo et al. 2000] provides another approach to manage the example set. Rather than always searching among all examples, the user can explicitly model the configuration space of the character via a simplicial complex. This allows the user to ensure that only compatible examples are combined in any situation. A user interface can facilitate navigation of the possibly high-dimensional complex in an intuitive way [Matusik et al. 2005].

Physics. Neither deformation transfer nor MESH IK employs physics, but both have much to gain from its application. As discussed in Section 2.3.3, the physical laws that govern motion in the real world are an integral component of stylized and realistic animation alike. A substantial amount of research in computer graphics focuses on physics-based character animation and there are many opportunities to incorporate it into the problems I have presented.

Adding physics to the transfer process could ensure that the target character moves in a physically realistic way even when the size and shape of the source and target are different. Or, the physical properties of a fluid simulation, such as velocity or pressure, could be transferred instead of mesh deformation. Physics transfer would facilitate the construction of a library of physical phenomena that could be applied to new domains without the need to resimulate.

MESH IK could be modified to capture dynamic effects such as inertia and follow-through. A comprehensive treatment would introduce dynamic features and a mechanism for matching both static and dynamic feature vectors. Such a system might ultimately provide a practical compromise between the automation offered by physical simulations and the control provided by keyframing techniques.

Bibliography

7

- ABE, Y., LIU, C. K., AND POPOVIĆ, Z. 2004. Momentum-based parameterization of dynamic character motion. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 173–182.
- AGARWALA, A., DONTCHEVA, M., AGRAWALA, M., DRUCKER, S., COLBURN, A., CURLESS, B., SALESIN, D., AND COHEN, M. 2004. Interactive digital photomontage. *ACM Transactions on Graphics* 23, 3 (Aug.), 294–302.
- ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 157–164.
- ALEXA, M. 2001. Local control for mesh morphing. In *Proceedings of the International Conference on Shape Modeling and Applications (SMI'01)*, 209–215.
- ALEXA, M. 2002. Linear combination of transformations. In *Proceedings of SIGGRAPH '02*, 380–387.
- ALEXA, M. 2002. Recent advances in mesh morphing. *Computer Graphics Forum* 21, 2, 173–196.
- ALEXA, M., 2003. Personal communication.
- ALEXA, M. 2003. Differential coordinates for mesh morphing and deformation. *The Visual Computer* 19, 2, 105–114.
- ALIAS. 2005. *Maya 7 Documentation*. <http://www.alias.com/>.
- ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2002. Articulated body deformation from range scan data. *ACM Transactions on Graphics* 21, 3 (July), 612–619.
- ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2003. The space of human body shapes: Reconstruction and parameterization from range scans. *ACM Transactions on Graphics* 22, 3 (July), 587–594.

- ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2004. Exploring the space of human body shapes: Data-driven synthesis under anthropometric control. In *Proceedings of the SAE Digital Human Modeling for Design and Engineering Conference*.
- ALLIEZ, P., COHEN-STEINER, D., YVINEC, M., AND DESBRUN, M. 2005. Variational tetrahedral meshing. *ACM Transactions on Graphics* 24, 3 (Aug.), 617–625.
- ANGELIDIS, A., CANI, M.-P., WYVILL, G., AND KING, S. 2004. Swirling-sweepers: Constant-volume modeling. In *12th Pacific Conference on Computer Graphics and Applications (PG'04)*, 10–15.
- ANGUELOV, D., SRINIVASAN, P., KOLLER, D., PANG, H.-C., AND DAVIS, J. 2004. The correlated correspondence algorithm for unsupervised registration of nonrigid surfaces. In *NIPS*.
- ANGUELOV, D., SRINIVASAN, P., KOLLER, D., THRUN, S., RODGERS, J., AND DAVIS, J. 2005. SCAPE: Shape completion and animation of people. *ACM Transactions on Graphics*.
- ARIKAN, O., AND FORSYTH, D. A. 2002. Interactive motion generation from examples. *ACM Transactions on Graphics* 21, 3 (July), 483–490.
- ARONOV, B., SEIDEL, R., AND SOUVAIN, D. 1993. On compatible triangulations of simple polygons. *Computational Geometry: Theory and Applications* 3, 1, 27–35.
- AUBEL, A., AND THALMANN, D. 2001. Interactive modeling of the human musculature. In *Proceedings of Computer Animation 2001*.
- BARR, A. H. 1984. Global and local deformations of solid primitives. In *Computer Graphics (Proceedings of SIGGRAPH 84)*, vol. 18, 21–30.
- BERG, A. C., BERG, T. L., AND MALIK, J. 2005. Shape matching and object recognition using low distortion correspondences. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2005)*, 26–33.
- BLAIR, P. 1994. *Cartoon Animation*. Walter Foster Publishing.
- BLANZ, V., AND VETTER, T. 1999. A morphable model for the synthesis of 3D faces. In *Proceedings of SIGGRAPH 99, Computer Graphics Proceedings, Annual Conference Series*, 187–194.
- BLOOMENTHAL, J. 1994. An implicit surface polygonizer. In *Graphics Gems IV*, P. S. Heckbert, Ed. Academic Press, 324–349.
- BOIER-MARTIN, I., RUSHMEIER, H., AND JIN, J. 2004. Parameterization of triangle meshes over quadrilateral domains. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, ACM Press, New York, NY, USA, 193–203.
- BOTSCH, M., AND KOBBELT, L. 2003. Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum* 22, 3, 483–483.
- BOTSCH, M., AND KOBBELT, L. 2004. An intuitive framework for real-time freeform modeling. *ACM Transactions on Graphics (Proc. SIGGRAPH)* 23, 3, 630–634.

- BOTSCH, M., BOMMES, D., AND KOBBELT, L. 2005. Efficient linear system solvers for mesh processing. In *IMA Conference on the Mathematics of Surfaces*, 62–83.
- BOYKOV, Y., AND KOLMOGOROV, V. 2001. An experimental comparison of min-cut/max-flow algorithms for energy minimization in vision. In *EMMCVPR '01: Proceedings of the Third International Workshop on Energy Minimization Methods in Computer Vision and Pattern Recognition*, Springer-Verlag, London, UK, 359–374.
- BRAND, M., AND HERTZMANN, A. 2000. Style machines. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 183–192.
- BREGLER, C., LOEB, L., CHUANG, E., AND DESHPANDE, H. 2002. Turning to the masters: Motion capturing cartoons. *ACM Transactions on Graphics* 21, 3 (July), 399–407.
- BROGAN, D. C., METOYER, R. A., AND HODGINS, J. K. 1998. Dynamically simulated characters in virtual environments. *IEEE Comput. Graph. Appl.* 18, 5, 58–69.
- BRUDERLIN, A., AND WILLIAMS, L. 1995. Motion signal processing. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, 97–104.
- CALLENNEC, B. L., AND BOULIC, R. 2004. Interactive motion deformation with prioritized constraints. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 163–171.
- CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. Interactive skeleton-driven dynamic deformation. In *Proc. of SIGGRAPH*.
- CAPELL, S., GREEN, S., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2002. A multiresolution framework for dynamic deformations. In *Proceedings of the ACM SIGGRAPH Symposium on Computer Animation*.
- CAPELL, S., BURKHART, M., CURLESS, B., DUCHAMP, T., AND POPOVIĆ, Z. 2005. Physically based rigging for deformable characters. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 301–310.
- CHADWICK, J. E., HAUMANN, D. R., AND PARENT, R. E. 1989. Layered construction for deformable animated characters. In *Computer Graphics (Proceedings of SIGGRAPH 89)*, vol. 23, 243–252.
- CHEN, D. T., AND ZELTZER, D. 1992. Pump it up: Computer animation of a biomechanically based model of muscle using the finite element method. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, vol. 26, 89–98.
- CHI, D. M., COSTA, M., ZHAO, L., AND BADLER, N. I. 2000. The emote model for effort and shape. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 173–182.
- CHOI, K.-J., AND KO, H.-S. 2000. On-line motion retargeting. *The Journal of Visualization and Computer Animation* 11, 5.
- COHEN-STEINER, D., ALLIEZ, P., AND DESBRUN, M. 2004. Variational shape approximation. *ACM Transactions on Graphics* 23, 3 (Aug.), 905–914.

- COHEN, M. F. 1992. Interactive spacetime control for animation. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, vol. 26, 293–302.
- COQUILLART, S. 1990. Extended free-form deformation: A sculpturing tool for 3D geometric modeling. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, 187–196.
- CURLESS, B., AND LEVOY, M. 1996. A volumetric method for building complex models from range images. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 303–312.
- DAVIS, T. A., GILBERT, J. R., LARIMORE, S., AND NG, E. 2004. COLAMD, an approximate column minimum degree ordering algorithm. *ACM Transactions on Mathematical Software* 30, 3, 377–380.
- DAVIS, T. A. 2003. UMFPACK version 4.1 user guide. Tech. rep., University of Florida. TR-03-008.
- DEROSE, T. D., KASS, M., AND TRUONG, T. 1998. Subdivision surfaces in character animation. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 85–94.
- DO CARMO, M. 1976. *Differential Geometry of Curves and Surfaces*. Prentice Hall.
- DONG, S., BREMER, P.-T., GARLAND, M., PASCUCCI, V., AND HART, J. C. 2005. Quadrangulating a mesh using laplacian eigenvectors. Tech. Rep. UIUCDCS-R-2005-2583, University of Illinois at Urbana-Champaign, June.
- ECK, M., DEROSE, T. D., DUCHAMP, T., HOPPE, H., LOUNSBERRY, M., AND STUETZLE, W. 1995. Multiresolution analysis of arbitrary meshes. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, 173–182.
- FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 1997. Dynamic free-form deformations for animation synthesis. *IEEE Transactions on Visualization and Computer Graphics* 3, 3, 201–214.
- FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composable controllers for physics-based character animation. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 251–260.
- FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. The virtual stuntman: Dynamic characters with a repertoire of autonomous motor skills. *Computers and Graphics* 25, 6, 933–953.
- FANG, A. C., AND POLLARD, N. S. 2003. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics* 22, 3 (July), 417–426.
- FATTAL, R., AND LISCHINSKI, D. 2004. Target-driven smoke animation. *ACM Transactions on Graphics* 23, 3 (Aug.), 441–448.
- FATTAL, R., LISCHINSKI, D., AND WERMAN, M. 2002. Gradient domain high dynamic range compression. *ACM Transactions on Graphics* 21, 3 (July), 249–256.
- FERNANDO, R., AND KILGARD, M. J. 2003. *The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics*. Addison Wesley Professional.

- GARLAND, M. 1999. *Quadric-Based Polygonal Surface Simplification*. PhD thesis, Carnegie Mellon University.
- GILL, P. E., MURRAY, W., AND WRIGHT, M. H. 1989. *Practical Optimization*. Academic Press, London.
- GLEICHER, M., SHIN, H. J., KOVAR, L., AND JEPSEN, A. 2003. Snap-together motion: Assembling run-time animations. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 181–188.
- GLEICHER, M. 1997. Motion editing with spacetime constraints. In *1997 Symposium on Interactive 3D Graphics*, 139–148.
- GLEICHER, M. 1998. Retargeting motion to new characters. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 33–42.
- GLEICHER, M. 1998. Retargeting motion to new characters. In *Proceedings of ACM SIGGRAPH 1998*, Computer Graphics Proceedings, Annual Conference Series, 33–42.
- GLEICHER, M. 2001. Comparing constraint-based motion editing methods. *Graphical Models* 63, 2, 107–134.
- GOLUB, G. H., AND LOAN, C. F. V. 1996. *Matrix Computations*, third ed. Johns Hopkins University Press, Baltimore, Maryland.
- GOULD, N. I. M., AND AN J. A. SCOTT, Y. H. 2005. A numerical evaluation of sparse direct solvers for the solution of large sparse, symmetric linear systems of equations. Tech. Rep. RAL-TR-2005-005, Council for the Central Laboratory of the Research Councils.
- GREISSMAIR, J., AND PURGATHOFER, W. 1989. Deformation of solids with trivariate B-splines. In *Eurographics '89*, 137–148.
- GROCHOW, K., MARTIN, S., HERTZMANN, A., AND POPOVIĆ, Z. 2004. Style-based inverse kinematics. *ACM Transactions on Graphics (Proc. SIGGRAPH)*.
- GUSKOV, I., SWELDENS, W., AND SCHRÖDER, P. 1999. Multiresolution signal processing for meshes. In *Proceedings of ACM SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 325–334.
- GUSKOV, I., VIDIMCE, K., SWELDENS, W., AND SCHRÖDER, P. 2000. Normal meshes. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 95–102.
- HIROTA, G., MAHESHWARI, R., AND LIN, M. C. 1999. Fast volume-preserving free form deformation using multi-level optimization. In *SMA '99: Proceedings of the fifth ACM symposium on Solid modeling and applications*, ACM Press, New York, NY, USA, 234–245.
- HIROTA, G., FISHER, S., STATE, A., LEE, C., AND FUCHS, H. 2001. An implicit finite element method for elastic solids in contact. In *Proceedings of Computer Animation 2001*.
- HODGINS, J. K., AND POLLARD, N. S. 1997. Adapting simulated behaviors for new characters. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 153–162.

- HODGINS, J. K., WOOTEN, W. L., BROGAN, D. C., AND O'BRIEN, J. F. 1995. Animating human athletics. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, 71–78.
- HONG ZHU, Q., CHEN, Y., AND KAUFMAN, A. 1988. Real-time biomechanically-based muscle volume deformation using fem. *Computer Graphics Forum* 17, 3, 275–284.
- HSU, W. M., HUGHES, J. F., AND KAUFMAN, H. 1992. Direct manipulation of free-form deformations. In *Computer Graphics (Proceedings of ACM SIGGRAPH 92)*, vol. 26, 177–184.
- HSU, E., PULLI, K., AND POPOVIĆ, J. 2005. Style translation for human motion. *ACM Transactions on Graphics* 24, 3 (Aug.), 1082–1089.
- IGARASHI, T., MATSUOKA, S., AND TANAKA, H. 1999. Teddy: A sketching interface for 3d freeform design. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 409–416.
- IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. Spatial keyframing for performance-driven animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 107–115.
- IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Transactions on Graphics* 24, 3 (Aug.), 1134–1141.
- INTEL. 2004. *Math Kernel Library Reference Manual*. <http://www.intel.com/>.
- JAMES, D. L., AND TWIGG, C. D. 2005. Skinning mesh animations. *ACM Transactions on Graphics* 24, 3 (Aug.), 399–407.
- JOHNSTON, O., AND THOMAS, F. 1995. *The Illusion of Life: Disney Animation*. Disney Editions.
- JU, T. 2004. Robust repair of polygonal models. *ACM Transactions on Graphics* 23, 3 (Aug.), 888–895.
- KÄHLER, K. 2003. *A Head Model with Anatomical Structure for Facial Modeling and Animation*. PhD thesis, Max-Planck-Institut für Informatik.
- KARYPIS, G., AND KUMAR, V. 1998. METIS: Serial graph partitioning, version 4.0.1. <http://www.cs.umn.edu/~metis>.
- KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Transactions on Graphics* 22, 3 (July), 954–961.
- KAVAN, L., AND ŽÁRA, J. 2003. Real-time skin deformation with bones blending. In *WSCG Short Papers Proceedings*.
- KAVAN, L., AND ŽÁRA, J. 2005. Spherical blend skinning: A real-time deformation of articulated models. In *2005 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, ACM Press, 9–16.
- KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of ACM SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 105–114.

- KOBBELT, L., VORSATZ, J., AND SEIDEL, H.-P. 1999. Multiresolution hierarchies on unstructured triangle meshes. *Computational Geometry* 14, 1-3, 5–24.
- KOBBELT, L. P., VORSATZ, J., LABSIK, U., AND SEIDEL, H.-P. 1999. A shrink wrapping approach to remeshing polygonal surface. In *Computer Graphics Forum (Eurographics '99)*, The Eurographics Association and Blackwell Publishers, P. Brunet and R. Scopigno, Eds., vol. 18, 119–130.
- KOBBELT, L., BAREUTHER, T., AND SEIDEL, H.-P. 2000. Multiresolution shape deformations for meshes with dynamic vertex connectivity. *Computer Graphics Forum (Proc. Eurographics)* 19, C249–C260.
- KOBBELT, L. 1997. Discrete fairing. In *Proceedings of the Seventh IMA Conference on the Mathematics of Surfaces*, 101–130.
- KOMURA, T., LEUNG, H., AND KUFFNER, J. 2004. Animating reactive motions for biped locomotion. In *VRST '04: Proceedings of the ACM symposium on Virtual reality software and technology*, ACM Press, New York, NY, USA, 32–40.
- KONDO, R., KANAI, T., AND ICHI ANJYO, K. 2005. Directable animation of elastic objects. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 127–134.
- KOVAR, L., AND GLEICHER, M. 2001. Simplicial families of drawings. In *UIST '01: Proceedings of the 14th annual ACM symposium on User interface software and technology*, ACM Press, New York, NY, USA, 163–172.
- KOVAR, L., AND GLEICHER, M. 2003. Flexible automatic motion blending with registration curves. In *2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 214–224.
- KOVAR, L., GLEICHER, M., AND PIGHIN, F. 2002. Motion graphs. *ACM Transactions on Graphics* 21, 3 (July), 473–482.
- KRAEVOY, V., AND SHEFFER, A. 2004. Cross-parameterization and compatible remeshing of 3D models. *ACM Transactions on Graphics* 23, 3 (Aug.), 861–869.
- KRY, P. G., JAMES, D. L., AND PAI, D. K. 2002. EigenSkin: Real time large deformation character skinning in hardware. In *Proceedings of the Symposium on Computer Animation*, ACM Press, 153–159.
- LAI, W. M., RUBIN, D., AND KREMPL, E. 1993. *Introduction to Continuum Mechanics*, 3rd ed. Elsevier.
- LASSETER, J. 1987. Principles of traditional animation applied to 3D computer animation. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, vol. 21, 35–44.
- LASSETER, J. 2001. Tricks to animating characters with a computer. *SIGGRAPH Comput. Graph.* 35, 2, 45–47.
- LASZLO, J., VAN DE PANNE, M., AND FIUME, E. L. 2000. Interactive control for physically-based animation. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 201–208.

- LASZLO, J., NEFF, M., AND SINGH, K. 2005. Predictive feedback for interactive control of physics-based characters. In *Computer Graphics Forum, Proceedings of Eurographics 2005*, Blackwell, Dublin, Ireland, vol. 24, Eurographics, 257–265.
- LEE, J., AND LEE, K. H. 2004. Precomputing avatar behavior from human motion data. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 79–87.
- LEE, J., AND SHIN, S. Y. 1999. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 39–48.
- LEE, Y., TERZOPOULOS, D., AND WATERS, K. 1995. Realistic modeling for facial animation. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, 55–62.
- LEE, A. W. F., SWELDENS, W., SCHRÖDER, P., COWSAR, L., AND DOBKIN, D. 1998. MAPS: Multiresolution adaptive parameterization of surfaces. In *Proceedings of SIGGRAPH 98*, Computer Graphics Proceedings, Annual Conference Series, 95–104.
- LEE, A., MORETON, H., AND HOPPE, H. 2000. Displaced subdivision surfaces. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 85–94.
- LEE, J., CHAI, J., REITSMA, P. S. A., HODGINS, J. K., AND POLLARD, N. S. 2002. Interactive control of avatars animated with human motion data. *ACM Transactions on Graphics* 21, 3 (July), 491–500.
- LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformations: A unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 165–172.
- LEWIS, J. P., MOOSER, J., DENG, Z., AND NEUMANN, U. 2005. Reducing blendshape interference by selected motion attenuation. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 25–29.
- LIEPA, P. 2003. Filling holes in meshes. In *SGP '03: Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 200–205.
- LINDHOLM, E., KILGARD, M. J., AND MORETON, H. 2001. A user-programmable vertex engine. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 149–158.
- LIPMAN, Y., SORKINE, O., COHEN-OR, D., LEVIN, D., RÖSSL, C., AND SEIDEL, H.-P. 2004. Differential coordinates for interactive mesh editing. In *Proceedings of Shape Modeling International*.
- LIPMAN, Y., SORKINE, O., ALEXA, M., COHEN-OR, D., LEVIN, D., RÖSSL, C., AND SEIDEL, H.-P. 2005. Laplacian framework for interactive mesh editing. *International Journal of Shape Modeling* 11, 1, 43–61.
- LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph.* 24, 3, 479–487.

- LIU, C. K., AND POPOVIĆ, Z. 2002. Synthesis of complex dynamic character motion from simple animations. *ACM Transactions on Graphics* 21, 3 (July), 408–416.
- LIU, Z., GORTLER, S. J., AND COHEN, M. F. 1994. Hierarchical spacetime control. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, 35–42.
- LIU, P.-C., WU, F.-C., MA, W.-C., LIANG, R.-H., AND OUHYOUNG, M. 2003. Automatic animation skeleton construction using repulsive force field. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, 409.
- LIU, C. K., HERTZMANN, A., AND POPOVIĆ, Z. 2005. Learning physics-based motion style with nonlinear inverse optimization. *ACM Transactions on Graphics* 24, 3 (Aug.), 1071–1081.
- LLAMAS, I., KIM, B., GARGUS, J., ROSSIGNAC, J., AND SHAW, C. D. 2003. Twister: A space-warp operator for the two-handed editing of 3D shapes. *ACM Transactions on Graphics* 22, 3 (July), 663–668.
- LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3D surface construction algorithm. In *Computer Graphics (Proceedings of SIGGRAPH 87)*, vol. 21, 163–169.
- LOUNSBERY, M., DEROSE, T. D., AND WARREN, J. 1997. Multiresolution analysis for surfaces of arbitrary topological type. *ACM Transactions on Graphics* 16, 1 (Jan.), 34–73.
- LOYALL, A. B., REILLY, W. S. N., BATES, J., AND WEYHRAUCH, P. 2004. System for authoring highly interactive, personality-rich interactive characters. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 59–68.
- MACCRACKEN, R., AND JOY, K. I. 1996. Free-form deformations with lattices of arbitrary topology. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, ACM Press, New York, NY, USA, 181–188.
- MADSEN, K., NIELSEN, H., AND TINGLEFF, O. 2004. Methods for non-linear least squares problems. Tech. rep., Informatics and Mathematical Modelling, Technical University of Denmark.
- MATUSIK, W., ZWICKER, M., AND DURAND, F. 2005. Texture design using a simplicial complex of morphable textures. *ACM Transactions on Graphics* 24, 3 (Aug.), 787–794.
- MCMANARA, A., TREUILLE, A., POPOVIĆ, Z., AND STAM, J. 2004. Fluid control using the adjoint method. *ACM Transactions on Graphics* 23, 3 (Aug.).
- MEREDITH, M., AND MADDOCK, S. 2005. Adapting motion capture data using weighted real-time inverse kinematics. *Computers in Entertainment (CIE)* 3, 1, 5–5.
- MEYER, M., DESBRUN, M., SCHRÖDER, P., AND BARR, A. H. 2003. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and Mathematics III*, H.-C. Hege and K. Polthier, Eds. Springer-Verlag, Heidelberg, 35–57.
- MINKA, T. P. 2000. Old and new matrix algebra useful for statistics. Tech. rep., MIT Media Lab.
- MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph.* 22, 3, 562–568.

- MOHR, A., TOKHEIM, L., AND GLEICHER, M. 2003. Direct manipulation of interactive character skins. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, ACM Press, New York, NY, USA, 27–30.
- MONZANI, J.-S., BAERLOCHER, P., BOULIC, R., AND THALMANN, D. 2000. Using an intermediate skeleton and inverse kinematics for motion retargeting. *Computer Graphics Forum* 19, 3 (August).
- MURRAY, R. M., LI, Z., AND SASTRY, S. S. 1994. *A mathematical introduction to robotic manipulation*. CRC Press.
- NANCY S. POLLARD, P. S. A. R. 2001. Animation of humanlike characters: Dynamic motion filtering with a physically plausible contact model. In *Yale Workshop on Adaptive and Learning Systems*.
- NEALEN, A., SORKINE, O., ALEXA, M., AND COHEN-OR, D. 2005. A sketch-based interface for detail-preserving mesh editing. *ACM Transactions on Graphics* 24, 3 (Aug.), 1142–1147.
- NEFF, M., AND FIUME, E. 2002. Modeling tension and relaxation for computer animation. In *ACM SIGGRAPH Symposium on Computer Animation*, 81–88.
- NEFF, M., AND FIUME, E. 2003. Aesthetic edits for character animation. In *2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 239–244.
- NEFF, M., AND FIUME, E. 2004. Methods for exploring expressive stance. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 49–58.
- NEFF, M., AND FIUME, E. 2005. AER: Aesthetic exploration and refinement for expressive character animation. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 161–170.
- NGO, J. T., AND MARKS, J. 1993. Spacetime constraints revisited. In *Proceedings of SIGGRAPH 93, Computer Graphics Proceedings, Annual Conference Series*, 343–350.
- NGO, T., CUTRELL, D., DANA, J., DONALD, B., LOEB, L., AND ZHU, S. 2000. Accessible animation and customizable graphics via simplicial configuration modeling. In *Computer Graphics (Proceedings of SIGGRAPH 2000)*, ACM SIGGRAPH, 403–410.
- NOH, J., AND NEUMANN, U. 2001. Expression cloning. In *Proceedings of ACM SIGGRAPH 2001, Computer Graphics Proceedings, Annual Conference Series*, 277–288.
- NOORUDDIN, F. S., AND TURK, G. 2003. Simplification and repair of polygonal models using volumetric techniques. *IEEE Transactions on Visualization and Computer Graphics* 9, 2, 191–205.
- O'ROURKE, J. 2000. *Computational Geometry in C*. Cambridge University Press.
- PARK, S. I., SHIN, H. J., AND SHIN, S. Y. 2002. On-line locomotion generation based on motion blending. In *ACM SIGGRAPH Symposium on Computer Animation*, 105–112.
- PÉREZ, P., GANGNET, M., AND BLAKE, A. 2003. Poisson image editing. *ACM Transactions on Graphics* 22, 3 (July), 313–318.

- PERLIN, K., AND GOLDBERG, A. 1996. Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 205–216.
- PERLIN, K. 1995. Real time responsive animation with personality. *IEEE Transactions on Visualization and Computer Graphics* 1, 1, 5–15.
- PIEGL, L. 1991. On NURBS: A survey. *IEEE Comput. Graph. Appl.* 11, 1, 55–71.
- POLLARD, N. S., AND BEHMARAM-MOSAVAT, F. 2000. Force-based motion editing for locomotion tasks. In *Proceedings of the 2000 IEEE International Conference on Robotics and Automation (ICRA 2000)*, 663–669.
- POPOVIĆ, Z., AND WITKIN, A. P. 1999. Physically based motion transformation. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, 11–20.
- POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. P. 2000. Interactive manipulation of rigid body simulations. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, 209–218.
- POPOVIĆ, J., SEITZ, S. M., AND ERDMANN, M. 2003. Motion sketching for control of rigid-body simulations. *ACM Transactions on Graphics* 22, 4 (Oct.), 1034–1054.
- PRATSCHER, M., COLEMAN, P., LASZLO, J., AND SINGH, K. 2005. Outside-in anatomy based character rigging. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 329–338.
- PRAUN, E., SWELDENS, W., AND SCHRÖDER, P. 2001. Consistent mesh parameterizations. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, 179–184.
- RAIBERT, M. H., AND HODGINS, J. K. 1991. Animation of dynamic legged locomotion. In *Computer Graphics (Proceedings of SIGGRAPH 91)*, vol. 25, 349–358.
- RAIBERT, M. H. 1986. *Legged robots that balance*. Massachusetts Institute of Technology, Cambridge, MA, USA.
- RAPPOPORT, A., SHEFFER, A., AND BERCOVIER, M. 1996. Volume-preserving free-form solids. *IEEE Transactions on Visualization and Computer Graphics* 2, 1, 19–27.
- REITSMA, P. S. A., AND POLLARD, N. S. 2004. Evaluating motion graphs for character navigation. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 89–98.
- ROSE, C. F., GUENTER, B., BODENHEIMER, B., AND COHEN, M. F. 1996. Efficient generation of motion transitions using spacetime constraints. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, 147–154.
- ROSE, C., COHEN, M. F., AND BODENHEIMER, B. 1998. Verbs and adverbs: Multidimensional motion interpolation. *IEEE Computer Graphics and Applications* 18, 5, 32–40.
- SAFONOVA, A., HODGINS, J. K., AND POLLARD, N. S. 2004. Synthesizing physically realistic human motion in low-dimensional, behavior-specific spaces. *ACM Transactions on Graphics* 23, 3 (Aug.), 514–521.

- SAND, P., McMILLAN, L., AND POPOVIĆ, J. 2003. Continuous capture of skin deformation. *ACM Transactions on Graphics* 22, 3 (July), 578–586.
- SCHARSTEIN, D., AND SZELISKI, R. 2002. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International Journal of Computer Vision* 47, 1-3, 7–42.
- SCHEEPERS, F., PARENT, R. E., CARLSON, W. E., AND MAY, S. F. 1997. Anatomy-based modeling of the human musculature. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 163–172.
- SCHREINER, J., ASIRVATHAM, A., PRAUN, E., AND HOPPE, H. 2004. Inter-surface mapping. *ACM Transactions on Graphics* 23, 3 (Aug.), 870–877.
- SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, vol. 20, 151–160.
- SEO, H., AND MAGNENAT-THALMANN, N. 2003. An automatic modeling of human bodies from sizing parameters. In *2003 ACM Symposium on Interactive 3D Graphics*, 19–26.
- SEO, H., CORDIER, F., AND MAGNENAT-THALMANN, N. 2003. Synthesizing animatable body models with parameterized shape modifications. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 120–125.
- SHAPIRO, A., PIGHIN, F., AND FALOUTSOS, P. 2003. Hybrid control for interactive character animation. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, 455.
- SHARF, A., ALEXA, M., AND COHEN-OR, D. 2004. Context-based surface completion. *ACM Transactions on Graphics* 23, 3 (Aug.), 878–887.
- SHEFFER, A., AND KRAEVOY, V. 2004. Pyramid coordinates for morphing and deformation. In *Proceedings of the 2nd Symposium on 3D Processing, Visualization and Transmission*.
- SHEWCHUK, J. R. 1998. A condition guaranteeing the existence of higher-dimensional constrained delaunay triangulations. In *SCG '98: Proceedings of the fourteenth annual symposium on Computational geometry*, ACM Press, New York, NY, USA, 76–85.
- SHI, L., AND YU, Y. 2005. Taming liquids for rapidly changing targets. In *SCA '05: Proceedings of the 2005 ACM SIGGRAPH/Eurographics symposium on Computer animation*, ACM Press, New York, NY, USA, 229–236.
- SHIN, H. J., LEE, J., GLEICHER, M., AND SHIN, S. Y. 2001. Computer puppetry: An importance-based approach. *ACM Transactions on Graphics* 20, 2 (Apr.), 67–94.
- SHIN, H. J., KOVAR, L., AND GLEICHER, M. 2003. Physical touch-up of human motions. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, IEEE Computer Society, Washington, DC, USA, 194–203.

- SHOEMAKE, K., AND DUFF, T. 1992. Matrix animation and polar decomposition. In *Proceedings of Graphics Interface '92*, 259–264.
- SINGH, K., AND FIUME, E. L. 1998. Wires: A geometric deformation technique. In *Proceedings of ACM SIGGRAPH 1998*, Computer Graphics Proceedings, Annual Conference Series, 405–414.
- SINGH, K., AND KOKKEVIS, E. 2000. Skinning characters using surface oriented free-form deformations. In *Graphics Interface*, 35–42.
- SLOAN, P.-P. J., ROSE, III, C. F., AND COHEN, M. F. 2001. Shape by example. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, ACM Press, 135–143.
- SORKINE, O., AND COHEN-OR, D. 2004. Least-squares meshes. In *Proceedings of Shape Modeling International*, IEEE Computer Society Press, 191–199.
- SORKINE, O., LIPMAN, Y., COHEN-OR, D., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Proceedings of the Eurographics / ACM SIGGRAPH symposium on Geometry processing*, 179–188.
- SORKINE, O. 2005. State-of-the-art report: Laplacian mesh processing. In *Eurographics 2005—State of the Art Reports*, The Eurographics Association, Dublin, Ireland, Eurographics, 53–70.
- SULEJMANPAŠIĆ, A., AND POPOVIĆ, J. 2005. Adaptation of performed ballistic motion. *ACM Transactions on Graphics* 24, 1 (Jan.), 165–179.
- SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Transactions on Graphics* 23, 3 (Aug.), 399–405.
- SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *ACM Transactions on Graphics* 24, 3 (Aug.), 488–495.
- SUN, J., ZHENG, N.-N., AND SHUM, H.-Y. 2003. Stereo matching using belief propagation. *IEEE Trans. Pattern Anal. Mach. Intell.* 25, 7, 787–800.
- TAK, S., AND KO, H.-S. 2005. A physically-based motion retargeting filter. *ACM Transactions on Graphics* 24, 1 (Jan.), 98–117.
- TAK, S., YOUNG SONG, O., AND KO, H.-S. 2000. Motion balance filtering. *Computer Graphics Forum* 19, 3 (August), 437–446.
- TERAN, J., BLEMKER, S., HING, V. N. T., AND FEDKIW, R. 2003. Finite volume methods for the simulation of skeletal muscle. In *2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 68–74.
- TERAN, J., SIFAKIS, E., BLEMKER, S. S., NG-THOW-HING, V., LAU, C., AND FEDKIW, R. 2005. Creating and simulating skeletal muscle from the visible human data set. *IEEE Transactions on Visualization and Computer Graphics* 11, 3, 317–328.
- TERRA, S. C. L., AND METOYER, R. A. 2004. Performance timing for keyframe animation. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 253–258.
- THORNE, M., BURKE, D., AND VAN DE PANNE, M. 2004. Motion doodles: An interface for sketching character motion. *ACM Transactions on Graphics* 23, 3 (Aug.), 424–431.

- TIPPING, M. E., AND BISHOP, C. M. 1999. Probabilistic principal component analysis. *Journal of the Royal Statistical Society, Series B* 61, 3, 611–622.
- TOLEDO, S., 2003. TAUCS: A library of sparse linear solvers, version 2.2. <http://www.tau.ac.il/~stoledo/taucs>.
- TREUILLE, A., MCNAMARA, A., POPOVIĆ, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Transactions on Graphics* 22, 3 (July), 716–723.
- UNUMA, M., ANJYO, K., AND TAKEUCHI, R. 1995. Fourier principles for emotion-based human figure animation. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, 91–96.
- VAN DE PANNE, M., FIUME, E., AND VRANESIC, Z. 1990. Reusable motion synthesis using state-space controllers. In *Computer Graphics (Proceedings of SIGGRAPH 90)*, vol. 24, 225–234.
- VICON. 2004. *Vicon iQ 2.0 Reference Manual*. <http://www.vicon.com/>.
- VLASIC, D., BRAND, M., PFISTER, H., AND POPOVIĆ, J. 2005. Face transfer with multilinear models. *ACM Trans. Graph.* 24, 3, 426–433.
- WADE, L., AND PARENT, R. E. 2002. Automated generation of control skeletons for use in animation. *The Visual Computer* 18, 2.
- WANG, X. C., AND PHILLIPS, C. 2002. Multi-weight enveloping: Least-squares approximation techniques for skin animation. In *ACM SIGGRAPH Symposium on Computer Animation*, 129–138.
- WEBER, J. 2000. Run-time skin deformation. In *Proceedings of the 2000 Game Developers Conference*. <http://www.gdconf.com/archives/2000/>.
- WEINSTOCK, R. 1974. *Calculus of Variations*. Dover Publications, Inc.
- WELCH, W., AND WITKIN, A. 1992. Variational surface modeling. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, vol. 26, 157–166.
- WHITAKER, H., AND HALAS, J. 2002. *Timing for Animation*. Focal Press.
- WILHELMS, J., AND GELDER, A. V. 1997. Anatomically based modeling. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, 173–180.
- WILHELMS, J. 1997. Animals with anatomy. *IEEE Comput. Graph. Appl.* 17, 3, 22–30.
- WITKIN, A., AND KASS, M. 1988. Spacetime constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, vol. 22, 159–168.
- WITKIN, A. P., AND POPOVIĆ, Z. 1995. Motion warping. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, 105–108.
- WOOTEN, W. L., AND HODGINS, J. K. 1996. Animation of human diving. *Computer Graphics Forum* 15, 1, 3–14.

- WOOTEN, W. 1998. *Simulation of leaping, tumbling, landing, and balancing humans*. PhD thesis, Georgia Institute of Technology.
- XU, D., ZHANG, H., WANG, Q., AND BAO, H. 2005. Poisson shape interpolation. In *Proceedings of ACM Symposium on Solid and Physical Modeling*.
- YAMANE, K., AND NAKAMURA, Y. 2003. Dynamics Filter—Concept and Implementation of On-line Motion Generator for Human Figures. *IEEE Transactions on Robotics and Automation* 19, 9 (jun), 421–432.
- YANG, P.-F., LASZLO, J., AND SINGH, K. 2004. Layered dynamic control for interactive character swimming. In *2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 39–47.
- YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Transactions on Graphics* 23, 3 (Aug.), 644–651.
- ZAYER, R., RÖSSL, C., KARNI, Z., AND SEIDEL, H.-P. 2005. Harmonic guidance for surface deformation. In *Computer Graphics Forum, Proceedings of Eurographics 2005*, Blackwell, Dublin, Ireland, vol. 24, Eurographics, 601–610.
- ZHANG, L., SNAVELY, N., CURLESS, B., AND SEITZ, S. M. 2004. Spacetime faces: High-resolution capture for modeling and animation. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 548–558.
- ZHAO, J., AND BADLER, N. I. 1994. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Transactions on Graphics* 13, 4 (Oct.), 313–336.
- ZHAO, P., AND VAN DE PANNE, M. 2005. User interfaces for interactive control of physics-based 3D characters. In *Proceedings of the 2005 symposium on Interactive 3D graphics and games*, ACM Press, New York, NY, USA, 87–94.
- ZORDAN, V. B., AND HODGINS, J. K. 1999. Tracking and modifying upper-body human motion data with dynamic simulation. In *Computer Animation and Simulation '99*.
- ZORDAN, V. B., AND HODGINS, J. K. 2002. Motion capture-driven simulations that hit and react. In *ACM SIGGRAPH Symposium on Computer Animation*, 89–96.
- ZORDAN, V. B., MAJKOWSKA, A., CHIU, B., AND FAST, M. 2005. Dynamic response for motion capture animation. *ACM Transactions on Graphics* 24, 3 (Aug.), 697–701.
- ZORIN, D., AND SCHRÖDER, P. 2000. Subdivision for modeling and animation. SIGGRAPH 2000 Course notes.
- ZORIN, D., SCHRÖDER, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. *Computer Graphics* 31, 259–268.
- ZWICKER, M., PAULY, M., KNOLL, O., AND GROSS, M. 2002. Pointshop 3D: An interactive system for point-based surface editing. *ACM Transactions on Graphics* 21, 3 (July), 322–329.