Embedded Deformation for Shape Manipulation

Robert W. Sumner Johannes Schmid Mark Pauly

Applied Geometry Group, ETH Zurich

Abstract

We present an algorithm that generates natural and intuitive deformations via direct manipulation for a wide range of shape representations and editing scenarios. Our method builds a space deformation represented by a collection of affine transformations organized in a graph structure. One transformation is associated with each graph node and applies a deformation to the nearby space. Positional constraints are specified on the points of an embedded object. As the user manipulates the constraints, a nonlinear minimization problem is solved to find optimal values for the affine transformations. Feature preservation is encoded directly in the objective function by measuring the deviation of each transformation from a true rotation. This algorithm addresses the problem of "embedded deformation" since it deforms space through direct manipulation of objects embedded within it, while preserving the embedded objects' features. We demonstrate our method by editing meshes, polygon soups, mesh animations, and animated particle systems.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—Modeling packages

Keywords: Geometric modeling, Deformation, Shape editing

1 Introduction

Direct manipulation has proven to be an invaluable tool for mesh editing since it provides an intuitive way for the user to interact with a mesh during the modeling process. Sophisticated deformation algorithms propagate the user's changes throughout the mesh so that features are deformed in a natural way. However, modeling is only one of the many instances in which a user must interact with a computer-generated object. Likewise, meshes are but one of many representations in use today. While recent algorithms provide powerful manipulation paradigms for mesh modeling, few apply to other manipulation tasks or geometry representations.

Our work endeavors to extend the intuitive nature of mesh modeling beyond the realm of meshes. Ultimately, direct manipulation with natural feature deformation should apply to anything that can be embedded in space. We refer to this overall problem as "embedded deformation" since the algorithm must deform space through direct manipulation of objects embedded within it, while preserving the embedded objects' features. With this goal in mind, we propose an algorithm motivated by the following principles:

Generality. In order to accommodate a wide range of shape representations, we incorporate a deformation model based on space deformation that provides a global remapping of the ambient space. Any geometric primitive embedded in this space can be deformed.

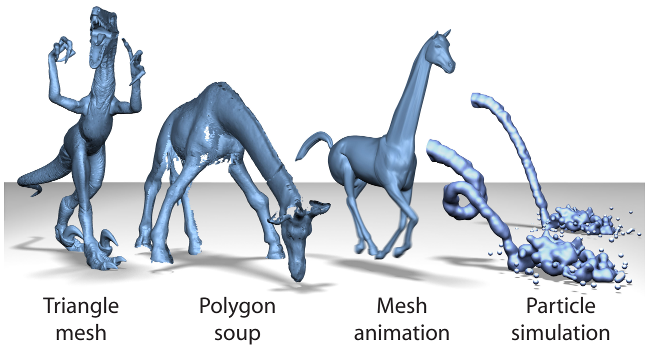

Figure 1: *Embedded deformation of several shape representations.*

The space deformation in our algorithm is defined by a collection of affine transformations, each of which induces a deformation on the nearby space. Primitives are deformed by blending the effect of transformations with overlapping influence.

Efficiency. Since the geometric complexity of objects can be enormous, efficiency considerations dictate a reduced deformable model that separates the complexity of the deformation algorithm from the complexity of the geometry. We propose a reduced model called a "deformation graph" that is simple, general, and independent of any particular geometry representation. A deformation graph consists of nodes connected by undirected edges. One affine transformation is associated with each node so that the nodes provide spatial organization to the resulting deformation. Graph edges connect nodes of overlapping influence and provide a means for information exchange so that a globally consistent deformation can be found. Due to its simple structure, there are many ways to build a deformation graph including point sampling, simplification, particle tracing, or even hand design.

Detail preservation. Detail preservation is a well-established goal of any editing application: small-scale details should be preserved when a broad change in shape is made. Practically, this requirement means that local features should rotate during deformation, rather than stretch or shear. Applying this criterion to the deformation graph framework is straightforward. Since the affine transformations associated with the graph nodes represent localized deformations, details are best preserved when these transformations represent rotations.

Direct manipulation. We formulate deformation as an optimization problem in which positional constraints are specified on points that define an embedded object. In general, any point in space can be constrained to move to any other point. As the user manipulates the constraints, the algorithm finds optimal values for the affine transformations. Detail preservation is encoded directly in the objective function by measuring the deviation of each transformation from a true rotation. A regularization term ensures that neighboring transformations are consistent with respect to one another.

Our framework has a number of advantages. Unlike previous methods, our deformation algorithm is independent of both the shape's representation and its geometric complexity while still providing intuitive detail preserving edits via direct manipulation. Since feature rotation is encoded directly in the optimization procedure, natural edits are achieved solely through positional constraints. More cumbersome frame transformations are not required. The simplicity and flexibility of the deformation graph make it easy to construct, since a rough distribution of nodes in the region that the user wishes to modify is sufficient. Although the optimization is nonlinear, complex edits can be achieved with only a few hundred nodes. Thus, the number of unknowns is small compared to the geometric complexity of the embedded object. With our efficient numerical implementation, even very detailed shapes can be edited interactively.

Our primary contribution is a novel deformation representation and optimization procedure that unites the proven paradigms of direct manipulation and detail preservation with the flexibility of space deformations. We highlight the conceptual challenge of embedded deformation and provide a solution that expands intuitive editing to situations where it was previously lacking. Our method accommodates traditional meshes with multiple connected components, polygon soups, point-based models with no connectivity information, and mesh animations. Our system also allows the user to interactively sculpt the result of a simulated particle system, easily creating effects that would be cumbersome and costly to achieve by tweaking simulation parameters (Figure 1).

2 Background

Early work in shape modeling focuses on space deformations [Barr 1984] that provide a global remapping of space. Free-form deformation (FFD) [Sederberg and Parry 1988] parameterizes a space deformation with a 3D lattice and provides an efficient way to apply *coarse* deformations to *complex* shapes. However, achieving a fine-scale deformation may require a detailed, hand-designed control lattice [Coquillart 1990; MacCracken and Joy 1996] and an inordinate amount of user manipulation. Although more intuitive control can be provided through direct manipulation [Hsu et al. 1992], the user is still restricted by the expressibility of the FFD algorithm.

With their "Wires" concept, Singh and Fiume [1998] present a flexible and effective space deformation algorithm motivated by armatures used in traditional sculpting. A collection of space curves tracks deformable features of an object, providing a coarse approximation to the shape and a means to deform it. Singh and Kokkevis [2000] generalize this concept to a polygon-based deformer. In both cases, the user interacts solely with the proxy curves or polygons rather than directly with the object being deformed. Rotations, scales, and translations are inferred from the user interaction and applied to the object. These methods give the user powerful tools to design deformations and add detail to a shape. However, they are not well suited to modify shapes that *already* are highly detailed since the user must design armature curves or control polygons that conform to details at the proper scale in order for the deformation heuristics to generate acceptable results.

Due to the widespread availability of very detailed scanned meshes, recent research focuses on high-quality mesh editing through intuitive user interfaces. Detail preservation is a central goal of such algorithms. Multiresolution methods achieve detail-preserving edits at varying scales by generating a hierarchy of simplified meshes together with corresponding detail coefficients [Kobbelt et al. 1998; Botsch and Kobbelt 2004]. While models with large geometric details may lead to local self-intersections or other artifacts [Botsch et al. 2006b], the modeling metaphor presented by Kobbelt and colleagues [1998] in which a region-of-interest and handle region are defined directly on the mesh is especially notable as it has been applied in nearly every subsequent mesh editing paper.

Algorithms based on differential representations extract local shape properties, such as curvature, scale, and orientation. By representing a mesh in terms of these values, editing can be phrased as an energy minimization problem that strives to preserve them [Sorkine 2005]. Methods that perform only linear system solves require heuristics or other special treatment of feature rotation, since natural shape deformation is inherently nonlinear [Botsch and Sorkine 2007]. Volumetric methods (e.g., [Zhou et al. 2005; Shi et al. 2006]) build a dissection of the interior and nearby exterior space for better volume preservation, while subspace methods [Huang et al. 2006] build a subspace structure for increased efficiency and stability. Nonlinear methods (e.g., [Sheffer and Kraevoy 2004; Huang et al. 2006; Botsch et al. 2006a]) yield the highest quality edits, although at higher computational costs.

These algorithms provide powerful tools for detail-preserving mesh editing. However, these and other mesh editing techniques do not meet the goals of embedded deformation since the deformation algorithm is intimately tied to the shape representation. For example, in the method presented by Huang and colleagues [2006], detail preservation is expressed as a mesh-based Laplacian energy that is computed in terms of vertices and their one-ring neighbors. The work of Shi and colleagues [2006] and Zhou and colleagues [2005] both use a Laplacian energy term based on a mesh representation. The prism-based technique of Botsch and colleagues [2006a] uses a deformation energy defined through a coupling of prisms along mesh edges and requires a mesh representation with consistent connectivity. These techniques do not apply to non-meshes, such as point-based representations, particle systems, or polygon soups where no connectivity structure can be assumed.

With our method, we adapt the intuitive click-and-drag modeling metaphor used in mesh editing to the context of space deformations. Like Wires [Singh and Fiume 1998] and its polygon-based extension [Singh and Kokkevis 2000], our method is not tied to one particular representation and can be applied to any primitive defined by points in 3D. However, unlike Wires or other space deformation algorithms that do not explicitly preserve details [Hsu et al. 1992; Botsch and Kobbelt 2005], we successfully formulate detail preservation within the space deformation framework. The complexity of our deformation graph is independent of the complexity of the shape being edited so that our technique can handle detailed shapes interactively. The graph need not be a volumetric dissection and is simpler to construct than the volumetric or subspace structures used by previous methods. The optimization problem is nonlinear and exhibits comparable quality to nonlinear mesh-based algorithms with less computational cost. Thus, our algorithm combines the flexibility of space deformations to deform any primitive independent of its geometric complexity with a simple and intuitive click-and-drag interface and high-quality detail preservation.

3 Deformation Graph

The primary challenge of embedded deformation is to find a deformation model that is general enough to apply to any object embedded in space yet still provides intuitive direct manipulation, natural feature preservation, and efficiency. We meet these goals with a novel reduced deformable model called a "deformation graph" that can express complex deformations of a variety of shape representations. In this model, a space deformation is defined by a collection of affine transformations. One transformation is associated with each node of a graph embedded in $\mathbb{R}^3$, so that the graph provides spatial organization to the deformation. Each affine transformation induces a localized deformation on the nearby space. Undirected edges connect nodes of overlapping influence to indicate local dependencies. The node positions are given by $\mathbf{g}_j \in \mathbb{R}^3, j \in 1 \dots m$, and the set $\mathcal{N}(j)$ consists of all nodes that share an edge with node j. The affine transformation for node j is specified by a 3×3 matrix $\mathbf{R}_j$ and a 3×1 translation vector $\mathbf{t}_j$. The influence of the transformation is centered at the node's position so that it maps any point $\mathbf{p}$ in $\mathbb{R}^3$ to the position $\tilde{\mathbf{p}}$ according to

$$\tilde{\mathbf{p}} = \mathbf{R}_j(\mathbf{p} - \mathbf{g}_j) + \mathbf{g}_j + \mathbf{t}_j. \qquad (1)$$

A deformed version of the graph itself is computed by applying each affine transformation to its corresponding node. Since $\mathbf{g}_j - \mathbf{g}_j$

is the zero vector, the deformed position $\tilde{\mathbf{g}}_j$ of node j is simply equal to $\mathbf{g}_j + \mathbf{t}_j$.

More interestingly, the deformation graph can be used to deform any geometric model defined by vertices in $\mathbb{R}^3$. Since transferring the deformation to an embedded shape requires computation proportional to the shape's complexity, efficiency is of paramount importance. Consequentially, we employ an algorithm similar to the widely used and highly efficient skeleton-subspace deformation from character animation. The influence of individual graph nodes is smoothly blended so that the deformed position $\tilde{\mathbf{v}}_i$ of each shape vertex $\mathbf{v}_i$, $i \in 1 \dots n$, is a weighted sum of its position after application of the deformation graph affine transformations:

$$\tilde{\mathbf{v}}_i = \sum_{j=1}^{m} w_j(\mathbf{v}_i) \left[\mathbf{R}_j(\mathbf{v}_i - \mathbf{g}_j) + \mathbf{g}_j + \mathbf{t}_j \right]. \qquad (2)$$

While linear blending may result in some artifacts for extremely coarse graphs, they are negligible for moderately dense ones like those shown in our examples. This result is not surprising, since only a few extra joint transformations are needed to greatly reduce artifacts exhibited by skeleton-subspace deformation [Weber 2000; Mohr and Gleicher 2003]. In our case, the nodes are evenly distributed over the entire shape so that the blended transformations are very similar to one another.

Normals are transformed similarly, according to the weighted sum of each normal transformed by the inverse transpose of the node transformations, and then renormalized:

$$\tilde{\mathbf{n}}_i = \sum_{j=1}^{m} w_j(\mathbf{v}_i) \mathbf{R}_j^{-1\top} \mathbf{n}_i. \qquad (3)$$

The weights $w_j(\mathbf{v}_i)$, $j \in 1 \dots m$, are spatially varying and thus depend on the vertex position. Due to the graph structure, transformations that are close to one another will be the most similar. Thus, for consistency and efficiency, we limit the influence of the deformation graph on a particular vertex to the k-nearest nodes. The weights for each vertex are precomputed according to

$$w_j(\mathbf{v}_i) = (1 - \|\mathbf{v}_i - \mathbf{g}_j\| / d_{\max})^2 \qquad (4)$$

and then normalized to sum to one. Here, $d_{\max}$ is the distance to the $k+1$-nearest node. We use $k = 4$ for all examples, except the experiment in Figure 5 (d), where $k = 8$.

The layout of the deformation graph nodes should roughly conform to the shape of the model being edited. In our experiments, a uniform sampling of the model surface produces the best results. Such a sampling is easily accomplished by distributing points densely over the surface, and repeatedly removing all points within a given radius of a randomly chosen one until a desired sampling density is reached. For meshes, simplification algorithms also produce good results when the triangle aspect ratio is restricted to avoid long and skinny triangles. For particle simulations, a simple and efficient construction of the node layout can be achieved by sampling particle paths through time. The number of graph nodes determines the expressibility of the deformation graph. Coarse edits can be made with a small number of nodes, while highly detailed ones require denser sampling. We find that full body pose changes are effectively accomplished with 200 to 300 nodes.

Graph edges connect nodes of overlapping influence and are used to enforce consistency in the overall deformation. Once the node positions are determined, the connectivity is computed automatically by creating an edge between any two nodes that influence the same vertex. Thus, the graph structure depends on how it is evaluated.

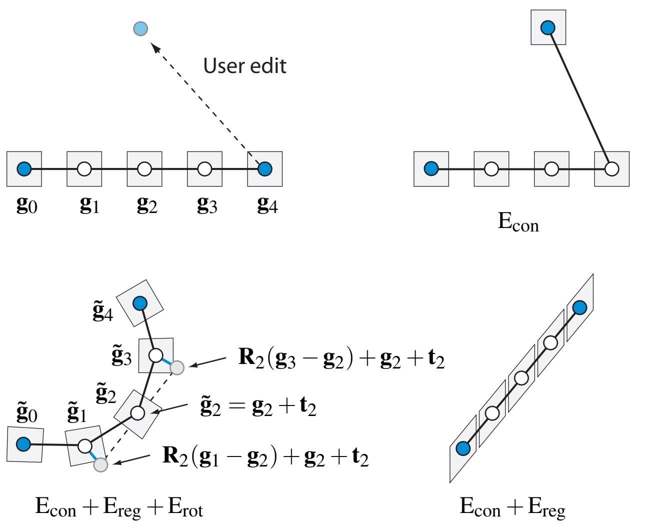
User edit
$\mathbf{g}_0$ $\mathbf{g}_1$ $\mathbf{g}_2$ $\mathbf{g}_3$ $\mathbf{g}_4$
E_{con}
$\tilde{\mathbf{g}}_4$
$\tilde{\mathbf{g}}_3$
$\tilde{\mathbf{g}}_2$
$\tilde{\mathbf{g}}_0$ $\tilde{\mathbf{g}}_1$
$\mathbf{R}_2(\mathbf{g}_3 - \mathbf{g}_2) + \mathbf{g}_2 + \mathbf{t}_2$
$\tilde{\mathbf{g}}_2 = \mathbf{g}_2 + \mathbf{t}_2$
$\mathbf{R}_2(\mathbf{g}_1 - \mathbf{g}_2) + \mathbf{g}_2 + \mathbf{t}_2$
$E_{\text{con}} + E_{\text{reg}} + E_{\text{rot}}$
$E_{\text{con}} + E_{\text{reg}}$

Figure 2: *A simple deformation graph shows the effect of the three terms of the objective function. The quadrilaterals at each graph node illustrate the deformation induced by the corresponding affine transformation. Without the rotational term, unnatural shearing can occur, as shown in the bottom right. The transformation for node $\mathbf{g}_2$ is applied to neighboring nodes $\mathbf{g}_1$ and $\mathbf{g}_3$, yielding the predicted positions shown on the bottom left as gray circles. The regularization term minimizes the squared distance between these predicted positions and their actual positions $\tilde{\mathbf{g}}_1$ and $\tilde{\mathbf{g}}_3$.*

4 Optimization

Once the deformation graph has been specified, the user manipulates an embedded primitive by selecting vertices and moving them around. The vertices serve as positional constraints for an optimization problem in which the affine transformations of the deformation graph comprise the unknown variables. The objective function encodes detail preservation directly by specifying that the affine transformations should be rotations. Consequently, local features deform as rigidly as possible. A second energy term serves as a regularizer for the deformation by indicating that the affine transformations of adjacent graph nodes should agree with one another.

Rotation. In order for a 3×3 matrix $\mathbf{R}$ to represent a rotation in SO(3), it must satisfy six conditions: each of its three columns must be unit length, and all columns must be orthogonal to one another [Grassia 1998]. The squared deviation from these conditions is given by the function Rot($\mathbf{R}$):

$$\begin{aligned} \mathrm{Rot}(\mathbf{R}) = & (\mathbf{c}_1 \cdot \mathbf{c}_2)^2 + (\mathbf{c}_1 \cdot \mathbf{c}_3)^2 + (\mathbf{c}_2 \cdot \mathbf{c}_3)^2 + \\ & (\mathbf{c}_1 \cdot \mathbf{c}_1 - 1)^2 + (\mathbf{c}_2 \cdot \mathbf{c}_2 - 1)^2 + (\mathbf{c}_3 \cdot \mathbf{c}_3 - 1)^2 \end{aligned} \qquad (5)$$

where $\mathbf{c}_1$, $\mathbf{c}_2$, and $\mathbf{c}_3$ are the 3×1 column vectors of $\mathbf{R}$. This function is nonlinear in the matrix entries. The term E_{rot} sums the rotation error over all transformations of the deformation graph:

$$E_{\text{rot}} = \sum_{j=1}^{m} \mathrm{Rot}(\mathbf{R}_j). \qquad (6)$$

Regularization. Conceptually, each of the affine transformations represents a localized deformation centered at a graph node. Since nearby transformations have overlapping influence, we must ensure that the computed transformations are consistent with respect to one another. We add a regularization term to the optimization inferred from the graph structure. If nodes j and k are neighbors, they affect a common subset of the embedded shape. The position of node k

predicted by node j's affine transformation should match the actual position given by applying node k's transformation to itself (Figure 2). The regularization error $\mathrm{E_{reg}}$ sums the squared distances between each node's transformation applied to its neighbors and the actual transformed neighbor positions:

$$\mathrm{E_{reg}} = \sum_{j=1}^{m} \sum_{k \in \mathcal{N}(j)} \alpha_{jk} \left\| \mathbf{R}_j(\mathbf{g}_k - \mathbf{g}_j) + \mathbf{g}_j + \mathbf{t}_j - (\mathbf{g}_k + \mathbf{t}_k) \right\|_2^2. \quad (7)$$

The weight α_{jk} should be proportional to the degree to which the influence of nodes j and k overlap. However, the exact amount of overlap is ill defined for many shape representations, such as point-based models and animated particle systems. In order to meet our goal of generality, we use $\alpha_{jk} = 1.0$ for all examples. We notice no artifacts compared to experiments using other weighting schemes.

This regularization equation bears some resemblance to the deformation smoothness energy term used by previous work on template deformation [Allen et al. 2003; Sumner and Popović 2004; Pauly et al. 2005]. However, the transformed vertex positions are compared, rather than the transformations themselves, and the transformations are relative to the node positions, rather than to the global coordinate system.

Constraints. The user controls the optimization through direct manipulation of the embedded shape and need not be aware of the underlying deformation graph. To facilitate editing, our algorithm supports two types of constraints: handle constraints, where a collection of model vertices are selected and become handles that are manipulated by the user, and fixed constraints, where a collection of model vertices are selected and guaranteed to be fixed in place.

Handle constraints comprise the interface with which the user interacts with an embedded object. These positional constraints are specified by selecting and moving model vertices. They influence the optimization since the deformed vertex positions are a function of the graph's affine transformations. We enforce these constraints using a penalty formulation according to the term $\mathrm{E_{con}}$ which is included in the objective function:

$$\mathrm{E_{con}} = \sum_{l=1}^{p} \left\| \tilde{\mathbf{v}}_{\mathrm{index}(l)} - \mathbf{q}_l \right\|_2^2. \quad (8)$$

Vertex $\tilde{\mathbf{v}}_{\mathrm{index}(l)}$ is deformed by the deformation graph according to Eq. 2. The vector $\mathbf{q}_l$ is the user-specified position of constraint l, and $\mathrm{index}(l)$ is the index of the constrained vertex.

Fixed constraints are specified through the same selection mechanism as handle constraints. However, they are implemented by treating all node transformations that influence the selected vertices as constants, rather than free variables, and removing them from the optimization procedure. Their primary function is to allow the user to define the portion of the mesh which is to be edited. Fixed constraints incur no computational overhead. Conversely, they speed up the computation by reducing the number of unknowns. Thus, the user can make a fine-scale edit by using a dense deformation graph and marking all parts of the embedded object not in the edit region as fixed.

Numerics. Our shape editing framework solves the following optimization problem:

$$\begin{aligned} &\min_{\mathbf{R}_1,\mathbf{t}_1 \ldots \mathbf{R}_m,\mathbf{t}_m} \quad w_{\mathrm{rot}}\mathrm{E_{rot}} + w_{\mathrm{reg}}\mathrm{E_{reg}} + w_{\mathrm{con}}\mathrm{E_{con}}. \\ &\text{subject to } \mathbf{R}_q = \mathbf{I}, \mathbf{t}_q = \mathbf{0}, \forall q \in \text{fixed ids} \end{aligned} \quad (9)$$

We use the weights $w_{\mathrm{rot}} = 1$, $w_{\mathrm{reg}} = 10$, and $w_{\mathrm{con}} = 100$ for all examples. Eq. 9 is nonlinear in terms of the $12m$ unknowns that define the affine transformations. Fixed constraints are handled trivially by treating the constrained variables as constants, leaving $12m - 12q$ free variables if there are q fixed transformations. We implement the iterative Gauss-Newton algorithm to solve the resulting unconstrained nonlinear least-squares problem [Madsen et al. 2004].

The Gauss-Newton algorithm linearizes the nonlinear problem with Taylor expansion about $\mathbf{x}$:

$$\mathbf{f}(\mathbf{x} + \delta) = \mathbf{f}(\mathbf{x}) + \mathbf{J}\delta \quad (10)$$

The vector $\mathbf{f}(\mathbf{x})$ stacks the equations that define the objective function so that $\mathbf{f}(\mathbf{x})^\top \mathbf{f}(\mathbf{x}) = F(\mathbf{x}) = w_{\mathrm{rot}}\mathrm{E_{rot}} + w_{\mathrm{reg}}\mathrm{E_{reg}} + w_{\mathrm{con}}\mathrm{E_{con}}$, the vector $\mathbf{x}$ stacks the entries in the affine transformations, and $\mathbf{J}$ is the Jacobian matrix of $\mathbf{f}(\mathbf{x})$. Each Gauss-Newton iteration solves a linearized problem to improve $\mathbf{x}_k$, the current estimate of the unknown transformations:

$$\begin{aligned} \delta_k &= \operatorname*{argmin}_{\delta} \|\mathbf{f}(\mathbf{x}_k) + \mathbf{J}\delta\|_2^2 \\ \mathbf{x}_{k+1} &= \mathbf{x}_k + \delta_k. \end{aligned} \quad (11)$$

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

$$\begin{aligned} |F_k - F_{k-1}| &< \varepsilon(1 + F_k) \\ \|\nabla F_k\|_\infty &< \sqrt[3]{\varepsilon}(1 + F_k) \\ \|\delta_k\|_\infty &< \sqrt[2]{\varepsilon}(1 + \|\delta_k\|_\infty). \end{aligned} \quad (12)$$

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

In each iteration, we solve the resulting normal equations by Cholesky factorization. Although the linear system $\mathbf{J}^\top\mathbf{J}$ is very sparse, it depends on $\mathbf{x}$ and thus changes at each iteration. Therefore, the full factorization cannot be reused. However, the non-zero structure remains unchanged so that a fill-reducing permutation of the matrix and symbolic factorization based only on its non-zero structure can be precomputed and reused [Toledo 2003]. These steps, together with careful implementation of the routines to build $\mathbf{J}$ and $\mathbf{J}^\top\mathbf{J}$, result in a very efficient solver. As shown in Table 1, each iteration requires about 20ms for the presented examples.

5 Results

We have implemented the deformation graph optimization both as an interactive editing application as well as an offline system for applying scripted constraints to animated data. Live edits with the interactive application are demonstrated in the conference video, and key results are highlighted in this section.

Detail preservation. Figure 3 demonstrates that our algorithm preserves features of the embedded shape. A bumpy plane is modified by fixing vertices on the left in place and translating those on the right upward. Although this edit is purely translational, the optimization finds node transformations that are as close as possible to true rotations while meeting the vertex constraints and maintaining consistency. As a result, the bumps on the plane deform in a natural fashion without shearing artifacts. These results are comparable to the nonlinear prism-based approach of Botsch and colleagues [2006a]. However, our algorithm uses a deformation graph of only 299 nodes, whereas Botsch's method performs the optimization on the full 40,401 vertex model and requires a consistent meshing of the surface. Figure 3 also demonstrates that our method preserves details better than the radial-basis function (RBF) approach of Botsch and Kobbelt [2005], where feature rotation is not considered.

Figure 4 demonstrates detail preservation on a more complex example. With a graph of only 222 nodes, our approach achieves a deformation comparable in quality to the subspace method of Huang and

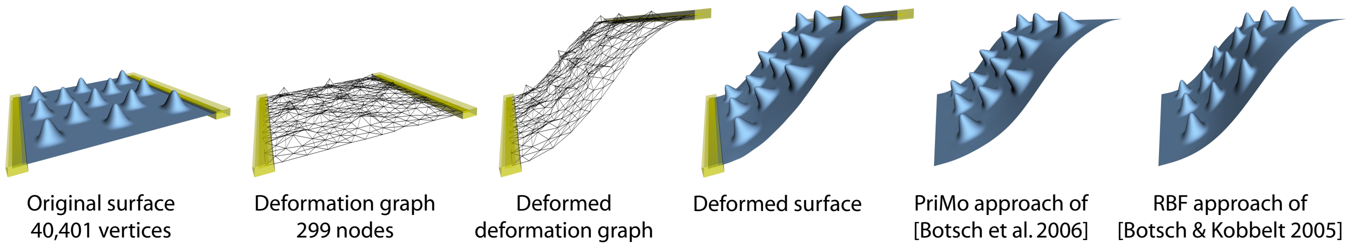

Figure 3: *When used to deform a bumpy plane, our method accurately preserves features without shearing or stretching artifacts. The quality of our results is comparable to the "PriMo" approach of Botsch and colleagues [2006a] and superior to the radial-basis function method of Botsch and Kobbelt [2005].*

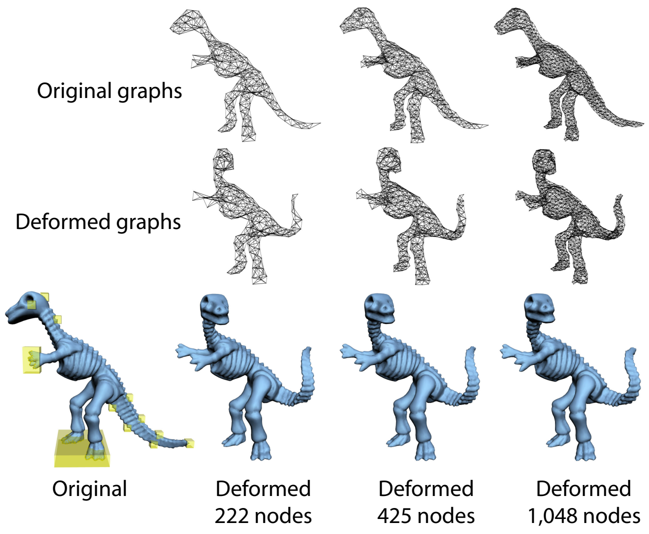

Figure 4: *We perform an edit similar to the one shown in Figure 9 of the work of Huang and colleagues [2006]. With a graph of only 222 nodes, our results are of comparable quality to Huang's subspace gradient domain method. Performing the identical edit with more complex graphs does not yield a significant change in quality.*

colleagues [2006] in which the Laplacian energy is enforced on the full mesh. Higher resolution graphs do not significantly improve quality. Performing the same editing task with graphs of 425 and 1,048 nodes yields nearly identical results. Of course, if the graph becomes too sparse to match the complexity of the deformation, artifacts will occur, as can be expected with any reduced deformable model. Likewise, in the highly regular setting shown in Figure 5, minor artifacts appear as a slight striped pattern. If additional nodes are used for interpolation or a less regular graph (Figure 3), no artifacts are noticeable.

Intuitive editing. Figures 6 and 7 demonstrate the intuitive editing framework enabled by our system. High-quality edits are achieved by placing only a handful of single-vertex handle constraints on the shape. Figure 6 shows detail-preserving edits on a mesh consisting of 85,792 vertices. The raptor is deformed by displacing positional constraints only, without the need to explicitly specify frame rotations. Fine-scale details such as the teeth and wrinkles are preserved. Furthermore, when the head or body is manipulated and the arms are left unconstrained, the arms rotate in a natural way to follow the body movement. Thus, features are preserved at a wide range of scales. In this example, a full body pose is sculpted using a graph of 226 nodes. The tail is lifted, arms crossed, left leg moved forward, and head rotated to look backward. Then, localized changes to the head are made with a more detailed graph of 840 nodes. However, fixed constraints specified by selecting the

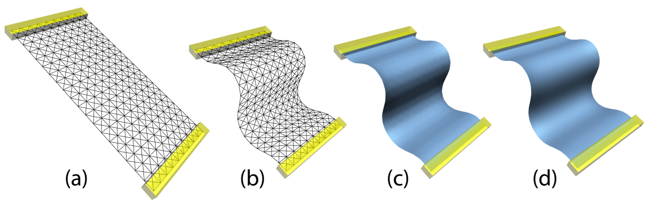

Figure 5: *A highly regular deformation graph with 200 nodes, shown in (a), is used to create the deformation in (b). In this structured setting, minor artifacts are visible on the 13,024 vertex plane, shown in (c), as a slight striped pattern when k=4 graph nodes are used for transforming the mesh vertices. These artifacts disappear in (d) when k=8 nodes are used and are not present with less structured graphs (Figure 3).*

raptor's body (green) leave only 138 active nodes for the head edit so that the system remains interactive.

Figure 7 shows interactive edits on a scanned toy giraffe. The model consists of a set of un-merged range scans that contain many holes and outliers, with a total of 79,226 vertices in 180 separate connected components. The deformation graph consisting of 221 nodes is built automatically via uniform sampling, allowing the user to directly edit the shape without time-consuming pre-processing to obtain a consistent mesh representation.

Mesh animations. In addition to static geometry, our approach also supports effective editing of dynamic shapes. The mesh animation of Figure 8 is modified to lengthen the horse's legs and neck, and turn its head. The deformation graph, constructed with mesh simplification, is advected passively with the animated mesh. Since the graph nodes are chosen to coincide with mesh vertices, no additional computation is required for the node positions to track the animation. The user can script edits by setting keyframes on a single pose. Translational offsets are computed from this keyframe data and applied frame-by-frame to the animation sequence with our offline application. The graph structure and weighting remains fixed throughout the animation. The output mesh animation incorporates the user's edits while preserving geometric details, such as the horse's facial features, as well as high-frequency motion, such as the head bobbing. No multiresolution hierarchy or signal processing is needed, unlike the method of Kircher and Garland [2006]. Although we do not address temporal coherence directly, we noticed no coherence artifacts in our experiments.

Particle simulations. The particle simulation shown in Figure 9 is another example of a dynamic shape that can be edited with the deformation graph framework. Our system allows small-scale corrections that would be tedious to achieve by tweaking simulation parameters, as well as more drastic modifications that go beyond the capabilities of a pure simulation. In this example, particle positions are precomputed with a fluid simulation. A linear deformation graph is built by sampling the path that a single particle travels over

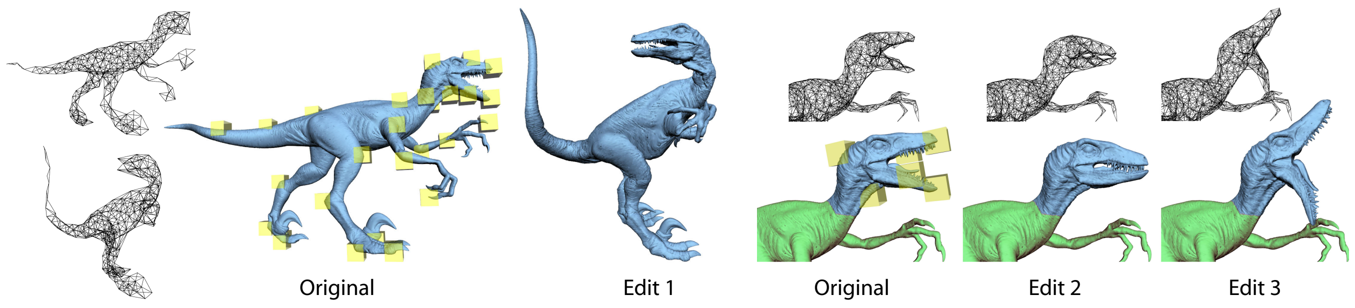

Figure 6: *The user sculpts full body pose changes as well as detailed modifications to the raptor's head. Yellow boxes indicate single-vertex handle constraints, while the green region determines the fixed constraints.*

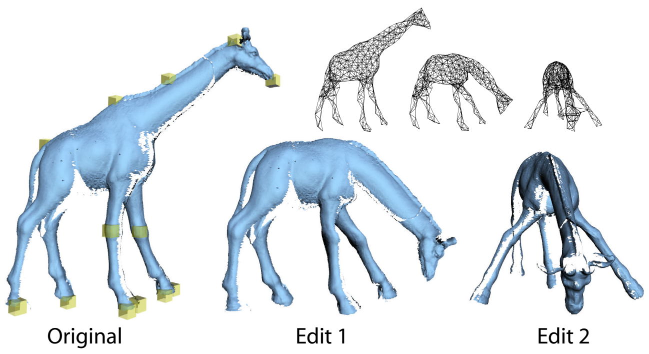

Figure 7: *Interactive edits on un-merged range scans.*

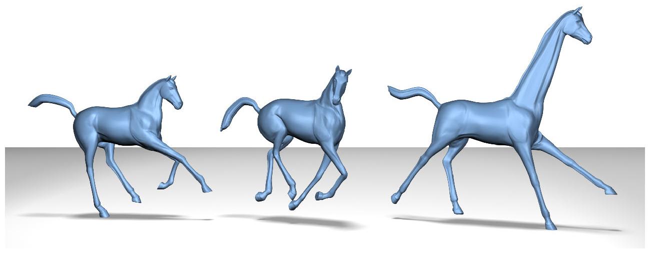

Figure 8: *Three frames from an edited mesh animation.*

time. Nodes are created at fixed time intervals, and graph edges connect each sample to the previous and subsequent ones. The influence of the graph on each particle is not fixed but varies with time as the particle moves through space. As demonstrated in the video, the user can interactively sculpt the shape of the particle stream. The edited version retains the dynamic appearance of the simulation, while conforming to the desired edits.

Efficiency. As Table 1 indicates, our implementation achieves interactive performance for all examples shown. The time required for each Gauss-Newton iteration, indicated by the "Solve" column, is usually around 20 to 30ms. For the giraffe model, we show timing statistics for as many as 5,000 handle constraints, listed in the "Csts" column. The solver still maintains interactive performance with only 38ms required for each Gauss-Newton iteration. For the dino model, the timing increases from 19ms for a graph of 222 nodes to 148ms for a graph of 1,048 nodes. The graph deformation is transferred to the embedded shape using a software implementation of Eq. 2, with timing shown in the "Def" column. A GPU implementation may further improve performance. Our application deforms the mesh, displays the result, and polls the user-interface at fixed intervals of 100ms, or sooner if the solver converges.

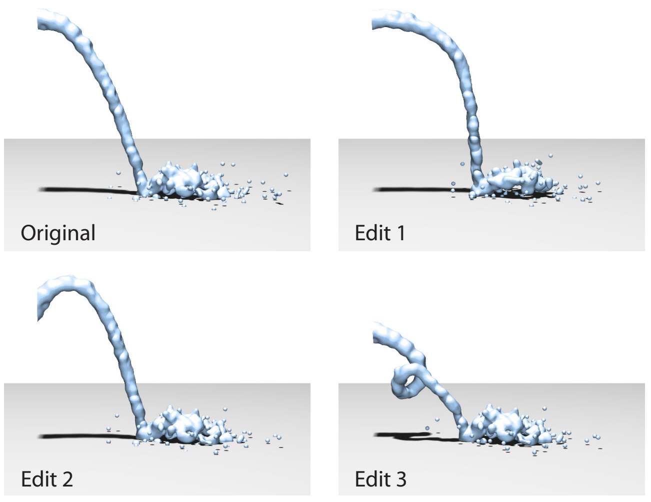

Figure 9: *Direct manipulation of simulated particles.*

6 Conclusion

We have presented a shape manipulation algorithm for embedded deformation that meets the goals of detail-preserving mesh editing in the framework of space deformations. Our method is notable due to its simplicity, efficiency, and versatility. For traditional mesh editing, our algorithm performs comparably to existing nonlinear methods. However, it applies as well to meshes with many connected components, polygon soups, point-based representations, and mesh animations. The deformation graph is easy to construct and corresponds closely to the embedded shape. It facilitates mesh animations since the graph can be passively moved with the mesh. The reduced model ensures that the complexity of the deformation algorithm is not tied to the geometric complexity of the embedded object, while the optimization procedure strives to preserve shape features in a natural fashion.

Current limitations of our method direct us to areas of future work. The structure of the deformation graph is static during an editing operation and determines the achievable deformation complexity. Fine-scale edits cannot be represented by a coarse-scale graph. However, in our experience, this restriction is not severe, since graph creation is light-weight. A coarse graph can easily be discarded and replaced with a more detailed one. Nevertheless, a more elegant solution would be to dynamically update the graph structure based on the user's edits. The individual terms of the objective function tell us when and where errors occur and indicate when the graph is poorly suited for the desired deformation. These error terms provide a starting point for a dynamic update strategy.

Model	Vertices	Nodes	Csts	Solve	Def
Raptor	85,792	226	27	20 ms	22 ms
Giraffe	79,226	221	13	17 ms	20 ms
Horse	8,431	303	10	23 ms	2 ms
Fluid	~1,070	107	8	4 ms	0.5 ms
Giraffe	79,226	221	670	20 ms	20 ms
Giraffe	79,226	221	1,900	25 ms	20 ms
Giraffe	79,226	221	5,000	38 ms	20 ms
Dino	10,002	222	540	19 ms	3 ms
Dino	10,002	425	540	41 ms	3 ms
Dino	10,002	1,048	540	148 ms	3 ms

Table 1: *Statistics and performance data for the interactive edits. Timing is measured in milliseconds on a 2.4GHz Intel Core 2 Duo machine with 2GB of RAM. Our solver is single-threaded and uses only one core. The number of vertices for the fluid model represents the average per frame.*

The spatial nature of our method may result in part of the graph influencing an undesired area of the shape, such as a hand node influencing the leg if a character's arms are by its side. With mesh representations, this problem is solved by considering geodesic distances along the surface. However, in our general setting, geodesics are ill defined. Since these problems are rare, we believe the most appropriate solution is to rely on the user to fine-tune the spatial associations with a simple user interface to "cut" undesired connections. For example, the user would draw a stroke between the leg and the hand when the undesired influence becomes noticeable.

In our prototype implementation, we have incorporated only a handful of different shape representations. Future work will explore additional ones, such as NURBS, subdivision, and implicit surfaces, 2D images, and 3D volumetric data such as MRI scans, the visible human data set, or simulated pressure fields. For NURBS and subdivision surfaces, a straightforward extension to the positional constraints would allow the user to directly manipulate points on the surface itself, rather than those that define the control hull. The extension to implicit surfaces and sampled grid data is less clear. However, one could imagine building a deformation graph by sampling a user-selected level set and resampling the data after applying the solved-for deformation. Pursuing these goals will extend intuitive editing to additional situations where it is lacking.

Acknowledgments

We are grateful to Niloy Mitra for his assistance with the point sampling routine, Hao Li for editing the video, Richard Keiser for help with the fluid simulation, and Simone Riva for work on the Gauss-Newton implementation.

References

ALLEN, B., CURLESS, B., AND POPOVIĆ, Z. 2003. The space of human body shapes: Reconstruction and parameterization from range scans. *ACM Trans. Graph. 22*, 3.

BARR, A. H. 1984. Global and local deformations of solid primitives. In *Proc. of SIGGRAPH*.

BOTSCH, M., AND KOBBELT, L. 2004. An intuitive framework for real-time freeform modeling. *ACM Trans. Graph. 23*, 3.

BOTSCH, M., AND KOBBELT, L. 2005. Real-time shape editing using radial basis functions. *Computer Graphics Forum 24*, 3.

BOTSCH, M., AND SORKINE, O. 2007. On linear variational surface deformation methods. *To appear in IEEE Transaction on Visualization on Computer Graphics*.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: Coupled prisms for intuitive surface modeling. In *Fourth Eurographics Symposium on Geometry Processing*.

BOTSCH, M., SUMNER, R. W., PAULY, M., AND GROSS, M. 2006. Deformation transfer for detail-preserving surface editing. In *Vision, Modeling & Visualization 2006*.

COQUILLART, S. 1990. Extended free-form deformation: A sculpturing tool for 3D geometric modeling. In *Proc. of SIGGRAPH*.

GILL, P. E., MURRAY, W., AND WRIGHT, M. H. 1989. *Practical Optimization*. Academic Press, London.

GRASSIA, F. S. 1998. Practical parameterization of rotations using the exponential map. *J. Graph. Tools 3*, 3.

HSU, W. M., HUGHES, J. F., AND KAUFMAN, H. 1992. Direct manipulation of free-form deformations. In *Proc. of SIGGRAPH*.

HUANG, J., SHI, X., LIU, X., ZHOU, K., WEI, L.-Y., TENG, S.-H., BAO, H., GUO, B., AND SHUM, H.-Y. 2006. Subspace gradient domain mesh deformation. *ACM Trans. Graph. 25*, 3.

KIRCHER, S., AND GARLAND, M. 2006. Editing arbitrarily deforming surface animations. *ACM Trans. Graph. 25*, 3.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *Proc. of SIGGRAPH*.

MACCRACKEN, R., AND JOY, K. I. 1996. Free-form deformations with lattices of arbitrary topology. In *Proc. of SIGGRAPH*.

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.

MOHR, A., AND GLEICHER, M. 2003. Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22*, 3.

PAULY, M., MITRA, N. J., GIESEN, J., GROSS, M., AND GUIBAS, L. J. 2005. Example-based 3d scan completion. In *Third Eurographics Symposium on Geometry Processing*.

SEDERBERG, T. W., AND PARRY, S. R. 1988. Free-form deformation of solid geometric models. In *Proc. of SIGGRAPH*.

SHEFFER, A., AND KRAEVOY, V. 2004. Pyramid coordinates for morphing and deformation. In *Proc. of the 2nd Symposium on 3D Processing, Visualization and Transmission*.

SHI, L., YU, Y., BELL, N., AND FENG, W.-W. 2006. A fast multigrid algorithm for mesh deformation. *ACM Trans. Graph. 25*, 3.

SINGH, K., AND FIUME, E. L. 1998. Wires: A geometric deformation technique. In *Proc. of SIGGRAPH*.

SINGH, K., AND KOKKEVIS, E. 2000. Skinning characters using surface oriented free-form deformations. In *Graphics Interface*.

SORKINE, O. 2005. Laplacian mesh processing. In *State of the Art Reports*, Eurographics.

SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3.

TOLEDO, S., 2003. TAUCS: A library of sparse linear solvers, version 2.2. http://www.tau.ac.il/∼stoledo/taucs.

WEBER, J. 2000. Run-time skin deformation. In *Proceedings of the 2000 Game Developers Conference*.

ZHOU, K., HUANG, J., SNYDER, J., LIU, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2005. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph. 24*, 3.