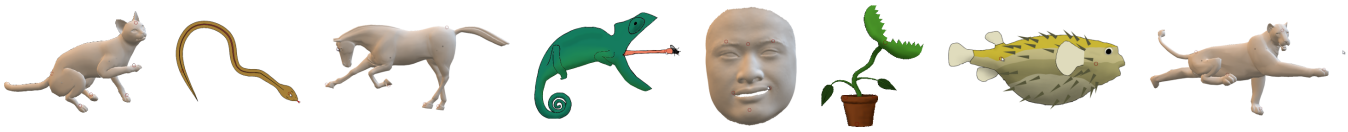
Fast and Reliable Example-Based Mesh IK for Stylized Deformations

Kevin Wampler*
Adobe Systems Inc.



Abstract

Example-based shape deformation allows a mesh to be easily manipulated or animated with simple inputs. As the user pulls parts of the shape, the rest of the mesh automatically changes in an intuitive way by drawing from a set of exemplars. This provides a way for virtual shapes or characters to be easily authored and manipulated, or for a set of drawings to be animated with simple inputs. We describe a new approach for example-based inverse kinematic mesh manipulation which generates high quality deformations for a wide range of inputs, and in particular works well even when provided stylized or "cartoony" examples. This approach is fast enough to run in real time, reliably uses the artist's input shapes in an intuitive way even for highly nonphysical deformations, and provides added expressiveness by allowing the input shapes to be utilized in a way which spatially varies smoothly across the resulting deformed mesh. This allows for rich and detailed deformations to be created from a small set of input shapes, and gives an easy way for a set of sketches to be brought alive with simple click-and-drag inputs.

Keywords: deformation, shape modeling, as-rigid-as-possible, shape space

Concepts: •**Computing methodologies** → *Animation; Mesh models;*

1 Introduction

Even a relatively simple shape in computer graphics is typically represented by a polygonal mesh which is too complex to manually modify vertex by vertex, and certainly not in a real-time application. Higher level tools are instead necessary to control the shape by manipulating a small set of parameters. Shape deformation provides one convenient way of doing this. The user controls the position of a few vertices or localized areas of the mesh, and the proper distortion for rest of the shape is automatically inferred.

The user's input while interacting with a deformable mesh can be modeled as constraints which must be satisfied by the resulting shape – for instance specifying the position to which a particular vertex must be moved. Traditionally, the deformed shape is then computed as one which bends or stretches the input shape as smoothly as possible while still satisfying the specified constraints. Unfortunately both real-world and artist-designed shapes often do not deform in this manner, instead deforming in a way which is more complex: sections will twist, thin, or balloon, muscles will bulge, specific parts will curve or bend, etc.

We consider the creation and manipulation of models capturing these sorts of complex deformations. We focus in particular on the interactive manipulation or animation of virtual shapes or characters, including highly stylized artist-created examples. To ensure that our deformable models are easy to create, even for non-technical users, we base our approach within the paradigm of example-based deformations where multiple shapes are provided as input and the range of features observed in these shapes is used to inform the resulting deformation. This has the advantage that it can be applied directly to free form meshes without any need for complicated character rigs or manually crafted parameterizations.

After a set of input shapes is provided, the natural question that arises is precisely how they should be combined to create a deformed shape based on a user's inputs. In the context of interactive shape manipulation or animation, this technique should be both *fast* enough to run in real time, and *reliable* in that the result reflects the design of the inputs shapes in an intuitive and easily predictable manner, even for highly stylized inputs. Finally, because of the effort required in an artist creating a large number of input shapes, the approach should also be *expressive* in that a wide range of visually plausible deformations can be created from a relatively small set of inputs.

With this in mind, we propose a novel approach to example-based shape deformation. This approach generalizes the traditional as-rigid-as-possible deformation energy to allow for example-based deformations, and is fast enough to run in real time. We achieve reliable deformations by viewing the issue of reliability as a combination of a scattered data interpolation and an elastic deformation, and deriving a new form of an example-based elastic energy combining properties of both. We also support the easy authoring of expressive models by supporting deformations that employ the provided input shapes in a manner which is spatially localized and can smoothly vary over the mesh.

Our primary contribution is the development of a technique for real-time kinematic example-based mesh deformation which achieves high-quality and intuitive results across a wide range of cases with no parameter tuning, including highly non-physical stylized inputs. This involves three sub-contributions working in concert:

1. We develop a new approach for generalizing a non-example-based elastic energy into an example-based elastic energy which improves how reliably the inputs shapes are used, by considering not only the space spanned by the input shapes, but also the desirability of different shapes *within* this space. This combines the strengths of existing shape space ap-

*e-mail:kwampler@adobe.com

SA '16 Technical Papers,, December 05-08, 2016, , Macao
ISBN: 978-1-4503-4514-9/16/12
DOI: http://dx.doi.org/10.1145/2980179.2982433

ACM Reference Format
Wampler, K. 2016. Fast and Reliable Example-Based Mesh IK for Stylized Deformations. ACM Trans. Graph. 35, 6, Article 235 (November 2016), 12 pages. DOI = 10.1145/2980179.2982433 http://doi.acm.org/10.1145/2980179.2982433.

proaches with some beneficial properties typical of scattered data interpolation.

2. We give an example-based generalization of an as-rigid-as-possible energy as applied to a mesh parameterized with linear blend skinning, as well as an associated optimization algorithm. This allows high-quality deformations to be calculated in real time even for high-resolution meshes.

3. We describe an example-based elastic energy which allows the degree to which each input shape is used to smoothly vary over the resulting mesh while retaining real-time performance, building upon previous approaches for plastic deformations [Jones et al. 2016].

2 Related Work

The ability to deform a potentially complex input shape by providing a small set of user-specified constraints is an important and well-studied problem in computer graphics, a survey of which is provided by Botsch and Sorkine [2008]. Typically the input shape is represented as a triangulated mesh or tetrahedralized volume so that calculating the deformation amounts to computing new positions for the vertices in the shape. The most popular way of approaching this problem for free form meshes is to minimize an elastic energy on the shape subject to some set of user-specified constraints. This energy measures the accumulation of local distortions of the deformed shape with respect to the input shape. Of particular relevance to our work is as-rigid-as-possible (ARAP) shape deformation [Igarashi et al. 2005; Sorkine and Alexa 2007; Liu et al. 2008; Chao et al. 2010; Jacobson et al. 2012a; Levi and Gotsman 2015] which measures local distortions by the degree of non-rigidity they induce. ARAP deformations are relatively simple to formulate, efficient to compute, and normally give rise to visually pleasing results.

Most shape deformation approaches are formulated with respect to a single input shape, which does not allow detailed control over how the deformation should deviate from this base shape. To remedy this, our method draws inspiration from an existing class of techniques which rely on *multiple* input shapes. One facet of this is the problem of shape interpolation, in which a deformed shape is generated by blending the input shapes with different pre-specified weights. In some contexts such as the use of blendshapes in face animation [Bergeron and Lachapelle 1985] this interpolation can be performed directly in the space of vertex positions, but in most cases this leads to obvious artifacts, and variety of alternatives have been proposed. These include using alternative parameterizations such as pyramid coordinates [Sheffer and Kraevoy 2004], interpolating along geodesics in a Riemannian space [Kilian et al. 2007], guiding the interpolation by clustering poses in a database [Gao et al. 2013], or by explicitly considering the rigid and non-rigid components of the interpolation in an as-rigid-as-possible approach [Alexa et al. 2000; Chao et al. 2010; Levi and Gotsman 2015].

Shape interpolation is sufficient if it is reasonable to explicitly specify the interpolation weights, but in many situations it is desirable to infer these weights automatically. Depending on the context, this can be done either with a kinematic technique such as ours, or with a dynamic technique when one wishes to calculate the deformations in the context of a physical simulation. The most common formulation within this latter category augments the physical dynamical equations with a constraint that the simulated shape lie near the subspace spanned by the input shapes [Martin et al. 2011; Koyama et al. 2012; Schumacher et al. 2012; Song et al. 2014; Zhang et al. 2015]. Jones et al. [2013] take a similar approach and constrain the simulation to lie within a user-specified simplicial complex on the input shapes. A related method is described by Jones et al. [2016] in the context of plastic, rather than elastic deformations.

Our approach is kinematic rather than dynamic in that it does not rely on any physical simulation and instead calculates the deformed shape purely from the input shapes and user-specified constraints. This is done in the spirit of inverse kinematics where the user provides positional constraints on localized portions of the mesh. Inverse kinematic approaches are popular with kinematic skeletons, a context in which Grochow et al. [2004] use a set of example skeletal poses to guide the result. An analogous operation in the context of free form mesh deformation was introduced by Sumner et al. [2005] and later improved by Der et al. [2006] and Fröhlich and Botsch [2011]. Alternatively, Lewis and Anjyo [2010] and Seo et al. [2011] perform a similar operation in the specific context of face deformation. The basic technique used by these approaches is similar to that of example-based dynamics in that the deformation is chosen to lie as close as possible to a linear or convex subspace spanned by the example shapes.

As an alternative to choosing the deformed shape as one nearest to a subspace spanned by the input shapes, it is natural to view calculating the deformation as a scattered data interpolation problem. That is, choose the deformed shape not based on an energy measuring the distance from a subspace, but instead as one which blends between nearby input shapes. Often this is done by parameterizing the deformation within an abstract pose space [Lewis et al. 2000] and using traditional scattered data interpolation techniques such as radial basis functions [Sloan et al. 2001; Zhang et al. 2004] or kernel CCA [Feng et al. 2008] within this space, after which the interpolated shape is calculated, sometimes with explicit detail preservation [Weber et al. 2007; Huang et al. 2011; Hahn et al. 2014]. In many cases, particularly in restricted contexts such as skeletal rigs or face deformation, these methods perform well. With respect to elastic energy-based approaches, however, they have the disadvantage that they require a (sometimes large) set of input shapes and are not designed to calculate deformations when given only one or very few input meshes. One notable approach which combines some aspects of elastic energy with scattered data interpolation is given by Milliez et al. [2013], who discretely choose pieces of a larger model from a set of example shapes by projecting to the nearest neighbor as measured by an ARAP-based distance metric.

Also related to our approach is the topic of mesh skinning [Sederberg and Parry 1986], whereby the deformation of a shape is directly parameterized by a small set of controls, with the influence of these controls propagated to the entire mesh. Using a cage for this parameterization is a popular approach, and a survey is given by Nieto and Susn [2013]. In the case of character animation it may make more sense to bind the mesh to the pose of an underlying kinematic skeleton [Baran and Popović 2007; Wareham and Lasenby 2008]. In our implementation, the user places a set of *handles* at localized positions on the mesh, and the influence of these handles over the mesh is computed with bounded biharmonic weights [Jacobson et al. 2011]. Jacobson et al. [2012b] and Wang et al. [2015] provide alternative methods for handle-based skinning.

3 Overview

The input to our mesh deformation method is a set $\mathcal{S}$ of example shapes. Each of these shapes is represented by a triangular mesh and describes an example rest shape for some object. These meshes must have the same vertex set $\mathcal{P}$ and the same triangle set, so they only differ in the positions of their vertices. The shape of the s^{th} example mesh, denoted by $\boldsymbol{p}_s$, is represented with a one-dimensional array of length $|\mathcal{P}|$, each element of which is a k-dimensional vector describing the position of a single vertex, where k is either two or three depending on the dimension of the space the mesh is embedded in. All of the $\boldsymbol{p}_s$ example meshes are collected together into a single $|\mathcal{S}| \times |\mathcal{P}|$ array denoted $\boldsymbol{P}$. Recall that each element of $\boldsymbol{P}$

is a k-dimensional vector, rather than a single scalar.

We allow a user to manipulate a mesh by controlling a set $\mathcal{H}$ of *handles*. Each handle is associated with a subset of the vertices of the mesh. A deformed mesh $\boldsymbol{q}$ is computed by the user specifying the position of each handle, from which the positions of the remaining vertices are computed automatically. For any particular setting of the desired positions of each handle in $\mathcal{H}$, there is an associated constraint set $\mathcal{C}$ such that $\boldsymbol{q} \in \mathcal{C}$ if and only if each handle in $\boldsymbol{q}$ is located at its desired position.

As is standard with existing example-based mesh deformation approaches, we compute a deformed mesh $\boldsymbol{q}$ by minimizing an elastic energy $E(\boldsymbol{q}, \boldsymbol{P}, \boldsymbol{b})$. Here $\boldsymbol{q}$ is an array of length $|\mathcal{P}|$, each element of which is a k-dimensional vector describing the position of a single vertex in the deformed mesh. $\boldsymbol{P}$ is an array combining the vertex positions of all of the $\boldsymbol{p}_{\text{s}}$ input meshes as previously described, and $\boldsymbol{b}$ is a length-$|\mathcal{S}|$ array of scalars such that $\boldsymbol{b}_{\text{s}}$ gives the degree of contribution for the input mesh $\boldsymbol{p}_{\text{s}}$ to $\boldsymbol{q}$. We will further enforce that $\boldsymbol{b}$ is nonnegative and sums to one. In section 5 we will show how $\boldsymbol{b}$ can be extended to allow spatially localized interpolations, but for now will use this simplified form to make it easier to compare to previous work.

The general form of the optimization used to solve for an example-based mesh deformation as applied to the context of mesh-based inverse kinematics can then be written:

$$\boldsymbol{q}_{opt}, \boldsymbol{b}_{opt} = \underset{\boldsymbol{q} \in \mathcal{C}, \boldsymbol{b}}{\arg\min}\, E(\boldsymbol{q}, \boldsymbol{P}, \boldsymbol{b}) \tag{1}$$

The primary concern of this paper is to derive a definition for E appropriate for fast and reliable mesh-based inverse kinematics. It is tempting to view this problem as either a trivial generalization of mesh interpolation, or as a trivial restriction of example-based elastic material simulation. It is perhaps surprising then that there are difficulties specific to mesh IK that do not appear in either of these two contexts. To better motivate our approach we will begin by examining these alternatives.

To provide a pedagogical example to compare the different approaches, consider a simple square-shaped mesh. As the square is stretched it should first puff outward, and then thin inward as it is stretched yet further. This behavior is specified with the three input meshes shown in figure 1. Each of three input shapes is shown in a different color. The deformation is controlled by specifying the position of three handles, each of which is shown as a black dot. We will consider in particular what happens as the user drags the rightmost handle.

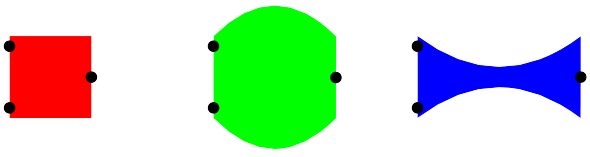


Figure 1

Although this would intuitively appear to be a very simple example, it poses surprising difficulties for existing approaches. Broadly speaking these existing approaches can be divided into two categories depending on how E is specified.

3.1 Motivation: Energy Interpolation

One class of existing technique defines E by interpolating between the values of a non-example-based elastic energy E_d as applied to each $\boldsymbol{p}_{\text{s}}$ shape independently:

$$E(\boldsymbol{q}, \boldsymbol{P}, \boldsymbol{b}) = \sum_{\text{s}}^{|\mathcal{S}|} \boldsymbol{b}_{\text{s}}\, E_d(\boldsymbol{q}, \boldsymbol{p}_{\text{s}}) \tag{2}$$

An energy of this form is typically applied in the context of shape interpolation, where $\boldsymbol{b}$ is assumed to be fixed in advance and only $\boldsymbol{q}$ is optimized over [Alexa et al. 2000; Chao et al. 2010; Levi and Gotsman 2015]. If we instead apply an energy of this form to the problem of mesh IK by optimizing over both $\boldsymbol{q}$ and $\boldsymbol{b}$ as prescribed by equation 1, the results exhibit a rather serious artifact, as shown in figure 2 row c. The form of equation 2 guarantees that E is always minimized when $\boldsymbol{b}$ is zero except for a one at the entry of the input shape corresponding to the minimal value of $E_d(\boldsymbol{q}, \boldsymbol{p}_{\text{s}})$. The resulting deformation thus discretely jumps from one shape to another as the user drags the handles, rather than smoothly deforming as the artist likely intended.

Energies of this sort have a natural interpretation as a scattered data interpolant. Since the optimal value of $\boldsymbol{b}$ is always zero except for the input shape for which E_d is minimized, it corresponds to nearest neighbor interpolation by projection to the nearest Voronoi cell using a measure of distance as defined by $E_d(\boldsymbol{q}, \boldsymbol{p}_{\text{s}})$. Because of this, despite the defect of discontinuous transitions, all of the input shapes are reliably used.

3.2 Motivation: Shape Spaces

The second class of elastic energies used for example-based mesh deformation rely on the concept of a *shape space*, also sometimes called an *example manifold*. As with energy interpolation this also relies on a non-example-based elastic energy E_d, but applies it to an interpolation of the input shapes, or of features computed from the input shapes. Using $\text{blend}(\mathbf{P}, \boldsymbol{b})$ to denote a function which interpolates the $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{|\mathcal{S}|}$ input meshes using $\boldsymbol{b}$ as a vector of interpolation weights, these energies can be expressed as:

$$E(\boldsymbol{q}, \boldsymbol{P}, \boldsymbol{b}) = E_d\left(\boldsymbol{q}, \text{blend}(\mathbf{P}, \boldsymbol{b})\right) \tag{3}$$

This class of approaches is sometimes used for mesh IK [Sumner et al. 2005; Der et al. 2006; Fröhlich and Botsch 2011], but has seen particular success when applied to example-based elastic material simulation [Martin et al. 2011; Koyama et al. 2012; Schumacher et al. 2012; Song et al. 2014; Zhang et al. 2015], where it performs quite admirably. Since the goal for simulation is to have the mesh deform in a physically plausible manner guided by the example shapes, the passive dynamics of the mesh act as a powerful regularizer for the deformation. Shape spaces are a natural fit here, since within the shape space, the passive dynamics dominate. For inverse kinematics, however, this approach also has artifacts. The nature of the artifacts depends on whether $\text{blend}(\mathbf{P}, \boldsymbol{b})$ is a linear or nonlinear function.

If $\text{blend}(\mathbf{P}, \boldsymbol{b})$ is linear, as in the linear deformation gradients of Sumner et al. [2005], then deformation of the example in figure 1 is often ambiguous, since many linear combinations of the input shapes may all satisfy the user's IK constraints. This is shown in figure 2 row d, where the green shape is essentially ignored and a 50/50 blend of the red and blue shapes is used instead.

It is more common that $\text{blend}(\mathbf{P}, \boldsymbol{b})$ is nonlinear and that it computes a feature vector larger than the number of degrees of freedom in $\boldsymbol{q}$. As described by Schumacher et al. [2012], this means that E will typically be zero only when $\boldsymbol{q}$ is exactly equal to some $\boldsymbol{p}_{\text{s}}$. This avoids the ambiguity inherent in linear shape spaces, but adds the complication that a nonlinear shape space may give rise to optimization difficulties such as local minima. Examples of artifacts

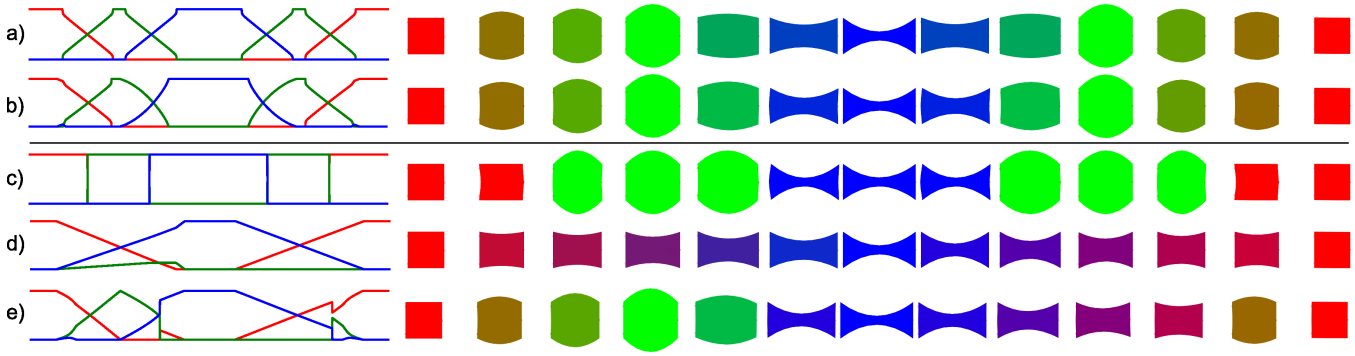


Figure 2: *Each row shows a different deformation method applied to the inputs in figure 1 as the rightmost handle is dragged to stretch and then un-stretch the shape. The graphs on the left show the blending weights over time. From top to bottom: a) Our method without rotations, b) Our method with rotations, c) energy interpolation [Chao et al. 2010], d) linear shape space (our method with $\lambda = 0$), e) nonlinear shape space [Schumacher et al. 2012].*

resulting from this are shown in figure 2 row e, which exhibits both discontinuous jumps as a new local minimum arises or disappears and the optimization converging to a suboptimal local minimum when the box is contracted, using a blend of the red and blue shapes when the green shape would lead to a lower error.

4 Interpolation Energy

At the core of our method is an elastic energy which, when minimized subject to some inverse kinematic constraints, interpolates efficiently and reliably between a set of example meshes, even for highly stylized and exaggerated deformations. Although our elastic energy is based on an as-rigid-as-possible (ARAP) energy, we will consider it first in a simplified form. This will make it easier to compare against the alternative energies described in section 3 without the additional complications introduced by the full ARAP-based formulation.

As with previous approaches such as energy interpolation (section 3.1) and shape space energies (section 3.2), we construct an example-based elastic energy $E(\boldsymbol{q}, \mathbf{P}, \boldsymbol{b})$ by modifying a non-example-based energy $E_d(\boldsymbol{q}, \boldsymbol{p})$. Here, $\boldsymbol{q}$, $\mathbf{P}$, $\boldsymbol{p}$, and $\boldsymbol{b}$ take the same meanings as previously introduced in section 3. At runtime, the vertex positions in the deformed mesh $\boldsymbol{q}$ are generated by minimizing E subject to some user-specified constraints enforcing that $\boldsymbol{q} \in \mathcal{C}$ where $\mathcal{C}$ represents the subset of deformations in which the handles are correctly located at their user-specified positions.

A primary function of the elastic energy E is used to interpolate between the artist-provided input shapes as the user-specified constraint set $\mathcal{C}$ changes (i.e. as the user clicks and drags the handles). Given this, it is useful to consider what properties E should satisfy not only as an elastic energy, but also what properties its minimization has as a method for scattered data interpolation.

Firstly, E should satisfy some basic properties which define it as an example-based energy. Both energy interpolation and shape space energies already satisfy these:

1. If $\boldsymbol{b}_\mathrm{s} = 0$, then $E(\boldsymbol{q}, \mathbf{P}, \boldsymbol{b})$ is independent of $\boldsymbol{p}_\mathrm{s}$.
2. When $\boldsymbol{b}$ is entirely zero except for $\boldsymbol{b}_\mathrm{s} = 1$, then $E(\boldsymbol{q}, \mathbf{P}, \boldsymbol{b}) = E_d(\boldsymbol{q}, \boldsymbol{p}_\mathrm{s})$. As a consequence, E reduces to E_d when there is only one example mesh.
3. E should inherit any symmetries of E_d. For example, if E_d is invariant to rigid transformations, then E should be too.

Next, from the perspective of scattered data interpolation there are some properties that E should satisfy in how the contribution of each of the input shapes relates to the constraint set $\mathcal{C}$. There are two classes of artifacts that need to be avoided. Firstly small changes in the handle positions should not lead to discontinuous changes in the result, as in figure 2 row c. Secondly, we want to ensure that interpolations of similar shapes are preferred over interpolations of disparate shapes, avoiding the artifact in figure 2 row d where the green shape is ignored in favor of a blend of the red and blue shapes.

4. In practice, the vector $\boldsymbol{q}_{opt} = \arg\min_{\boldsymbol{q} \in \mathcal{C}, \boldsymbol{b}} E(\boldsymbol{q}, \mathbf{P}, \boldsymbol{b})$ should vary C^0 continuously with respect to $\mathcal{C}$.
5. Given some deformed mesh $\boldsymbol{q}$ and defining $\boldsymbol{b}_{opt} = \arg\min_{\boldsymbol{b}} E(\boldsymbol{q}, \mathbf{P}, \boldsymbol{b})$, then $\boldsymbol{b}_{opt}$ should typically be zero at entries corresponding to input shapes which are dissimilar from $\boldsymbol{q}$. Relatedly, $\min_{\boldsymbol{b}} E(\boldsymbol{q}, \mathbf{P}, \boldsymbol{b})$ should be zero if and only if $\boldsymbol{q} = \boldsymbol{p}_\mathrm{s}$ for some s (up to the symmetries of E).

Finally, although it is not possible to entirely eliminate local minima since they are present even in a standard non-example-based ARAP energy, we wish to mitigate their effects. We formulate this by ensuring that:

6. If E_d is convex, then so is E.

This last point has a small subtlety which is worth noting. We require not only that a convex E_d leads to a convex E, but also that E does not depend on the nonconvexity or nonlinearity of E_d in order to satisfy properties 1-5. This is in contrast to shape space energies which, as illustrated in figure 2 rows d and e, give either a convex optimization (row d) or prefer interpolating between similar input shapes (row e), *but not both*.

To our knowledge no existing elastic energy satisfies all of these properties. For example although most techniques for example-based mesh deformation satisfy properties 1-3, energy interpolation fails to satisfy property 4, shape space energies with a linear blend function fail to satisfy property 5, and shape space energies with a nonlinear blend function fail to satisfy property 6. Each of these leads to noticeable artifacts when applied to stylized mesh-based inverse kinematics, as shown in figure 2 rows c-e.

Our technique provides example-based deformations satisfying the above six properties by employing a novel method for generalizing a non-example-based energy $E_d(\boldsymbol{q}, \boldsymbol{p})$ to an example-based energy $E(\boldsymbol{q}, \mathbf{P}, \boldsymbol{b})$. At its core, this approach is quite simple, and begins

by calculating a set of $|\mathcal{S}|$ scalar constants $d_1, \ldots, d_{|\mathcal{S}|}$ as:

$$d_{\mathrm{s}} = \min_{\boldsymbol{q}_{\mathrm{s}} \in \mathcal{C}} E_d(\boldsymbol{q}_{\mathrm{s}}, \boldsymbol{p}_{\mathrm{s}}) \tag{4}$$

Here $\boldsymbol{q}_{\mathrm{s}}$ is used instead of $\boldsymbol{q}$ to indicate that a deformed mesh is computed independently for each d_{s}. The different d_{s} values are collected together into a single length-$|\mathcal{S}|$ vector $\boldsymbol{d}$. Our example-based elastic energy E is then defined as.

$$E(\boldsymbol{q}, \mathbf{P}, \boldsymbol{b}) = E_d(\boldsymbol{q}, \mathrm{blend}(\mathbf{P}, \boldsymbol{b})) + \lambda \boldsymbol{d}^T \boldsymbol{b}$$

The optimization to solve for $\boldsymbol{q}$ can then be simply defined as:

$$\begin{aligned} \underset{\boldsymbol{q} \in \mathcal{C}, \boldsymbol{b}}{\arg\min} \quad & E_d(\boldsymbol{q}, \mathrm{blend}(\mathbf{P}, \boldsymbol{b})) + \lambda \boldsymbol{d}^T \boldsymbol{b} \\ \text{s.t.} \quad & \sum_{\mathrm{s}}^{|\mathcal{S}|} \boldsymbol{b}_{\mathrm{s}} = 1 \\ & 0 \leq \boldsymbol{b}_{\mathrm{s}} \qquad 1 \leq \mathrm{s} \leq |\mathcal{S}| \end{aligned} \tag{5}$$

where λ is a pre-specified scalar (set to 0.5 unless otherwise noted). Note that the value of each d_{s} only depends on $\boldsymbol{p}_{\mathrm{s}}$ and $\mathcal{C}$ so in practice they need only be recomputed once each time the user moves a handle. We also note that the simple formulation shown here will be complicated somewhat later on when we integrate it with an ARAP energy, particularly by the allowance that the $\boldsymbol{b}_{\mathrm{s}}$ values may spatially very over the mesh, but the core intuition behind it will remain the same.

This optimization can be interpreted in several different ways. Firstly it can be viewed as a weighted-L_1 regularization applied to the shape space distance $E_d(\boldsymbol{q}, \mathrm{blend}(\mathbf{P}, \boldsymbol{b}))$, where the weights in the regularization are themselves defined by minimizing E_d. This has the effect of penalizing the example meshes for which $\boldsymbol{p}_{\mathrm{s}}$ is far from $\mathcal{C}$, as defined by E_d. The particular weights in $\boldsymbol{d}$ are important to the correct functioning of our energy. For example, using an unweighted L_1 regularizer on $\boldsymbol{b}$ has no effect, since the constraint that $\boldsymbol{b}$ be nonnegative and sum to one already enforces that $\|\boldsymbol{b}\|_1 = 1$, so the regularizer only adds a constant term and does not alter the location of the optimum. Alternatively equation 5 can be viewed as taking an energy interpolation formulation and using a distance-from-subspace term $E_d(\boldsymbol{q}, \mathrm{blend}(\mathbf{P}, \boldsymbol{b}))$ to smooth out the discontinuous transitions. We prefer to interpret it as a kind of hybrid between energy interpolation and shape spaces, and indeed it is equivalent to a shape-space energy when $\lambda = 0$ and approaches energy interpolation as $\lambda \to \infty$.

When applied to example-based mesh deformation, we find that the value of $\lambda = 0.5$ works well. Larger values of λ increasingly tend toward abrupt transitions between the input shapes. Smaller (but non-zero) values instead tend toward a generalized version of piecewise linear interpolation, with the drawback that the influence of an input shape can fall off increasingly abruptly when the deformation moves outside the convex span of the input shapes. See figure 8 for examples of these behaviors. We prefer defining E_d using the distance $\|\boldsymbol{q} - \boldsymbol{p}\|_2$ instead of the squared distance $\|\boldsymbol{q} - \boldsymbol{p}\|_2^2$ since within span of the input shapes the non-squared version better approximates a piecewise linear interpolation for non-infinitesimal values of λ. We should also note that this method only satisfies the continuity property (number 4 in the list at the beginning of this section) when E_d is strictly convex, so using a non-squared distance for E_d does not necessarily reconstruct f as an everywhere continuous function. Although this is possible to fix by redefining E_d to be strictly convex (or, in our case, strictly convex when holding a set of local rotations fixed), we have observed that the discontinuities are rare enough that they do not visually impact the quality of the results.

5 As Multi-Rigid as Possible Deformations

We use the approach described in section 4 to derive a new technique for example based mesh manipulation based on an as-rigid-as-possible elastic energy. We call this technique *as-multi-rigid-as-possible* (AMRAP) mesh deformation. Since this approach is based on the optimization in equation 5, it inherits many of the desirable properties of as-rigid-as-possible deformations, such as invariance to rigid transformations and minimization of scale and shear distortions as well as the ability to robustly and smoothly interpolate between the input shapes. In addition, this approach also supports spatially localized changes in how the different example shapes are combined. To allow real-time interaction even with high-resolution meshes, we employ a reduced coordinate model based on linear blend skinning, described next.

5.1 Deformation Model

As noted in section 3, we allow a user to manipulate a mesh by controlling a set $\mathcal{H}$ of *handles*. We employ a linear blend skinning model where each handle is associated with a per-shape $k \times (k+1)$-dimensional matrix representing an affine transform for the handle. These transforms are packed into a $|\mathcal{H}| \times |\mathcal{S}|$ array $\mathbf{T}$ so that the affine transform for the h^{th} handle and s^{th} shape is $\mathbf{T}_{\mathrm{hs}}$. The position of the complete set of vertices in $\mathcal{P}$ describing a deformed mesh is computed from $\mathbf{T}$ by linear blend skinning. This uses a $|\mathcal{H}| \times |\mathcal{P}|$ matrix of skinning weights W where W_{hp} gives the amount of influence that the h^{th} handle has on the position of the p^{th} point. These skinning weights can be defined in many ways, including painting by an artist, but in our implementation we select a subset of the vertices in the mesh for each handle, fix their skinning weights to 1, and extend the weights to the rest of the mesh using bounded biharmonic weights [Jacobson et al. 2011]. The length-$|\mathcal{P}|$ array of k-dimensional points $\boldsymbol{q}$ giving the shape of the deformed mesh is then computed as:

$$\begin{aligned} \boldsymbol{q}_{\mathrm{p}} &= \sum_{\mathrm{hs}} W_{\mathrm{hp}} \mathbf{T}_{\mathrm{hs}} \begin{bmatrix} \boldsymbol{P}_{\mathrm{sp}} \\ 1 \end{bmatrix} \\ &= \sum_{\mathrm{hs}} \boldsymbol{M}_{\mathrm{hsp}} \mathbf{T}_{\mathrm{hs}}^T \end{aligned} \tag{6}$$

where to reduce notational clutter the bounds of the sum (i.e. $\mathrm{h} = 1 \ldots |\mathcal{H}|, \mathrm{s} = 1 \ldots |\mathcal{S}|$) are taken to be implicit from the names of the indices it is summing over. $\boldsymbol{M}$ is a $|\mathcal{H}| \times |\mathcal{S}| \times |\mathcal{P}|$ array of length-$k+1$ row vectors combining the effect of W and $\boldsymbol{P}$ on $\boldsymbol{q}$ in terms of $\mathbf{T}$.

Although this deformation model is mathematically equivalent to standard linear blend skinning, its interpretation is slightly different since it combines both different handles and different shapes. For instance, scaling a particular $\mathbf{T}_{\mathrm{hs}}$ by a constant factor increases or decreases the contribution of the s^{th} shape to $\boldsymbol{q}$ in the vicinity of the h^{th} handle. On the other hand, multiplying each $\mathbf{T}_{\mathrm{hs}}$ by the same rigid transformation for $1 \leq \mathrm{s} \leq |\mathcal{S}|$ has the effect of rigidly transforming the portion of $\boldsymbol{q}$ near the h^{th} handle. Equation 6 thus describes $\boldsymbol{q}$ as a combination of spatially localized geometric transformations *and interpolations* of the input meshes.

This deformation differs from that described by Jones et al. [2016] in that it is entirely linear. This simplifies its use in the AMRAP optimization and is helpful in allowing our method to run in real time. Although this linear nature means that it is capable of representing highly distorted deformations, the fact that the AMRAP energy attempts to preserve local rigidity naturally avoids these sit-

uations and we have not observed any artifacts resulting from this linearity in practice.

5.2 Deformation Energy

Describing the shape of a deformed mesh by manually specifying $\mathbf{T}$ is tedious, particularly so when there is more than one shape. We instead allow a user to directly manipulate the shape with a simple click-and-drag interface and automatically infer $\mathbf{T}$ via an energy minimizing optimization. This leads to both automatic geometric distortions and automatic determination of how much each of the input shapes contributes to different regions of the final mesh.

Our deformation energy is based on generalizing an as-rigid-as-possible (ARAP) energy [Igarashi et al. 2005] with equation 5 to allow interpolation between a set of shapes. This leads to an as-multi-rigid-as-possible energy where the rigidity is measured with respect to the *set* of input shapes described by $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{|\mathcal{S}|}$ rather than with respect to a single mesh.

A standard ARAP energy, denoted E_{arap}, measures the distortion of a shape $\boldsymbol{q}$ with respect to a reference shape $\boldsymbol{p}$ by a sum of local deviations from perfect rigidity. We represent both $\boldsymbol{q}$ and $\boldsymbol{p}$ as an array of size $|\mathcal{P}|$, each element of which is a k-dimensional vector representing the position of a single vertex. Restated here for reference in the form given by [Jacobson et al. 2012a], this energy can be written:

$$\begin{aligned} E_{arap} &= \frac{1}{2}\sum_{\mathrm{g}} \sum_{(\mathrm{p}_1,\mathrm{p}_2)\in\mathcal{G}_\mathrm{g}} c_{\mathrm{gp}_1\mathrm{p}_2} \left\| \left(\boldsymbol{q}_{\mathrm{p}_2} - \boldsymbol{q}_{\mathrm{p}_1}\right) - \mathbf{R}_\mathrm{g}\left(\boldsymbol{p}_{\mathrm{p}_2} - \boldsymbol{p}_{\mathrm{p}_1}\right) \right\|^2 \\ &= \frac{1}{2}\sum_{\mathrm{g}} \sum_{\mathrm{e}\in\mathcal{G}_\mathrm{g}} c_{\mathrm{ge}} \left\| \sum_{\mathrm{p}} A_{\mathrm{ep}}\boldsymbol{q}_\mathrm{p} - \mathbf{R}_\mathrm{g}\sum_{\mathrm{p}} A_{\mathrm{ep}}\boldsymbol{p}_\mathrm{p} \right\|^2 \\ &= \frac{1}{2}\sum_{\mathrm{g}} E_\mathrm{g}(\boldsymbol{q}, \mathbf{R}_\mathrm{g}, \boldsymbol{p}) \end{aligned} \tag{7}$$

Where $\mathcal{G}$ is a set of edge groups and g indexes over $1, \ldots, |\mathcal{G}|$. As in equation 6, the bounds of the summation are taken to be implicit from the names of the indices being summed over. Further, $\mathbf{R}_\mathrm{g} \in SO(k)$ is a local rotation matrix associated with the g^{th} edge group, c_ge is a weighting term typically calculated with the cotangent Laplacian [Chao et al. 2010], and A is the $|\mathcal{E}| \times |\mathcal{P}|$ (sparse) directed adjacency matrix corresponding to the set $\mathcal{E}$ of edges of the triangular mesh on $\boldsymbol{q}$. As suggested by [Jacobson et al. 2012a] we compute $\mathcal{G}$ by performing k-means clustering on W with the number of clusters set to $2|\mathcal{H}|$. This allows us to use a relatively small number of edge groups and is important for real-time performance since the support for spatially localized blending includes additional variables and constraints for each edge group (equation 20).

Our AMRAP energy is based on generalizing equation 7 to interpolate between multiple shapes using the format outlined previously in section 4. Doing so is slightly more involved than equation 5 would make it appear for two reasons. Firstly we must modify the blend function to account for the fact that an ARAP energy explicitly optimizes over a local rotation matrix for each edge group in $\mathcal{G}$. Secondly, the blend function will need to be further modified in order to support interpolations which spatially vary over the mesh.

Both these modifications are achieved by defining the local rotation matrices and the blend weights both per shape and per edge group. Accordingly, we represent the collection $\mathbf{R}$ of all the local rotations with a $|\mathcal{G}| \times |\mathcal{S}|$ array of $k \times k$ rotation matrices. Similarly, the per-edge-group blend weights for the different $\boldsymbol{p}_1, \ldots, \boldsymbol{p}_{|\mathcal{S}|}$ shapes in $\boldsymbol{P}$ are collected into a $|\mathcal{G}| \times |\mathcal{S}|$ matrix denoted B. The interpolation function used to define the analogue of a shape space for an AMRAP energy is then defined as:

$$\left[\mathrm{blend}_\mathrm{g}(\boldsymbol{P}, B)\right]_\mathrm{e} = \sum_{\mathrm{sp}} B_\mathrm{gs}\mathbf{R}_\mathrm{gs}A_\mathrm{ep}\boldsymbol{P}_\mathrm{sp} \tag{8}$$

To avoid scaling artifacts or negative shape contributions we also constrain $\forall \mathrm{g}, \mathrm{s} : B_\mathrm{gs} \geq 0$ and $\forall \mathrm{g} : \sum_\mathrm{s} B_\mathrm{gs} = 1$. Since $\mathrm{blend}_\mathrm{g}(\boldsymbol{P}, B)$ computes a *vector* of interpolated edge lengths, we use the notation $\left[\mathrm{blend}_\mathrm{g}(\boldsymbol{P}, B)\right]_\mathrm{e}$ to refer a single element corresponding to the e^{th} edge.

This function can be interpreted as, for the g^{th} edge group and each edge $\mathrm{e} \in \mathcal{G}_\mathrm{g}$ first rotating the corresponding edge for each shape in $\mathcal{S}$ by $\mathbf{R}_\mathrm{gs}$, then linearly blending the edges for the different shapes together according to the weights given by B_gs. Because the instances of an edge group within each shape are rotationally aligned, and because each edge group relates only to a small localized portion of the complete mesh we avoid the normal artifacts associated with linearly interpolating directly between vertex positions.

Given the definition of the blend function in equation 8, the elastic energy E_d on which the AMRAP energy is based is analogous to a standard ARAP energy, but employing the blend function to locally interpolate and rotate the different input shapes:

$$E_d(\boldsymbol{q}, B, \mathbf{R}, \boldsymbol{P}) = \frac{1}{2}\sum_{\mathrm{g}} \sum_{\mathrm{e}\in\mathcal{G}_\mathrm{g}} c_\mathrm{ge} \left\| \sum_{\mathrm{p}} A_\mathrm{ep}\boldsymbol{q}_\mathrm{p} - \mathrm{blend}_\mathrm{g}(\boldsymbol{P}, B)_\mathrm{e} \right\|^2 \tag{9}$$

We also require a definition for $\boldsymbol{d}$ in equation 5. To support spatially varying geometric deformations within an ARAP-style formulation, we extend this somewhat by instead computing a $|\mathcal{G}| \times |\mathcal{S}|$ matrix D. Each column of this matrix is defined by first solving $\boldsymbol{q}_\mathrm{s} = \arg\min_{\boldsymbol{q}\in\mathcal{C},\mathbf{R}} E_{arap}(\boldsymbol{q}, \mathbf{R}, \boldsymbol{p}_\mathrm{s})$ then extracting the per-edge-group costs by setting $D_\mathrm{gs} = E_\mathrm{g}(\boldsymbol{q}_\mathrm{s}, \mathbf{R}, \boldsymbol{p}_\mathrm{s})$. The function E_{arap} (from equation 7) appears instead of E_d since when applied to just a single shape $\boldsymbol{p}_\mathrm{s}$, $E_d(\boldsymbol{q}, B, \mathbf{R}, \boldsymbol{p}_\mathrm{s})$ reduces exactly to $E_{arap}(\boldsymbol{q}, \mathbf{R}, \boldsymbol{p}_\mathrm{s})$.

Finally, the full AMRAP elastic energy is written in a form analogous to that prescribed by equation 5.:

$$E(\boldsymbol{q}, B, \mathbf{R}, \boldsymbol{P}) = \sqrt{E_d(\boldsymbol{q}, B, \mathbf{R}, \boldsymbol{P})} + \lambda \sum_{\mathrm{gs}} B_\mathrm{gs}\sqrt{D_\mathrm{gs}} \tag{10}$$

The use of the square root in the definition allows for the results to better approximate piecewise linear interpolations as $\mathcal{C}$ is changed. The version without the square roots also works relatively well, and may be a suitable alternative in situations where a low computational cost is paramount. Nevertheless, since we have found that version in equation 10 can be computed quickly enough in our test cases, we use it for all of the examples in this paper.

5.3 Solving with Rotations Held Fixed

We solve for a deformed mesh by minimizing equation 10 with an alternating algorithm similar to that used for a standard ARAP energy. This algorithm alternates between two steps. First $\mathbf{R}$ is held fixed and $\mathbf{T}$ and B are solved for. Then in the second step $\mathbf{T}$ and B are held fixed and $\mathbf{R}$ is solved for. This process is repeated until convergence, or until a new frame needs to be displayed in which case it is terminated early. We begin by discussing how to solve for $\mathbf{T}$ and B under the assumption that $\mathbf{R}$ is held constant.

Since we use the linear blend skinning deformation model given in section 5.1, solving for $\mathbf{T}$ is equivalent to solving for the vector of vertex positions $\boldsymbol{q}$ representing the deformed mesh. Plugging

equation 6 into equation 9 to express $E_d(\boldsymbol{q}, B, \mathbf{R}, \boldsymbol{P})$ in terms of $\mathbf{T}$ as $E_d(\mathbf{T}, B, \mathbf{R}, \boldsymbol{P})$ and expanding yields E_d as the sum of three terms:

$$\begin{aligned} &E_d(\mathbf{T}, B, \mathbf{R}, \boldsymbol{P}) \\ &\quad = \frac{1}{2} E_{d1}(\mathbf{T}, \boldsymbol{P}) - E_{d2}(\mathbf{T}, B, \mathbf{R}, \boldsymbol{P}) + \frac{1}{2} E_{d3}(B, \mathbf{R}, \boldsymbol{P}) \end{aligned} \tag{11}$$

The terms in this equation are straightforward to calculate and to implement in code, if somewhat notationally awkward to express:

$$E_{d1}(\mathbf{T}, \boldsymbol{P}) = \sum_{\mathrm{h}_1\mathrm{h}_2\mathrm{s}_1\mathrm{s}_2} \mathrm{tr}\left(\mathbf{T}_{\mathrm{h}_1\mathrm{s}_1} \widetilde{\mathbf{L}}_{\mathrm{h}_1\mathrm{h}_2\mathrm{s}_1\mathrm{s}_2} \mathbf{T}^T_{\mathrm{h}_2\mathrm{s}_2}\right) \tag{12}$$

with

$$\widetilde{\mathbf{L}}_{\mathrm{h}_1\mathrm{h}_2\mathrm{s}_1\mathrm{s}_2} = \sum_{\mathrm{p}_1\mathrm{p}_2} L_{\mathrm{p}_1\mathrm{p}_2} \boldsymbol{M}_{\mathrm{h}_1\mathrm{s}_1\mathrm{p}_1} \boldsymbol{M}^T_{\mathrm{h}_2\mathrm{s}_2\mathrm{p}_2} \tag{13}$$

$$L_{\mathrm{p}_1\mathrm{p}_2} = \sum_{\mathrm{g}} \sum_{\mathrm{e}\in\mathcal{G}_\mathrm{g}} c_{\mathrm{ge}} A_{\mathrm{ep}_1} A_{\mathrm{ep}_2} \tag{14}$$

Where L is a $|\mathcal{P}| \times |\mathcal{P}|$ matrix and $\widetilde{\mathbf{L}}$ is a $|\mathcal{H}| \times |\mathcal{H}| \times |\mathcal{S}| \times |\mathcal{S}|$ array of $(k+1) \times (k+1)$ matrices.

Similarly, E_{d2} is expressed as:

$$E_{d2}(\mathbf{T}, B, \mathbf{R}, \boldsymbol{P}) = \sum_{\mathrm{ghs}_1\mathrm{s}_2} B_{\mathrm{gs}_1} \mathrm{tr}\left(\mathbf{R}_{\mathrm{gs}_1} \widetilde{\mathbf{K}}_{\mathrm{ghs}_1\mathrm{s}_2} \mathbf{T}^T_{\mathrm{hs}_2}\right) \tag{15}$$

with

$$\widetilde{\mathbf{K}}_{\mathrm{ghs}_1\mathrm{s}_2} = \sum_{\mathrm{p}} \boldsymbol{K}_{\mathrm{gs}_1\mathrm{p}} \boldsymbol{M}^T_{\mathrm{hs}_2\mathrm{p}} \tag{16}$$

$$\boldsymbol{K}_{\mathrm{gsp}_1} = \sum_{\mathrm{p}_2} \sum_{\mathrm{e}\in\mathcal{G}_\mathrm{g}} c_{\mathrm{ge}} A_{\mathrm{ep}_1} A_{\mathrm{ep}_2} \boldsymbol{P}_{\mathrm{sp}_2} \tag{17}$$

Where $\boldsymbol{K}$ is a $|\mathcal{G}| \times |\mathcal{S}| \times |\mathcal{P}|$ array of length-k vectors and $\widetilde{\mathbf{K}}$ is a $|\mathcal{G}| \times |\mathcal{H}| \times |\mathcal{S}| \times |\mathcal{S}|$ array of $k \times (k+1)$ matrices. Finally:

$$E_{d3}(B, \mathbf{R}, \boldsymbol{P}) = \sum_{\mathrm{gs}_1\mathrm{s}_2} B_{\mathrm{gs}_1} B_{\mathrm{gs}_2} \mathrm{tr}\left(\mathbf{R}_{\mathrm{gs}_1} \mathbf{H}_{\mathrm{gs}_1\mathrm{s}_2} \mathbf{R}^T_{\mathrm{gs}_2}\right) \tag{18}$$

with

$$\mathbf{H}_{\mathrm{gs}_1\mathrm{s}_2} = \sum_{\mathrm{p}_1\mathrm{p}_2} \sum_{\mathrm{e}\in\mathcal{G}_\mathrm{g}} c_{\mathrm{ge}} A_{\mathrm{ep}_1} A_{\mathrm{ep}_2} \boldsymbol{P}_{\mathrm{s}_1\mathrm{p}_1} \boldsymbol{P}^T_{\mathrm{s}_1\mathrm{p}_2} \tag{19}$$

Where $\mathbf{H}$ is a $|\mathcal{G}| \times |\mathcal{S}| \times |\mathcal{S}|$ array of $k \times k$ matrices.

Assuming the $\boldsymbol{P}$ is fixed in advance, these formulas for E_{d1}, E_{d2} and E_{d3} allow E_d to be calculated with a cost independent of $|\mathcal{P}|$ by precomputing $\widetilde{\mathbf{L}}$, $\widetilde{\mathbf{K}}$ and $\mathbf{H}$. A similar substitution of $\boldsymbol{q}$ in terms of $\mathbf{T}$ can be made to solve for D, in which case the result is given in previous work by [Jacobson et al. 2012a]. This allows equation 10 to be minimized with a cost independent of the resolution of the input meshes.

The final optimization for the deformed mesh involves minimizing equation 10 subject to user-specified constraints on the position of select vertices in $\boldsymbol{q}$. For each handle h this constraint is modeled by associating a single vertex $\mathrm{p_h}$ with the handle and enforcing $\sum_\mathrm{s} \boldsymbol{M}_{\mathrm{hsp_h}} \mathbf{T}^T_{\mathrm{hs}} = \boldsymbol{P}_{\boldsymbol{eq}_\mathrm{h}}$ where $\boldsymbol{P}_{\boldsymbol{eq}_\mathrm{h}}$ is the desired position of the vertex in the deformed mesh.

To summarize, the complete optimization for $\mathbf{T}$ and B with $\mathbf{R}$ held fixed is given by:

$$\begin{aligned} \underset{\mathbf{T},B}{\mathrm{argmin}} & \sqrt{E_d(\mathbf{T},B,\mathbf{R},\boldsymbol{P})} + \lambda \sum_{\mathrm{gs}} B_{\mathrm{gs}} \sqrt{D_{\mathrm{gs}}} \\ \text{s.t. } & \sum_\mathrm{s} \boldsymbol{M}_{\mathrm{hsp_h}} \mathbf{T}^T_{\mathrm{hs}} = \boldsymbol{P}_{\boldsymbol{eq}_\mathrm{h}} && 1 \le \mathrm{h} \le |\mathcal{H}| \\ & B_{\mathrm{gs}} \ge 0 && 1 \le \mathrm{g} \le |\mathcal{G}|, \quad 1 \le \mathrm{s} \le |\mathcal{S}| \\ & \sum_\mathrm{s} B_{\mathrm{gs}} = 1 && 1 \le \mathrm{g} \le |\mathcal{G}| \end{aligned} \tag{20}$$

To minimize equation 20, observe that $E_d(\mathbf{T}, B, \mathbf{R}, \boldsymbol{P})$ is quadratic in $\mathrm{vec}(\mathbf{T}, B)$ where vec is the vectorization operator stacking the scalar components of $\mathbf{T}$ and B into a single vector of length $|\mathcal{H}||\mathcal{S}|k(k+1) + |\mathcal{G}||\mathcal{S}|$. This implies that solving equation 20 for $\mathbf{T}$ and B reduces to minimizing the sum of the square root of a quadratic form and a linear function subject to a set of linear constraints. It is here that we see the benefits of the fact that equation 5 preserves convexity, as this problem is an instance of a well-studied form of convex optimization known as a second-order cone program (SOCP) [Boyd and Vandenberghe 2004] which can quickly be globally minimized. Although problems of this form are easily solved with an off-the-shelf conic optimizer, we have found it slightly more efficient to instead solve with a general nonlinear optimizer, for which we use SNOPT [Gill et al. 1997].

5.4 Solving for the Local Rotations

The second phase in our alternating minimization solves for $\mathbf{R}$ while holding $\mathbf{T}$ and B fixed. Minimizing equation 10 for $\mathbf{R}$ directly, however, leads to significant artifacts. In contrast to a standard ARAP energy, equation 8 employs a local rotation matrix $\mathbf{R}_{\mathrm{gs}}$ for each pair of edge group and base shape, rather than simply a rotation matrix for each edge group. This allows the different base shapes to be automatically locally rotationally aligned, but has the disadvantage that directly including the $\mathbf{R}_{\mathrm{gs}}$ rotations in an energy minimization gives rise to highly non-rigid transformations since the rotation matrices for the different shapes can sum to counteract each other.

We remedy this issue by solving for the local rotations independently for each input shape by aligning $\boldsymbol{p}_\mathrm{s}$ directly to $\boldsymbol{q}$ without any interpolation between different shapes. This is achieved by defining $\mathbf{R}_{\mathrm{gs}}$ as the g,s$^{\mathrm{th}}$ element of the array $\mathbf{R}$ minimizing equation 10 for just the single base shape $\boldsymbol{p}_\mathrm{s}$.

$$\mathbf{R}_{\mathrm{gs}} = \left[\underset{\mathbf{R}\in SO(k)}{\arg\min}\, E(\boldsymbol{q}, B, \mathbf{R}, \boldsymbol{p}_\mathrm{s})\right]_{\mathrm{gs}} \tag{21}$$

Similar to equation 8, the notation $[\mathbf{X}]_{\mathrm{gs}}$ is used to refer to a single element in the matrix $\mathbf{X}$. As when solving for $\mathbf{T}$ and B in section 5.3, this can be solved without explicitly representing $\boldsymbol{q}$ by instead representing the minimization in terms of $\mathbf{T}$ and B:

$$\begin{aligned} \mathbf{R}_{\mathrm{gs}} &= \left[\underset{\mathbf{R}\in SO(k)}{\arg\max}\, E_{d2}(\mathbf{T}, B, \mathbf{R}, \boldsymbol{p}_\mathrm{s})\right]_{\mathrm{gs}} \\ &= \underset{\mathbf{R}\in SO(k)}{\arg\max}\, \mathrm{tr}\left(\mathbf{R}\widehat{\mathbf{K}}_{\mathrm{gs}}\right) \end{aligned} \tag{22}$$

where $\widehat{\mathbf{K}}$ is the $|\mathcal{G}| \times |\mathcal{S}|$ array of $k \times k$ matrices representing contraction of $\widetilde{\mathbf{K}}$ by $\mathbf{T}$ and B:

$$\widehat{\mathbf{K}}_{\mathrm{gs}_1} = \sum_{\mathrm{hs}_2} B_{\mathrm{gs}_1} \widetilde{\mathbf{K}}_{\mathrm{ghs}_1\mathrm{s}_2} \mathbf{T}^T_{\mathrm{hs}2} \tag{23}$$

Note that the term E_{d3} does not appear in equation 22 because when computed for a single shape $\boldsymbol{p}_s$, the instances of $\mathbf{R}$ in equation 18 cancel, rendering E_{d3} independent of $\mathbf{R}$. The matrix $\mathbf{R}_{gs}$ maximizing equation 22 over $SO(k)$ can be found independently for each g, s by solving a Procustes alignment problem as $\mathbf{R}_{gs} = \mathbf{V}\mathbf{U}^T$ where $\mathbf{R}_{gs} = \mathbf{U}\mathbf{D}\mathbf{V}^T$ is the singular value decomposition of $\widehat{\mathbf{K}}_{gs_1}$ [Sorkine and Alexa 2007].

5.5 Complete Optimization

The complete AMRAP optimization algorithm alternates between solving for $\mathbf{T}$ and B as in section 5.3 and solving for $\mathbf{R}$ as in section 5.4. Technically, this alternating minimization is not guaranteed to converge, given that we solve for $\mathbf{R}$ by minimizing a different energy than given by equation 20. Surprisingly we have not observed this to be a serious problem in practice, and the rare cases where we have observed it at all have been limited to a relatively minor jittering. Nevertheless, if a guarantee of convergence is desired (at the cost of some additional complexity) we describe a modified optimization algorithm suited to this task in appendix A.

We would also like to draw attention to one simplification of the optimization algorithm which may be useful in situations where rotations are not required. If rotations are not included in the optimization, then it is sufficient to solve equation 20 once for $\mathbf{T}$ and B. In addition, each suboptimization involved in computing D reduces to solving a single linear system. This allows us to efficiently compute a *globally optimal* deformed mesh $\boldsymbol{q}$ by solving a series of $|\mathcal{S}|$ linear systems followed by a single SOCP. Since our method does not rely on nonlinearities in the objective function to reliably interpolate between shapes, when applicable this technique can give good results along with a guarantee of global optimality. We have found facial expression manipulation to be a domain where this applies well.

6 Results

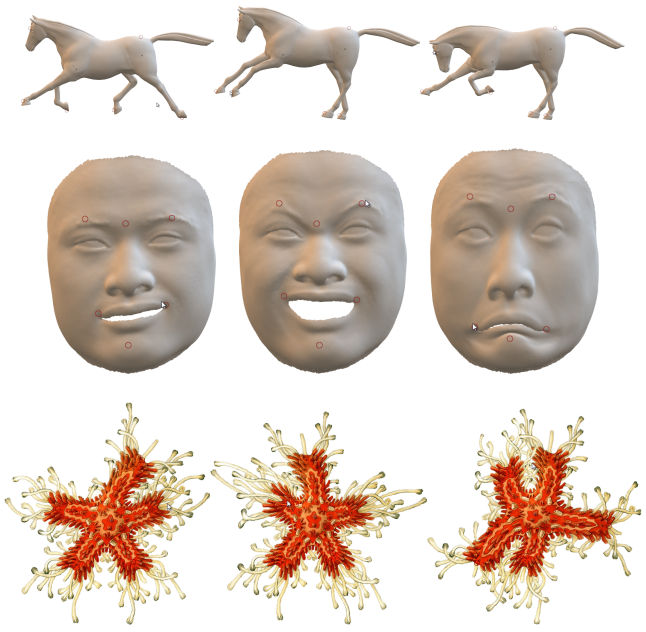


Figure 3: *AMRAP as applied to a range of different situations. All of these examples also make use of spatially localized blend weights.*

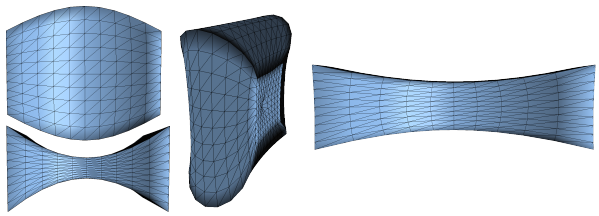


Figure 5: *Artifacts appearing in the MeshIK system of Sumner et al. [2005]. From left to right: discontinuous changes in shape, counterintuitive deformations resulting from local minima, and regression to the mean shape.*

We illustrate example-based mesh deformation using an AMRAP energy in the context of an interactive system for mesh-based inverse kinematics. After loading a set of example shapes and specifying the positions of a set of handles on those shapes by selecting vertices on one of the meshes, the user can interactively click and drag the handles around while the resulting mesh deforms accordingly. Our system can operate on either two- or three-dimensional meshes, and makes no requirement that the 3D inputs be tetrahedralizable. With respect to implementation complexity, our implementation of the AMRAP solver requires approximately 900 lines of C++ (including the trust region algorithm in appendix A). For comparison, our equivalently verbose implementation of the method of Jacobson et al. [2012a] requires approximately 300 lines of code. We have found AMRAP deformations to work well for a wide range of different cases, including click-and-drag animation of stylized creatures, posing articulated characters, and face expression manipulation (figures 9 and 3).

Our approach reliably interpolates between its set of input shapes in an intuitive manner while avoiding the artifacts present in both existing shape space and energy interpolation formulations. Figure 2 illustrates a comparison between our approach and a selection of existing methods as applied to the input meshes in figure 1. As the shape is stretched and contracted by the user dragging the rightmost handle, our approach smoothly interpolates between the three in put shapes, both without (row a) and with (row b) optimizing for local rotations. In contrast an energy interpolation approach such as that of Chao et al. [2010] exhibits discontinuous transitions (row c) while a nonlinear shape space method such as that of Schumacher et al. [2012] shows noticeable artifacts resulting from local minima in how the blending weights are computed. Similar artifacts can be observed using the MeshIK system of Sumner et al. [2005] (figure 5). Further examples how our approach avoids these sort of artifacts are shown in figure 6 and 4. We have found this property of reliability to be very useful in authoring deformable models, as it ensures that all of the artist-provided input shapes are actually used in an intuitive way in the final result.

We have also observed a benefit to the expressiveness of our deformations from the use of spatially varying blends of the input shapes. This is particularly the case with deformations of articulated characters, where it allows different limbs to draw from different exemplars. As shown in figure 7 omitting this capability from our approach can lead to noticeable artifacts in how the limbs of the character deform, and removes the ability of the user to freely manipulate the individual limbs without issue.

To motivate our choice of $\lambda = 0.5$ in equation 20, figure 8 illustrates how our method interpolates the set of three input meshes shown in figure 1 with respect to the position of the rightmost handle. As can be observed, setting λ in equation 20 to a large value approximates a piecewise constant deformation by projection into

Figure 4: *The top row shows a snake animated with AMRAP, while the bottom row shows the same inputs applied to the method of Schumacher et al. [2012].*

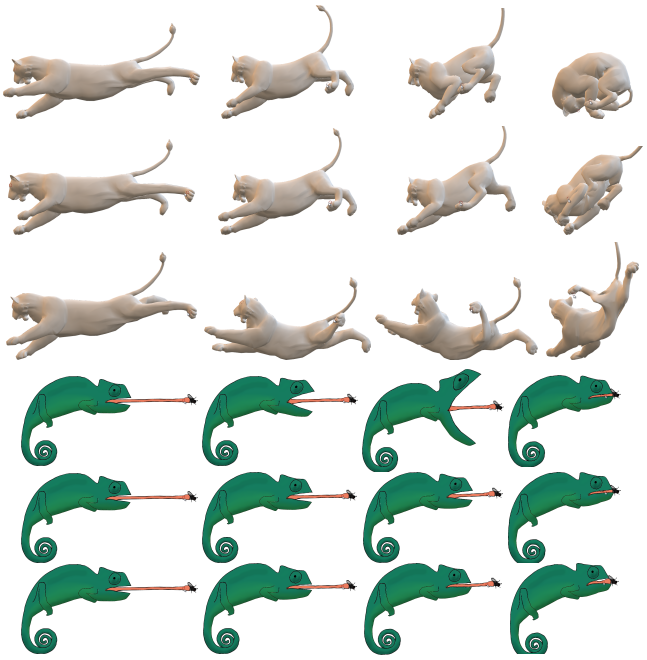


Figure 6: *For each of the lion and chameleon cases, the three rows show a different method applied to the same inputs. From top to bottom: AMRAP, shape-space-only AMRAP ($\lambda = 0$), and [Schumacher et al. 2012]. Note how without the energy in equation 5 the lion does not properly curl up, nor does the chameleon open its mouth as intended.*

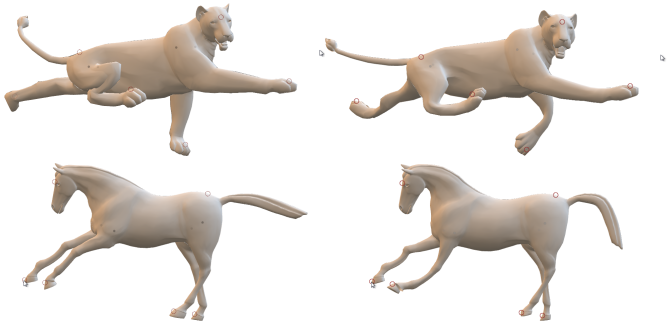


Figure 7: *The left column shows AMRAP deformations with spatially varying blending, while the right column shows AMRAP without spatially varying blending and its associated artifacts in the way the legs of the lion and horse bend.*

name	k	$\lvert\mathcal{P}\rvert$	$\lvert\mathcal{S}\rvert$	$\lvert\mathcal{H}\rvert$	precompute	runtime
snake	2	817	5	2	0.05	0.01
caterpillar	2	481	3	2	0.03	0.005
starfish	2	3532	3	6	0.65	0.015
plant	2	3152	3	4	0.8	0.01
horse	3	8431	3	10	30	0.05
lion	3	5000	6	7	5.0	0.15
cat	3	7307	6	7	8.0	0.15
head	3	15941	7	6	79	0.17
face	3	23725	6	9	74	0.22

Table 1: *The average computation time in seconds of our approach on various models.*

the nearest Voronoi cell. Setting a λ to a small value results in near piecewise linear interpolations, but tends to be overzealous in lessening the contribution of nearby input shapes when none of the input meshes can accurately match the user's constraints. Although it is not obvious in the figure, small values of λ also lead to a more poorly conditioned optimization and take longer to converge to a given accuracy. The default value of $\lambda = 0.5$ provides a good compromise.

Table 1 lists the runtime performance of our approach on a number of examples as measured on an unoptimized single-threaded C++ implementation running on a 2.4 GHz Intel®Xeon®E5-2630. The bulk of the runtime computation is spent within the nonlinear optimizer solving equation 20. The times shown in the table are for a single iteration of the alternating optimization. Since we initialize each frame with the result from the previous frame, this still produces high-quality results. Nevertheless in the videos accompanying this paper we run the optimizer for more iterations so long as this keeps the performance at real-time framerates. Since the head and face models do not benefit from rotations, we use the formulation without local rotations for them, which does not significantly change the performance. Depending on the model, between 50% and 90% of the precomputation time is spent calculating the skinning weights for each handle.

In testing AMRAP deformations on a variety of different models, we have generally found that very little tweaking or trial and error was needed, and for the most part we could go straight from the input shapes, define a set of handles, and the deformations would be as desired. In the case that some tuning was required, it was in the form of adding a few extra handles with positions that were allowed to vary freely as a means of injecting extra degrees of freedom into the deformation (a process analogous to the additional weight functions used by Jacobson et al. [2012a]). Of the examples appearing in this paper and its accompanying video, the lion and the horse are the only ones where this was necessary, where the additional handles appear as gray dots.

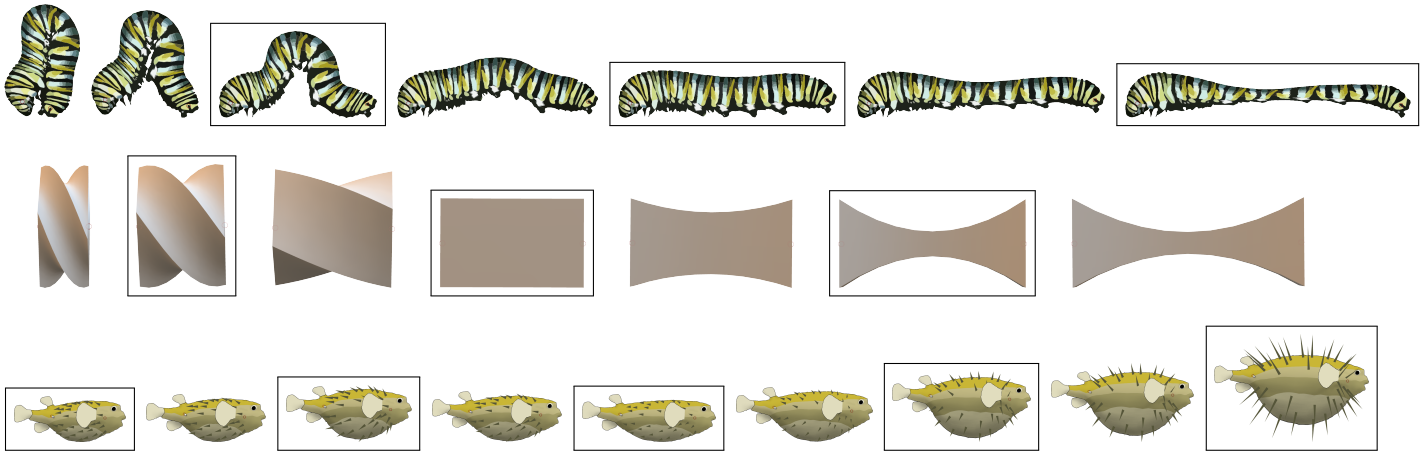


Figure 9: *Three, examples of results generated by our method, each shown on a different row. The input meshes are surrounded by a black box, while the other meshes are automatically generated by clicking and dragging a single handle on the shape to various locations.*

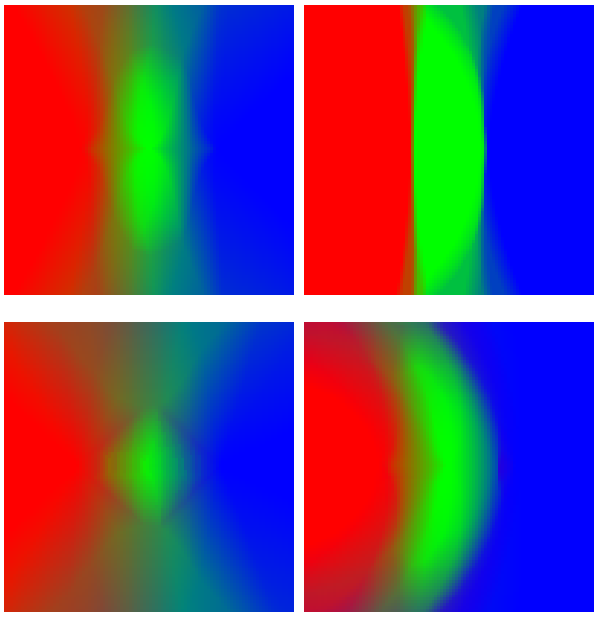


Figure 8: *The average value of B for a deformed mesh generated from the three input shapes in figure 1 as a function of the position of the rightmost handle. From top left to bottom right the first three are for the formulation without rotations using $\lambda = 0.5$, $\lambda = 10$ and $\lambda = 0.01$ respectively. The bottom right shows the full AMRAP optimization with $\lambda = 0.5$.*

7 Conclusions and Future Work

The ability to draw from multiple example shapes provides an easy way to create stylized and expressive deformations with simple user interactions. We have described an approach for animating and manipulating these deformations in real time which works with no parameter tuning across a wide range of different situations, including highly nonphysical stylized deformations.

There are a number of areas for improvements and extensions of this work. Probably foremost is the ability to handle large numbers of handles and input shapes. Although our formulation allows the deformation to be calculated with a speed independent of the input mesh, it bogs down when many handles or input shapes are specified. This makes it difficult to, for instance, apply a large library or motion stream of captured shapes. Since the run time is dominated by solving the conic optimization in equation 20, developing a fast specialized solver is a promising direction. In addition, although our method allows some basic control over how the different shapes are blended via the parameter λ, more rich control would be useful, for instance the ability to provide a higher order smoothness beyond C^0 continuity. Continuing further with this idea, it would also be valuable to have control over the temporal aspects of the deformation. As it stands our technique only uses the spatial positions of the handles, but when generating animations it is likely that some notion of time would be useful as well. This could come in the form of integrating dynamics into the deformation, or in a kinematic fashion where we interpolate between example animations instead of just example shapes.

Acknowledgments

The mesh data for the horse, lion, cat and head models was made available by Robert Sumner and Jovan Popović from the Computer Graphics Group at MIT [Sumner and Popović 2004]. The mesh data for the face model was made available by the University of Washington Graphics and Imaging Laboratory [Zhang et al. 2004]. Additional thanks to Christian Schumacher for assistance with [Schumacher et al. 2012], and to Alexandru Eugen Ichim and Lillian Kittredge for help with proofreading.

References

ALEXA, M., COHEN-OR, D., AND LEVIN, D. 2000. As-rigid-as-possible shape interpolation. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, SIGGRAPH '00, 157–164.

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3d characters. *ACM Trans. Graph. 26*, 3 (July).

BERGERON, P., AND LACHAPELLE, P. 1985. Controlling facial expressions and body movements in the computer generated an-

imated short 'Tony de Peltrie'. In *SigGraph '85 Tutorial Notes, Advanced Computer Animation Course*.

BOTSCH, M., AND SORKINE, O. 2008. On linear variational surface deformation methods. *IEEE Transactions on Visualization and Computer Graphics 14*, 1 (Jan.), 213–230.

BOYD, S., AND VANDENBERGHE, L. 2004. *Convex Optimization*. Cambridge University Press, New York, NY, USA.

CHAO, I., PINKALL, U., SANAN, P., AND SCHRÖDER, P. 2010. A simple geometric model for elastic deformations. *ACM Trans. Graph. 29*, 4 (July), 38:1–38:6.

DER, K. G., SUMNER, R. W., AND POPOVIĆ, J. 2006. Inverse kinematics for reduced deformable models. In *ACM SIGGRAPH 2006 Papers*, ACM, New York, NY, USA, SIGGRAPH '06, 1174–1179.

FENG, W.-W., KIM, B.-U., AND YU, Y. 2008. Real-time data driven deformation using kernel canonical correlation analysis. *ACM Transactions on Graphics (TOG) 27*, 3, 91.

FRÖHLICH, S., AND BOTSCH, M. 2011. Example-driven deformations based on discrete shells. *Computer Graphics Forum 30*, 8, 2246–2257.

GAO, L., LAI, Y.-K., HUANG, Q.-X., AND HU, S.-M. 2013. A data-driven approach to realistic shape morphing. In *Computer graphics forum*, vol. 32, Wiley Online Library, 449–457.

GILL, P. E., MURRAY, W., AND SAUNDERS, M. A. 1997. Snopt: An sqp algorithm for large-scale constrained optimization. *SIAM JOURNAL ON OPTIMIZATION 12*, 979–1006.

GROCHOW, K., MARTIN, S. L., HERTZMANN, A., AND POPOVIĆ, Z. 2004. Style-based inverse kinematics. *ACM Trans. Graph. 23*, 3 (Aug.), 522–531.

HAHN, F., THOMASZEWSKI, B., COROS, S., SUMNER, R. W., COLE, F., MEYER, M., DEROSE, T., AND GROSS, M. 2014. Subspace clothing simulation using adaptive bases. *ACM Trans. Graph. 33*, 4 (July), 105:1–105:9.

HUANG, H., ZHAO, L., YIN, K., QI, Y., YU, Y., AND TONG, X. 2011. Controllable hand deformation from sparse examples with rich details. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, ACM, New York, NY, USA, SCA '11, 73–82.

IGARASHI, T., MOSCOVICH, T., AND HUGHES, J. F. 2005. As-rigid-as-possible shape manipulation. *ACM Trans. Graph. 24*, 3 (July), 1134–1141.

JACOBSON, A., BARAN, I., POPOVIĆ, J., AND SORKINE, O. 2011. Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH) 30*, 4, 78:1–78:8.

JACOBSON, A., BARAN, I., KAVAN, L., POPOVIĆ, J., AND SORKINE, O. 2012. Fast automatic skinning transformations. *ACM Transactions on Graphics (proceedings of ACM SIGGRAPH) 31*, 4, 77:1–77:10.

JACOBSON, A., WEINKAUF, T., AND SORKINE, O. 2012. Smooth shape-aware functions with controlled extrema. *Computer Graphics Forum (proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing) 31*, 5, 1577–1586.

JONES, B., POPOVIC, J., MCCANN, J., LI, W., AND BARGTEIL, A. 2013. Dynamic sprites. In *Proceedings of Motion on Games*, ACM, New York, NY, USA, MIG '13, 17:39–17:46.

JONES, B., THUEREY, N., SHINAR, T., AND BARGTEIL, A. W. 2016. Example-based plastic deformation of rigid bodies. *ACM Trans. on Graphics 35*, 4 (July).

KILIAN, M., MITRA, N. J., AND POTTMANN, H. 2007. Geometric modeling in shape space. *ACM Trans. Graph. 26*, 3 (July).

KOYAMA, Y., TAKAYAMA, K., UMETANI, N., AND IGARASHI, T. 2012. Real-time example-based elastic deformation. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '12, 19–24.

LEVI, Z., AND GOTSMAN, C. 2015. Smooth rotation enhanced as-rigid-as-possible mesh animation. 264–277.

LEWIS, J. P., AND ANJYO, K.-I. 2010. Direct manipulation blendshapes. *IEEE Comput. Graph. Appl. 30*, 4 (July), 42–50.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ACM Press/Addison-Wesley Publishing Co., 165–172.

LIU, L., ZHANG, L., XU, Y., GOTSMAN, C., AND GORTLER, S. J. 2008. A local/global approach to mesh parameterization. In *Proceedings of the Symposium on Geometry Processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SGP '08, 1495–1504.

MARTIN, S., THOMASZEWSKI, B., GRINSPUN, E., AND GROSS, M. 2011. Example-based elastic materials. *ACM Trans. Graph. 30*, 4 (July), 72:1–72:8.

MILLIEZ, A., WAND, M., CANI, M.-P., AND SEIDEL, H.-P. 2013. Mutable elastic models for sculpting structured shapes. In *Computer Graphics Forum*, vol. 32, Wiley Online Library, 21–30.

NIETO, J., AND SUSN, A. 2013. Cage based deformations: A survey. In *Deformation Models*, M. Gonzlez Hidalgo, A. Mir Torres, and J. Varona Gmez, Eds., vol. 7 of *Lecture Notes in Computational Vision and Biomechanics*. Springer Netherlands, 75–99.

SCHUMACHER, C., THOMASZEWSKI, B., COROS, S., MARTIN, S., SUMNER, R., AND GROSS, M. 2012. Efficient simulation of example-based materials. In *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '12, 1–8.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. *SIGGRAPH Comput. Graph. 20*, 4 (Aug.), 151–160.

SEO, J., IRVING, G., LEWIS, J. P., AND NOH, J. 2011. Compression and direct manipulation of complex blendshape models. *ACM Trans. Graph. 30*, 6 (Dec.), 164:1–164:10.

SHEFFER, A., AND KRAEVOY, V. 2004. Pyramid coordinates for morphing and deformation. In *Proceedings of the 3D Data Processing, Visualization, and Transmission, 2Nd International Symposium*, IEEE Computer Society, Washington, DC, USA, 3DPVT '04, 68–75.

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, 135–143.

SONG, C., ZHANG, H., WANG, X., HAN, J., AND WANG, H. 2014. Fast corotational simulation for example-driven deformation. *Computers & Graphics 40*, 49–57.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SGP '07, 109–116.

SUMNER, R. W., AND POPOVIĆ, J. 2004. Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3 (Aug.), 399–405.

SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. In *ACM SIGGRAPH 2005 Papers*, ACM, New York, NY, USA, SIGGRAPH '05, 488–495.

TWIGG, C. D., AND KAČIĆ-ALESIĆ, Z. 2010. Point cloud glue: Constraining simulations using the procrustes transform. In *Proceedings of the 2010 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, SCA '10, 45–54.

WANG, Y., JACOBSON, A., BARBIC, J., AND KAVAN, L. 2015. Linear subspace design for real-time shape deformation. *ACM Trans. Graph. 34*, 4.

WAREHAM, R., AND LASENBY, J. 2008. Bone glow: An improved method for the assignment of weights for mesh deformation. In *AMDO*, Springer, F. J. P. Lpez and R. B. Fisher, Eds., vol. 5098 of *Lecture Notes in Computer Science*, 63–71.

WEBER, O., SORKINE, O., LIPMAN, Y., AND GOTSMAN, C. 2007. Context-aware skeletal shape deformation. In *Computer Graphics Forum*, vol. 26, Wiley Online Library, 265–274.

ZHANG, L., SNAVELY, N., CURLESS, B., AND SEITZ, S. M. 2004. Spacetime faces: High resolution capture for modeling and animation. *ACM Trans. Graph. 23*, 3 (Aug.), 548–558.

ZHANG, W., ZHENG, J., AND THALMANN, N. M. 2015. Real-Time Subspace Integration for Example-Based Elastic Material. *Computer Graphics Forum*.

A Trust Region Optimization

Although it normally works well in practice, the alternating minimization is section 5 is not guaranteed to converge because the step solving for $\mathbf{T}$ and B and the step solving for $\mathbf{R}$ do so by minimizing different energies. To guarantee convergence the nonlinearities introduced by the solution for $\mathbf{R}$ can be incorporated into the solution for $\mathbf{T}$ and B. As this adds a non-trivial additional complexity to the optimizer to cover relatively rare failure cases, we recommend that anyone implementing an AMRAP solver do so first as described in section 5, and only implement the additions here if it appears necessary to do so.

As $\mathbf{T}$ and B are updated, the value of $\mathbf{R}$ changes according to equation 22. To account for this, we linearize the effect of $\mathbf{R}$ on $E(\mathbf{T}, B, \mathbf{R}, \boldsymbol{P})$ and minimize equation 20 with a trust region approach. This linearization approximates E in the vicinity of $\mathbf{R} \approx \mathbf{R_0}$ as:

$$E(\mathbf{T}, B, \mathbf{R}, \boldsymbol{P}) \approx E(\mathbf{T}, B, \mathbf{R_0}, \boldsymbol{P}) + \partial_{\mathbf{R}} E \tag{24}$$

Where $\partial_{\mathbf{R}} E$ is the linearization of the effect the $\mathbf{R}$ has on E as $\mathbf{T}$ and B vary and can be simplified as:

$$\partial_{\mathbf{R}} E = \frac{1}{2\sqrt{E_d(\mathbf{T}, B, \mathbf{R}, \boldsymbol{P})}} \partial_{\mathbf{R}} E_d \tag{25}$$

$$= \frac{1}{4\sqrt{E_d(\mathbf{T}, B, \mathbf{R}, \boldsymbol{P})}} \partial_{\mathbf{R}} E_{d3} \tag{26}$$

Where the last equality results from observing that E_{d1} is independent of $\mathbf{R}$ and that the definition of $\mathbf{R}$ via equation 22 means that $\partial_{\mathbf{R}} E_{d2} = 0$. Letting x represent a single parameter of $\mathbf{T}$ and using equation 18 gives:

$$\partial_{\mathbf{R}} E_{d3} = \sum_{\mathrm{gs}_1 \mathrm{s}_2} B_{\mathrm{gs}_1} B_{\mathrm{gs}_2} \operatorname{tr}\left(\mathbf{H}_{\mathrm{gs}_1\mathrm{s}_2} \mathbf{dR}_{\mathrm{gs}_1\mathrm{s}_2}\right) \tag{27}$$

with

$$\mathbf{dR}_{\mathrm{gs}_1\mathrm{s}_2} = \mathbf{R}_{\mathrm{gs}_2}^T \left(\partial_x \mathbf{R}_{\mathrm{gs}_1}\right) + \left(\partial_x \mathbf{R}_{\mathrm{gs}_2}\right)^T \mathbf{R}_{\mathrm{gs}_1} \tag{28}$$

where $\partial_x \mathbf{R}_{\mathrm{gs}}$ is calculated as described as described by Twigg and Kačić-Alesić [2010]:

$$\partial_x \mathbf{R}_{\mathrm{gs}} = \mathbf{U} \boldsymbol{\Omega} \mathbf{V}^T \tag{29}$$

where $\mathbf{U}$ and $\mathbf{V}$ come from the singular value decomposition of $\widehat{\mathbf{K}}_{\mathrm{gs}}$ and $\boldsymbol{\Omega}$ is a $k \times k$ matrix such that:

$$\boldsymbol{\Omega}_{\mathrm{ij}} = \frac{\left[\mathbf{U}^T \left(\partial_x \widehat{\mathbf{K}}_{\mathrm{gs}}\right) \mathbf{V}\right]_{\mathrm{ij}} - \left[\mathbf{U}^T \left(\partial_x \widehat{\mathbf{K}}_{\mathrm{gs}}\right) \mathbf{V}\right]_{\mathrm{ji}}}{\mathbf{D}_{\mathrm{ii}} + \mathbf{D}_{\mathrm{jj}}} \tag{30}$$

The term $\partial_x \widehat{\mathbf{K}}_{\mathrm{gs}}$ in this equation represents the partial derivative of $\widehat{\mathbf{K}}_{\mathrm{gs}}$ with respect to x and is straightforward to compute by differentiating equation 23.

To complete the algorithm solving for $\mathbf{T}$ and B, we start with an initial guess $\mathbf{T_0}, B_0$ and minimize equation 24 subject to the constraints in equation 20 and the additional constraint that $\|\mathrm{vec}(\mathbf{T}, B) - \mathrm{vec}(\mathbf{T_0}, B_0)\| \leq r$. The scalar r is a trust radius ensuring that the optimization converges, and a sufficiently small value of r guarantees a decrease in the optimization's objective function. This optimization is another SOCP which may solved with the same technique as equation 20.

The trust radius r starts at $r = 1$ and varies dynamically during the course of the optimization. If solving for $\mathbf{T}$ and B then recomputing $\mathbf{R}$ yields a decrease in E, then we update $r' = \max(\rho r, 1)$ and proceed. Otherwise we set $r' = \max(\rho^{-1} r, 1)$ and re-solve. We use the relatively aggressive value $\rho = 10$ as a reflection of the empirical observation that the trust region is rarely necessary, but when it is r normally needs to be relatively small to ensure a decrease in E.