International Journal of Shape Modeling
© World Scientific Publishing Company

Interactive and Linear Material Aware Deformations

Tiberiu Popa
Computer Science Department, University of British Columbia
201-2366 Main Mall Vancouver, B.C. V6T 1Z4, Canada
stpopa@cs.ubc.ca

Dan Julius
Computer Science Department, University of British Columbia
201-2366 Main Mall Vancouver, B.C. V6T 1Z4, Canada
djulius@cs.ubc.ca

Alla Sheffer
Computer Science Department, University of British Columbia
201-2366 Main Mall Vancouver, B.C. V6T 1Z4, Canada
sheffa@cs.ubc.ca

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

Communicated by (xxxxxxxxxx)

Most real world objects consist of non-uniform materials; as a result, during deformation the bending and shearing are distributed non-uniformly and depend on the local stiffness of the material. In the virtual environment there are three prevalent approaches to model deformation: purely geometric, physically driven, and skeleton based.

This paper proposes a new approach to model deformation that incorporates non-uniform materials into the geometric deformation framework. Our approach provides a simple and intuitive method to control the distribution of the bending and shearing throughout the model according to the local material stiffness. It also provides a rich, flexible and intuitive user interface. Thus, we are able to generate realistic looking, material-aware deformations at interactive rates. Our method works on all types of models, including models with continuous stiffness gradation and non-articulated models such as cloth. The material stiffness across the surface can be specified by the user with an intuitive paint-like interface or it can be learned from a sequence of sample deformations.

Keywords: mesh deformation; modeling; geometry processing; material properties

1991 Mathematics Subject Classification: 22E46, 53C35, 57S20

1. Introduction

Mesh deformation is an important task in the modeling and the animation of digital models for computer graphics. Since most real world objects are made up of

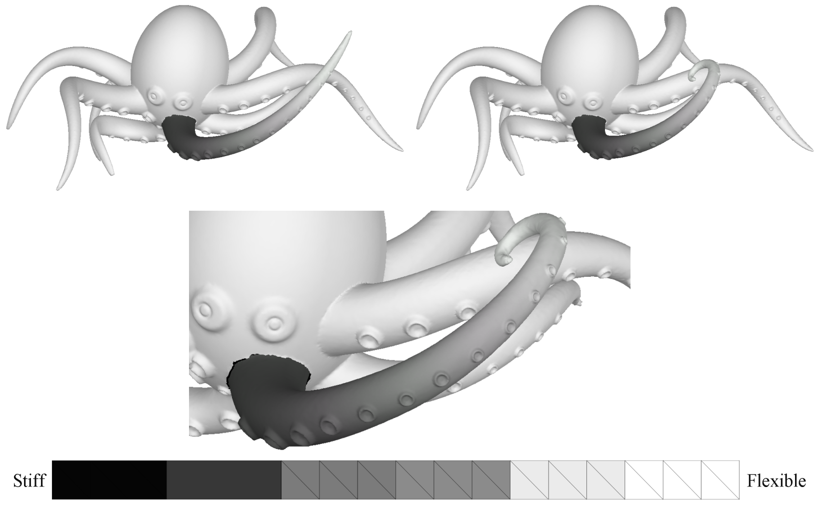

Fig. 1. Material-aware deformation. The stiffness of the tentacle is set to be proportional to its girth (left) resulting in a spiral like shape (right and bottom).

non-uniform materials, their behavior during deformation varies across the surface depending on the local material properties. Ideally, a mesh deformation tool should satisfy the following key requirements: physical plausibility of the results, ease of control, efficiency, and high degree of automation. Existing deformation approaches typically satisfy only a sub-set of these requirements. For example, physics based methods provide accurate model behavior, but they are often not intuitive to control and are usually relatively slow. Skeleton deformations are simple to control and can be implemented efficiently, but their range is typically limited to only a subset of models. Purely geometric deformation techniques are general, efficient and intuitive to control; however they usually ignore the properties of the underlying materials, and thus make it difficult to generate physically plausible deformations.

In this paper we introduce material-aware mesh deformation, a novel technique that uses material properties to guide geometric deformations. We use these properties to characterize the stiffness of the surface, and hence to provide continuous fine control of the surface behavior during deformation, while maintaining the efficiency, simplicity and control specific to geometric methods. The stiffnesses with respect to bending and shearing are represented as scalar fields over the surface. We use these scalar fields within the geometric deformation framework to distribute the deformation according to the local material properties to yield realistic-looking results (Figure 1). Often materials may exhibit anisotropic stiffness, for instance articulated models often have joints with only one degree of freedom. We support such anisotropic behavior by allowing three different scalar fields for the three orthogonal axes of rotation. We are the first, to our knowledge, to support this feature.

To control the deformation, users can specify the material properties using an intuitive paint-like interface. By simply marking a horse's head as stiff (Figure 2(c)), we direct the deformation to the neck of the horse and achieve more realistic results than in Figure 2(b) where the deformation is distributed uniformly. Although some existing geometric methods are capable of achieving similar results, typically they require more user effort to guide the deformation.

In many situations, physically or anatomically correct deformation samples of a given model may be available. In such cases our method can automatically learn the material properties from the sample set, thereby allowing users to create new deformations which are consistent with the sample set. Each of the deformed sample poses contains implicit knowledge of a subset of the material properties. By combining the information from all samples, we are able to reconstruct the scalar fields across the surface. For additional control, we also allow users to refine the acquired fields in specific areas of interest where the desired behavior differs from that of the sample poses.

A simple and intuitive modeling metaphor is an important feature of any mesh deformation tool. We provide a flexible and intuitive user interface that allows the user to choose between simplicity and control. The user can choose between a simple drag and drop interface to position vertices, and a more sophisticated user interface where the user has more control over the deformation behavior.

Our main contribution is the introduction of a compact representation for the local stiffness of a surface, and the integration of this material stiffness into the geometric deformation framework. We extend our previous work [1] by introducing a new mechanism which provides the user with a more flexible and intuitive user interface without increasing the asymptotic complexity of the formulation. Our method is linear, it is simple to use and control, and it creates realistic looking deformations as discussed in the results section.

The rest of this paper is organized as follows: Section 2 reviews previous work on deformation techniques. Sections 3 and 4 describe our deformation algorithm. Section 5 extends the formulation to allow positional constraints on vertices. Section 6 explains how we extend the method to support anisotropic behavior. Section 7 describes how material properties may be learned by example. Section 8 presents some example results. Finally, Section 9 summarizes our work.

2. Previous work

Researchers have addressed the problems of mesh editing and deformation for over twenty years, creating an impressive body of literature and generating several distinct approaches to the problem. One of the first, yet still actively researched mesh deformation frameworks is that of space warping deformations [2,3,4,5]. Since space deformation techniques transform the underlying space, rather than the vertices themselves, it is not possible to incorporate model specific properties such as materials into these techniques.

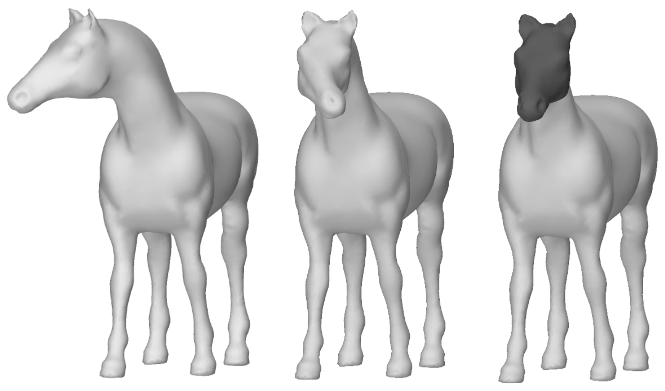

Fig. 2. Turning a horse's head: (a) original model; (b) deformation using uniform material; (c) material-aware deformation using two degrees of stiffness.

Another common approach is to use physical simulation methods [6,7,8,9]. These methods naturally incorporate the material properties and provide physically accurate behavior; however, they are often computationally intensive, and since their control parameters are typically derived from physical equations, they lack intuitive means of controlling the results.

For articulated models, it is common to use a skeleton in order to simplify the task of defining the deformation [10,11,12]. Most skeleton based deformation methods do not generalize to non-articulated models and often provide only binary gradation of stiffness, thus limiting the type of deformations created. In contrast, our method is not restricted to a particular type of models and supports continuous control over the stiffness of the mesh providing finer control of the deformation.

Geometric deformation techniques [13,14,15,16,17,18,19,20,21,22,23,24,25] that operate directly on the meshes have become increasingly popular in recent years. These methods are both efficient and intuitive to control. However, existing geometric techniques do not capture the material properties of the models . Our method uses a geometric deformation approach but introduces material awareness, into the framework to provide greater physical plausibility. The geometric deformation methods most related to our work are those of Yu et al. [17], Zayer et al. [21] and Igarashi and Moskovitc [18]. Yu et al. [17] perform 3D mesh deformation by means of gradient manipulation. First, the positions of some *anchor* vertices are manually modified by the user. Next, the resulting local triangle transformations are propagated to the rest of the mesh according to geodesic distances. Finally, the new vertex positions are computed using the Poisson equation. Zayer et al. [21] show that propagation of the transformations according to geodesic distances is sub-optimal and suggest using harmonic fields as an alternative. Neither method considers material properties in their formulation. Igarashi and Moskovitc [18] deform 2D meshes using a formulation based on an earlier morphing technique [26]. They manipulate the triangles independently and then compute common vertex positions. They show early research results for using material stiffness to control the deformation. A direct extension of their method to 3D would require a volumetric mesh, thus they acknowledge that such an extension may be difficult.

It is often tedious and difficult to define the exact physical properties of an object. One alternative, presented by two recent techniques [28,29], is to create realistic-looking deformations by mimicking existing physically correct example deformations. James and Twigg [28] automatically deduce the skeleton of an articulated model from a sample set of deformed models. Using the estimated skeleton and estimated blending weights they are able to create new deformations consistent with the sample set. Sumner et al. [29] use the set of sample models to create feature vectors that span the space of meaningful deformations. Using our method, we are able to use a set of sample poses as a source for automatically learning the stiffness of the mesh. The learned stiffness is used to create new poses consistent with the samples. In our setting the material properties are derived explicitly, therefore it is very easy for artists to modify and refine those if desired.

Most of the geometry deformation techniques that have linear formulations [14,15,20,21] are limited to a modeling metaphor that requires the user to provide rotational as well as positional constraints to obtain natural looking results. In our method, the user still has the choice of specifying both positional constraints and rotational constraints. But the user also has the choice to only specify positional constraints, in which case our system will compute appropriate rotations for the triangles in the mesh. The user can chose among several criteria to compute these rotations depending on the desired behavior. This way we provide a richer and more flexible front-end to our formulation. Moreover, the method in which the system computes the rotational component is generic and, therefore, can be applied to other methods that suffer from this limitation.

3. Method overview

We present a two-step method for 3D mesh deformation that takes into account the intrinsic material properties of the model. To generate the deformation users select a small set of triangles, called *anchor* triangles (Figure 3(a)) and apply the desired transformations using a click and drag motion. We support anchor transformations that include any combination of rotations and uniform scales. We then calculate transformations for the remaining triangles of the mesh based on the anchor transformations. The calculation takes into account the material properties of the model which are described as follows.

Material properties — To describe the impact of the material on the deformation we define the stiffness of the material with respect to bending and shearing. These stiffnesses are described by separate scalar fields defined across the mesh. Since materials often bend differently with respect to different directions, we also

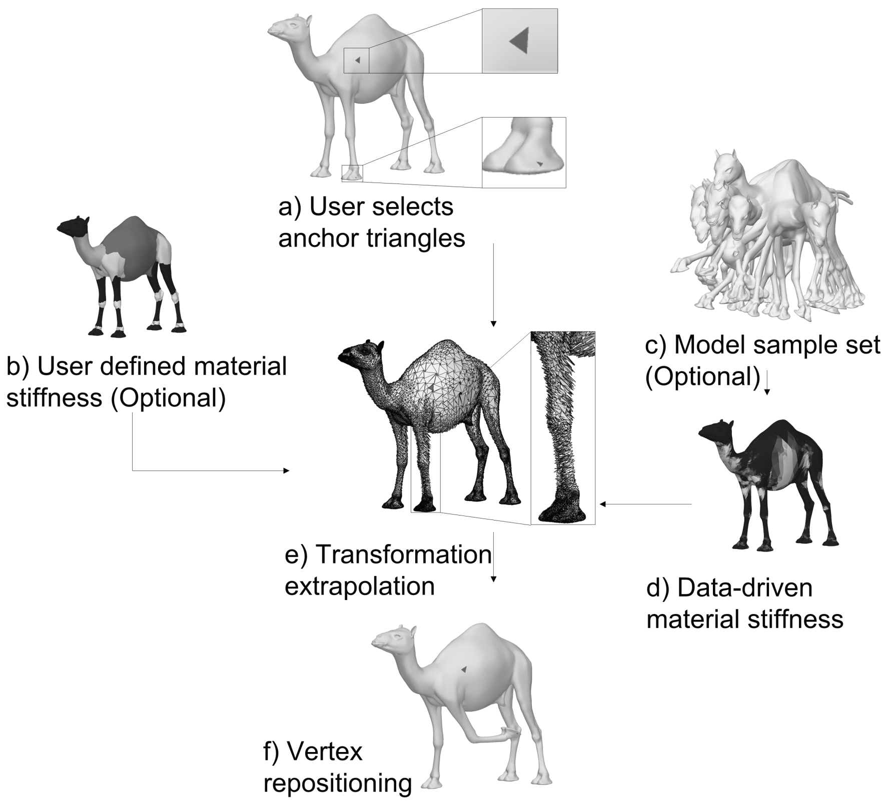
a) User selects anchor triangles
b) User defined material stiffness (Optional)
c) Model sample set (Optional)
d) Data-driven material stiffness
e) Transformation extrapolation
f) Vertex repositioning

Fig. 3. Algorithm flow.

allow users to define anisotropic bending stiffness fields. The user can define these scalar fields using a paintbrush-like tool (Figure 3(b)).

We also introduce a data-driven approach for defining the stiffnesses by automatically learning them from a set of example deformations (Figure 3(c)). By examining the shearing of each individual triangle and the difference in the transformations undergone by adjacent triangles within the entire sample set, we are able to identify degrees of stiffness and flexibility across the mesh, and thus reconstruct the stiffness fields (Figure 3(d)).

Next, we explain how these stiffness scalar fields, together with the user defined transformations at anchor triangles, are used in our algorithm.

Transformation extrapolation — The first step of the algorithm is to propagate automatically the transformations from the anchors to the remaining triangles in the mesh. Finding optimal transformations for each triangle is non-trivial since these must comply with a number of constraints. First, the transformations must be continuous across the surface to yield a smooth looking deformation. Next, the transformations must also be as-rigid-as-possible in order to maintain the details of the original surface [17]. Finally, in our setting we add one new constraint — the transformations must be consistent with the material properties. This final requirement is the one that ensures our deformations behave as desired.

We suggest that each triangle transformation be a weighted sum, or blend, of anchor transformations. Thus, our challenge is to find appropriate weights for blending that comply with the previous requirements. We formulate this as a linear optimization problem where the variables are the blending weights. The stiffness fields are introduced into the formulation to ensure that the deformation is distributed correctly throughout the model. Since the solution depends only on the selection of anchors, we need to solve the resulting system only once. To perform the actual blending we use the transformation algebra defined by Alexa [30]. For a discussions on the optimality of this method, see Appendix A

Vertex repositioning — It is easy to see that applying the resulting transformations to each of the triangles in the mesh independently will break the mesh connectivity, since adjacent triangles are not necessarily assigned identical transformations (Figure 3(e)). Therefore we apply a second step in which we calculate optimal vertex positions such that each triangle is transformed as closely as possible, in the least squares sense, to the previously calculated transformations (Figure 3(f)). We incorporate the shearing stiffness field into the formulation to ensure that most of the resulting shearing is concentrated in the flexible areas of the mesh.

Figure 3 summarizes our algorithm, and the following three sections describe it in detail: Section 4 explains how transformations are propagated and then optimal vertex positions are found, Section 6 explains how the method is extended to support anisotropic stiffnesses, and Section 7 explains how the material stiffnesses are estimated from sample deformations.

4. Method details

We begin this section by describing exactly how our material properties are defined (Section 4.1). Next, we define the gradient transformations and explain how these are propagated from anchor triangles (Section 4.2). Finally, we explain how optimal vertex positions are found (Section 4.3).

4.1. *Material properties*

We formulate our material properties in terms of material resistance to bending and shearing. This resistance is described by bending and shearing stiffness scalar fields defined across the mesh. This approach allows a high degree of control with smooth variations of stiffness across the mesh.

- The bending stiffness is associated with the mesh edges, reflecting the bending flexibility of each edge. Thus the bending scalar field is defined by a set of values φ_{ij} defined on the mesh edges (i, j). This field is used to propagate

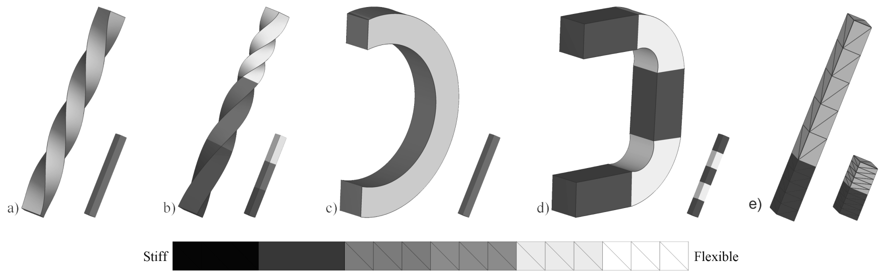

Fig. 4. Twisting, bending, and stretching a bar using two anchor triangles: (a), (c) uniform material; (b), (d), (e) non-uniform material.

the anchor transformations across the mesh (Equation 1).

- The shearing stiffness is associated with the mesh faces reflecting the resistance to shearing of each individual face. The shearing stiffnesses ψ_i defined for the mesh faces are used to find optimal vertex positions (Equation 3).

The bending stiffness often depends on the rotation axis. For example, articulated models often have joints with a limited number of axes of rotation. Therefore, we support anisotropic stiffness by specifying separate bending stiffness fields for three orthogonal axes of rotation.

We support both user-driven and data-driven methods for defining the material properties. In the user-driven setting we provide a simple paintbrush like tool to define the different degrees of stiffness. The users can paint two separate fields for bending and shearing. To simplify the interface, since often the two are linked, we allow the user to specify only one field and derive the other one from it. For instance, if the user specifies the shearing stiffness ψ_i, we obtain the bending stiffness by simply setting $\varphi_{ij} = (\psi_i + \psi_j)/2$. We use this simplified interface in most of our examples. Figure 4 demonstrates painting areas with different degrees of stiffness on a 3D bar and the resulting deformations.

For the data-driven approach, we estimate the material properties from a sample set of deformations automatically. Additionally, our formulation naturally supports user-intervention after the data-driven material estimation step. This property is important in a production setting where animators require simple user controls to fine-tune automatically generated results.

4.2. *Transformation extrapolation*

After the material properties and anchor transformations are defined we are ready to create the deformation. The entire mesh deformation can be expressed as a set of affine transformations $(Ax + b)$ of local coordinate frames defined per triangle. The deformation gradient of each transformation is the matrix A, which encapsulates the triangle transformation up to the translational component. Since the three vertices of a triangle do not determine a local frame, we augment the three vertex positions with a fourth point found by offsetting one of the vertices by the triangle normal [31]. Labeling the vertices of a given triangle as v_1, v_2 and v_3 and the additional vertex as v_4, the three vectors that define the local coordinate frame are $(v_4 - v_1, v_4 - v_2, v_4 - v_3)$.

After the user specifies the transformations for each of the anchor triangles the first step of our algorithm propagates the deformation gradients defined at the anchor triangles to the remaining triangles of the mesh. We achieve this by using a weighted blending of anchor transformations. As previously noted, our challenge is to find appropriate weights for this blending, subject to the material properties. In our setting, this implies that triangles in areas which are more resistant to bending should be assigned more similar transformations. In the isotropic setting, we formulate this as a linear optimization problem:

$$\min_{\omega_i \in R^k} \sum_{(i,j)\in E} \varphi_{ij} \parallel \omega_i - \omega_j \parallel_2^2, \tag{1}$$

where the unknowns are $\omega_i \in R^k$ the blending weights for face i. Each ω_i is a vector $(\omega_i^1, \omega_i^2, \ldots, \omega_i^k)$ where ω_i^a denotes the relative influence of anchor transformation a. k is the number of anchor triangles, E is the set of edges (excluding edges shared by two anchor triangles and boundary edges) and φ_{ij} are the bending stiffness values associated with each edge. The anisotropic setting is slightly different, and is explained in Section 6.

When φ_{ij} is large, ω_i and ω_j will have similar values; thus, the resulting transformations of the two adjacent triangles i and j will also be similar, and the mesh may be considered as locally stiff. Similarly, the converse argument can be made for small φ_{ij}.

Note that in this formulation, the weights ω_i depend only on the connectivity of the mesh and on the selection of anchor triangles. Therefore, the weights need to be computed only once per selection of anchors. Also, note that, since our anchor coefficients have exactly one non-zero entry, our weights are barycentric coordinates with respect to the anchors. As a result our method does not suffer from propagation problems noted by Zayer [21] encountered when using weights based on geodesic distances [17]. This formulation resembles that of [21], however in our formulation we added non-uniform stiffness support.

In order to perform the actual blending we use the commutative transformation matrix algebra defined by Alexa [30]. By defining two new operations denoted by $\oplus$ and $\odot$, the blended transformations T_i with weights ω^i are computed as:

$$T_i = \bigoplus_{a=1}^{k} \omega_i^a \odot T_a. \tag{2}$$

Details on these operators are found in the Appendix. Figure 3(e) illustrates the propagated transformations between two anchors on the camel's leg.

4.3. *Vertex repositioning*

Since the transformations obtained for adjacent triangles are typically not identical, applying each of the transformations as-is would result in ambiguous positions for the two shared vertices. Therefore, we need a second stage to compute the optimal position for each vertex.

Optimal vertex positions are found such that the gradient transformation of each triangle remains as close as possible, in the least squares sense, to the previously calculated transformation [31]. Since the process results in triangle shearing, we introduce the shearing stiffness into the formulation, to direct the distortion according to the flexibility of the mesh.

Sumner and Popovic [31] show that the transformation gradients can be expressed in terms of the vertex positions before and after the deformation:

$$A = \tilde{V}V^{-1},$$

where

$$\tilde{V} = (\tilde{v}_4 - \tilde{v}_1, \tilde{v}_4 - \tilde{v}_2, \tilde{v}_4 - \tilde{v}_3)\,,$$

$$V = (v_4 - v_1, v_4 - v_2, v_4 - v_3)\,,$$

and $\tilde{v}_i$ is the position of vertex v_i after applying the deformation transformation. Next, the system is reformulated such that the unknowns are the new vertex positions $\tilde{v}$.

The optimal solution is obtained when the resulting triangle transformations are as close as possible to the previously computed T_i's. Since the T_i's are blends of the original anchor transformations, they contain no shearing. Thus, the closer the final and the original transformations are, the smaller the triangle shearing. To account for the shearing stiffness we incorporated ψ_i into the formulation:

$$\min_{\tilde{v}} \sum_{i}^{n-k} \psi_i \|\tilde{V}_i V_i^{-1} - T_i\|_F^2, \tag{3}$$

where V_i and $\tilde{V}_i$ are the local frames before and after applying the deformation and T_i are the previously calculated transformations. We reformulate this as a linear optimization problem:

$$\min_{\tilde{v}} \|\Psi(A\tilde{v} - t)\|_2^2, \tag{4}$$

where A is a sparse matrix constructed using the pre-deformation local frames V, t is a vector composed of all the elements in T_i and Ψ is a diagonal matrix composed of ψ_i.

Large ψ_i will result in transformations which are very close to the originally computed T_i, and thus exhibit less shearing. Small ψ_i will allow for more shearing to take place. Figure 4(e) shows an example of stretching a bar with non-uniform shearing stiffness. As expected, most of the shearing occurs in the flexible region.

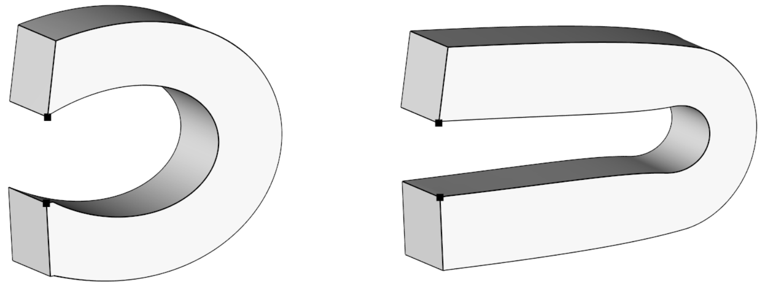

Fig. 5. Deformation of a bar specifying only positional constraints. The optimal rotations are computed automatically. (a) uniform materials (b) non-uniform materials

The solution of the system in Equation 4 provides a new position for each of the vertices up to a global translation of the model. To anchor the model in place, we fix the position of a single vertex by removing the corresponding variable and pre-multiplying its known position with the appropriate elements of A into the vector T [29]. In fact, multiple vertices may be set at fixed positions to apply boundary constraints such as regions of influence.

5. Positional Constraints

The formulation presented so far allows the user to only specify rotational constraints for the anchor triangles. In certain cases the user might want to specify positional constraints as well. Positional constraints on vertices can be added by augmenting the second stage of our system (equation 4) by adding more rows to the constraint matrix A to keep the anchor vertices in place. In this setup, the user has to specify positional and rotational constraints simultaneously to achieve the desired results which is less intuitive than a standard drag and drop interface. Most of the deformation methods that use linear formulations [14,15,20,21] also face the same limitation. Non-linear methods [16,24,25] that employ rotational invariant local coordinates do not have this problem, but non-linear formulations are in general less efficient.

In this work we develop a method where the user can specify only positional constraints on vertices using a drag and drop interface. The system then computes the optimal rotations automatically. Figure 5 shows an example where a straight bar is deformed by specifying a new position for one vertex while keeping the other vertex fixed. Note that the algorithm provided an intuitive rotation for both the uniform and non-uniform case yielding a natural looking result.

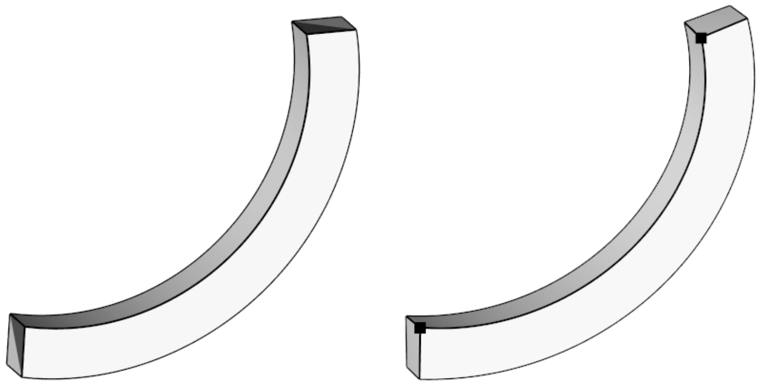

Fig. 6. Left: rotation of the bar using only rotational constraints. Right: rotation of the bar by dragging one anchor and using our method to determine the rotation.

5.1. *Finding Rotations*

We now describe more formally our method for computing the rotations For each user selected anchor vertex v_i we associate an adjacent anchor triangle t_i. As the user changes the position of v_i, the system computes the optimal rotation R_i for the associated anchor triangle t_i. It than propagates the rotations to all the triangles in the mesh as described in section 4.2. After that, it finds vertex positions as described in section 4.3 with the amendment that additional constraints are added in the constraint matrix A of Equation 4 for all anchor vertices v_i.

Theoretically, since a 3D rotation has 3 degrees of freedom we would need to solve for 3 new variables. However, we observe that in a typical drag and drop interface, an anchor vertex is moved on a plane parallel to the projection plane. This means that the anticipated rotation will typically be around the normal of the projection plane thus reducing the problem to only one degree of freedom. Details on how to solve for this latest degree of freedom are given in section 5.2. Figure 6 compares the results of a deformation using only rotational constraints and a deformation using our positional constraints. Note that in figure 6(b) the system found automatically the appropriate angle.

We choose as the optimal rotation angle for the anchor triangle, the angle that yields a result that best preserves certain properties (functionals) between the original object and the deformed object. We experimented with several such functionals:

- *Area preserving.* Minimize $\sum_{i \in T}(\psi_i * (A_i - A'_i))^2$ where A_i and A'_i are the areas of the original triangles and the triangles after deformations, respectively. ψ_i is the face stiffness.
- *Volume preserving.* Minimize the variation in total volume. The total volume is estimated using the standard method of summation over the signed tetrahedra volumes obtained by each face with an arbitrary reference point

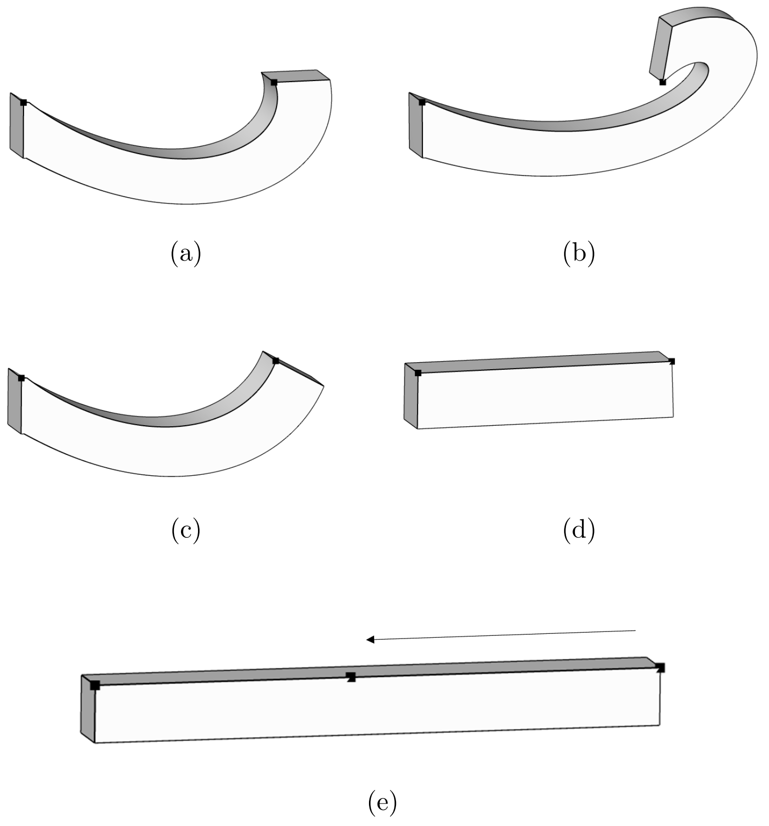

Fig. 7. Comparative results for various energies used: (a) area, (b) volume, (c) shearing (d) dihedral angle

P.

- *Shape preserving.* Minimize $\sum_{i \in T}(psi_i * s_i)^2$ where s_i is a measures the shearing between the original and the deformed triangles weighted by the stiffness of each triangle. We computed the shearing by first obtaining the 3×3 transformation matrix that transforms the frame of one triangle onto the other and we obtain the shearing coefficients from the polar decomposition.
- *Angle preserving.* Minimize the variation in bending between adjacent triangles: minimize $\sum_{(i,j) \in E}(\varphi_{ij} * (\alpha_{ij} - \alpha'_{ij}))^2$ where α_{ij} and α'_{ij} are the dihedral angle of the faces corresponding to edge (i, j) on the original mesh and deformed mesh respectively, and φ_{ij} is the edge stiffness coefficient of edge (i, j).

Figure 7 shows a comparative result of bending a bar using the different functionals. The deformations in figure 7(a-d) are done by keeping an anchor and its

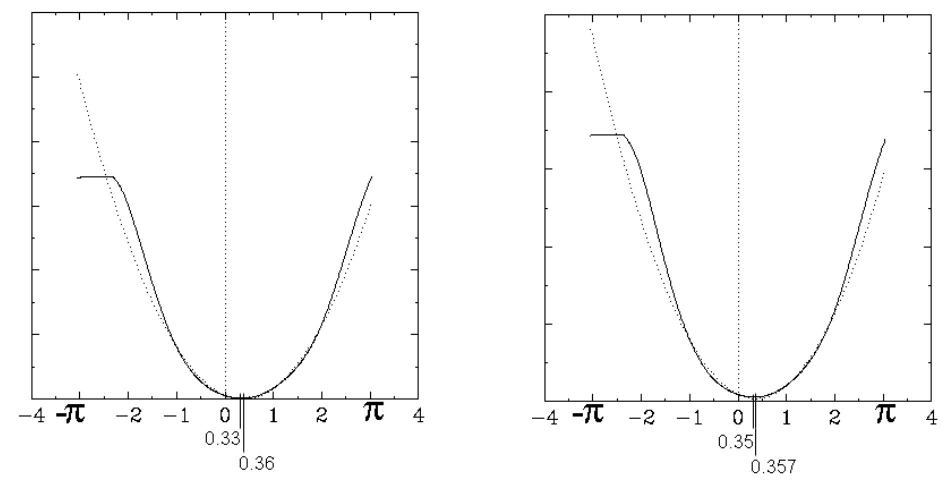

Fig. 8. The two figures show two plots of the error function against the angle of rotation of the moving anchor. The dotted line represents the parabola that we are trying to fit to the data. The two numbers at the bottom of each figure represent the true solution, and the solution computed using the parabolic strategy respectively.

associated anchor triangle in place and pushing the other towards it as illustrated in figure 7(e).

The *area preserving* and *shape preserving* functionals are in the spirit of the "as rigid as possible" deformations. In both cases we are preserving the shapes of the triangles in the least square sense, giving more priority to stiffer triangles. As shown in Figure 7, the *area preserving* functional allows for more bending yielding more natural looking results. In addition to that, the *area preserving* is also cheaper to evaluate than the *shape preservation* functional that requires a polar decomposition step for every triangle in the mesh. The *volume preserving* functional finds the rotation in the plane that makes the change in volume minimal. Hence it has to over-rotate in order to add more volume. Note that this functional minimizes the variation in total volume, but it does not guarantee that the volume is preserved exactly. The *angle preserving* functional collapses the bar since a zero rotation of the anchor triangle best preserves the dihedral angles. This functional can be used to simulate plastic deformations like clay. Note that affine combinations of these functionals can be used to get intermediate effects. In practice the *area preserving* functional yields the most natural results and it is also the most numerically stable, so this is the function that we used for all the other examples that use positional constraints.

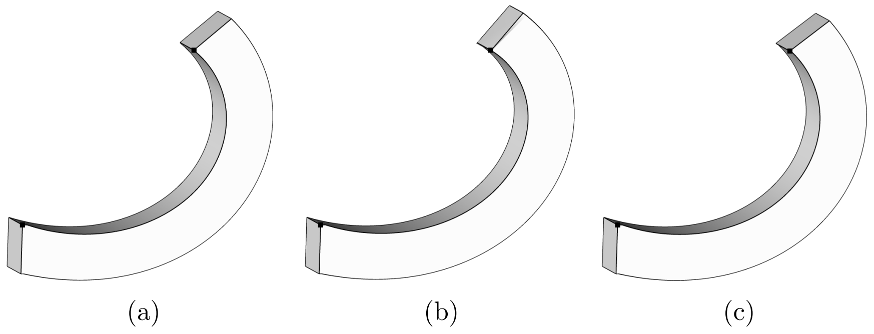

Fig. 9. Comparative results for the 3 solvers: (a) brute force (b) parabolic solver (c) binary search solver

5.2. *Solving*

Since we are looking for an angle, a brute-force approach to solving this problem is to densely sample the interval $[-\pi, \pi]$ and find which angle gives us optimal results. Moreover, we note that as the user drags the mouse on the screen, the angle change from one evaluation to another is relatively small. This shortens the range of values making the search more efficient. In practice, we observed that the $[-1, 1]$ range is usually sufficient. One evaluation of the functional requires solving the two systems described by equations 1 and 4. Since the two matrices associated with these systems need to be inverted only once, even the brute-force approach to finding the optimal angle yields interactive rates for small models. For larger models we devised two strategies that significantly speed up the process: a binary search strategy and a quadratic fit strategy.

In the binary search case, we estimate the derivative of the function using forward differences and we search for the root of the derivative function using a binary search strategy. Note that the search interval is so small that a more sophisticated Newton method would not significantly improve the convergence. Binary search takes usually around 6 iterations to converge, and since it takes two evaluations to compute the derivatives it takes around 12 evaluations.

While the binary search strategy works well in practice, we can speed up the method even more using the observation that in most cases the function has only one local minimum and the shape of the curve looks very much like a parabola as illustrated in figure 8. Therefore, we can find the rotation angle by fitting a parabola to the data and choosing the tip of the parabola as the minimum point. The difference between the estimated solution and the true solution is within a few degrees. This approach always requires a constant number of evaluations, namely 4. Three evaluations are required to find the equation of the parabola and the fourth is the evaluation that gives us the final solution. Using this strategy, the solution is still assymptotically linear with a constant factor of 4. The quadratic fit formulation, while faster, tends to be less stable than the binary search or brut

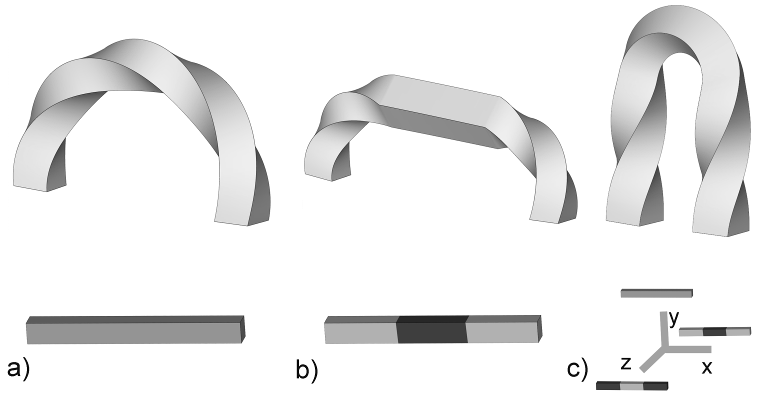

Fig. 10. Bending and twisting a bar with different material properties and a single transformation applied at the two end faces. (a) Uniform material. (b) Center region is isotropically more stiff. (c) Anisotropic stiffness - allowing the center region to bend around the z axis, and the side regions to twist around the x axis.

force methods in cases where the true solution is close to $-\pi$ or to π. Figure 9 shows a comparison of the solvers.

6. Anisotropic materials

The formulation presented above uses a single scalar bending stiffness field. This formulation assumes that bending flexibility is a non-directional property, which need not be true in practice. For example, a knee joint is only flexible in one direction and rigid in the other two. In this section we extend our method to allow different bending stiffnesses for different axes of rotation.

Instead of using one bending stiffness value per edge, we define three different values corresponding to rotations around three orthogonal axes x, y and z. To apply those stiffnesses when blending anchor transformations, we first need to decompose the anchor transformations into a scaling transformation and rotation transformations around the three axes x, y and z.

Using the polar decomposition $T_a = S_a Q_a$ such that Q_a is a rotation transformation [27], we identify the rotational component of each anchor transformation. Using the transformation matrix algebra described by Alexa [30], three matrices R_x^θ, R_y^θ, R_z^θ denoting rotations by an angle $\theta \in [0, \pi]$ around three orthogonal axes x, y, and z form a basis of the sub-space of rotations. Thus, for any given rotation matrix Q we can find a commutative decomposition such that

$$Q = c_x \odot R_x^\theta \oplus c_y \odot R_y^\theta \oplus c_z \odot R_z^\theta.$$

The coefficient c_x is found by simply computing the inner product

$$c_x = < log(Q), log(R_x^\theta) >,$$

where the inner product of transformation matrices is the sum of products of the corresponding matrix entries
$< A, B >= \sum a_{ij} b_{ij}$. c_y and c_z are found in a similar manner.

By defining $X_a = c_x^a \odot R_x^\theta$, $Y_a = c_y^a \odot R_y^\theta$ and $Z_a = c_z^a \odot R_z^\theta$ we get a unique decomposition of each anchor transformation T_a into a scaling matrix S_a and three rotation matrices X_a, Y_a and Z_a around the three orthogonal axes of rotation x, y and z.

To compute the new blending weights and transformations we solve Equation 1 separately for each axis of rotation using the appropriate bending stiffness to get three weight vectors ω_i^x, ω_i^y and ω_i^z per triangle. In the isotropic case, we used ω_i as weights for blending the anchor transformations. In the anisotropic setting we find the local triangle transformations T_i by combining the per-axis transformations. We rewrite Equation 2 as

$$T_i = \left(\bigoplus_{a=1}^{k} \tilde{\omega}_i^a \odot S_a \right) \left(\bigoplus_{a=1}^{k} \omega_i^{ax} \odot X_a \oplus \omega_i^{ay} \odot Y_a \oplus \omega_i^{az} \odot Z_a \right)$$

where T_i are the propagated transformations,$\omega_i^{ax}, \omega_i^{ay}$ and ω_i^{az} are the blending weights, $\tilde{\omega}_i^a$ is their average, and S_a, X_a, Y_a and Z_a are decompositions of the anchor triangle transformations T_a such that $T_a = S_a \left(X_a \oplus Y_a \oplus Z_a \right)$.

Since we only define anisotropic stiffnesses for bending the rest of the algorithm continues as previously described. Figure 10 shows the impact of using anisotropic stiffness fields when bending and twisting a 3D bar. First, we used a single material with uniform stiffness and applied two rotations around the x and z axes at one end of the bar while anchoring the other end. This resulted in a uniform deformation (Figure 10(a)). Next, we changed the model's material, specifying the center region as isotropically stiff and the sides as isotropically flexible. We then applied the same transformation. The result is shown in Figure 10(b) where the center region remains rigid. Finally, Figure 10(c) illustrates an anisotropic deformation. The center region material was defined to be stiff when rotating around the x axis, and flexible when rotating around the z axis. The side region materials were defined to deform in the exact opposite manner. The resulting deformation exhibits twisting (rotation around the x axis) only in the side regions, and bending (rotation around the z axis) only in the center region.

7. Material learning

In many situations, physically or anatomically correct deformation samples of a given model may already be available. In such cases we would like to derive the material properties from the sample poses to create new deformations which are consistent with the sample set and are therefore also correct. The following section describes our method of automatically acquiring the material properties from sample deformations.

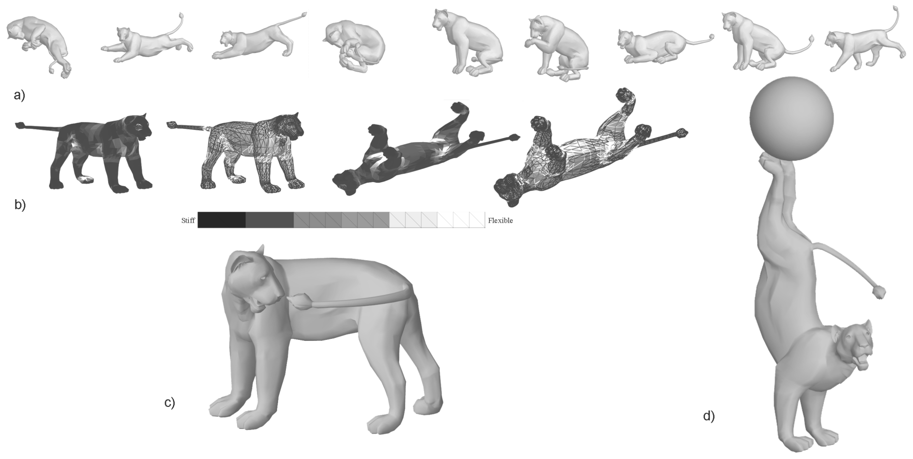

Fig. 11. Estimation of material stiffness from sample poses. (a) Sample poses. (b) Estimated bending and shearing stiffnesses visualized on the edges and faces respectively. (c), (d) New poses created using the estimated stiffnesses: (c) chasing the tail (4 anchors); (d) handstand (8 anchors).

Given a reference mesh P_0 with m triangles and a set of deformed meshes with the same connectivity P_s $s = 1, \ldots, l$, we first compute the deformation gradient for each of the triangles $i = 1, \ldots, m$ in P_s as follows:

$$T_{si} = \tilde{V}_{si} V_{0i}^{-1} \quad s = 1, \ldots, l, i = 1, \ldots, m,$$

where V and $\tilde{V}$ are the local frames in the reference and deformed meshes.

Next, we decompose each deformation gradient matrix using the polar decomposition $T_{si} = S_{si} Q_{si}$ such that Q_{si} is a rotation transformation, and S_{si} is a combination of scaling and shearing [27,28,29]. We use this decomposition to estimate the bending and shearing stiffness across the mesh.

The amount of bending the model undergoes locally is best reflected by the change in the dihedral angle around a mesh edge, and can be estimated by analyzing the rotations Q_i and Q_j of two triangles sharing the edge. In each example the difference between the two matrices Q_i and Q_j of adjacent triangles is roughly equivalent to the amount of bending around the shared edge. Since each sample only exhibits bending around a subset of the edges we need to combine values from multiple samples. The maximum difference observed for each edge is equivalent to the maximum bending around the edge among all samples and therefore is used to define the bending stiffness. The values are then scaled to be in $[\epsilon, 1]$.

$$\tilde{\varphi}_{ij} = \max_{s=1\ldots l} ||Q_{si} - Q_{sj}||_F^2 \quad (i, j) \in E,$$

$$\varphi_{ij} = 1 - \frac{\tilde{\varphi}_{ij}}{\max\limits_{\acute{i},\acute{j} \in E} \tilde{\varphi}_{\acute{i}\acute{j}} + \acute{\epsilon}} \quad (i,j) \in E.$$

For each face i, we estimate a shearing stiffness ψ_i based on the maximum distortion of the face found in the set of example deformations. The values are then scaled to be in $[\epsilon, 1]$.

$$\tilde{\psi}_i = \max_{s=1\ldots l} \|S_{si} - I\|_F^2 \quad i = 1 \ldots m,$$

$$\psi_i = 1 - \frac{\tilde{\psi}_i}{\max\limits_{\acute{i}=1\ldots m} \tilde{\psi}_{\acute{i}} + \acute{\epsilon}} \quad i = 1, \ldots, m.$$

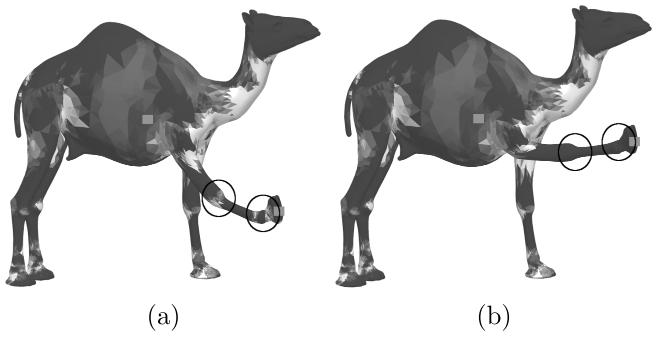

Fig. 12. Refining learned materials. (a) Bending the front leg of a camel using the estimated stiffness and two anchors. (b) Bending the leg after modifying the stiffness by marking both the knee and ankle joints as stiff.

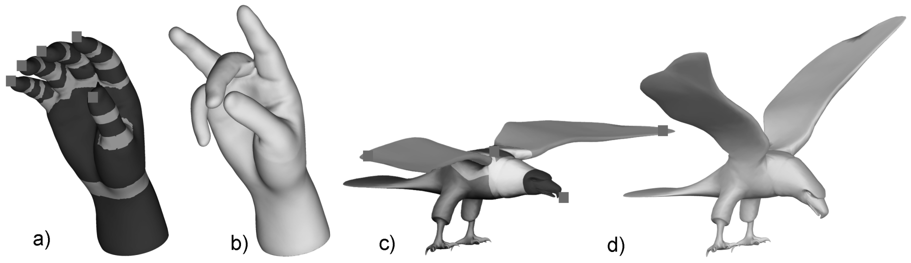

Fig. 13. Deformation of articulated models using our technique: (a), (c) original models and painted stiffnesses; (b), (d) deformation results.

For both scalar fields we found that clamping the bottom 1% of the values before scaling greatly improved the results.

Figure 11 shows an example of estimated bending and shearing stiffnesses learned from a sample sequence of deformations. As shown in the figure we use these as a basis for creating new poses which were not in the sample set.

Our method is much simpler than those of James and Twigg [28] and Sumner et al. [29], who also use polar decomposition as the first step in their methods. However, some problems can occur when stiffness coefficients of different parts of the body are learned independently from different poses. In these cases there is no information on the relative stiffness between these parts and hence the relative scale of the stiffness coefficients may not be correct. Nevertheless, our experimental results appear to correctly capture the model's material properties, and to provide realistic looking deformations. Furthermore, our method exhibits two nice properties: our learning algorithm is linear in the number of sample poses, and the learned stiffnesses can be further refined by the user for finer control.

8. Implementation and results

We now provide some implementational details and discuss some deformations created by our technique. We used Graphite [32] as a framework for implementing our deformation method. Material properties are defined using a simple color map with a paintbrush interface. To deform models users mark anchor triangles and then transform them by dragging the mouse. Note that in all our examples except Figure 4(e) we defined only the rotation and scaling for the anchor triangles without fixing their final positions. We let the algorithm find these optimal positions automatically.

We used the UMFPACK4.4 [33] solver to compute the solutions of equations 1 and 4. In each of the equations the required matrix inversion depends only on the selection of anchor triangles, the mesh connectivity, and the undeformed mesh geometry. We can therefore precompute the inverse of these matrices, allowing the system to work at interactive rates.

We support the concept of deforming only regions of interest, confining all the calculations to triangles within that region. When using such a region of interest, the vertices on its boundary are constrained to remain in their initial position. Our results demonstrate that the method may be applied to a large variety of model types.

Figures 2, 3(f), 13 and 14 demonstrate how a simple coloring scheme for defining the material properties allows us easily to deform articulated models. In the hand example from Figure 13, we simply painted the joints with a flexible material. We used three anchors per finger, one at the tip and two at the base, and then deformed each finger independently using a region of influence to speed up the computation. The eagle deformation in Figure 13 uses simple coloring of the head, neck and wings to illustrate that the head is the most rigid part while the wings are the most flexible.

Figure 14 demonstrates a combination of scaling and rotation applied to the

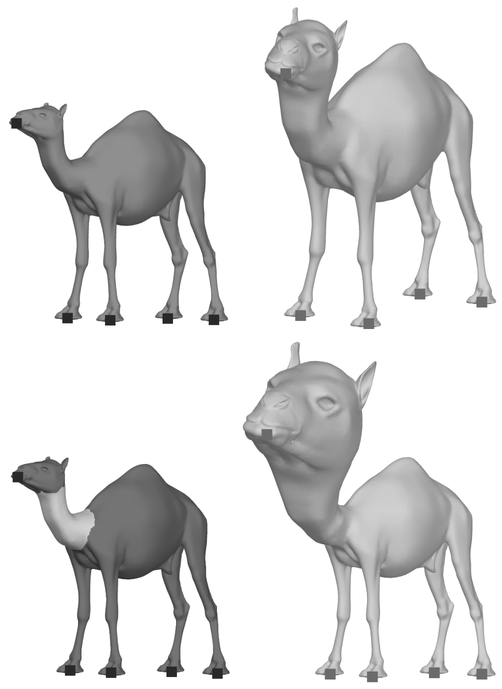

Fig. 14. Scaling and rotating a camel's head. Top: uniform material causes the entire camel to scale and rotate. Bottom: the stiff head scales uniformly, while the flexible neck absorbs the distortion.

head of the camel. The deformation was created using four anchors on the bottom of the feet, and one anchor on the tip of the nose. The feet anchors were kept in place and the anchor triangle on the nose was rotated and scaled to three times the original size. When using uniform materials (Figure 14(top)) the scaling is propagated uniformly through the model. Using different materials we are able to better control the propagation. In Figure 14(bottom), we defined the neck to be more flexible than the head and the body. With the new materials the head scales to three times the original size, the body remains undeformed, and the distortion is mostly concentrated at the neck. Using standard methods it is not possible to archive such deformation in a single operation. Even defining a region of influence to include only the head and neck would not be sufficient, since the head would not deform uniformly.

For many models the stiffness changes smoothly across the mesh. Tree branches, plant stems, and octopus tentacles are examples of models where the flexibility is proportional to the girth of the model. Figure 15 demonstrates how our material-aware deformation framework allows us to define correct bending behavior for an octopus's tentacle. When using a uniform material the tentacle takes the unnatural shape of an arc; however, using non-uniform materials results in the more natural

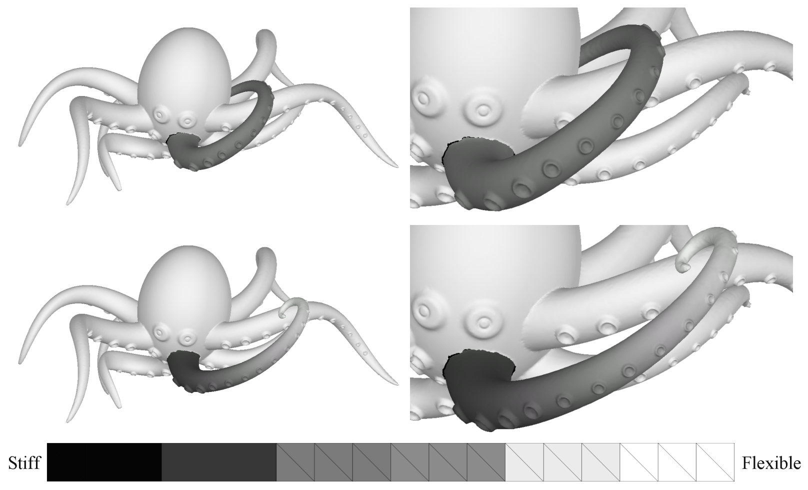

Fig. 15. Deformation comparison for octopus tentacle in figure 1 between uniform stiffness (top) and smooth variation of stiffness (bottom). In both cases we defined the same amount of rotation applied at the tip of the tentacle while the final position of the tip is calculated automatically.

shape of a spiral with the tip bending more than the base. In both deformations we applied the same rotation transformation to an anchor triangle at the tip of the tentacle, while the final position of the tip was computed automatically.

Figure 16 demonstrates the application of our algorithm to a non-articulated model. In order to deform the piece of cloth we used a texture map to define the material properties. The bold letters define very stiff areas, while the rest of the model varies in degrees of flexibility.

We have also tested our method for estimating the stiffness from sample poses. Figures 3(d), 11 and 12 demonstrate our results. For sample poses we used models obtained from [31]. For the camel (Figures 3(d) and 12) the stiffness fields were learned from ten sample poses in 9.5 seconds. For the lion (Figure 11) we used nine sample poses and the learning process took 2.1 seconds. As expected, in both cases the estimated stiffness scalar fields are anatomically correct, showing more flexible regions at the joints and stiffer regions along the bones. Figures 11(c) and (d) show examples of new anatomically correct poses created using the estimated stiffness fields. Figure 12 shows an example of manually modifying the stiffness fields in order to create a desired deformation which does not comply with the sample set. The learned fields allow bending of the leg at the knee and ankle, resulting in an anatomically feasible pose (Figure 12 (a)). To handicap the camel, we manually define these areas as stiff. Applying the same transformation at the tip of the foot causes the leg to raise without bending at the knee or ankle (Figure 12 (b)).

Figure 17 illustrates deformations using only positional constraints as described in section 5 The system automatically computes the appropriate rotation of the head. Since the neck of the camel is more flexible than the rest of the body, most of

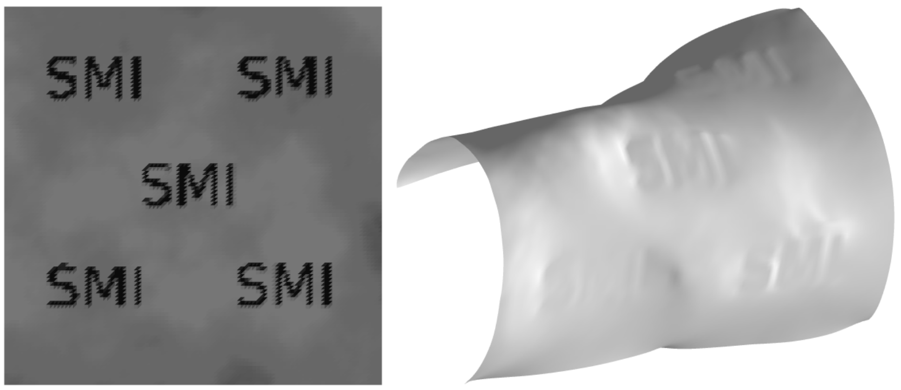

Fig. 16. Material-aware deformation of cloth. The printed letters are made up of stiffer material; the rest of the model consists of flexible material modulated using Perlin noise to create a natural wrinkled look.

the rotation will be at the neck, thus preserving the geometric detail of the head. This mesh has 19,536 faces and it was deformed interactively at a frame rate of 3-4 frames per second using the parabolic solver.

Central to our technique is the ability to capture the surface behavior in a pre-processing stage and encode it in the formulation. We successfully capture the material properties while preserving the simplicity and efficiency common to geometric deformation techniques. We demonstrated that our technique can be applied to a wide range of models producing complex results with only few anchors.

Table 1 summarizes the deformation statistics of the various models used in our results, measured on a 3GHz Intel Pentium IV with 2Gb of RAM.

9. Summary and future work

We presented a new mesh deformation technique that incorporates material properties into the geometric deformation framework. Using these properties, we provide a simple mechanism that allows material-aware behavior of the surface under deformation. Our method combines the efficiency, generality and control of geometric methods together with material awareness found in physical and skeleton based methods.

Material properties can be user-driven, where the stiffnesses are specified with a simple brush-like interface, or data-driven where stiffnesses are deduced from a set of given poses. It also can be a combination of the two where the user can override the learned material stiffnesses.

The formulation is simple and efficient. It requires solving only two linear systems, and thus works at interactive rates. The resulting deformations are as-rigid-as-possible, subject to the material stiffness and the user defined transformations, and therefore maintain the shape details as demonstrated in our results.

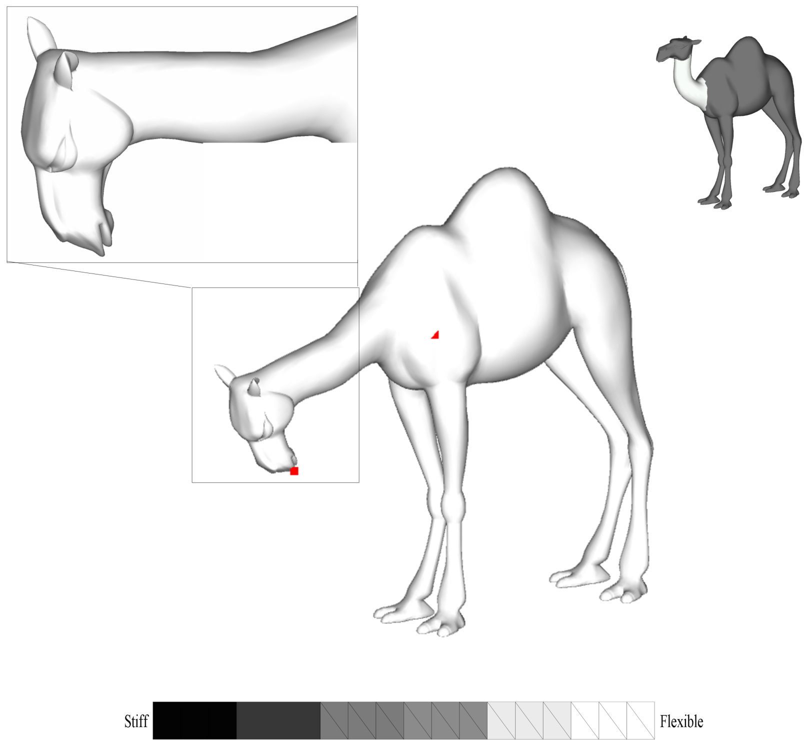

Fig. 17. Deformation of a camel using positional constraints. The neck of the camel is more flexible than the rest of the body. Three anchors were used: two at the back of the camel on each side of the neck and a third one on the head. The example was generated by simply dragging the vertex on the head to its new position while keeping the other two in place. Note that the geometric details of the head are very well preserved.

For future research we would like to improve the anisotropic model. The current anisotropic model uses a global coordinate frame to decompose local rotations instead of local coordinate frames. Decomposition around local coordinate frames would be preferred because the stiffness maps would be invariant to the position and orientation of an object in space. However, the challenge in doing so is the consistent construction of blending weights across different coordinate frames.

Acknowledgments

We would like to thank Ciaran Llachlan Leavitt for help with editing the paper. We would like to thank Robert Sumner for the lion and camel models and ETH Zurich for the octopus model. This project was partially funded by NSERC.

	Bar	Hand	Lion(c)	Lion(d)	Cloth	Horse	Eagle	Octopus	Camel
#Faces	1596	6274	9996	9996	19602	19996	29232	36542	43775
#Anchors	2	3	4	8	10	2	4	5	5
Preprocessing:									
Weight calculation(s)	0.03	0.23	0.51	1.03	2.91	0.63	1.50	2.76	2.95
Vertex repositioning factorization(s)	0.1	0.48	0.75	0.75	1.58	1.81	2.47	3.16	3.63
Total (s)	0.13	0.71	1.26	1.78	4.49	2.44	3.93	5.92	6.58
Real-time:									
Blending (ms)	10	30	30	30	70	60	100	130	150
Repositioning (ms)	10	60	60	60	90	120	140	210	130
Total (ms)	20	90	90	90	160	180	240	340	280

Table 1. Model deformation timings including pre-processing and actual interactive deformation. For the octopus and hand models the number of faces noted is the number of faces in the region of influence.

Appendix A. Appendices

Geometric transformations are typically represented as square matrices. Matrix multiplication is used to compose and apply the transformations. This representation has two key shortcomings:

- Rotation transformations cannot be interpolated by interpolating the matrix elements.
- Matrix multiplication is not commutative.

Both of these properties are crucial for us in order to propagate and later decompose anchor triangles.

To deal with these issues a number of interpolation methods for rotations have been developed over the years. When dealing with rotations there are three desired properties: torque-minimization, constant speed, and commutativity. Currently no interpolation method exhibiting all three properties exists. SLERP [34] exhibits constant speed and minimal-torque. LERP, popularized by Casey Muratori, is commutative and minimal-torque. The exponential map interpolation [35] is commutative and constant speed.

Alexa [30] extended the exponential map interpolation method into a commutative algebra of (almost) general transformations that supports matrix blending and interpolation. This algebra is commutative and interpolates transformations with constant speed. Furthermore, it is not limited to rotations only, thus simplifying the task of dealing with combinations of rotations and scales.

We have chosen to use the later method since it answers both our requirements. We now give a brief overview of the blending operators defined in this algebra.

By defining two new operations denoted by $\oplus$ and by $\odot$ (corresponding to matrix addition and scalar multiplication), the blending of transformations T_a with weights ω_a becomes $\bigoplus \omega_a \odot T_a$.

The two operators are based on matrix *exp* and *log* operators defined as follows:

$$\exp(A) = \sum_{k=0}^{\infty} \frac{A^k}{k!},$$

$$A = \log(X) \Leftrightarrow \exp(A) = X.$$

Alexa [30] shows that this sum is well defined and closed for 3x3 rotation matrices and non-uniform scales under some minimal conditions. The blending formula is defined as follows:

$$\bigoplus_{a=1}^{k} \omega_i^a \odot T_a = \exp(\sum_{a=1}^{k} \omega_i^a \log(T_a)).$$

More details as well as numerical methods to compute these operations are found in [30].

References

1. T. Popa, D. Julius, A. Sheffer, Material Aware Mesh Deformations *Proceedings of the International Conference on Shape Modeling & Applications '06* (2006) 141–152
2. A. H. Barr, Global and local deformations of solid primitives, *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques* (1984) 21–30
3. T. W. Sederberg and S. R. Parry, Free-form deformation of solid geometric models, *SIGGRAPH '86: Proceedings of the 13th annual conference on Computer graphics and interactive techniques* (1986) 151–160
4. A. Angelidis, M.-P. Cani, G. Wyvill and S. A. King, Swirling-Sweepers: Constant-Volume Modeling, *Pacific Conference on Computer Graphics and Applications* (2004) 10–15
5. M. Botsch and L. Kobbelt, Real-Time Shape Editing using Radial Basis Functions, *Computer Graphics Forum, (Eurographics 2005 proceedings)* (2005) 611–621
6. D. Terzopoulos, J. Platt, Alan Barr and Kurt Fleischer, Elastically deformable model, *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques* (1987) 205–214
7. D. Terzopoulos and K. Fleischer, Modeling inelastic deformation: viscolelasticity, plasticity, fracture, *SIGGRAPH '88: Proceedings of the 15th annual conference on Computer graphics and interactive techniques* (1988) 269–278
8. D. L. James and K. Fatahalian, Precomputing interactive dynamic deformable scenes, *SIGGRAPH '03: Proceedings of the 30th annual conference on Computer graphics and interactive techniques* (2003) 879–887
9. M. Muller, B. Heidelberger, M. Teschner and M. Gross, Meshless deformations based on shape matching, *SIGGRAPH '05: Proceedings of the 32th annual conference on Computer graphics and interactive techniques* (2005) 471–478

10. S. Capell, S. Green, B. Curless, T. Duchamp and Z. Popovic, Interactive skeleton-driven dynamic deformations, *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002) 586–593
11. B. Allen, B. Curless and Z. Popović, Articulated body deformation from range scan data, *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002) 612–619
12. S. Yoshizawa, A. G. Belyaev and H.-P. Seidel, Free-form skeleton-driven mesh deformation, *Proceedings of the eighth ACM symposium on Solid modeling and applications* (2003) 247–253.
13. M. Alexa, Local Control for Mesh Morphing, *Proceedings of the International Conference on Shape Modeling & Applications '01* (2001) 209
14. O. Sorkine, D. Cohen-Or, Y. Lipman, M. Alexa, C. Rössl and H.-P. Seidel, Laplacian surface editing, *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing* (2004) 175–184
15. Y. Lipman, O. Sorkine, D. Cohen-Or, D. Levin, C. Rössl and H.-P. Seidel, Differential Coordinates for Interactive Mesh Editing, *Proceedings of Shape Modeling International* (2004) 181–190
16. A. Sheffer and V. Kraevoy, Pyramid Coordinates for Morphing and Deformation *Second International Symposium on 3DPVT* (2004) 68–75
17. Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo and H.Y. Shum, Mesh editing with poisson-based gradient field manipulation, *ACM Trans. Graph.* **23** (2004) 644–651
18. T. Igarashi, T. Moscovich and J. F. Hughes, As-rigid-as-possible shape manipulation *SIGGRAPH '05: Proceedings of the 32th annual conference on Computer graphics and interactive techniques* (2005) 1134–1141
19. K. Zhou, J. Huang, J. Snyder, X. Liu, H. Bao, B. Guo and H.-Y. Shum, Large mesh deformation using the volumetric graph Laplacian, *SIGGRAPH '05: Proceedings of the 32th annual conference on Computer graphics and interactive techniques* (2005) 496–503
20. Y. Lipman, O. Sorkine, D. Levin and D. Cohen-Or, Linear Rotation-invariant Coordinates for Meshes, *SIGGRAPH '05: Proceedings of the 32th annual conference on Computer graphics and interactive techniques* (2005)
21. R. Zayer, C. Rössl, Z. Karni and H.-P. Seidel, Harmonic Guidance for Surface Deformation, *Computer Graphics Forum, Proceedings of Eurographics 2005* (2005) 601–609
22. L. Kobbelt, S. Campagna, J. Vorsatz, H.-P. Seidel, Interactive multi-resolution modeling on arbitrary meshes, *SIGGRAPH '98: Proceedings of the 25th annual conference on Computer graphics and interactive techniques* (1998) 105–114
23. M. Bostch and L. Kobbelt, An Intuitive Framework for Real-Time Freeform Modeling, *SIGGRAPH '04: Proceedings of the 31th annual conference on Computer graphics and interactive techniques* (2004) 628–632
24. M. Botsch, M. Pauly, M. Gross and L. Kobbelt, PriMo: Coupled Prisms for Intuitive Surface Modeling, *Eurographics Symposium on Geometry Processing* (2006) 11–20
25. J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, Shang-Hua Teng, Hujun Bao, Baining Guo and Heung-Yeung Shum, Subspace gradient domain mesh deformation, *SIGGRAPH '06: Proceedings of the 33th annual conference on Computer graphics and interactive techniques* (2006)1126–1134
26. M. Alexa, D. Cohen-Or and D. Levin, As-Rigid-As-Possible Shape Interpolation, *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000) 157–164
27. K. Shoemake and T. Duff, Matrix animation and polar decomposition, *Proceedings of the conference on Graphics interface '92* (1992) 258–264

28. D. L. James and . D. Twigg, Skinning mesh animations, *SIGGRAPH '05: Proceedings of the 32th annual conference on Computer graphics and interactive techniques* (2005) 399–407
29. R. W. Sumner, M. Zwicker, C. Gotsman, J. Popovic As-rigid-as-possible shape manipulation *SIGGRAPH '05: Proceedings of the 32th annual conference on Computer graphics and interactive techniques* (2005) 488–495
30. M. Alexa, Linear combination of transformations, *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques* (2002) 380–387
31. R. W. Sumner and J. Popovic, Deformation transfer for triangle meshes, *ACM Trans. Graph.* **23** (2004) 399–405
32. http://www.loria.fr/∼levy/Graphite/index.html
33. T. A. Davis, Algorithm 832: UMFPACK — an Unsymmetric-Pattern Multifrontal Method, *ACM Transactions on Mathematical Software* (2004) 196–199
34. K. Shoemake, Animating rotation with quaternion curves, *SIGGRAPH '85: Proceedings of the 12th annual conference on Computer graphics and interactive techniques* (1985) 245–254
35. F. S. Grassia, Practical parameterization of rotations using the exponential map, *Journal of Graphics Tools* (1998) 29–48