Context-Aware Skeletal Shape Deformation

Ofir Weber[1], Olga Sorkine[†2], Yaron Lipman[3] and Craig Gotsman[1]

[1]Technion, Israel [2]TU Berlin, Germany [3]Tel Aviv University, Israel

Abstract

We describe a system for the animation of a skeleton-controlled articulated object that preserves the fine geometric details of the object skin and conforms to the characteristic shapes of the object specified through a set of examples. The system provides the animator with an intuitive user interface and produces compelling results even when presented with a very small set of examples. In addition it is able to generalize well by extrapolating far beyond the examples.

Categories and Subject Descriptors (according to ACM CCS): I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism

1. Introduction

Editing and animation of 3D objects is an important problem in computer graphics, with applications in the film and gaming industry. In this work, we address the problem of deformations for complex articulated 3D shapes. By articulated shape we mean one whose general set of poses (or kinematics) can be described using a simple subspace representation, such as a skeleton, as in character animation of humans, animals and similar creatures. Given an arbitrary pose, our goal is to design a compelling, naturally-looking deformation of the rest shape in this pose.

There are three important properties which a deformation method should possess: (i) the deformed shape should preserve local details present in the rest shape, such as fine-scale geometric skin characteristics (e.g., bumps or wrinkles); (ii) the deformed shape should conform to the *characteristic shapes* of the specific character being animated: for example, muscles should bulge or cloth should crease; (iii) the deformation should be a smooth function of the pose, such that a plausible and aesthetically-pleasing animation of the shape is obtained as the deformation controls continuously modify the pose.

In this work we exploit recent developments in differential surface representations and detail-preserving surface deformation for the purpose of articulated character animation. These techniques are capable of intuitively and efficiently deforming complex surfaces while preserving local details at multiple scales. Employing these deformation techniques addresses the first and third criteria listed above, namely, local detail preservation and smooth dependency on the pose. However, pure geometric deformation methods cannot be aware of the characteristic shapes of the deformed object, and this information must come from an external source. We provide this *contextual* information in the form of *examples* of the animated object in a small number of poses. Our key observation is that the clear separation of characteristic shape from intrinsic surface detail allows effective interpolation *and* extrapolation of the example shapes. Combining these two independent components, we describe a deformation method that can handle shapes of high geometric complexity and produce natural skin deformation and animation, in an effective and space-efficient manner.

1.1. Background

The skin-skeleton paradigm is very popular in the animation industry and is the method of choice in most commercial modeling/animation packages. Its popularity stems from the intuitive manipulation, the ability to quickly solve inverse-kinematics on a small subspace (the skeleton) and efficient mapping of the animation mechanisms onto the graphics hardware. The difficult part is obtaining high-quality deformations of the skin given a pose of the skeleton. The most wide-spread skin deformation (or, in short, skinning) technique is the so-called Skeletal Subspace Deformation (SSD) method (see [LCF00] for a description), which transforms each vertex of the skin by a weighted linear blend of transformations associated with the skeletal joints influencing that vertex. These blending weights are typically set and tweaked by the animator; they make SSD notoriously difficult to control: the linear nature of the transformation blending causes the candy-wrapping and elbow-collapse artifacts. Consequently, the joint influence weights must be tweaked so that *every* possible pose looks reasonable, an extremely tedious, unintuitive and time-consuming chore. Lastly, SSD (like any other skinning technique) cannot achieve credible contextual movement, because the algorithm lacks any knowledge of these movements. A bend of an arm will not cause the muscle to bulge since all reachable shapes are limited to the linear subspace of transformations, none of which contain this information.

† Supported by the Alexander von Humboldt Foundation

To overcome the problem of context freedom, the Pose-Space Deformation (PSD) technique [LCF00, SCFRC01] augments SSD by morphing with *displacements* taken from a set of user-supplied example skins. Usually the animator designs a few key poses of the skin+skeleton, manually deforming the skin such that it attains the correct shape for the character in those poses. Alternatively, the example skins can be obtained by 3D acquisition [ACP03, ASK*05]; skeleton rigging and precise registration and correspondence of all skins is then required. The interpolation can be performed on a per example basis, assigning the same weight to all the skin vertices of the example (as in [LCF00, SCFRC01]), or on a per-vertex basis, assigning each vertex of the example a different weight ([KM04, RLN06]). The latter is more expensive, yet much better suited for situations where the examples are sparse. Kry et al. [KJP02] perform principle component analysis on the displacements learned from examples to compress their representation. Mohr and Gleicher [MG03] use examples to automatically find additional skeleton joints and weights which can reduce some of the undesirable effects of SSD.

One of the underlying problems with SSD-type deformation is lack of treatment of the skin surface as a connected manifold, because each vertex is transformed independently of its neighborhood on the surface. Therefore, self-intersections may easily occur. When using a differential surface representation such as we advocate, the deformed skin is obtained as the result of a global optimization process that accounts for the geodesic relationships between the skin vertices, resulting in a graceful and detail-preserving deformation. Recent skin deformation methods have adopted variants of this approach [ASK*05, SZGP05, DSP06, HSL*06, SYBF06, YHM06], yet the full potential of differential deformations, coupled with examples, has not yet been explored.

Surface deformation techniques have been recently shown to be effective in animation applications based on motion-capture data [BPGK06, HSL*06, KS06, SYBF06, YHM06]; most of these methods rely on non-linear global optimization and are context-free. The quality of the deformations they produce is typically much higher than that of SSD, including complete avoidance of candy-wrapping and joint collapse; however, this comes at a significantly higher computational price.

The MeshIK system [SZGP05, DSP06] supports example-based posing of a mesh (without a skeleton) by non-linear optimization in the space of the examples. While performing well in a dense space of examples, the technique has difficulty to extrapolate or operate with a sparse example set. Another problem with MeshIK is its strong dependence on the example shapes at run-time: the non-linear optimization procedure incorporates the complete set of examples within the global non-linear solution, which becomes very expensive as the number of examples increases. This problem was alleviated in [DSP06] by operating in a meaningful subspace consisting of a set of "bones", which, although one step back towards a skeletal structure, do not necessarily fit together to a well-formed skeleton, but rather define a set of rigidly-moving regions of the mesh.

Anguelov et al. [ASK*05] present a comprehensive framework, called SCAPE, for human shape acquisition and animation. SCAPE combines example-based deformation with linear differential deformations. More specifically, SCAPE assumes that each rest-pose triangle is first rigidly transformed by exactly one joint, thus propagating the skeleton transformations to the skin. Each triangle's transformation is additionally *corrected* by another transformation, learned from the example skins by linear regression methods. The correction transforms, learned per triangle, are functions of the pose configuration of the two closest influencing joints; up to 63 parameters per triangle needed to be stored in memory. The connected skin surface can then be reconstructed using Poisson stitching [YZX*04]. Due to the linear regression used, when the correction transformations contain rotational components, the result deformation will suffer from visual artifacts similar to those of SSD. Extending SCAPE to use non-linear regression can be expensive.

Our approach is similar to SCAPE [ASK*05] in that we also use an underlying linear deformation approach; however, we allow interactive marking of the joint influences, we do not restrict the influence of each skin element to one or two joints, and we propagate the skeleton transformation further using harmonic interpolation [ZRKS05]. In our framework the underlying deformation is already detail-preserving, thus local surface texture deformation need not be captured or corrected by the examples. When all the example data is present, our method will interpolate the example skins precisely when provided with the example skeleton configuration. This is not possible with SCAPE. In addition, we develop a compact representation of the example data, such that just a small number of localized skin elements need to contain augmented example information, with minimal impact on deformation quality.

Using example meshes as an input for a deformation method can be problematic for resource-demanding applications such as computer games. Typical games use a large number of different characters; many of them may be present in a single game scene. The amount of memory allocated for deformation purposes is usually small compared to textures, scene geometry, AI, physics, etc. Using examples to guide a deformation may easily lead to an increase in the amount of memory allocated for character geometry by few orders of magnitude. Our algorithm needs fewer examples than PSD [LCF00, SCFRC01] in order to produce high-quality results. In addition, the examples themselves are represented in a compact manner.

In parallel to us, Wang et al. [WPP07] developed an approach similar to ours. However, contrary to our method, their technique relies on regression methods and needs many examples in order to learn the deformation model. An important contribution of their work is the formulation of an approximation to the Poisson system, which improves performance.

1.2. Outline and contributions

Our deformation framework enables interactive posing and animation, and is based on the skin-skeleton paradigm familiar to artists. We allow intuitive control over the skin animation by painting a rough influence field of each joint on the skin. Given a desired skeleton pose, the skin is first deformed using the detail-preserving shape editing approach. This already gives reasonable results; however, if contextual data is present, the result is also influenced by the characteristic shapes extracted from the examples via a differential "morphing" process that enables correct extrapolation (Sec. 2). The characteristic shapes are compactly represented, such that essentially only a fraction of the skin elements (called anchors) contains example information; the correction for the entire shape is performed in real-time by smooth interpolation across the surface (Sec. 3). Our main contributions in this work are:

- An example-based skin deformation framework that works well with a very sparse example set and enables interpolation as well as meaningful extrapolation of example data.
- A framework based on the clear separation of intrinsic surface detail information from pose-dependent characteristic shape information.
- Compact representation of the example data, enabled by the generally low-frequency nature of the characteristic shapes. This leads to space- and time-efficient deformation with an intuitive tradeoff of memory consumption and performance for result quality.

2. Shape deformation framework

We describe the setup and interface of our deformation framework first from the user's point of view, and then detail the core algorithms present in the system.

2.1. User setup and notations

Our system requires minimal user effort to setup for skeletal shape deformation. The input is a manifold triangle mesh, which we call the *rest shape* $\mathcal{S}_0 = (\mathcal{V}_0,\mathcal{F})$ and its associated skeleton structure having joints $\mathcal{B} = \{b_1,...,b_K\}$ and links between the joints. In general, we denote the mesh geometry (vertex positions) by $\mathcal{V} = \{v_1,...,v_N\}$ and the connectivity (the set of faces in the mesh) by $\mathcal{F} = \{f_1,...,f_M\}$. In addition to providing the skeleton, the user is required to establish a basic correspondence between the skeleton and the rest shape: for each joint b_k, the vertices of $\mathcal{S}_0$ associated with b_k are specified, meaning that the user marks regions $H_k \subset \mathcal{V}$ of the rest shape surface that should completely follow the movement of each joint. We call these regions *joint handles*. As we will see later, when no example shapes are provided, the joint handles are transformed exclusively by the corresponding joint transformation; when examples are present this behavior is modified by mimicking the examples. Note that the handles should be disjoint, but need not cover the entire mesh surface. Their marking can be performed via a simple binary painting interface (see Fig. 1 and the accompanying video). No numerical input of influence weights is required; in a sense our system automatically deduces the joint influences on the entire surface from the painted regions.

This basic setup is enough to perform detail-preserving skeletal deformation with the algorithm described next. However, this will typically not capture the "characteristic behavior" of the object – its "context". To increase the realism of the deformation, the system incorporates example shapes that put the deformation into context, demonstrating characteristic deformations of the shape, such as bulging of muscles and appearance of folds for human or animal shapes. Our system accepts such examples in the form of D additional mesh geometries $\mathcal{V}_1,...,\mathcal{V}_D$ (the connectivity $\mathcal{F}$ is shared by all the shapes), coupled with corresponding skeleton poses, specified by joint transformations $P_j = \{\mathbf{R}_{j,1}, \mathbf{R}_{j,2}, ..., \mathbf{R}_{j,K}\}$ for each example $\mathcal{V}_j$. Only a small number of examples are needed to produce highly realistic deformations; an example shape can exhibit several characteristic behaviors simultaneously (for instance, all the limbs of the character may bend). In addition to providing the example shapes, the user can specify the desired tradeoff between efficiency (i.e., space and time complexity) and quality of deformation by controlling the compactness of the example shape representation; this is described in more detail in Sec. 3.

2.2. Basic skeletal surface deformation

Once the joint handles H_k are specified, the surface of the rest shape $\mathcal{S}_0$ can be deformed to conform to any given skeleton pose. For this purpose we associate a continuous scalar field $\mathbf{w}_k$ with each joint b_k, which assigns an influence value $\mathbf{w}_k(v)$ to each mesh vertex v. The vertices that "belong" to the joint handle are assigned the value $\mathbf{w}_k(v) = 1$, while vertices belonging to other joint handles are assigned $\mathbf{w}_k(v) = 0$. All other vertices are assigned values $\mathbf{w}_k(v) \in [0,1]$ obtained as discrete harmonic functions over the mesh. We compute these values for each mesh *vertex* by solving the Laplace equation:

$$\mathbf{L}\mathbf{w}_k = 0, \tag{1}$$

subject to the Dirichlet boundary conditions $\mathbf{w}_k(v) = 1$ for $v \in H_k$ and $\mathbf{w}_k(v) = 0$ for $v \in H_l$ where $l \neq k$. The operator $\mathbf{L}$ is the Laplace-Beltrami operator associated with the rest shape $\mathcal{S}_0$, discretized using the cotangent weights [PP93]. After computing the values of $\mathbf{w}_k$ for the mesh vertices, we compute corresponding scalar fields, $\mathbf{h}_1,...,\mathbf{h}_K$ for the mesh triangles by averaging the values at the three vertices of each triangle. Since the joint handles are mutually exclusive, $\mathbf{w}_1,...,\mathbf{w}_K$ are a partition of unity over the mesh. As a result, $\mathbf{h}_1,...,\mathbf{h}_K$ sum to unity on each face.

The influence fields $\mathbf{h}_k$ are computed once and can then be used to perform arbitrary deformations of the shape: they serve as blending weights to distribute the transformation of the joints to the skin mesh. Given an arbitrary skeletal pose $P = \{\mathbf{R}_1, \mathbf{R}_2, ..., \mathbf{R}_K\}$, where $\mathbf{R}_k$ are relative joint transformations with respect to the rest pose, each mesh face f is assigned a blended transformation, expressed as $\mathbf{R}(f) = \mathbf{h}_1(f)\mathbf{R}_1 \oplus \mathbf{h}_2(f)\mathbf{R}_2 \oplus ... \oplus \mathbf{h}_K(f)\mathbf{R}_K$. We assume that

the transformations $\mathbf{R}_k$ are rotations, as this is a common case in skeletal animation (although arbitrary transformations can be easily handled via polar decomposition).

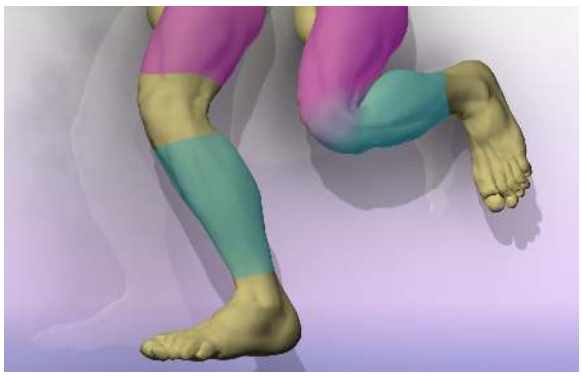

Figure 1: *Painting joint influences and deforming the skin.* **(*left*)** *The colored regions designate the joint handles that are under the influence of corresponding joints (the knee and the thigh), as painted by the user.* **(*right*)** *The detail-preserving deformation uses these regions as modeling constraints that move rigidly with the joints; the rest of the skin deforms according to the optimization process.*

To correctly blend between rotations, the operator $\oplus$ cannot be a simple addition – this would lead to the well-known artifacts of linear blend skinning (as in SSD). Instead, we represent the rotations using log-quaternions [Gra98]; these are 3-vectors whose direction is the axis of rotation and the magnitude is the rotation angle. Linear combinations of log-quaternions still represent rotations, and are therefore able to efficiently blend between more than two rotations. Note that in general, when linearly interpolating log-quaternions, the interpolation path is not a geodesic on the sphere of unit quaternions. Thus, interpolation of log-quaternions is sensitive to the initial orientation of the quaternions involved. This means that the mesh is not invariant to a common rotation of all skeleton joints. In addition, in order to avoid the singularities of the quaternion's logarithm, we prefer to use log vectors with smaller magnitude. Therefore, when blending several rotations $\mathbf{R}_1, ..., \mathbf{R}_K$, we think of one of them (say, $\mathbf{R}_1$) as the identity and represent the other rotations relative to it. The result is a new set of rotations: $\mathbf{R'}_1, ..., \mathbf{R'}_K = \mathbf{I}, \mathbf{R}_1^{-1}\mathbf{R}_2, ..., \mathbf{R}_1^{-1}\mathbf{R}_K$. After blending these relative representations, we multiply back by $\mathbf{R}_1$ to obtain the absolute result. Such relative blending of rotations is insensitive to the skeleton orientation. When blending between the identity and another rotation, linear combination of the log-quaternion representation gives the same effect as spherical linear interpolation (slerp). This means that when only 2 joints are involved, the resulting interpolation technique is equivalent to slerp.

Once the blended rotations $\mathbf{R}(f)$ are computed per face, they are applied to the rest shape triangles, transforming each of them independently. To obtain the final connected mesh, we apply Poisson stitching to these transformed triangles [SP04, YZX*04], namely, we compute the gradients of the transformed triangles and solve the Poisson equation for the deformed mesh vertices:

$$\mathbf{L}[\mathbf{x}\,\mathbf{y}\,\mathbf{z}] = \mathrm{div}[\mathbf{g}_x\,\mathbf{g}_y\,\mathbf{g}_z], \tag{2}$$

where **x, y, z** are the deformed mesh coordinate functions and $\mathbf{g}_x\ \mathbf{g}_y\ \mathbf{g}_z$ are the 3×3 stacked gradient matrices of the transformed triangles. **L** is the same Laplace operator used in Eq. (1). Note that the Poisson equation requires boundary conditions because the gradients are translation-invariant; we use the positions of the vertices associated with the root joint of the skeleton as Dirichlet boundary conditions. Botsch et al. [BSPG06] recently showed that instead of plugging the triangles gradients to the right-hand side of Eq. (2), it is sufficient to use the so-called deformation gradients [SP04]. This will lead to the following, equivalent, linear system:

$$\mathbf{L}[\mathbf{x}\,\mathbf{y}\,\mathbf{z}] = \mathrm{div}[\mathbf{S}], \tag{3}$$

We simply treat the triangle rotations, $\mathbf{R}(f)$ as 3×3 deformation gradient **S** matrices and plug them into the right-hand side of Eq. (3). This eliminates the need to constantly compute the gradients during deformation. For details on the Poisson system setup and the equivalence of the two linear systems, see [BSPG06]. Figs. 2 and 3 compare what we get vs. SSD skinning.

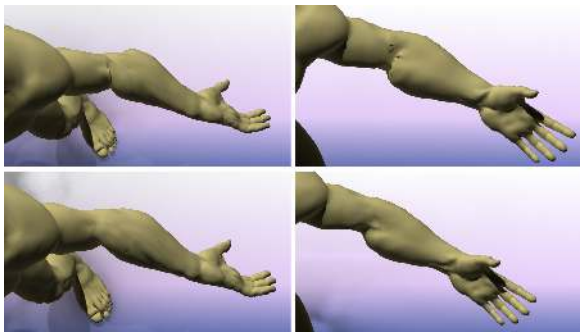

Figure 2: *Skinning using SSD and our method without examples.* **(*top row*)** *SSD. Notice the artifacts in the elbow area.* **(*bottom row*)** *Our method.*

The deformation technique described above is a variant of the method proposed by Zayer et al. [ZRKS05] and Lipman et al. [LCOGL07], adapted to the skeletal deformation setup. Zayer et al. [ZRKS05] did not explicitly discuss the limitations of transformation blending when more than two handle regions are involved; Lipman et al. [LCOGL07] used a non-linear system to correctly blend the transformations. A somewhat similar approach to skeletal deformation was demonstrated by Shi et al. [SYBF06]; however, they rely on a manual assignment of skin vertices to bones and joints and require an augmented volumetric mesh structure, which increases the complexity of the solution.

2.3. Using context: example shapes

When example shapes are provided, our system incorporates the additional information about the shape's characteristic behavior, as contained in the examples. Since the geometric deformation method described in the previous section is detail-preserving, and thus faithful to the local geometric texture of the shape, the difference between an

example shape and the rest shape deformed into the pose of the example, is typically smooth and varies slowly across the surface. We use a differential representation of this difference, on a per-face basis, and enable its blending for arbitrary poses. The low-frequency nature of this additional deformation information also allows compact representation, as described in the next section.

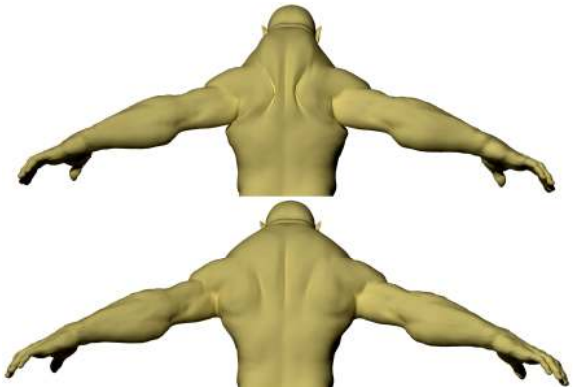

Figure 3: *Skinning using SSD and our method without examples.* ***(top)*** *SSD. Notice the artifacts in the shoulder area.* ***(bottom)*** *Our method.*

To extract the deformation nuances from an example shape $\mathcal{V}_j$, we first compute a rotation for each triangle of the example. The rotations are computed according to the algorithm described in Sec. 2.2, using the example pose P_j as input. We apply the inverse rotation to each of the example's triangles. This can be thought of as a process of transforming the example into the pose of the rest shape. The result is a set of new normalized examples, all in a unified coordinate system. We then compute the relative transformation between the rest shape triangle and the corresponding triangle in the *normalized* example (we use two triangle edge vectors and the normal to disambiguate the transformation). We obtain the so-called deformation gradient $\mathbf{T}_{j,m}$ (for the j'th shape and m'th triangle) which encodes the example shape in a rotation-invariant manner, because the rotation due to pose has been factored out. The full representation of the characteristic behavior present in the j'th example shape is, therefore, its skeleton pose P_j and the set of relative transformations $\{\mathbf{T}_{j,1},...,\mathbf{T}_{j,M}\}$.

When an arbitrary pose P is given, we would like to deform the shape in such a way that it benefits from *all* the examples, proportionally to the similarity between the skeleton poses. In other words, we would like to create meaningful interpolations and extrapolations of the examples in the parameter space of skeletal joint transformations. The similarity metric should be local, in the sense that when only part of P (say, the elbow configuration) is close to an example pose, then the corresponding part of the shape is significantly influenced by the example, whereas the rest of the shape is not. This allows the use of a very small number of example shapes that exhibit several characteristic behaviors of different limbs simultaneously.

We employ the weighted pose-space deformation (WPSD) ideas [KM04] to compute the similarity metric between poses and the blending weights. In a nutshell, WPSD accepts a set of poses $\{P_j\}$ described by the Euler angles of the joint transformations, and K scalar fields defined on the skin mesh elements, defining the influence of each joint on each mesh element. Traditionally, these joint influences were manually painted and tweaked by the animator; we replace them by the harmonic influence fields $\mathbf{h}_k$ described earlier, significantly simplifying the process. Then, for a given pose P and triangle f_m, WPSD computes D weights $a_{j,m}$ that describe how close that element is to its counterpart in each of the examples. The weights $a_{j,m}$ are computed using RBF interpolation on the example poses; in addition, we incorporated the linear extrapolation technique proposed by [SCFRC01] for simple pose-space deformation (PSD) and adapted it to WPSD.

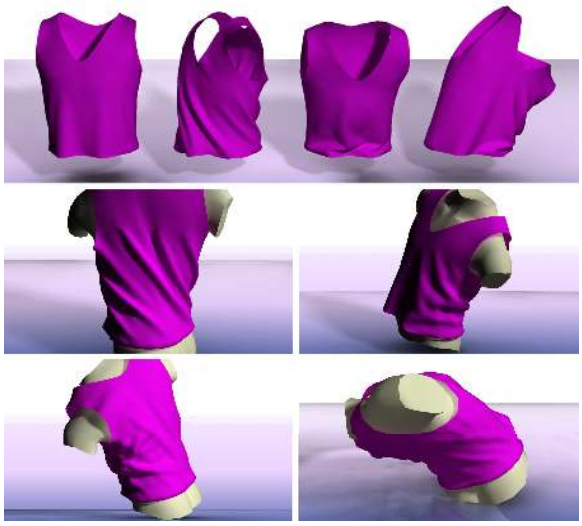

Figure 4: *Deformation using characteristic shapes.* ***(top row)*** *Rest shape and 3 examples.* ***(bottom two rows)*** *Deformations of the rest shape in arbitrary poses.*

The final construction of the deformed shape is performed by computing the total deformation gradient of each triangle, applying the geometric deformation method $\mathbf{R}(f_m)$ to the triangle augmented by the blend of the example representations. Since $\mathbf{T}_{j,m}$ may contain rotational components, linear blending may lead to visual artifacts. For a precise combination, we compute the polar decomposition of each relative example transformation $\mathbf{T}_{j,m} = \mathbf{Q}_{j,m}\mathbf{S}_{j,m}$ and separately combine the rotations $\mathbf{Q}_{j,m}$ and the skew components $\mathbf{S}_{j,m}$. The final transformation for the m'th triangle is then given by:

$$\begin{aligned} \mathbf{T'}_m &= \mathbf{R}(f_m)\mathbf{A}_m \\ \mathbf{A}_m &= \left[\bigoplus_{j=0}^{D} a_{j,m}\mathbf{Q}_{j,m}\right]\left[\sum_{j=0}^{D} a_{j,m}\mathbf{S}_{j,m}\right] \end{aligned} \tag{4}$$

We again use log-quaternions to represent and combine the rotations $\mathbf{Q}_{j,m}$. By applying the transformations $\mathbf{T'}_m$ to the right-hand side of Eq. (3) and solving the Poisson stitching, we arrive at the final deformed shape. Note that only one Poisson stitching process is needed, since we do not actu-

ally solve for the intermediate shape obtained by purely geometric deformation, but only use its transformations $\mathbf{R}(f_m)$ obtained by combining the pre-computed harmonic fields $\mathbf{h}_k$. See Fig. 4 for an example.

3. Compact representation of examples

As mentioned above, the characteristic deformation behavior of a shape tends to have a low-frequency nature, meaning that the difference between a basic skeletal deformation and the given example shape is a smooth function over the surface. Thus there is significant coherence within the representation of an example mesh, i.e. between $\{\mathbf{T}_{j,m}\}$, the relative transformations of the individual mesh triangles and also between $\{\mathbf{A}_m\}$. We exploit this fact to "compress" the example representation, expressing each $\mathbf{A}_m$ as a combination of a small number of basis functions, centered around particular mesh elements called *anchors*. This way only the relative transformations of the anchors need to be stored, instead of the entire set of $D{\cdot}M$ matrices $\mathbf{T}_{j,m}$. A conceptually similar approach was used in [SCOIT05] for geometry compression; here we apply it to deformation compression.

Assume that instead of storing all the matrices $\mathbf{T}_{j,m}$ we choose a subset of triangles $\{f_i\}$, $i\in\mathcal{C} = \{c_1,...,c_{M'}\}$, $M'<<M$ for which we store this information. At run time, we can approximate the entire set of $\mathbf{A}_m$'s by smoothly interpolating this subset across the mesh. Since the $\mathbf{A}_m$'s contain both a rotational component and a skew component, we could have used the polar decomposition again in order to perform the interpolation on the two components separately. In practice, if the subset of anchors is nicely distributed over the mesh, linear interpolation will suffice. In total we have 9 scalar values to interpolate over the mesh. Denote by $(d_1,d_2,...,d_M)^T = \mathbf{d}$ one of these scalar fields; the values d_i, $i\in\mathcal{C}$ are known. We can approximate the rest of $\mathbf{d}$ by solving a Laplace problem for $\mathbf{d'}$:

$$\mathbf{L}\mathbf{d'} = 0, \tag{5}$$

with the boundary conditions $d'_i = d_i$ on the anchors. The operator $\mathbf{L}$ is the triangle-based Laplacian operator, defined on the graph dual to the mesh. For simplicity, we define $\mathbf{L}$ with uniform weights. The quality of approximating $\mathbf{d}$ by $\mathbf{d'}$ depends of course on the number and position of the anchors $\mathcal{C}$. We employ a greedy algorithm to choose these anchors in the preprocess stage, as proposed in [SCOIT05]. We use the rotation-skew decomposed components of the $\mathbf{T}_{j,m}$'s separately in an attempt to optimize the approximation of both components of the $\mathbf{T}_{j,m}$'s. In the beginning, we initialize $\mathcal{C}$ with a small number of randomly distributed anchors. Then we solve (5) for the 3 rotational (log-quaternion) scalar fields and find the triangle whose entry in $\mathbf{d'}$ has the maximal error with respect to its counterpart in $\mathbf{d}$. The error is defined as the angle of the relative rotation between the element in $\mathbf{d'}$ and the element in $\mathbf{d}$. This triangle is added to $\mathcal{C}$ and the process repeats. After the desired number of rotational anchors is obtained, we proceed to compute the skew anchors in the same fashion. Since the skew matrix is symmetric, we have 6 independent scalar fields to solve for. The error metric for the skew component is the Frobenius matrix norm. The process is then repeated for all example meshes. The total number of anchors M' is limited by the space consumption constraints provided by the user (we need to store $9{\cdot}D{\cdot}M'$ scalars to sparsely represent all the examples).

At run-time, the D sets of anchor values for the rotational and the skew components are blended using the WPSD weights, as described in Sec. 2.3, resulting in a single set of blended anchors. The transformations $\mathbf{A}_m$ are then computed for the entire mesh by solving (5) with the blended anchor boundary constraints.

Our experiments show that for natural shape examples, only a small fraction of the mesh triangles need to be designated as anchors; typically up to 5% is enough to obtain an excellent approximation; a significantly reduced number of anchors leads to smoothed incorporation of example behavior since the harmonic interpolation over-smoothes the relative transformations. The advantage of such a sparse representation is two-fold: it drastically reduces the memory consumed by the examples by at least a factor of 20, and reduces the time spent on computing WPSD weights and evaluating Eq. (4) at run-time, since the computation is performed only on the anchors. The price paid is the solution of an additional linear system; however, this can be significantly accelerated, as explained in the next section.

4. Implementation issues

The main computational bottleneck in our approach is the solution of sparse linear systems. In the preprocess stage we compute the harmonic fields $\mathbf{w}_k$ (Eq. (1)), as well as the greedy choice of the anchors for sparse example representation (solving Eq. (5) with varying constraints). At run-time, given a skeleton pose, solving a sparse system is required in order to reconstruct the blended examples information (solving Eq. (5) for 9 right-hand sides) and in the final Poisson stitching (Eq. (3)). Naturally, it is desirable to optimize the run-time performance, even at the expense of a longer pre-processing stage.

We use a direct sparse solver [Tol03] to precompute the Cholesky factorization of the Laplacian matrix involved in the Poisson stitching; since the system matrix does not change at run-time, the factorization can be reused to solve for multiple right-hand sides by back substitution, which is very fast compared to standard iterative solvers [BBK05]. Solving the linear system (5) for the anchors can be avoided altogether at run-time by representing the solution $\mathbf{d'}$ as an affine combination of M' harmonic basis functions $\mathbf{u}_i$, $i\in\mathcal{C}$, constructed as in Sec. 2.2:

$$\mathbf{L}\mathbf{u}_i = 0,$$

subject to $\mathbf{u}_i(c_j)=\delta_{ij}$. Then:

$$\mathbf{d'} = \sum_{i\in\mathcal{C}} d_i\mathbf{u}_i$$

We only need to pre-compute the $\mathbf{u}_i$'s and combine them at run-time, which is cheaper than back-substitution. For ad-

ditional speedup and space conservation, we selectively reduce the amount of $\mathbf{u}_i$ values stored per triangle: we sort them in descending order and keep only the larger values (e.g., such that they sum to 0.95; we then rescale them to sum to 1). The cut-off parameter may be exposed to the user for better quality control. Typically most of the original values are nearly zero because each triangle is mostly affected by the closest anchors. We also use the same process to reduce the amount of joint influence weights $\mathbf{h}_k$ stored per triangle. Hence, at run-time only one equation (Eq. (3)) needs to be solved (using back-substitution only).

5. Experimental results

Our mesh deformation system has been fully implemented as a plugin to the Maya® commercial animation system. The code is not optimized, although it does use the acceleration methods described in Sec. 4.

Our system can generate deformations without the use of any example meshes, thus we can compare our results to traditional linear weighted blending of transformations (SSD). As Figs. 1-3 demonstrate, our method produces much smoother and more natural results on a humanoid model. The muscles contract naturally although only the shoulder joints rotate. Notice the self intersections near the scapula region and the over-stretching of the shoulder area when SSD is applied (Fig. 3).

When presented with example shapes, the deformation captures the context illustrated by the examples and reproduces it. Moreover, it has significant extrapolation capabilities, as illustrated in Fig. 5. When the example shows how a muscle bulges slightly and the fingers bend slightly as the arm and wrist are bent, the same muscle continues to bulge more dramatically as the arm is bent further and the fingers continue to rotate as the wrist is bent further. Note that there is no skeletal structure whatsoever in the fingers! The conventional PSD method [LCF00, SCFRC01], which is based on displacement vectors, cannot handle the large rotation of the fingers correctly. SCAPE [ASK*05] will also fail on this data since its regression model is linear.

Cloth deformations that do not exhibit strong dynamic effects can be modeled with our system. We use three examples to guide the deformation of a shirt, controlled by the spine joints. Fig. 4 shows the application of our deformation technique to the shirt. This exhibits quite complex deformations with large local rotations.

Another scenario is depicted in Fig. 6. Here the rest mesh is a flat mesh with sinusoidal waves on it. These waves are the fine details. An example is presented which shows the mesh folded over in a circular shape by approximately 180°. This example suffices to indicate what the mesh should look like when it is bent by 360°. The result is exactly as we would expect – a cylindrical form with no distortion of the fine detailed sinusoidal waves.

The example meshes which supply the context can occupy a large amount of memory. This can be a serious problem for many applications. We represent our examples in a compact manner using a small subset of the mesh triangles – so-called anchors. Even when a very small amount of anchors are used, the fine geometric details of the mesh are preserved. The flat mesh in Fig. 6 is a classic example of a very smooth and uniform deformation. This leads to extreme compression capabilities where only 30 anchors among 24,000 triangles were used.

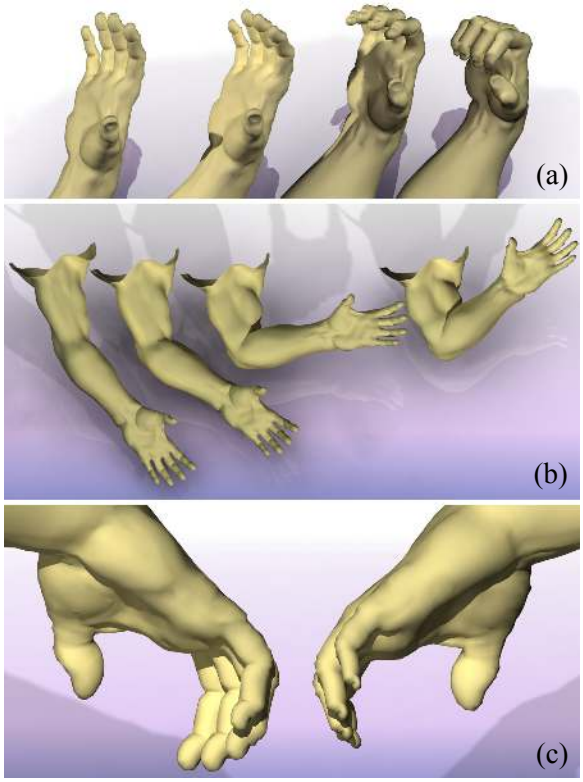

Figure 5: *Deformation using characteristic shapes.* (a) ***left to right -*** *Rest shape; one example with slight bend of fingers; two deformations of the rest shape based on the example. Note the significant bend of the fingers. There are no skeleton joints in the fingers!* (b) ***left to right -*** *Rest shape; one example with slight muscle bulge; two deformations of the rest shape based on the example. Note the significant and natural bulge of the muscle.* (c) *Comparison of* ***(left)*** *our deformation and* ***(right)*** *that of* [*LCF00, SCFRC01*]. *Note the shrinkage in the fingers.*

Fig. 7 shows what happens when we bend the Armadillo leg by rotating the knee joint. On the left is an example mesh. When we ignore it, the deformation preserves the fine details of the mesh but does not deform as expected and the muscles do not bulge. When we use the example, the results are improved. The same figure shows the deformation achieved when all or just some of the triangles of the example are used as anchors. Using fewer anchors leads to less memory consumption and better performance. Note that the 2% anchor image is almost indistinguishable from the 100% anchor image, even though only 290 anchors were used instead of 14,606. Even when only 73 anchors (0.5%) are used, the fine scale details of the original mesh are not harmed.

5.1. The video

The video that accompanies this paper (which can be found at www.cs.technion.ac.il/~gotsman/shape/EG07.zip and should be played using QuickTime 7.0) depicts an interactive session with our system and shows how the results described in the previous section were obtained. The interaction is easy and natural. Influence regions of the various joints may be painted onto the mesh. Then the mesh is deformed by posing its skeleton. After a pre-processing step, real-time performance is obtained during the deformation process. The video shows a comparison between using SSD and our method to deform a humanoid figure without examples, the deformation of the arm model using our method with and without examples, and the deformation of a shirt and a flat mesh using our method based on a small number of examples.

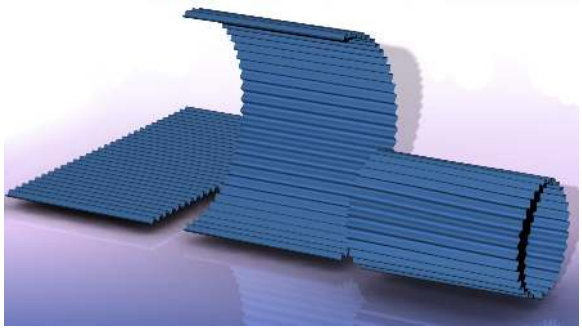

Figure 6: *Deformation using characteristic shape.* ***(left to right)*** *Rest shape of flat mesh with waves; one example with bend; deformation of the rest shape based on the example. Note the significant extrapolation of the bend. Only 30 anchors out of 24,000 triangles were used.*

5.2. Complexity

A main component in our computation is solving a linear system involving the sparse mesh Laplacian matrix. We do this by factoring the matrix by a Cholesky decomposition in a pre-process, which allows to solve the system faster during interaction by back-substitution only. Cholesky factorization takes less than a second on the arm mesh containing 10,000 triangles, and back-substitution can be done in real-time for meshes containing up to 60,000 triangles. We are optimistic that significant speedups can still be obtained by performing much of these computations on the GPU (see next section).

Our anchor selection process is greedy and quite slow, since it checks the quality of each selection by solving the equation associated with it. This, of course, is linear in the size of the example meshes and their number. Despite using the fast factorization update scheme [SCOIT05], it can take up to a minute per example mesh, but since this is also done in a pre-process, it is not critical. We are also sure that a more efficient procedure is possible.

6. Discussion and future work

We have presented a system for skeletal shape deformation that operates within the context of given deformation examples – so-called characteristic shapes. The system has a very simple set-up compared to traditional skinning techniques, and it is robust: small variations in the specified joint handle regions lead to only small changes in the deformation results. Our system uses a differential detail-preserving deformation technique as its underlying deformation core, and thus produces plausible results even when the desired skeleton pose is very different from the provided example poses. The differential encoding of the characteristic behavior exhibited in the examples enables high-quality interpolation and extrapolation for arbitrary poses. Moreover, only a small number of example meshes is sufficient thanks to the WPSD-based interpolation which takes advantage of partial matching between poses. The computational cost of WPSD is significantly reduced by a compact representation of the example deformations, since it is applied only to representative triangles (anchors) which are a small fraction of the entire mesh.

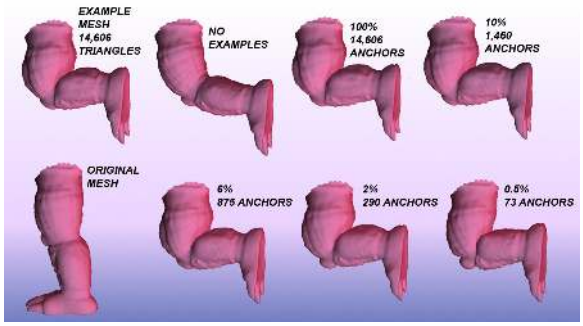

Figure 7: *The effect of using different amount of anchors when deforming the original mesh using one example. Using just 2% of the triangles as anchors produces a muscle bulging effect almost indistinguishable from that produced with 100% anchors.*

The compact representation of examples allows the saving of precious storage space and also offers a viable trade-off between deformation quality and space- and time-efficiency. We use a simple greedy algorithm for the anchors selection which is far from being optimal and also quite slow, requiring re-solving of Eq. (5) with an increasing number of constraints. Although not the bottleneck in our computations, this aspect of our system could definitely be improved.

The first stage in our deformation algorithm is a detail preserving gradient-based technique. We approximate the gradients field of the deformed mesh using harmonic functions guided by the skeleton joints rotations. In some cases, this could lead to a translational gap between a limb and its corresponding joint (see Fig. 7 – NO EXAMPLES image). A possible solution is to constrain the positions of the handle's vertices, found in the H_k's, to the joints by adding additional boundary conditions to Eq. (3). Since gradients are translation-insensitive, using positional constraints will not alter the gradients any further and may lead to visual artifacts. We chose not to use positional constraints in all the examples being presented in the paper and the video. A more sophisticated solution may use a deformation mechanism that handles translations correctly; this will probably lead to a non-linear system and will complicate the solu-

tion. The main reason that we were satisfied with the linear system behavior is the fact that, as soon as examples are added, the approximation of the gradients, achieved by the first stage, is dramatically improved and naturally eliminates the artifact (see Fig.7 and the video).

A possible future direction is to implement the algorithm on the GPU. In fact, apart from the linear system solver, the entire algorithm maps perfectly to the GPU. For the linear solver, we currently use a Cholesky factorization and perform back-substitution during interaction. Back-substitution is a sequential process, hence it does not map well to the GPU. A solver based on multigrid methods could be used, and since multigrid can be easily parallelized, this is suitable for GPU implementation [BFGS03, GWL*03]. However, typical multigrid methods assume that the mesh is structured, which is not the case for arbitrary mesh data, such as the inputs we deal with, so this aspect of the problem requires some additional research.

Acknowledgments

We thank Tamir Shemesh for valuable help with the artwork and Debbie Miller for her help with the video narrative. The models of the arm, leg and beast body are courtesy of Autodesk®. This research has been partially funded by European FP6 IST NoE grant 506766 (AIM@SHAPE) and the Alexander von Humboldt foundation.

References

[ACP03] ALLEN B., CURLESS B., POPOVIĆ Z.: The space of human body shapes: reconstruction and parameterization from range scans. *ACM Trans. Graph. 22*, 3 (2003), 587–594.

[ASK*05] ANGUELOV D., SRINIVASAN P., KOLLER D., THRUN S., RODGERS J., DAVIS J.: SCAPE: shape completion and animation of people. *ACM Trans. Graph. 24,* 3 (2005).

[BBK05] BOTSCH M., BOMMES D., KOBBELT L.: Efficient linear system solvers for mesh processing. *IMA Mathematics of Surfaces XI, Lecture Notes in Computer Science 3604* (2005).

[BFGS03] BOLZ J., FARMER I., GRINSPUN E., SCHRÖDER P.: Sparse matrix solvers on the GPU: conjugate gradients and multigrid. *ACM Trans. Graph. 22,* 3 (2003), 917–924.

[BPGK06] BOTSCH M., PAULY M., GROSS M., KOBBELT L.: PriMo: Coupled prisms for intuitive surface modeling. In *Proceedings of the Symposium on Geometry Processing* (2006), pp. 11–20.

[BSPG06] BOTSCH M., SUMNER R., PAULY M., GROSS M.: Deformation transfer for detail-preserving surface editing. In *Proceedings of VMV* (2006), pp. 357–364.

[DSP06] DER K. G., SUMNER R. W., POPOVIĆ J.: Inverse kinematics for reduced deformable models. *ACM Trans. Graph. 25*, 3 (2006), 1174–1179.

[Gra98] GRASSIA F. S.: Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools 3,* 3 (1998).

[GWL*03] GOODNIGHT N., WOOLLEY C., LEWIN G., LUEBKE D., HUMPHREYS G.: A multigrid solver for boundary value problems using programmable graphics hardware. In *Proceedings of Graphics Hardware* (2003), pp. 102–111.

[HSL*06] HUANG J., SHI X., LIU X., ZHOU K., WEI L.-Y., TENG S., BAO H., GUO B., SHUM H.-Y.: Subspace gradient domain mesh deformation. *ACM Trans. on Graph. 25,* 3 (2006).

[KJP02] KRY P. G., JAMES D. L., PAI D. K.: Eigenskin: real time large deformation character skinning in hardware. In *Proceedings of the Symposium on Computer Animation* (2002).

[KM04] KURIHARA T., MIYATA N.: Modeling deformable human hands from medical images. In *Proceedings of the Symposium on Computer Animation* (2004), pp. 355–363.

[KS06] KRAEVOY V., SHEFFER A.: Mean-value geometry encoding. *International Journal of Shape Modeling 12*, 1 (2006), 29–46.

[LCF00] LEWIS J. P., CORDNER M., FONG N.: Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *Proceedings of ACM SIGGRAPH* (2000), pp. 165–172.

[LCOGL07] LIPMAN Y., COHEN-OR D., GAL R., LEVIN D.: Volume and shape preservation via moving frame manipulation. *ACM Trans. on Graph. 26*, 1 (2007).

[MG03] MOHR A., GLEICHER M.: Building efficient, accurate character skins from examples. *ACM Trans. Graph. 22,* 3 (2003).

[PP93] PINKALL U., POLTHIER K.: Computing discrete minimal surfaces and their conjugates. *Experiment. Math. 2*, 1 (1993).

[RLN06] RHEE T., LEWIS J., NEUMANN U.: Real-time weighted pose-space deformation on the GPU. *Computer Graphics Forum 25,* 3 (2006), 439–448.

[SCFRC01] SLOAN P.-P. J., CHARLES F. ROSE I., COHEN M. F.: Shape by example. In *Proceedings of I3D* (2001).

[SCOIT05] SORKINE O., COHEN-OR D., IRONY D., TOLEDO S.: Geometry-aware bases for shape approximation. *IEEE TVCG 11,* 2 (2005), 171–180.

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. *ACM Trans. Graph. 23*, 3 (2004), 399–405.

[SYBF06] SHI L., YU Y., BELL N., FENG W.-W.: A fast Multigrid algorithm for mesh deformation. *ACM Trans. Graph. 25*, 3 (2006), 1108–1117.

[SZGP05] SUMNER R. W., ZWICKER M., GOTSMAN C., POPOVIĆ J.: Mesh-based inverse kinematics. *ACM Trans. Graph. 24,* 3 (2005), 488–495.

[Tol03] TOLEDO S.: TAUCS: *A Library of Sparse Linear Solvers, version 2.2*. Tel-Aviv University, Available online at http://www.tau.ac.il/~stoledo/taucs/, Sept. 2003.

[WPP07] WANG R. Y., PULLI K., POPOVIĆ J.: Real-Time Enveloping with Rotational Regression. *ACM Trans. Graph. 26,* 3 (2007).

[YHM06] YAN H.-B., HU S.-M., MARTIN R. R.: Skeleton based shape deformation using simplex transformations. In *Proceedings of CGI* (2006), pp. 66–77.

[YZX*04] YU Y., ZHOU K., XU D., SHI X., BAO H., GUO B., SHUM H.-Y.: Mesh editing with Poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3 (2004), 644–651.

[ZRKS05] ZAYER R., RÖSSL C., KARNI Z., SEIDEL H.-P.: Harmonic guidance for surface deformation. In *Computer Graphics Forum* (2005), pp. 601–609.