Scalable and Shape Sensitive As-Rigid-As-Possible Deformations

Paper ID 1234

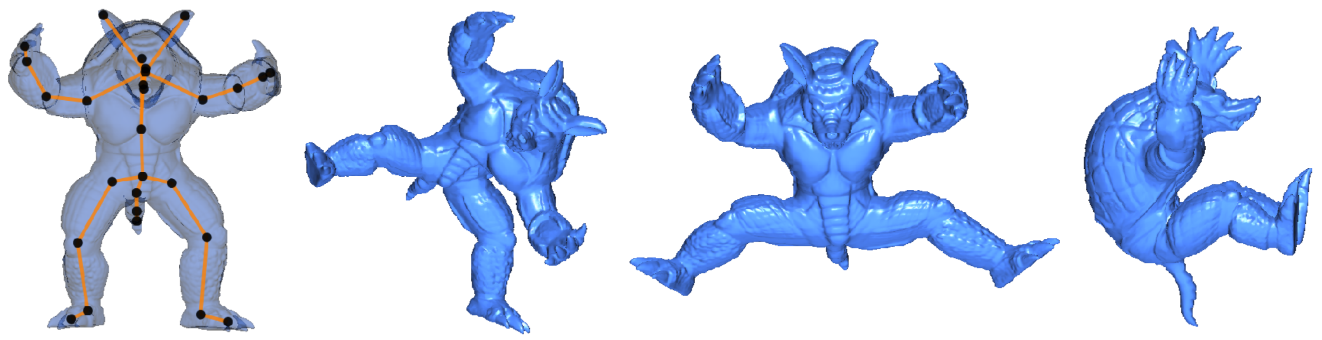

Figure 1. Several poses obtained with skeleton-driven as-rigid-as-possible deformations.

Abstract

The use of as-rigid-as-possible deformations in standard applications such as character animation or modeling has been hindered by two factors: its relatively high cost, specially when aimed at large models, and the fact that it was originally conceived as a space-deforming scheme, being virtually blind to the shape of the model. In this paper we discuss two enhancements of this technique: a less expensive formulation which provides approximate results, and a skeleton-driven deformation scheme capable of deforming meshes in a shape-sensible way. Both methods are capable of handling models of up to a few hundred thousand vertices at interactive rates.

1. Introduction

The ability to edit large 3D models in a controlled way is useful in a variety of applications including modeling, animation, shape interpolation, illustrative visualization, surgical simulation, games and many others. The problem usually imposes some restrictions on the desired result, such as minimizing detail distortion, preserving volumes or avoiding self-intersections. It is also common to require that the method is able to apply deformations at interactive rates.

In this work, we study how to perform the so-called *as-rigid-as-possible* deformations [2] on large models, i.e., models composed of up to a few hundred thousand vertices. The term "as-rigid-as-possible" arises from the fact that the deformation applied to each point of the space is expressed as a rigid transformation. Schaefer et al. [18] have shown how such deformations can be applied to 2D images by using a Moving Least Squares (MLS) minimization scheme. In that work, the authors noted that the extension of their method to 3D would probably lead to an eigenvector problem. Recently, a method capable of handling 3D models [7] was presented. In that approach, the standard eigenvalue problem is replaced by the solution of a depressed quartic equation for each vertex. Although that approach is suitable for editing small models, it fails to yield interactive rates when applied to dense meshes. The two main contributions of this work are aimed at improving the scalability of the process of applying as-rigid-as-possible deformations on 3D models. Namely, we describe a mathematical framework capable of computing such deformations in an approximate way, and a skeleton-driven deformation approach suitable for character animation.

In a nutshell, the approximate method follows closely the mathematical reasoning of [7], i.e., the rotation transformation to be applied to each vertex of the model is expressed in terms of a rotation axis and an angle which maximize a nonlinear system. However, rather than determining one eigenvalue for each mesh vertex, an approximate solution is computed by solving a linear 3×3 system. The second proposal adapts the MLS space deformation scheme into a method fit for deforming meshes by switching from a purely Euclidean distance metric to a shape-sensible metric based on skeletons. The transformation for each mesh vertex is computed by linear interpolation of the rigid transformations of the associated skeleton joints. Experiments show performance improvements over the approach of [7] of around $3\times$ for both the approximate method and the skeleton-driven deformation.

2. Related work

Algorithms and techniques for applying deformations to 3D models have been intensely investigated in the last two decades. Alexa et al. recently surveyed the area [1], dividing the various approaches into freeform methods, which are mostly aimed at producing global smooth deformations, and detail-preserving methods. The former class is further divided into surface-based and space-based methods, while the latter class includes, among others, multiresolution techniques and methods based on differential coordinates.

Several of these techniques cope well with large-sized models. Multiresolution methods work by reducing the original model's cardinality to a coarse representation [13, 15, 16]. Others simply divide the models into parts on which deformations may be applied using a quick method and later combine the deformed parts into a smooth whole [10, 4]. Skeleton-driven deformation approaches (e.g., [14, 22]) are special cases of this idea where the partition is induced by an articulated structure which is also used to specify the desired deformation. A key problem of these approaches is how to extend the deformation of the parts to the complete model. The most commonly used method for performing this task is known as *skeletal subspace deformation* (SSD) or linear blend skinning. SSD is a popular because of its simplicity and efficiency. The idea is to embed a skeleton in the model and assign its vertices to bones of the skeleton. Each vertex may be bound to multiple bones with a set of weights to indicate the influence of each bone. In this context, a separate problem consists of building a suitable skeleton. This can be done either manually or by means of some skeleton generation algorithm (see [6] for a recent survey).

One of the key disadvantanges of SSD is that linear interpolation may generate artifacts. This has led to the proposal of enhancements such as extra weights [21], extra bones [17] and nonlinear interpolation [19]. Recently, Kavan et al. [12] show how to address some of the SSD drawbacks by using dual quaternions to perform the skinning process.

The so-called *as-rigid-as-possible* deformation methods are of special interest, since they tend to produce physically plausible results by avoiding unnatural shearing and non-uniform scaling of the model. In particular, by making use of a Moving Least Squares approach, Schaefer et al. [18] have recently presented a way of computing such deformations for 2D images.

In order to produce as-rigid-as-possible deformations in $\Re^3$, it is necessary to obtain a rigid transformation which best approximates a mapping $f : \Re^3 \mapsto \Re^3$, $f(\mathbf{p}_i) = \mathbf{q}_i$, for given sets of points $\{\mathbf{p}_i\}, \{\mathbf{q}_i\}$. This problem is also known as point registration, or, more specifically, the Absolute Orientation Problem. Analytical solutions have been proposed, for instance, based on SVD (*Singular Value Decomposition*) [3], quaternions [8, 11], orthonormal matrices [9] and dual quaternions [20]. Another related group of techniques includes those based on iterative schemes such as the ICP (*Iterative Closest Point*) algorithm [5] and its variants. In [7] a solution is proposed where the axis and angular parameters of the rotation are obtained. That method requires a smaller number of computations than matricial and even quaternion-based approaches, but is still quite computationally expensive, as it requires solving a depressed quartic equation for every vertex of the mesh.

3. Moving least squares deformation

The Moving Least Squares (MLS) formulation can be thought of as an extension of the traditional Least Squares minimization technique. Rather than finding a global optimum solution for the problem, MLS tries to find continuously varying solutions for all points of the domain. Let us define the deformation operation as a transformation which maps a set of points $\{\mathbf{p}_i\}$ of the domain onto new positions $\{\mathbf{q}_i\}$. Thus, solving the problem for a given point $\mathbf{v} = [x\ y\ z]$ of the domain can be reduced to finding the best transformation $l_{\mathbf{v}}(\mathbf{x})$ that minimizes

$$\sum_i w_i |l_{\mathbf{v}}(\mathbf{p}_i) - \mathbf{q}_i|^2, \tag{1}$$

where w_i are weights of the form $w_i = |\mathbf{p}_i - \mathbf{v}|^{-u}$ for some integer constant $u > 0$.

Let us define the deformation function f as $f(\mathbf{v}) = l_{\mathbf{v}}(\mathbf{v})$. We observe that when $\mathbf{v}$ is close to some constraint $\mathbf{p}_i$, then w_i tends to infinity, which means that f is interpolating with respect to the constraint points, i.e., $f(\mathbf{p}_i) = \mathbf{q}_i$. Furthermore, if $\mathbf{q}_i = \mathbf{p}_i$, then $f(\mathbf{v}) = \mathbf{v}$, for all $\mathbf{v}$, meaning that, in this case, f is the identity function. Finally, it can be shown that f is smooth everywhere for $u \geq 2$. This defines the Moving Least Squares minimization in which the sought transformation $l_{\mathbf{v}}$ depends on the point of evaluation $\mathbf{v}$.

3.1. As rigid as possible MLS

By imposing different additional requirements on the form of $l_{\mathbf{v}}$, we may obtain different results. We may require, for instance, that $l_{\mathbf{v}}$ is a general affine transformation, in which case the classical normal equations solution can be applied directly [18]. For obtaining deformations which are *as rigid as possible*, $l_{\mathbf{v}}$ must be constrained to be a rigid transformation, i.e., $l_{\mathbf{v}}$ must be of the form: $l_{\mathbf{v}}(\mathbf{x}) = \mathbf{x}\mathrm{R} + \mathrm{T}$, where R is a rotation matrix and T is a translation vector. Solving for T yields $\mathrm{T} = \mathbf{q}^* - \mathbf{p}^*\mathrm{R}$, where $\mathbf{q}^*$ and $\mathbf{p}^*$ are the weighted centroids of $\{\mathbf{q}_i\}$ and $\{\mathbf{p}_i\}$ respectively:

$$\mathbf{q}^* = \frac{\sum_i w_i\,\mathbf{q}_i}{\sum_i w_i}. \quad \mathbf{p}^* = \frac{\sum_i w_i\,\mathbf{p}_i}{\sum_i w_i}, \tag{2}$$

This permits us to factor out the translation from (1) by rewriting it as

$$\sum_i w_i |\hat{\mathbf{p}}_i \mathrm{R} - \hat{\mathbf{q}}_i|^2, \tag{3}$$

where $\hat{\mathbf{q}}_i = \mathbf{q}_i - \mathbf{q}^*$ and $\hat{\mathbf{p}}_i = \mathbf{p}_i - \mathbf{p}^*$. Expanding (3), we infer that R minimizes (3) if and only if it maximizes

$$\sum_i w_i \hat{\mathbf{q}}_i \mathrm{R} \hat{\mathbf{p}}_i^{\mathsf{T}}. \tag{4}$$

3.2. 3D rigid transformations

In 3D space, R may be defined as a rotation of an angle α around an axis $\mathbf{e}$. Applying such a rotation on a vector $\mathbf{v}$ yields:

$$\mathrm{R}_{\mathbf{e},\alpha}(\mathbf{v}^{\mathsf{T}}) = \mathbf{e}^{\mathsf{T}}\mathbf{e}\mathbf{v}^{\mathsf{T}} + \cos(\alpha)(\mathrm{I} - \mathbf{e}^{\mathsf{T}}\mathbf{e})\mathbf{v}^{\mathsf{T}} + \sin(\alpha)(\mathbf{e} \times \mathbf{v}). \tag{5}$$

By replacing this definition of R in (4) we obtain

$$\sum_i w_i \hat{\mathbf{q}}_i \mathrm{R} \hat{\mathbf{p}}_i^{\mathsf{T}} = \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}} + \cos(\alpha)(\mathrm{E} - \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}}) + \sin(\alpha)\mathrm{V}\,\mathbf{e}^{\mathsf{T}},$$

where

$$\begin{aligned}
\mathrm{M} &= \textstyle\sum_i w_i \hat{\mathbf{q}}_i^{\mathsf{T}} \hat{\mathbf{p}}_i = \begin{pmatrix} \sum_i w_i \hat{\mathbf{q}}_{ix}\hat{\mathbf{p}}_{ix} & \sum_i w_i \hat{\mathbf{q}}_{ix}\hat{\mathbf{p}}_{iy} & \sum_i w_i \hat{\mathbf{q}}_{ix}\hat{\mathbf{p}}_{iz} \\ \sum_i w_i \hat{\mathbf{q}}_{iy}\hat{\mathbf{p}}_{ix} & \sum_i w_i \hat{\mathbf{q}}_{iy}\hat{\mathbf{p}}_{iy} & \sum_i w_i \hat{\mathbf{q}}_{iy}\hat{\mathbf{p}}_{iz} \\ \sum_i w_i \hat{\mathbf{q}}_{iz}\hat{\mathbf{p}}_{ix} & \sum_i w_i \hat{\mathbf{q}}_{iz}\hat{\mathbf{p}}_{iy} & \sum_i w_i \hat{\mathbf{q}}_{iz}\hat{\mathbf{p}}_{iz} \end{pmatrix}, \\
\mathrm{E} &= \textstyle\sum_i w_i \hat{\mathbf{q}}_i \cdot \hat{\mathbf{p}}_i = \mathrm{Trace}(\mathrm{M}), \\
\mathrm{V} &= \textstyle\sum_i w_i \hat{\mathbf{p}}_i \times \hat{\mathbf{q}}_i = (\mathrm{M}_{23} - \mathrm{M}_{32} \quad \mathrm{M}_{31} - \mathrm{M}_{13} \quad \mathrm{M}_{12} - \mathrm{M}_{21}).
\end{aligned} \tag{6}$$

3.3. Optimization problem

Thus, the optimization problem can be written as

$$\begin{aligned}
\max\ \mathrm{F}(\mathbf{e},\alpha) &= \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}} + \cos(\alpha)(\mathrm{E} - \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}}) + \sin(\alpha)\mathrm{V}\,\mathbf{e}^{\mathsf{T}} \\
\text{s.t.} \quad & \|\mathbf{e}\| = 1, \\
& \cos(\alpha)^2 + \sin(\alpha)^2 = 1.
\end{aligned} \tag{7}$$

By considering the optimality conditions (Kuhn-Tucker) for this problem, the solutions must satisfy

$$(1 - \cos(\alpha))\,\mathbf{e}(\mathrm{M} + \mathrm{M}^{\mathsf{T}}) + \sin(\alpha)\mathrm{V} = k_1\,\mathbf{e}, \tag{8}$$

$$\begin{pmatrix} \mathrm{E} - \mathbf{e}\,\mathrm{M}\,\mathbf{e}^{\mathsf{T}} \\ \mathrm{V}\,\mathbf{e}^{\mathsf{T}} \end{pmatrix} = k_2 \begin{pmatrix} \cos(\alpha) \\ \sin(\alpha) \end{pmatrix} \tag{9}$$

If these conditions are satisfied with $\alpha = 0$ or $k_2 = 0$ then $\mathrm{F}(\mathbf{e},\alpha) = \mathrm{E}$. While searching for $(\mathbf{e},\alpha)$ such that $\mathrm{F}(\mathbf{e},\alpha) > \mathrm{E}$, we can, therefore, assume that both these conditions do not hold. If that search does not succeed, the null rotation is a solution of (7).

Define $\mathrm{N} = \mathrm{M} + \mathrm{M}^{\mathsf{T}}$. Under the assumptions above the system (8) can be re-written in a more compact form as:

$$(\mathrm{N} - \lambda\mathrm{I})\,\mathbf{u}^{\mathsf{T}} = \mathrm{V}^{\mathsf{T}} \tag{10}$$

$$\mathrm{V}\mathbf{u}^{\mathsf{T}} = \|\mathbf{u}\|(\mathrm{V}\mathbf{e}^{\mathsf{T}}) = 2\mathrm{E} - \lambda. \tag{11}$$

where

$$\mathbf{u} = \frac{1 - \cos(\alpha)}{\sin(\alpha)}\,\mathbf{e} \quad \text{and} \quad \lambda = \frac{k_1}{(1 - \cos(\alpha)).} \tag{12}$$

It is possible to show that the optimal rotation of a vector $\mathbf{v}$ can be computed by

$$\mathrm{R}_{\mathbf{e},\alpha}(\mathbf{v}) = \mathbf{v} - \frac{2(\mathbf{u} \times (\mathbf{v} \times \mathbf{u}) + (\mathbf{v} \times \mathbf{u}))}{1 + \|\mathbf{u}\|^2} \tag{13}$$

Finally, it can be proved that $\mathbf{u}$ is related to the quaternion $\mathbf{q}$ of the optimal rotation $\mathrm{R}_{\mathbf{e},\alpha}$ by the Equation $\mathbf{q} = \cos(\alpha/2)(1, \mathbf{u})$.

Now using Equation 11 to eliminate λ from (10), so that $\mathbf{u}$ becomes the only unknown, we obtain

$$(\mathrm{N} - (2\mathrm{E} - \mathrm{V}\mathbf{u}^{\mathsf{T}})\mathrm{I})\mathbf{u}^{\mathsf{T}} = \mathrm{V}^{\mathsf{T}}, \tag{14}$$

which is non-linear.

Now, let us consider a typical application where the deformation process is performed interactively. In this case, the user employs some GUI to drag one or more control points while examining the intermediate results. This means that the deformation is computed repeatedly, with the result for each step differing from that of the previous step by a small amount. Let index k refer to the k^{th} step of such an interactive deformation session, for which equation (14) must hold, that is,

$$(\mathrm{N}_k - (2\mathrm{E}_k - \mathrm{V}_k\mathbf{u}_k^{\mathsf{T}})\mathrm{I})\mathbf{u}_k^{\mathsf{T}} = \mathrm{V}_k^{\mathsf{T}}. \tag{15}$$

For sufficiently high sampling rates, however, we may assume $\mathbf{u}_{k-1} \sim \mathbf{u}_k$. This suggests a linearization of (15), allowing us to obtain $\mathbf{u}_k$ directly from

$$(\mathrm{N}_k - (2\mathrm{E}_k - \mathrm{V}_k\mathbf{u}_{k-1}^{\mathsf{T}})\mathrm{I})\mathbf{u}_k^{\mathsf{T}} = \mathrm{V}_k^{\mathsf{T}}, \tag{16}$$

where the initial solution $\mathbf{u}_0$ is set at 0.

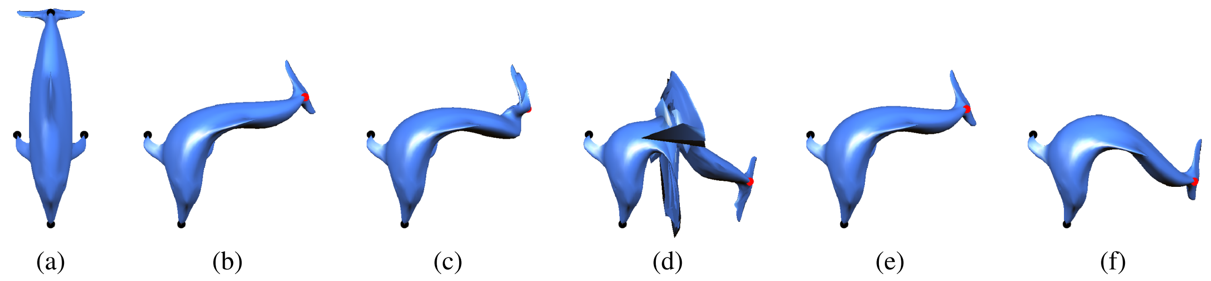

Figure 2. Reinitializing the process. The method works correctly if det$(\mathbf{M}) > 0$ (a) and (b), but results in distortions for det$(\mathbf{M}) \leq 0$ (c) and (d). Automatic reinitialization yields correct results (e) and (f).

3.4. Correlation matrix updating

The components of system (16) $\mathrm{N}_k, \mathrm{E}_k, \mathrm{V}_k$ can all be obtained from de correlation matrix M_k. In relation to the cost of computing that matrix we observe that in the common case where some control points are translated by the same vector – d_k in iteration k – that matrix can be obtained by adding to M_0 the result of an unique tensor product $d_k \otimes \sum_{i \in \mathcal{M}} w_i \hat{\mathbf{p}}_i$ where $\mathcal{M} = \{i \quad | \quad \mathbf{p}_i$is non-fixed$\}$.

Thus, we see that, at each step, a new solution may be obtained very cheaply from the solution computed in previous step. Essentially, the work needed for each vertex is $O(n)$, where n is the number of control points being moved. This is similar to the complexity of [7]. Note, however, that the constant work needed for each vertex is dominated by the solution of a 3×3 linear system. This means that the computational effort is only a fraction of the cost of exact approaches which require the computation of eigenvalues of the correlation matrix M and its eigenvectors [3, 8, 9, 11].

Incidentally, we have tried other ways for linearizing (15) with less convincing results. For instance, $\mathbf{u}_k$ can be obtained by adding an increment to $\mathbf{u}_{k-1}$ which is determined by solving a first order approximation of (15). In that case, however, the average error is not reduced and the performance becomes worse since there are more variables to be computed.

We conducted a simple experiment to assess how the error between the exact and approximate solutions accumulates during the deformation process of the position of the vertices during a 10-step deformation of the dolphin model (see Figure 2(a)). We have observed that the biggest maximum error value corresponds to an angular difference smaller than $3°$, while the biggest average error is smaller than $1°$.

3.5. Reinitializing the process

A deformation session usually assumes an initial model on which control points $\{\mathbf{p}_i\}$ are placed and some sort of user interface for establishing the displaced positions $\{\mathbf{q}_i\}$ of the control points. It is reasonable to think that after a few interactions, the user may choose to restart the process by using the deformed model as an initial model on which further deformation is to be applied. Alternatively, such a reinitialization be done automatically by the system. At any rate, the current values $\{\mathbf{q}_i\}$ then become the initial values $\{\mathbf{p}_i\}$ and the corresponding weights $w_i(\mathbf{v})$ must be recomputed.

It should be clear, however, that frequent restarts will end up erasing the memory of the initial model, which may be regarded as contradictory with the idea of "as-rigid-as-possible" deformations. On the other hand, keeping the initial values of $w_i(\mathbf{v})$ during the whole deformation process may cause undesirable twisting effects as illustrated in Figure 2. These effects are due to the discontinuities in the rotation function $\mathrm{R}_{\mathbf{e},\alpha}(\cdot)$. However, since these discontinuities can only occur if $\det(\mathrm{M}) \leq 0$, they can be totally eliminated by checking the sign of $\det(\mathrm{M}(\mathbf{v}))$ for all $\mathbf{v}$ and restarting the process if a non-positive value is found. Thus the computation of sign(det(M($\mathbf{v}$))) must be efficient so that the overall performance of the system is not impacted. This can be accomplished in the case of the most common interaction which corresponds to keeping some control points fixed and displacing the others by the same vector. In this case sign(det(M($\mathbf{v}$))) can be determined with only 3 multiplications, provided that some auxiliary values are computed in a preprocessing step.

4. Skeleton-driven mesh deformation

The scheme described in the earlier sections produces a *space* deformation, i.e., it defines a $\Re^3 \rightarrow \Re^3$ mapping. The modeling of a particular deformation is influenced merely by a discrete set of $\{\mathbf{p}_i, \mathbf{q}_i\}$ pairs. As such, any model immersed in 3-space will be subject to the same deformation, with no consideration to its shape. This is unfortunate in many important applications – character animation, in special – where the desired deformation must

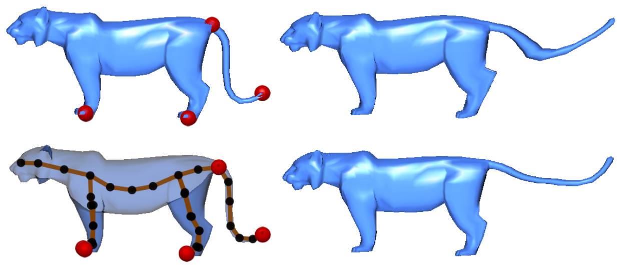

Figure 3. MLS-based space (top) and skeleton-driven (bottom) deformation.

take into account model peculiarities such as overall shape, bone structure and the position and geometry of articulations. As an illustration, contrast the unnatural deformation shown in Figure 3(b) with the more reasonable deformation exhibited in Figure 3(c). In this section, we describe a scheme whereby as-rigid-as-possible space deformations can be made to adapt to mesh models by combining them with skeleton-based animation techniques. Our goal is to obtain realistic model poses specified by suitably placing a small set of control points.

Traditional methods used in character animation for obtaining deformed models involve the creation of a "skeleton", i.e., an articulated structure of line-segment "bones", which are used to set the model pose. Additionally, surface mesh elements (vertices, mostly) must be assigned to individual bones, thus making the model shape conform to the skeleton pose. The term *rigging* is frequently used to refer to the process of creating a skeleton and associating its bones to surface parts. On the other hand, the smoothness of the skeleton-induced deformation must be assured by some interpolation scheme, a process known as *skinning*.

Both the creation of the skeleton and the assigning process may be done manually by the animation artist or by means of some automatic or semi-automatic process [6, 14, 17, 21, 22]. In the method we propose for achieving mesh deformation, skeletons are not manipulated directly by the animator, rather, they are used as a medium for estimating distances in a shape-sensible way. In a nutshell, we employ a variation of the MLS-based deformation algorithm in which Euclidian metrics are replaced by a skeleton-based metric. This algorithm is applied to a few points of the skeleton, namely, on the joints. The transformations of the joints are then transmitted to all vertices of the model surface by means of an interpolation scheme.

4.1. Rigging

We propose a rigging scheme which supports both user-defined and automatically constructed skeletons. Once the skeleton is known, we define the skin S_i of a bone b_i as the union of two sets of mesh vertices: The *primary skin* P_i of a bone b_i is the set of mesh vertices $\mathbf{v}$ such that, among all bones of the skeleton which are *visible* to $\mathbf{v}$, b_i is the closest. By visible, it is meant that a line segment from $\mathbf{v}$ to the closest point in b_i does not intersect the mesh. The *secondary skin* R_i of a bone b_i is the set of vertices for which bone b_i is the solution of the problem

$$\min_{b_j} g(\mathbf{v}, P_j), \tag{17}$$

where $g(a, X)$ is the *geodesic* distance between a point a and a set of points X. Recall that the geodesic distance between two points of a surface is the length of the shortest path on the surface which connects them.

It should be mentioned that exact algorithms to compute these sets are fairly costly. Thus, rather than computing exact geodesic path lengths, we restrict paths to be composed of mesh edges. This makes it possible to employ Dijkstra's path length algorithm starting from vertices on the border of primary skin sets to find secondary skin vertices.

Similarly, visibility determination using, say, a ray-casting algorithm is unduly expensive for the task at hand. Rather, we use a visibility diffusion algorithm which employs a "local" visibility property. A bone b_i locally visible to a vertex $\mathbf{v}$ if the angle between the (inwards-facing) normal at $\mathbf{v}$ and the vector from $\mathbf{v}$ to the closest point of b_i is smaller than 90 degrees (we assume that bones are always placed inside the mesh). Let $L(b_i)$ be the set of mesh vertices locally visible to b_i and which are closer to b_i than to any other bone b_j to which it is also locally visible. Then, the algorithm for finding the primary skin of bone b_i is the following:

1. Let $P(b_i)$ be $\{$**seed**$\}$, where **seed** is the vertex of $L(b_i)$ which is closest to b_i.
2. Search $L(b_i) - P(b_i)$ for a vertex $\mathbf{v}'$ such that $\mathbf{v}'$ is an edge neighbor of some vertex $\mathbf{v} \in P(b_i)$.
3. If such a vertex is found, place it in $P(b_i)$ and repeat step (2). Otherwise, stop.

This algorithm is very quick, as finding edge neighbors and testing for local visibility are constant-time operations. Eventually, other seeds must be found by an elementary ray-casting method and the process above repeated for them. We remark that the context of that ray-casting process is constrained to a neighborhood of b_i. In our experience, even if a vertex that should belong to a primary skin has not been found, in most of the times, the algorithm to compute secondary skins has assigned that vertex to the correct bone. Moreover, it should be stressed that the skinning process used for deforming the model is fairly robust to vertex assignment errors. Figure 4 illustrates the primary and secondary skins.

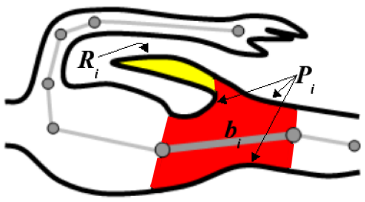

Figure 4. Rigging. the primary skin P_i of b_i is the border of the red region, and the secondary skin R_i is the border of the yellow region.

4.2. Skinning

The skeleton-driven deformation process starts by applying as-rigid-as-possible transformations to the skeleton joints. This follows the same lines discussed in Section 3, except that Euclidian distances are replaced by path distances along the skeleton. The joint transformations, in turn, will define the transformations along the connected bones using linear interpolation. If we consider that a bone b_i contributes to the transformation of a vertex $\mathbf{v}$ by a factor proportional to $\rho(b_i, \mathbf{v})$, then that contribution is $\rho(b_i, \mathbf{v}) \cdot T(n_i(\mathbf{v}))$, where $n_i(\mathbf{v})$ is the point of bone b_i closest to $\mathbf{v}$ and $T(x)$ stands for the transformation of a point x of a bone.

All it remains is to find a weighting scheme capable of ascertaining the smoothness of the deformed mesh. Let $h : [0, \infty) \rightarrow [0, 1]$ be any smooth function such that $h(0) = 1$, h decreases in $[0, r]$ and is null in $[r, \infty)$ for a given value $r > 0$ which acts as the maximum amplitude of a diffusion process. Thus, the weight $\rho(b_i, \mathbf{v})$ is given by $\rho(b_i, \mathbf{v}) = h(g(\mathbf{v}, S_i))$. Possible choices for h in $[0, r]$ are:

$$h(x) = \frac{G_{r/3}(x) - G_{r/3}(r)}{G_{r/3}(0) - G_{r/3}(r)}, \tag{18}$$

where G_σ is the Gaussian function relative to the normal distribution of average 0 and standard deviation σ or

$$h(x) = (1 - \frac{x}{r})^2, \tag{19}$$

which was most often used in our experiments due to its simplicity.

In our implementation, vertex weights are computed by extending the process used to obtain secondary skins, i.e., visiting all mesh points with a geodesic distance smaller than r from the border of S_i.

Finally, the transformation applied on any given vertex $\mathbf{v}$ is given by

$$T(\mathbf{v}) = \frac{\sum_{b_i} \rho(b_i, \mathbf{v}) T(n_i(\mathbf{v}))}{\sum_{b_i} \rho(b_i, \mathbf{v})}. \tag{20}$$

We conclude by observing that this formulation can be made more efficient by expressing the vertex transformation directly as a function of the transformations of the joints.

5. Implementation and Results

Our prototype was implemented in C++ and OpenGL under Linux. All the experiments were rendered on screens with 600 $\times$ 600 pixels. Times have been taken on a PC equipped with a Pentium-IV Core 2 Duo processor running at 2 $\times$ 2.13 GHz, with 2GB of main memory and a NVidia GeForce 7600GT graphics card.

The algorithm inputs are the model mesh, with n_v vertices, and the control point sets $\{\mathbf{p}_i, \mathbf{q}_i\}$, with cardinality n_c. The implementation of the deformation algorithm is divided in two phases, a setup phase and an interaction phase. The first phase consists of pre-computing several tables. It takes place just after the first step of the interaction, i.e., for $k = 1$. In the approximate deformation case, the following values are pre-computed:

(1) The table of weights w_{ij} of size $n_v \times n_c$.

(2) The vector table $\hat{\mathbf{p}}_{ij}$ of size $n_v \times n_c$.

(3) The vector array $\hat{\mathbf{v}}_i$ of size n_v.

(4) The array of matrices $\{\mathrm{M}_i\}$ of size n_v.

(5) The vector array $\{\mathbf{u}_i\}$ of size n_v.

(6) The vector array $\hat{\mathbf{p}}_{i \in \mathcal{M}}$, where $\mathcal{M}$ is the set of non-fixed constraint points.

The initialization code is a direct implementation of the formulae presented in Section 3. Recall that the values of $\mathbf{u}_i$ are initially set to 0. Observe that items (4), (5) and (6) are not necessary for the exact deformation algorithm.

All the experiments involving skeleton-driven deformation use the exact algorithm. For these, however, the rigging procedure must be included in the pre-processing phase. If the data structure for storing the skeleton has n_j joints, then the process stores for each mesh vertex an array with n_j weights ρ. Notice that only the skeleton joints are actually deformed in an as-rigid-as-possible way, i.e., $n_v = n_j$.

The second phase performs the deformation in every frame after the user changes the control points positions $\{\mathbf{q}_i\}$. Notice that values (1), (2), and (3) are constant during the interaction. For the approximate deformation each new step in the interaction starts by filling up variables (4) and (5) with the new values for $\{\mathbf{q}_i\}$ and $\{\Delta\mathbf{q}_i\}$.

Experiments

The first batch of experiments consisted of deforming the Armadillo model in several resolutions using: (i) the exact method of [7], (ii) the approximate method using only the

Vertices	CtrlPoints	Exact	+M	+u	+M+u
50K	5	22	31	39	62.5
100K	5	10.5	15	18	29
165K	5	6.5	9	11.5	17.5
50K	15	13	31	18	62.5
100K	15	6.5	15	9.5	29
165K	15	4	9	6.0	17.5

Table 1. Frame rates for the approximate deformation of the Armadillo model.

Model	Vertices	Joints	Ctrls	Exact	r_0	r_1	r_2
Armadillo	165K	33	5	6.5	17	18	19.5
Elephant	185K	29	5	5	13	14	16
Hand	195K	32	6	4.5	12	13	14.5
Dragon	247K	26	5	4	10	11	12

Table 2. Deformation frame rates using the skeleton-driven scheme.

continuous updating of M_i, (iii) the approximate method using only the continuous updating of $\mathbf{u}_i$, and (iv) the approximate method with continuous updating of both M_i and $\mathbf{u}_i$. The frame rates obtained in these experiments are shown in Table 1. In general, the rendering speed increases roughly $3\times$ when we contrast the exact method with the approximate method. Moreover, notice that the continuous updating of M_i does help more if the number of constraints is not small.

The second batch of experiments compare space deformations with skeleton-driven mesh deformations. Table 2 presents frame rates obtained with both deformation algorithms for the models in Figure 5, where the factor r which controls the diffusion process was varied between 0.1 and 0.3. We notice that the skeleton-driven deformation scheme is approximately $3\times$ faster than the space-based scheme. This was expected, given that the costly as-rigid-as-possible deformation is applied to much fewer vertices in the former case. The influence of r can be explained by the fact that smoother deformations require that more joints be taken into account for many vertices.

Another important consideration is that, in all experiments, vertex normals were merely transformed from their values in the original model, rather than re-estimated from face normals at each frame.

6. Conclusions and future work

This work describes two alternatives aimed at making 3D "as-rigid-as-possible" MLS a competitive option for deforming mesh models. In a purely algebraic context, we have proposed a method for obtaining approximate solutions without solving more costly eigenvalue/eigenvector problems. On the other hand, a skeleton-driven scheme was developped, transforming the original space-deforming algorithm into one which is more sensible to the geometry of meshes. As a side-effect, this approach is also more efficient, since only skeleton joints must be transformed directly by the MLS formulation.

The proposed skeleton-driven method employs rigging and skinning procedures which require a very simple set of operations – essentially, the computation of distances from vertices to bones and a progressive expansion process in the mesh graph.

Experiments show that these alternatives substantially increase the frame rates of interactive deformation sessions of models with up to a few hundred thousand vertices. In particular, the skeleton-driven framework is suitable for use in character animation.

We are currently working on GPU-based implementations of these algorithms. Early results indicate that a GPU implementation of the exact method improves it by factor of up to $2\times$. Two variants of the present work are being studied. One of them is to provide skeleton joints and bones with a cost structure to make it possible to express properties like stiffness or degrees of liberty. The other variant concentrates in building a more natural user interface suitable for character animation.

References

[1] M. Alexa. Interactive shape editing. ACM SIGGRAPH Courses, 2006.

[2] M. Alexa, D. Cohen-Or, and D. Levin. As-rigid-as-possible shape interpolation. *Proceedings of SIGGRAPH 2000*, pages 157–164, July 2000. New Orleans, Louisianna USA, July 23-28.

[3] K. S. Arun, T. S. Huang, and S. D. Blostein. Least-squares fitting of two 3-d point sets. *IEEE Trans. Pattern Anal. Mach. Intell.*, 9(5):698–700, 1987.

[4] O. K.-C. Au, H. Fu, C.-L. Tai, and D. Cohen-Or. Handle-aware isolines for scalable shape editing. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2007)*, 26(3):to appear, 2007.

[5] P. J. Besl and N. D. McKay. A method for registration of 3-D shapes. *IEEE Transactions on Pattern Analysis and machine Intelligence*, 14(2):239–258, Feb. 1992.

[6] N. D. Cornea, D. Silver, and P. Min. Curve-skeleton properties, applications, and algorithms. *Visualization and Computer Graphics, IEEE Transactions on*, 13(3):530–548, May-June 2007.

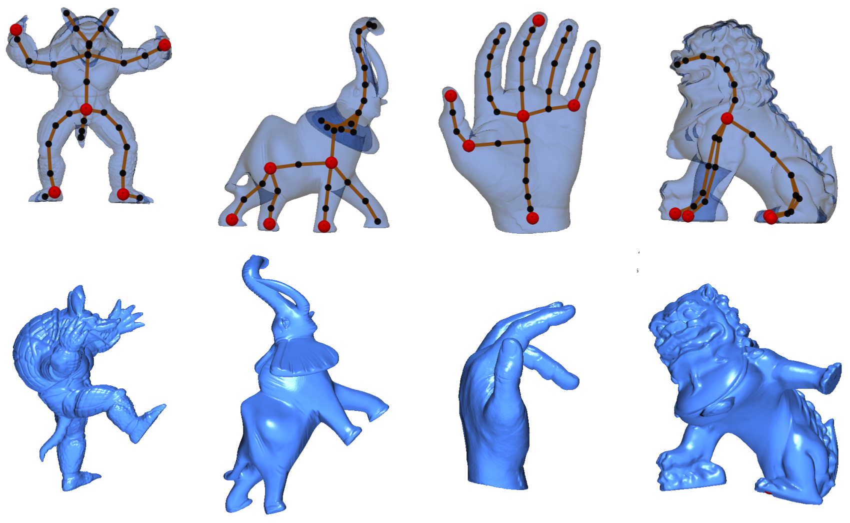

Figure 5. Deformation examples.

[7] A. Cuno, C. Esperança, A. Oliveira, and P. R. Cavalcanti. 3D as-rigid-as-possible deformations using MLS. In *CGI 2007: Proceedings of the 27th Computer Graphics International Conference*, page to appear, 2007.

[8] B. K. P. Horn. Closed form solutions of absolute orientation using unit quaternions. *Journal of the Optical Society of America*, 4(4):629–642, Apr. 1987.

[9] B. K. P. Horn, H. M. Hilden, and S. Negahdaripour. Closed form solutions of absolute orientation using orthonormal matrices. *Journal of the Optical Society of America*, 5(7):1127–1135, 1988.

[10] J. Huang, X. Shi, X. Liu, K. Zhou, L.-Y. Wei, S.-H. Teng, H. Bao, B. Guo, and H.-Y. Shum. Subspace gradient domain mesh deformation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Papers*, pages 1126–1134, New York, NY, USA, 2006. ACM Press.

[11] K. Kanatani. Analysis of 3-d rotation fitting. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(5):543–549, 1994.

[12] L. Kavan, S. Collins, J. Zara, and C. O'Sullivan. Skinning with dual quaternions. In *I3D '07: Proceedings of the 2007 symposium on Interactive 3D graphics and games*, pages 39–46, New York, NY, USA, 2007. ACM Press.

[13] L. Kobbelt, S. Campagna, J. Vorsatz, and H.-P. Seidel. Interactive multi-resolution modeling on arbitrary meshes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 105–114, New York, NY, USA, 1998. ACM Press.

[14] J. P. Lewis, M. Cordner, and N. Fong. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH '00: Proceedings of the 257th annual conference on Computer graphics and interactive techniques*, pages 165–172, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[15] M. Marinov, M. Botsch, and L. Kobbelt. Gpu-based multiresolution deformation using approximate normal field reconstruction. *ACM Journal of Graphics Tools, 2007, to appear*, 2007.

[16] L. K. Mario Botsch. Multiresolution surface representation based on displacement volumes. *Computer Graphics Forum*, 22(3):483–491, 2003.

[17] A. Mohr and M. Gleicher. Building efficient, accurate character skins from examples. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 562–568, New York, NY, USA, 2003. ACM Press.

[18] S. Schaefer, T. McPhail, and J. Warren. Image deformation using moving least squares. *ACM Trans. Graph.*, 25(3):533–540, 2006.

[19] N. M. Thalmann, F. Cordier, H. Seo, and G. Papagianakis. Modeling of bodies and clothes for virtual environments. In *CW '04: Proceedings of the 2004 International Conference on Cyberworlds (CW'04)*, pages 201–208, Washington, DC, USA, 2004. IEEE Computer Society.

[20] M. W. Walker, L. Shao, and R. A. Volz. Estimating 3-D location parameters using dual number quaternions. *CVGIP: Image Understanding*, 54(3):358–367, Nov. 1991.

[21] X. C. Wang and C. Phillips. Multi-weight enveloping: least-squares approximation techniques for skin animation. In *SCA '02: Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 129–138, New York, NY, USA, 2002. ACM Press.

[22] H.-B. Yan, S.-M. Hu, and R. Martin. Skeleton-based shape deformation using simplex transformations. In *Computer Graphics International*, pages 66–77, 2006.