Mesh Puppetry: Cascading Optimization of Mesh Deformation with Inverse Kinematics

Xiaohan Shi* Kun Zhou† Yiying Tong‡ Mathieu Desbrun‡ Hujun Bao* Baining Guo†
*State Key Lab of CAD&CG, Zhejiang University †Microsoft Research Asia ‡Caltech

Abstract

We present *mesh puppetry*, a variational framework for detail-preserving mesh manipulation through a set of high-level, intuitive, and interactive design tools. Our approach builds upon traditional rigging by optimizing skeleton position and vertex weights in an integrated manner. New poses and animations are created by specifying a few desired constraints on vertex positions, balance of the character, length and rigidity preservation, joint limits, and/or self-collision avoidance. Our algorithm then adjusts the skeleton and solves for the deformed mesh simultaneously through a novel cascading optimization procedure, allowing realtime manipulation of meshes with 50*K*+ vertices for fast design of pleasing and realistic poses. We demonstrate the potential of our framework through an interactive deformation platform and various applications such as deformation transfer and motion retargeting.

Keywords: Mesh deformation, nonlinear optimization, inverse kinematics, geometry processing.

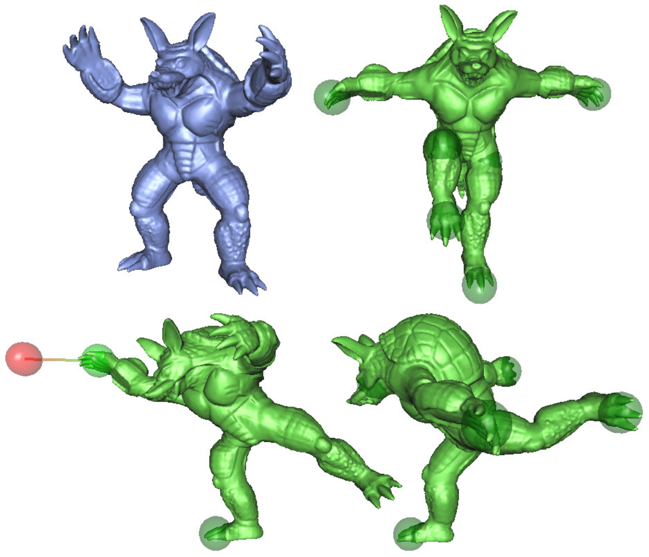

Figure 1: *Armadillo Olympics: The Armadillo model (top left) can be deformed to take various sport poses in a matter of seconds. Its body automatically leans forward and raises its left leg backward to keep balance when trying to reach the user-specified (red) target (shot put, bottom left). Fixing only the positions of its hands and feet is enough to make the Armadillo look like it is bouncing off a springboard (high diving, top right); or the pose of a sprint athlete on the finish line (100m, bottom right). With simple, high-level constraints on length, balance, and joint angles, our framework offers an* intuitive and realtime tool *to design pleasing and/or realistic poses while preserving fine-scale geometric details.*

1 Introduction

Recent years have seen the emergence of detail-preserving mesh deformation techniques [Sorkine et al. 2004; Yu et al. 2004; Lipman et al. 2005; Zhou et al. 2005; Botsch et al. 2006], offering a powerful alternative to free-form deformation for direct manipulation of high-quality meshes. However, *rigging* (*i.e.*, adding a skeleton to control and animate a mesh) remains a preferred way in the graphics industry to efficiently design poses and deformation: a properly-designed skeleton can be quickly and easily manipulated to deform the posture of a character, then skinning (or "binding") is used to match the fine-detail shape of the mesh to the bones' positions. While these approaches to mesh deformation both have their pros and cons, neither allows for interactive design of high-quality, large-scale or fine-scale deformation of detailed meshes. In this paper, we propose to combine mesh deformation techniques and skeleton-based rigging to offer *mesh puppetry*, a fast, intuitive, and general tool for mesh manipulation and motion retargeting. At the core of our method is a novel optimization procedure which enforces the user-selected deformation constraints onto a mesh through a cascading, multi-threaded process—making mesh puppetry viable even on regular PCs.

†e-mail:{kunzhou,bainguo}@microsoft.com
‡e-mail: {yiying,mathieu}@caltech.edu

1.1 Related Work

Our approach is related to a long series of work on mesh and skeletal deformation; we only cover the most relevant references below.

Mesh Deformation Early detail-preserving mesh deformation techniques were based on multi-resolution techniques [Zorin et al. 1997; Kobbelt et al. 1998; Guskov et al. 1999], offering mostly local shape control. To allow for more global and complex deformation, many authors proposed to cast mesh deformation as an energy minimization problem [Sorkine et al. 2004; Yu et al. 2004; Lipman et al. 2005; Nealen et al. 2005; Zayer et al. 2005; Zhou et al. 2005; Lipman et al. 2006; Shi et al. 2006]. Typically, the energy functionals used in these techniques contain terms to preserve detail (often through Laplacian coordinates), as well as position-constraint terms to allow for direct manipulation. Introducing more terms in the optimization (*e.g.*, volume or skeleton constraints) was even advocated in [Huang et al. 2006] as a convenient way to design more complex deformation with ease, without the traditional shearing artifacts appearing in large scale deformation. However, these existing techniques do not currently scale: the optimizations involved are often nonlinear and require slow-converging Gauss-Newton iterations. This limitation can be overcome through a coarser mesh embedding (using, *e.g.*, mean value coordinates [Ju et al. 2005]) at the price of significantly less design control. Recent progress proposed to handle deformation via a mesh-based Inverse Kinematics, and to "learn" the space of natural deformations from a series of example meshes [Sumner et al. 2005; Der et al. 2006], enhancing the efficiency of deformation design by restricting the results to acceptable ones. Our paper offers a quite different solution to a similar problem by removing the need for example meshes and by formulating a large set of (mostly nonlinear) constraints to define the kinematic behavior of the character.

Skeleton Subspace Deformation and Rigging Skeleton Subspace Deformation (SSD) [Magnenat-Thalmann et al. 1988] and several variants have been used in the graphics industry for quite some time as a natural and efficient representation for character animation in games and films. However, while their success lies in restricting deformation to a particular subspace for efficiency, the characteristic "collapsing joint" defect (see Figure 7) as well as the tedious tweaking of vertex weights are well known shortcomings. Despite significant improvements [Lewis et al. 2000; Mohr et al. 2003] to allow for easier mesh manipulation and interpolation, these rigging tools do not allow for fast design of really complex deformation.

Inverse Kinematics Inverse Kinematics (IK) techniques are now very mature, offering robust and efficient solutions to on-line or off-line *postural control* of complex articulated figures [Badler et al. 1987; Zhao and Badler 1994; Boulic et al. 1997]. Unlike mesh deformation techniques that focus on shape and detail preservation, IK allows efficient design of *large-scale deformation of skeletal figures* through optimization of an energy functional in the null space of constraints. Current methods [Yamane and Nakamura 2003; Baerlocher and Boulic 2004; Le Callennec and Boulic 2006] even handle the fulfillment of conflicting constraints through prioritization, offering robust posture design.

1.2 Approach and Challenges

Rigging is extremely practical when it comes to designing mesh motions. Indeed, a skeleton provides a very natural and intuitive underlying structure to the mesh: the skeleton is "carrying" the mesh like skin around bones. Each bone in the skeleton influences (at least locally) the shape and position of the mesh such that, as the skeleton is animated, the surrounding mesh geometry moves accordingly and reflects the current pose of the skeleton. In practice, a "skinned mesh" is animated by simply moving its skeleton, and the mesh follows—often with unpleasant consequences (local artifacts), particularly near joints. On the contrary, mesh deformation based on functional optimization and differential coordinates is usually applied to the design of high-quality *static* poses. Although the resulting meshes can be made artifact-free (with nonlinear energy minimization), the time complexity involved in practice inhibits direct manipulation for animation. In this paper, we thus propose to mix these two approaches into a unified framework. Our mesh puppetry approach uses an approximate skeletal structure of a mesh to enable fast design of large scale deformation while optimizing the induced local deformation to maintain small-scale features.

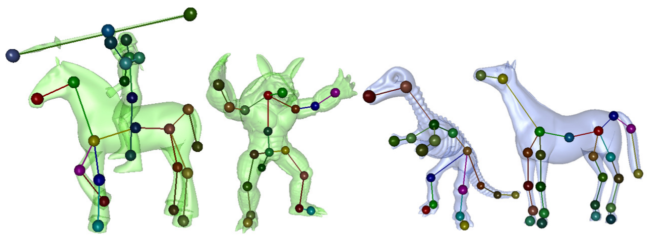

Figure 2: *Reference meshes and skeletons used in this paper.*

However, this task presents many challenges. First, a deformation energy involving variables as different as vertex positions and bone transformations is delicate to formulate. Second, traditional IK constraints are highly nonlinear, sometimes even involving inequalities. Finally, using existing mesh deformation solvers to solve these nonlinear constraints would, even with multigrid strategies, lead to poor performance, ruling out interactive applications.

1.3 Contributions

In this paper we present *mesh puppetry*, a variational framework for detail-preserving mesh deformation through a set of high-level, intuitive design tools. Although we make use of a simplified skeletal representation of the mesh in our algorithm to speed up the treatment of large deformation, the user *directly manipulates the mesh*. New poses and animations are created by specifying a few desired constraints on either vertex positions, balance of the character, length and rigidity preservation, or joint limits. Given these constraints, our algorithm adjusts the skeleton and solves for the deformed mesh simultaneously through a novel cascading optimization procedure, allowing realtime manipulation of meshes with 50K+ vertices. These contributions result in interactive mesh puppetry as the high-level constraints defined in this paper induce pleasing and realistic poses (see Figure 1). We demonstrate the potential of our framework through an interactive deformation system and several applications such as animation transfer and motion retargeting.

2 Set-Up and Nomenclatures

We assume that the user has, as an input, a reference (triangle) mesh $\mathcal{M}$ (with V vertices) and a rough skeleton $\mathcal{S}$ of this mesh (represented as a graph made out of B bones, see Figure 2). We also assume that, along with the skeleton, the corresponding partition of the mesh is known so that every vertex has the indices of the bones it is associated with. Note that if only the mesh is known, existing skeletonization techniques (*e.g.*, [Teichmann and Teller 1998]) can provide both the skeleton and the corresponding partition automatically. Alternatively, the user can manually create the skeleton and the corresponding partition very efficiently (the camel's skeleton in Figure 19 was created in three minutes from scratch).

2.1 Skinned Mesh

We denote by $\bar{\mathbf{x}}_i$ the (3D) position of a vertex of the reference mesh $\mathcal{M}$ and by $\overline{\mathbf{X}} = \{\bar{\mathbf{x}}_i, i \in [1..V]\}$ the vector of all these positions. We will be looking for a *deformed* pose of $\mathcal{M}$, i.e., for another mesh with the exact same connectivity, but a new set of vertex positions $\mathbf{X} = \{\mathbf{x}_i, i \in [1..V]\}$. To express these mesh coordinates, we use the now traditional skinned mesh setup: the deformation is encoded as one affine transformation $\mathbf{T}_b$ per bone, and a $B \times V$ matrix of weights $\mathbf{W} = \{w_{bv}, v \in [1..V], b \in [1..B]\}$, such that:

$$\mathbf{x}_i = \Big(\sum_{b \in \text{bones}} w_{bi} \mathbf{T}_b \Big)\, \bar{\mathbf{x}}_i. \qquad (1)$$

Each vertex position is thus defined as a linear combination of the locations where the vertex would be if it was only following the transformation imposed by a neighborhood bone. If we call $\mathbb{T}$ the row vector of all matrices $\mathbf{T}_b$, the deformed mesh $\mathbf{X}$ is concisely expressed through:

$$\mathbf{X} = \mathbb{T}\mathbf{W}\overline{\mathbf{X}}.$$

Note that the matrix $\mathbf{W}$ is extremely sparse: a vertex only has blending weights to its associated bone and possibly to the direct neighboring bone(s) if the vertex is near a joint.

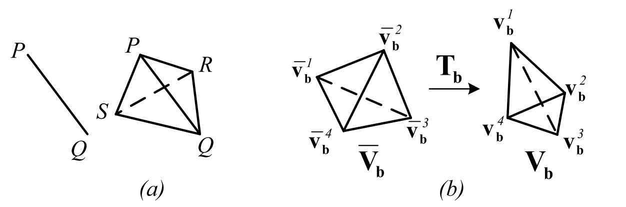

Figure 3: *(a) Tetrabone: given a bone $b = PQ$ (P and Q being joints), we introduce two points R and S such that the tetrahedron (P,Q,R,S) is regular. This tetrahedron, associated to bone b, is called its tetrabone $\mathcal{T}_b$. (b) Bone Transformation: the bone b and its associated tetrabone is in its original pose $\overline{V_b}$ (on the left), and in its deformed pose V_b (on the right). The tetravertices can be used to capture any affine transformation $\mathbb{T}_b$ that the bone undergoes.*

2.2 Tetrabones to Encode Transformation

In order to optimize position only (instead of both positions and transformation, thus avoiding the use of trigonometric functions in the optimization procedure), we associate to each bone two additional, virtual vertices so as to create a non-degenerate tetrahedron. The initial construction is trivial: for each bone $b = PQ$ (P and Q being joints), we introduce two points R and S such as the tetrahedron (P,Q,R,S) is *regular* (see Figure 3). This tetrahedron, associated to bone b, is called its *tetrabone* $\mathcal{T}_b$, and its vertices (that we generically denoted as P,Q,R and S so far) will be called *tetravertices* $(\bar{\mathbf{v}}_b^1, \bar{\mathbf{v}}_b^2, \bar{\mathbf{v}}_b^3, \bar{\mathbf{v}}_b^4)$ to avoid any confusion with mesh vertices.

With those tetrabones in place on the undeformed mesh $\mathcal{M}$, we can now capture any affine transformation $\mathbf{T}_b$ that a bone undergoes through the displacement of its tetravertices (see Figure 3). If we call $\overline{\mathbf{V}}_b$ the matrix with each column being the original homogeneous coordinates (*i.e.*, with 1 as the last coordinates) of the vertices $(\bar{\mathbf{v}}_b^1, \bar{\mathbf{v}}_b^2, \bar{\mathbf{v}}_b^3, \bar{\mathbf{v}}_b^4)$, and $\mathbf{V}_b$ a similar matrix but now with the homogeneous coordinates of the deformed tetrabone $(\mathbf{v}_b^1, \mathbf{v}_b^2, \mathbf{v}_b^3, \mathbf{v}_b^4)$, the following linear relation trivially holds:

$$\mathbf{V}_b = \mathbf{T}_b \overline{\mathbf{V}}_b.$$

Provided that $\overline{\mathbf{V}}_b$ is a full-rank matrix (since the initial tetrahedron is regular, thus non-degenerate), we simply get:

$$\mathbf{T}_b = \mathbf{V}_b \overline{\mathbf{V}}_b^{-1} \tag{2}$$

Substituting Eq.(2) into Eq.(1), we get

$$\mathbf{x}_i = \left(\sum_{b \in \text{bones}} w_{bi} \mathbf{V}_b \overline{\mathbf{V}}_b^{-1} \right) \bar{\mathbf{x}}_i.$$

Thus if we define $\mathbb{V}$ as the row vector of all $\mathbf{V}_b$ matrices and $\overline{\mathbb{V}}^{-1}$ as block-diagonal matrix of all $\overline{\mathbf{V}}_b^{-1}$, we now can write our skinned mesh setup as a function of only the mesh vertices and the tetravertices:

$$\mathbf{X} = \mathbb{V}\, \overline{\mathbb{V}}^{-1}\, \mathbf{W}\overline{\mathbf{X}}. \tag{3}$$

2.3 Objective Function

With this setup, we can define the variational problem that our mesh puppetry algorithm is based upon: we look for a deformed mesh $\mathcal{M}$ with vertex positions $\mathbf{X}$ (as a function of $\mathbb{V}$ and $\mathbf{W}$) such that it minimizes a *global deformation energy* $\mathcal{E}$:

$$\mathcal{M} = \underset{\mathbf{X}=\mathbb{V}\,\overline{\mathbb{V}}^{-1}\,\mathbf{W}\overline{\mathbf{X}}}{\arg\min}\ \mathcal{E}(\mathbf{X}), \tag{4}$$

where $\mathcal{E}$ encapsulates a set of constraints that the deformation must satisfy. We will show in the next section how to deal with convenient constraints that make the design of new poses easy and intuitive. In particular, we will use Laplacian constraint for the preservation of surface details (similar to, for instance, [Huang et al. 2006]); position constraints to allow direct manipulation of the mesh for intuitive design; and Inverse Kinematics constraints (on balance and bone lengths in particular) that guarantees natural poses *with minimal user interaction*.

3 Deformation Energy and Constraints

In this section, we elaborate on the constraints offered in our framework. Each constraint is implemented through the addition of an energy term to the global deformation energy. We also present inequality constraints (namely, the joint limit and the balance constraints), implemented as conditional energy terms.

3.1 General Constraints

The first and foremost energy term we will introduce in our deformation energy is the Laplacian constraint described in [Huang et al. 2006], using *Laplacian coordinates* [Sorkine et al. 2004] to preserve the surface details of the undeformed mesh. Another simple, yet essential constraint is to allow the user to assign a target position for either a mesh vertex or a linear combination of mesh vertices. The energy term we will use for each such position constraint in our solver is:

$$\left\| \mathbf{AX} - \widehat{\mathbf{X}} \right\|^2 \ \text{ with: } \mathbf{AX} = \frac{1}{\sum_{j \in \mathcal{I}} \alpha_j} \sum_{i \in \mathcal{I}} \alpha_i \mathbf{x}_i \tag{5}$$

where $\mathcal{I}$ is the set of indices corresponding to the vertices involved, the α_i's are weights, and $\hat{\mathbf{X}}$ is the target position. This formulation can be used for either vertex-position constraints or arbitrary point constraints. Note that these two constraints involve both tetrabones and mesh vertices. Next we describe energy terms depending solely on $\mathbb{V}$ or $\mathbf{W}$.

3.2 Constraints on Tetrabones Only

Constraints on tetrabones can be quite powerful at preventing a pose from being grossly distorted or kinematically unnatural.

Length Constraint Controlling the change in length of the bones is particularly useful when dealing with humanoid figures. We implement such a length constraint by adding an energy term of the form:

$$\sum_{(i,j) \in \text{bones}} \left(\|\mathbf{v}_i - \mathbf{v}_j\| - L_{ij} \right)^2 \tag{6}$$

where $\mathbf{v}_i$ and $\mathbf{v}_j$ are the position of joints i and j, respectively, and L_{ij} is the original (or more generally, the desired) length of bone

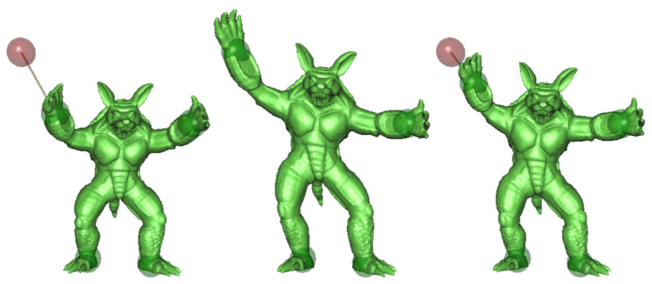

Figure 4: *Length Constraint: The Armadillo's right hand is dragged while its left hand and the feet are held fixed (left). Without length constraint, the body stretches out to fit the point constraint (middle). With length constraint, no stretching occurs (right).*

(i,j). Figure 4 shows an example of the effects of this constraint by comparing two poses, one without and one with this energy term. Although our length constraint resembles the skeleton constraint in [Huang et al. 2006], note that our formulation uses the skeleton instead of the mesh; we will also show in Section 4.2 that our solver will provide much improved convergence rates (factor 30) when dealing with this particular constraint.

Rigidity Constraint If we not only want the bones to keep their length, but also wish the skin around them to mostly deform as a rigid object, we can use a stronger version of the length constraint where we now have one length-based penalty term *per tetrabone edge*:

$$\sum_{(i,j)\in tetra(b)} \left(\|\mathbf{v}_i - \mathbf{v}_j\| - l_{ij} \right)^2$$

where $tetra(b)$ is the tetrabone b, $\mathbf{v}_i$ and $\mathbf{v}_j$ are the position of tetravertices i and j, respectively, while l_{ij} is the original distance between tetravertices i and j. Forcing each edge of the tetrabone to be of equal length amounts to imposing a *rigid body transformation* of the bone: it renders the deformation of each limb as rigid as possible. Figure 5 illustrates the effects of this constraint.

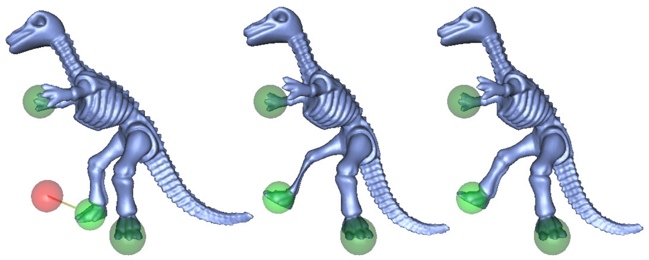
Figure 5: *Rigidity Constraint: The Dinosaur's right foot is dragged while its right hand and left foot are held fixed (left). Without rigidity, the right leg significantly shrinks (middle). With rigidity, the model recovers its normal leg (right).*

Balance Constraint A particularly nice tool for mesh puppetry is to automatically constrain a new pose to be physically realizable. To keep their balance, humanoid characters must have their center of mass above their supporting area (*e.g.*, above the convex hull of the vertices touching the ground). Enforcing this constraint guarantees a well-balanced, thus visually-plausible pose [Baerlocher and Boulic 2004]. Let's first examine how to achieve this balance constraint by forcing the projection of the barycenter onto the floor to a *given position* $\mathbf{g}$. We first precompute the barycenter and estimated "mass" of each bone based on the local volumes of the partition of the mesh in the original pose. The new barycenter, *i.e.*, the center of mass of a deformed pose, will thus be the mass-weighted sum of these barycenters once their corresponding affine transformations are applied. We implement this idea by adding the following energy term:

$$\|\mathbf{GV} - \mathbf{g}\|^2$$

where $\mathbf{G}$ is a matrix which maps $\mathbf{V}$, using the precomputed bone masses and a projection onto a given floor, to the location of the barycenter of the deformed mesh.

We found that a more general balance constraint, where the center of mass is *constrained to be over a given (convex) area* of the floor, is more helpful for fast design. Thus, we propose to turn this constraint on or off *within the solver*, depending on whether the current center of mass is over the desired area: when activated, the constraint uses a target location $\mathbf{g}$ which is the closest point from the current projection on the floor to the desired area. As Figure 6 demonstrates, such a simple constraint results in the ability to get natural poses with very little user input.

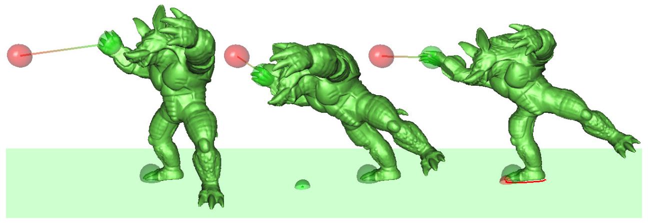
Figure 6: *Balance Constraint: a user drags the right hand of the Armadillo with its right foot held fixed (left). Without balance constraint, the Armadillo seems to lean excessively (middle). With balance constraint, the Armadillo (naturally) leans its body forward trying to reach the target point while (even more naturally) raising its left leg backward to keep its balance (right). The small green sphere shows the projection of the center of mass onto the floor, while the small red sphere shows the target position of barycenter at the border of the red polygon (demarcating the supporting area).*

Joint Limit Constraint Joints of skeletal structures often have restricted ranges. Such joint limit constraints can be enforced by simply introducing an energy term to the minimization process. We control the relative orientation between two adjacent bones by simply constraining the *distances* between their corresponding tetravertices. For two bones b_1 and b_2, if a joint angle is exceeding the prescribed limit, we add a term of the form:

$$\sum_{(i,j)\in \mathrm{pairs}(b_1,b_2)} \|(\mathbf{v}_i - \mathbf{v}_j) - \theta_{ij}\|^2$$

where $\mathbf{v}_i$ and $\mathbf{v}_j$ are the position of tetravertices from each bone, and θ_{ij} is the *target vector* between them to enforce the correct limit angle; $\mathrm{pairs}(b_1,b_2)$ denotes the few pairs of tetravertices of b_1 and b_2 required to resolve the joint limit currently violated. The pairs involved in this energy term depend on whether the joint limit is imposed laterally (rotation, see Figure 7), or axially (twisting, see Figure 8). We will detail these terms in Section 4.2.

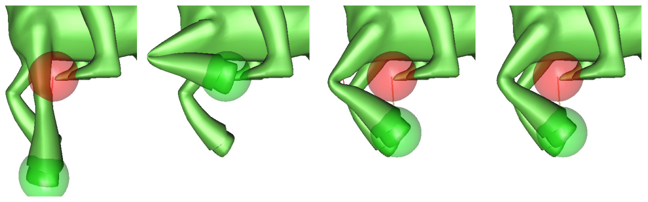
Figure 7: *Lateral Joint Limit: a user drags the horse's hoof up (left). Without joint limit constraint, the mesh self-intersects (second). Our joint limit constraint prevents the foreleg of the horse from bending too much; however, the joint looks collapsed (third). After our full-space postprocessing of the vertices near joints (see Section 4.4), the leg looks natural while the lateral angle limit is still maintained (right).*

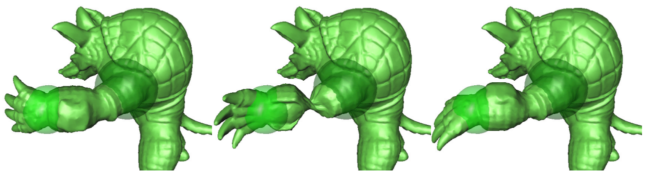
Figure 8: *Axial Joint Limit: the Armadillo's arm is twisted by the user with its shoulder fixed (left). Without this constraint, the arm twists excessively (middle). With the constraint, the twisting is automatically limited, and the arm looks natural.*

3.3 Constraints on Vertex Weights Only

Smooth dependence of the vertices on the transformation of the bones is finally added through an energy term that penalizes strong

```
$s = 0$
**repeat**
    // PERFORM ONE V-PHASE
    **for** $i = 1$ to v-step-cnt **do**
        // *s-th iteration in convergence sequence*
        $s{+}{+}$
        // *For each (active) level*
        **for** $k = 1$ to $\min(s,5)$ **do**
            // *m-th iteration on this level k*
            $m = s - k$
            // *Sum up first k constraints*
            $\mathcal{E} = \sum_{j=1}^{k} \mathcal{E}_j(m,k)$
            Linear solve to get $\mathcal{M}^{[m,k]} = \arg\min \mathcal{E}$
    // PERFORM ONE W-PHASE
    **for** $i = 1$ to w-step-cnt **do**
        $\mathcal{E} = \mathcal{E}_1(\mathcal{M}) + \mathcal{E}_W(\mathcal{M})$
        Linear solve to get $\mathcal{M} = \arg\min \mathcal{E}$
**until** (convergence criterion met)
```

Figure 9: *Pseudocode of our solver.*

local variations of the skin weights. This smoothness constraint, dependent *only* on $\mathbf{W}$, is enforced as follows:

$$\sum_{w_{bi}\in W}\left(w_{bi} - \frac{1}{|\mathcal{N}(i)|}\sum_{j\in\mathcal{N}(i)} w_{bj}\right)^2 + \sum_{i\in[1..V]}\left(\sum_{b\in B} w_{bi} - 1\right)^2 \qquad (7)$$

where i is the index of the vertex, b is the index of the bone and $\mathcal{N}(i)$ is the one-ring neighborhood of vertex i. The reader will notice that the first term of this energy effectively tries to annihilate the biLaplacian of the weights (by minimizing the Laplacian squared), a well-known way to regularize a scalar field. We opted not to use the geometric Laplacian matrix of the original mesh to avoid any numerical issue due to negative cotangents. Instead, we use the graph Laplacian, as it suffices to make the weights well-behaved without creating geometric artifacts. Notice that this smoothness constraint also takes care of the usual overfitting issues reported in, for instance, [James and Twigg 2005]. The last term of the energy is added to get the weights of the vertices around the joints to form a partition of unity. Although this condition is unnecessary for static poses, getting to be close to a unit sum for each vertex's weights will guarantee that if a rigid transformation is applied to the skeleton during modeling, the vertices will follow rigidly and thus will require no optimization.

4 Custom Solver through Cascading Optimization

Now that all the different energy terms have been described, one could directly apply a nonlinear solver (such as sequential quadratic programming) to the total energy in order to solve for the deformed mesh. Given the number of unknowns involved in the solve, this turns out to be quite a herculean task for large meshes—preventing any hope of realtime manipulation. In this section we present our optimization procedure, which exploits the structure of the problem to optimize efficiency.

4.1 Overview of Solver

We are looking for a deformed pose $\mathbf{X} = \mathbb{V}\overline{\mathbb{V}}^{-1}\mathbf{W}\bar{\mathbf{X}}$ such that $\mathbf{X}$ minimizes the energy functional we defined. Optimizing for $\mathbb{V}$ and $\mathbf{W}$ simultaneously is intractable, due to the number of unknowns involved and the highly nonlinear nature of the total energy $\mathcal{E}$. In order to reach interactive speed at least for moderately large meshes, we propose a custom optimization scheme (we give pseudocode in Figure 9).

Alternating Phases First, we leverage the fact that $\mathbf{X}$ is a linear function of $\mathbb{V}$ when $\mathbf{W}$ is held fixed, and a linear function of $\mathbf{W}$ when $\mathbb{V}$ is fixed, by adopting a Lloyd-like approach [Lloyd 1982]. That is, we find the optimal deformed mesh by alternating between optimizations of $\mathbb{V}$ with $\mathbf{W}$ fixed and of $\mathbf{W}$ with $\mathbb{V}$ fixed: each optimization phase is still a nonlinear problem, but with significantly less variables. We will call *V-phase* the optimization of $\mathbb{V}$ only, and *W-phase* the other optimization.

V-phases Note that some constraints involved in our optimization are more stringent than others; while the position constraint alone can be optimized through a simple linear solve, the other constraints are nonlinear, and the balance and joint limit constraints are inequality constraints. Moreover, constraints can be conflicting: some deal with *detail* preservation, while others control *large-scale* deformation. The mixed nature of these constraints makes any brute-force numerical attempt to optimize the total objective function in the V-phase inefficient. After trying several optimization strategies, we settled for a cascading objective optimization method, where objective increments are added in cascade to the optimization procedure to accelerate convergence as described in Section 4.2.

W-phases Contrary to a V-phase, a W-phase contains very few contraints (only the Laplacian, position, and smoothness constraints) that are straightforward to minimize. Hence, we perform a traditional iterative method to optimize this objective function.

Polishing Phase When high quality results are desired, a final phase is added to rectify the vertex positions around joints, as our SSD-based approach can suffer from the "collapsing joint" effect for extreme poses (see Figure 7).

Note finally that every iteration performed during this optimization procedure (be it for the V-phase or the W-phase) consists of a *simple linear solve*: as in the Gauss-Newton method and as advocated in [Huang et al. 2006], we deal with the nonlinear optimization using repeated quadratic approximations.

4.2 V-Phase: Cascading Objective Optimization

Our V-phase is devised to optimize the affine transformation (via the positions $\mathbf{V}$ of the skeleton's tetrabones) applied to each skeleton bone, in order to minimize the objective function containing (some of, or all) the constraints mentioned in Section 3.

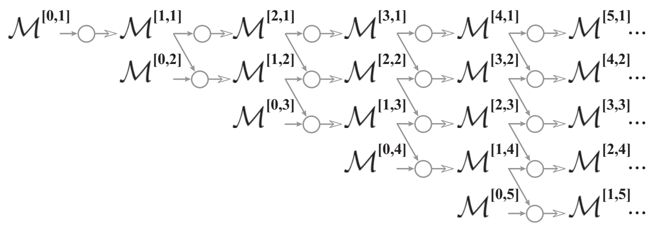

Figure 10: *Cascading Algorithm: each row represents the workflow of a thread. $\mathcal{M}^{[m,k]}$ denotes the result from the m-th state of thread k while the arrows indicate data dependency. A new state is solved for as soon as all its prerequisites are finished so as to exploit thread parallelism. Note that there are many ways to parallelize this process: a thread per line or per diagonal are the two most obvious parallelization techniques that this dependency graph calls for.*

Threads with Increasing Number of Constraints We divide the V-phase into several *threads*, in which increasingly-constrained optimizations are performed in parallel. Instead of using all the constraints at once, we first start a thread that evolves the mesh from its current shape towards satisfying *only the most basic constraints*. In parallel, we start a second thread of optimization which now uses the results of the first thread as initial conditions, and an objective function including the basic constraints *plus one more* constraint, and so on (see Figure 10). The final thread contains all the desired constraints, and will result in the final, deformed mesh (Figure 11 illustrates the results of each thread on a simple example).

This may, at first glance, seem like a lot of additional operations to perform compared to a regular optimization of the total energy; but optimizing fewer constraints is very efficient, and using the result of a previous thread to guide a later thread will *greatly benefit* the efficiency of the solver, making the optimization process more stable and efficient. We will show that this *cascading approach* accelerates convergence significantly, and is easy to implement on multicore processors. A byproduct of this cascading increment-objective approach is that the terms in earlier threads are better minimized, which allows us to define the order in which constraints are added according to their respective importance. Note that the idea of adding constraints one at a time during the optimization was already proposed in genetic algorithms [Chen and Guan 2004]; we pushed this idea further by using concurrent threads which feed on each other to accelerate convergence.

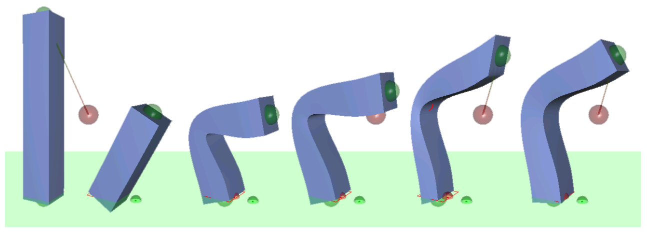

Figure 11: *Final States of Each Thread: from left to right, the deformation results of each thread for a 2-link stick being bent (rest pose, Laplacian and position constraints, balance constraint added, length constraint added, joint limit constraint added, and rigidity constraint added).*

Optimization Procedure Each optimization step is achieved by taking only partial sums of quadratic approximations of the various constraint energies, making use of the results of earlier steps to guide later steps. We now detail each of the five threads of our cascading optimization procedure described in Figure 9. We will denote the optimization result from the m-th state of thread k as $\mathcal{M}^{[m,k]}$, which contains the positions of all the mesh vertices $\mathbf{X}^{[m,k]}$ and of all the tetravertices $\mathbf{V}^{[m,k]}$.

- **1. Laplacian and Position Constraints** We implement Laplacian constraint by adding the following energy term into the deformation energy:

$$\mathcal{E}_1(m,k) = \left\| \mathbf{L}\mathbf{X}^{[m,k]} - \hat{\delta}\mathbf{X}^{[m-1,k]} \right\|^2 \qquad (8)$$

where $\hat{\delta}\mathbf{X}^{[m-1,k]}$ is defined (as in [Huang et al. 2006]) as:

$$\hat{\delta}\mathbf{X}^{[m-1,k]} = \frac{\mathbf{L}\mathbf{X}^{[m-1,k]}}{\left\|\mathbf{L}\mathbf{X}^{[m-1,k]}\right\|} \left\|\mathbf{L}\overline{\mathbf{X}}\right\|$$

where $\overline{\mathbf{X}}$ is the mesh in rest pose. This means that we use the Laplacian coordinates of the *previous* result of the same thread as the target direction, while taking the magnitude of the original Laplacian coordinates as the target magnitude. This is simply a linearization of the Laplacian constraint energy term, which will therefore help preserving the surface details of the undeformed mesh. As for position constraints, their implementation is achieved by adding:

$$\mathcal{E}_1(m,k) += \left\| \mathbf{A}\mathbf{X}^{[m,k]} - \widehat{\mathbf{X}} \right\|^2,$$

where $\mathbf{A}$ and $\widehat{\mathbf{X}}$ are defined in Eq. (5) to impose the desired point-constraints on top of detail preservation.

- **2. Balance constraint** To deal with the balance constraint, we first need to test whether the current barycenter lies over the desired region. This is easily done using the matrix $\mathbf{G}$ defined in Section 3.2 by checking $\mathbf{G}\mathbf{V}^{[m,k-1]}$. If the constraint must be activated, we add the following energy term:

$$\mathcal{E}_2(m,k) = \left\| \mathbf{G}\mathbf{V}^{[m,k]} - \mathbf{g}^{[m,k-1]} \right\|^2$$

where $\mathbf{g}^{[m,k-1]}$ is the closest point from the center of mass of $\mathcal{M}^{[m,k-1]}$ to the supporting area as defined in 3.2.

- **3. Length Constraint** We implement our length constraint by adding the following energy term:

$$\mathcal{E}_3(m,k) = \left\| (\mathbf{v}_i^{[m,k]} - \mathbf{v}_j^{[m,k]}) - \frac{\mathbf{v}_i^{[m,k-1]} - \mathbf{v}_j^{[m,k-1]}}{\|\mathbf{v}_i^{[m,k-1]} - \mathbf{v}_j^{[m,k-1]}\|} L_{ij} \right\|^2,$$

which corresponds to a linearization of the deformation energy defined in Section 3.2 around the *previous* result. This implementation, although similar to the one in [Huang et al. 2006], differs by the fact that the direction of the bones are guided by the deformation result of the previous thread, and *not* the deformation result from the previous iteration of the same thread. The difference is significant: their convergence rate was impaired by trying to maintain the directions of the last iteration. Instead, we rely on the result of the previous thread, which is *not* using the length constraint. As a result, on a simple example like a rotating bone (illustrated in Figure 12), our length-constraint algorithm converges more than one order of magnitude faster. We measured an average improvement factor of 30 on complex models for this particular energy optimization.

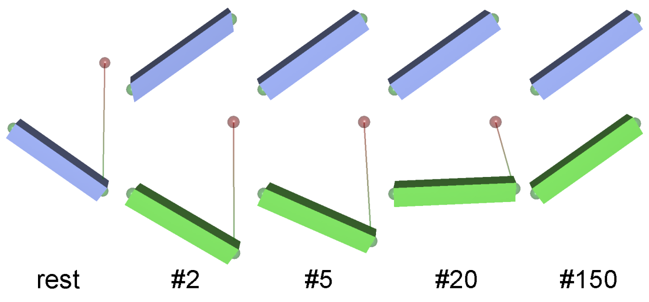

Figure 12: *Convergence of Length Constraint: when the user tries to rotate a rod-like mesh through a position constraint, our length constraint (top) converges in* 5 *iterations, whereas [Huang et al. 2006] takes* 150 *iterations (bottom).*

- **4. Joint Limit Constraint** For each constrained joint in the skeleton, we first check if the relative orientation between the two consecutive bones at the joint in its current state (encoded by

$\mathbf{V}^{[m,k-1]}$) is beyond the imposed angle limits[1]. If so, the joint limit constraint is activated. For the two bones b_1 and b_2 adjacent at the joint, we implement our joint limit constraint by adding the following energy term into the deformation energy:

$$\mathcal{E}_4(m,k) = \sum_{(i,j)\in \text{pairs}(b_1,b_2)} \left\| (\mathbf{v}_i^{[m,k]} - \mathbf{v}_j^{[m,k]}) - \theta_{ij}^{[m,k-1]} \right\|^2$$

where $\theta_{ij}^{[m,k-1]}$ is the target vector (i.e., distance and direction) between tetravertices to enforce joint limit.

The three possible degrees of freedom for a joint can be expressed by the direction that b_2 is along (2 DoF's) and by how much b_2 is twisted, relative to b_1. If the limit of the former is violated, we try to align b_2 to the closest direction that does not violate the limit by aligning $\mathbf{v}_i - \mathbf{v}_j$ to θ_{ij}, where i and j are the two vertices of the bones not shared by each other. If the latter limit is exceeded, we align the displacement between the virtual vertices of b_2 (and/or b_1) to a direction that satisfies the twist angle limit. To determine the closest directions within joint limit, we could rotate both bones to change their relative orientation. In order to get more balanced results, instead of fixing one of the bones or distributing the rotation evenly to both bones, we distribute rotation *proportionally to changes in the direction of the two bones in the previous thread*, as it indicates the susceptibility of each bone to rotation. Figure 13 shows that this simple strategy gives very natural results.

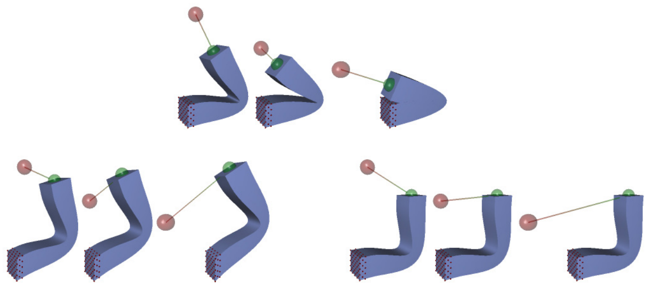

Figure 13: *Enforcing Joint Limit Constraints: a 2-link rod-like mesh is bent with its one end held fixed. Without joint limit constraint, the fixed end never rotates (top). With joint limit constraint, a naive approach to derive a target direction during the optimization can lead to an undesired rotation of the link with fixed end (bottom left). Using our approach to get the target direction, the fixed end does not rotate (bottom right), behaving as expected.*

- **5. Rigidity constraint** For each bone b, we implement our rigidity constraint by adding the following energy term into the deformation energy:

$$\mathcal{E}_5(m,k) = \sum_{(i,j)\in tetra(b)} \left\| \left(\mathbf{v}_i^{[m,k]} - \mathbf{v}_j^{[m,k]}\right) - \mathbf{d}_{ij}^{[m,k-1]} l_{ij} \right\|^2$$

where $tetra(b)$ is the tetrahedron of bone b, $\mathbf{d}_{ij}^{[m,k-1]}$ gives the target direction and l_{ij} gives the target length as defined in Eq. (6).

Many possible target directions can be used. Since we aim to keep the directions of the bones affected as little as possible, we found the following approach to provide the best results. Instead of obtaining the direction through polar-decomposition, which may undesirably change the directions of bones, we simply move the two virtual vertices to positions that make the tetrabone regular while maintaining the bisector of the dihedral angle of the bone itself (as described in Figure 14).

[1]In general, joint limits are specified as a subset of $SO(3)$, or of the unit quaternion ball, or as ranges of Euler angles.

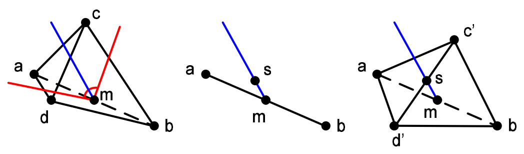

Figure 14: *Placing virtual vertices in rigidity constraint: given the position of the four tetra-vertices a, b, c and d resulting from the previous thread, we first determine the bisector (blue line) of the dihedral angle (red lines and arc) formed by the bone $\overline{ab}$ and its two neighboring faces that goes through $\overline{ab}$'s midpoint m (left). We determine a point s on the bisector such that $|\overline{ms}| = \frac{\sqrt{2}}{2}|\overline{ab}|$ (middle). We then find c' and d' such that $\overline{sc'}$ and $\overline{sd'}$ are perpendicular to both $\overline{ab}$ and $\overline{ms}$ with their lengths being $\frac{1}{2}|\overline{ab}|$ (right). The vertices c' and d' are used as the new positions of the virtual vertices.*

4.3 W-Phase: Direct Optimization

The W-phase does not require any specific treatment as the constraints are much milder. Therefore, we use only one thread and the quadratic objective function to minimize is simply set to:

$$\mathcal{E} = \left\| \mathbf{L}\mathbf{X}^{[k]} - \hat{\delta}\mathbf{X}^{[k-1]} \right\|^2 + \left\| \mathbf{A}\mathbf{X}^{[k]} - \widehat{\mathbf{X}} \right\|^2 + \sum_{w_{bi}\in W} \left(w_{bi} - \frac{1}{|\mathcal{N}(i)|} \sum_{j\in\mathcal{N}(i)} w_{bj} \right)^2 + \sum_{i\in[1..V]} \left(\sum_{b\in B} w_{bi} - 1 \right)^2.$$

Notice that this energy includes only the constraints involving $\mathbf{W}$ (*i.e.*, the Laplacian, position, and smoothness constraints). Figure 15 shows how optimizing vertex weights reinstates the shape and details of the mesh.

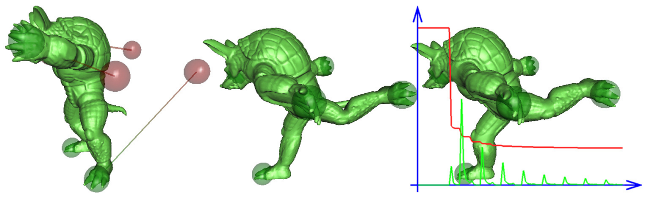

Figure 15: *Optimizing Vertex Weights: the Armadillo model (left) is deformed to look like a sprint athlete on the finishing line. The arm and legs exhibit distortion after the V-phase (middle). After optimizing vertex weights, the arm and legs recover their shape (right). The red curve shows the deformation energy, while the green curve shows the L^2 norm of the distance between two consecutive meshes during the optimization.*

4.4 Polishing Phase: Full Space Postprocessing

Our optimization procedure heavily benefits from the fact that we use a skinned mesh. However, a major shortcoming of all SSD methods is that excessive deformation can result in the collapse of joint regions (see Figure 7). To avoid this situation when high-quality meshes are desired, we process the result of our optimization by further optimizing the vertex positions directly in *full space* (as opposed to the subspace they were constrained to via vertex blending while satisfying the IK constraints). For each bone, the vertices with a single weight of 1 (i.e., most of the vertices) are maintained fixed. The rest of the mesh (*i.e.*, the vertices around joints) is automatically deformed using the two-step method as described in [Lipman et al. 2005]. Specifically, the harmonic field [Zayer et al. 2005] of the transformations is first used to modify the Laplacian coordinates of the joint vertices, then the final, deformed mesh is solved so that these joint vertices satisfy these Laplacian coordinates. Since this postprocessing is only performed on the vertices near joints, the added overhead is insignificant. See Figure 7 for an example of postprocessing.

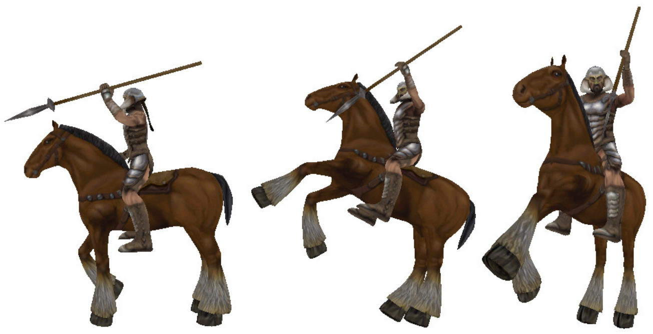

Figure 16: *Deformation of Multiple Objects: The horseman model (left) consists of three parts: a horse, a warrior, and a spear. The warrior is glued onto the saddle of the horse and the spear is glued into the warrior's left hand. A balance constraint is applied to the warrior, with his hips set as supporting area. The joint limit constraint is applied to the wrist to prevent it from bending too much. Editing is performed by manipulating the horse's forelegs, the warrior's head and the direction of the spear (middle and right). Notice how the warrior automatically leans forward to keep his balance on the saddle when the horse is moved (see accompanying video).*

5 Applications and Results

Our implementation of the two-phase cascading solver presented in this paper is fast enough to allow direct manipulation of moderately large meshes (50K+ vertices) in real-time on a dual-core Pentium 4 PC (see Table 1 for detailed timing and model statistics for some of the examples shown in this paper, and Figure 2 for the skeletons used for our different examples). Notice that having all five threads concurrently only slows down each step of the optimization by 60% compared to solely enforcing Laplacian and position constraints (*i.e.*, a single thread), demonstrating that our cascading approach makes efficient use of each thread's progress to help the following threads. The temporal coherency of the mouse position throughout the design of new mesh poses results in quick convergence of the various threads, as the difference between two consecutive poses is often small. For larger meshes we also reduce the number of threads to three during realtime interaction, running the full five threads only as a final, polishing phase when the user stops dragging the mesh around. Larger meshes can even be first simplified via mesh decimation for their interactive manipulation, then finalized through optimization initialized with the optimal pose of the simplified mesh. Note that in our experiment and all the examples of this paper, it took from 5 to 10 iterations for the V-phase to converge, while it usually takes 2 to 5 iterations for the W-phase. We thus recommend such an average scheduling for these two alternating phases. Finally, we wish to point out that weighting of various deformation energies can be used as a further tool to "tailor" the way models react to constraint, *i.e.*, to define an implicit prioritization of the constraints—although we never had to recourse to this option in our tests.

To validate the efficiency of our cascading approach, we implemented a brute-force solver that minimizes the total energy directly via repeated Gauss-Newton iterations as shown in [Huang et al. 2006]. That is, each iteration of the brute-force minimization is exactly the same computational complexity as our last thread. However, our cascading approach pays off nonetheless: the convergence time for a deformation of the Armadillo (30K vertices) is 0.45s in our case, compared to 17s for the brute-force way. The difference in timings is explained by the huge discrepancy in iterations needed for convergence: we converge in 10 iterations of our cascading solver, while brute-force solving requires over 1000 iterations. So despite the larger cost of our iterations, the total timings are dramatically improved.

This level of interactivity, along with the numerous design constraints we introduced (automatic balancing and most-rigid constraints in particular), makes our interactive mesh deformation framework a true *mesh puppetry* workshop: the user can easily manipulate an inanimate mesh to create natural, life-like poses in no time, and our position constraint, when used with length/balance constraints, is akin to a puppeteer pulling the strings of a puppet.

5.1 Interactive Multiple-Object Deformation

Rather than handling individual models, our system can naturally support simultaneous deformation of multiple objects. With the features defined in this paper, complex deformation tasks on multiple objects can be defined efficiently and intuitively. In particular, relative positions of objects can be easily maintained or adjusted through common point constraints. For example, in Figure 16 the warrior and his horse are two separate meshes, maintained into one single entity by constraining the warrior's hip onto the horse's saddle and enforcing the balance of the warrior on the saddle. The user can then raise the body of the horse: the warrior will automatically lean forward in order to stay on the horse and maintain a natural balance (Figure 16).

Self-Collision Detection and Handling An important improvement of our technique in the context of multiple-object editing is the treatment of collisions. Our cascading optimization solver enables (self-)collision detection and handling rather well, contrary to existing mesh deformation techniques. We achieve this collision handling by adding one more thread (of very low cost compared to the others in order to maintain interactive rates). Each V-phase starts by performing a collision detection routine using the COLDET package [ColDet 2002]. In practice we use a fast, approximate collision detection using either bounding boxes, or the rigid mesh partition of each bone. If a collision is detected, we add a deformation energy (inserted in between the Laplacian and the balance constraints) very similar to the length constraint case to force the middle of each bone to be sufficiently away from the other bone's center: the two bones will thus move away from each other, preventing self-collision. Figure 17 shows a result of this collision handling where the armadillo performs a ballet routine: since the motion capture data comes from a real dancer with a much less prominent torso, the armadillo shows a self-collision if no special treatment is provided. Once activated, the collision-handling thread removes the issue and the process is only slowed down by 5%. Note that while our simple procedure avoids most intersections and greatly improves the quality of the deformation results, we cannot formally guarantee self-intersection-free deformation results contrary to [von Funck et al. 2006].

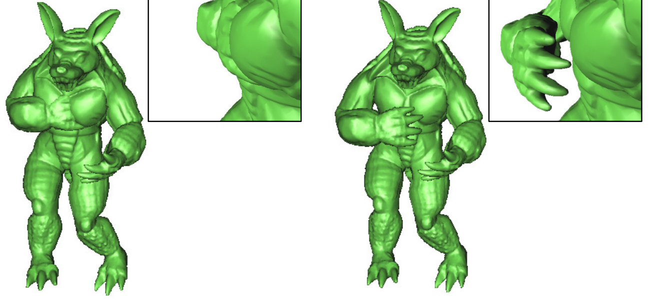

Figure 17: *The Armadillo Dance: Motion retargeting of long sequences inevitably induces self-collision. With our self-collision handling, the hand of the Armadillo does not penetrate his thorax as it dances.*

model	# vertices	# bones	# tetravertices	1 thread	2 threads	3 threads	4 threads	5 threads
Horse	8,431	23	70	169	116	95	77	65
Cavalry	17,808	24	75	98	67	54	43	37
Camel	21,887	19	58	77	53	43	35	30
Dinosaur	25,002	17	52	63	45	37	30	26
Armadillo	50,002	18	55	39	26	21	17	15

Table 1: *Performance statistics, including sizes of the meshes and the frames-per-second (fps) rates that we achieved when using only a few or all five threads in our cascading optimization solver. All timings reported here were measured on a 3.2GHz Intel Xeon workstation with 4GB RAM.*

5.2 Deformation Transfer and Motion Retargeting

Our framework can be used to *transfer the pose of a model to another* very easily: this pose transfer is achieved by establishing a few correspondences between the original model and a target mesh and applying our deformation algorithm. As demonstrated in Figure 18, we can put point constraints on the legs, head, neck, and tail of some existing cat's poses and transfer them onto a horse while asking for the horse's limbs to keep their length and rigidity. Although the horse's neck and legs are longer than the cat's, our mesh deformation procedure successfully produces plausible results interactively.

We can also transfer a whole existing animation to a skinned mesh by transferring each individual pose across time. A few correspondences are defined on the very first frame and derived automatically for the rest of the animation (we suppose here that the model discretization remains the same throughout the animation). Putting these correspondences as position constraints for another skinned mesh, and minimizing the resulting deformation energy for each frame will automatically re-target the motion. Since the previous pass is used as the initial condition of the next pose's optimization, we obtain excellent time coherency (see our accompanying video). One can define additional IK constraints to enforce specific characteristics of the target mesh so as to best capture the initial animation while sticking to the constraints of the target mesh. For instance, adding a limp to a character only requires enforcing a stiff knee by adding a joint limit constraint: the original animation will be followed as closely as possible while preventing any bending of the knee (see accompanying video for such an example on the Armadillo). Note that we do not have restrictions on the format of the source animation: it could be a mesh sequence, or even a point-set sequence (tracker positions). We do not have restrictions on the compatibility between skeletons of the source animation and skinned mesh either. In fact, only a few correspondences and the skeleton of the resulting skinned mesh animation needs to be specified. Note that the resulting animation is compactly represented by a series of skeleton poses together with a set of weights for vertices near joints.

In the accompanying video, we show another example of motion retargeting. Using as reference the cloth-like camel animation falling down on the ground, we created a simple skinned camel, and then transferred the animation onto this skinned mesh. We can further edit this animation by adding IK constraints (here, length and joint limit constraints) to significantly alter the motion with very little user interaction: the camel now appears to slip on an icy surface (Figure 19 shows snapshots).

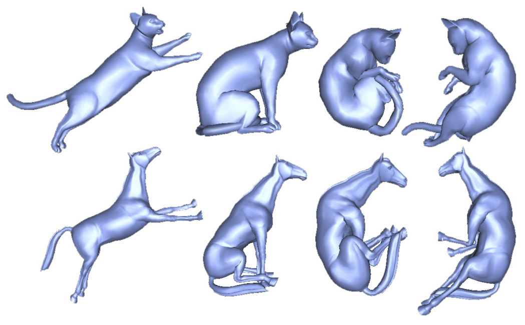
Figure 18: *Pose transfer: various poses of a cat can be transferred onto a horse model through a few point constraints, despite the disparity of the neck and leg size between the two animals.*

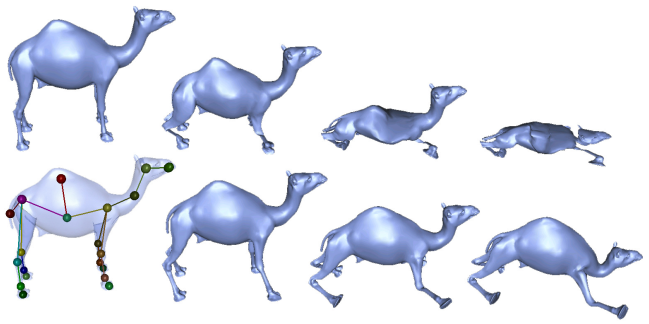
Figure 19: *Camel Falling: the falling motion of the cloth-like camel (top row, courtesy of [Sumner and Popović 2004]) is transferred onto a skinned mesh of the same camel; the use of length, rigidity, and joint limit constraints dramatically changes the animation (with little user supervision), as the camel now seems to slip on ice (bottom row).*

Limitations Our approach can only handle articulated models, and does not apply to the modeling of plastic deformations. Another limitation comes from our soft constraint enforcement. If a handle is dragged *too far* from the character (i.e., far from the character's reach), the position constraint and the length and rigidity constraints can start conflicting a lot. Consequently, the length/rigidity constraints may no longer be fulfilled, possibly leading to unnatural deformation—although altering the weight of these constraints can partially resolve this issue.

6 Conclusion

We have presented a variational approach for mesh puppetry. Our framework supports direct manipulation of vertices in the mesh, various IK-based deformation constraints, and self-intersection avoidance. Our solver, using a novel two-phase cascading optimization procedure, provides an efficient numerical tool to solve the optimization of all the constraints. Our experiments demonstrate that our custom solver can handle meshes with 50K vertices in realtime (15 frames/s) on a dual-core Pentium 4 PC. We have also shown that several applications can benefit from this fast solver, including deformation transfer and collision detection/handling.

The applications of such a framework are manifold. First, our solver's strategy could be used in games to improve the efficiency of realtime animation. An intriguing application would also be to offer an interactive platform for *template matching*: medical datasets of body parts (with known structures and specificities) could be mapped to a particular patient's part through a fast and simple user-guided procedure based on our approach—potentially providing an extremely robust shape matching. As for future work, we wish to develop additional high-level constraints to further improve design efficiency.

Acknowledgement

We would like to thank the reviewers for their valuable comments, and Shichao Hu and Bennett Wilburn for their help during demo and video production. Xiaohan Shi and Hujun Bao were partially supported by the 973 Program of China (No. 2202CB312100) and the Cultivation Fund of the Key Scientific and Technical Innovation Project, Ministry of Education of China (No. 705027). Additional funding include NSF (CAREER CCR-0133983 and ITR DMS-0453145), DOE (DE-FG02-04ER25657), Pixar Animation Studios, and the Okawa Foundation.

References

BADLER, N. I., MANOOCHERHRI, K. H., AND WALTERS, G. 1987. Articulated figure positioning by multiple constraints. *IEEE Comput. Graph. Appl. 7*, 6, 28–38.

BAERLOCHER, P., AND BOULIC, R. 2004. An inverse kinematic architecture enforcing an arbitrary number of strict priority levels. *Visual Computer 20*, 6, 402–417.

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: Coupled prisms for intuitive surface modeling. In *Eurographics Symposium on Geometry Processing*, 11–20.

BOULIC, R., MAS-SANSO, R., AND THALMANN, D. 1997. Complex character positioning based on a compatible flow model of multiple supports. *IEEE Transactions on Visualization and Computer Graphics 3*, 3, 245–261.

CHEN, Q., AND GUAN, S.-U. 2004. Incremental multiple objective genetic algorithms. *IEEE Trans. on Systems, Man, and Cybernetics 34*, 3 (June), 1325–1334.

COLDET. 2002. Free 3d collision detection library (gnu lgpl). *http://sourceforge.net/projects/coldet*.

DER, K. G., SUMNER, R. W., AND POPOVIĆ, J. 2006. Inverse kinematics for reduced deformable models. *ACM Trans. Graph. 25*, 3, 1174–1179.

GUSKOV, I., SWELDENS, W., AND SCHRODER, P. 1999. Multiresolution signal processing for meshes. In *SIGGRAPH 99 Conference Proceedings*, 325–334.

HUANG, J., SHI, X., LIU, X., ZHOU, K., WEI, L.-Y., TENG, S.-H., BAO, H., GUO, B., AND SHUM, H.-Y. 2006. Subspace gradient domain mesh deformation. *ACM Trans. Graph. 25*, 3, 1126–1134.

JAMES, D. L., AND TWIGG, C. D. 2005. Skinning mesh animations. *ACM Trans. Graph. 24*, 3, 399–407.

JU, T., SCHAEFER, S., AND WARREN, J. 2005. Mean value coordinates for closed triangular meshes. *ACM Trans. Graph. 24*, 3, 561–566.

KOBBELT, L., CAMPAGNA, S., VORSATZ, J., AND SEIDEL, H.-P. 1998. Interactive multi-resolution modeling on arbitrary meshes. In *SIGGRAPH 98 Conference Proceedings*, 105–114.

LE CALLENNEC, B., AND BOULIC, R. 2006. Interactive motion deformation with prioritized constraints. *Graphical Models 68*, 2 (March), 175–193.

LEWIS, J. P., CORDNER, M., AND FONG, N. 2000. Pose space deformation: a unified approach to shape interpolation and skeleton-driven deformation. In *SIGGRAPH 2000 Conference Proceedings*, 165–172.

LIPMAN, Y., SORKINE, O., LEVIN, D., AND COHEN-OR, D. 2005. Linear rotation-invariant coordinates for meshes. *ACM Trans. Graph. 24*, 3, 479–487.

LIPMAN, Y., COHEN-OR, D., GAL, R., AND LEVIN, D. 2006. Volume and shape preservation via moving frame manipulation. *ACM Trans. Graph.*, to appear.

LLOYD, S. P. 1982. Least squares quantization in pcm. *IEEE Transactions on Information Theory 28*, 2, 129 – 137.

MAGNENAT-THALMANN, N., LAPERRIÈRE, R., AND THALMANN, D. 1988. Joint-dependent local deformations for hand animation and object grasping. In *Proceedings on Graphics interface '88*, 26–33.

MOHR, A., TOKHEIM, L., AND GLEICHER, M. 2003. Direct manipulation of interactive character skins. In *Proceedings of the 2003 symposium on Interactive 3D graphics*, 27–30.

NEALEN, A., SORKINE, O., ALEXA, M., AND COHEN-OR, D. 2005. A sketch-based interface for detail-preserving mesh editing. *ACM Trans. Graph. 24*, 3, 1142–1147.

SHI, L., YU, Y., BELL, N., AND FENG, W.-W. 2006. A fast multigrid algorithm for mesh deformation. *ACM Trans. Graph. 25*, 3, 1108–1117.

SORKINE, O., COHEN-OR, D., LIPMAN, Y., ALEXA, M., RÖSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Eurographics Symposium on Geometry Processing*, 175–184.

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

SUMNER, R. W., ZWICKER, M., GOTSMAN, C., AND POPOVIĆ, J. 2005. Mesh-based inverse kinematics. *ACM Trans. Graph. 24*, 3, 488–495.

TEICHMANN, M., AND TELLER, S. 1998. Assisted articulation of closed polygonal models. In *SIGGRAPH '98: ACM SIGGRAPH 98 Conference abstracts and applications*, 254.

VON FUNCK, W., THEISEL, H., AND SEIDEL, H.-P. 2006. Vector field based shape deformations. *ACM Trans. Graph. 25*, 3, 1118–1125.

YAMANE, K., AND NAKAMURA, Y. 2003. Natural motion animation through constraining and deconstraining at will. In *IEEE Transactions on Visualization and Computer Graphics*, no. 3.

YU, Y., ZHOU, K., XU, D., SHI, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2004. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph. 23*, 3, 644–651.

ZAYER, R., RÖSSL, C., KARNI, Z., AND SEIDEL, H.-P. 2005. Harmonic guidance for surface deformation. In *Eurographics 2005*, 601–609.

ZHAO, J., AND BADLER, N. I. 1994. Inverse kinematics positioning using nonlinear programming for highly articulated figures. *ACM Trans. Graph. 13*, 4, 313–336.

ZHOU, K., HUANG, J., SNYDER, J., LIU, X., BAO, H., GUO, B., AND SHUM, H.-Y. 2005. Large mesh deformation using the volumetric graph laplacian. *ACM Trans. Graph. 24*, 3, 496–503.

ZORIN, D., SCHRÖDERR, P., AND SWELDENS, W. 1997. Interactive multiresolution mesh editing. In *SIGGRAPH 97 Conference Proceedings*, 259–268.