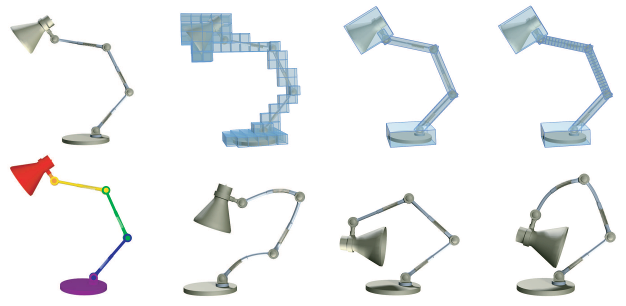
Joint-aware Manipulation of Deformable Models

Weiwei Xu* Jun Wang† KangKang Yin* Kun Zhou‡ Michiel van de Panne§ Falai Chen† Baining Guo*

*Microsoft Research Asia †Univ. of Science and Technology of China ‡Zhejiang Univ. §Univ. of British Columbia


(a)

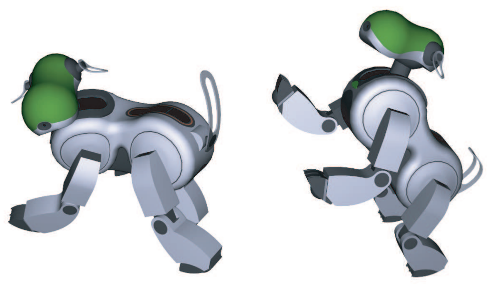

(b)

Figure 1: *Two representative models that users can interactively manipulate within our deformation system. (a) (column 1:) A desk lamp connected by revolute joints, and its color-coded components. The lampshade is manipulated with the same handle trajectory for three cases: (column 2:) joint-unaware deformation has difficulty facing the lampshade backward because of immovable joints, and links are bent unnaturally(131 cells). (column 3:) joint-aware deformation with fully rigid links(6 cells). (column 4:) joint-aware deformation with two deformable links in the middle(76 cells). (b) An Aibo-like robot dog with a soft tail, a soft body, and two soft ears interactively posed to walk and stand up.*

Abstract

Complex mesh models of man-made objects often consist of multiple components connected by various types of joints. We propose a joint-aware deformation framework that supports the direct manipulation of an arbitrary mix of rigid and deformable components. First we apply slippable motion analysis to automatically detect multiple types of joint constraints that are implicit in model geometry. For single-component geometry or models with disconnected components, we support user-defined virtual joints. Then we integrate manipulation handle constraints, multiple components, joint constraints, joint limits, and deformation energies into a single volumetric-cell-based space deformation problem. An iterative, parallelized Gauss-Newton solver is used to solve the resulting nonlinear optimization. Interactive deformable manipulation is demonstrated on a variety of geometric models while automatically respecting their multi-component nature and the natural behavior of their joints.

CR Categories: I.3.5 [Computer Graphics]: Computational Geometry and Object Modelling—Geometric Algorithms; I.3.5 [Computer Graphics]: Three Dimensional Graphics and Realism—Animation

Keywords: space deformation, joint constraint, inverse kinematics, slippable motions

1 Introduction

Traditional space deformation algorithms largely assume that an embedded object should be treated as a single component [Gain and Bechmann 2008]. However, many 3D models, particularly those of man-made CAD objects, consist of multiple components. Recent years have seen considerable progress in making geometric deformations more content aware, such as material-aware mesh deformation [Popa et al. 2006] and non-homogeneous resizing of complex models [Kraevoy et al. 2008]. These methods acknowledge that complex models usually have multiple parts with different properties and features, which should be treated differently during manipulation. Where possible, deformation methods should attempt to respect any constraints, semantic or otherwise, that might be implicit in the geometry.

Inspired by the aforementioned work, and by models of the type shown in Figure 1, we observe that constituent components of a model are commonly connected by joints of either mechanical or biological origin. These joints serve not only to segment the complex model into components, but also to constrain the relative spatial configurations of neighboring components. We propose a deformation system that models and respects these joint constraints. The benefits of such an approach are two-fold. First, the deformations of different components can be represented independently of each other, which serves to eliminate the unnatural coupling that otherwise exists when multiple objects inhabit a shared space-based deformation. Second, joints define natural degrees of freedom afforded by the geometry, which allows for natural and physically-plausible deformations and poses. As illustrated in the rightmost column of Figure 1(a), inter-component articulations as well as intra-component deformations can be achieved simultaneously.

Joint constraints have long been used in skeletal animation to help in posing and skinning virtual characters [Magnenat-Thalmann et al. 1988; Lewis et al. 2000]. However, this comes with two caveats. First, a matching skeleton (i.e., joint hierarchy) has to be defined and rigged. This is often a non-trivial task, although significant progress has recently been made towards automating this process [Baran and Popović 2007; Au et al. 2008]. The joint detection we employ can be seen as extending automatic skeleton creation techniques to a significant new class of geometry. Second, skeletal animation systems require the skinning weights for each vertex of the mesh to be carefully assigned. We shall rely on the surface reconstruction of our volume-based space deformation to return a

*e-mail:wwxu,kkyin,bainguo@microsoft.com
†e-mail:v-junwa@microsoft.com,chenfl@ustc.edu.cn
‡e-mail:kunzhou@acm.org
§e-mail:van@cs.ubc.ca

smooth representation of the deformed model. A further distinction of the deformation approach is the direct manipulation of mesh vertices as handles, as compared to skeleton links or end effectors.

Joint constraints have also been considered in the context of deformation models. Some methods require support from an underlying skeleton [Huang et al. 2006; Shi et al. 2007]. Others detect near-rigid components from example poses [James and Twigg 2005]. In this work, we focus on articulated mechanisms whose components are connected by mechanical joints. We apply a shape analysis algorithm, originally designed to segment kinematic surfaces of 3D scanned shapes [Gelfand and Guibas 2004], to extract joint constraints. We further augment these joints with automatically detected parameters that prescribe the available range of motion for each joint. Virtual joints can also be inserted into disconnected models or single-component models. The joint constraints are then directly incorporated into the deformation objective to maintain physically plausible spatial relationships between components.

Our space deformation algorithm follows the philosophy of reduced deformable models and subspace techniques to decouple the deformation complexity from the geometric complexity [Der et al. 2006; Huang et al. 2006]. More specifically, we use an aggregation of elastically-coupled as-rigid-as-possible cuboid cells to enclose the model of interest. Each cell has its own associated affine transformation to be optimized, which will then be interpolated using Moving-Least-Squares (MLS) in order to reconstruct the deformed model [Kaufmann et al. 2008]. Rigid or near-rigid volumetric cells have been proven to be robust and to yield physically-inspired behavior [Botsch et al. 2007]. The component-based space deformation we propose allows different components to have independent spatial discretizations. Rigid components are represented by only one cell, and deformable components employ multiple cells. Objects can thus consist of an arbitrary mix of rigid and deformable components.

We integrate manipulation handle constraints, multiple components, joint constraints, joint limits, and deformation energies into a single volumetric-cell-based space deformation framework. The transformations associated with each cell form the optimization parameters. An iterative, parallelized Gauss-Newton solver is used to solve the resulting non-linear optimization problem.

Contributions: We present a novel deformation framework that naturally supports arbitrary mixes of rigid and deformable components, connected by a variety of joint types. To the best of our knowledge, we are the first to apply slippage analysis for the automatic detection of joint constraints – we develop the assorted steps that are necessary beyond the slippage analysis to make the joint analysis work. Our implementation and results demonstrate the combined promise of these ideas.

2 Related Work

Mesh Deformation: Surface-based mesh deformation methods have been widely used in mesh editing and animation. Representative works include multi-resolution editing [Zorin et al. 1997; Kobbelt et al. 1998], Laplacian surface editing [Sorkine et al. 2004; Yu et al. 2004; Lipman et al. 2005; Botsch and Sorkine 2008], and coupled prisms [Botsch et al. 2006]. These methods target the preservation of surface details on single-component models, i.e., a single connected mesh. Although potentially applicable to multiple components, these methods on their own do not provide the mechanisms to handle boundary conditions and spatial relationships between components imposed by joint constraints.

Volume-based space deformation methods deform a 3D model by warping its ambient space [Sederberg and Parry 1986; Huang et al. 2006; Sumner et al. 2007; Botsch et al. 2007; Shi et al. 2007]. Reduced models and subspace techniques are commonly exploited to decouple the deformation complexity from the underlying geometric complexity, thus enabling interactive manipulation of high resolution meshes. Our work follows the same principle to achieve interactive performance. More specifically, we formulate space deformation as a nonlinear optimization problem, similar to the recent work on embedded deformation and rigid cell deformations [Sumner et al. 2007; Botsch et al. 2007]. The transformations associated with each deformation unit, be it deformation graph nodes or volumetric cells, form the optimization parameters. Unlike previous work, we focus on complex models with multiple components. Volumetric cells for different components are decoupled so that the deformation system can exploit the inter-component degrees of freedom provided by joints.

Material-aware mesh deformation incorporates non-uniform materials into the geometric deformation framework [Popa et al. 2006]. Material stiffness can be specified with a paint-like interface or can be learned from a sequence of example deformations. A knee joint can be emulated by specifying an anisotropic material that is flexible in one direction and rigid in the other two. Mechanical joints cannot be realized this way, however. Moreover, our work handles multiple types of joints with joint limit constraints.

Non-homogeneous resizing is needed to preserve important structure and features, such as circular shapes, of complex models consisting of multiple components [Kraevoy et al. 2008]. A protective grid, or vulnerability map, is first constructed. A space deformation technique then scales different regions non-homogeneously to respect this map. Our method addresses issues that are complementary to such resizing problems. Cylindrical and spherical joints between components, such as hinges and ball-and-socket joints, are automatically preserved by our joint-aware deformation, although for different underlying reasons than those proposed by Kraevoy et al.[2008].

Inverse Kinematics: Inverse Kinematics(IK) is a standard problem in robotics and posing characters in computer animation [Murray et al. 1994; Parent 2008]. Given an articulated kinematic chain of rigid bodies, IK solves for joint angles that achieve a desired configuration of the end effector. IK usually deals with under-constrained problems when there are more joint degrees of freedom (DOFs) available than the DOFs of the end effector. IK methods commonly use the inverse Jacobian or cast the problem as an optimization. Mesh-based Inverse Kinematics (MESHIK) considers the problem of finding meaningful mesh deformations that meet specified vertex constrains [Sumner et al. 2005; Der et al. 2006]. This requires a collection of sample poses, which is often not readily available for complex models. Constraint-based mesh deformation techniques can usually incorporate joint constraints to some extent, if a reasonable skeleton is provided [Huang et al. 2006; Shi et al. 2007]. Our work integrates articulation and deformation in a skeleton-free way, handles more types of joints, and automatically detects joints that are implicit in the geometry.

Shape Analysis: Shape analysis algorithms study a variety of geometric, structural, or semantic features and metrics, including mesh saliency, symmetry, up-right orientation, and feature vulnerability, to name a few. A wide spectrum of applications, such as mesh segmentation, viewpoint selection, reverse engineering, shape retrieval, and shape recognition, can benefit from such analysis [Katz and Tal 2003; Attene et al. 2006; Mitra et al. 2007; Kraevoy et al. 2008; Fu et al. 2008]. Similar to our consideration of complex models, 3D exploded view diagrams often take complicated mechanical assemblies with multiple parts as input of interest. To visualize the spatial relationships between parts, blocking constraints along explosion directions have to be investigated when generating such di-

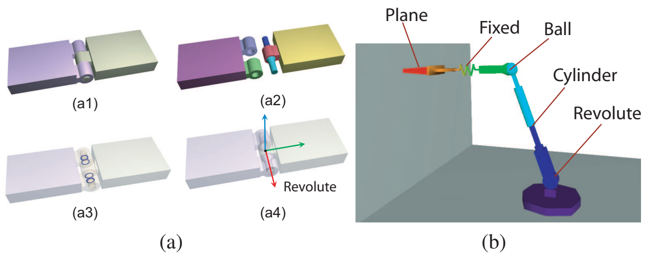


Figure 2: *Joint analysis: (a1) An input model with two components. (a2) Parts segmented. (a3) Valid points on the intersection surfaces passed to shape analysis. (a4) The joint frame associated with the identified revolute joint. (b) A robot brush with components shown in different colors and joints labelled.*

agrams [Li et al. 2008]. Our application requires analysis of diverse joints to constrain the relative configurations and motions between components. Slippable motion analysis [Gelfand and Guibas 2004] lies at the core of our joint analysis algorithm and will be discussed in detail shortly (§3.1).

3 Joint Constraint Analysis

In mechanics and robotics, the words *joint* and *constraint* are often used interchangeably to represent a relationship that is enforced between two bodies so that they can only have certain positions and orientations relative to each other. Our deformation system models motion constraints for articulation with typical types of joints used in mechanics.

3.1 Slippable Motions

Common mechanical joints have characteristic shapes, such as the revolute joint shown in Figure 2(a). The core of our joint analysis is based on the notion of slippable shapes and slippable motions [Gelfand and Guibas 2004]. Slippable motions are defined as rigid motions which, when applied to a shape, slide the transformed version against the original copy without forming any gaps. That is, the shape is invariant under its slippable motions. Slippable shapes include rotationally and translationally symmetrical shapes such as planes, spheres, and cylinders. Touching slippable shapes can undergo their corresponding slippable motions without penetrating each other, and therefore are often found in joints for mechanical models. For instance, the slippable motions for a cylinder include rotations around the cylinder's axis and translations along the axis.

Slippage analysis was originally designed to reverse engineer CAD objects, and segment complex shapes into simple geometric parts. Slippable motions can be computed as a least-squares problem whose minimum is the solution of a linear system $\mathbf{C}\mathbf{x} = 0$. The slippable motions of a Pointset **P** are those that belong to the null space of the covariance matrix **C**. Eigenvectors of **C** whose corresponding eigenvalues are zero correspond to the slippable motions of **P**. In practice, due to noise **C** is likely to be full rank, and we choose those eigenvectors whose eigenvalues are sufficiently small as slippable motions. We refer the reader to [Gelfand and Guibas 2004] for more details. Here we simply state that we can effectively determine the valid relative motions between two components by detecting their intersection surfaces and calculating their slippable motions. Table 1 shows the slippable motions of different surfaces.

3.2 Joint Detection

Given a complex mesh model as input, we first analyze the connectivity of triangles and separate them into connected components. Smaller components or semantically coupled components can be merged into larger components by users as they see fit. Intersecting surfaces of adjacent components are then passed to the slippable motion analyzer to identify potential degrees of freedom of the relative motions between the surfaces, such as translations and rotations. We further model the detected DOFs of slippable motions as different types of mechanical joints, such as revolute and prismatic joints. From an input model to final joint constraints, there are four major steps involved: intersection surface detection, slippable motion analysis, range of motion detection, and mapping of allowable DOFs to joints. We now describe each step in detail.

Intersection surface detection: We begin by searching for the shortest distance between each pair of components in the complex model. If this minimal distance is less than a user specified threshold, the two components are selected as candidates for further intersection surface detection. We first segment components into near-convex semantic parts [Katz and Tal 2003]. Convex hulls are then computed for each part and intersections between each pair of convex hulls are located. The intersection surfaces are simply the surfaces in the intersecting regions. If two components are in contact and do not intersect with each other, there will be no intersection from their convex hulls. In this case, we look for vertices which are within the distance threshold to each other from the two components under inspection. The vertices of the intersection and contact surfaces are then passed to slippable motion analysis as input.

Erroneous vertices may be detected in the above step. To filter these, we project a vertex of one component along its normal until it intersects with a triangle on the other component, and then compute the normal of the intersected triangle. Only when the angle between these two normals exceed a certain threshold (145 degrees for all the examples shown in this paper), can the vertex be kept for subsequent analysis.

Slippable motion analysis: By default, joints between two components are fixed. When there are more than five valid vertices detected between two components as described in the previous step, we analyze the allowable slippable motions between these components. The output of the slippable motion analysis are the number of translational and rotational degrees of freedom, and their corresponding axes.

Note that slippable motion analysis may not be completely accurate for digital mesh models. There are two factors that most affect the stability and accuracy of the analysis. Let $\lambda_1 \le \lambda_2 \le \lambda_3 \le \lambda_4 \le \lambda_5 \le \lambda_6$ be the eigenvalues of **C**. We call the eigenvalue λ_j small if the ratio $\frac{\lambda_6}{\lambda_j}$ is greater than a chosen threshold g. The condition number g determines how many slippable motions are returned, and is adjusted so that the maximum number of slippable motions is three. By default, we choose g conservatively in the range of 100 to 200 in our implementation. Users can also adjust g interactively from the graphical user interface (GUI) we provided.

Slippable motion analysis is more sensitive to the model resolution than the value of the condition number g. For example, if a sphere is discretized too coarsely, there may not be any numerically detectable slippable motions. Segmentation errors in the intersection surface detection step can also degrade the quality of slippage analysis. Nonetheless, given a complex model, slippable motion analysis can save a considerable amount of effort for users in arriving at a reasonable initial joint classification and joint frame alignment. Table 2 provides a quantitative summary.

Type of Surface	Num. small Eigenvalues	Slippable Motions	Type of Joint
sphere	3	3 rot.	ball
plane	3	2 tran., 1 rot.	plane
cylinder	2	1 tran., 1 rot.	cylinder
linear extrusion	1	1 tran.	prismatic
surface of revolution	1	1 rot.	revolute
non-slippable	0	0 tran., 0 rot.	fixed

Table 1: *Slippable motions of various surfaces, and the corresponding mapping to joints.*

Range of motion detection: Slippable motion analysis only outputs the DOFs of valid motions between two components. Range of motion information, such as angle limits of rotational joints, or translational limits of prismatic joints, is not provided. We thus need to additionally discover these joint parameters. Feasible range of joint limits should keep two components in close proximity without any visible penetration. To this end, we design a trial-and-error bisection process. For instance, translational limits are probed by sliding a component along its translational axis until penetration occurs. Angle limits of rotational joints are detected in a similar fashion. If the feasible range of a motion direction is less than a chosen threshold, this DOF will be removed from the system.

Mapping to joints: We model detected slippable motion constraints as typical joint types used in mechanics, as shown in Table 1. While prismatic, revolute and ball joints are standard joints in mechanics, the plane, cylinder and fixed joints require additional elaboration. Plane joints describe the case of two components have planar contacts and can slide and rotate on the plane relative to each other. Cylinder joints have an additional translational DOF compared to revolute joints. Fixed joints maintain a fixed relative position and orientation between two bodies. That means there are no allowable motions between these two components. Fixed joints exist because all eigenvalues computed by slippable motion analysis can be larger than a certain threshold, and semantically these components should not move relatively to each other. The robot brush as shown in Figure 2(b) illustrates most of the joint types we implement. We also allow users to overwrite the type of joints and adjust their range of motion parameters, if they are not satisfied with the results suggested by the system.

We represent joints with local coordinate frames for the convenience of the constraint formulation to be introduced shortly. Local coordinate frames are computed through PCA analysis on valid vertices of the intersection surfaces, and then aligned with the detected translation or rotation axes.

4 Space Deformation

Given a set of identified components and their joint constraints, a deformation framework needs to be developed that supports this rich class of constraints. Prior space deformation methods which employ a single spatial grid for the whole model cannot easily incorporate and maintain spatial relationships among components. Our deformation algorithm advocates building a local spatial grid for each component, and assigns transformation parameters for each local grid. Joints detected by slippage analysis as described in the previous step are used to constrain the transformations of local grids.

We formulate space deformation as a nonlinear optimization problem, which supports deformation edits and joint constraints in a unified manner. The objective function is comprised of energy terms corresponding to shape deformation and error terms related to joint constraints and manipulation goals. The transformations associated with every component-based local grid represent the optimization parameters. If the user wishes to edit the shape of a component, she subdivides its local grid into multiple cells. The system then allocates transformation variables for each cell. An iterative Gauss-Newton solver converges to solutions at rates that support interactive manipulation. From transformations of the coarse cuboid cells, we reconstruct the mesh from moving-least-squares interpolation, similar to [Kaufmann et al. 2008].

In the following, we first introduce necessary notations and then detail the energy and constraint formulations used in the objective function. We then describe the Gauss-Newton method and relevant techniques for numerical acceleration.

4.1 Spatial Grid Generation

In order to decouple motions of different components, we create an individual grid for each identified component. Specifically, we calculate an oriented bounding box (OBB) C_k for each component k. If the user wants to freely deform component k, C_k can further be subdivided into cells using either octree or uniform subdivision. C_k^m denotes the mth cell of component k, and $\mathbf{T}_k^m = \{\mathbf{R}_k^m, \mathbf{p}_k^m\}$ is its associated transformation, where $\mathbf{R}_k^m$ is a 3×3 matrix and $\mathbf{p}_k^m$ is a 3×1 translation vector. The vertices of the cell C_k^m are denoted by $\mathbf{v}_k^{m,i}, i = 0 \ldots 7$.

We define the *influenced region* of a joint as the OBB of the valid vertices from the intersection surfaces that generate the joint. The cells intersecting with or contained by these influenced regions are called influenced cells. All influenced cells from the same component share a common transformation. This is because one joint can only relate two transformations. In Section 4.2.2, $\mathbf{T}_{k1}^m, \mathbf{T}_{k2}^n$ are the two transformations influenced by a particular joint, one for the mth cell from component $k1$, and the other for the nth cell from component $k2$. All other influenced cells have transformations that are identical to either $\mathbf{T}_{k1}^m$ or $\mathbf{T}_{k2}^n$, depending on which component they belong.

4.2 Optimization Objectives

Freeform deformations and joint constraints are implemented by separate terms in the optimization objective function. We seek cell transformations that minimize a weighted sum of the deformation energies and constraint errors.

4.2.1 Deformation Energies

We use cells that are as rigid as possible. Freeform deformation is achieved by allowing different cells, which are "elastically glued" together, to have different transformations. Deformation handles are formulated as positional constraints that can be interactively defined on the complex model and directly manipulated by users.

Rigidity: Rigidity measures how much a cell preserves its original shape. For a rigid transformation $\mathbf{T}_k^m$, $\mathbf{R}_k^m$ is a rotation in SO(3). We measure the rigidity of a transformation by computing the deviation of $\mathbf{R}_k^m$ from a pure rotation [Sumner et al. 2007]:

$$\begin{aligned} Rigid(\mathbf{R}_k^m) \quad = \quad & (\mathbf{c}_{k,1}^m \cdot \mathbf{c}_{k,2}^m)^2 + (\mathbf{c}_{k,1}^m \cdot \mathbf{c}_{k,3}^m)^2 + (\mathbf{c}_{k,2}^m \cdot c_{k,3}^m)^2 \\ & + (\mathbf{c}_{k,1}^m \cdot \mathbf{c}_{k,1}^m - 1)^2 + (\mathbf{c}_{k,2}^m \cdot \mathbf{c}_{k,2}^m - 1)^2 \\ & + (\mathbf{c}_{k,3}^m \cdot \mathbf{c}_{k,3}^m - 1)^2 \end{aligned}$$

where $\mathbf{c}_{k,i}^m, i = 1,2,3$ are the column vectors of matrix $\mathbf{R}_k^m$. The energy term E_{rigid} is formed by accumulating the rigidity of every cell:

$$E_{rigid} = \sum_{k,m} Rigid(\mathbf{R}_k^m) \quad (1)$$

Elastic strain energy: This term measures the local variation of transformations, i.e., differences of neighboring cells' motions [Botsch et al. 2007]. It emulates the ability of elastic materials to resist bending and stretching.

$$E_{strain} = \sum_{k,m} \sum_{j \in \mathcal{N}_k^m} \|\mathbf{T}_k^m \mathbf{v}_k^{m,i} - \mathbf{T}_k^j \mathbf{v}_k^{m,i}\|^2, i = 0 \dots 7 \quad (2)$$

where $\mathcal{N}_k^m$ denotes the set of neighboring cells of C_k^m, and $\mathbf{v}_k^{m,i}, i = 0 \dots 7$ denotes the eight vertices of cell C_k^m. Note that the above formulation only works for uniform subdivisions. For octree-like subdivisions, we use a weighting scheme similar to Botsch et al. 2007.

Position Constraints: Deformation handles allow for direct user manipulations and are therefore commonly considered to be intuitive to use. We support deformation handles by constraining the distance between the actual and desired handle positions:

$$E_{pos} = \sum_i \|\mathbf{T}_k^m \mathbf{v}_i - \mathbf{q}_i\|^2 \quad (3)$$

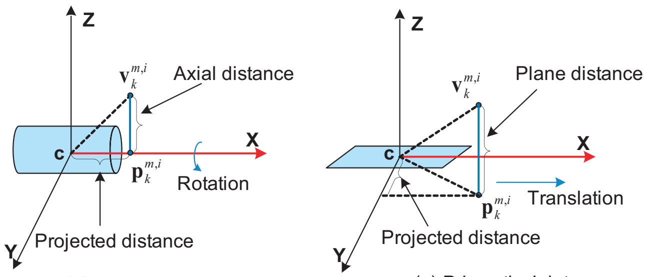
where $\mathbf{v}_i$ is the position of the selected vertex on the model at the reference pose, and $\mathbf{q}_i$ is its target location. $\mathbf{T}_k^m$ is the transformation associated with cell C_k^m which contains vertex $\mathbf{v}_i$. There are usually multiple handles, of which only one is actively controlled by the user at a particular instant in time. We visualize the active handle as a red cube, and inactive handles will be yellow.

4.2.2 Joint Constraint Errors

There are two common ways to represent joint constraints in kinematics [Murray et al. 1994]. One is to use the reduced coordinates, commonly known as joint angles, to parameterize joints. This approach cannot be seamlessly integrated into our cell based deformation framework. The other is to use the full coordinates supplemented with constraints that remove redundant DOFs. By posing joint constraints on the cell transformations in a similar fashion, we can develop a unified optimization framework that can maintain joint constraints and achieve freeform deformations at the same time. Another benefit of the constraint formulation of joints is that it is more modular and flexible than using reduced coordinates, which in consequence greatly simplifies the task of slippable motion analysis and better supports interactive editing of joint types.

Let us denote a joint and its attached local frame by $\{J, \mathbf{c}, \mathbf{F} = \{\mathbf{X}, \mathbf{Y}, \mathbf{Z}\}\}$. J represents the type of the joint, which is an element from the set $\{Revolute, Cylinder, Prismatic, Plane, Ball, Fixed\}$. $\mathbf{F}$ represents the three mutually orthogonal axes of the local joint frame, and $\mathbf{c}$ is its origin. Each type of joint defines a set of geometric invariants, such as distances between cell vertices and/or joint axes. Preserving these invariants during deformation enforces the joint constraints accordingly. We now derive a penalty formulation for each type of joint in Table 1.

Revolute Joint: A revolute joint $\{Revolute, \mathbf{c}, \{\mathbf{X}, \mathbf{Y}, \mathbf{Z}\}\}$ only has one rotational degree of freedom, where $\mathbf{c}$ is its rotation center, and $\mathbf{X}$ its rotation axis. The geometric invariants of a revolute joint are the projected distance and the axial distance between a cell vertex and the rotation axis as illustrated in Figure 3(a). The rotation axis should also remain the same under any valid rotation of the revolute joint. We formulate the three geometric invariants of a revolute joint



(a) Revolute Joint (b) Prismatic Joint

Figure 3: *Geometric invariants of revolute and prismatic joints*

as follows:

$$\begin{aligned} E_{RJ1} &= \sum_i \|(\mathbf{T}_{k1}^m \mathbf{v}_{k1}^{m,i} - \mathbf{T}_{k2}^n \mathbf{c}) \bullet \mathbf{R}_{k2}^n \mathbf{X} - (\mathbf{v}_{k1}^{m,i} - \mathbf{c}) \bullet \mathbf{X}\|^2 \\ E_{RJ2} &= \sum_i (\|\mathbf{T}_{k1}^m \mathbf{v}_{k1}^{m,i} - \mathbf{T}_{k2}^n \mathbf{p}_{k1}^{m,i}\|^2 - \|\mathbf{v}_{k1}^{m,i} - \mathbf{p}_{k1}^{m,i}\|^2)^2 \\ E_{RJ3} &= \|\mathbf{R}_{k1}^m \mathbf{X} - \mathbf{R}_{k2}^n \mathbf{X}\|^2 \end{aligned}$$

where $\bullet$ represents dot product. $\mathbf{T}_{k1}^m$ and $\mathbf{T}_{k2}^n$ are the transformations of two cells influenced by the revolute joint, and $\mathbf{p}_{k1}^{m,i}$ is the projection point of vertex $\mathbf{v}_{k1}^{m,i}$ on the $\mathbf{X}$ axis. Minimizing E_{RJ1} and E_{RJ2} maintains the invariance of the projected and axial distances, and a zero E_{RJ3} enforces a static rotation axis under rotations. The error term of a revolute joint is given by the sum of the above three terms:

$$E_{Rev} = E_{RJ1} + E_{RJ2} + E_{RJ3} \quad (4)$$

Detected joint limits (§3.2) need to be implemented in order to prevent components connected by the revolute joint from penetrating each other. We again express these as geometric constraints. For each cell C_{k1}^m influenced by a revolute joint, we compute a vector $\mathbf{d}_{k1}^{m,i} = \mathbf{v}_{k1}^{m,i} - \mathbf{p}_{k1}^{m,i}$ for its ith vertex. $\mathbf{d}_{k1}^{m,i}$ is then rotated about the rotation axis X. $\mathbf{d}_l^i$ and $\mathbf{d}_u^i$ denote the lower and upper bounds when $\mathbf{d}_{k1}^{m,i}$ reaches the joint limits. We define a penalty term E_{RM} to force vector $\mathbf{d}_{k1}^{m,i}$ lie in between these two limit vectors $\mathbf{d}_l^i$ and $\mathbf{d}_u^i$. When the transformed vector $\mathbf{d}_{k1}^{m,i}$ is within the valid range of motion, the penalty term returns zero. Otherwise it returns the distance to the closest bounding vector $\mathbf{d}^i$:

$$E_{RM} = \sum_i \|(\mathbf{T}_{k1}^{m,i} \mathbf{v}_{k1}^{m,i} - \mathbf{T}_{k2}^n \mathbf{p}_{k1}^{m,i}) - \mathbf{T}_{k2}^n \mathbf{d}^i\|^2 \quad (5)$$

Cylinder Joint: A cylinder joint has one more translational degree of freedom than a revolute joint. Hence we should allow a changeable projected distance during manipulation of the components. We simply remove the projected distance term E_{RJ1} from Equation 4, resulting in the following error term for cylinder joints:

$$E_{Cyn} = E_{RJ2} + E_{RJ3} \quad (6)$$

Prismatic Joint: Prismatic joints are widely used in mechanisms to constrain one component to translate along a fixed axis without any rotation. For a prismatic joint $\{Prismatic, \mathbf{c}, \{\mathbf{X}, \mathbf{Y}, \mathbf{Z}\}\}$, axis $\mathbf{X}$ is the sliding axis along which the component can slide. The geometric invariants of prismatic joints are the distance between a cell vertex and the $\mathbf{XY}$ plane, and the projected distance of a cell vertex on the $\mathbf{Y}$ axis, as illustrated in Figure 3(b). We can specify the geometric invariants as follows:

$$\begin{aligned} E_{PJ1} &= \sum_i \|(\mathbf{T}_{k1}^m \mathbf{v}_{k1}^{m,i} - \mathbf{T}_{k2}^n \mathbf{c}) \bullet \mathbf{R}_{k2}^n \mathbf{Z} - (\mathbf{v}_{k1}^{m,i} - \mathbf{c}) \bullet \mathbf{Z}\|^2 \\ E_{PJ2} &= \sum_i \|(\mathbf{T}_{k1}^m \mathbf{v}_{k1}^{m,i} - \mathbf{T}_{k2}^n \mathbf{c}) \bullet \mathbf{R}_{k2}^n \mathbf{Y} - (\mathbf{v}_{k1}^{m,i} - \mathbf{c}) \bullet \mathbf{Y}\|^2 \end{aligned}$$

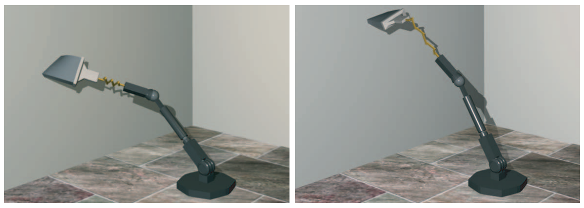


Figure 4: *(a) The user can wipe a wall using the robot brush by directly dragging the tip of the brush (b) The yellow W-shaped segment can extend and fold after the user changes the fixed joints between the rods to revolute joints.*

where $\mathbf{T}_{k1}^{m}$ and $\mathbf{T}_{k2}^{n}$ are cell transformations from components $k1$ and $k2$ respectively that are constrained by the prismatic joint. The total error term for prismatic joints is thus a sum of the above two terms:

$$E_{Prism} = E_{PJ1} + E_{PJ2} \tag{7}$$

In a similar fashion to revolute joints, we denote $\mathbf{d}_{k1}^{m,i}$ as the vector from the cell vertex $\mathbf{v}_{k1}^{m,i}$ to its projection on the **XY** plane $\mathbf{p}_{k1}^{m,i}$. The limit vectors $\mathbf{d}_{l}^{i}, \mathbf{d}_{u}^{i}$ of $\mathbf{d}_{k1}^{m,i}$ can be calculated as before, and a penalty term similar to Equation 5 can be formed.

Plane Joint: Plane joints are used to enforce planar contacts between two components. Their geometric invariant is the distance between a cell vertex and the sliding plane. The error function is simply:

$$E_{Plane} = E_{PJ1} \tag{8}$$

Ball Joint: Ball-and-socket joints have three rotational degrees of freedom and are common in biological systems, such as the hip and shoulder joints of modelled human characters. When two components are connected by a ball-and-socket joint $\{Ball, \mathbf{c}, \{\mathbf{X}, \mathbf{Y}, \mathbf{Z}\}\}$, they both attach to the center $\mathbf{c}$. Consequently, transformations of each component should keep the anchor point $\mathbf{c}$ together. Therefore the error term to impose ball-and-socket joints is:

$$E_{Ball} = \|\mathbf{T}_{k1}^{m}\mathbf{c} - \mathbf{T}_{k2}^{n}\mathbf{c}\|^2 \tag{9}$$

To enforce joint limits for a ball joint, we first decompose the rotation between $\mathbf{T}_{k1}^{m}, \mathbf{T}_{k2}^{n}$ into Euler angles. If they are out of bounds, we project them back into the valid range and compute the corresponding limit vector $\mathbf{d}^{i}$ by this projected valid rotation. The penalty term is then constructed similar to Equation 5.

Fixed Joint: A fixed joint holds two components fixed with respect to each other. All cells influenced by the fixed joint from both components should have identical transformations. This can be implemented as a hard constraint in a pre-processing stage before the optimization to ensure two components will not move relative to each other around the fixed joint.

4.3 Non-linear Optimization

Our shape deformation solves the following unconstrained nonlinear optimization problem:

$$\min_{\mathbf{x}} \mathbf{f}(\mathbf{x})^T \mathbf{f}(\mathbf{x}) = E_{DF} + w_j E_{JNT} + w_p E_{RM} \tag{10}$$

$$\begin{aligned} \text{where: } E_{DF} &= w_{rigid}E_{rigid} + w_{strain}E_{strain} + w_{pos}E_{pos} \\ E_{JNT} &= E_{Rev} + E_{Cyn} + E_{Prism} + E_{Plane} + E_{Ball} \end{aligned}$$

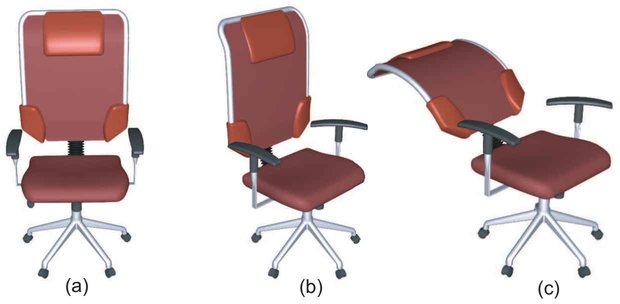


Figure 5: *(a) The original model of an office chair. (b) Seat swivelled and armrests adjusted. (c) Back support tilted and bended.*

Here, $\mathbf{x}$ is the aggregation of all cell transformations $\mathbf{T}_{\mathbf{k}}^{\mathbf{m}}$, $k = 1 \ldots j, m = 1 \ldots j_k$. Since each cell transformation has 12 DOFs, the total dimension of the optimization parameter $\mathbf{x}$ is around twelve times of the number of cells. E_{JNT} enforces joint invariants, and E_{RM} penalizes violations of joint limits. Fixed joints do not appear in the objective function because we explicitly model them by mapping all the transformations of influenced cells to one single transformation variable. Currently we treat all joints equally in the deformation, so that all joint constraints have the same weight. Although there are several weights to be set, we found a large range of values work well and a minimal amount of example-specific tuning was required. For most of the demo examples, we use 1e5 for w_j and w_p, since they represent "hard" joint constraints; 1e4 for w_{rigid} because we want near-rigid cells; 1e3 for w_{strain} to allow deformations; and 1e2 for w_{pos} because we treat user manipulations "softer". To achieve special effects such as the highly stretchable arms and trunk of the Asimo-like robot as shown in Figure 7(d), we use a low $w_{strain} = 10$.

Numerical Solution: We implement an iterative Gauss-Newton method for the above nonlinear least squares problem [Nocedal and Wright 1999; Madsen et al. 2004; Sumner et al. 2007]. At each iteration t, the algorithm solves a linearized subproblem, and computes an updating vector $\mathbf{d}_t$ to improve the current solution $\mathbf{x}_t$:

$$\begin{gathered} \min_{\mathbf{d}_t} \|\mathbf{J}_t \mathbf{d}_t + \mathbf{f}(\mathbf{x}_t)\|^2 \\ \mathbf{x}_{t+1} = \mathbf{x}_t + \mathbf{d}_t \end{gathered} \tag{11}$$

where $\mathbf{J}_t$ is the Jacobian of $\mathbf{f}(\mathbf{x}_t)$.

The analytic Jacobian $\mathbf{J}_t$ is sparse, and its non-zero structure remains the same across iterations. We can thus reuse a pre-computed symbolic factorization of $\mathbf{J}_t^T \mathbf{J}_t$ to accelerate the numerical factorization at every iteration. Furthermore, updating of $\mathbf{J}_t$ can be parallelized on multi-core platforms commonly available today to achieve improved performance. For all our demonstrations, we use the PARDISO (Parallel Direct Sparse Solver) solver from the Intel Math Kernel Library 10.0.

Inverse Kinematics is fundamentally an under-determined problem with possible singular configurations for models with long IK chains. We use the damped pseudo-inverse to achieve singularity robust IK solutions [Parent 2008]. This effectively eliminates oscillatory motions resulting from high joint velocities near a singular configuration, and generally smooths out motions when tracking user-manipulated handles.

5 Results

We demonstrate the capability of the proposed deformation framework on a variety of models in different application scenarios. In addition to the pictorial illustrations shown in this section, readers

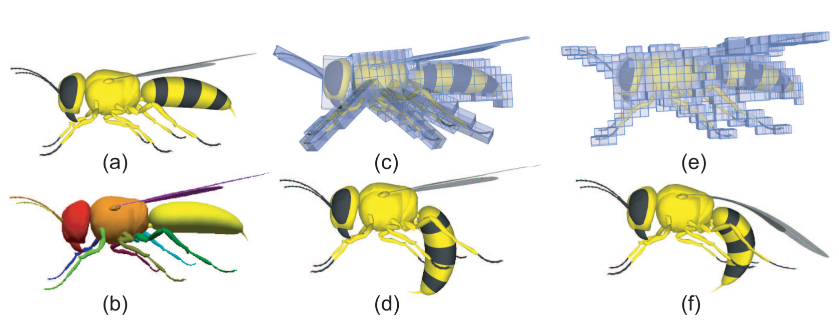


Figure 6: *A comparison of our method and [Botsch et al. 2007]. (a)(b) The input bee model and its components shown in different colors. (c)(d) Our algorithm uses a total of 421 cells defined on component-based multiple grids, and dragging the bee's stinger up and down does not affect the wings or legs, which are attached to the thorax. (e)(f) [Botsch et al. 2007] uses a single spatial grid of 1364 cells, and the deformation of the stinger undesirably disturbs the wings and legs.*

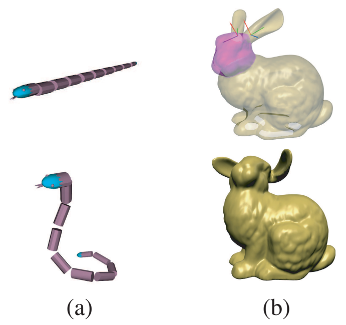


Figure 8: *(a) A cartoon snake with multiple disconnected segments. Virtual ball joints are added between adjacent segments to make the snake dance. (b) The single-component Stanford bunny. Virtual revolute joints are created between the ears and the rigid head.*

are encouraged to see the accompanying video for an interactive and animated demonstration of the results. Table 2 shows performance statistics of our experiments. The accuracy of joint analysis gives the percentage of correctly detected and classified joints out of the total number of joints used in the demos. Interactive performances can be achieved for all models on a 2.99GHz Intel Quad core machine with 4GB of RAM. The time of one Gauss-Newton iteration (column Solve) is related to the number of cells and joints. The number of iterations required depends on the model complexity, the stopping criteria, and the magnitude of user manipulations. In general we observe fast convergence. Qualitatively, three to five iterations can move the handle close to the target. The MLS interpolation time is related to the number of cells and vertices.

Jointed multi-component models: The ideal input to our deformation system are jointed multi-component models, such as CAD models. Figure 4(a) shows the direct manipulation of the tip of the robot brush to wipe a wall. The brush can deform naturally, and the articulation of the mechanical arm operates cooperatively. If the user is not happy with the joint types suggested by the system, she can interactively change the type of joints and produce totally different animations in just a few seconds. As shown in Figure 4(b), she can instantly make the W-shaped segment extend and fold by changing the fixed joints between the yellow rods into revolute joints.

Figure 5 demonstrates an adjustable office chair being manipulated using our deformation system. The seat and armrests can swivel, the armrests can rise and drop, and the back support can tilt and bend. Figure 1(b) shows that an Aibo-like robot dog can be interactively posed to walk and stand up. Its soft tail, body and ears can also deform simultaneously.

Figure 6 compares our method to the approach of [Botsch et al. 2007], showing the strength of component-based discretization and deformation. Without the concept of components, even an adaptive method with much higher cell numbers risks having artifacts. For example, dragging the bee's stinger, which is at the back of its abdomen, affects the bee's wings or legs, which are attached to its thorax. Such undesired correlations between spatially nearby parts which are geodesically and semantically distant, the bee's abdomen and wings in this case, can be eliminated with a smaller number of cells using our method. To eliminate the effect of rotational joints from our method for fair comparison, we use fixed joints to connect all components so that deformation of one component can pass to its neighboring components. Note that the information of components can be used in the method of Botsch et al. as well.

Figure 7(d) demonstrates the advantage of our system over conventional IK with an Asimo-like robot. Since we allow for deformable limbs and accessories (e.g., arm tubes), stretching effects can be easily achieved. Building deformations into an IK system enables artistic exaggerations and supports the artistic license that can be used with respect to rigid skeletons, as suggested by [Lasseter 1987; Harrison et al. 2004]. Figure 7(c) demonstrates motion retargeting of skeletal animations to the robot. Note that the arm tubes deform properly as the motion changes. Chosen body parts can be rigid or deformable to achieve different styles.

Non-jointed Models: We cannot apply the joint analysis algorithm to disconnected multi-component models and single-component models. However, such models can still benefit from our deformation framework with assistance from users to define desired joints.

Figure 8(a) illustrates the effect of joint-aware deformation on a cartoon snake consisting of multiple disconnected segments. We manually create ball joints between adjacent segments to allow the snake to dance. To test our deformation framework for single-component models, we use the Stanford bunny model. We manually create two virtual revolute joints between the ears and the body as shown in Figure 8(b). The head is made fully rigid by designating a single shared transformation for all the cells belonging to the head region. Because of the existence of the virtual joints, the elastic strain energy is discontinuous around the base of the ears. Such deformations are likely difficult to achieve with the user-painted rigidity weights of previous methods.

6 Discussion

The success of joint analysis depends on the existence, resolution, and clean instantiation of the conjunction that represents a joint.

Model	#Vert.	#Cells	#Comp.	#Joints	Accu.	Solve	MLS
Brush	6885	176	15	15	100	14.90	1.35
Lamp	25862	76	6	5	100	3.42	10.40
Chair	23279	227	6	5	100	7.45	6.70
Asimo	38807	477	20	21	81	29.58	13.20
Office	53457	70	25	10	70	3.95	5.89
Aibo	14587	153	13	12	60	6.69	2.80
Bee	10607	421	13	13	NA	12.04	3.58
Snake	597	10	10	9	NA	0.54	0.06
Bunny	34835	1895	1	2	NA	103.09	35.74

Table 2: *Test data and performance statistics. Timing is measured in milliseconds on a 2.99GHz Intel Quad core machine with 4GB of RAM. From left to right: Number of vertices, number of cells, number of components, number of joints, accuracy of joint analysis in percentage, time of one Gauss-Newton iteration, MLS interpolation.*

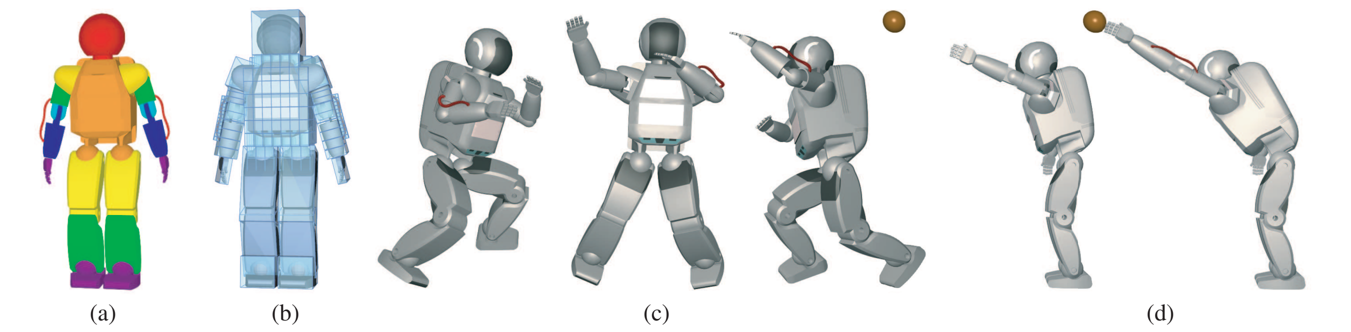


Figure 7: *(a) An Asimo-like robot with 20 components shown in different colors. There is one joint between each pair of adjacent components. Our joint-aware deformations mostly use revolute joints except that the hips are ball-and-socket joints and the wrist and arm tube attaching points are fixed joints. (b) The cell decomposition for deformable motion retargeting and inverse kinematics. (c) The robot driven by motion capture data to shadow box. (d) IK with rigid(left) and deformable(right) body parts.*

The major limitation of this system is that joint analysis is not fully automatic for models that do not have reasonable geometric joints. From our experience, CG modelers tend to only model joints that are visually important when given no instructions on the intended application. For example, the knees of Aibo were modelled properly but the hip and neck joints were skipped. As a consequence we can only analyze 60% of its joints correctly. Figure 9 shows a close-up view of the neck joint of Aibo. No valid vertices can be detected within the intersection region, so the joint is erroneously labelled as a fix joint. The proposed deformation framework mainly targets models that have built-in joint connections in the mesh, such as CAD models for example. However, we are hoping with the existence of a tool like ours, modelers can become more joint-aware for CG models as well.

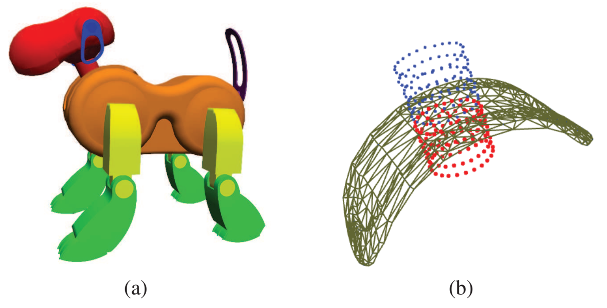


Figure 9: *(a) Color-coded components of Aibo. (b) A close-up view of the neck where it intersects the body. Red vertices are located within the intersecting region, but no body vertices are within the given thresholds of both Euclidean and normal distances. As a result, the joint is labelled fixed.*

Although the joint analysis cannot achieve 100% accuracy, it significantly reduces the amount of manual input required. For models that do not have joints, or only have joints that look right but are mechanically wrong, we allow users to interactively create and edit joints. This is a critical component complementary to the shape analysis step. Automating the addition of virtual joints will considerably enhance the usability for single-component models. Part-aware shape analysis may offer useful suggestions to the users [Liu et al. 2009]. If a set of example poses are available, we can also detect near-rigid components and place joints accordingly [James and Twigg 2005; Theobalt et al. 2007]. We are also interested in examining the possibility of deformable articulation for dynamic animations as in [Faloutsos et al. 1997; Galoppo et al. 2007].

To the best of our knowledge, there are currently no joint-aware deformation tools available in commercial software packages. Imitating soft links in a skeleton requires setting up many small bones and careful tuning of the skinning weights. Plane and cylinder joints are also not supported. We would like to better understand how animators will use a system like ours as they gain experience with it. One potential problem is that animators may require some time to familiarize themselves with the new found freedom presented by a system that integrates both articulation and deformation. For example, achieving effects such as having Asimo stretch its legs while still being bent at the knee requires some thoughtful setup, such as setting a fixed handle at the knee or setting a desired pose, as by default the leg will straighten before it stretches.

7 Conclusion

We have presented an interactive, joint-aware deformation system for complex models. In contrast to previous work that has focused on low-level feature preservation, we consider the constrained spatial relationships represented by joints. Such higher-level semantic between components is an important clue for achieving deformations that preserve the design intent of the modeler and the intent of subsequent user manipulations. Articulation and deformation are integrated seamlessly and flexibly with our method. Joints can be automatically inferred by slippable motion analysis, or interactively defined by users. Space deformation and joint constraints are framed as one nonlinear optimization problem, which is then solved by a fast parallelized Gauss-Newton method. The proposed scheme is demonstrated on a range of models, including connected multi-component models, disconnected multi-component models, and single-component models.

Acknowledgements: Kun Zhou is partially supported by NSFC (No.60825201), the 973 program of China (No.2009CB320801) and NVIDIA. Michiel van de Panne gratefully acknowledges NSERC support. Falai Chen is partially supported by NSFC (No.60873109). We thank Shuo Wang and Mu He for constructing some of the mesh models. The mocap data for retargeting was obtained from mocap.cs.cmu.edu (NSF EIA-0196217).

References

ATTENE, M., FALCIDIENO, B., AND SPAGNUOLO, M. 2006. Hierarchical mesh segmentation based on fitting primitives. *The Visual Computer 22*, 3, 181–193.

AU, O. K.-C., TAI, C.-L., CHU, H.-K., COHEN-OR, D., AND LEE, T.-Y. 2008. Skeleton extraction by mesh contraction. *ACM Trans. Graphics 27*, 3, 1–10.

BARAN, I., AND POPOVIĆ, J. 2007. Automatic rigging and animation of 3d characters. *ACM Trans. Graphics 26*, 3, 72.

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

BOTSCH, M., PAULY, M., GROSS, M., AND KOBBELT, L. 2006. Primo: coupled prisms for intuitive surface modeling. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, 11–20.

BOTSCH, M., PAULY, M., WICKE, M., AND GROSS, M. 2007. Adaptive space deformations based on rigid cells. *Computer Graphics Forum 26*, 3, 339–347.

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

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 1997. Dynamic animation synthesis with free-form deformations. *IEEE Transactions on Visualization and Computer Graphics 3*, 3, 201–214.

FU, H., COHEN-OR, D., DROR, G., AND SHEFFER, A. 2008. Upright orientation of man-made objects. *ACM Trans. Graphics 27*, 3, 1–7.

GAIN, J., AND BECHMANN, D. 2008. A survey of spatial deformation from a user-centered perspective. *ACM Trans. Graphics 27*, 4, 1–21.

GALOPPO, N., OTADUY, M. A., TEKIN, S., GROSS, M., AND LIN, M. C. 2007. Soft articulated characters with fast contact handling. *Computer Graphics Forum 26*, 3 (Sept.), 243–253.

GELFAND, N., AND GUIBAS, L. J. 2004. Shape segmentation using local slippage analysis. In *SGP '04: Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, 214–223.

HARRISON, J., RENSINK, R. A., AND VAN DE PANNE, M. 2004. Obscuring length changes during animated motion. *ACM Trans. Graphics 23*, 3, 569–573.

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

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

KATZ, S., AND TAL, A. 2003. Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM Trans. Graphics 22*, 3, 954–961.

KAUFMANN, P., MARTIN, S., BOTSCH, M., AND GROSS, M. 2008. Flexible simulation of deformable models using discontinuous galerkin fem. In *2008 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 105–115.

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

KRAEVOY, V., SHEFFER, A., SHAMIR, A., AND COHEN-OR, D. 2008. Non-homogeneous resizing of complex models. In *ACM SIGGRAPH Asia 2008*, 1–9.

LASSETER, J. 1987. Principles of traditional animation applied to 3d computer animation. In *SIGGRAPH Conference Proceedings*, 35–44.

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

LI, W., AGRAWALA, M., CURLESS, B., AND SALESIN, D. 2008. Automated generation of interactive 3d exploded view diagrams. *ACM Trans. Graphics 27*, 3, 1–7.

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

LIU, R., ZHANG, H., SHAMIR, A., AND COHEN-OR, D. 2009. A part-aware surface metric for shape analysis. In *Eurographics*, vol. 28, to appear.

MADSEN, K., NIELSEN, H., AND TINGLEFF, O. 2004. Methods for nonlinear least squares problems. Tech. rep., Informatics and Mathematical Modelling, Technical University of Denmark.

MAGNENAT-THALMANN, N., LAPERRIERE, R., AND THALMANN, D. 1988. Joint dependent local deformations for hand animation and object grasping. In *Graphics Interface*, 26–33.

MITRA, N. J., GUIBAS, L. J., AND PAULY, M. 2007. Symmetrization. *ACM Trans. Graphics 26*, 3, 63.

MURRAY, R. M., LI, Z., AND SASTRY, S. S. 1994. *A Mathematical Introduction to Robotic Manipulation*. CRC Press.

NOCEDAL, J., AND WRIGHT, S. J. 1999. *Numerical Optimization*. Springer.

PARENT, R. 2008. *Computer Animation: Algorithms and Techniques*, second ed. Morgan Kaufmann.

POPA, T., JULIUS, D., AND SHEFFER, A. 2006. Material-aware mesh deformations. In *Proceedings of the IEEE International Conference on Shape Modeling and Applications 2006*, 22.

SEDERBERG, T. W., AND PARRY, S. R. 1986. Free-form deformation of solid geometric models. In *SIGGRAPH Conference Proceedings*, 151–160.

SHI, X., ZHOU, K., TONG, Y., DESBRUN, M., BAO, H., AND GUO, B. 2007. Mesh puppetry: cascading optimization of mesh deformation with inverse kinematics. *ACM Trans. Graphics 26*, 3, 81.

SORKINE, O., LIPMAN, Y., COHEN-OR, D., ALEXA, M., ROSSL, C., AND SEIDEL, H.-P. 2004. Laplacian surface editing. In *Proceedings of the Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*, 179–188.

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

SUMNER, R. W., SCHMID, J., AND PAULY, M. 2007. Embedded deformation for shape manipulation. *ACM Trans. Graphics 26*, 3, 80.

THEOBALT, C., RÖSSL, C., DE AGUIAR, E., AND SEIDEL, H.-P. 2007. Animation collage. In *Symposium on Computer Animation*, 271–280.

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. Graphics 23*, 3, 644–651.

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