EUROGRAPHICS 2012 / P. Cignoni, T. Ertl
(Guest Editors)

Volume 31 (2012), Number 2

Automatically Rigging Multi-component Characters

Gaurav Bharaj, Thorsten Thormählen, Hans-Peter Seidel, Christian Theobalt

MPI Informatik, Germany

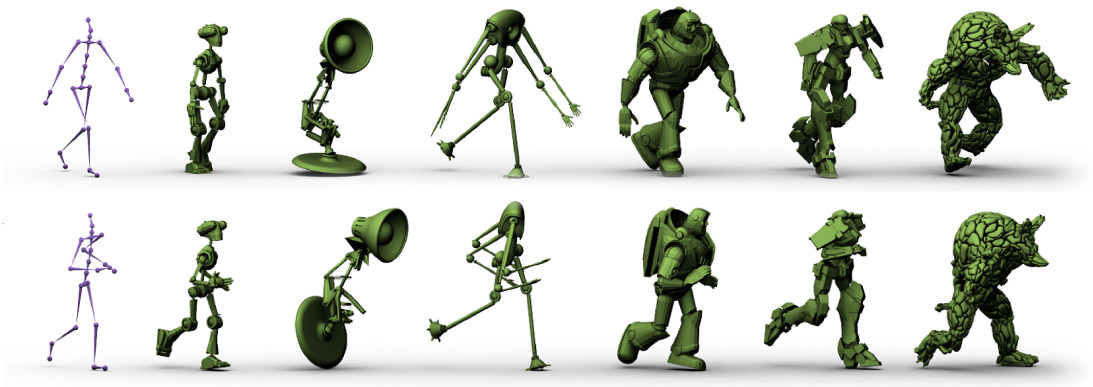

Figure 1: Our approach creates rigs for multi-component meshes that can be mapped to an input animation skeleton (far left).

Abstract

Rigging an arbitrary 3D character by creating an animation skeleton is a time-consuming process even for experienced animators. In this paper, we present an algorithm that automatically creates animation rigs for multi-component 3D models, as they are typically found in online shape databases. Our algorithm takes as input a multi-component model and an input animation skeleton with associated motion data. It then creates a target skeleton for the input model, calculates the rigid skinning weights, and a mapping between the joints of the target skeleton and the input animation skeleton. The automatic approach does not need additional semantic information, such as component labels or user-provided correspondences, and succeeds on a wide range of models where the number of components is significantly different. It implicitly handles large scale and proportional differences between input and target skeletons and can deal with certain morphological differences, e.g., if input and target have different numbers of limbs. The output of our algorithm can be directly used in a retargeting system to create a plausible animated character.

Categories and Subject Descriptors (according to ACM CCS): I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—Animation

1. Introduction

Computer animation draws a lot of its appeal from being able to animate the impossible. Often it is the case that man-made objects are to be animated. Famous examples are the two lamps in Pixar's Luxo Jr. (1986), or multi-component characters in feature films, such as in Toy Story (1995), Robots (2005), Transformers (2007), etc. Creating such animations is a technically challenging process that even takes experienced artists days if not weeks of work. In a common scenario, the artist is provided with input motion in the form of a motion skeleton that was captured through motion capture or key-framed by an artist. That motion has to be transferred to a target character which is only given as a static shape representation, but without an animation skeleton.

In a process called rigging, the artist manually defines a skeleton for the target model and attaches geometric segments to it through surface skinning. The target skeleton does not only need to conform with the structure of the input motion skeleton, it also needs to comprise joints that are placed in physically appropriate locations. Finally, the animator needs to define correspondences between input and

target skeleton joints, before the target rig and input motion can be passed to a retargeting system to synthesize the final animation. Very common challenges arise if input and output skeletons are of very different proportions, or if they are morphologically different, e.g., if the target model has more limbs than the input animation skeleton.

In this paper, we propose a new algorithm that automatically creates skeleton rigs for animation of multi-component character models, as they are typically found in online shape repositories. Multi-component models are composed of tens to hundreds of individual closed triangle meshes, however without component labels. Our algorithm expects as input a multi-component model and a moving animation skeleton. The output is a target skeleton for the model, including rigid skinning weights, as well as joint correspondences between the input animation skeleton and the target skeleton.

The algorithm resides on the following core contributions:

- a new approach for skeleton generation that starts from a contact graph between components and simplifies this initial representation by graph clustering to derive a meaningful target skeleton with rigid skinning weights.
- a multi-component many-to-one joint mapping scheme between input and target skeleton, based on dynamic programming.

Our method also succeeds if the number of components in the model is orders of magnitude different from the number of joints in the input animation, and it is robust to differences in skeletal dimensions. It inherently handles certain morphological differences between input skeleton and target, such as different numbers of limbs. The approach proposes plausible skeleton locations, despite the fact that joints are typically not modeled explicitly in most multi-component models. Joint correspondences between input and target skeleton are computed in such a way that the structure of the input skeleton is reflected, and thus, the output animation preserves the characteristics of the input. Furthermore, our algorithm is robust to varying input mesh resolution, and reliably finds rigs even for input shapes made of thousands of components. The method is generic and succeeds without user intervention and does not requires any semantic labelling, like in [HRE*08], or point-by-point correspondence, as in [ZXTD10].

2. Related Work

In this section, we review important related work for subproblems that we address in our approach.

Automatic Rigging. Many related papers attempt to animate collections of mechanical components, assuming that interactions between components have been modeled in a physically plausible way, i.e., through explicitly modeled joint geometry or plausibly defined interactions at contact surfaces. Hahn's approach [Hah88] animates rigid body assemblies by studying their physical characteristics such as friction, mass and moment of inertia. The system does not attempt to animate as complex multi-component models as we aim for. Sims [Sim94] presents techniques to evolve locomotion of morphologically varying simple creatures using an artificial neural networks-based learning approach. Xu et al. [XWY*09] animate 3D (CAD) models, but expect correctly modeled geometry of real joints including, e.g., revolute, prismatic, or ball-and-socket joints. Other methods, such as proposed by Baran et al. [BP07], automatically rig single-component deformable models. Hecker et al. [HRE*08] describe an approach to rig user-created virtual characters. However, their approach expects explicit user-input, e.g., user-given semantic labels for rigged limbs, such as arms or legs. Another animation paradigm is mesh-based deformation transfer between two models with known surface correspondences [ZXTD10, SP04]. In contrast, our system rigs and animates 3D models composed of hundreds of small components using a fully-automatic pipeline.

Shape analysis. Shape analysis for multi-component CAD models has recently made great strides ahead. Some works explore how components mechanically interact with each other in a physically correct sense. There are approaches based on slippage analysis [GG04, XWY*09] that try to extract correct mathematical joint types from modeled joint geometry. Recently, Mitra et al. [MYY*10] explored how statically modeled mechanical assemblies would move; however CAD models with detailed modeled geometry are required.

In the models from online databases, which we explored, most models are created for artistic purposes and not industrial CAD. Intuitively, we know how the components of the artistic models are supposed to interact with each other. However, a closer inspection shows that artists generally refrain from the effort to model contact surfaces or actual geometry of joints explicitly. As a consequence, slippage analysis is not applicable for most of these models.

Skeleton creation, fitting and skinning. Katz and Tal [KT03] and Au et al. [ATC*08] propose methods to extract skeletons for single-component deformable models. Baran et al. [BP07] propose a methods to embed a given skeleton into a single-surface biped character and compute skinning weights. Hecker et al. [HRE*08] model a character by combining components with known skeletons using user-provided semantic component labels. Other approaches, such as [XWY*09], use multi-component CAD models and convert the meshes into a single voxel grid to animate it. Such a method does not directly comply with the established animation pipeline in the industry and thus, requires a completely different approach than skeleton retargeting for animation.

Our method generates an animation rig for arbitrary compartmentalized rigid body assemblies. Hence approaches where skinning is calculated for single shell surfaces, such as in [BP07] and [KSO10], cannot be applied directly.

Finding correspondences for retargeting. In skeleton-

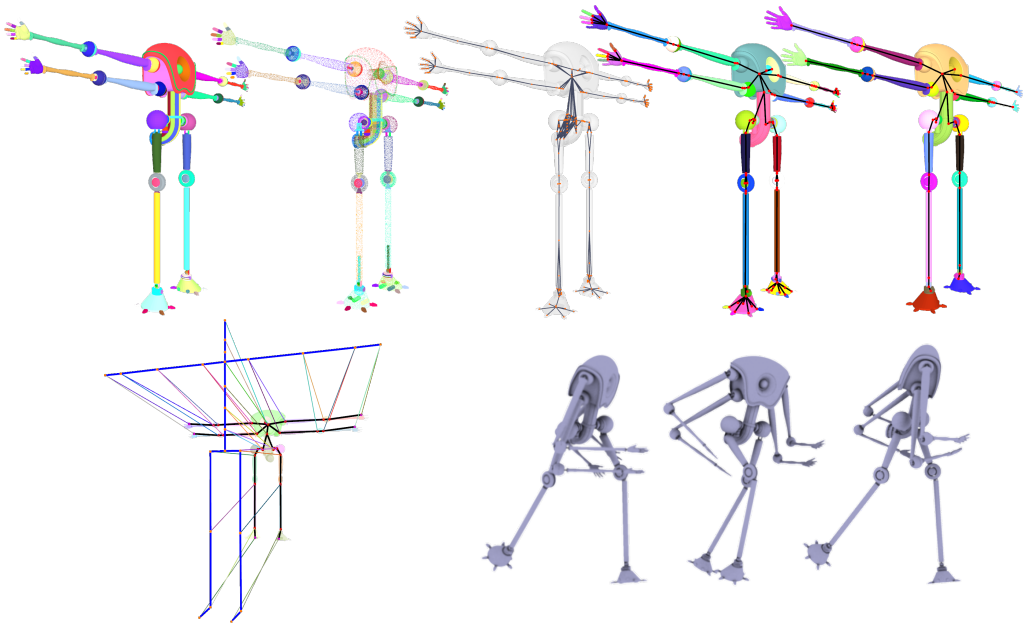

Figure 2: Overview of our approach (top left to bottom right): segmented input mesh; point based representation; extracted contact graph; high-resolution clustering of the contact graph (defining the output skeleton); low-resolution clustering; input animation skeleton and joint mapping to the output skeleton; different poses of rigged input mesh that are generated with the joint mapping.

based animation, the term retargeting refers to the process of mapping joint angle data from an input skeleton to a target skeleton such that certain constraints for plausible motion are fulfilled [Gle01, CK99]. Alternatively, in a skeletonless domain, retargeting can implicitly be obtained by resorting to mesh-based animation transfer [ZXTD10, SP04]. Also similar in spirit is the work [YAH10] in which the authors transfer stylized motion between characters. All approaches require semantic correspondences between input and output skeletons or surfaces, that are typically provided by the user. In contrast, our approach computes such correspondences automatically.

3. Automatic Rigging and Joint Mapping

An overview of the proposed approach for automatic rigging and joint mapping of a multi-component character is shown in Fig. 2. The input to our pipeline is a 3D mesh that is composed of many individual components. Each component's surface is sampled to a point cloud. Based on the point cloud data, a contact analysis of components is performed. This results in a graph in which each component is represented by a node, and in which there is an edge between two nodes if the corresponding components are in contact. This contact graph is typically quite complex and is not directly suited for rigging of the input model. The contact graph is then simplified and transformed into a tree by clustering nodes. Thereby, a high- and a low-resolution clustering of the graph are generated. A second input to our pipeline is an input animation skeleton. Correspondences are estimated between the clustered (high- and low-resolution) contact graphs and the joints of the input animation skeleton. This results in a rigged input mesh, i.e., an output skeleton for the multi-component model comprising a bone hierarchy and interconnecting joints, as well as rigid skinning weights associating each component with the appropriate bone. The model can be animated by retargeting the animation data of the input skeleton to the generated rig.

In the following sections each step of our automatic rigging and animation pipeline is explained in detail.

3.1. Inputs

Our approach typically requires two inputs: firstly, a user provided 3D mesh and, secondly, an input animation skeleton.

We expect that the 3D mesh is composed of many individual components, but component labels for triangles are not available, i.e., the model is a priori just a set of unclustered triangle patches. This is typically the case for man-made objects, such as robots, ships, cars, household appliance, etc. In particular, our approach does not work for meshes that are made out of a single shell, as it is often the case for 3D models of humans or animals.

Input 3D meshes can be modeled manually in any 3D modelling package or can be found, for example, in Internet model databases, such as Dosch Design, Turbosquid, or GoogleWarehouse. The majority of 3D meshes in such internet databases are not rigged with a skeleton for animation. The second input is an animation skeleton. This skele-

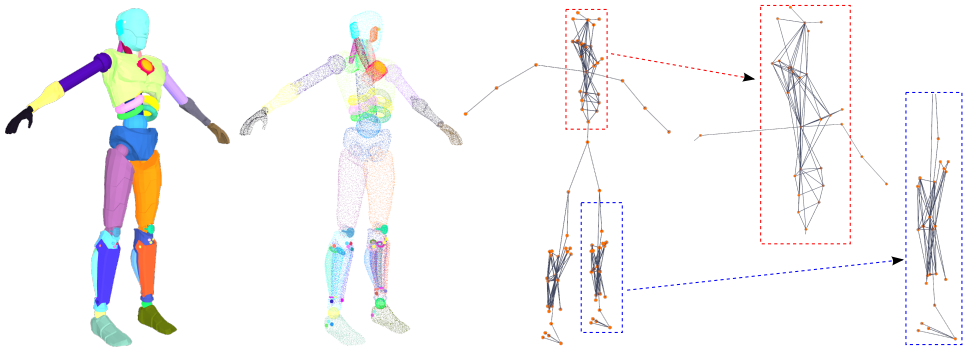

Figure 3: Generation of a contact graph: (left to right) multi-component mesh, point cloud, calculated contact graph, high frequency components, i.e., overly densely connected components of the contact graph: magnification of the chest region, magnification of the foot region

ton is given by a tree hierarchy of joints that are connected by bones. Motion data defining joint transformations are also provided. Such animated skeletons can be either modeled manually or can be found in motion-capture databases, such as the CMU Mocap Database or the Documentation Mocap Database (HDM05).

3.2. Contact Graph

In order to transfer the motion of the animation skeleton to the input 3D mesh, we need to establish correspondences between all components of the 3D mesh and the joints of the animation skeleton. This is a quite challenging task. Firstly, there is no one-to-one mapping between components and joints. Instead many components can be associated with the same joint. Secondly, the structure of the input mesh may be significantly different from that of the animation data, e.g., there may be different numbers of arms or legs. Consequently, in order to solve this challenging correspondence problem, we need to extract an output skeletal structure from the input mesh that is compatible with the animation skeleton.

The task of extracting an initial skeletal structure for a multi-component input mesh is very different from generating a skeleton for a mesh that is made out of a single shell. Unfortunately, despite the comparmentalized geometry, it is still not trivial to precisely determine appropriate joint locations that will make the model articulate in a plausible manner. The reason is that in most multi-component models, joints or contact surfaces were not modeled as real geometric entities, which greatly reduces modeling time, but removes valuable information that could be exploited in skeleton extraction. Thus we have to develop a strategy to determine potential joint positions, despite this missing geometry information. For multi-component meshes the connectivity between components is already an important cue and it is a valid assumption that points of articulation lie near contact points between components. Thus, the first step in our processing pipeline is to build a *contact graph*.

In a contact graph $\alpha(N,E)$ every node N_i is a mesh component and there exists an edge $E_{i,j}$ if the component N_i and N_j are in contact.

The input mesh is segmented into components by searching for connected polygons in the polygonal mesh. A component is a set of polygons of the input model. The process of assigning polygons to components is a simple repetitive algorithm. First a new component is created. We then start at any polygon and perform region-growing over the edges of the polygonal mesh. Each polygon that is reachable by region-growing belongs to the same component and we add it to this component's polygon set. Once region growing has stopped, we repeat the process by creating the next component and perform region-growing over all polygons that are not yet assigned to that component. This process stops once all polygons are assigned to a component.

We now want to determine which components are in contact. To avoid problems with non-uniform mesh resolution, we first resample all component surfaces into point clouds using Hammersley point sampling as proposed in [WLH97]. We also compute the oriented bounding-box of each component. Then, for any two given components, we check if their oriented bounding-boxes collide. If this is the case, we check if any two points from the components are less than a distance threshold ε apart. We empirically chose ε to be 0.01 percent of the complete mesh size. For any two components N_i and N_j where two points are close enough, we generate a contact, i.e., an edge $E_{i,j}$. An example for a segmented input mesh and the generated contact graph is shown in Figure 3.

3.3. Graph Clustering

The contact graph is already a quite good representation of the skeletal structure of the input mesh. In theory, one could think about using graph matching between the contact graph and the input animation skeleton to derive the correspondence. However, the resolution and structure of the contact graph and the animation skeleton are very dif-

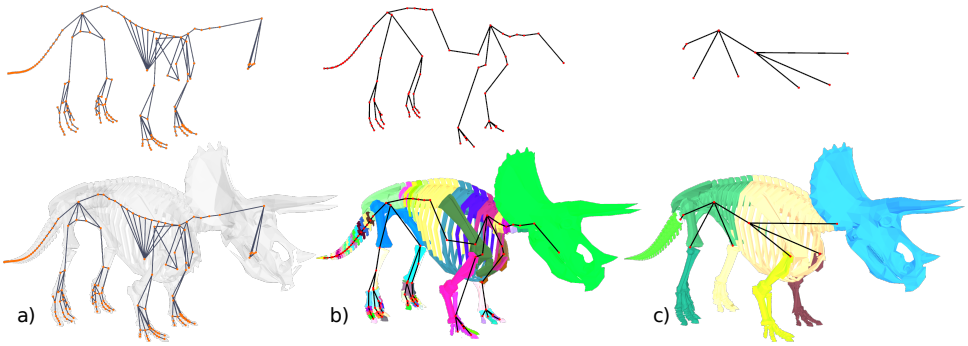

Figure 4: Result of graph clustering: a) Contact graph and input mesh, b) high-resolution clustering (those components of the mesh that belong to the same cluster center are indicated by the same color), c) low resolution graph

ferent and, therefore, graph matching techniques, such as [KA09, LH05, GR96], do not result in reasonable mappings. Furthermore, general graph matching approaches do not handle the problem of many-to-one mapping if the animation skeleton and the input mesh are morphologically different, e.g., have different numbers of limbs. Hence, we need to generate a simpler skeletal structure for the input mesh that is more suited for comparison with the animation data.

In particular, as the animation skeleton is essentially a tree hierarchy, we also need to extract a suitable tree from our contact graph. Additionally, the contact graph features subsets of overly densely connected segments, in other words, high-frequency components that are far too detailed to make up plausible collections of articulated components, cp. Fig. 3. We want to simplify and cluster these high-frequency components to establish information about the coarser semantically more plausible structure of the input mesh. We thereby algorithmically imitate the work of a human animator who would look at the input from a more high-level perspective, seeking functionally plausible structures.

Such a coarse version of the contact graph $\alpha(N,E)$ can be generated by clustering some of its nodes N_i. For this purpose we adopt a Quadratic Error Metric(QEM)-like method introduced in [GH97], where the application is mesh simplification. It was shown before by Au et al. [ATC*08] that this approach can be applied for graph clustering as well. Conceptually, an edge E_{ij} of the graph is treated as an edge of a triangle and a node N_i is equivalent to a vertex of a geometric mesh.

Clustering algorithm. The clustering algorithm iteratively collapses edges of the graph until there are no more edges to collapse. Let the 3-vector $\mathbf{n}_i = (n_x, n_y, n_z)^\top$ be the center of mass of the point cloud of the components that is associated to node N_i. If an edge E_{ij} collapses, the resulting collapsed node center $\hat{\mathbf{n}}$ is always calculated as the geometric average of the two involved node centers $\mathbf{n}_i$ and $\mathbf{n}_j$. We define an edge E_{ij} to be collapsible if it fulfils criterion A, which requires that

A.1 the degree of node N_i or N_j is larger than 2, or

A.2 there is more than one path that connects the nodes N_i and N_j, or

A.3 the surface area of components assigned to either N_i or N_j is smaller than a given threshold, or

A.4 the length of edge E_{ij} is smaller than a given threshold.

The collapsibility criteria formalize the strategy for graph clustering that was intuitively outlined above. The output skeleton should not contain too densely connected sections of too small components, such sections are likely to belong to the same entity, i.e., the same bone. Also, if many links exist between smaller components, a main path is to be identified, that corresponds to the principal chain of bones. To determine which edge to collapse in each iteration, each collapsible edge is assigned an edge cost $C(E_{ij})$. In each iteration the edge with the smallest edge cost is collapsed. The edge cost is defined as

$$C(E_{ij}) = w_1\, C_{\text{Shape}}(E_{ij}) + w_2\, C_{\text{Samp}}(E_{ij}) \quad , \qquad (1)$$

where C_{Shape} is a shape cost and C_{Samp} a sampling cost and their weights are chosen to be $w_1 = 1.0$ and $w_2 = 0.1$, respectively. The shape cost helps in preserving the overall shape of the graph, while the sampling cost helps to prevent the generation of long edges. The shape cost is given by

$$C_{\text{Shape}}(E_{ij}) = \varepsilon(\hat{\mathbf{n}})_i + \varepsilon(\hat{\mathbf{n}})_j \qquad (2)$$

where the error metric $\varepsilon(\hat{\mathbf{n}})_i$ and $\varepsilon(\hat{\mathbf{n}})_j$ is the sum of distances of the collapsed node center $\hat{\mathbf{n}}$ to all of the adjacent edges of $\mathbf{n}_i$ and $\mathbf{n}_j$, respectively, which can be calculated by

$$\varepsilon(\mathbf{p})_i = \check{\mathbf{p}}^\top \left[\sum_{(i,j)\in\mathcal{E}_i} \mathtt{K}_{ij}^\top \mathtt{K}_{ij} \right] \check{\mathbf{p}} = \check{\mathbf{p}}^\top \mathtt{Q}_i \check{\mathbf{p}} \quad , \qquad (3)$$

where $\check{\mathbf{p}} = (p_x, p_y, p_z, 1)$ is the homogeneous representation of the 3-vector $\mathbf{p}$, $\mathcal{E}_i$ is the set of edges connected to node N_i, and the 3×4 matrix $\mathtt{K}_{ij}$ is given by

$$\mathtt{K}_{ij} = \begin{bmatrix} 0 & -a_z & a_y & -b_x \\ a_z & 0 & -a_x & -b_y \\ -a_y & a_x & 0 & -b_z \end{bmatrix}$$

with $\mathbf{a} = (\mathbf{n}_i - \mathbf{n}_j)/\|\mathbf{n}_i - \mathbf{n}_j\|$ and $\mathbf{b} = \mathbf{a} \times \mathbf{n}_i$.
The sampling cost penalizes edge collapses that would generate long edges and is given by

$$C_{\text{Samp}}(E_{ij}) = \|\mathbf{n}_i - \mathbf{n}_j\| \sum_{(i,k)\in\mathcal{E}_i} \|\mathbf{n}_i - \mathbf{n}_k\| \quad . \tag{4}$$

Once the edge with the lowest edge score is found, this edge is collapsed. The collapsed node center $\hat{\mathbf{n}}$ is given by the arithmetic average $\hat{\mathbf{n}} = 0.5(\mathbf{n}_i + \mathbf{n}_j)$. All adjacent edges of N_i and N_j become adjacent edges of the collapsed node $\hat{N}$ (except for duplicates). Furthermore, the merged matrix $\hat{\mathtt{Q}}$ of the collapsed node is given by $\hat{\mathtt{Q}} = \mathtt{Q}_i + \mathtt{Q}_j$. In the next iteration N_i and N_j are removed and the collapsed node $\hat{N}$ is treated exactly as any other node N of the graph. The only difference is that its merged matrix $\hat{\mathtt{Q}}$ is used in Eq. 3 instead of calculating an updated $\mathtt{Q}$ via the sum over the current $\mathtt{K}_{ij}$. This iterative process is repeated until there are no more collapsible edges according to criterion A. During graph clustering we maintain a history of collapsed nodes by recording which nodes were merged in each iteration. This information will become relevant later on in the pipeline.

Low and high-resolution graph. We generate two different versions of the clustered graph, namely a *high-resolution graph* and a *low-resolution graph*. The low-resolution graph is generated as described above with all four sub-criteria of criterion A enforced.
The high-resolution graph is generated by enforcing only sub-criteria A.1 and A.2. By dropping sub-criteria A.3 and A.4, we keep all nodes which are leafs or nodes that are reachable only via a single path, no matter how small their surface area or what the edge length is.
Although both representations preserve the overall shape of the multi-component mesh, each has its own advantages. While the high-resolution graph gives more details, the low-resolution graph better represents the coarse structure of the mesh. Fig. 4 shows the generated low and high-resolution graph for an example model.

Tree extraction. Because of sub-criterion A.2 each node in the clustered graph is reachable only via a single path. Such an undirected, acyclic graph can be directly transformed into a tree if one of its nodes is selected as root node. We choose the root node to be the node N_i of the original contact graph $\alpha(N,E)$ with the highest betweenness-centrality. If the node N_i is no longer available in the clustered graph, we use the recorded clustering history to find out its representative node (i.e., the node evolving from it) in the clustered graph and use this as the root of the tree. The betweenness-centrality takes into consideration both the local and global structure and connectivity of a graph [Fre77] and determines the most important node in the contact graph. We also experimented with eigen-, degree-, and closeness-centrality and found that betweenness-centrality works best. After this stage, we have two versions of an output animation skeleton for the shape model, one in high resolution, and one in low resolution.

Rigid Skinning. As we have maintained a history of collapsed nodes during graph clustering, we can determine which node from the contact graph is assigned to a node in the final clustered tree. We also know which vertices of the multi-component mesh belong to which node of the contact graph. This information can be used to assign a vector of rigid skinning weights to each vertex of the multi-component mesh. Each element in this vector represents a node in the final clustered tree. In rigid skinning, a vector element is either 1 for the node of the final clustered tree to which the vertex belongs, or 0 for all other nodes of the clustered tree. In Fig. 4 the rigid skinning is visualized. Each set of vertices that belongs to a particular node of the clustered tree is shown in a different color. The advantage of rigid skinning is that our results are directly compatible with the linear blend skinning (LBS) [MTLT88] format that is supported by almost all professional 3D animation packages; however, since each vertex is uniquely assigned to one node, there is no deformation blending around joints. Instead, each vertex is moves rigidly with the transformation of to its assigned node. This has the desired effect that all components of our multi-component mesh that belong to a particular node of the clustered tree move as a rigid cluster during animation.

3.4. Joint Mapping

Retargeting algorithms, such as [Gle01, CK99] or Autodesk's HumanIK typically produce good results. Hecker et al. [HRE*08] propose an automatic retargeting system; however, it is assumed that the semantic labelling of each limb is already known. For 3D models from internet databases typically no such semantic information is provided. Hence, most retargeting pipelines require a manual step, i.e., a user manually defines the mapping between the joints of the target skeleton and the input animation skeleton. In this section we propose an algorithm, which calculates this mapping automatically. The method capitalizes on the extracted high- and low-resolution graphs from the previous section.
Essentially joint-mapping is a graph (or tree) matching problem. Again, it would be possible, in theory, to apply general graph matching approaches [LH05,GR96,KA09] to find a mapping between our clustered graph and the animation skeleton. This will work better than trying to find a mapping between the contact graph and the animation skeleton directly. However, it is still unreliable and does not scale to models with several hundreds of components if the input animation skeleton is comparably coarse. Also an approach similar to Electoral-voting [ATCO*10] cannot be used with our setup because we also want a mapping for characters where input skeleton and output skeleton morphologies differ (i.e., allow for many-to-one limb matching). We also want to find mappings for characters that are not modeled exactly in a reference pose, such as a T-pose.
Further on, our output skeleton's structure may still not exactly match the structure of the input animation skeleton, for instance since the number of joints in certain sub-chains may

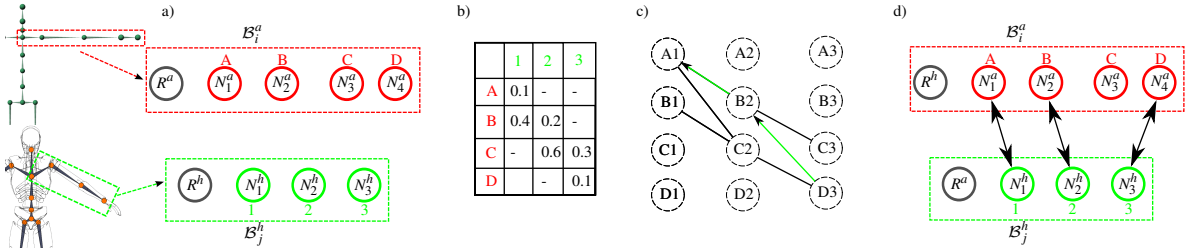

Figure 5: Matching nodes of two limbs: a) the nodes N_t^a of the branch $\mathcal{B}_i^a$ of the animation skeleton tree (top) are matched to the nodes N_s^h of the branch $\mathcal{B}_i^h$ of the high-resolution tree (bottom), R denotes the root node; b) cost $K(N_s^h, N_t^a)$ of assigning two node c) because nodes can only be matched in consecutive order, only certain assignments are possible as shown here in this Trelis diagram. The assignment with the lowest branch cost Q(green path) can be found by dynamic programming; d) generated mapping with the lowest branch cost.

differ. There is thus no unique way of mapping each input skeleton joint to an output skeleton joint, and there is no unique way of leaving joints unmapped, i.e., making them rigidly move with the nearest mapped joint in the hierarchy. We therefore want to develop a mapping approach that automatically creates the most plausible set of joint correspondences, such that the structure of the input skeleton is best mapped to the output rig, and the essence of the input animation can be transferred to the target model.

In the following, we propose a formulation for skeletal joint mapping that meets these requirements. The joint mapping uses the character's high- and low-resolution skeleton-trees, which are generated with the graph clustering approach of the previous section and denoted here as $\tau^h(N^h, E^h)$ and $\tau^l(N^l, E^l)$, respectively. The structural basis for the final mapped output skeleton is $\tau^h(N^h, E^h)$, while $\tau^l(N^l, E^l)$ merely supports the algorithm in finding the best mapping. The input animation skeleton is also given as a tree structure and is denoted by $\tau^a(N^a, E^a)$. The idea is to identify the number of limbs in the input mesh by the number of branches in the low-resolution tree. A branch $\mathcal{B}$ is defined as the set of all nodes N_i that lie on the path from the root node R to a leaf node. For example, in Fig. 4c the low-resolution tree has six branches.

Joint-mapping algorithm. The actual joint-mapping between the j-th branch $\mathcal{B}_j^a$ of the animation skeleton-tree and the output skeleton is performed by matching nodes on the branches $\mathcal{B}_i^h$ of the high-resolution tree (as only the branches of the high-resolution tree contain a sufficient number of nodes). Typically several branches of the high-resolution tree are overlapping with a branch from the low-resolution tree, and consequently, we know that we only need to assign a mapping to one of these branches in the end. To find out which of these branches to choose, we calculate a mapping for all these branches of the high-resolution tree and keep only the one with the lowest mapping cost.

More formally, for each branch $\mathcal{B}_i^h$ of the high-resolution tree there must exist a corresponding branch $\mathcal{B}_j^a$ of the animation skeleton and for all nodes of this branch there must also exist a corresponding node in the animation tree branch:

$$\forall \mathcal{B}_i^h : \exists\, \mathcal{B}_j^a \quad \text{and} \quad (\forall N_s^h \in \mathcal{B}_i^h) : (\exists\, N_t^a \in \mathcal{B}_j^a) \quad . \tag{5}$$

Furthermore, the mapping of nodes must be in consecutive order, which means that if N_s^h maps to N_t^a, then N_{s+1}^h can only map to N_{t+u}^a with $u > 0$. This ordering constraint also ensures that there are no double assignments of nodes. Here and in the following description of the algorithm, we assume that there are fewer nodes in branch $\mathcal{B}_i^h$ than in branch $\mathcal{B}_j^a$. If this is not the case, the same algorithm is used to find a mapping but the role of the branches $\mathcal{B}_i^h$ and $\mathcal{B}_j^a$ are swapped.

Let I be the number of branches $\mathcal{B}_i^h$ in the high-resolution tree and J the number of branches $\mathcal{B}_j^a$ in the animation skeleton tree. To find a solution to our mapping problem, we form a complete $I \times J$ branch matrix of mapping costs Q

$$\mathtt{B} = \begin{pmatrix} Q(\mathcal{B}_0^h, \mathcal{B}_0^a) & Q(\mathcal{B}_0^h, \mathcal{B}_1^a) & \cdots & Q(\mathcal{B}_0^h, \mathcal{B}_J^a) \\ Q(\mathcal{B}_1^h, \mathcal{B}_0^a) & Q(\mathcal{B}_1^h, \mathcal{B}_1^a) & \cdots & Q(\mathcal{B}_1^h, \mathcal{B}_J^a) \\ \vdots & \vdots & \ddots & \vdots \\ Q(\mathcal{B}_I^h, \mathcal{B}_0^a) & Q(\mathcal{B}_I^h, \mathcal{B}_1^a) & \cdots & Q(\mathcal{B}_I^h, \mathcal{B}_J^a) \end{pmatrix} . \tag{6}$$

For all elements $b_{i,j}$ of the branch matrix $\mathtt{B}$ this mapping cost Q is calculated by solving an optimization problem.

The optimization procedure finds the mapping of nodes N_s^h of branch $\mathcal{B}_i^h$ to nodes N_t^a of branch $\mathcal{B}_j^a$ using a node mapping cost K such that the overall mapping cost Q of the whole branch is minimized

$$Q(\mathcal{B}_i^h, \mathcal{B}_j^a) = \underset{t(s)}{\operatorname{argmin}} \sum_{s=1}^{S} K(N_s^h, N_{t(s)}^a) \tag{7}$$

subject to the above mentioned constraint that the mapping of nodes $s \rightarrow t$ must be in consecutive order. The node mapping cost K is given by:

$$\begin{aligned} K(N_s^h, N_t^a) \;=\; \Big(&\alpha_1 K_{\text{Pos}}(N_s^h, N_t^a) + \alpha_2 K_{\text{Perc}}(N_s^h, N_t^a) \\ &+ \alpha_3 K_{\text{Deg}}(N_s^h, N_t^a) \\ &- \alpha_4 K_{\text{Bet}}(n_k^\gamma) - \alpha_5 K_{\text{Area}}(n_k^\gamma)\Big) \quad . \end{aligned} \tag{8}$$

Here, the term K_{Pos} is the positional difference (Euclidean distance) between the two nodes in world space, K_{Perc} is the difference of the nodes' position along their individual branches given in percentage of the complete branch length, $K_{\text{Deg}}(N_s^h, N_t^a)$ is the difference of the nodes' degree centrality, and K_{Bet} and K_{Area} are the normalized betweenness-centrality and the normalized area of the node in the high-resolution tree. The weighting factors for each of the terms are experimentally chosen to be $\alpha_1 = 10.0$, $\alpha_2 = 2.0$, $\alpha_3 = 0.2$, $\alpha_4 = 1.0$, and $\alpha_5 = 0.5$.

Solving this optimization problem by brute force search has a combinatoric $O(n^2)$ complexity. However, the problem is similar to the sequence matching problem along the scanline of stereo cameras [VMVPVG02]. Using the same algorithm, the optimal solution for the optimization problem of Eq. 7 can be efficiently found by dynamic programming (as illustrated in Figure 5).

Once all elements $b_{i,j}$ of the branch matrix B are generated, we can find the best mapping for a branch $\mathcal{B}_i^h$ by choosing the corresponding $\mathcal{B}_j^a$ that produces the minimal cost $Q(\mathcal{B}_i^h, \mathcal{B}_j^a)$ for each row of matrix B. As a result we have found one branch mapping for each of the branches in the high-resolution tree to one branch in the animation skeleton tree, as well as the corresponding node mappings.

Reducing the number of mappings. We reduce this set of branch mappings by analysing which branches of the high-resolution tree are overlapping with a branch from the low-resolution tree and keep only the best mapping with the lowest mapping cost Q. As an example, in Fig. 4 the high-resolution tree has $I = 17$ branches and the low-resolution tree has $L = 6$ branches, so we keep only 6 of the 17 available branch mappings. In order to find out which branches overlap, we look at the nodes of the original contact graph from which both the high- and the low-resolution tree originated. As we have maintained a history of collapsed nodes, we know which nodes from the contact graph are clustered together to form a resulting node of the high- or low-resolution tree. Each branch of the high-resolution tree is assigned to the branch of the low resolution tree with which it shares the most contact graph nodes. As a result, we obtain a partitioning of the branches of the high-resolution tree into L sets. For each of these sets we keep only the assignment with the lowest branch matching cost within this set.

Handling conflicting mappings. Separate branches can overlap and share nodes at the overlapping branch segments. As we treat each branch separately, it may happen that the same node from the high-resolution skeleton is mapped to multiple nodes of the animation skeleton. In this situation we apply a winner-takes-all approach that removes all conflicting mappings from the node and only keeps the mapping for the branch for which the resulting cost Q (according to Eq. 7) is lowest.

Updated rigid skinning. It may happen that a node N_i^h does not get a mapping from an animation skeleton node N_i^a. In this case, we have to update its rigid skinning weight. All vertices that were attributed to node N_i^h are now assigned to the node's parent in the branch. This process is repeated until a parent node is reached that has a mapping from the animation skeleton.

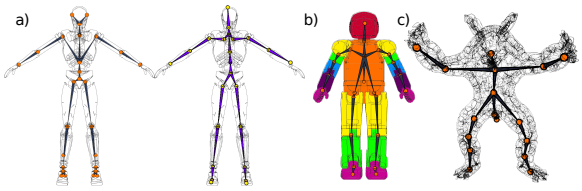

Figure 6: Comparison: a) our automatically created rig (left) is very similar to a rig made by an artist (right); b) Asimo with modeled joint geometry: the joints found by our method are correctly placed near the modeled joints; c) our approach also succeeds on a highly-fractured multi-component object.

4. Results

In this section, we show some of the results for skeleton creation, rigid skinning/clustering, and joint mapping that were generated with our approach. Additional results can be found in the supplemental video where we also show animated characters. To generate these animations, Autodesk's HumanIK was used for the motion retargeting step. All of our input models are taken directly from internet repositories, without modification.

As shown in Figs. 1 and 7, the proposed approach is capable of handling quite different morphologies (e.g., biped, quadruped, and uniped). Also the number of components of the shown models varies significantly (#components angel bot=165, t-rex=183, toilet=9, big robot=220, horse=152, armadillo=567). We tested different input animation skeletons (biped, quadruped), that sometimes differ starkly from the multi-component model in scale and morphology. In all cases, very plausible rigs are generated, and retargeting of a wide range of input motions (ranging from walking to complex martial arts moves) to these rigs consistently leads to believable animations. For all shown models the system requires less than half-a-minute to perform all steps of the pipeline (C++ code on a single CPU). On average, Contact Graph calculation takes 1.9 seconds, while Graph Clustering requires 2.0 second, and Joint Mapping takes 0.784 seconds. Fig. 6 shows that the skeleton created with our approach is comparable to a skeleton made by an artist. Our method also succeeds on meshes with explicitly modeled joint geometry. For example, the generated skeleton for the Asimo model, shown in 6b, is comparable to the result generated by an approach that requires explicit joint geometry for its analysis [XWY*09]. Also, multi-component objects, as used in [ZXTD10], fit well into our pipeline (as shown in 6c), which further illustrates the versatility of our method.

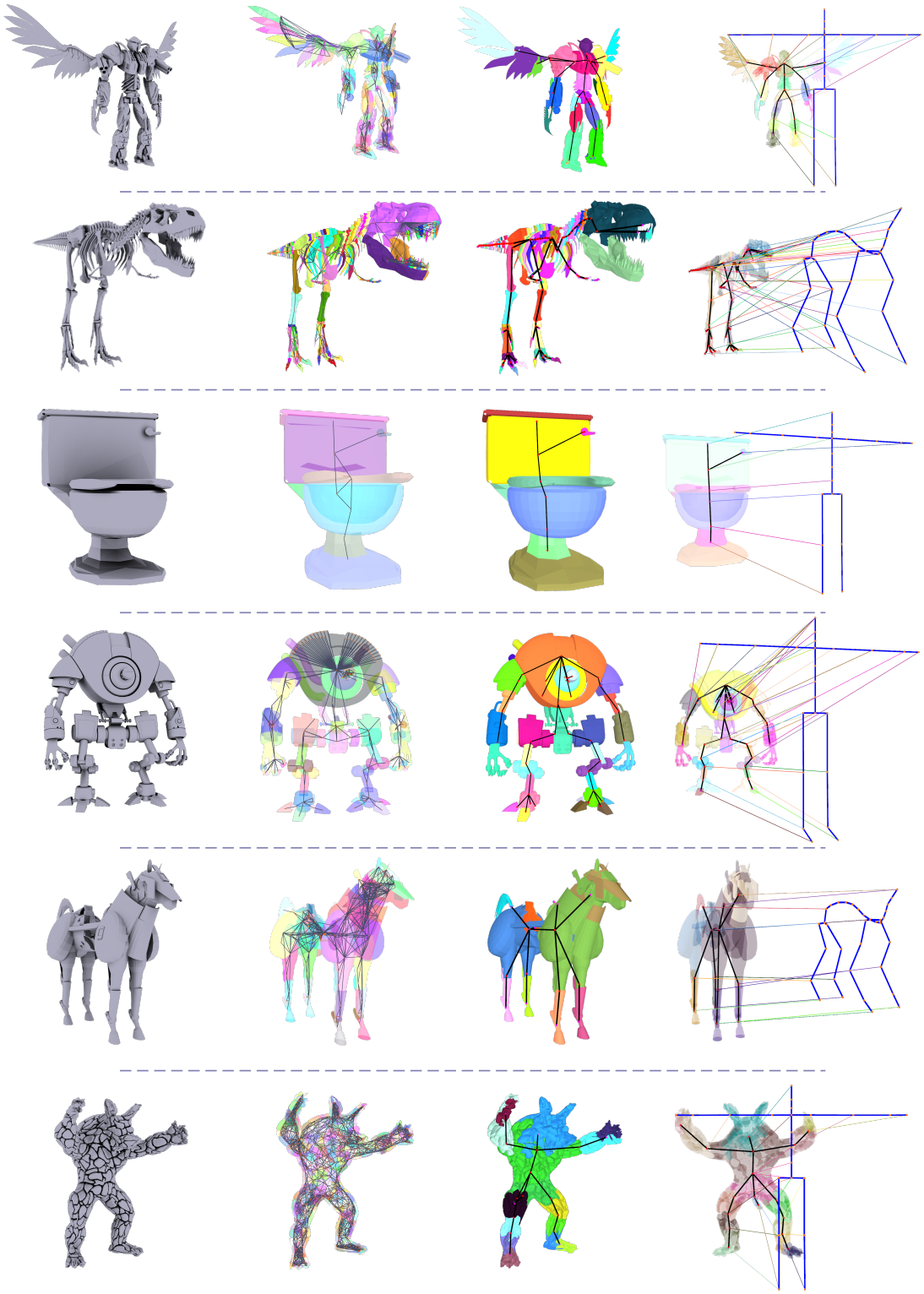

Figure 7: Results for five models with different morphology: (left to right) input multi-component model; segmentation and contact graph; generated high-resolution skeleton; joint mapping of the generated skeleton to the input animation skeleton.

5. Discussion and Future Work

Since the outputs of our pipeline uses the (industry standard) linear blend skinning format, the calculated target skeleton and rigid skinning weights can be easily used by a professional artist; however, an automated system like ours, also enables novice users to animate 3D models because they only have to choose an input mesh and the desired motion from an Internet repository. Most models, such as the results shown here and in the video, do not require any manual intervention. Nevertheless, the results of our automatic method are sometimes subject to estimation errors. Especially, as we do not use any a priori or user-given semantic information, the resulting joint mapping and clustering may result in unrealistic functioning of the model under animation. Modifying automatic joint mappings that enable a more plausible mechanical operation could easily be done in such cases using standard 3D animation software. Also, it would be relatively straightforward to modify the approach such that it proposes a range of possible mappings from which the user selects the preferred one.

Another limitation of our approach is that it requires an approximate initial alignment of the multi-component mesh and the animation skeleton, i.e., they should be roughly in the same pose (at least in one frame). This is necessary because a positional term is used in Eq. 9. This does not mean that both inputs have to be in perfect T-pose; but both have to be upright and facing in the same direction. Furthermore, the input model and the animation skeleton are scaled to a default height before they are processed by our pipeline.

In future work, we would also like to extend our approach to also take the particular motion of the animation skeleton into account to estimate the best joint mapping for a given motion. Finally, we plan to improve the method by using explicit symmetry criteria. Most of the generated rigs are in fact symmetric in practise, but this is currently not enforced within the algorithm.

6. Conclusion

We have presented a system for automatic rigging and joint mapping calculation for multi-component models that have no a priori component labels. By identifying coherent components, and analysing contacts between them, an initial graph is generated that can be simplified to a coarse tree by graph clustering. Having generated a skeleton representation for the model, a joint-mapping to an animation skeleton is performed. The approach is fast and successfully handles models with hundreds of components, as well as with morphology differences between input animation skeleton and extracted model skeleton.

References

[ATC*08] AU O. K.-C., TAI C.-L., CHU H.-K., COHEN-OR D., LEE T.-Y.: Skeleton extraction by mesh contraction. *ACM Trans. Graph. 27* (2008). 2, 5

[ATCO*10] AU O. K.-C., TAI C.-L., COHEN-OR D., ZHENG Y., FU H.: Electors voting for fast automatic shape correspondence. *Computer Graphics Forum 29* (2010). 6

[BP07] BARAN I., POPOVIĆ J.: Automatic rigging and animation of 3d characters. *ACM TOG 26* (2007). 2

[CK99] CHOI K.-J., KO H.-S.: On-line motion retargetting. *Journal of Visualization and Computer Animation 11* (1999). 3, 6

[Fre77] FREEMAN L. C.: A set of measures of centrality based on betweenness. *Sociometry 40* (1977). 6

[GG04] GELFAND N., GUIBAS L. J.: Shape segmentation using local slippage analysis. In *Proc. SGP* (2004). 2

[GH97] GARLAND M., HECKBERT P. S.: Surface simplification using quadric error metrics. In *Proc. SIGGRAPH* (1997). 5

[Gle01] GLEICHER M.: Comparing constraint-based motion editing methods. *Graph. Models 63* (2001). 3, 6

[GR96] GOLD S., RANGARAJAN A.: A graduated assignment algorithm for graph matching. *IEEE TPAMI 18* (1996). 5, 6

[Hah88] HAHN J. K.: Realistic animation of rigid bodies. *SIGGRAPH Comput. Graph. 22* (1988). 2

[HRE*08] HECKER C., RAABE B., ENSLOW R. W., DEWEESE J., MAYNARD J., VAN PROOIJEN K.: Real-time motion retargeting to highly varied user-created morphologies. *ACM TOG 27* (2008). 2, 6

[KA09] KELLER Y., AGOZI A.: A probabilistic approach to spectral graph matching. *Most* (2009), 1–7. 5, 6

[KSO10] KAVAN L., SLOAN P.-P., O'SULLIVAN C.: Fast and efficient skinning of animated meshes. *Comput. Graph. Forum 29* (2010). 2

[KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. *ACM TOG 22* (2003). 2

[LH05] LEORDEANU M., HEBERT M.: A spectral technique for correspondence problems using pairwise constraints. In *Proceedings ICCV* (2005). 5, 6

[MTLT88] MAGNENAT-THALMANN N., LAPERRIÃLRE R., THALMANN D.: Joint-dependent local deformations for hand animation and object grasping. In *Graphics Interface* (1988). 6

[MYY*10] MITRA N. J., YANG Y.-L., YAN D.-M., LI W., AGRAWALA M.: Illustrating how mechanical assemblies work. *ACM TOG 29* (2010). 2

[Sim94] SIMS K.: Evolving virtual creatures. In *Proc. SIGGRAPH* (1994). 2

[SP04] SUMNER R. W., POPOVIĆ J.: Deformation transfer for triangle meshes. *ACM Trans. Graph. 23* (2004). 2, 3

[VMVPVG02] VAN MEERBERGEN G., VERGAUWEN M., POLLEFEYS M., VAN GOOL L.: A hierarchical symmetric stereo algorithm using dynamic programming. *IJCV 47* (2002). 8

[WLH97] WONG T.-T., LUK W.-S., HENG P.-A.: Sampling with hammersley and halton points. *J. Graph. Tools 2* (1997). 4

[XWY*09] XU W., WANG J., YIN K., ZHOU K., VAN DE PANNE M., CHEN F., GUO B.: Joint-aware manipulation of deformable models. *ACM TOG 28* (2009). 2, 8

[YAH10] YAMANE K., ARIKI Y., HODGINS J.: Animating non-humanoid characters with human motion data. In *Proc. SCA* (2010). 3

[ZXTD10] ZHOU K., XU W., TONG Y., DESBRUN M.: Deformation transfer to multi-component objects. *Comput. Graph. Forum 29* (2010). 2, 3, 8