Skeleton-based Matching for Animation Transfer and Joint Detection

Martin Madaras* Michal Piovarči Jana Běhal Dadová Roman Franta Tomáš Kovačovský

Comenius University Bratislava, Slovakia

Abstract

In this paper we present a new algorithm for establishing correspondence between objects based on matching of extracted skeletons. First, a point cloud of an input model is scanned. Second, a skeleton is extracted from the scanned point cloud. In the last step, all the extracted skeletons are matched based on valence of vertices and segment lengths. The matching process yields into two direct applications - topological mapping and segment mapping. Topological mapping can be used for detection of joint positions from multiple scans of articulated figures in different poses. Segment mapping can be used for animation transfer and for transferring of arbitrary surface per-vertex properties. Our approach is unique, because it is based on matching of extracted skeletons only and does not require vertex correspondence.

CR Categories: Computer Graphics [I.3.8]: Applications

Keywords: skeleton extraction, skeleton matching, joint detection, animation transfer

1 Introduction

A skeleton is a structure encoding topology of an object. It is useful in many applications including animation, shape recognition or retrieval, surface parameterization, procedural modelling etc. There are many algorithms for extracting skeletons from input models. These methods differ in robustness and type of input they require. We propose an extension for extracted skeletons based on matching between two skeletons. Skeleton matching has a variety of applications in computer graphics. Having an one-to-one mapping between sets of skeleton segments, we are able to transfer model information as animation matrices, animation rigs or other surface properties. We focus on two types of skeletons mappings. First one we call topological mapping, which maps topological branches of one skeletons onto topological branches of another skeleton. This can be used to merge two skeletons of one model in different poses into one union-skeleton. The union-skeleton can be further used for detection of joints of an articulated figures. Second one we call segment mapping, which maps each bone of one skeleton onto a sequence of skeleton bones in the second skeleton. Applications of segment mapping include animation transfer, transfer of surface properties or surface cross-parameterization.

Related work to skeleton extraction, animation transfer and topological mapping is presented in Section 2. In section 3 we focus on scanning of input models, specially point clouds of articulated figures in different poses. A modified version of Au's algorithm [Au et al. 2008] for extraction of skeletons from point clouds is described in Section 4. In Section 5 we propose our skeleton-based matching with definitions of expressions used in the text. Next, both topological and segment mapping are described and results of detection of joints and animation transfer are presented in Section 6. A Conclusion and ideas for future work are mentioned in Section 7. All the pseudocodes of matching algorithm can be found in Appendix.

*e-mail: madaras@sccg.sk

SCCG 2014, Smolenice, Slovakia, May 28 – 30, 2014.
© 2014 ACM 978-1-4503-3070-1/14/0005 $15.00

2 Related Work

Skeleton Extraction Numbers of algorithms have been proposed to compute a skeleton from an input mesh geometry. In [Shapira et al. 2008] authors proposed skeleton extraction based on a shape diameter function (SDF). The SDF is a scalar function defined on the mesh surface that expresses a measure of the diameter of the object's volume in the neighborhood of each point on the surface. Thus, a set of random vertices is pushed in an inward normal direction into the volume of the model by a distance that equals to half of the SDF values and a least-squares method is used to fit a high-degree curve into the shifted points. Similar approach was used in [Liu et al. 2003], where authors used so called repulsive force field. Sharf et. al [Sharf et al. 2007] introduced a method that is able to perform skeleton extraction on both, point clouds and polygonal meshes. The method uses evolution of a deformable model inside of the mesh. The initial extracted graph is noisy, and extraction of final skeleton require further filtering and merging.

Reeb graph based methods need a suitable real-value function defined on the model surface for a successful extraction of a skeleton. Using this function, nodes of a 1D graph can be computed. In [Hilaga et al. 2001] a geodesic function was used for Reeb graph extraction. Alternatively, a method based on a harmonic function proposed by Aujay et. al [Aujay et al. 2007] captures after resampling all the features of the model well, but requires the user to specify the boundary condition explicitly.

Au et. al [Au et al. 2008] introduced a Laplacian smoothing based method that works directly on the mesh geometry. The main idea of this approach is to apply a well defined filter on mesh vertices. In the first step, an input mesh is contracted using iterative Laplacian contraction. Then, a mesh decimation is used to simplify the contracted mesh into a curve-skeleton. Cao et. al [Cao et al. 2010] extended the idea of Laplacian contraction [Au et al. 2008] for a point cloud input. They used a definition of the Laplacian operator for a point cloud in order to perform a similar weighted filtering. When the mesh is contracted, mesh decimation cannot be used, because an edge connectivity is not defined. Authors made selection of contracted points to be connected based on their Euclidean distance. Therefore, the ability to extract a proper skeleton depends on the distance between samples on the manifold being smaller than the distance between the two structures.

In [Teichmann and Teller 1998] authors extract skeleton by simplifying the Voronoi skeleton with a small amount of user assistance. Another Voronoi diagram based method [Dey and Sun 2006] computes the skeleton from the medial axis obtained by extracting the internal edges of the Voronoi diagram. These methods are quite slow in comparison to Reeb graph or Laplacian smoothing based

ones and do not guarantee that the result will capture all desired features.

Animation Transfer Gleicher et al. [Gleicher 1998] introduced possibility of motion retargeting between articulated figures with same topology. This approach required many user defined constraints. Recent methods are more automated but use special retargeting to create interacting figures with each other [Ho et al. 2010] or with the environment [Liu et al. 2010]. All of these methods focus on mapping between figures with the same topology only.

For purpose of motion retargeting for different morphologies, a system for animating characters whose morphologies are unknown at the time the animation is created was developed [Hecker et al. 2008]. However, the system needs an information about the semantics of the characters. Thus, deformable anatomical body parts have to be chosen from a palette and special queries have to be used during the animation transfer. Chang et al. uses consistent volume parameterization [Chang et al. 2006] to transfer the animation. However, a feature points have to be set to obtain vertex cross-correspondence.

Topological Mapping and Cross-parameterization Shape cross-parameterization is a fundamental operation in many mesh processing algorithms such as deformation transfer, shape blending and surface detail transfer [Sumner and Popović 2004; Praun et al. 2001]. These methods for cross-parameterization are based on vertex-to-vertex mapping and require solving of minimization system for all the vertices.

Schreiner et al. [Schreiner et al. 2004] created continuous map between two arbitrary meshes. Furthermore, map distortion is minimized during the refinement and as a result a map that naturally align corresponding shape elements is obtained. The method require an initial feature correspondence to be set by a user.

Fan et al. [Fan et al. 2005] introduced a morphing technique based on polycube [Tarini et al. 2004] cross-parameterization. Polycube-based parameterization is robust and works for meshes with arbitrary genus. Furthermore, they are able to build maps with singularities and perform morphing between models with different genus, however, an user-driven face matching is needed.

Some existing graph-based approach do exist for evaluation of similarity of two graphs. A flooding algorithm [Melnik et al. 2002] recursively computes similarity of nodes in two graphs. In [Zager and Verghese 2008] authors derive pairwise similarity scores for the nodes of two different graphs. However, these methods use recursively computation of similarity and the computation of similarity is based on pairwise comparison.

We are interested in topological similarity only. In our case, number of nodes of two extracted skeletons may differ. Thus, we propose two-step matching between graphs based on topological and segment mappings. Our method is based on matching of extracted skeletons only, therefore a calculation of vertex correspondence is not needed. Moreover, our approach is able to obtain mapping for models with different topology as well. In this case, only the homotopic parts of the graphs are mapped and skeleton segments that cannot be mapped are ignored.

3 Model Scanning

The complete, high resolution point-clouds of articulated figures are obtained with our structured light optical 3D scanner SMISS [Kovačovský 2012]. The system uses projector-camera setup to reconstruct the surface geometry of target figure using triangulation principle. We use Gray and sinusoidal coded patterns to solve correspondence problem with sub-pixel accuracy. The scanning system also incorporates the method of High Dynamic Range scanning using projector weight maps. The intensity of projector lighting is weighted per pixel, so that surfaces with higher reflectance receive less illumination. This allows us to capture Higher Dynamic Range scenes with just constant increase of scanning time for weight map capture and computation. The scanner head is mounted on a motorized arm, which allows for vertical positioning of the sensor. The articulated figure is placed on a motorized glass table for automatic object rotation. These two axes allow us to position the scanning head on arbitrary point of a sphere with a predefined radius with respect to a center of an object. The complete 3D cloud is merged from partial scans from different viewpoints. The pose of individual scans is obtained from capturing our newly designed pose-estimation marker field of multiple circles with binary coded positions (see Figure 1). The object is laid on our marker field and in every scan, multiple circles of the field are localized and pose estimation is computed from image to world space correspondence and the known calibration parameters (world coordinates are encoded as two 16 bit floats), so that the partial point clouds share the marker fields coordinate system. After all necessary partial scans are captured, the ICP algorithm is used for more precise alignment of individual point clouds.

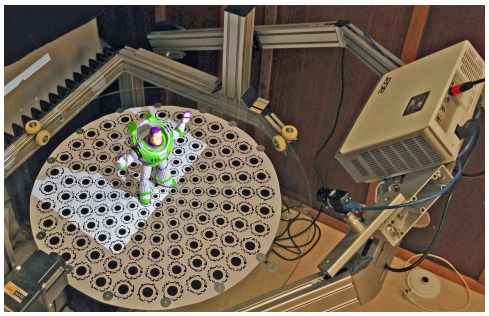

Figure 1: Our setup for model scanning (SMISS).

4 Skeleton Extraction

For skeleton extraction we use a modified version of Au's algorithm [Au et al. 2008] extended to point clouds in which we use a construction of local Delaunay's triangulations for Laplacian estimation. Similar solution was presented in [Cao et al. 2010], but the authors computed Euclidian distance between samples for construction of skeleton from the contracted point cloud. Such a distance grouping yields into problems when the structures in different skeleton branches are closer than point samples in the neighborhood of the vertex. Therefore, to construct the final skeleton, a simplification of a triangulation instead of Euclidian distance is used. This triangulation is constructed in natural neighbors [Boissonnat and Cazals 2002] manner by composition of local one-ring Delaunay's triangulations into the global triangulation.

A process from scanning of articulated figure in different poses to extracted skeletons of those poses can be seen in Figure 12 enclosed in Appendix.

4.1 Extension to Point Clouds

Given a point cloud set P consisting of n points $P = \{\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n\}$, a local neighborhood L_i of each vertex $\mathbf{p}_i \in P$ is

computed. For each point $\mathbf{p}_i$, the closest k-neighborhood L_i is calculated using a kD tree. Similarly as in [Cao et al. 2010], for each local neighborhood L_i a tangent plane using PCA [Jolliffe 2002] is computed. Then, all the points in local neighborhood are projected into the tangent plane and a local Delaunay triangulation D_i is computed. Local Delaunay triangulation is a set of edges $D_i = \{e_i^1, e_i^2, \ldots, e_i^{\mathrm{noe}(i)}\}$, where $\mathrm{noe}(i)$ is number of edges of ith local Delaunay triangulation. Finally a global triangulation D is created as a composition of all the local triangulations as

$$D = \bigcup_{i=1}^{n} \{e_i^1, e_i^2, \ldots, e_i^{\mathrm{noe}(i)}\}. \quad (1)$$

Note that the global triangulation D does not have to be 2D-manifold. In this implementation The parameter k was experimentally set $k = n \times 0.02$ and restricted to range $[8..12]$.

For computation of Laplacian flow on the surface using cotangent schema , a local triangulation D_i is used for each point $\mathbf{p}_i$. Finally, when the point cloud is contracted, a skeleton is constructed by simplification of global triangulation D. For this simplification a modified version of QEM simplification is used [Au et al. 2008]. An example of such a global triangulation can be seen in Figure 2.

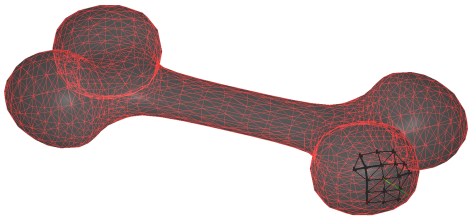

Figure 2: An example of global triangulation D composed from local Delaunay triangulations. As an example, one local Delaunay triangulation is highlighted with bold black line.

5 Skeleton-based Matching

In this section a heuristics for matching of skeletons is described. However, first definitions used in text are introduced.

5.1 Our definitions

Skeleton of an object is a rooted graph encoding the topology of the object.

Skeleton cycle is a cycle in the skeleton graph. Each cycle is representing a topological handle in a model the skeleton was extracted from. Skeleton can be converted into a tree by breaking all the cycles. If an edge in the cycle is broken, one node is duplicated in the same position and a special pointer referencing this duplicated node is introduced.

Skeleton node is a node of the skeleton. The node contains an information about its 3D coordinates. If the skeleton is represented as a tree, the node contains also information about his father, sons and a pointer to the duplicated vertex, if the skeleton graph has contained cycles before conversion to tree structure.

Skeleton segment is an edge of skeleton connecting two skeleton nodes. Skeleton segment is also referenced as a bone in some literature.

Branch node is a skeleton node with valence higher than two. If represented as a tree, branch node is a node that is not a root and has more than one son, or skeleton root with more than two sons.

Skeleton path is an ordered sequence of nodes such that from each of its nodes there is an edge to the next node in the sequence.

Topological branch is a skeleton path where the start node and the end node are either two branch nodes, or a branch node and a node with valence one (a leaf in tree representation).

Skeleton matching is a fitting process of two skeletons based on valence of branch nodes and lengths of topological branches. The input is pair of skeletons and the output of the process is sequence of best evaluated mappings ordered from best to worst. The matching is evaluated as a weighted sum of difference of valences of matched branch nodes and a difference of lengths between two paths.

Skeleton topological mapping is a mapping between topological branches of one skeleton and topological branches of second skeleton. A topological branch can be mapped to another topological branch or to nothing.

Skeleton segment mapping is a mapping between segments of one skeleton and segments of another skeleton. A segment can be mapped to a segment, topological path or to nothing.

5.2 Matching Algorithm

We are looking for mapping from topological branch of one skeleton to topological branch of another skeleton. All the pseudocodes concerning skeleton-based matching and skeleton mapping can be found in Appendix. In the preprocessing step, for all topological branches we collapse whole skeleton path into a single segment (see Figure 3). By removing the connection nodes, we lose the original information. To rectify this, we extend the edges of the input skeleton such that they contain data that help during the matching phase. Currently, we store the length of the removed path and the number of nodes that were on the path. In next phase, we are looking for matching between two skeletons using graph representation. Therefore, we convert skeletons to graphs with undirected edges. We use simple backtracking to find all the possible solutions (Algorithm 2). We impose two constraints to reduce the searching area (Algorithm 3). The first constraint is that each newly matched node from the first skeleton must have the same neighbors as corresponding node in the second skeleton. The second constraint is that a node can only match onto a node with the same or higher number of neighbors. Two-pass filtering is applied on the rated solutions. In the first pass, the matching is penalized for unequal number of vertices on the skeleton path (Algorithm 4). Solutions are then sorted from best to worst. We pick only the solutions, which has rating within a threshold of tolerance from the best solution. In the second pass, we sort these solutions again with different rating which measures Euclidean distance between matched nodes in skeletons (Step 6 in Algorithm 1). A pseudocode of matching process is shown in Algorithm 1.

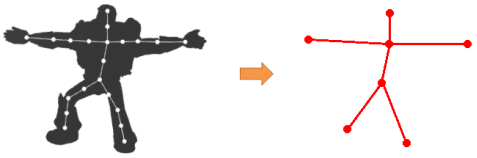

Figure 3: Collapsing of topological branches into a single segment.

5.3 Topological Mapping

We are going to create an union-skeleton from the input set of skeletons (see Figure 4). We pick a skeleton with the lowest number of vertices from the input set and match it with each other input skeleton using Algorithm 1. This way an intersection skeleton contained in each of the input skeletons is found. We reorder the child nodes of each node so that the matched child nodes are first and then the unmatched ones. In order to create the union-skeleton we add each matched skeleton to intersection skeleton. Next, we add the union-skeleton to each input skeleton. In both cases the Algorithm 5 is used and skeletons are treated as trees. We call Algorithm 5 on skeleton roots with zero distance. After that, all skeletons have the same topology and the degrees of freedom (DoFs) can be extracted.

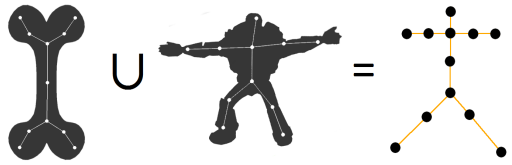

Figure 4: An ilustration of topological mapping applied on two skeletons with different topology resulting into a union-skeleton.

5.4 Segment Mapping

We match two input skeletons using Algorithm 1. When we have the matched skeletons, we loop through all matched pairs of edges of both skeletons. First, we reintroduce the previously collapsed nodes into each edge. The nodes are ordered naturally along the edge, matched parent node is the first node, matched child node is the last node and reintroduced nodes have ascending ordered by their distance to parent node. Next, we normalize the length of each edge so that the distance between the first and the last node of each edge equals to 1. This can be seen in Figure 5, blue nodes correspond to skeleton which we are matching and green nodes correspond to skeleton which is being matched. Smaller newly inserted nodes are distributed along the normalized edge. Each newly inserted node position defines a threshold up to which all nodes from matched skeleton correspond to it. In Figure 5 thresholds generated by inserted nodes are marked with cyan lines along the edge. These thresholds split the matched edge into segments. Each segment and its nodes are then mapped onto the node that generated the segment (see Algorithm 6).

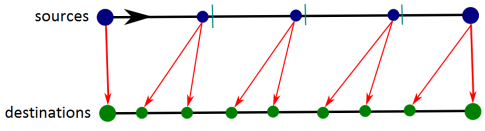

Figure 5: Mapping of upper path node to nodes in lower path.

6 Results

Detection of joints and composition of all the skeletons (see Figure 6) into union-skeleton is based on topological mapping. Transfer of animation including transfer of per-node transformation matrices and per-vertex skinning data is based on segment mapping.

6.1 Detection of Joints

We merge all the matched skeleton segments into one union-skeleton. For every union-skeleton node, we measure the change in rotation between original skeleton pose and the matched union-skeleton. The nodes, where rotation changes are higher than the predefined threshold are detected as joints (see Figure 7). Finally, a skeleton symmetry is used to detect joints in symmetrical body parts of the model.

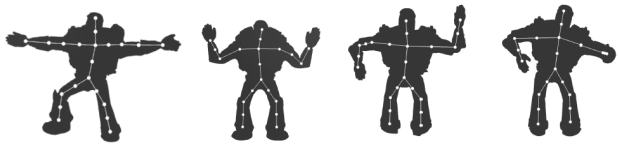

Figure 6: Extracted skeletons of input articulated figure in different poses.

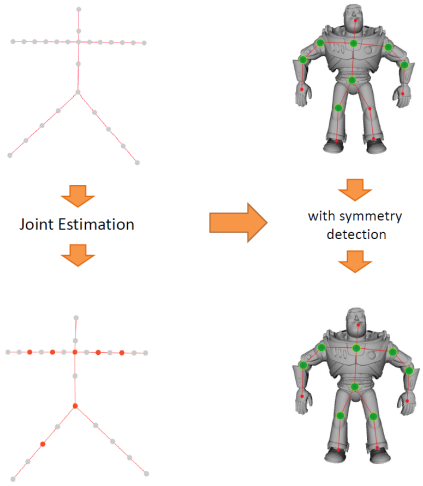

Figure 7: A union-skeleton is composed from all the skeletons. Next, the nodes where the rotation changes are higher than the predefined threshold are detected as joints (red nodes).

6.2 Animation Transfer

For motion retargeting, we have to fulfil one condition. The roots of skeletons have to be in the same position relatively to model, e.g. center of mass of the model or chest of the character. We want to ensure that the target skeleton is more sampled than the source skeleton. This means that for each node of source skeleton, there has to be a node to map in the target skeleton. If this condition is not satisfied, we iteratively subdivide the destination skeleton until the condition is satisfied (see Figure 8).

Given mapping of bones from the source skeleton to the target skeleton we transfer animation using quaternions for the rotation transformation. For the transfer, source skeleton needs to have preset animation using key frames and rotation quaternions. During process of the transfer, firstly we identify these key frame poses and respective bone rotations in source skeleton. Afterwards target skeleton bones are set to rest rotations as they are without any animation in source skeleton. This we accomplish with target skeleton breadth-first graph search. In each joint, we rotate target bone by an

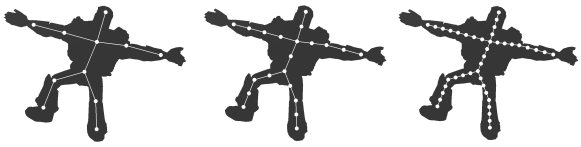

Figure 8: Subdivision of skeleton. From left to right: an original extracted skeleton, skeleton after one subdivision step, skeleton after two subdivision steps.

angle that is between the bone and mapped bone in source skeleton. Rotation axis is cross product of these vectors to avoid rotations around bone axis.

Secondly, when the target skeleton is aligned with the rest pose of the source skeleton, we calculate rotations of the target skeleton for the current pose in selected keyframe. Again we use target skeleton breadth-first graph search and apply rotations to the bones from the bones of the source skeleton current pose. We set these pose rotations for each keyframe that was set for the source skeleton animation (see Figure 9).

Figure 9: Example of motion retargeting for three different poses.

Afterwards, we resolve self-penetrations of the target object. Firstly, we identify vertex groups for the bones and calculate possible penetrations in transferred animation. We define penetration as an intersection of an edge from one vertex group with a face from another vertex group. Mostly we take into account only penetration between vertex groups that are not in neighborhood of each other. When a penetration occurs, we adjust bone rotations of intersecting vertex groups for a given pose. If the result of this automated method is not suitable for the final rendering, it is always possible to improve it using user input. With all rotations for each key frame, animation is successfully transferred and we use B-spline curves with same setting as are in source animation to render final result (see Figure 13 in Appendix).

6.3 Transfer of Information Stored in Vertices

During the animation transfer process, skinning weights have to be transferred. If the animation skeletons of both models are *reliable* [Cornea et al. 2007] a Skeleton Texture Mapping (STM) [Madaras and Ďurikovič 2013] technique can be used. Reliability refers to the property of the skeleton that every boundary point (point on objects surface) is visible from at least one curve-skeleton location. In both skeletons, each skeleton segment is used for skeleton-based parameterization of surrounding model surface (see Figure 10). Thus, using STM we get texture coordinates for model vertices and skinning weights can be written into texture. In our tests, we used four skeleton segments to control one vertex. Therefore, skinning weights could be stored in one texture in RGBA channels for whole model. In case when more bones are influencing vertices, higher number of textures have to be used. In the first step, skinning weights from the input model are stored in STM texture. STM texture is composed of rectangular sub-textures for each skeleton segment. Now, having the segment mapping, we know where each sub-texture is mapped in the STM of the second output model. Each sub-texture is linearly stretched to fit destination dimensions and texels are copied. In a case, when one segment is mapped onto more segments, the sub-texture is split according to the relative length of these segments. Vice-verse, when more segments are mapped onto one segment, sub-textures are composed to produce one connected sub-texture.

There are two ways how to transfer the weights. The first option is to use one STM for one bone. In this case, the STM would be a one-channel texture describing influence of this bone over the mesh surface. This approach is good for visualization. The second option is to use all the four channels of texture. We assume that each node is controlled with maximum of four bones, which is number usually used in all real time deformation engines. Now, we are going to have two sets of STM (see Figure 11). STM_i is going to store all the indices of bones controlling the vertices. Here, an conversion from integer to float has to be performed. STM_w is going to store all the weights controlling the vertices. If a higher number of control bones than four is desired, a higher number of STM has to be used as well.

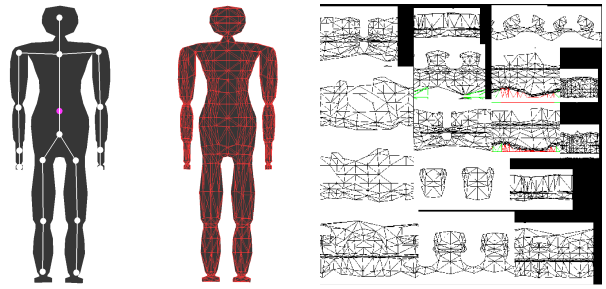

Figure 10: (Left) A reference figure with predefined skeleton and per vertex skinning data as weights and indices. (Right) Triangulation of reference figure rendered into texture using STM.

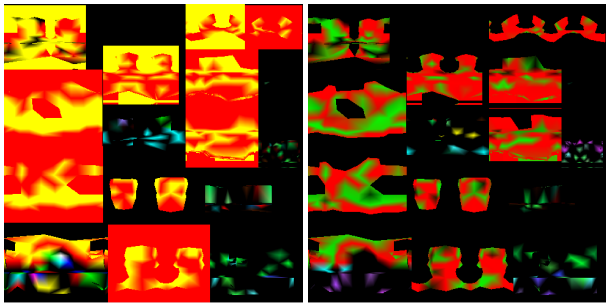

Figure 11: Extracted STM with skinning indices (left) and skinning weights (right). Some parts of the sub-textures are black because weight/index stored in alpha channel is high.

7 Conclusion and Future Work

We have developed the method for skeleton matching. Skeletons are extracted from scanned point clouds. Our skeleton matching is used for topological mapping and segment mapping of skeletons. If the scanned model is an articulated figure, we are able to detect the joint positions. Moreover, our system enables to transfer surface and skeleton information from one model to another. This can

be used for animation transfer from predefined motion to scanned figure.

Our animation transfer system has some limitations we would like to face in the future work. The main limitation is that both models have to be approximately in the same pose. Mostly in animation transfer applications a bind pose is used. However, it is able to transfer the animation from different pose, but self-collision tests have to be performed or maximal angles in joints have to be checked. Another limitation is that for transferring of skinning per-vertex data, both animation skeletons have to be reliable so skeleton-based parameterization can be used. If at least one of the skeletons is not reliable, the skinning weights have to be set manually or using animation software package. Therefore, in the future work we would like to focus on converting an input animation skeleton into an animation skeleton which satisfies the reliability condition and produces the same animation.

References

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

AUJAY, G., HÉTROY, F., LAZARUS, F., AND DEPRAZ, C. 2007. Harmonic skeleton for realistic character animation. In *Proceedings of the 2007 ACM SIGGRAPH*, 151–160.

BOISSONNAT, J.-D., AND CAZALS, F. 2002. Smooth surface reconstruction via natural neighbour interpolation of distance functions. *Comput. Geom. Theory Appl. 22*, 1-3 (May), 185–203.

CAO, J., TAGLIASACCHI, A., OLSON, M., ZHANG, H., AND SU, Z. 2010. Point cloud skeletons via laplacian based contraction. In *Proceedings of the 2010 SMI Conf.*, 187–197.

CHANG, Y.-T., CHEN, B.-Y., LUO, W.-C., AND HUANG, J.-B. 2006. Skeleton-driven animation transfer based on consistent volume parameterization. In *Proceedings of the 24th International Conference on Advances in Computer Graphics*, Springer-Verlag, Berlin, Heidelberg, CGI'06, 78–89.

CORNEA, N. D., SILVER, D., AND MIN, P. 2007. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics 13*, 3, 530–548.

DEY, T. K., AND SUN, J. 2006. Defining and computing curve-skeletons with medial geodesic function. In *Proceedings of the 4th EG symposium on Geom. processing*, 143–152.

FAN, Z., JIN, X., FENG, J., AND SUN, H. 2005. Mesh morphing using polycube-based cross-parameterization: Animating geometrical models. *Comput. Animat. Virtual Worlds 16* (July), 499–508.

GLEICHER, M. 1998. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '98, 33–42.

HECKER, C., RAABE, B., ENSLOW, R. W., DEWEESE, J., MAYNARD, J., AND VAN PROOIJEN, K. 2008. Real-time motion retargeting to highly varied user-created morphologies. In *Proceedings of ACM SIGGRAPH '08*. http://chrishecker.com/Real-time$_{M}$otion$_{R}$etargeting$_{t}$o$_{H}$ighly$_{V}$aried$_{U}$ser – Created$_{M}$orphologies.

HILAGA, M., SHINAGAWA, Y., KOHMURA, T., AND KUNII, T. L. 2001. Topology matching for fully automatic similarity estimation of 3d shapes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 203–212.

HO, E. S. L., KOMURA, T., AND TAI, C.-L. 2010. Spatial relationship preserving character motion adaptation. *ACM Trans. Graph. 29*, 4 (July), 33:1–33:8.

JOLLIFFE, I. 2002. *Principal Component Analysis*. Springer Series in Statistics. Springer.

KOVAČOVSKÝ, T. 2012. Scalable multifunctional indoor scanning system. In *Bulletin of the ACM Slovakia*, 47–48.

LIU, P.-C., WU, F.-C., MA, W.-C., LIANG, R.-H., AND OUHYOUNG, M. 2003. Automatic animation skeleton construction using repulsive force field. In *Proceedings of the 11th Pacific Conference on CG and Applications*, 409–413.

LIU, L., YIN, K., VAN DE PANNE, M., SHAO, T., AND XU, W. 2010. Sampling-based contact-rich motion control. *ACM Trans. Graph. 29*, 4 (July), 128:1–128:10.

MADARAS, M., AND ĎURIKOVIČ, R. 2013. Skeleton texture mapping. In *Proceedings of the 28th Spring Conference on Computer Graphics*, ACM, New York, NY, USA, SCCG '12, 121–127.

MELNIK, S., GARCIA-MOLINA, H., AND RAHM, E. 2002. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *Proceedings of the 18th International Conference on Data Engineering*, IEEE Computer Society, Washington, DC, USA, ICDE '02.

PRAUN, E., SWELDENS, W., AND SCHRÖDER, P. 2001. Consistent mesh parameterizations. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, ACM, New York, NY, USA, SIGGRAPH '01, 179–184.

SCHREINER, J., ASIRVATHAM, A., PRAUN, E., AND HOPPE, H. 2004. Inter-surface mapping. *ACM Trans. Graph. 23* (Aug.), 870–877.

SHAPIRA, L., SHAMIR, A., AND COHEN-OR, D. 2008. Consistent mesh partitioning and skeletonisation using the shape diameter function. *Vis. Comput. 24* (March), 249–259.

SHARF, A., LEWINER, T., SHAMIR, A., AND KOBBELT, L. 2007. On-the-fly curve-skeleton computation for 3D shapes. *Computer Graphics Forum, (Proceedings Eurographics 2007) 26*, 3, 323–328.

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

TARINI, M., HORMANN, K., CIGNONI, P., AND MONTANI, C. 2004. Polycube-maps. In *In Proceedings of SIGGRAPH 2004*, 853–860.

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

ZAGER, L. A., AND VERGHESE, G. C. 2008. Graph similarity scoring and matching. *Applied Mathematics Letters 21*, 1, 86 – 94.

Appendix

Algorithm 1 MatchSkeletons

Require: $S1 \leftarrow InputSkeleton1$
Require: $S2 \leftarrow InputSkeleton2$
1: $S'1, formerNodes1 \leftarrow Collapse(S1)$
2: $S'2, formerNodes2 \leftarrow Collapse(S2)$
3: $A \leftarrow Skeleton2Graph(S'1, formerNodes1)$
4: $B \leftarrow Skeleton2Graph(S'2, formerNodes2)$
5: $matches \leftarrow MatchGraphs(A, B)$
6: $matches \leftarrow GeometrySort(matches)$
7: **return** $matches.first()$

Algorithm 2 MatchGraphs

Require: $A \leftarrow InputGraph1$
Require: $B \leftarrow InputGraph2$
Require: $match \leftarrow InputMatch$
  if $match.size() == A.nodes.size()$ **then**
    $Evaluate(A, B, match)$
  else
    for each $node$ **in** $B.nodes$ **do**
      if $CanMatch(B, A.nodes[match.size()], node, match)$ **then**
        $match \leftarrow match \cup node$
        $MatchGraphs(A, B, match)$
        $match \leftarrow match \setminus node$
      end if
    end for
  end if

Algorithm 3 CanMatch

Require: $B \leftarrow InputGraph2$
Require: $nodeA \leftarrow InputNodeFromA$
Require: $nodeB \leftarrow InputNodeFromB$
Require: $match \leftarrow InputMatch$
  if $AlreadyMatched(nodeB, match)$ **then**
    return false
  end if
  if $nodeA.neighbors.size() > nodeB.neighbors.size()$ **then**
    return false
  end if
  for each $node$ **in** $nodeA.neighbors$ **do**
    $matched \leftarrow match[node]$
    if $matched \neq -1$ **then**
      if not $AreNeighbors(B.nodes[matched], nodeB)$ **then**
        return false
      end if
    end if
  end for
  return true

Algorithm 4 Evaluate

Require: $A \leftarrow InputGraph1$
Require: $B \leftarrow InputGraph2$
Require: $match \leftarrow InputMatch$
  $error \leftarrow 0$
  for each $node$ **in** $A.nodes$ **do**
    if not $Matched(node, match)$ **then**
      $error \leftarrow error + node.formerNodes$
    else
      $nodeB \leftarrow B.nodes[match[node]]$
      $diff \leftarrow abs(nodeB.formerNodes - node.formerNodes)$
      $error \leftarrow error + diff$
    end if
  end for
  $InsertAscending(match, error, matches)$

Algorithm 5 AddSkeleton

Require: $dstNode \leftarrow InputNode1$
Require: $srcNode \leftarrow InputNode2$
Require: $srcDist \leftarrow InputDistOfNode1$
Require: $dstDist \leftarrow InputDistOfNode2$
  if $srcNode = NULL$ **then**
    return
  end if
  if $dstNode = NULL$ **then**
    $InsertToSkeleton(srcNode)$
  end if
  if $abs(srcDist - dstDist) < threshold$ **then**
    $dstNode.matched \leftarrow dstNode.matched + 1$
    for each $nodeA, nodeB$ **in** $dstNode.nodes, srcNode.nodes$ **do**
      $dA \leftarrow nodeA.parentDist$
      $dB \leftarrow nodeB.parentDist$
      $AddSkeleton(nodeA, dA, nodeB, dB)$
    end for
  end if
  if $dstDist < srcDist$ **then**
    $node \leftarrow dstNode.nodes[0]$
    $srcDist \leftarrow srcDist - dstDist$
    $AddSkeleton(node, node.parentDist, srcNode, srcDist)$
  end if
  if $srcDist < dstDist$ **then**
    $InsertToSkeleton(srcNode)$
    $node \leftarrow srcNode.nodes[0]$
    $dstDist \leftarrow dstDist - srcDist$
    $AddSkeleton(dstNode, dstDist, node, node.parentDist)$
  end if

Algorithm 6 MapSegments

Require: $sources \leftarrow InputDestOrderedPairs$
Require: $dests \leftarrow InputSourceOrderedPairs$
Require: $map \leftarrow InputMapping$
Require: $threshold \leftarrow InputThreshold$
  $source \leftarrow sources.first()$
  for each $dest$ **in** $dests$ **do**
    if $dest.dist \leq source.dist + threshold$ **then**
      $map[source] \leftarrow map[source] \cup dest$
    else
      $source \leftarrow source.next()$
    end if
  end for

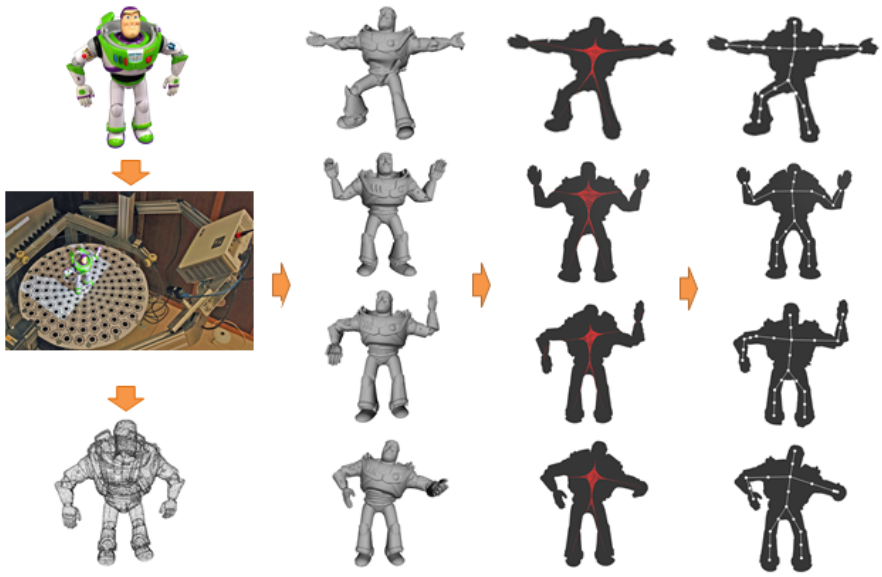

Figure 12: From left to right: scanning of an articulated figure, input point clouds, contracted point cloud using Laplacian smoothing, extracted skeletons.

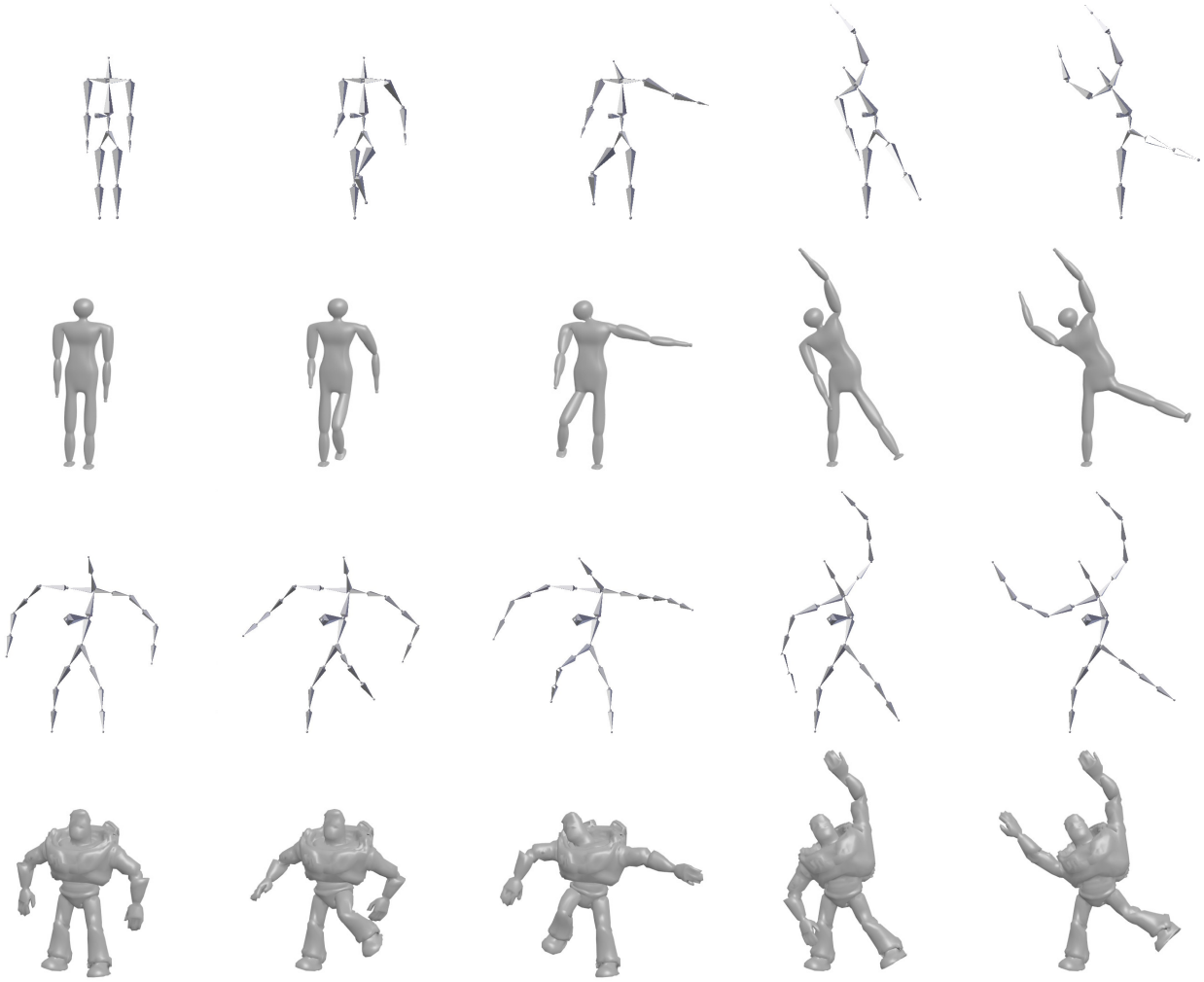

Figure 13: Animation transfer from reference motion to scanned figure.