Extraction of skinning data by mesh contraction with Collada 1.5 support

Martin Madaras*
Supervised by: Tomáš Ágošton†

Faculty of Mathematics, Physics and Informatics
Comenius University
Bratislava / Slovakia

Abstract

The most common approach to animate models and determine their shape attributes in computer graphics is using skeletons. The skeleton and skinning weights can be either assigned manually or computed from an input mesh. This paper proposes the extraction of a skeleton and skinning weights from a mesh, describes how to store computed data in Collada 1.5 and use it for an animation. Firstly, the mesh is contracted using constrained Laplacian smoothing in a few iterations. Then the most important vertices from the contracted mesh are chosen as control points. Multiple edges are removed and vertices that are very close to each other are merged. We select and collapse a vertex pair with the minimum cost in every iteration using a greedy algorithm. The greedy selection is applied repeatably until we have the requested number of bones. In the next step the skinning weights are computed, according to if we want rigid or soft skinning. In the postprocessing stage the user can inspect the skeleton by previewing skinning deformations, make desired changes and export the skeleton to Collada 1.5. Transformation matrices used in a hierarchical skeleton tree are not transformed to joint's local transformation frame, so they are immediately compatible with majority of animation software and libraries. After the Collada file with the mesh, the skeleton and skinning data is exported, data can be imported in animation software such as 3D Studio Max, Blender or Maya and a skinning animation can be rendered.

Keywords: skeleton extraction, mesh contraction, Collada 1.5, skinning

1 Introduction

A frequently used approach for animation and modification of 3D models is based on creating articulated hierarchical structures - skeletons. Skinning data as a skeleton tree and weights can be either assigned manually or computed from an input mesh. The first option is most often chosen by artists, although sometimes it is unnecessary and time consuming. The skeleton has to be created (or imported from templates), rigged into the mesh and influence weights have to be set. Skilled artists are able to create and rig the skeleton in a short time, but sometimes they have to make a lot of rigging adjustments during the skinning process. In this paper we present how to automatically compute the hierarchical skeleton and skinning weights from an input mesh and use them in a skinning animation using Collada 1.5 as export format between our application and a graphic animation software such as 3D Studio Max, Blender or Maya. Our application also provides a way how to examine the computed skeleton before exporting. The skeleton can be inspected by applying skinning deformations using direct kinematics. The interface also allows to dynamically add or remove a bone or a branching if the user thinks some changes are needed.

2 Related work

2.1 Skeleton extraction

Numbers of algorithms have been proposed to compute a skeleton from the mesh geometry. There are three main groups of algorithms: volumetric methods, example based methods and geometric methods. In this paper we focus on geometric methods, they are the most suitable for meshes, because there is no conversion needed. The geometric methods work directly on polygon meshes. The most widely used geometric methods are Reeb graph based methods [2], Voronoi diagram based [7] and Laplacian smoothing based methods [1].

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 1D graph can be computed. This graph encodes topology of the mesh and after resampling it is used as a base for the skeleton. The method based on a harmonic function proposed by [2] captures after resampling all the features of the model well, but requires the user to specify the boundary condition explicitly.

Laplacian smoothing based methods work directly on

*martin.madaras@gmail.com
†tomas.agoston@abyss-studios.sk

the mesh geometry. The main idea of this approach is to apply a well defined filter on mesh vertices. These methods solve the Laplacian system with different weights to constrain the global smoothness and the volume preservation.

A few more approaches to the skeleton extraction problem are worth to mention. [13] extract skeleton by simplifying the Voronoi skeleton with a small amount of user assistance. [11] use repulsive force fields to find a skeleton. The problem has received a lot of attention in recent years and yet the design of a simple and robust method for extracting curve-skeletons remains a research challenge [5].

2.2 Skinning

Many types of mesh deformations can be performed by a skeleton-driven deformation. However, there are types of mesh deformations such as wrinkles, skin folds and another non-bone-driven deformations, where skeleton-driven deformation is not a sufficient option. Examples of non-bone-driven deformation methods are surface based methods [10, 15] and volume based methods [16]. Unfortunately, these methods are not suitable for a real time animation of high resolution meshes in present. Because of its efficiency and simple GPU implementation the most popular skeleton-driven method still remains linear blend skinning (LBS), also known as skeleton subspace deformation. Some real time skinning works have focused on improving the LBS by inferring the character articulation from multiple meshes.

A few solutions to the problem of finding skinning weights were proposed [3], but the methods are either resolution dependent [9] or the weights do not vary smoothly along the mesh [14], causing artifacts with high resolution models.

3 Graph conversion

For running our graph algorithms, we need to have the input mesh as one connected object. The object needs to be converted into a 3D graph, defined by an edge matrix E. It is quite common that models are composed of more objects. These objects appear to be connected visually, but edges in the model structure between these objects are not defined - Figure 1. Also the opposite problem has to be considered. There can be edges defined in an input mesh which connect parts that should not be connected - Figure 2. These edges are remains of the work of graphic designers or artifacts after format conversion and therefore have to be excluded.

3.1 Joining and splitting of objects

Using a simple depth-first search suitable joining distances can be found to connect or disconnect all graph components. The algorithm works in two phases. First, we construct a mesh graph from an input mesh. This mesh graph is constructed in a straight-forward way from the original model structure and may consist of components. In the next step, we compute distance between each pair of components. For each component, the joining distance is computed as a minimum of distances to each other component. In the second phase, we construct the mesh graph again, using joining distance tolerance computed in first phase. This means that each vertex is joined with vertices which lie in the joining distance radius. This condition joins the closest vertices in neighbouring components which creates a one-component graph. An opposite approach can be used when we want to avoid cycles in the final skeleton. We can either compute or manually set splitting distance tolerance. All the edges with a distance smaller than this tolerance, will be removed.

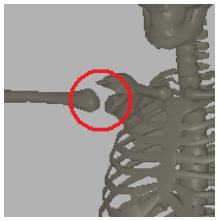

Figure 1: Joining of the mesh graph is needed.

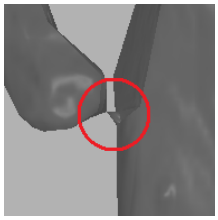

Figure 2: Splitting of the mesh graph is needed.

4 Mesh contraction

For contraction of the generated mesh graph we use the contraction algorithm using Laplacian smoothing proposed by [1]. The algorithm does not alter geometry connectivity (final skeleton curve is homotopic to the original mesh), is noise sensitive and works directly on the mesh geometry (the model does not have to be resampled). Geometry contraction removes details from the surface by applying Laplacian smoothing.

4.1 Laplacian smoothing

Vertex positions are smoothly contracted along their normals by solving the equation (2). The Laplacian smoothing operator (1) was introduced by [6] for surface smoothing. Laplacian smoothing operator L is the $n \times n$ square matrix. This operator is applied on n vertices in vector V as a filter. Term LV approximates curvature flow normals,

so solving $LV' = 0$ removes normal components of vertices and contracts the geometry, resulting into a new set of vertices V'.

$$L_{ij} = \begin{cases} w_{ij} = \cot\alpha ij + \cot\beta ij & \text{if } (i,j) \in E \\ \sum_{(i,k)\in E}^{k} -w_{ik} & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

where:

E – is the set of edges defined in the previous section during a graph conversion process

$\alpha ij, \beta ij$ – are the opposite angles corresponding to the edge (i,j) [6]

4.1.1 Linear equation

Unconstrained solving of this equation contracts the mesh graph into a single point, so the equation is solved in more iterations with carefully chosen weights which control the contractions. W_L and W_H are diagonal weighting matrices which control the contraction process. Weighting matrices have to be updated after each iteration to drive the iteration process into a desired state. By increasing $W_{L,i}$ we can increase the collapsing speed for vertex i and by increasing $W_{H,i}$ we increase the attraction weight to attract vertex i to its current position. All the $W_{L,i}$ are in the next step multiplied by a predefined constant (s_L) and $W_{H,i}$ are updated in such a way, that the attraction weight is multiplied by a ratio of the change of the area of faces adjacent to the vertex i. If the area of adjacent faces is smaller, the multiplicative term is higher, vertices are more attracted into their current positions and in the next iteration the geometry is less contracted in these vertices.

$$\begin{bmatrix} \mathrm{W}_L L \\ \mathrm{W}_H \end{bmatrix} V' = \begin{bmatrix} 0 \\ \mathrm{W}_H V \end{bmatrix} \quad (2)$$

Each step of the iterative contraction process works as follows (t denotes the iteration number):

1. Solve $\begin{bmatrix} \mathrm{W}_L^t L^t \\ \mathrm{W}_H^t \end{bmatrix} V^{t+1} = \begin{bmatrix} 0 \\ \mathrm{W}_H^t V^t \end{bmatrix}$
2. Update $W_L^{t+1} = s_L W_L^t$ and $W_{H,i}^{t+1} = W_{H,i}^0 \sqrt{A_i^0 / A_i^t}$, where A_i^0 and A_i^t are the original and current areas of adjacent faces for the vertex i, respectively.
3. Compute the new Laplacian operator L^{t+1} with the vertex positions computed from the previous iteration V^{t+1} using equation (1).

The iterations converge when the volume is close to zero. After each iteration, the volume approximation has to be computed. In our implementation we used an approximation algorithm which subdivides the bounding box of the model into an octree structure.

Figure 3: The half-edge collapse $(\tilde{v_2} \to \tilde{v_1})$.

5 Skeleton construction

The contracted mesh graph from the last iteration is simplified, very close vertices are merged and a greedy algorithm is used to select the most important control points. In this section we are going to work with already contracted vertices from previous section, they will be denoted as $\tilde{v_i}$. In our implementation the user can choose, how many control points he wants. We used 24 control points by default which worked well for almost all GPUs and also it is a sufficient number to control the skinning process of complex high resolution models. During the collapsing process, for each control point, the collapsed vertices into this control point are stored in a hash map. Vertices are merged into control points and every control point is shifted into the center of its local mesh area which can be computed from the stored hash map. For each control point, the volume of the mesh region the control point represents in the original mesh is computed. The control point which represents the largest volume is chosen as the skeleton root.

5.1 Mesh graph simplification

The first step in the mesh graph simplification is to collapse all edges, whose vertex distance is smaller than a predefined threshold. Vertices in such pairs can be interpreted as the same control point, because the distance between them is very small and the influence over the mesh vertices is almost the same. The second step is to collapse edges which are the least important. For every edge the cost value is computed and edges with minimum cost are collapsed. For simplicity we apply half-edge collapse. The half-edge collapse $(i \to j)$ merges vertex i to vertex j and removes all the faces that are incident to the collapsed edge. The half-edge collapse $(\tilde{v_2} \to \tilde{v_1})$ is shown in Figure 3. This step is repeated so many times that in the end we will have the desired number of control points. The cost value is computed as a weighted sum of a sampling cost term and a shape cost term.

The sampling cost term (3) penalizes collapsing which generates long edges, because in that way we will loose the good mapping between the skeleton and the surface. The term is computed as a weighted sum of distance of adjacent vertices to the collapsed vertex.

$$\mathrm{SCT_a}(i,j) = \|\tilde{v_i} - \tilde{v_j}\| \sum_{(i,k)\in\tilde{E}} \|\tilde{v_i} - \tilde{v_k}\| \quad (3)$$

where:

$\tilde{E}$ – is the current simplified edge set

The shape cost term (4) works in almost the same way as in QEM simplification method [8] with one change. The error matrices are computed over the edges, because the contracted mesh has zero area faces, so the original volume based approach cannot be used. A symmetric 4×4 matrix Q is associated with every vertex. Q is defined in a such way, that term $\mathrm{F_i}(p) = p^T Q_i p$ is a squared distance between point p and the edge (i, j). The initial error matrix for vertex i is the sum of all squared distances to its adjacent edges. For more detailed description of these matrices, their initialization and their use in calculation of the shape cost term we refer to [1]. For a given contraction $(\tilde{v_i}, \tilde{v_j})$ a new matrix Q needs to be derived to approximate the error at $\tilde{v_j}$. Error matrices from previous iterations are stored, so each cost update involves only matrix addition. The shape cost term guarantees to keep the shape of the contracted mesh graph as undisturbed as possible during the simplification. The idea to assign these cost terms to each edge after the iterative contraction converges successfully origins from [1].

$$\mathrm{SCT_b}(i, j) = \mathrm{F_i}(\tilde{v_j}) + \mathrm{F_j}(\tilde{v_j}) \tag{4}$$

6 Binding skin vertices

Once we get the skeleton, we bind the mesh vertices to its joints. If we attach a rigid model, the skin is supposed to be inflexible. Therefore we only anchor a mesh vertex to one nearest control point. In other way, when we want to animate a character, we want the vertices to transform smoothly. In this case, mesh vertices have to be anchored to more control points with corresponding weights.

6.1 Skinning weights

Skinning indices are computed by finding a set of closest control points to each vertex. The geodesic distance is used as a distance measure. A distance between each pair of vertices on the mesh graph from $0th$ iteration (after conversion from an input mesh) is calculated and stored in matrix D. It is calculated using Floyd-Warshall algorithm [4], before the mesh graph is contracted. For each control point C_k, the closest mesh graph vertex is found, for instance v_j. Then, the resulting geodesic distance (5) between the control point C_k and the mesh vertex v_i is computed as a sum of distance between v_j and v_i on the mesh graph calculated by Floyd-Warshall algorithm and the euclidean distance between C_k and v_j. The illustration is shown in Figure 4.

$$\mathrm{gd}(i,k) = \mathrm{D}[i, j] + \mathrm{d}(C_k, v_j) \tag{5}$$

where:

$\mathrm{d}(C_i, v_k)$ – is euclidean distance between the mesh vertex v_j and k^{th} control point C_k

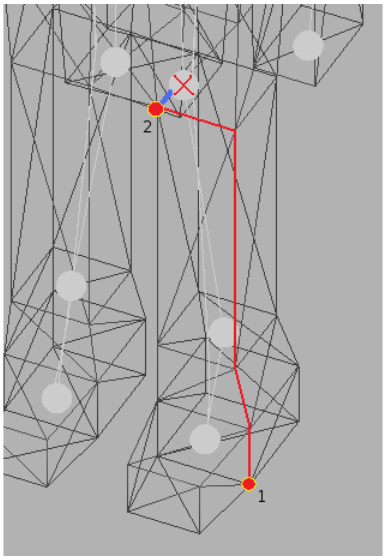

Figure 4: This figure shows computation of the geodesic distance between the control point marked with red cross and the mesh vertex (1) on the bottom right. The red path is the shortest path on the mesh calculated by Floyd-Warshall algorithm and the blue line is the euclidean distance between the selected control point and the closest vertex (2) to this control point on the mesh.

Weights (6) are assigned in a way that weight sum for each vertex is equal to 1.0. Fractions are constructed in a way that weights are indirectly dependent on the geodesic distance. This construction guarantees that the closer control points will have greater influence over mesh vertices than the further ones. The geodesic distance is a real-value function defined on the mesh surface. Because the function varies smoothly along the mesh, the resulting weights are fluently distributed over the mesh regions.

$$\mathrm{weight}(i,k) = \frac{\frac{1}{\mathrm{gd}(i,k)}}{\sum_{k' \in S} \left(\frac{1}{\mathrm{gd}(i,k')} \right)} \tag{6}$$

where:

$\mathrm{gd}(i,k)$ – is geodesic distance between the mesh vertex v_i and k^{th} control point C_k

S – is the set of control point indices controlling the vertex v_i

Floyd-Warshall algorithm has time complexity $O(n^3)$, so it takes quite a long time on models with higher number of vertices. To optimize that time, we can use a downsampled mesh for this computation. For downsampling, we used previously mentioned QEM simplification method [8]. The downsampled mesh preserves the mesh branching, tunnels and important vertices.

6.2 Joint matrices

Bind pose matrices and current transformation matrices for all the nodes are stored in the global (root local) space. They are not transformed into node's local space. The disadvantage is that during the skinning preview the matrices have to be transformed into node's local space. On the other hand, the main advantage is that the skeleton structure is compatible with the majority of animation software.

6.3 Skinning on GPU

Our framework provides linear blend skinning implemented on GPU for real-time examination of computed skeletons. It is the most suitable method how to inspect the skeleton structure and data, because of its efficiency and simple GPU implementation. After the skeleton is computed, the "bind pose" world-space snapshot of all transformation matrices of the skeleton nodes is taken, denoted as B_i, for each skeleton node. During the real-time deformation process, transformation matrices are computed. Each transformation matrix, storing the current affine transformation, denoted as P_i, is computed each time the user manually changes skeleton nodes. In each frame, the resulting transformation M_i is computed as $M_i = P_i B_i^{-1}$ on CPU and uploaded into GPU. New deformed vertices are computed using GLSL shader as :

$$v' = \sum_{i=0}^{n} w_i M_i v_i \tag{7}$$

where:

M_i – is the resulting transformation matrix, computed on the CPU as $M_i = P_i B_i^{-1}$

w_i – is the associated weight

v_i – is the original vertex position in the M_i coordinates system

6.4 Robustness

The mesh contraction and the skeleton extraction phases are pose independent. It enables extraction of compatible skeletons from different poses of the same model. By compatible, we mean compatible in the sense of the same branching, tunnels and the homotopy with an input mesh. Also, the length of preserved edges will be the same, because edges are collapsed in the same order. In the end of the skeleton construction process, all corresponding skeleton bones will have the same lengths. The geometry meshes in different poses have corresponding edges of the same length as well, so the skinning weights and indices will result into identical values.

7 Collada 1.5 support

Model data and all important skinning data is saved in a compatible way into the Collada *.dae* XML file. Collada is a Collaborative Design Activity for establishing an interchange file format for interactive 3D applications. Collada supports storing of the mesh geometry, the hierarchical skeleton structure, indices, weights and inverse bind pose matrices. With version 1.5 comes the possibility to store kinematics as well. To use the full support of a kinematics model in Collada 1.5, computation of approximation of the minimum, the maximum and the current angle for each joint is needed. Angles can be computed using the inverse kinematics approach or assigned manually by the user. In our implementation we use the latest version of Collada DOM 2.2 with Collada 1.5 [12] support.

8 Results

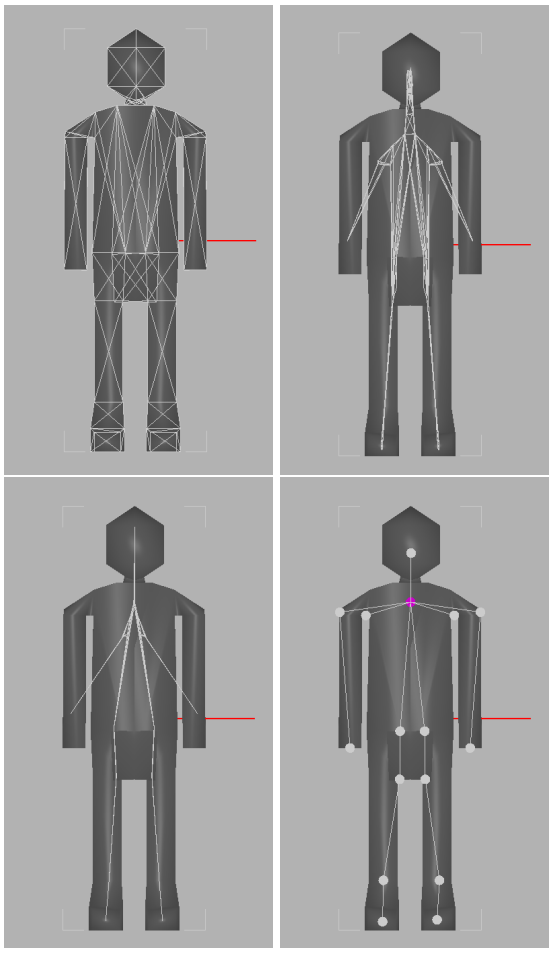

Figure 5: The contraction and the extraction of the skeleton from low resolution geometry. (Top-left) The converted mesh graph. (Top-right) The mesh graph after 2 iterations. (Bottom-left) The mesh graph after the last iteration, the volume approximation is close to zero. (Bottom-right) The extracted skeleton.

We have tested the framework on a wide range of different models. We have achieved good results with both high resolution and also with low resolution models. The more

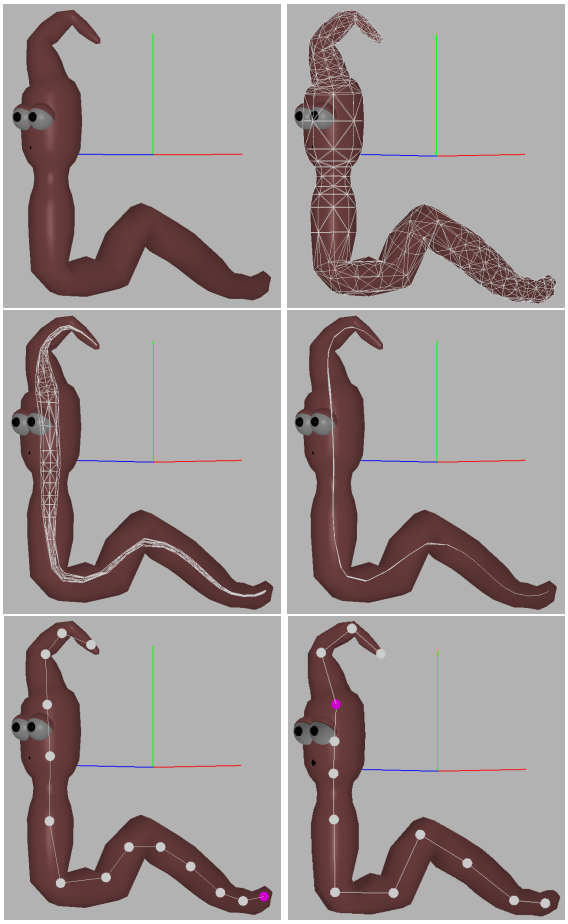

Figure 6: The contraction and the extraction of the skeleton in few iterations from higher resolution geometry and its comparison to manually rigged skeleton by an artist. (Top-left) An input model. (Top-right) The converted mesh graph. (Middle-left) The mesh graph after 2 iterations. (Middle-right) The mesh graph after the last iteration. (Bottom-left) The extracted skeleton. (Bottom-right) The skeleton rigged by an artist.

vertices the model has, the better skeleton can be computed, but the algorithm takes more time. Also the low resolution models (less than 5000 polygons) can be contracted in a good way, but it is harder to set good contraction weights. Setting the right contraction weights is the most problematic part of this approach. Low resolution models are more sensitive for high curvature differences and can be easily over-contracted. Weights often have to be set manually and the user needs some experience. Contraction of high resolution models is more deterministic and good weights can be set automatically. Contraction of geometry with lower number of vertices can be seen in Figure 5 and contraction of geometry with more vertices can be seen in Figure 6 and 7. The model of a worm in Figure 6 was published with a manually rigged skeleton (Bottom-right image). After the comparison we can conclude that the extracted skeleton is very close to the manu-

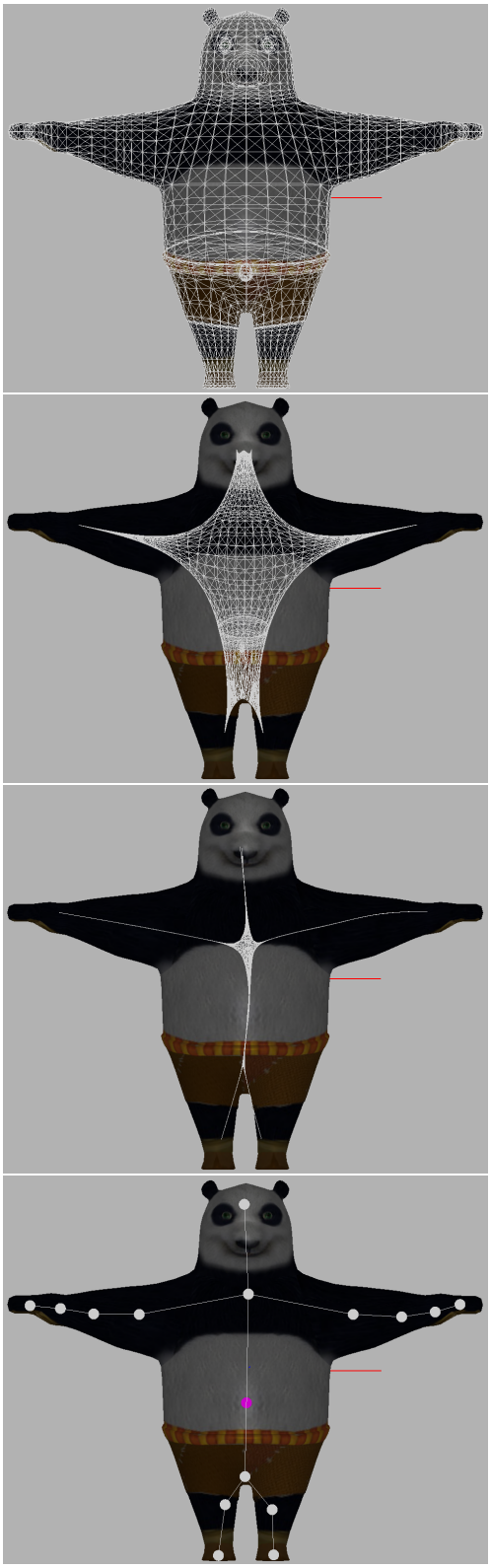

Figure 7: Another example of higher resolution geometry. The extracted skeleton has sparser nodes at the core parts. This feature can be observed, because many faces at core parts are contracted into the same region. The points are also moved towards the mesh boundary of the limbs, because of the shifting of the control points to the center of its local mesh.

ally rigged one. It can be used as a sufficient supplicant for a manually created and rigged skeleton. The major difference can be observed on the both ends of the worm and in the largest bend. Nodes of the skeleton tree were pushed into their centers of local mesh areas and that is why they were pushed inside, away from the mesh boundary.

9 Conclusion

In this paper we propose a framework for extracting a skeleton and skinning data from the geometry mesh using an iterative mesh contraction. The approach begins with converting the geometry into a mesh graph. This graph is iteratively contracted and a greedy algorithm is applied to choose the most important subset of vertices. In the next step, these vertices are converted into a hierarchical skeleton. Important skinning data such as indices and weights which are controlling the influence of the skinning process over vertices are computed as well. When the data is computed, our framework also provides a way to inspect the skeleton, simulate the skinning deformation process with full GPU support and allow the user to change the skeleton branching, if it is needed. After all, geometry and skinning data can be exported into Collada 1.5 *.dae* file and transferred into an external application to create animations.

The extracted skeletons have sparser nodes at the core parts of the model. This feature can be observed, because many faces at core parts are contracted into the same region. Computed skeletons are independent of the size and resolution of the models. The approach is insensitive to noise, but works only for closed mesh models with 2D manifold connectivity.

References

[1] Oscar Kin-Chung Au, Chiew-Lan Tai, Hung-Kuo Chu, Daniel Cohen-Or, and Tong-Yee Lee. Skeleton extraction by mesh contraction. In *SIGGRAPH '08: ACM SIGGRAPH 2008 papers*, pages 1–10, 2008.

[2] Grégoire Aujay, Franck Hétroy, Francis Lazarus, and Christine Depraz. Harmonic skeleton for realistic character animation. In *Symposium on Computer Animation*, pages 151–160, 2007.

[3] Ilya Baran and Jovan Popović. Automatic rigging and animation of 3d characters. In *SIGGRAPH '07: ACM SIGGRAPH 2007 papers*, page 72, 2007.

[4] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*, pages 558–565. MIT Press and McGraw-Hill, first edition, 1990.

[5] Nicu D. Cornea, Deborah Silver, and Patrick Min. Curve-skeleton properties, applications, and algorithms. *IEEE Transactions on Visualization and Computer Graphics*, 13(3):530–548, 2007.

[6] Mathieu Desbrun, Mark Meyer, Peter Schröder, and Alan H. Barr. Implicit fairing of irregular meshes using diffusion and curvature flow. In *Proceedings of ACM SIGGRAPH 99*, pages 317–324, 1999.

[7] Tamal K. Dey and Jian Sun. Defining and computing curve-skeletons with medial geodesic function. In *SGP '06: Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 143–152, 2006.

[8] Michael Garland and Paul S. Heckbert. Surface simplification using quadric error metrics. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[9] Sagi Katz and Ayellet Tal. Hierarchical mesh decomposition using fuzzy clustering and cuts. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pages 954–961, 2003.

[10] Yaron Lipman, Olga Sorkine, David Levin, and Daniel Cohen-Or. Linear rotation-invariant coordinates for meshes. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 479–487, 2005.

[11] Pin-Chou Liu, Fu-Che Wu, Wan-Chun Ma, Rung-Huei Liang, and Ming Ouhyoung. Automatic animation skeleton construction using repulsive force field. In *PG '03: Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, page 409, 2003.

[12] Sony Computer Entertainment Inc. *COLLADA Digital Asset Schema Release 1.5.0*, apr 2008.

[13] Marek Teichmann and Seth Teller. Assisted articulation of closed polygonal models. In *SIGGRAPH '98: ACM SIGGRAPH 98 Conference abstracts and applications*, page 254, 1998.

[14] Lawson Wade and Richard E. Parent. Fast, fully-automated generation of control skeletons for use in animation. In *CA '00: Proceedings of the Computer Animation*, page 164, 2000.

[15] Yizhou Yu, Kun Zhou, Dong Xu, Xiaohan Shi, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Mesh editing with poisson-based gradient field manipulation. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Papers*, pages 644–651, 2004.

[16] Kun Zhou, Jin Huang, John Snyder, Xinguo Liu, Hujun Bao, Baining Guo, and Heung-Yeung Shum. Large mesh deformation using the volumetric graph laplacian. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers*, pages 496–503, 2005.