An Interactive Editor for Curve-Skeletons: SkeletonLab

Simone Barbieri[a], Pietro Meloni[b], Francesco Usai[b], L. Davide Spano[b], Riccardo Scateni[b]

[a]*Centre for Digital Entertainment, Bournemouth University, Bournemouth, United Kingdom*
[b]*Department of Mathematics and Computer Science, University of Cagliari, Cagliari, Italy*

Abstract

Curve-skeletons are powerful shape descriptors able to provide higher level information on topology, structure and semantics of a given digital object. Their range of application is wide and encompasses computer animation, shape matching, modelling and remeshing. While a universally accepted definition of curve-skeleton is still lacking, there are currently many algorithms for the curve-skeleton computation (or *skeletonization*) as well as different techniques for building a mesh around a given curve-skeleton (*inverse skeletonization*). Despite their widespread use, automatically extracted skeletons usually need to be processed in order to be used in further stages of any pipeline, due to different requirements. We present here an advanced tool, named SkeletonLab, that provides simple interactive techniques to rapidly and automatically edit and repair curve skeletons generated using different techniques proposed in literature, as well as handcrafting them. The aim of the tool is to allow trained practitioners to manipulate the curve-skeletons obtained with skeletonization algorithms in order to fit their specific pipelines or to explore the requirements of newly developed techniques.

Keywords: Curve-Skeleton, 3D Meshes, Geometry Processing, Interactive Editing

1. Introduction

A curve-skeleton is a compact mono-dimensional representation able to provide meaningful information about both topology and the volume of a shape. While the skeleton of a two dimensional shape is defined as its *Medial Axis Transform* [1], which is the locus of centres of its maximal inscribed discs, this definition leads, for three dimensional shapes, to a collection of connected curves and sheets that are impractical to use in most real world applications.

Constraining the skeleton of a 3D shape to be a one-dimensional structure results in a simpler and more intuitive representation, that is also easy and natural to manipulate. As of today, however, there is no unique, precise and universally accepted definition of curve-skeletons, hence different approaches for its computation have been proposed, each obtaining results with different features, characteristics and defects [2].

The range of applications in which curve-skeletons are used is wide and comprises computer animation [3], shape matching [4], modelling [5], remeshing and quad layout extraction [6], polycube [7], and hexahedral mesh construction [8]. Since each of these applications requires skeletons with different properties, further processing is often required in order to obtain optimal results in the pipeline in which they will be used. Moreover, it is often difficult to define algorithms to process a skeletal structure in order to reflect the required features, due to the semantic nature of the information it conveys.

Tools for interactive creation of curve-skeletons already exist (e.g. in the the medical imaging field [9]). These tools usually allow the user to create a curve-skeleton from scratch so that it can be used as the input for an inverse skeletonization algorithm. The main difference with our tool is that, due to our different goals, the basic operations they offer are not sufficient for editing an automatically extracted skeleton.

In [6] an algorithm for computing a coarse quad-layout starting from a shape and its curve skeleton was introduced. We discovered that using automatically extracted skeletons often brought to sub-optimal results even when using some simplification strategies. This led to the development of the tool we present in this paper, which has shown to be easy-to-use and practical for interactively editing curve-skeletons. It allowed us to obtain optimal results within our quad-layout computation method and we think it can allow other researchers or practitioners to obtain, even within minutes, curve-skeletons that are tailored for their specific tasks.

The rest of the paper is organized as follows: in Section 2 we will give a brief overview of the different algorithms for computing curve-skeletons at the state-of-the-art, interactive tools for curve-skeleton handcrafting will be discussed as well; in Section 3 we will discuss the limitation of using the current skeletonization approaches in real world applications and why the tool we are presenting in this paper could be a useful resource; in Section 4 we will give an overview of our tool, describing its functionalities; in Section 5 we will report the results of a complete user evaluation session conducted in our lab; in Section 6 we will present an example of a specific use case, to give an idea of the capabilities of our tool; in Section 7 we will draw our conclusions, consider the limitations, and explain what could be done in the future to improve the work.

The project page with binaries and link to the repository containing the entire source code of the presented tool is available at `http://francescousai.info/skel_lab`. This paper is a substantial extension and revision of the work “Skeleton Lab:

an Interactive Tool to Create, Edit, and Repair Curve-Skeletons" [10], presented at the 2015 EuroGraphics Italian Chapter Conference, held in Verona in October 2015. The work received the Best Paper Award at the end of the Conference.

2. Related work

Several automatic skeletonization methods have been presented in the last two decades. These are commonly categorised based upon their input, which can be a surface mesh [11, 12], a point cloud [13, 14] or a voxel grid [15, 16]. Volumetric representations, while important, are not handled by our tool, hence they will be not discussed. We refer to [2] for a recent survey. Currently, a universally accepted definition of *curve-skeleton* is still lacking. The *medial geodesic skeleton* defined in [17] is the only meaningful attempt to fill this gap; however, while the theoretical definition is precise and sound, the implementation of the skeletonization algorithm provided by the authors presents some defects, such as long computational times and sensitivity to small perturbations of the surface. Both these issues arise due to the well known sensitivity of the medial axis itself, which is computed and processed to obtain the final skeleton. The main issue is the presence of noisy skeletal paths and spurious branches, especially between nodes where the branching occurs.

Since different skeletonization approaches exist, it is difficult to compare the quality of the results of each method. Important information about the desirable properties of a curve-skeleton can be found in [18].

The current trend in skeletonization is to use a contraction-based approach, which relies on contracting the 3D shape represented as a triangle mesh accordingly to its mean curvature-flow until it collapses to a monodimensional object. We refer to [15, 19] for a qualitative comparison of the most representative methods based on this idea. A recent contraction-based method is [11] which is a robust skeletonization algorithm whose resulting skeletons are topology-preserving, usually well centered and smoother than previous similar methods.

Notable exceptions to this trend are [20, 21, 22] which, rather than relying on the geometric properties of the input, operate emulating human perception, synthesising the curve-skeleton of a 3D object from those of its 2D silhouettes (obtained from different points of view).

This approach, although less robust than [17, 23, 11] presents heuristics for collapsing spurious branches and closing loops whenever two terminal nodes have intersecting maximal balls, which are beneficial for the resulting structure. A particularly notable example of point cloud based methods is the work of Huang et al. [14], which allows to obtain high quality skeletons from raw and potentially incomplete scans by using the L1-median operator.

2.1. Skeleton editing tools

The term *skeletonization* denotes the process of computing a curve-skeleton from a 3D shape. Similarly, *inverse skeletonization* can be defined as the process of building a 3D shape around a curve-skeleton respecting its structural and volume information.

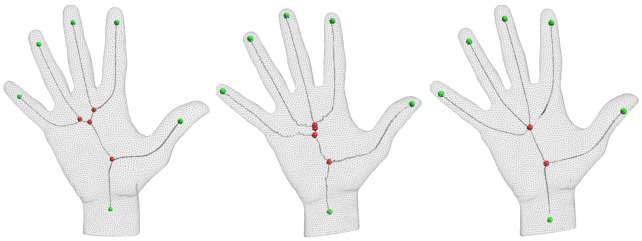


Figure 1: A direct comparison of three automatic skeletonization methods. [11] (left), [17] (center) and [21] (right) on the same hand model. One can notice that the terminal nodes are the same in all the three skeletons, while the internal nodes are different in number and position. It is difficult to tell which one is the most correct.

In recent years, inverse skeletonization became a quite active research topic, giving rise to different research works [24, 25, 26] and commercial solutions [27, 28] for building 3D shapes, especially creature-like, from a user created skeleton.

The aim of these tools is to quickly create 3D models that will have to be further refined. They are designed to allow the user to handcraft a skeleton, providing a small set of simple skeleton editing operations such as adding new nodes, moving them around and changing the radii of their associated balls. These basic operations do not provide a sufficiently wide toolset for a user that needs to process the results of one of the presented skeletonization methods. Our tool addresses this need, allowing users to operate on curve-skeletons at different granularities with functionalities that are absent in other tools and have shown to be useful.

3. Motivation

From a practical point of view all the skeletonization methods presented in the previous section have specific properties and issues, and their resulting skeletons are typically difficult to use without processing. We noticed, from our own experience and the results of the experiments found in literature, that curve-skeletons are usually assumed to have some specific properties or are manually edited to have them (e.g. in [26, 7]).

Each of the methods presented in section 2 requires some input parameters to be set in order to fine-tune how the algorithm behaves, especially regulating its ability to catch small scale features. This brought us to make the following observations:

- The computation of a good curve-skeleton is not a fully automated operation. It requires a, sometimes complex, parameter setting that could be tightly coupled to both the specific application and the specific shape; this usually takes several attempts before obtaining the desired skeleton.
- Most of the parameters pertain to the detection of small scale features. On the one hand, more sensitivity usually means more noise and spurious branches; on the other hand, less noise could potentially mean missing meaningful features of the shape being processed.
- Given the semantic nature of the information encoded in a curve-skeleton it is difficult, from an application agnostic

point of view, to define what a meaningful feature is and how much noise can be tolerated in order to capture it.

As can be seen in Fig. 1 different skeletonization methods give very different results even on models of moderate complexity. The three methods used in the experiment have different approaches and lead to distinct results. Since we can not use a single universal and formal definition of 3D curve-skeleton, it is not possible to say which of the three skeletons is correct. Furthermore we can say that rather than evaluating if an extracted curve-skeleton is correct or not, it is worth to evaluate if it meets certain desired properties and if it can bring optimal results or not.

We have observed that it is usually not practical to rely on a completely automatic pipeline, because the input parameters of these algorithms must be fine tuned. Moreover, the user has to decide the balance between meaningful features versus noise and spurious branches. Taking this into consideration, we think that working in a pipeline where an automatic extraction stage (with a limited choice of standard parameter values) followed by an interactive editing stage is more practical.

3.1. Applicative contexts

As mentioned in the introduction, in [6] we proposed an algorithm for computing a coarse quad-layout starting from a shape and its curve skeleton. We found that optimal skeletons for this application are robust, connected, and reliable [18]. Strong homotopy is not required but it is beneficial. Optimal results have been obtained by using skeletons composed of a small number of nodes that still approximate the shape accurately. Another important requirement is for the skeletons to have the associated balls of the endpoints not intersecting other balls. In some cases, optimal results have been achieved with a little interactive editing of the automatically extracted skeleton by collapsing spurious branches, manually adjusting the missing ones and sub-sampling the branches while retaining only the most representative nodes.

Another applicative context which benefits from dealing with a small number of nodes is the definition of kinematic skeletons. They are in fact usually composed of few nodes for each branch, requiring particular care on the nodes' position. In this scenario an automatic simplification of a skeleton can potentially lead to results that are not well suited for the purpose. This could happen because the nodes associated with the shape's articulations have to be preserved, but the presence of an articulation itself is a semantic feature that is usually complex to capture algorithmically.

4. Skeleton Lab

The presented tool, Skeleton Lab, allows the user to both handcraft curve-skeletons and edit the existing ones, hence it provides functionalities that are absent in the tools described in Section 2.1, due to their different purposes. This section will present its distinguishing features, their purpose and some design choices.

We consider the skeleton as an attributed graph $G = (N, L)$ where N is the set of nodes and L the set of links between nodes. Each node $N_i \in N$ has a few attributes:

- Position in space.
- Radius of the associated ball.
- Type of the node (the classification follows).
- List of its neighbouring nodes, since we do not explicitly store the links.

We classify the nodes of the skeleton as: *Joint nodes* (**Jn**) that have two incident arcs; *Leaf nodes* (**Ln**) that have only one incident arc; and *Branching nodes* (**Bn**) that have more than two incident arcs. We also allow the user to assign a higher semantic value to a **Jn** marking it as an *Articulation* (**An**), e.g. in case of a human ankle or elbow.

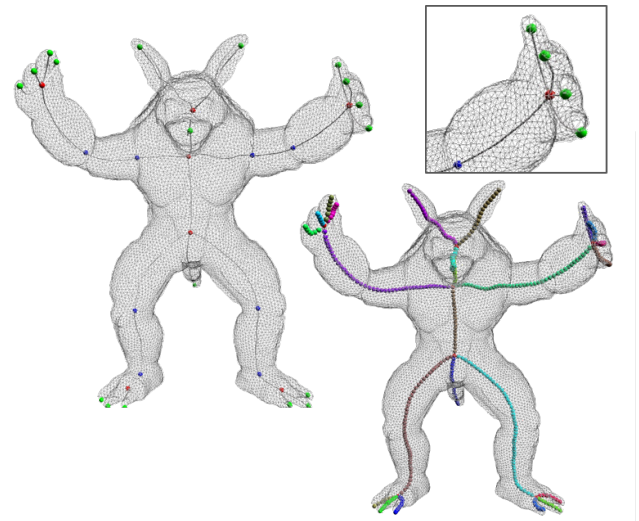


Figure 2: A visual explanation of the terminology we introduced. In the left image each **Bn** is red, each **Ln** is green, and each **An** is blue, while no **Jn** is shown. In the right image the nodes of **Branch** have a different color.

With the term **Branch** we refer to a sequence of linked nodes starting and ending with either a **Bn** or a **Ln**. When a branch ends with a **Ln** we call it ***Ln**-ending*, if both the starting and ending nodes are **Bn** it is said to be *internal*, while when the starting and the ending nodes are exactly the same, we have a *Looping-Branch*.

With our tool the user is allowed to visualize and manipulate a curve-skeleton and optionally view the triangle mesh it refers to. We experimented our tool with skeletons obtained with different methods, namely the ones presented in [17, 20, 11].

By design we have chosen to group functionalities into two different modes:

Node mode in which each operation refers to the node or the set of nodes currently selected.

Branch mode allows direct manipulation of a selected branch with operations that are meaningful only when considering an entire branch.

Moreover, there are several operations that affect the skeleton as a whole. From our experience manually editing a curve-skeleton is usually performed through a trial-and-error approach, hence undo/redo is available for both operations and selection.

4.1. Basic operations

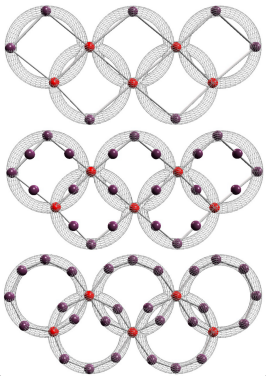
All the tools presented in Section 2 allow the user to perform basic operations only, that are: adding a new node connected to an existing one; translating and rotate a selected set of nodes; changing the radii of their associated balls. These basic functionalities of a skeleton editor are available in our tool. They are trivial and, thus, they will not be described further.

4.2. Node mode

Here we present all the possibilities offered in node mode.

4.2.1. Adding midpoint and constraining movement

Once two existing and directly connected nodes are selected, the user is allowed to create a new node connected to them and placed at the midpoint of the segment that connects them (see inset).



Strictly related to this operation, is the constrained movement of nodes. Our tool allows the user to move a selected **Jn** constraining its movement only to points in space that are linearly interpolated between the position of its two neighbours, in order to fine tune its position. These two operations have shown to be useful when creating a skeleton from scratch with a progressive refinement, where only **Bn**'s and **Ln**'s are created at first and branches can be progressively modelled adding **Jn**'s between them.

4.2.2. Node Removal

Removing one or more nodes is a simple operation with a number of different cases that have to be taken into account. In fact, when removing a single node there are three possible cases depending on what type of node is going to be removed:

- **Ln**: the node is removed without any further operations.
- **Jn**: the node is removed and the two nodes linked to it are then connected to each other.
- **Bn**: the user is required to choose between two possibilities. The first one is to delete all the node's links. Since we do not allow to create separate components, all the **Ln**-ending branches connected to the **Bn** will be removed. Other branches, the ones connected to other **Bn**, will be kept unchanged except for the fact that one of their endpoints will be removed, becoming a **Ln**-ending branch. Alternatively, the user can transfer all the links of the **Bn** that is going to be deleted to one of its neighbouring nodes, selected by the user.

The tool allows to handle isolated nodes or branches but does not allow to explicitly create them. This is a design choice: since a curve-skeleton is always a single connected component, SkeletonLab does not allow the user to create disconnected components, but it allows to load a disconnected skeleton in order to repair it interactively. When a user tries to remove a set of nodes in *Node Mode*, the operation is performed as a sequence of single node removals. When **Bn**'s are involved, all the **Ln**-ending branches that are connected to them are traversed and their nodes removed.

4.2.3. Copy and paste

Especially when creating a skeleton from scratch, the user may want to replicate a part of the skeleton that (s)he is editing. For this reason our tool allows to copy and paste a set of connected nodes. When copying, the user is required to choose a *Source* node, when pasting a *Destination* one. The *Source* node has to be chosen from the ones that are being copied, the *Destination* node can be any node of the skeleton. When pasting the copied nodes, *Source* will be merged with *Destination* in order to keep a single connected component. *Destination* will be modified only inheriting all *Source*'s copied neighbours. Copy and paste are disjoint operations, and, as such, one can perform them in different moments. The process is depicted in figure 3.

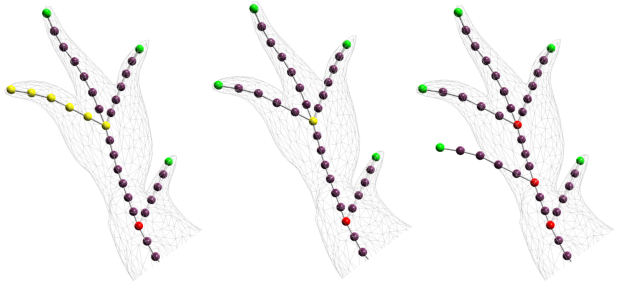


Figure 3: Copy&paste. The user selects a sequence of nodes (left), activates the copy (similarly to other softwares that supports this functionality), selects the *source* node (middle); then selects the destination node and pastes the copied nodes (right). The selected destination node has now become a **Bn** (right).

4.2.4. Creating and breaking links

A curve-skeleton is essentially an undirected graph. Skeleton Lab allows the user to explicitly manipulate the graph's connectivity with three operations:

- Merge two unconnected nodes creating a looping-branch.
- Create a link between two unconnected nodes.
- Break an existing link between two nodes. We use the Dijkstra's algorithm to check if breaking the link will create two separate components, if so we prevent the operation.

4.3. Branch mode

The operations explained so far operate on one or more selected nodes, while the operations that will be presented in this section operate on a higher level of granularity, allowing the users to work on an entire branch of the skeleton.

4.3.1. Pruning

The pruning operation allows the users to easily shorten the selected **Ln**-ending branch, removing its current **Ln**.

4.3.2. Branch Removal

If the branch is a **Ln**-ending one, it will be removed without further operations. In case of an internal branch the user will be prompted to choose whether (s)he prefers to merge the two **Bn**'s at their midpoint, or (s)he would rather retain the position of one of the two. This operation is particularly useful when repairing skeletons, since one of the most common defects is the presence of short spurious branches.

4.3.3. Spurious branches removal

A spurious branch is a branch of the skeleton that breaks homotopy, in the sense that it does not represent any meaningful component of the shape. Identifying spurious branches would require to understand the shape's features at a semantic level, even if it can be approximated with the use of the volumetric information provided by the radii of the associated balls. Skeleton Lab provides the capability of collapsing such branches implementing the topological operations proposed in [20]. In particular, every branch for which the Zones of Influence [29] of its two endpoints intersect (i.e. their associated balls intersect) will be removed.

4.3.4. Resampling

Skeletons computed with state-of-the-art skeletonization algorithms are often composed of a large number of nodes, in fact they can easily reach a thousand nodes, even for models of moderate complexity. In usual applications, this level of detail leads to redundancy and is not necessary, indeed it could be preferable to deal with a smaller number of nodes.

In order to cope with this issue, we allow the users to change the number of nodes of a branch with a *resampling* operation.

Resampling is done by arc-length parametrization adopting the method presented in [30] with slight modifications in order to take into account both position and radii of the involved nodes. In this way, the three-dimensional structure of the branch is maintained and the radii of the new nodes are approximated from the original ones. Both sub-sampling and super-sampling are allowed, but only super-sampling is a smoothness preserving operation.

A special case for resampling occurs when the branch contains some **An**'s, since they have been marked by the user as semantically relevant **Jn**'s, their position and radii will be preserved.

Furthermore since subsampling must retain these nodes, the minimum number of nodes a user can choose for resampling is equal to $E + \#\mathbf{An}$, where E is the number of distinct endpoints of the branch.

4.3.5. Approximation

The typical use case for resampling is skeleton simplification. However, it could be sometimes required to maintain a subset of the original nodes while simplifying. In these cases skeleton simplification through resampling cannot be applied. A different approach to lowering the number of nodes of a skeletal branch is to approximate it. We implemented two different solutions. The first one relies on the Douglas-Peucker algorithm [31]. The second is a simple heuristic that operates traversing the branch and removing all the nodes for which their distance to the last node visited and not removed is smaller than a given threshold (i.e., calculated as the sum of their associated ball radii multiplied by a scaling factor). These two approaches approximate the branches keeping the original skeletal nodes, thus not guaranteeing to obtain a smooth curve with uniformly distributed nodes.

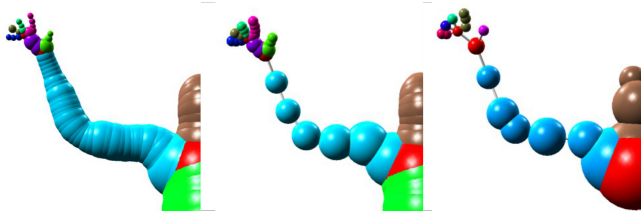


Figure 4: The original branch (left) composed by 50 nodes can be simplified by resampling (middle) or approximated with Douglas-Peucker algorithm (right). Both results are composed of 8 nodes.

4.4. Skeleton-wide operations

We here present a few operations that allow the user to manipulate the skeleton position inside the shape described by a mesh that has been loaded. These operations require an efficient implementation of intersections and distance queries computation, hence we decided to rely on the AABB tree implementation found in [32].

4.4.1. Fix the external nodes

When a mesh is loaded along with a skeleton it is possible to check if some of the skeleton's nodes are outside the mesh and move them inside. This can be useful since most of the skeletonization algorithms cannot guarantee that all the nodes are internal to the mesh; even the more robust ones as [11] with particular parameter settings can lead to branches crossing the mesh boundaries instead of flowing inside them.

For each node N_i we find its closest face of the mesh F with normal N_F. If N_i is outside the mesh we calculate its new position as

$$N_i - [d(N_i, F) + \epsilon] \cdot N_F$$

where d is the Euclidean distance and ϵ a small positive number.

4.4.2. Skeleton recentering

Centeredness is one of the desired properties of a curve-skeleton, which can be defined as the property of being medial to the shape a skeleton describes. As stated in [18], a skeleton is *perfectly centered* if it lies on the medial surface and it is centered with respect to it. *Perfect centeredness* is difficult to obtain, but it is also often undesirable due to the sensitivity of

the medial axis to small perturbations on the object's surface. In order to achieve *perfect centeredness* a posteriori, one would need to have at hand both the curve-skeleton and the shape's medial axis, while state-of-the-art algorithms do not provide such output. Moreover, several skeletonization algorithms do not compute the medial axis in their pipeline. Skeleton Lab implements three different algorithms for recovering the centeredness of the skeleton and estimating the radii of the maximal balls.

The presented methods rely on the definition of the local skeletal direction of each node N_i denoted with $\overrightarrow{N_i}$ and computed as:

$$\overrightarrow{N_i} = \begin{cases} \overrightarrow{N_j N_i} & \text{if } N_i \text{ is } \textbf{Ln} \\ \overrightarrow{N_i N_k} & \text{if } N_i \text{ is } \textbf{Bn} \\ \overrightarrow{N_j N_i} + \overrightarrow{N_i N_k} & \text{if } N_i \text{ is } \textbf{Jn} \end{cases}$$

where N_j and N_k are, respectively, the predecessor and the successor of N_i in the branch they belong to, choosing an arbitrary traversal order. The position of N_i together with the direction $\overrightarrow{N_i}$ defines a plane $\mathcal{P}_i$.

Iterative recentering

The first recentering algorithm relies on the description of the *approximate centeredness* proposed in [18]. For each skeletal node N_i we cast n uniformly distributed radial rays that lie on $\mathcal{P}_i$ and have origin in N_i, computing their intersections with the object surface. For each ray R_h we consider only the intersections obtained pinching the object from the inside and we store only the intersection I_h that is closest to N_i. We also discard all the pairs of opposite rays for which at least one valid intersection is not found.

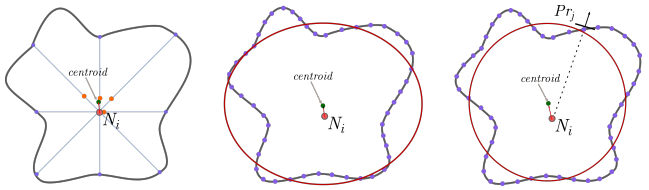


Figure 5: A section of the tubular branch where the node N_i is centered. On the left an application of iterative recentering; in the middle ellipse fitting; on the right SQEM for which only one of the tangent planes has been depicted.

For each pair of opposite rays (R_h, R_h') we calculate the midpoint C_h of the intersection pair (I_h, I_h'), then update the position of N_i as the centroid of all the computed midpoints C_h's and the new associated ball radius as the average semi-distance of all the intersection pairs. If N_i is a **Bn** we calculate its new position and radius, respectively, as the centroid of the positions and mean radius calculated separately for each incident branch. We observed that a single iteration of the procedure is usually not sufficient to obtain optimal results. Moreover, this procedure does not guarantee convergence, hence it is up to the user to iterate until the results are satisfactory.

Recentering using ellipse fitting

The great part of the shapes for which *makes sense* to compute a curve-skeleton, belong to the class of generalized cylinder assemblies. According to this assumption we can compute an approximated centered position for the skeletal nodes using an approach similar to the recentering procedure described in [14]. Each branch is traversed and for each node N_i, the intersection between its associated plane $\mathcal{P}_i$ and the object is computed. This intersection could generate different components, hence we choose the connected component that is closest to the node N_i. The selected component will be an arrangement of segments from which we compute a set of points R composed of all the segments' endpoints and midpoints. We then use the approach proposed [33] to fit an ellipse on the set of points R, which lie on both the surface and $\mathcal{P}_i$, updating the position of N_i to the center of the ellipse thus obtained. The position of **Bn**'s is computed applying the described procedure for each outgoing branch and computing the centroid of the center of the ellipses. The radii of the associated balls is approximated as the distance of each node from its closest point on the object's surface.

Recentering using Spherical Quadric Error Metric

The Spherical Quadric Error Metric (SQEM) has been introduced in [34] and allows to compute the sphere that optimally fits an input set of oriented planes. We use this metric to compute the new positions of the skeletal nodes and the radii of their associated balls. We follow an approach similar to the previous one. For each node N_i we compute the intersection of the object with $\mathcal{P}_i$ and the point set R. The point set R is filtered from the outliers, removing all points for which the ZScore is greater than 1.96 according to their distance from N_i. For each remaining point r_j in R we compute the direction $\overrightarrow{N_i r_j}$ which, together with r_j, will define a plane Pr_j. The set of all the planes Pr_j will be the input of the SQEM computation.

When N_i is a **Bn**, its new position is obtained computing the SQEM using as input the set of all the planes Pr_j obtained from the intersections computed for each outgoing direction of the **Bn** N_i.

4.4.3. Comparison of the recentering algorithms

All the three algorithms are able to improve the centeredness of the skeleton, however all of them have their own disadvantages. The iterative approach is particularly sensitive to the nodes position and local direction of the original skeleton, and the resulting paths are usually not smooth. Moreover, it is not able to deal with nodes that are outside the shape, hence their position needs to be fixed beforehand, as described in 4.4.1. Both ellipse fitting and SQEM based algorithms are more robust than the iterative procedure and they are able to produce good results even with nodes that are slightly outside the shape. By contrast they are not able to compute a reasonable position for those nodes lying at the branching parts of the shape. The ellipse fitting approach usually fails to reposition this kind of nodes, while SQEM provides reasonable results only for the shapes which are composed of only tubular parts (e.g., the *cactus*) and fails on the others (e.g., the *hand*). A comparison is shown in

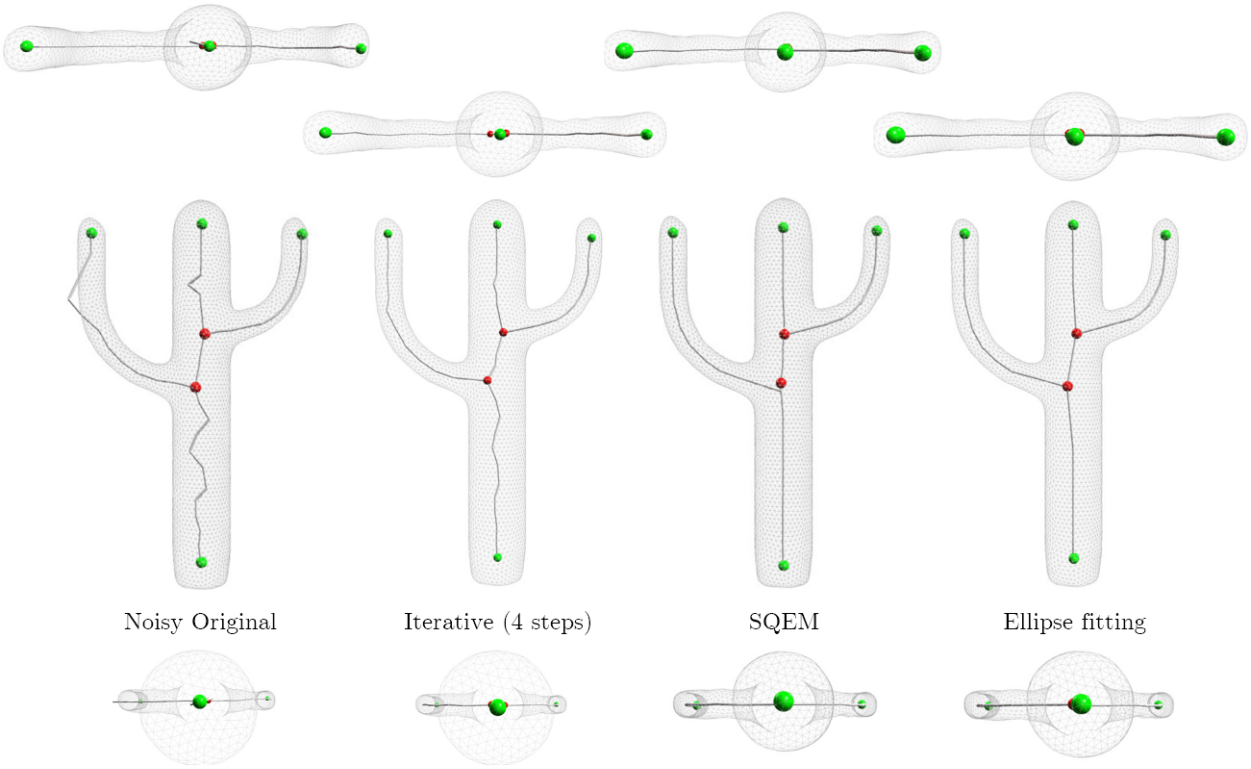


Figure 6: A comparison of the three recentering methods available in our tool. In the leftmost image we show a skeleton artificially deformed adding noise.

Figure 6. This problem arises due to the fact that the branching parts of a shape should be entirely represented by a single **Bn**, but skeletonization algorithms are usually not able to differentiate these parts. One possible strategy to alleviate this issue is to not perform the recentering procedure to all nodes at once, but use a two step approach. In the first step the positions of all the nodes that fall inside each **Bn**'s Zone of Influence can be kept unchanged. At the second step their new position can be computed using linear interpolation .

4.4.4. *Posing the skeleton*

While editing a skeleton, the user might need to move a set of nodes and at the same time preserving the proportion and the relative distance of the branch. This allows a user to manually create a curve-skeleton in a canonical pose that can be modified once the work is finished.

For this reason, the users are allowed to move a section of the branch from an **An** (or a **Bn**) to a **Ln**, while keeping the hierarchy intact, namely the distance and the relative position with the nodes in between.

To achieve this, the user selects an **An** or a **Ln**. Let us identify this node with N_i^b, where b identifies the branch and i is the position of the node in the branch (with N_0^b as the **Bn** and N_n^b as the **Ln**).

Then, the following set of nodes will be marked:

$$marked = \{N_j^b \mid 0 \leq j < l\}$$

where N_l^b is the next **An** from N_i^b toward the **Bn**. If there are no other **An**'s, then N_l^b will be N_0^b, namely the **Bn**. These marked nodes are then involved in a constrained rotation centered on N_l^b.

This method uses the **An**'s to pose just the desired portion of the branch (see an example in Figure 7). The **An**'s however, are not required in case of leaf-ending nodes: if there are no **An**'s, the entire branch will be posed.

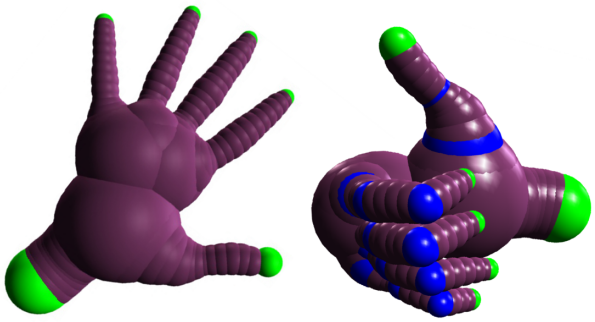


Figure 7: An example of how the user can pose a hand rotating the mesh around the **An**'s (in blue in the "thumb up" model).

5. User Evaluation

A user test has been carried out in order to evaluate the skeleton editing tool with regard to usability, performance, cognitive load and the overall experience. The aim was to get feedback on the current status of the tool and to collect suggestions for further development of its interface.

5.1. Test design

The test was organized in order to provide both quantitative and qualitative data on different aspects of the tool. Before starting the trial, the users were asked to read a short document describing SkeletonLab, together with a short help on how to activate the different tool functionalities. After reading the document, which can be found in the additional material, and providing some demographic information, the users were asked to execute different tasks, which we divided in two categories, namely A and B.

In the A category we included a set of eight introductory tasks. The tasks A1 to A7 required the users to test a single function of the tool, while task A8 is a warming-up activity before starting the evaluation on complex tasks. The aim of this group was to help the users to understand the capabilities of SkeletonLab and to prepare them for next phase. The tasks in the A category are the following:

A1 Creating a new node and changing its position.

A2 Changing the radius of the maximal ball associated to a node.

A3 Loading a skeleton from the disk and saving it into another file.

A4 Activating the branch mode, deleting a terminal node and the entire branch.

A5 Resampling a branch.

A6 Loading the *Cactus* mesh, its skeleton and aligning them [1]

A7 Deleting a branching node from a skeleton and transfer its links to another one.

A8 Modifying an existing skeleton to obtain a specific shape (the user was provided with the image of the desired configuration).

After each task, the user was asked to complete the Subjective Mental Effort Question (SMEQ)[35] for evaluating the task load. The evaluator registered the completion time.

After finishing the first group, the user was asked to complete a set of three additional tasks, which required the composition of different functionalities for reaching the desired configuration. Each task represent a typical use case for SkeletonLab. The users were provided with an image showing the desired output, to be considered as success criterion. The tasks in the B category are the following:

B1 Loading the mesh *Guy* and creating its skeleton.

B2 Cleaning the skeleton of the octopus as showed in figure 11 (reducing the number of nodes, deleting spurious branches etc.)

[1] Please note that task A6 is different from the recentering operator. Task A6 required the users to rigidly move the skeleton in order to align it with the mesh, while recentering independently moves each node of the skeleton in order to improve their centeredness. An example of task A6 can be seen in the accompanying video and at `https://youtu.be/iGhfPvxk_uc`

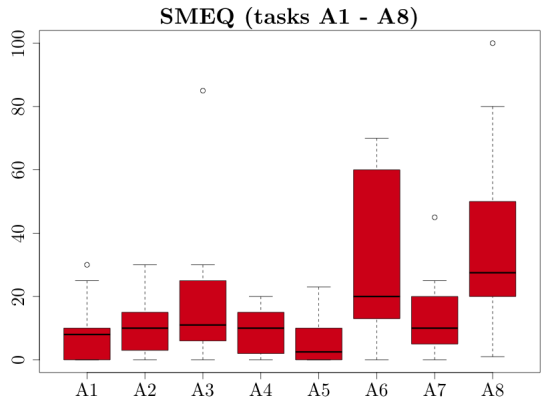


Figure 8: SMEQ ratings for group A tasks. The minimum value is 1, while the maximum is 150.

B3 Loading the skeleton of a hand and modifying it in order to get the thumb up pose (figure 7).

After finishing each task, the users were asked to fill the NASA-TLX questionnaire for assessing the task load [36] and the evaluator tracked the completion time. At the end of the test, the users were requested to fill the SUS [37] questionnaire in order to evaluate the overall usability. The test is composed of three open-ended questions, regarding which aspects they liked and disliked in the tool and improvement suggestions for the interface.

5.2. Test results

Fourteen users participated to the test, 11 males and 3 females, aged between 30 and 21 ($\bar{x}$ = 25.07, s = 2.8). Two of them have a high school degree, 3 a bachelor, 8 a master degree and one a PhD. They have an average experience in editing 3D meshes ($\bar{x}$ = 3.14, s = 1.61 in a 1-5 Likert scale), while they had little knowledge on creating and manipulating curve skeletons ($\bar{x}$ = 1.57, s = 0.85).

5.2.1. Introductory tasks

All users were able to complete the introductory tasks (type A). The box plot in figure 8 shows the SMEQ ratings. All the single function tasks required little effort for the users, who rated the tasks from A1 to A5 and A7 less than 13 on average. According to [35], all these tasks are *not very hard to do*. The difficulty of the other ones (A6 and A8) is between a *bit hard to do* and *fairly hard to do*.

We expected that modifying an existing skeleton to obtain a specific shape (A8) was the most difficult task. Indeed, it requires several operations for obtaining the desired result. We registered a high variability also for the recentering operation (A6), caused by the changes of the scene point of view needed for checking and modifying the alignment.

The analysis of the completion time (figure 9, red boxes) confirms the conclusions on SMEQ scores: completing task A6 and A8 takes longer than the other ones. We found a significant difference between the completion time of A8 and all the other tasks but A6. The difference was significant also between A6

Task A on Completion Time:
$F(3.96, 47.48) = 12.78$, $p = 10^{-11}$, $\eta^2_{partial} = 0.51$, $\epsilon = 0.56$

Pair	95% c.i.	p	Pair	95% c.i.	p
A6-A1	$[77s; 300s]$	.001	A8-A1	$[126s; 512s]$	.007
A6-A2	$[55s; 255s]$	.01	A8-A2	$[96s; 475s]$	.02
A6-A4	$[31s; 243s]$	.04	A8-A4	$[76s; 459s]$	.03
A6-A5	$[100s; 289s]$	.001	A8-A5	$[138s; 513s]$	.003
A6-A7	$[21s; 264s]$	.001	A8-A7	$[76s; 470s]$	.02

Table 1: Summary of the one-way ANOVA for repeated measures on the task completion time (type A). Since the repeated measures violated the sphericity assumption, we applied the Greenhouse-Geisser procedure [38] for correcting the degrees of freedom of the F-distribution (ϵ). We applied the Bonferroni correction for counteracting the multiple (pairwise) comparison problem [39]. We report only the significant differences ($p < .05$).

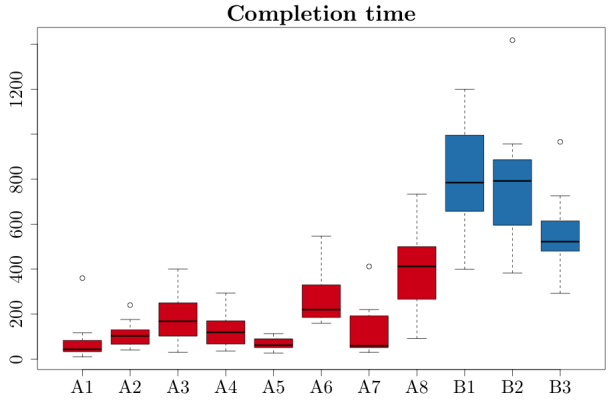


Figure 9: Completion time for task groups A (red) and B (blue) in seconds.

and the other tasks except A8 and A3. We report the details on the one-way ANOVA for repeated measures in table 1.

5.2.2. *Use cases*

The second part of the test included a set of more complex tasks (B1 to B3). The first task (creating a new skeleton) was completed by all users, two users abandoned B2 (cleaning a skeleton) and one B3 (modifying a skeleton pose). Users that abandoned the tasks shared the opinion that it was caused by the inability to figure out the sequence of actions for reaching the desired configuration. Considering that the experience level with the editing of curve skeletons was low for all users, this may be motivated with the need of a longer learning phase for solving complex tasks.

Figure 10 shows the results of NASA-TLX ratings for B1, B2 and B3. We report the overall difficulty evaluation (Tot) and the rating for each of the six factors considered in the questionnaire: the mental demand (MD), the physical demand (PD), the temporal demand (TD), the effort (Eff), the overall performance (Pr) and the frustration level (Fr). The task load is acceptable for all tasks, considering a scale from 1 to 100, the average values are between 33 and 43.

The analysis of each factor highlighted a significant effect of the considered task in the overall performance ratings. However, the pairwise comparison resulted only in a small difference between B1 and B2. The task has a significant impact on the

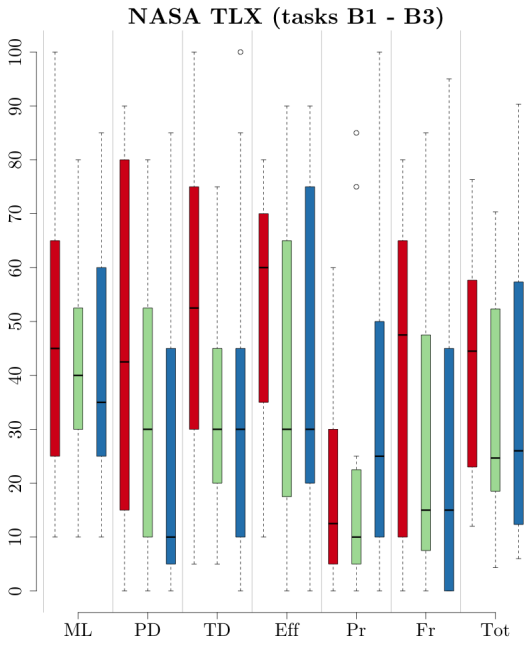


Figure 10: NASA-TLX ratings for group B1 (red), B2 (green) and B3 (blue). The results for the overall performance (Pr) have been reversed for consistency with the other dimensions (the lower, the better).

completion time, and in particular the difference in time between B1 and B3 is around 4 minutes. Additional details on the analysis are available in Table 2.

We conclude that users require a similar effort in all the envisioned use cases. They seem to be more efficient in changing the configuration of a skeleton rather than performing editing operations.

5.2.3. *Post test*

The SUS post test questionnaire shows that SkeletonLab has an average overall usability ($\bar{x} = 63.04$, $s = 16.35$). In particular, the users gave low ratings to question 3 (*I thought the system was easy to use*, $\bar{x} = 2.0$, $s = 0.92$) and question 7 (*I would*

Task B on Performance (Pr) ratings:
$F(1.32, 13.2) = 11.45$, $p = .003$, $\eta^2_{partial} = 0.055$, $\epsilon = 0.66$

Pair	95% c.i.	p
B1-B2	$[-28.8; 25.9]$	.007

Task B on Completion Time:
$F(1.32, 13.2) = 11.45$, $p = .003$, $\eta^2_{partial} = 0.055$

Pair	95% c.i.	p
B1-B3	$[15s; 480s]$	.08

Table 2: Summary of the one-way ANOVA for repeated measures for evaluating the effect of the task (type B) on the overall performance ratings (Pr in the NASA TLX questionnaire) and completion time. We applied the Greenhouse-Geisser correction [38] for sphericity on performance ratings and the Bonferroni correction [39] on the pairwise comparisons. We report the ANOVA results and we list the task pairs having a practically significant difference ($p < .1$) between them.

imagine that most people would learn to use this system very quickly, $\bar{x} = 2.0$, $s = 1.03$). The latter result may be explained again considering our users' low knowledge on curve skeletons.

We can explain the low ratings to question 3 through the answers to the open-ended questions: almost all users suggested to modify the keyboard shortcuts that are difficult to remember and they found some combinations difficult to be performed (especially the cmd+alt+shift ones). However, two users acknowledged that, once learned, the shortcuts allowed them to perform the operations quickly and they felt confident in using them. Therefore, we will consider to change some key combinations and to include buttons and an explicit graphical guidance for performing the basic operations on skeletons. We expect that, after the learning phase, most of the users will perform the manipulations through keyboard shortcuts.

The users found additional difficulties in changing the scene point of view while manipulating the skeleton, especially during the alignment operation. In this case we should consider to include predefined view camera positions (e.g. front, side and top) in order to help the users while aligning the skeleton to an existing mesh. Finally, the procedure for transferring the links to another branching node was not intuitive for them.

The users appreciated the node color feedback and the overall manipulation of the skeleton parts.

6. Results and Discussion

In this section we show how some automatically extracted curve-skeletons have been processed with our tool in order to be used as input in the pipeline presented in [6].

Figure 12 shows how the skeleton of the *Stanford Dragon* model has been edited in order to compute its quad layout. It has been one of the most challenging skeletons we have faced. The skeleton originally had 1050 nodes and, as can be seen in the close-up, some of the branches of the horns were disconnected. The skeleton has been simplified resampling most of its branches and removing 6 spurious ones, and it is now composed of 193 nodes. The complete editing process, from a) to c), took about 15 minutes.

Another example showing how some state-of-the-art methods, while successfully completing the skeletonization, may fail at conveying the correct semantic information can be seen in Figure 11. Since a human would usually describe an octopus as a living being composed of a big head and 8 tentacles starting from the bottom side of the head, we edited the skeleton removing all the spurious branches that was present inside the head and one of the tentacles. The original skeleton contained 23 branches while the result shown on the right of Figure 11 contained the 9 branches one would expect. The complete editing took less than 4 minutes to a trained user. It is also important to note that in this specific case, using the original skeleton the resulting quad-layout would have been completely wrong, while the edited version brought to an optimum. In our experiments the tool shown to be versatile enough to be used for both quick-fixes and complex editing, and for repairing tasks.

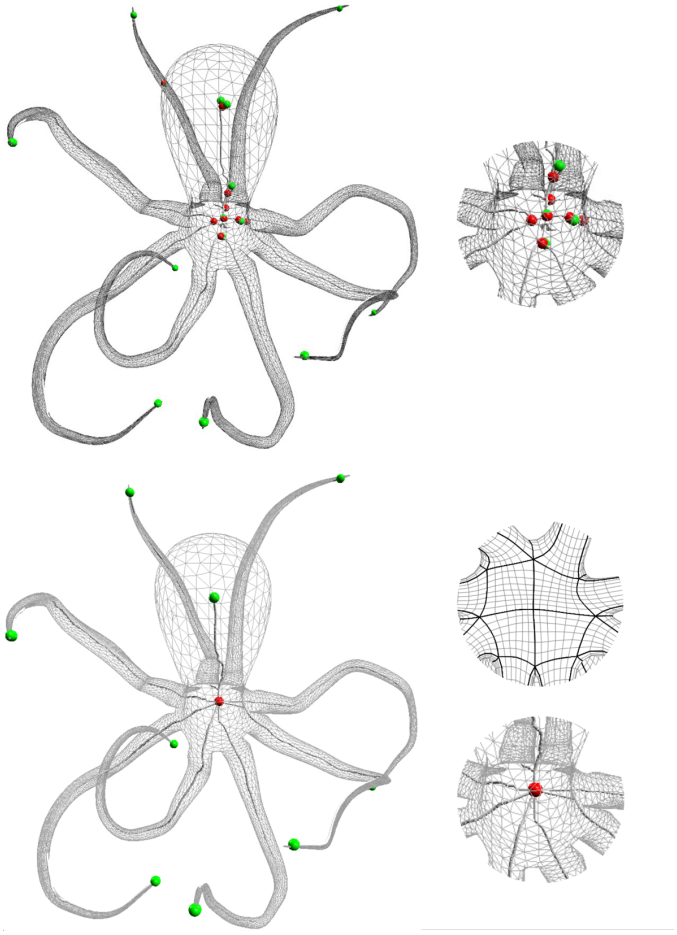


Figure 11: Skeleton of an octopus model extracted using [11] (top) and its edited version (down) with all the spurious branches collapsed. Close-ups on right side of the image show how the unnecessary **Bn** 's were removed in order to obtain the quad layout depicted on the lower right.

Our tool relies on the Qt Framework, libQGLviewer [40] and CGAL [32] , and it is able to load curve-skeletons produced by the available implementations of [11, 17, 21].

7. Concluding remarks

In this paper we presented a novel tool for the interactive processing of curve-skeletons in order to make them fit the application in which they will be used. We observed that the evaluation of what is a good curve-skeleton is strictly dependent to the application in which it will be used and none of the existing skeletonization methods provide curve-skeletons that can be directly used in every application bringing to optimal results. This is not a defect of the proposed approaches. The topological and semantic information that a skeleton encodes are not easy to capture algorithmically, and fine-tuning the parameters required by the skeletonization methods is often necessary but not resolutive. While developing the method presented in [6] we have observed that an effective and practical approach to obtain optimal results in our pipeline, was to automatically extract the curve-skeleton (choosing the method we experienced

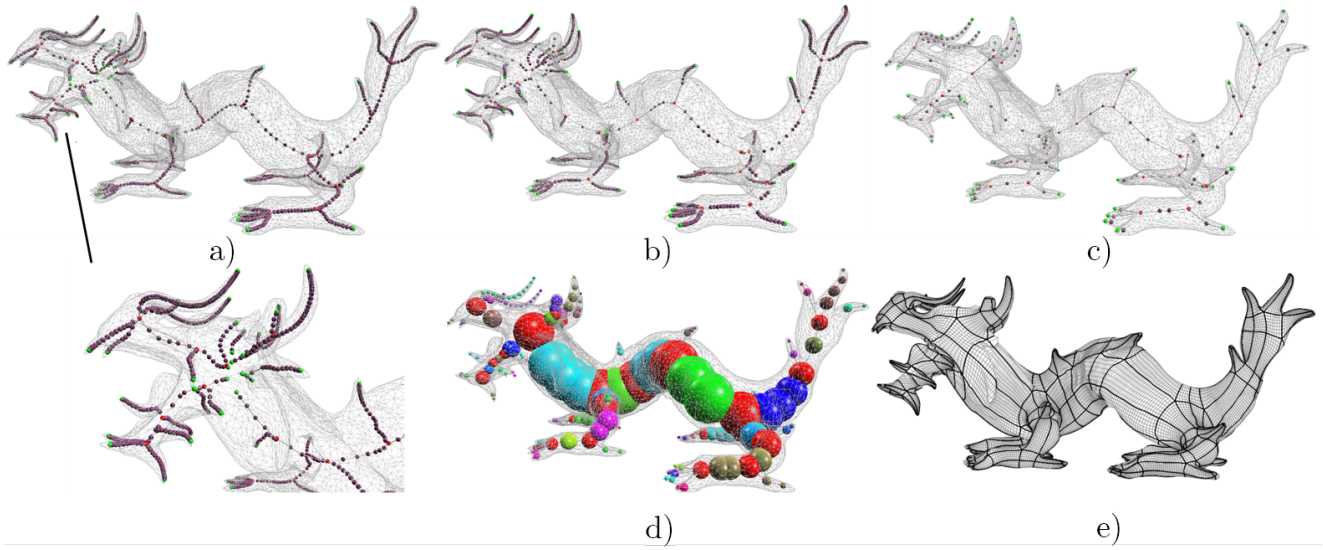


Figure 12: Editing process of a complex skeleton extracted with [17], a) has been computed with a number of disconnected branches (close-up) due to a buggy implementation or erroneous parameter setting. The skeleton was first reconnected b), and simplified c). In the second row we show the skeleton with its associated maximal balls d) and the quad-layout obtained with it e).

to be the best suited for the considered shape using standard parameter values) and process it with the presented tool in order to optimally fit our pipeline requirements. The tool has also been made publicly available both as source code and binaries in order to allow other researchers and practitioners to use it in their applicative scenarios.

Future work

Interesting directions for future work include the possibility to extend Skeleton Lab implementing a modified version of the method presented in [6] in order to be used as an inverse skeletonization tool, providing high quality base quad meshes with a low and controllable number of irregular vertices. Moreover it can be extended in order to produce rigged meshes. Loading a mesh and a skeleton (either automatically extracted or handcrafted), the latter can be edited according to the user's needs, decimated retaining only **Bn**'s, **Ln**'s and **An**'s. The rigging weights can then be computed in order to obtain a rigged mesh ready to be used for animation purposes.

[1] Blum H. A Transformation for Extracting New Descriptors of Shape. In: Models for the Perception of Speech and Visual Form. 1967, p. 362–380.

[2] Tagliasacchi A. Skeletal Representations and Applications. arXiv preprint arXiv:13016809 2013;abs/1301.6809.

[3] Baran I, Popović J. Automatic Rigging and Animation of 3D Characters. ACM Trans Graph 2007;26(3).

[4] Hilaga M, Shinagawa Y, Kohmura T, Kunii TL. Topology Matching for Fully Automatic Similarity Estimation of 3D Shapes. In: Proceedings of SIGGRAPH '01. 2001, p. 203–212.

[5] Bærentzen JA, Abdrashitov R, Singh K. Interactive Shape Modeling Using a Skeleton-mesh Co-representation. ACM Trans Graph 2014;33(4):132:1–132:10.

[6] Usai F, Livesu M, Puppo E, Tarini M, Scateni R. Extraction of the Quad Layout of a Triangle Mesh Guided by Its Curve Skeleton. ACM Trans Graph 2015;35(1):6:1–6:13.

[7] Liu L, Zhang Y, Liu Y, Wang W. Feature-preserving T-mesh construction using skeleton-based polycubes. Computer-Aided Design 2015;58:162–172.

[8] Zhang Y, Bazilevs Y, Goswami S, Bajaj CL, Hughes TJ. Patient-specific vascular {NURBS} modeling for isogeometric analysis of blood flow. Computer Methods in Applied Mechanics and Engineering 2007;196(29–30):2943–2959.

[9] Abeysinghe SS, Ju T. Interactive skeletonization of intensity volumes. The Visual Computer 2009;25(5-7):627–635.

[10] Barbieri S, Meloni P, Usai F, Scateni R. Skeleton Lab: an Interactive Tool to Create, Edit, and Repair Curve-Skeletons. In: Smart Tools and Apps for Graphics - Eurographics Italian Chapter Conference. 2015, p. 121–128.

[11] Tagliasacchi A, Alhashim I, Olson M, Zhang H. Mean Curvature Skeletons. Comp Graph Forum 2012;31(5):1735–1744.

[12] Jalba A, Sobiecki A, Telea AC. An Unified Multiscale Framework for Planar, Surface, and Curve Skeletonization. Pattern Analysis and Machine Intelligence, IEEE Transactions on 2016;38(1):30–45.

[13] Tagliasacchi A, Zhang H, Cohen-Or D. Curve Skeleton Extraction from Incomplete Point Cloud. ACM Trans Graph 2009;28(3):71:1–71:9.

[14] Huang H, Wu S, Cohen-Or D, Gong M, Zhang H, Li G, et al. L1-medial Skeleton of Point Cloud. ACM Trans Graph 2013;32(4):65:1–65:8.

[15] Sobiecki A, Jalba A, Telea A. Comparison of curve and surface skeletonization methods for voxel shapes. Pattern Recognition Letters 2014;47:147–156.

[16] Serino L, Sanniti di Baja G, Arcelli C. Distance-Driven Skeletonization in Voxel Images. IEEE Transactions on Pattern Analysis and Machine Intelligence 2011;33(4):709–20.

[17] Dey TK, Sun J. Defining and Computing Curve-skeletons with Medial Geodesic Function. In: Proceedings of SGP '06. 2006, p. 143–152.

[18] Cornea ND, Silver D, Min P. Curve-Skeleton Properties, Applications, and Algorithms. IEEE Transactions on Visualization and Computer Graphics 2007;13(3):530–548.

[19] Sobiecki A, Yasan HC, Jalba AC, Telea AC. Qualitative Comparison of Contraction-Based Curve Skeletonization Methods. In: Mathematical Morphology and Its Applications to Signal and Image Processing; vol. 7883 of *Lecture Notes in Computer Science*. 2013, p. 425–439.

[20] Livesu M, Guggeri F, Scateni R. Reconstructing the Curve-Skeletons of 3D Shapes Using the Visual Hull. Visualization and Computer Graphics, IEEE Transactions on 2012;18(11):1891–1901.

[21] Livesu M, Scateni R. Extracting curve-skeletons from digital shapes using occluding contours. The Visual Computer 2013;29(9):907–916.

[22] Kustra J, Jalba A, Telea A. Probabilistic View-based 3D Curve Skeleton Computation on the GPU. In: Proceedings of the VISAPP '13. 2013, p. 237–246.

[23] Au OKC, Tai CL, Chu HK, Cohen-Or D, Lee TY. Skeleton Extraction by Mesh Contraction. ACM Trans Graph 2008;27(3):44:1–44:10.

[24] Bærentzen JA, Misztal MK, Wełnicka K. Converting skeletal structures to quad dominant meshes. Computers & Graphics 2012;36(5):555 – 561.
[25] Ji Z, Liu L, Wang Y. B-Mesh: A Modeling System for Base Meshes of 3D Articulated Shapes. Comp Graph Forum 2010;29(7):2169–2178.
[26] Hijazi Y, Bechmann D, Cazier D, Kern C, Thery S. Fully-automatic Branching Reconstruction Algorithm: Application to Vascular Trees. In: Proceedings of SMI '10. 2010, p. 221–225.
[27] AutoDesk. 123D. `http://www.123dapp.com/`; 2016. [Online; accessed 29 January 2016].
[28] ZBrush. ZSpheres. `http://pixologic.com/zbrush/features/zspheres/`; 2016. [Online; accessed 29 January 2016].
[29] Serino L, Sanniti di Baja G, Arcelli C. Object Decomposition Via Curvilinear Skeleton Partition. In: Pattern Recognition (ICPR), 2010 20th International Conference on. 2010, p. 4081–4.
[30] Heckbert PS. Bilinear Coons Patch Image Warping. In: Graphics gems IV. 1994, p. 438–446.
[31] Douglas DH, Peucker TK. Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. International Journal for Geographic Information and Geovisualization 1973;10(2):112–122.
[32] Cgal. computational geometry algorithms library. `http://www.cgal.org`; 2016. [Online; accessed 29 January 2016].
[33] Fitzgibbon AW, Pilu M, Fisher RB. Direct least-squares fitting of ellipses. Pattern Analysis and Machine Intelligence, IEEE Transactions on 1999;21(5):476–480.
[34] Thiery JM, Guy E, Boubekeur T. Sphere-Meshes: Shape Approximation using Spherical Quadric Error Metrics. ACM Trans Graph 2013;32(6):178:1–178:12.
[35] Zijlstra FRH, van Doorn L. The Construction of a Scale to Measure Subjective Effort. Tech. Rep.; Delft University of Technology, Department of Philosophy and Social Sciences; Delft, Netherlands; 1985.
[36] Hart SG, Staveland LE. Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research. In: Human Mental Workload; vol. 52 of *Advances in Psychology*. 1988, p. 139 – 183.
[37] Brooke J. SUS: A "quick and dirty" usability scale. In: Usability Evaluation in Industry. 1996, p. 189–194.
[38] Geisser S, Greenhouse SW. An Extension of Box's Results on the Use of the F Distribution in Multivariate Analysis. The Annals of Mathematical Statistics 1958;29(3):885–91.
[39] Dunn OJ. Multiple Comparisons among Means. Journal of the American Statistical Association 1961;56(293):52–64.
[40] libQGLViewer. `http://libqglviewer.com/`; 2015. [Online; accessed 29 January 2016].