UNIVERSITÀ DEGLI STUDI DI CAGLIARI

Facoltà di Scienze

Corso di Laurea Magistrale in Informatica

Skeleton Editing and Mesh Reconstruction from Skeleton

Supervisor
Prof. Riccardo Scateni

Candidates
Simone Barbieri

Anno Accademico 2013/2014

Sommario

Uno scheletro è un insieme di nodi interconnessi tra loro in una struttura dati chiamata grafo ed è uno strumento fondamentale per molti campi della computer grafica. Creare e modificare scheletri complessi, ad oggi, richiede lo studio di strumenti particolarmente complicati.

L'editor di scheletri che è stato sviluppato permette la modifica di scheletri e la generazione di una mesh tridimensionale a partire da esso. Gli scheletri possono essere preesistenti o creati interamente attraverso il programma sviluppato.

Lo scopo finale del progetto è avere uno strumento affidabile per la manipolazione di scheletri e la scheletonizzazione inversa, in quanto non sono disponibili, al momento della stesura della corrente tesi, strumenti open source e di facile utilizzo che lo permettano.

Abstract

A skeleton is a set of nodes that are interconnected and arranged in a graph data structure and it is an essential instrument in several applications of computer graphic. Nowadays, creating and editing complex skeletons requires the study of intricate and sophisticate tools.

The newly developed skeleton editor allows editing of skeleton itself and a sequent generation of a tridimensional mesh. Skeletons can be already built or created using the developed program.

The final purpose of the project is the fruibility of a reliable instrument for skeleton manipulation and inverse skeletonization, since right now there are no available simple open source tools that allow it.

Contents

Chapter 1

Introduction

In recent years, the use of high details models is escalating swiftly both in the field of animations and videogames. Eelke Folmer claims that investments, especially in the videogames field, are increasingly high, and they can exceed tens of million of dollars [9]. Besides, he claims that the majority of the cost to videogame development is spent in art and animation. As a consequence, a way to curb the cost is to make the artist more efficient, through better animation and modeling tools.

In this thesis, we developed a software which allow the creation and the editing of skeletons, and generate a three-dimensional mesh directly from it. It is also possible to load existing mesh to assist the users in the creation of skeleton.

Below we will introduce some basic concepts needed to the properly understanding of the thesis.

A *skeleton* is a thin representation of a shape which is equidistant from its boundaries. Skeleton give prominence to geometrical and topological properties such as connectivity, topology, length and width.

In the plane, skeletons are better defined as *medial axis*. The medial axis of a planar shape is the locus of centres of its maximals inscribed disks [8]. The medial axis can be easily extended to three dimensions, but it becomes a surface, while we need a mono-dimensional representation.

Representing three-dimensional objects with mono-dimensional skeletons is a complex and important problem in computer science. These representations are called *curve-skeletons* and are used in a comprehensive assortment of applications,

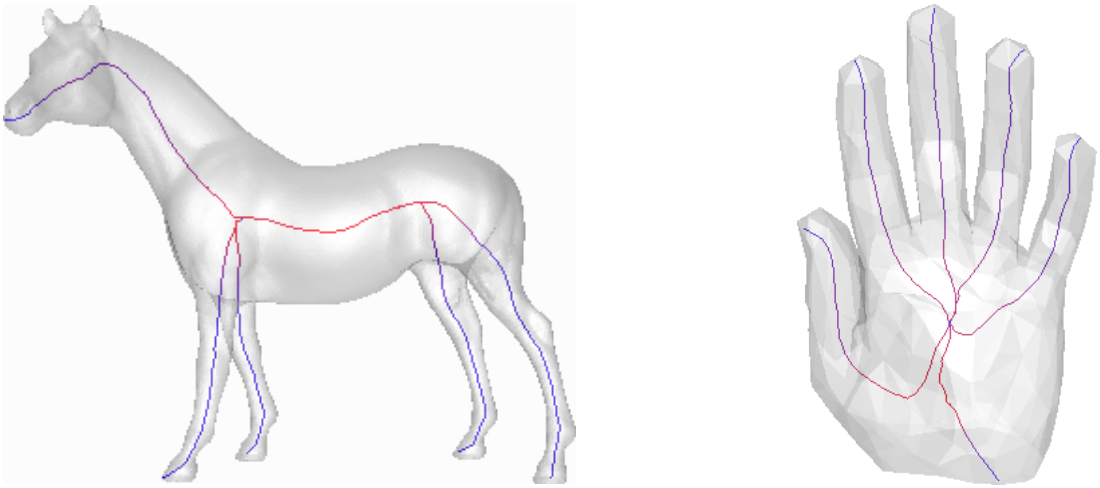


Figure 1.1: Examples of curve-skeletons.

for instance character animations, shape morphing, shape recognition, shape retrieval, etc. However, curve-skeletons are ill-defined objects. In fact, scientific literature introduce a large amount of extraction algorithms and heuristics which use different definitions and parameters to denote curve-skeletons.

In this thesis, we will represent curve-skeletons as three-dimensional graphs.

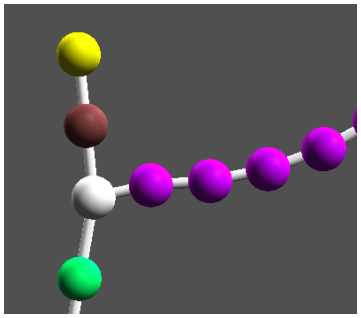
A *graph* is an ordered pair $G = (V, E)$ comprising a set V of vertices, or *nodes*, and a set E of edges, or *arcs*, which are pairs of a subset of V. An arc is a relation of two nodes, and this relation is represented as an unordered pair of the nodes.

There are several types of graphs in literature, but the kind of graph which we will use is a *connected undirected graph*. A graph is called *connected* if every pair of distinct nodes in the graph is connected (two nodes are called connected if the graph contains a path between the two nodes). Moreover, a graph is called *undirected* if the arcs have no orientation. This means that an arc between the node a and the node b is identical to the arc between the node b and the node a.

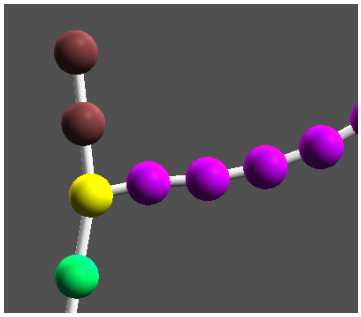
The nodes, as can be seen on figure 1.2, can be of three different types:

- *leaf* node: a leaf node is a node with only one connection.
- *joint* node: a joint node is a node with two connections.
- *branching* node: a branching node is a node with three or more connections.

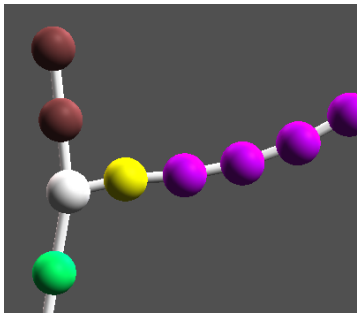
A *bone* is a set of nodes in the same path between two branching nodes or between a branching node and a leaf.



(a) A leaf node



(b) A branching node



(c) A joint node

Figure 1.2: Examples of nodes.

There are two more definitions of nodes. A node is called *articulation* if is a node that define an articulation point inside the bone or a point in which the shape keeps prefixed angles, such as the heel in a human being, the head in a bird, etc. An articulation node will not be changed by operations that operate with multiple nodes, such as the re-sampling of a bone. A node is called *feature point* if it is a leaf node, a branching node or an articulation node.

Finally, we introduce the concept of mesh. A *mesh* is a collection of vertices, edges and faces which defines the shape of a polyhedral object in three-dimensional computer graphics and modeling. The faces conventionally consist of triangles, quadrilaterals or other simple polygons, in order to simplify rendering.

Despite there are several algorithms to automatically extract the skeleton from a three-dimensional model, these are not perfect. Often the user must find a compromise between having a high resolution skeleton or having a consistent topology with the three-dimensional shape. For instance, if in a horse model the user wants details in the skeleton, such as horse's ears, probably he will get branching nodes inside the body which are not part of the structure. An other example is the human hand, in which there is just one branching node, while the correct structure should have a branching node that connect just the thumb and the wrist, and another one to connect the other four fingers.

Therefore, having a simple instrument to edit skeletons and swiftly solve these situations is useful indeed.

In chapter 2 the state of the art is described. The explanation of the skeleton editor and its functions is available in a more detailed process in chapter 3, while

in chapter 4 is explained the mesh generator. At last, in chapter 5 we will show the reasoning derived from our work and future work that can improve the software.

Chapter 2

State of the Art

As said in the introduction, the purpose of this thesis is creating an easy-to-use and economic instrument for the creation and editing of skeletons and, from them, the automatic generation of a three-dimensional mesh. Anyway, there already exist other skeleton editors in literature, as well as there are also mesh generators from skeletons. Usually, the softwares that create mesh with the inverse skeletonization, generate quadrilateral meshes, because the 3D modelist generally use them, in spite of triangular meshes.

2.1 Origin of the idea

2.1.1 Solidifying Wireframes

The mesh reconstruction from a skeleton, better defined as *inverse skeletonization*, is a relatively recent idea. One of the first works in this field is from Vinod Srinivasan, Esan Mandal and Ergun Akleman, in which they explain how to solid wireframe [12].

This work does not treat skeletons, as there was not yet this field of study, but considers wire-frames. A *wire-frame* model is a visual presentation of a three-dimensional object often used in computer graphics. It is created by stating the edges of the object where two continuous smooth surfaces connect, or by linking object's vertices using straight lines or curves. The object is projected by drawing lines at the location of each edges.

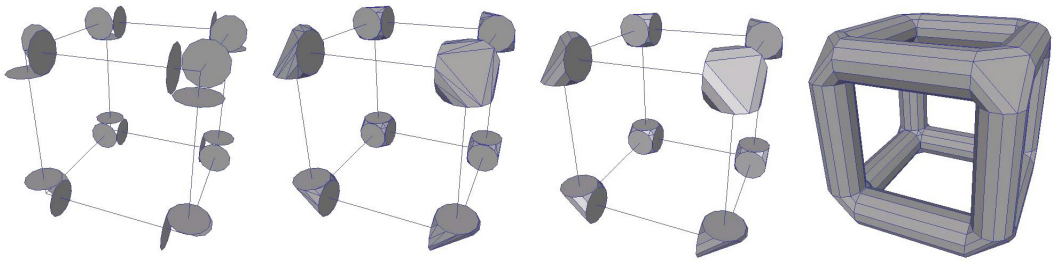


Figure 2.1: Creating a solid wireframe from a cube.

The first step of the process to solidify the wireframes is to create three-dimensional joints at every vertex of the input wireframe. The shape of the three-dimensional joint depends on the number of edges incident on that vertex and the desired thickness and cross-section of the three-dimensional pipes. There are three steps to create the joints.

The first step is the computation of the end-faces. A three-dimensional pipe is with the thickness and the cross-section selected from the user is calculated, and its axis is aligned with the edge. It is also established a link between each vertex of the wireframe and the end-faces that will be used to build the three-dimensional joint at that vertex. The centroid of the end-faces lies on the edge, with the normal aligned with the edge itself. All the end-faces linked to a vertex are positioned at the same distance from it. The distance is calculated to avoid intersections with other end-faces.

In the second step a convex hull is computed using the vertices of the end-faces of the three-dimensional joint. This step defines the shape of the joints. Since in the first step the algorithm avoided intersections between the end-faces, all their vertices will be part of the convex hull.

The third step will clean the created convex hull. The convex hull create a triangulated mesh, but it is necessary to clean the mesh to have a face on each end-face deleting the edges that were not part of that end-face. The mesh can be cleaned further deleting edges between connecte co-planar faces.

Once the three-dimensional joints are created, the generation of the three-dimensional pipes is almost straightforward, including the creation of a handle between matching faces of the joints.

2.2 Successive works

This publication was one of the first which treated the problem of the inverse skeletonization. Nowadays, there are two different approaches to inverse skeletonization:

- models that create pipe-like regions around the leaf nodes and connect them when they converge on branching nodes;
- models that create polyhedron on branching nodes and then extrude it to reach the leaf nodes.

These two approaches can be synthesized in deciding when handle the complex part of the work, that is managing the branching nodes.

2.2.1 B-Mesh

One modeling system that observe the first approach is B-Mesh [10]. B-Mesh gives the possibility to create the skeleton and from it, generate a quad-mesh. It has the characteristic of creating meshes with few singularities (verticed with valency greater than four) and the generated meshes are similar to those manually created.

B-Mesh works with two fundamental operations: sweeping and stitching. The sweeping is the operation that generate the mesh parts that covers the bones that ends with a leaf. Starting from the branching node, the initial cross section is extruded and translated towards the skeleton limbs and rotated around the joint nodes. This operation is executed for each bone starting from the branching nodes.

The stitching generate the mesh part that surround the branching nodes. At each branching nodes, it is constructed the convex hull around it and the convex

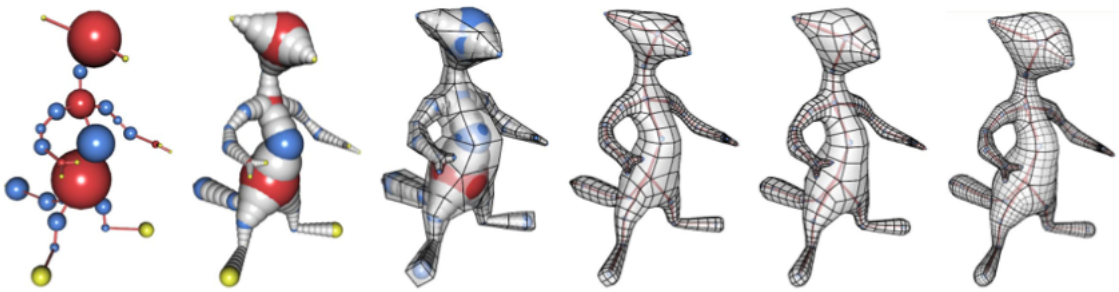


Figure 2.2: B-Mesh modeling approach.

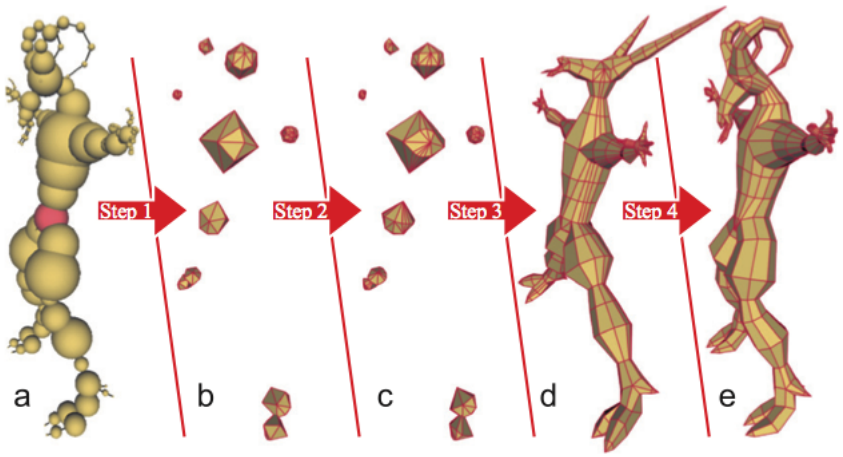


Figure 2.3: SQM modeling approach.

hull is triangulated. Then two adjacent triangles are merged into a quadrangle in a descending order of a score between them. The score is defined as

$$score = A(n_1 \cdot n_2)$$

where A is the sum of the area of the two triangles, n_1 and n_2 are their unit normals respectively.

B-Mesh is a good instrument, but sometimes in branching nodes remain some triangular faces.

2.2.2 Skeleton to Quad-dominant polygonal Mesh

Instead, an algorithm that follows the second approach is *Skeleton to Quad-dominant polygonal Mesh* (SQM) [7]. The SQM algorithm is based on the concept of *polar-annular meshes* [6].

A polar-annular mesh is a mesh composed only of triangles and quadrilaterals such that:

- each quadrilateral belongs to a single ring of quadrilaterals denoted as annular region;
- each triangle belongs to a single fan of triangles denoted as polar region;
- the faces incident on a vertex form a region homeomorphic to a disc, and

vertices may not be incident on edges.

The algorithm has four different steps, shown in figure 2.3. The first step is the creation of the branch node polyhedra (BNP). BNPs are polyhedron, built on branching nodes, whose vertices correspond to the paths incidents on the node.

The second step takes the pairs of BNP connected by a path and create a tube between them, and removing the path nodes and their incident triangles and fill the holes with tubular meshes. It can happen that the path vertices have different valencies, so it is impossible to create a tube of quadrilaterals only. In this case, the algorithm split the edges in the link of the vertex with lower valency, in order to have the same valency on both the vertices.

The third part of the algorithm can actually remove the incident triangles on the vertices connected by a path and bridge the holes with a tube composed by quadrilaterals only, since in the second step all valencies incongruencies are removed. In this step, all faces of the original BSP are removed, except those incident on path vertices corresponding to lead nodes. These faces remains triangles and form a polar region, while the faces replaced by quadrilaterals only tube are the annular region. The vertices are temporary moved to better connect the bones.

The last step execute some refinement operations, including restore vertices positions.

2.3 Other mesh generator softwares

2.3.1 Z-Sphere

Z-Brush is a digital sculpture tool that combines three-dimensional and two-and-a-half-dimensional modeling, texturing and painting. Z-Brush is used for creating high-resolution models for use in movies, games and animations by companies such as Electronic Arts [3].

Z-Brush was developed by the company *Pixologic Inc.* It was presented in 1999 at SIGGRAPH. The latest version of the software, the forth, offers integration with the most important 3D computer graphics applications, such as *Autodesk Maya* [1] and *Cinema 4D* [2].

Figure 2.4: An example of skeleton created with Z-Sphere and the mesh edited with Z-Brush.

Z-Brush is composed by a number of tools, among which Z-Spheres. *Z-Spheres* is the tool that allow Z-Brush users to create their skeletons from just one sphere, called, of course, Z-Sphere. When the basic shape is created, the user can create a base mesh with uniform topology and convert it into a sculptable model.

The basic operations allowed on the nodes are translation, rotation, scaling and the creation of a new sphere connected to the previous one. Z-Sphere, such as B-Mesh, has a mirroring feature that consent the user to create two different symmetric structures.

2.3.2 123D Creatures

Another software that allow to build skeletons and generate a mesh is 123D Creature. 123D Creature is a part of *Autodesk 123D*, a suite of CAD and three-dimensional modeling tools developed by Autodesk in 2009.

This software allow the user to create in an easy way a mesh that can later be printed. It permit the user to load a mesh created from other users too, in order to print it. A big limit of this application is that the generated meshes cannot used for

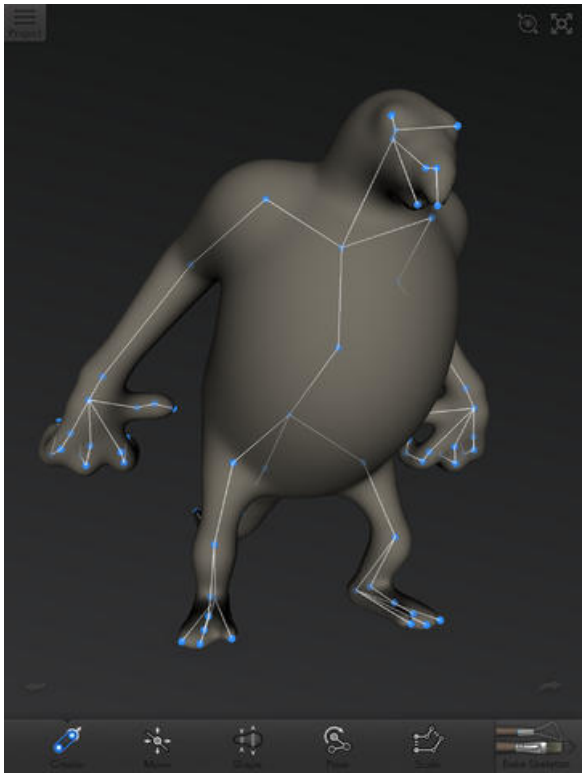


Figure 2.5: A view of 123D Creature's interface.

other purposes that is not printing.

123D Creature is the only skeleton editor and mesh generator accessible on mobile devices only. In fact, it is available only on Apple iPad.

Chapter 3

The Skeleton Editor

One of the purpose of the current thesis was to develop a reliable program that could create and edit skeleton. The editor had to be capable of a number of functions as the already existing skeleton editors lack of a large number of useful features. In this chapter, we will going to explain all the features included in the software and an explanation of the reasons that brought us to include that feature. Next, there will be an overview of the used technologies and finally a comment on a failed approach to integrate our work with an existing environment.

3.1 Skeleton Editor

As explain in a more detailed way in paragraph 3.3, the editor was thought as a plug-in of the comprehensively employed software *Meshlab*. Anyway, its limitations brought us to develop an entire environment. In this way, we were able to develop the editor without from any limitation.

3.1.1 Interface

The main interface of the application is composed of two parts, the viewer and the control panel.

The viewer is the main component of the interface. From it is possible view the skeleton currently in editing (it is possible view also a mesh, if the user loads one), change the camera, zoom in and out, and select nodes of the skeleton, in order to

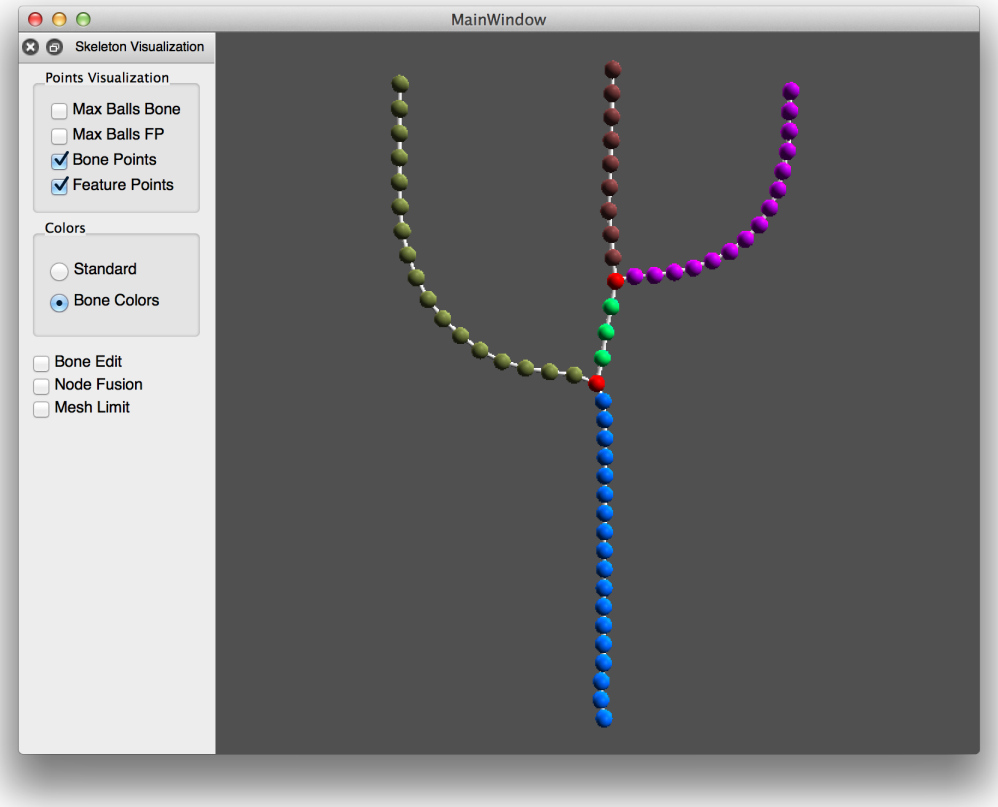


Figure 3.1: The main interface of the skeleton editor.

execute an operation on them.

The control panel is a useful console which allow the user to quickly change interface and functional settings. It is formed of three parts.

The first part take into account the nodes visualization. There are four different node visualization options that can be active at the same time. The first two allow the user to show the skeleton's nodes with their properly dimension or with a standard dimension. Respectively, the two checkboxes are for the feature points and for bone points. The standard dimension of the nodes can be changed by the user. The other two checkboxes let the user show the features points or the bone nodes.

The second part regards the nodes colors. There are two different kind of colors. The first one is the *standard* color. With the standard colouration, all the bone nodes are white, while the branching nodes are red, the leaf nodes are green and

the articulation nodes are blue. The second is the *bone* colouration. With the bone color, the nodes are coloured depending on the bone they belong with the exception of the branching nodes, that are red.

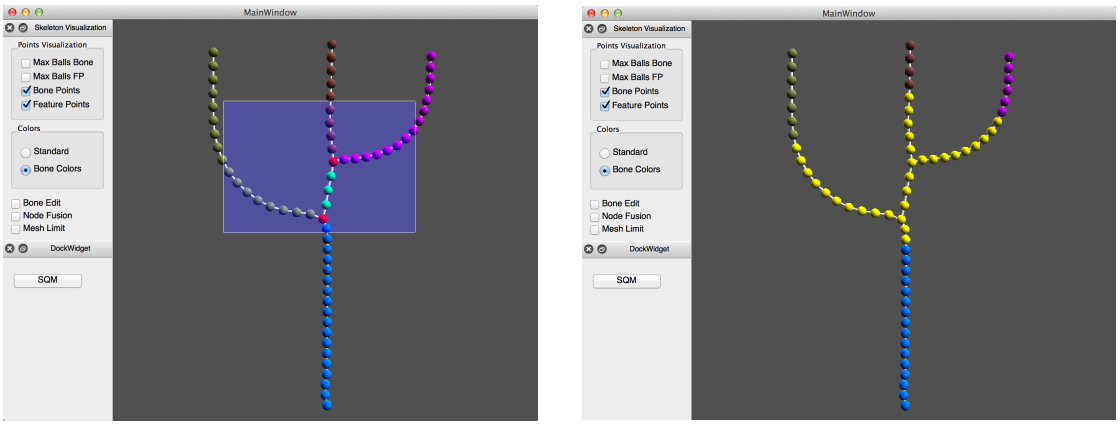
The third part consists of some options for the editing of the skeleton. The first one activate the bone edit mode. The bone edit mode allow the user to use operations bone-specific, such as the pruning and the re-sampling. The bone edit mode is better explained in the paragraph 3.1.9. The second checkbox is a simple options that allow user to enable or disable the node fusion when he moves two nodes such that their maximal spheres have an intersection bigger than a given threshold. Finally, the third checkbox allow the user to enable or disable the constraint that the nodes must stay inside the mesh, if there is a loaded mesh.

To help the user with the commands, it is possible in any moment to open a window with all the commands and a brief description for each one.

3.1.2 Selection and de-selection

The selection and the de-selection of the nodes is one of the fundamental feature of an editor. We included two types of selection and de-selection. The first one is the multiple one. It allow the user to select or deselect one or more nodes at a time.

The second kind of selection is the precise one. With this type of selection, the user can select in a more precise way only one node at a time.



(a) Before the selection. (b) After the selection.

Figure 3.2: Examples of multiple selection.

It is important to notice that the selection is not exclusive but *incremental*. This means that if the user selects one node, and then he selects another one, both nodes will result selected. To deselect the nodes, the user have to unselect them manually or deselect all with a dedicate command.

3.1.3 Translation, Rotation and Scaling

The translation, the rotation and the scaling of the nodes are important features for the editing of the skeleton, and represent the basic operations for its modification.

To define the translation in the space we use the matrix notation. If we have a point in the coordinates (x, y, z), and we translate it adding a vector $< p, q, r >$, the resulting position of the point will be:

$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & p \\ 0 & 1 & 0 & q \\ 0 & 0 & 1 & r \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

The translation of a node can be done in a different way for the joint nodes. In fact, for this kind on nodes, we implemented a method to move it in the direction of the two edges incident on it.

For the rotation, we use the quaternions. Quaternions are a formal object of this kind:

$$a + bi + cj + dk$$

where a, b, c, d are real numbers and $\mathbf{i}, \mathbf{j}, \mathbf{k}$ are lecteral symbols that observe the following rule:

$$i^2 = j^2 = k^2 = ijk = -1$$

In three-dimensional space, any rotation of a coordinate system about a fixed point is equivalent to a single rotation by a given angle θ about a fixed axis (called *Euler axis*), that runs through the fixed point. The Euler axis is usually represented by a versor $\overrightarrow{u}$. Consequently, any rotation in three-dimensions can be described as

a combination of a versor $\overrightarrow{u}$ and a scalar θ.

An Euclidean vector such as $(2, 3, 4)$ can be rewritten as $(2\mathbf{i}, 3\mathbf{j}, 4\mathbf{k})$, where $\mathbf{i}, \mathbf{j}, \mathbf{k}$ represent the three Cartesian axes. A rotation through an angle θ, obtained from a comparing between the previous mouse position and the new one, around the axis defined by a versor

$$\overrightarrow{u} = (u_x, u_y, u_z) = u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k}$$

can be represented by a quaternion. This can be performed employing an extension of Euler's formula:

$$\mathbf{q} = e^{\frac{\theta}{2}(u_x\mathbf{i}+u_y\mathbf{j}+u_z\mathbf{k})} = \cos\frac{\theta}{2} + (u_x\mathbf{i} + u_y\mathbf{j} + u_z\mathbf{k})\sin\frac{\theta}{2}$$

The desired rotation can be applied to an ordinary vector $\mathbf{p} = (p_x, p_y, p_z) = p_x\mathbf{i} + p_y\mathbf{j} + p_z\mathbf{k}$ in three-dimensional space by evaluating the conjugation of p by q:

$$\mathbf{p}' = \mathbf{q}\mathbf{p}\mathbf{q}^{-1}$$

using the Hamilton product, where $\mathbf{p}' = (p'_x, p'_y, p'_z)$ is the new position vector of the point after the rotation.

The scaling is an important information of the nodes, especially if the skeleton will be used to generate a mesh. For this reason, we implemented the scaling of any node with a simple movement of the wheel of the mouse.

3.1.4 Adding new nodes

Since adding a new node is one of the most common action that the user will use while using the application, we wanted to make it available in the simpler way possible. So, the standard way to add a new node is select the starting node and then drag the node with the right button. A new node will come out from the original node and, if the new node is on an allowed positions, it will be created.

To realize this, when the user move the mouse with the right button pressed and only one node selected, a new node is created and a translation simultaneously started with the new node. When the mouse button is released, a check take action on, to verify that the node is not too close to the original one. If the distance

between the two centres of the nodes is bigger than the radius of the smaller node, then the position is valid, and the node is placed.

There are another two ways to add nodes. The first one is with re-sampling of the bone, as explained in the paragraph 3.1.9. The second one is by adding a node between two existing ones. We realized that sometimes can be useful add a new node directly halfway between two connected nodes, and usually it is a very long operation. We decided, therefore, to add a method to achieve this result in just a keyboard shortcut.

3.1.5 Removing nodes

Removing nodes is an operation with a number of different cases that must be considered. In this section we will analyse every possible way to delete one or more node. A node has no dimension limit, but if the distance between the node we are scaling and another node is less then the minor of the radius of the two nodes, then the scaling is blocked.

Single node removal

The basic way to delete a node is to select it and activate the removal. Depending on the kind of selected node, there will be a different effect. A leaf node will be simply deleted, and from the only node connected to it will be removed from the neighbours list. A joint node will be removed and the two adjacent nodes' neighbours lists will be modified to add each other in their list, to establish a link between them.

The last possibility is the removal of a branching node. When a branching node is selected for elimination, the user is questioned for two different possibilities. The first one, is to delete all connections of the node. This means that all bone that end with a leaf connected with the branching node, and the branching node itself, will be deleted. The bones that end with another branching node will remain untouched.

The second possibility is to transfer the links on the branching node that is going to be deleted to another node. The node on which transfer the links must be a neighbour of the original branching point.

The removal of the branching nodes has been handled in this way because we wanted to have only one connected component in the skeletons, because it is easier

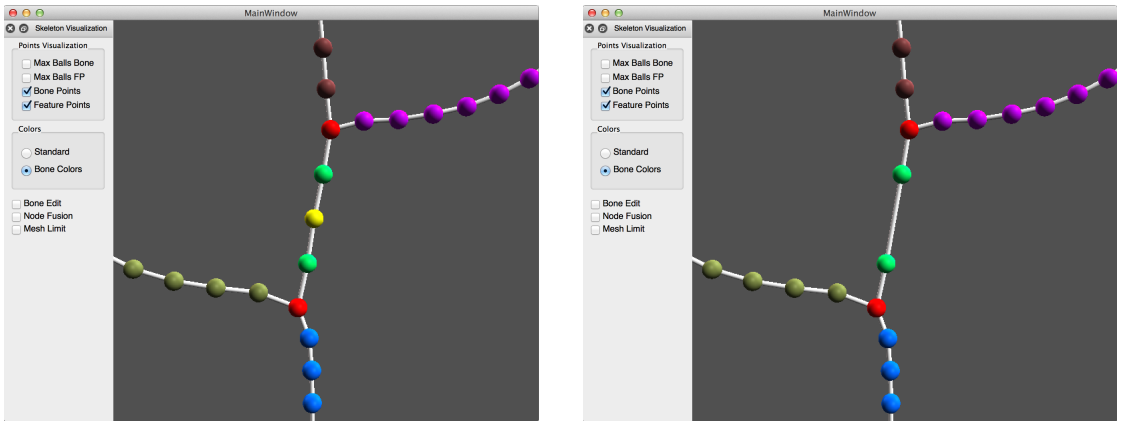
to control and fundamental for the mesh generation. In this way, we have made impossible to not have only one connected component, but the user can operate as he desire with his skeleton.

Multiple node removal

We illustrated in the previous paragraph the removal of a single node, but it possible to delete more than one node at a time.

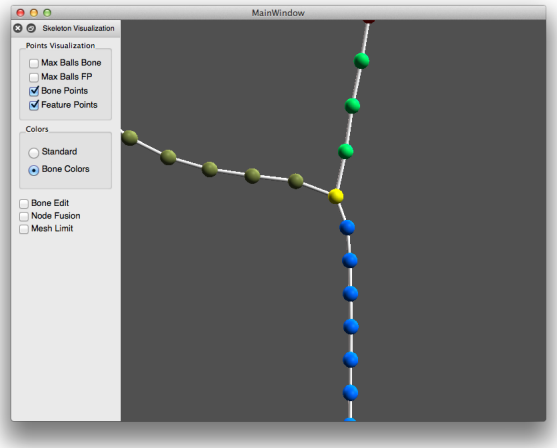
There are different scenarios for the multiple node erasing. The first one is the case in which unconnected nodes are selected, except branching nodes. In this situation, all the nodes are deleted in the same way as they were removed one at a time. In the situation that even a branching node is selected, the deletion of the branching node will be the same as the user selected to delete all connections as explained in the previous paragraph. It is not possible delete two or more branching nodes at a time.

When the user select more than one joint node, all of them are deleted and the neighbour of the first one is linked to the neighbour of the last one. If a leaf is selected too, then no new links are created, but only the first branching node is removed from the neighbours list of its neighbours.

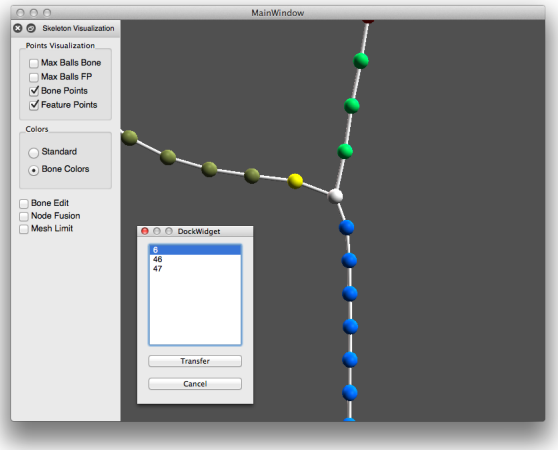


(a) Before the removal. (b) After the removal.

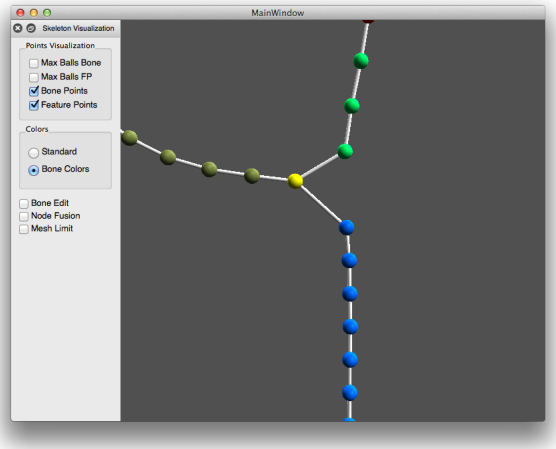
Figure 3.3: Example of a joint node deletion.



(a) Selection of the branching node to delete.



(b) Selection of the node on which transfer.

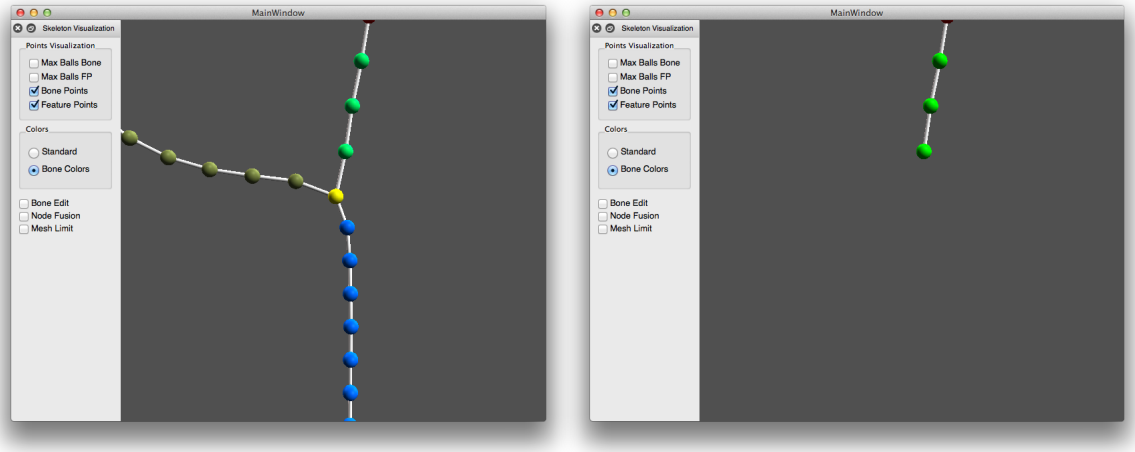


(c) After the transfer.

Figure 3.4: Example of a branching node deletion with the transfering of the connections.

Removing node in bone edit mode

There are several ways to remove nodes even in bone edit mode. They are explained in the paragraph 3.1.9.



(a) Selection of the branching node to delete.

(b) After the removal of the branching node.

Figure 3.5: Example of a joint node deletion and its connections. The green nodes are connected with another branching nodes, so they are not deleted.

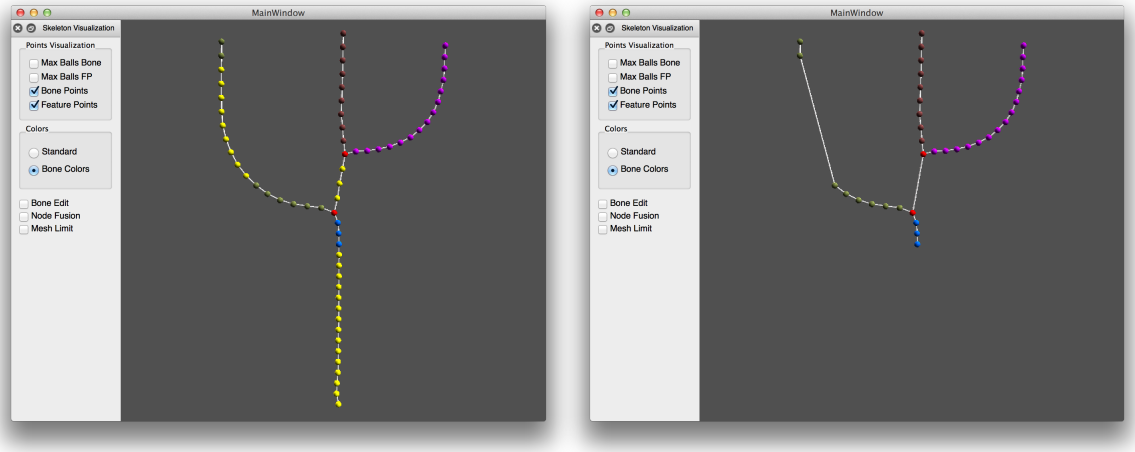
3.1.6 Undo and redo changes

Edit a skeleton is a complex job, it is easy make mistakes and sometimes it is necessary a trial and error approach. For this reason, it is useful having an instrument that allow users to go back on their steps, and cancel their last edits. In order to confer the feature a more useful employ, we made possible to restore even the removed selections, in this way, if a user go wrong and cancel his selection, he will be able to put back it and continue his work straightforwardly. Of course, it is possible to cancel the undo, and bring back the modifications previously revoked. There is not a limit of changes that is possible to cancel and restore, but, once a new edit is done, it is impossible to get back the modifications undone.

3.1.7 Copying nodes

Sometimes, a user creates skeleton parts that he wants to replicate in other sections of it. Therefore we implemented a way to copy some component of the skeleton in order to paste them in other sections.

To perform it, the user has to select the nodes that wants to copy. Once selected



(a) Before the removal. (b) After the removal.

Figure 3.6: Example of a multiple node deletion.

and started the copy, the user is asked to choose the node that will be the center of the duplicate. That node will be merged with the node on which the component copied will be pasted.

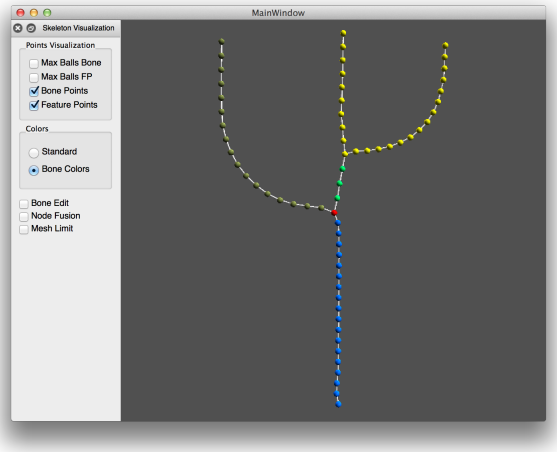
It is not mandatory to paste right after the copy. The user can execute some operations after the copy without losing it. It will be lost only after another copy.

3.1.8 Cycles

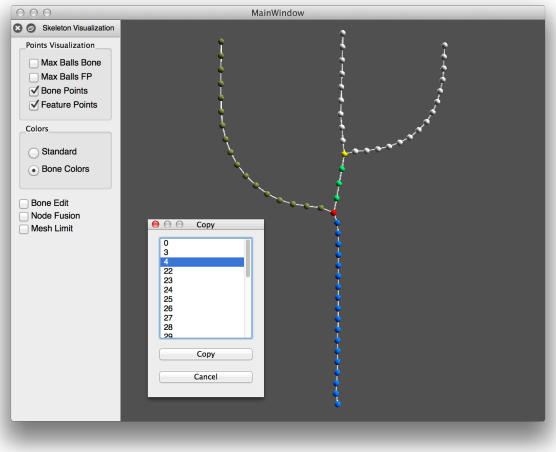
Skeleton, as explained in the introduction, are graphs. Therefore, there is the chance that a cycle occur.

We implemented two features that involve cycles. The first one is the node fusion. The node fusion is an activable mode that admit the fusion of two nodes that are too close. When the user translates a node, if the distance between the two nodes is less then the minor of the radius of the two nodes, then the nodes will be merged. With the fusion of the two nodes, a cycle is created.

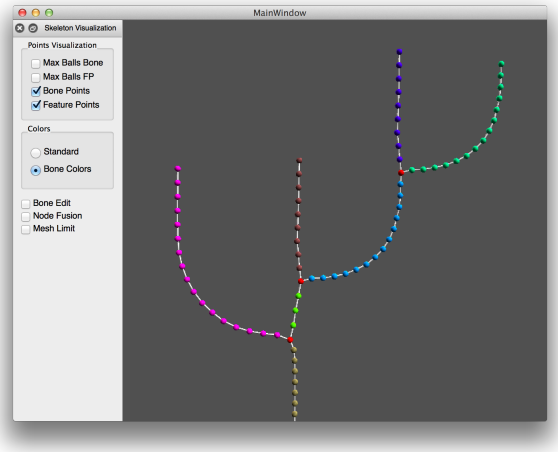
The second feature allows the user to break a cycle deleting the link between two nodes. This function works only on cycles. In fact, Dijkstra algorithm is used to check if, once deleted the link, it is possible to reach all the nodes from one of the nodes unlinked. If from the Dijkstra algorithm results that all nodes would be



(a) Selection of nodes to copy.



(b) Selection of the node to merge.



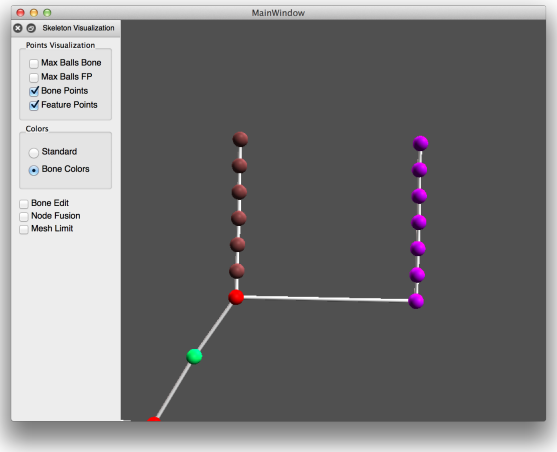
(c) After the paste.

Figure 3.7: Example of copy and paste.

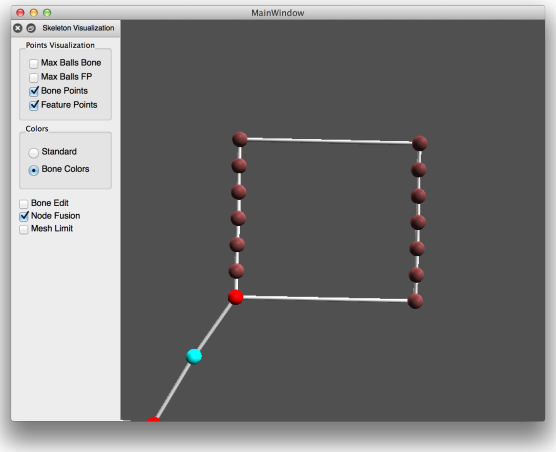
still reachable, then the link can be broken and is removed.

3.1.9 Bone edit mode

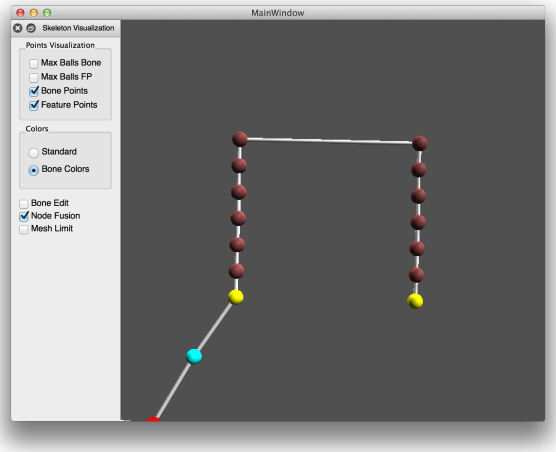
Editing the skeleton node by node it is not the only possible way. There are three operations that works only on bones, and not on single nodes. These operations can be used only activating the *bone edit mode*. The bone edit mode is an activable editing form, in which the selection act on entire bones and is possible to use bone-



(a) Bones before editing.



(b) A cycle is created.



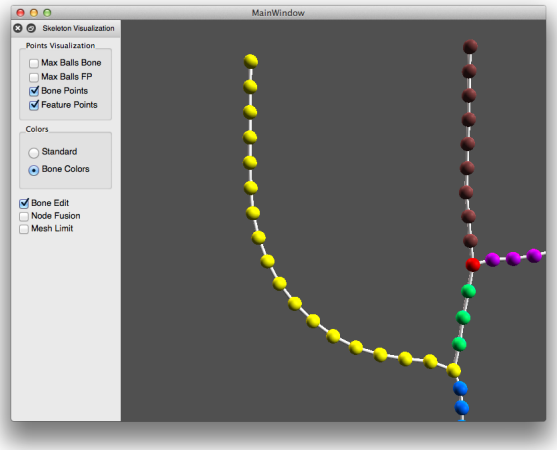
(c) The cycle is broken.

Figure 3.8: Cycle management.

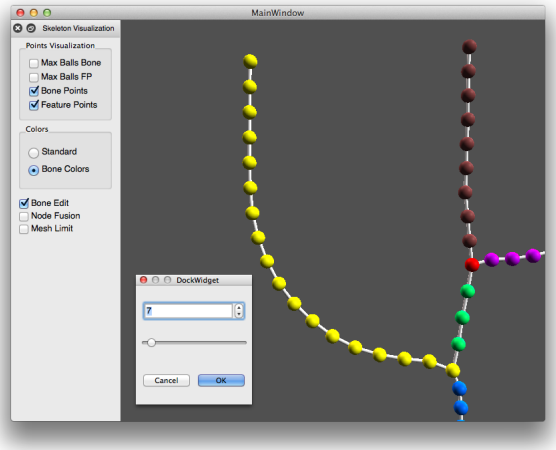
specific operations.

The first operation implemented for the bone edit mode is the *pruning*. The pruning is a functionality, available only for the bone that ends with a leaf node, that allow the user to remove a given number of nodes of the bone, starting from the leaf.

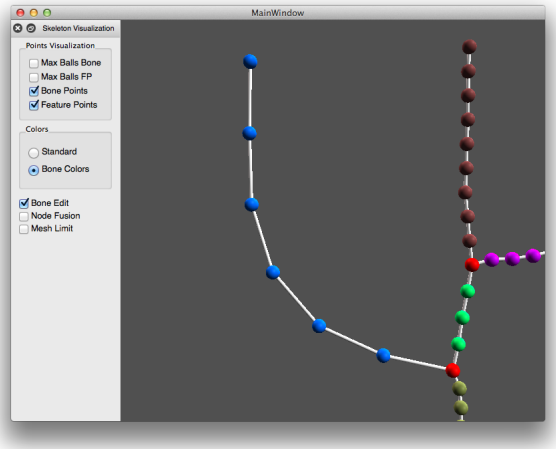
The second operation is the *deletion of an entire bone*. Unlike the pruning, this feature allow the user to delete an entire bone, but, similarly to pruning, the bone



(a) Selected bone.



(b) New node number selected.



(c) After the resample.

Figure 3.9: Re-sampling example.

must ends with a leaf. It is not allowed to delete a bone between two branching nodes or a cycle that starts and ends in the same node.

The third operation is the *re-sampling*. The re-sampling is a feature that allow the user to take a bone and change the number of the nodes composing that bone. The user can choose the new number of the nodes between two and one hundred. Two is the minimum number that can be chose because the ends of the bone must be untouched. The way we implemented this feature allowed us to keep the three-

dimensional structure of the bone intact, as long as the user re-sample the bone with a number of nodes not too small. Even the dimension of the radii of the nodes is considered in the process. Besides, we change the number of nodes interpolating positions and radii of the maximal spheres in order to better approximate the curve of the bone.

To maintain the three-dimensional structure, we take the bone and we stretch it. Then, we subdivide the stretched bone for a number of times equal to the new number of nodes minus one and we insert the new nodes. Once done, we can bend the bone as it was before.

To maintain the radii dimensions of the nodes, we use an interpolation process. We interpolate a function that, given the distance in the stretched bone, returns the dimension of the node. Once interpolated the function, we can easily obtain the dimension of a node in any point of the bone.

The re-sampling is a very useful functionality, in fact the automatically extracted skeletons are often composed by a very large amount of nodes, thus they are not ready to be used for some application, including the mesh generator that we developed (described in chapter 4). Having the chance to adjust the number of nodes which compose each bone is a very nice feature.

3.2 Used Technologies

In this section we will show the technologies employed in the realization of the project and a quick analysis and explanation of the reasons of their choice.

3.2.1 Qt

Qt (pronounced *cute*) is a cross-platform application and user interface framework widely used for applications, especially for software with a graphical user interface.

Qt uses standard C++, but it uses a special code generator, called *Meta Object Compiler*, and several macros to enrich the language. Qt can also be used in other programming languages via language bindings. The language binding is an application programming interface which provides glue code (code used only to meet program's requirements) to use the framework in a particular programming

language.

Qt is available under a commercial licence, GPL v3 and LGPL v2.

3.2.2 libQGLViewer

libQGLViewer is a C++ library based on Qt that makes easier the develop of applications which use OpenGL 3D viewers.

The major advantage of using libQGLViewer instead of using directly OpenGL libraries is that the first already includes the typical three-dimensional viewer functionalities, such as the possibility to move camera using the mouse, manipulate frames, object selection, etc.

Since libQGLViewer is based on Qt toolkit, it is cross-platform. Besides, it is available with both GNU/GPL and commercial licence.

3.3 Meshlab

As said at the beginning of the section 3.1, at first it was taken the decision to not develop an entire environment for the creation of the skeleton, but to create a plug-in for the widely used program Meshlab [4].

Meshlab is an open source software system for the processing and editing of three-dimensional triangular meshes. It is developed by ISTI (Institute of Information Science and Technology) - CNR (National Research Council), but initially it was created as a course assignment at University of Pisa in 2005. The system is aimed at the processing of not-so-small unstructured three-dimensional models that came to light in the 3D scanning pipeline. At last, Meshlab is available for most platform, including Windows, Linux, Mac OS X, and, with reduced functionality, on iOS and Android.

An important feature of Meshlab is its extensibility. As a matter of fact, Meshlab allows developers to add their own plug-ins to the system, in order to add various functions to it.

Anyway, even if Meshlab proved to be an excellent system to work on single meshes, it is not a good choice for working with multiple meshes. It was our need to add nodes to our skeletons, but Meshlab has a mesh management that is not

compatible with our requirements. For instance, if we load a skeleton in Meshlab, it is loaded as a single mesh; then, if we want to add a new node, it will be added as a new mesh. Since Meshlab does not allow to interact with multiple meshes, but they are shown as different layers, the user needs to change layer before he can interact with it. This made the user interaction extremely cumbersome.

Due to this reason, we decided to develop the editor without the support of other environment.

Chapter 4

Mesh Reconstruction from Skeleton

The second purpose of the thesis was to allow the user to generate a three-dimensional mesh starting from the skeleton created. In this chapter we will illustrate the approach that we used to generate the mesh and the basis of our work.

Our work consisted in re-elaborating an existing program which projected the point on an existing triangular mesh to obtain a quad-mesh. We re-elaborate this work to obtain a quad-mesh without using a triangular-mesh, but projecting the points using just the radii of the nodes of the skeleton.

The input of the algorithm that creates the mesh is a curve-skeleton, represented with a graph. As described in the chapter 3, we created a software with which the users could create a skeleton valid for this algorithm or edit a previuously created one.

4.1 Hexahedral boxes of the branching nodes

The first step of the algorithm consists in creating the modeling primitives, necessary for the successives steps. The basic modeling primitive of this algorithm is a hexahedral box, that are built on the branching nodes. From these boxes will be created *tubes* meeting at branching nodes, necessaries for the following projection based on the nodes radii. With this method, along the tubes there will be only regular vertices, while the singularities will appear on branching nodes. The hexahedral boxes, placed at each branching node can be subdivided into *facets* with the

purpose of the extrusion of an arbitrary number of tubes from that face.

Each hexahedral box is submitted to two operations. The first one is the *box alignment*. The branching nodes keep informations about how the tubular structures are connected. In the case of three branches, the most common intersection are T- and Y-branchings. With such configurations, it is easy to realize cubic joints. However, with the Y-branching, the upper face of the branching node box is crossed by two branches, forcing the face to be splitted in two facets, and a split of this kind make complex to obtain T-junctions along the branches. For branching nodes with more branches, the configurations can be more complex, so it is important to select the better configuration in a general way.

The boxes are oriented at branching nodes trying to keep their faces as orthogonal as possible with respect to the directions of incoming branches. In order to do this, the following optimization problem has to be solved indipendently at each branching node. Given the branching node n_i and the set of directions of its incident arcs $\{\mathbf{d}_1^i, \ldots, \mathbf{d}_{k_i}^i\}$, we search for the orthogonal basis $U_iV_iW_i$ that minimize the function:

$$f(U_i, V_i, W_i) = \sum_{j=1}^{k_i} |\mathbf{d}_j^i \cdot U_i| + |\mathbf{d}_j^i \cdot V_i| + |\mathbf{d}_j^i \cdot W_i|$$

The brancing node will be a box aligned with the axis $U_iV_iW_i$. Each term of the sum reach its minimum when one of the three coordinate is aligned with the branch, and its maximum when the branch run through any corner of the box.

The second operation on the hexahedral box is the *box subdivision*. Let B_i be the box of the branching node n_i, each arc a incident at n_i is associated to the face of B_i stabbed by the branch containing a. More than one arc can be associated to the same face of B_i. These faces are splitted into *facets* in order to allow that the tubular structures corresponding to different arcs can be extruded independently. The intersection points between a face f and its incident branches are calculated and f is splitted in a number of parallel rectangular facets equal to the number of incident branches on f. In order to choose the direction in which the face will be splitted, its intersection points are projected to the two cardinal directions parallel to the sides of f (in the $U_iV_iW_i$ basis), and it is selected the direction on which the projection cross a larger interval.

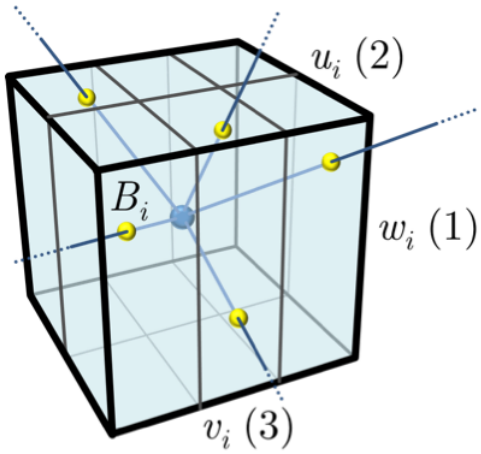


Figure 4.1: An example of how a cube on a branching node is created.

4.2 Building of the tubes

The next step of the algorithm is to build the tubes with a quadrilateral section that connects two different branching nodes or a branching node and a leaf node. In order to do this, a square ring is associated to each joint node, and next they are concatenated with longitudinal edges almost parallel with the skeleton. The square ring is placed in a transversal plane orthogonal to the tangent direction of the skeleton, computed as the average direction of its two incident arcs. The ring orientation is set according to a given frame.

For the tubes between a branching node and a leaf node, the algorithm starts removing the face, or the facet, pierced by the branch of the skeleton and extruding the tube along it. A two-dimensional reference frame is set aligned with the side of the removed face or facet. An open hexahedrical box is placed centered on the leaf node and oriented in the same direction of the incident arc of the skeleton. The tube is built by joining correspondig corners of the square rings, starting from the open face or the facet of the brancing node up to the open face of the leaf node. The orientation of the two-dimensional frames set the orientation of the rings about the branch, so there are not torsions along it.

For the tubes between two branching nodes, as done for the other case, the reference frame is set fixing it at one of the two ends. In general, there will be a

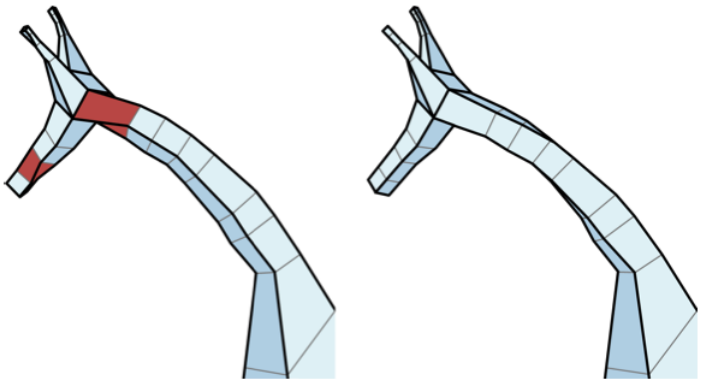


Figure 4.2: An example of how the torsion is handled.

mismatch between the orientation of the reference frame and the orientation of the branching box at the other end. This mismatch introduce a torsion on the tube, that can be distributed by interpolating the torsion angle along the branch and rotating the rings at intermediate joint nodes accordingly with the interpolation. This is done by parametrizing the arc length along the brances.

4.3 Projection of the vertices

Once built the basic mesh, the original algorithm projected the points on the already existing triangular-mesh. We modified the algorithm to generate a quad-mesh with an input skeleton only. Each vertex on the mesh keep a reference of which skeleton node has generated it.

For the vertices generated from a branching node, we need to take all the incident edges on that vertex and discard all the edges which are part of the branching node's cube. To get the direction on which the new vertex will be generated, we need to normalize the remaining edges and summing them, in order to obtain an average vector of the directions. Once obtained the direction, we can compute how much to move the node in the following way:

$$p_i = r_i - d_i$$

where p_i is the new *projected coordinates*, r_i is the radius of the skeleton node

corresponding to the vertex i and d_i is the current distance of the vertex i from the center of the associated node.

Instead, for the other vertices, to compute the projection direction, we use the following formula:

$$W = v - r$$

where

$$r = n_a * \frac{v \cdot n_a}{n_a^2}$$

and n_a is the plane normal, v is the vector we want to project, a is the plane.

To compute how much the node has to move, we use the same method of the branching nodes.

Chapter 5

Conclusions and future work

In this chapter we will analize the results obtained from the realized work and we will show further operations that can be intergrated in the skeleton editor.

5.1 Results

To summarize, we have realized a fully functional skeleton editor, which allow user to create a complex and articulated skeleton or to load already built one and edit it, and we realized a mesh generator based on the radii of the nodes.

The results, with both the softwares, are good. We developed all the features that we decided to implement at the beginning of the development and we tested multiple times all the features. For what concern the mesh generator, sometimes with complex models the algorithm produces some auto-intersections. This should be solved in the future. However, the results are good both for number and valency of the singularities and for the edge flow.

The pictures in this chapter shows three different built skeletons with all nodes, with the feature points at their real size and finally the generated mesh.

5.2 Future works

During the developement of the skeleton editor and the mesh generator, we focused on the main features that should have to be implemented to be a complete and

valid software. However, there are some feature that could be developed to make the program even more complete.

The first one is a multi view modality. The idea was to split the main viewer of the skeleton editor, in order to have four different point of view of the skeleton that is being created. One is the usual view that we implemented, the other three are fixed on the top, the front and the side of the skeleton, to help the user to move and edit nodes in an easier way.

Another feature that would useful to implement is the mirroring. Other programs, such as B-Mesh and Z-Sphere allow the user to build two separate skeleton section totally symmetrical between them. It is useful in many cases, because avoid the user to create two identical segment of the skeleton, with the problem to make them of the same sizes.

At last, would be really useful to extend the support of the skeletons format, in order to facilitate the spread of the program.

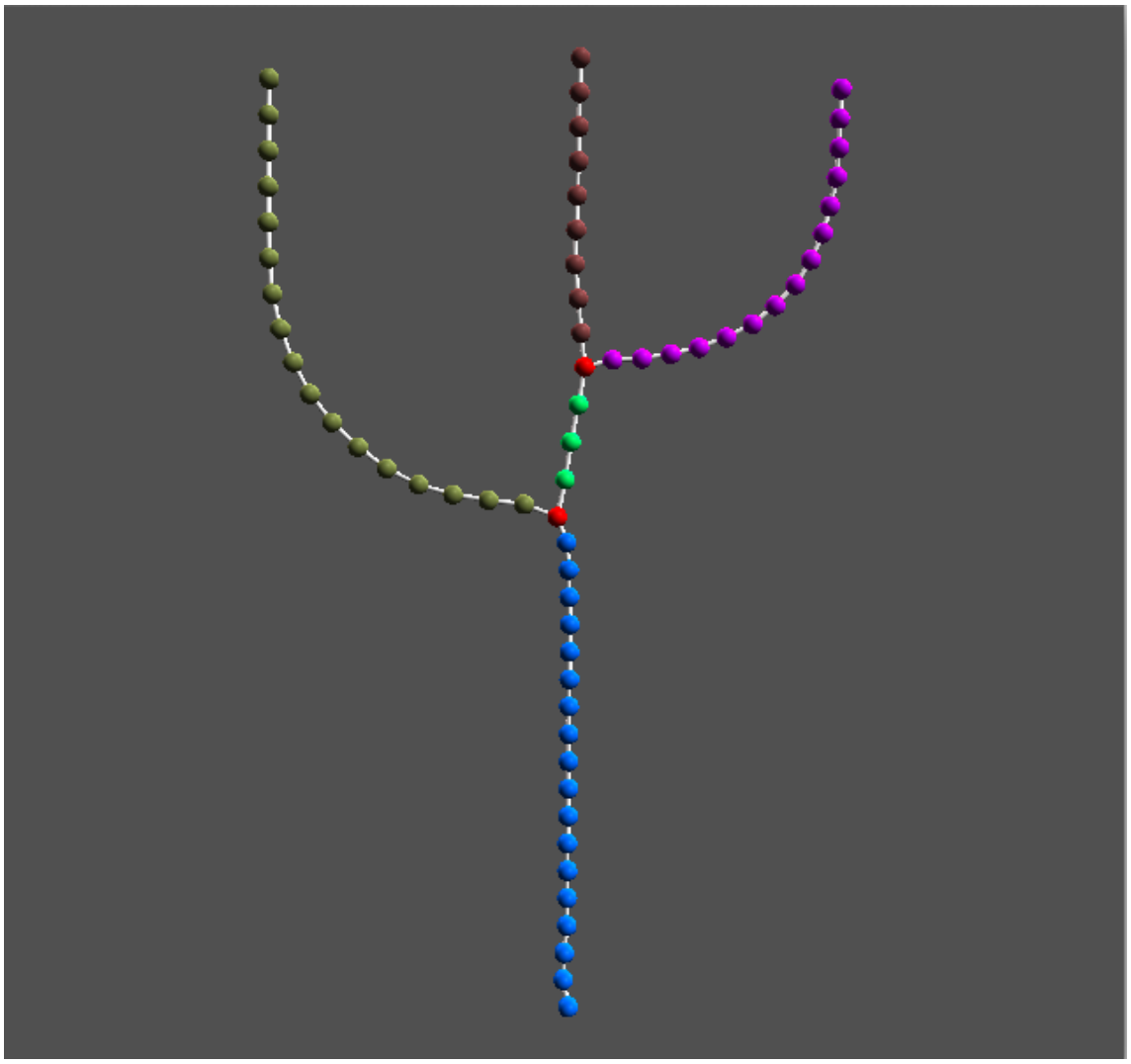


Figure 5.1: The cactus skeleton with all nodes.

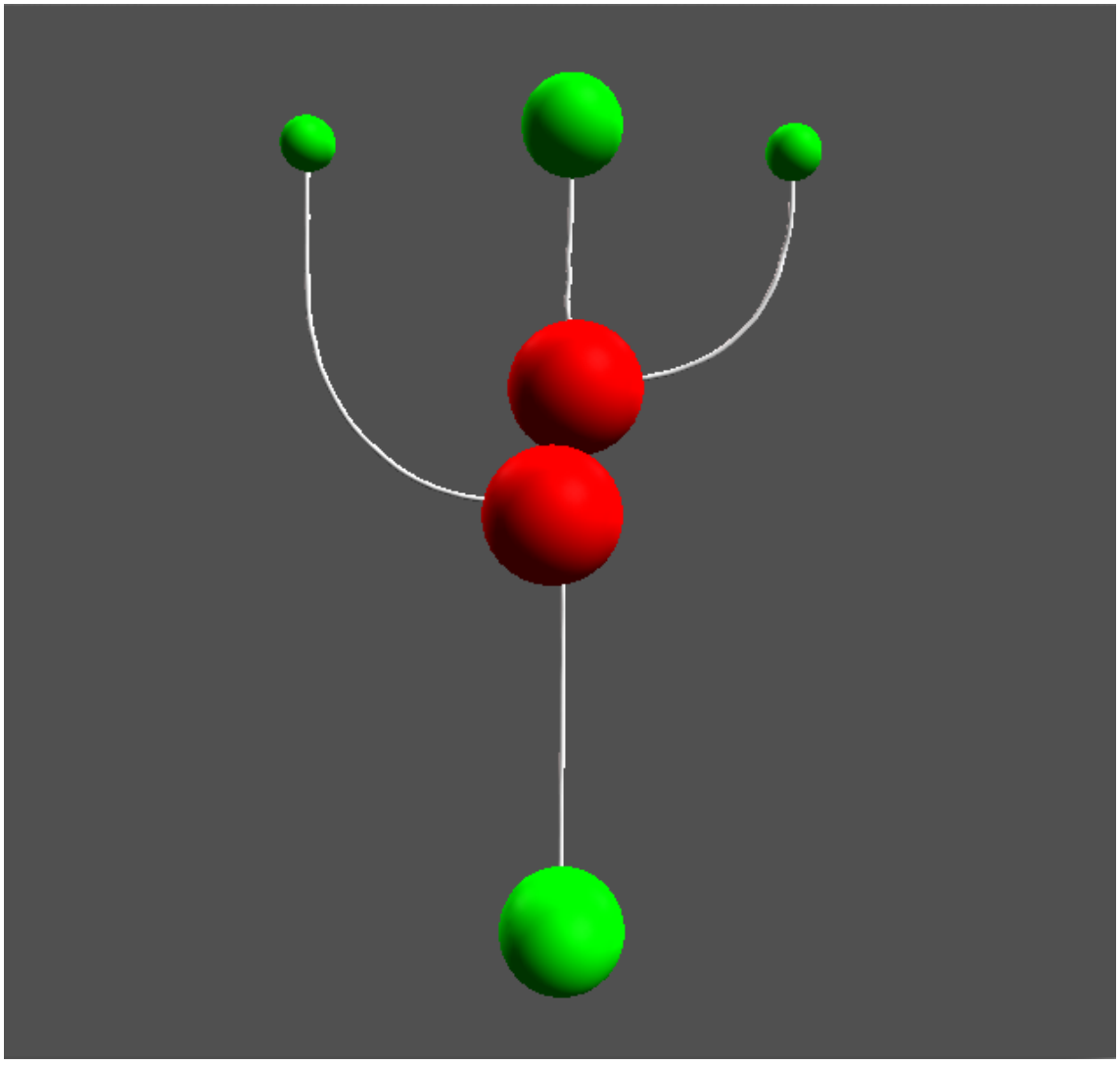


Figure 5.2: The cactus skeleton with the feature points at the real size.

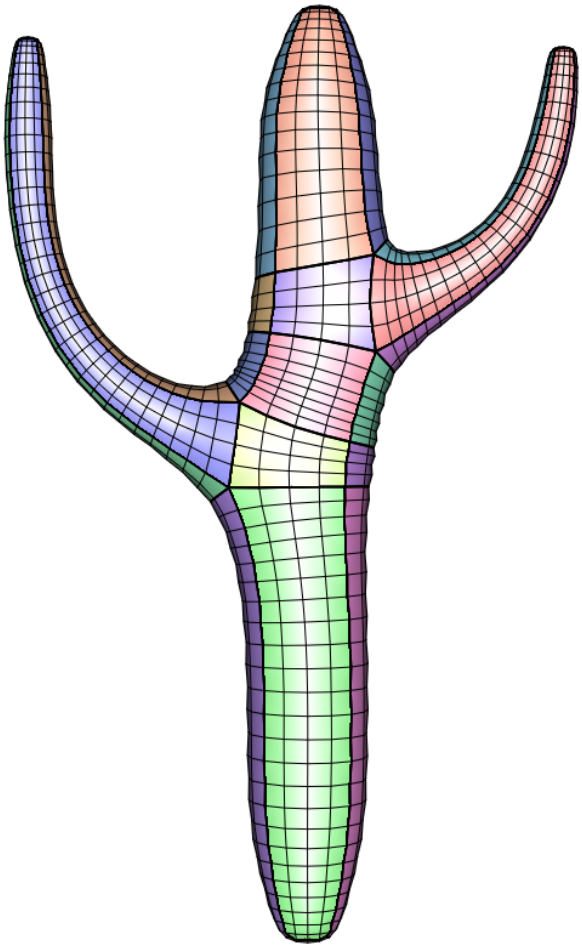


Figure 5.3: The cactus mesh generated from the skeleton.

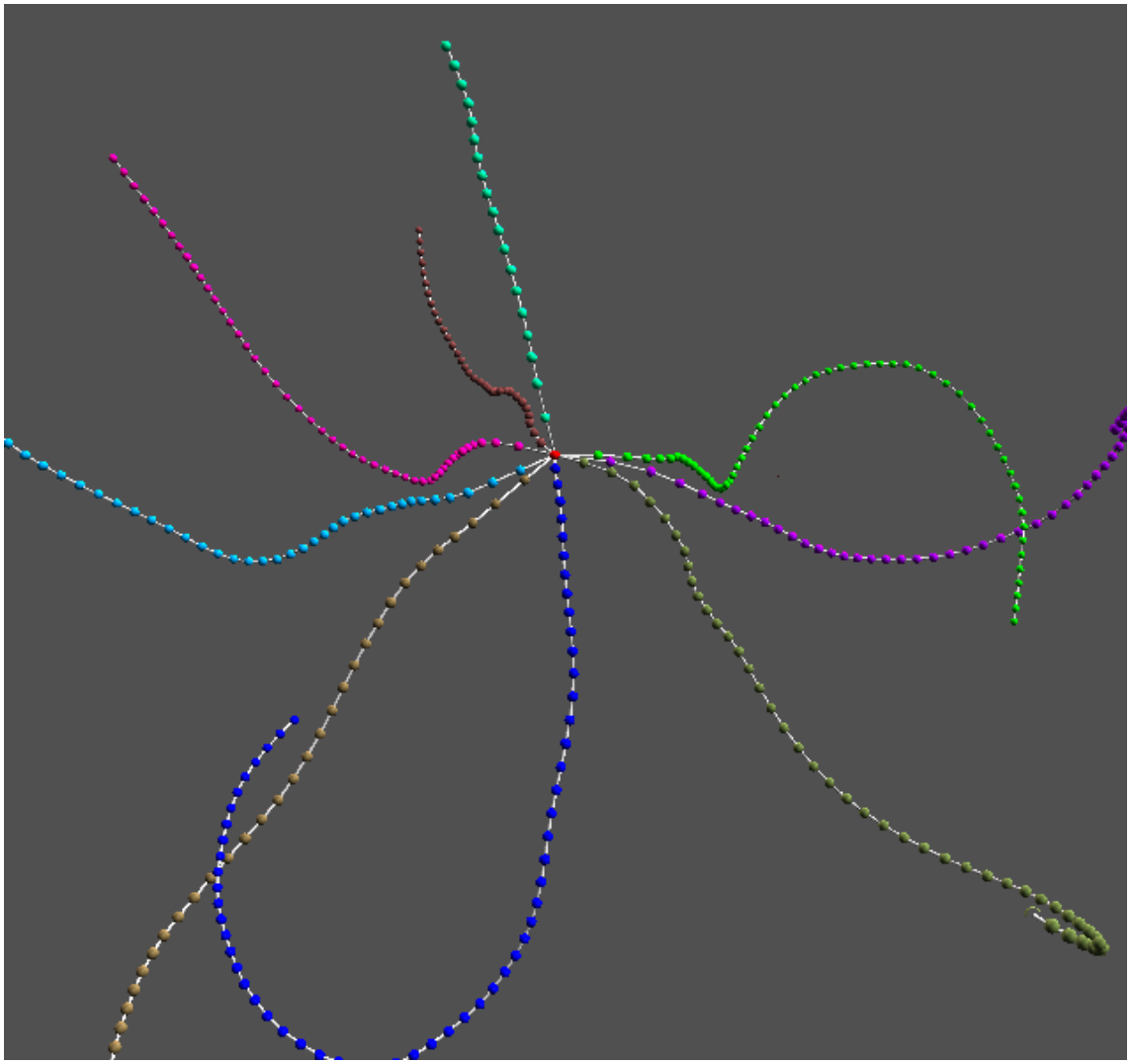


Figure 5.4: The octopus skeleton with all nodes.

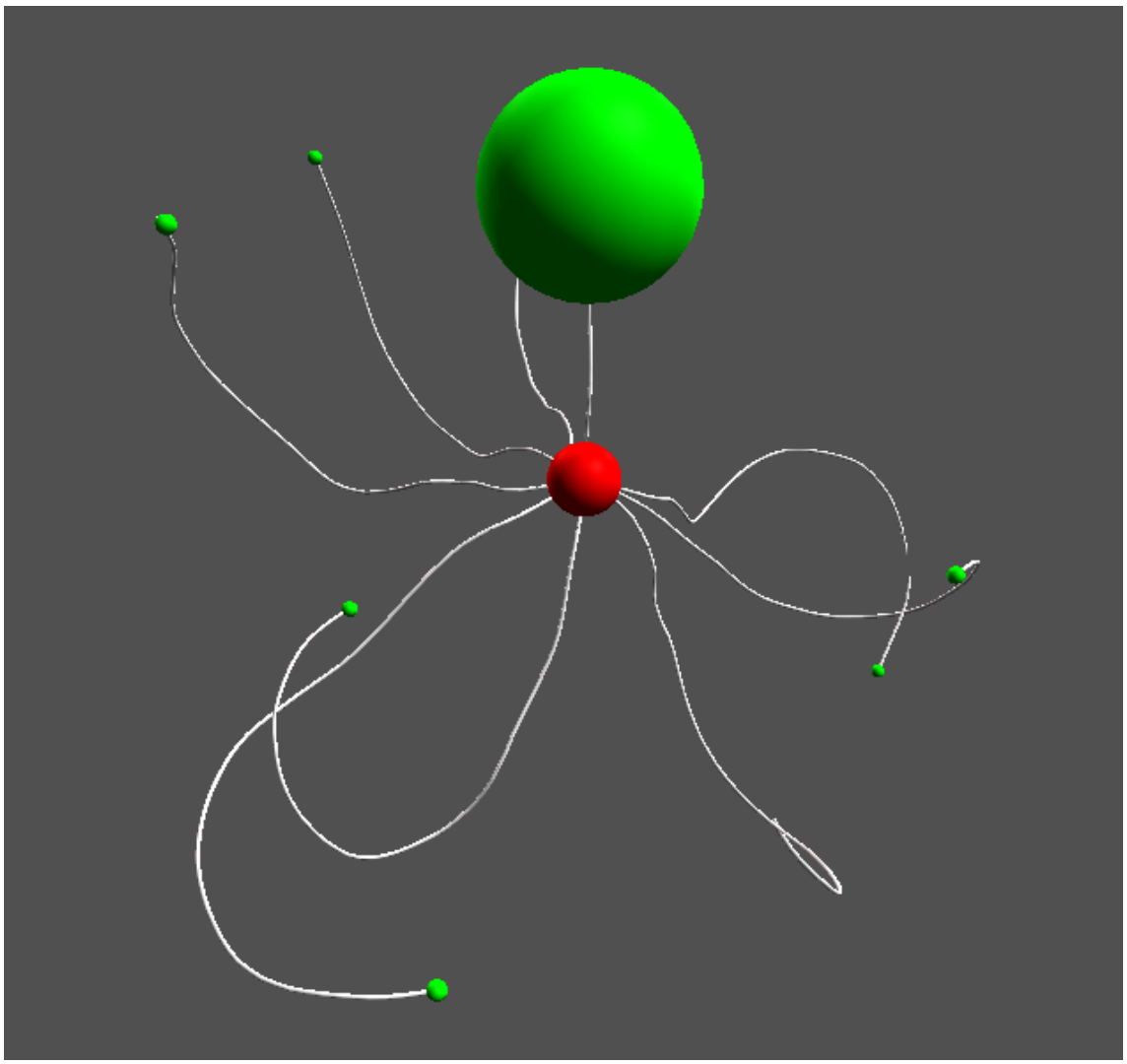


Figure 5.5: The octopus skeleton with the feature points at the real size.

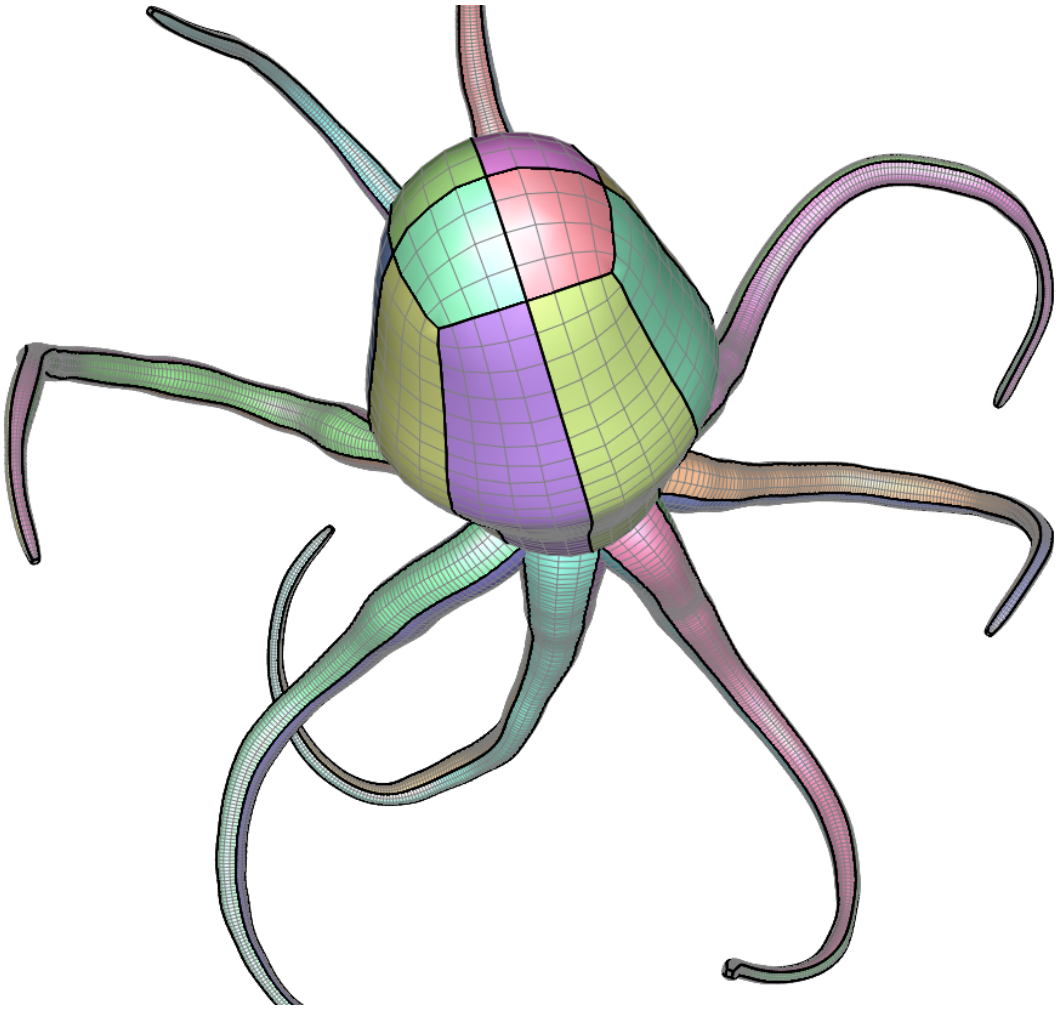


Figure 5.6: The octopus mesh generated from the skeleton.

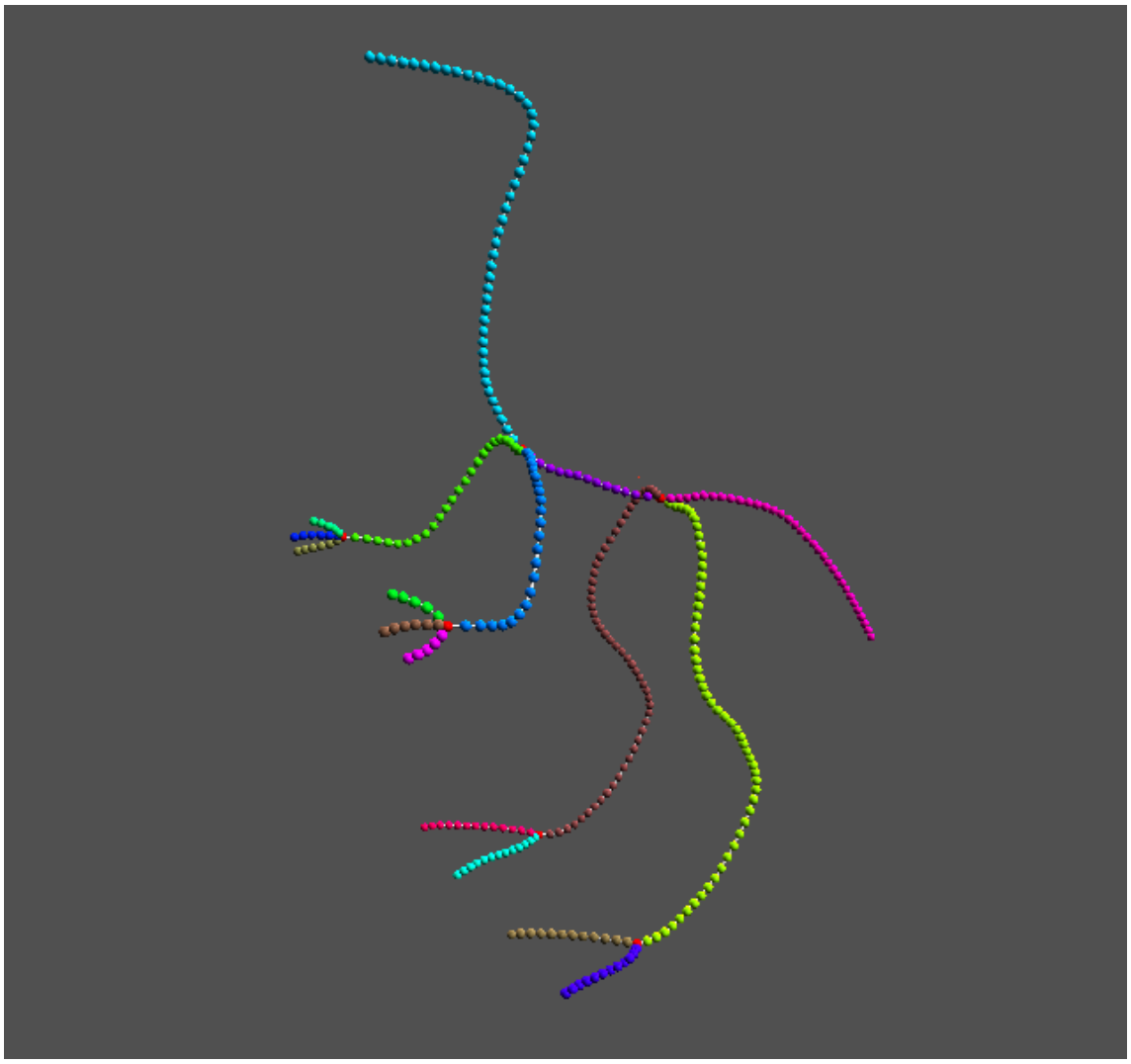


Figure 5.7: The dinopet skeleton with all nodes.

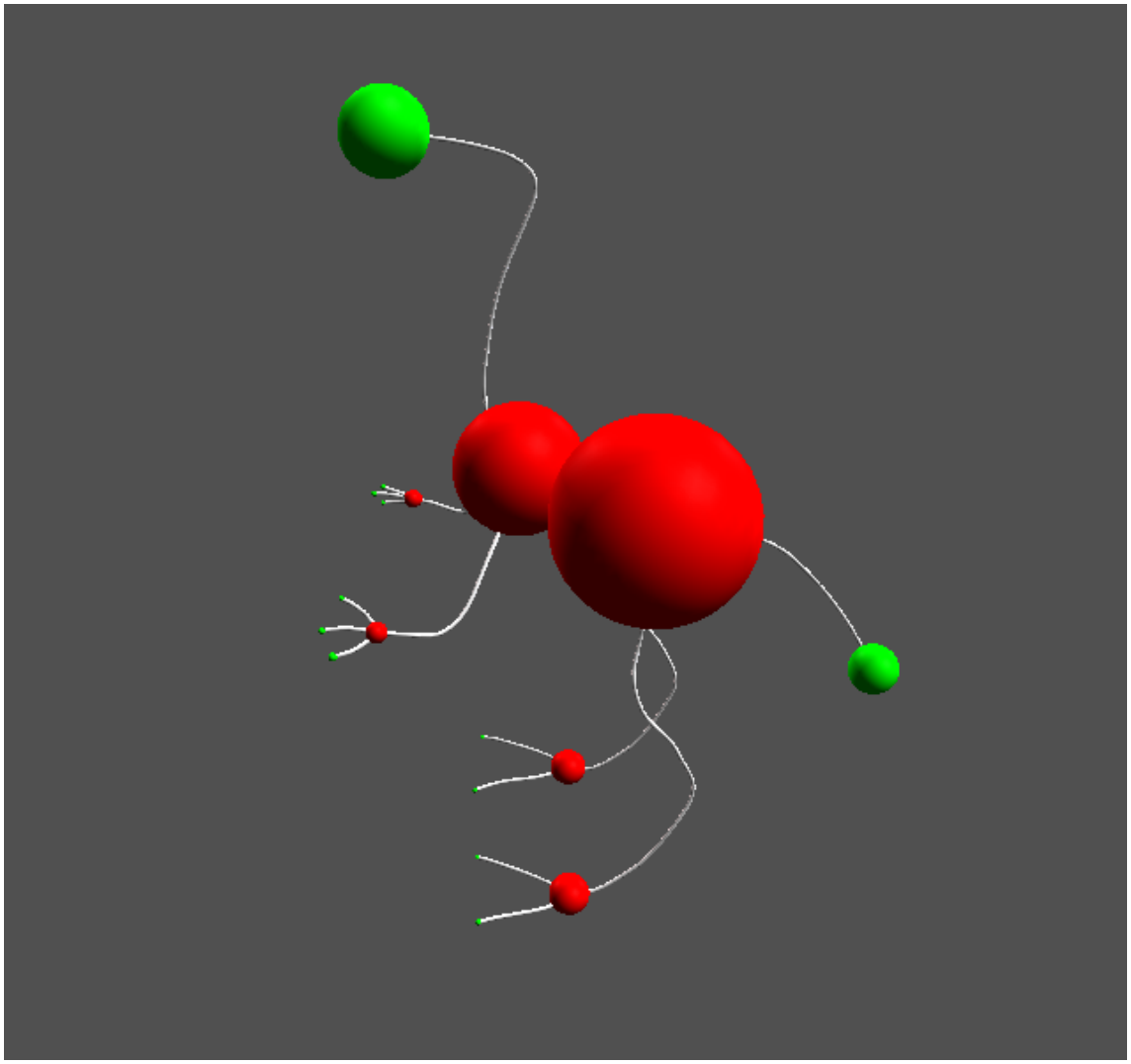


Figure 5.8: The dinopet skeleton with the feature points at the real size.

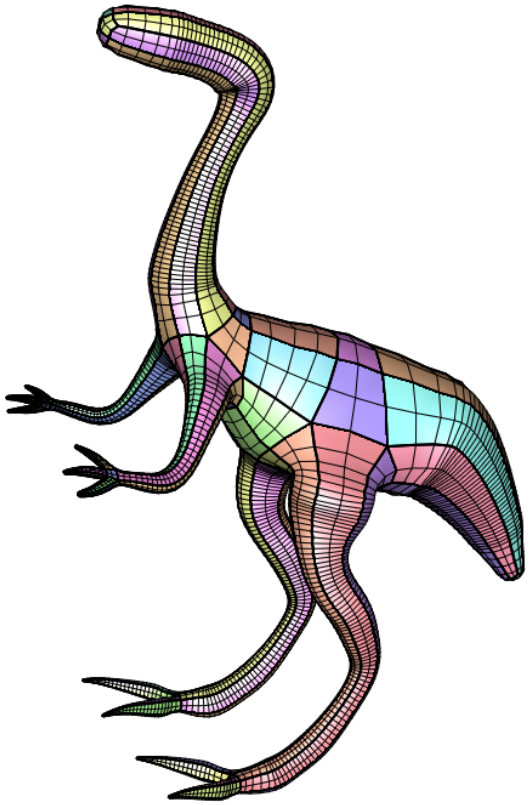


Figure 5.9: The dinopet mesh generated from the skeleton.

Bibliography

[1] Autodesk maya. http://www.autodesk.com/products/maya/overview.

[2] Cinema4d. http://www.maxon.net.

[3] Electronic arts. http://www.ea.com.

[4] Meshlab. http://meshlab.sourceforge.net.

[5] Z-brush. http://pixologic.com/zbrush/features/overview/.

[6] Jacob Andreas Bærentzen, Rinat Abdrashitov, and Karan Singh. Interactive shape modeling using a skeleton-mesh co-representation. In *41st International Conference and Exhibition on Computer Graphics and Interactive Techniques*.

[7] Jakob Andreas Bærentzen, Marek Krzysztof Misztal, and K Wełnicka. Converting skeletal structures to quad dominant meshes. *Computers & Graphics*, 36(5):555–561, 2012.

[8] Harry Blum et al. A transformation for extracting new descriptors of shape. *Models for the perception of speech and visual form*, 19(5):362–380, 1967.

[9] Eelke Folmer. Component based game development–a solution to escalating costs and expanding deadlines? In *Component-Based Software Engineering*, pages 66–73. Springer, 2007.

[10] Zhongping Ji, Ligang Liu, and Yigang Wang. B-mesh: A modeling system for base meshes of 3d articulated shapes. In *Computer Graphics Forum*, volume 29, pages 2169–2177. Wiley Online Library, 2010.

[11] Marco Livesu and Riccardo Scateni. Extracting curve-skeletons from digital shapes using occluding contours. *The Visual Computer*, 29(9):907–916, 2013.

[12] Vinod Srinivasan, Esan Mandal, Ergun Akleman, et al. Solidifying wireframes. In *Renaissance Banff: Mathematics, Music, Art, Culture*, pages 203–210. Canadian Mathematical Society, 2005.

List of Figures