

Harmonic Skeleton for Realistic Character Animation

Grégoire Aujay¹ Franck Hétroy¹ Francis Lazarus² Christine Depraz¹

¹EVASION - LJK (CNRS, INRIA and Univ. Grenoble)

²GIPSA-Lab (CNRS and Univ. Grenoble)

Abstract

Current approaches to skeleton generation are based on topological and geometrical information only; this can be insufficient for realistic character animation, since the location of the joints does not usually match the real bone structure of the model. This paper proposes the use of anatomical information to enhance the skeleton. Using a harmonic function, this information can be recovered from the skeleton itself, which is guaranteed not to have undesired endpoints. The skeleton is computed as a Reeb graph of such a function over the surface of the model. Starting from one point selected on the head of the character, the entire process is fast, automatic and robust; it generates skeletons whose joints can be associated with the character's anatomy. Results are provided, including a quantitative validation of the generated skeletons.

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

1. Introduction

A common technique for animating a 3D model consists of creating a hierarchical articulated structure, named skeleton (or IK skeleton), whose deformation drives the deformation of the associated model. The location and displacement of the skeleton's joints dictate how the model moves (see Figure 1 for an example). A skeleton attached to a 3D model (usually represented as a mesh) can be either created by hand or computed. In the case of the *realistic* animation of a character (be it a human, an animal or a made-up monster), the first option is most often chosen by artists, although it is a time-consuming task which needs a skilled user. Indeed, professional artists may create an initial skeleton relatively quickly, but often need to make many adjustments during the rigging process because the skin is very sensitive to the exact location of the skeleton's joints: they often have to go back and forth several times between skeleton skinning and testing animation before getting it right. Automatic or semi-automatic methods have several drawbacks: they often allow little control over the result, they can produce noisy skeletons with unwanted joints, and most importantly they rely on the topology and the geometry of the model only, which is not sufficient for realistic animation where the anatomy of the model does not completely match its geometry. For instance, in most cases the spine of a character is close to its

back, while the corresponding axis in computer-generated skeletons is usually centered within the body (see Figure 12). Moreover, animation skeletons may have some joints which do not match any anatomical part of the model but are useful for animation purpose (e.g., on the head, see Figure 2).

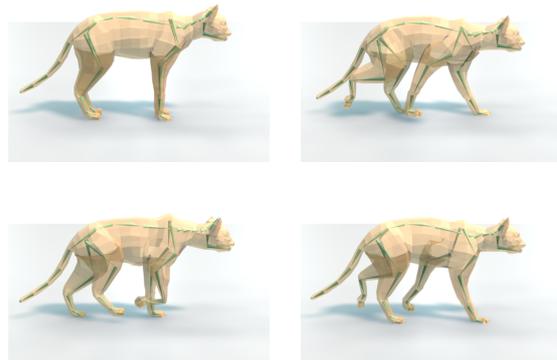


Figure 1: *Walking cat. These images are taken from an animation created using our harmonic skeleton (see the video): joints deformation drives the mesh deformation.*

This paper explains how to automatically, robustly and efficiently compute skeletons adapted to realistic character an-

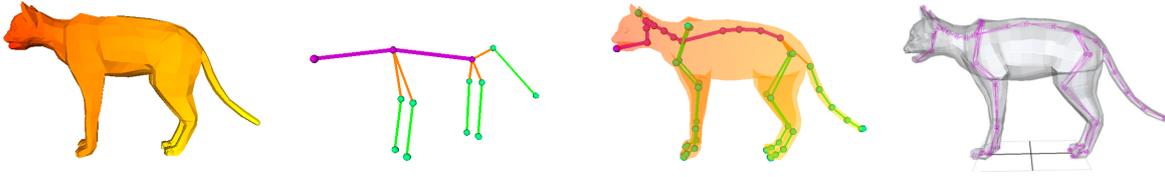


Figure 2: From left to right: a cat model, the computed harmonic graph with its symmetry axis, the computed harmonic skeleton compared to a previously handmade animation skeleton.

imation, starting from a single point selected by the user on the model. Generated skeletons match the ones that are created by hand by professionals in most biped and quadruped cases. Moreover, they carry anatomical information (that is to say, we know which joint corresponds to which part of the model), allowing a semantic decomposition of the input meshes.

1.1. Related work

Numerous algorithms have been proposed to compute skeletons of 3D shapes from their geometry. Bloomenthal and Lim [BL99] were among the first ones to point out that these geometric skeletons can be converted to IK skeletons and then used for animation purposes. However, to be useful for animation, skeletons should be structured as graphs, whose nodes correspond to the joints and whose edges correspond to their hierarchy. This discards two-dimensional skeletons such as the Medial Axis [Blu67].

Graph-like skeleton generation algorithms start either from the boundary surface [KT03, LWM*03, DS06, LKA06, TVD06] of the input model, or from its inner volume [GS01, WP02, CSYB05]. Methods working on a volumetric representation of a model have a major drawback: only features with a size greater than the voxel size can be taken into account. This often leads to computationally expensive algorithms.

Katz and Tal [KT03] extract a skeleton from a meshed model using a hierarchical decomposition of this mesh into meaningful parts. Generated skeletons are star-shaped (they contain a root joint, located in the center of mass of the model, from which all other joints derive) and thus are not suited for realistic animation. Lien et al. [LKA06] generate shape decomposition and skeleton simultaneously; the skeleton is computed using centroids and principal axes of the shape's components, which gives a skeleton with geometrically but not necessarily anatomically meaningful positions. The same problem appears with Dey and Sun's robust skeleton computation from the Medial Axis [DS06]. Liu et al. [LWM*03] propose to use a repulsive force field to position the joints. This method is quite slow (as reported in the paper, it takes several minutes to compute the skeleton for a model containing about 10,000 triangles), and does not guarantee that the result will capture all desired features.

Following Shinagawa et al. [SKK91], several authors have proposed to use a mathematical tool called the *Reeb graph* to capture the model's topology, before possible refinements to capture its geometry. A Reeb graph is defined with respect to a mathematical function, and the result highly depends on the choice of this function. In the next section, we precisely define the Reeb graph and then list some existing methods using this mathematical notion.

The algorithm we propose takes as input a triangle mesh. It first computes a Reeb graph of this mesh, in a fast and robust way (that is to say, the graph's leaves are only the desired ones). This abstract graph is then refined and embedded in the 3D space in order to be useful for realistic character animation; this is made possible thanks to a semantic decomposition of the model, given by the graph. Our algorithm computes the skeleton of a model with several hundred of thousand faces in no longer than a few seconds on a low-end computer.

1.2. Mathematical background

Let $f : M \rightarrow \mathbb{R}$ be a function defined over a 2-manifold M with or without boundary (that is to say, a surface for which each point has a neighborhood homeomorphic to a disk or half-disk). Level sets of f are the sets $f^{-1}(u) = \{x \in M, f(x) = u\}$. Each of these sets, if it exists, can be connected or not. For instance, on Figure 3, where f is a height function, $f^{-1}(u)$ is connected for low and high values of u , but is made of several connected components for values around -0.7 , 0 and 0.7 . For some special values, the number of connected components of the level set changes: these values are called *critical values*, and the corresponding points x on the surface are called *critical points*. The *Reeb graph* of f [Ree46] is a graph whose nodes correspond to these critical points, and which encodes the connectivity between them (see Figure 3). In particular, notice that the leaves of the Reeb graph exactly match the local maxima and minima of f . Mathematically speaking, the Reeb graph of f is defined as the quotient space M / \sim , with \sim the following equivalence relation on M :

$$x_1 \sim x_2 \iff \begin{cases} f(x_1) = f(x_2) \\ \text{and } x_1 \text{ and } x_2 \text{ belong to the same} \\ \text{connected component of } f^{-1}(f(x_1)) \end{cases}$$

More details about these notions can be found in e.g. [FK97].

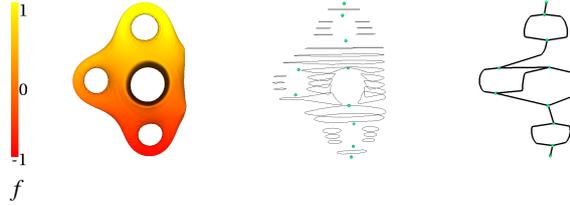


Figure 3: From left to right: a surface, some level sets of f , the Reeb graph of f .

A Reeb graph *w.r.t.* a triangulated surface with n edges can be computed in $O(n \log n)$ time [CMEH*03]. However, the choice of the function f is a key issue in revealing information about the surface, and several proposals have been made in order to obtain a relevant graph: the distance on the mesh to a source point [LV99], the integral over the mesh to such a distance (in order to avoid the choice of the source point) [HSKK01], a mapping function that highlights the relevant features [TVD06], etc. Following an idea proposed by Ni et al. [NGH04], we choose to find a “fair” function f , whose extrema will be anatomically significant, by solving Laplace’s equation $\Delta f = 0$. Steiner and Fischer did the same [SF01], but their Reeb graph captured neither geometrical nor anatomical features, only the topology of the model.

The main property of such functions f , called *harmonic functions*, is their lack of extrema except at boundary points [NGH04]. They also have the following property: let M be a compact surface, B_M its boundary and $g : M \rightarrow \mathbb{R}$ a function; there exists a unique solution $f : M \rightarrow \mathbb{R}$ to the following system, called Laplace’s equation with non-homogeneous Dirichlet boundary conditions:

$$\begin{cases} \Delta f(x) = 0 & \forall x \in M \\ f(x) = g(x) & \forall x \in B_M \end{cases} \quad (1)$$

In our case, B_M will be a (disjoint) set of vertices of the mesh, corresponding to anatomically significant parts of the model. We will compute the Reeb graph of the solution $f_{B_M, g}$ to the previous system (1), for some function g which will be described in section 2.1.1; thanks to the property of harmonic functions, the leaves of this graph will exactly match the chosen vertices: in other words, the graph is guaranteed not to be noisy.

1.3. Algorithm overview

The Reeb graph of a function *w.r.t.* a surface is a pair (V, E) with V a set of nodes and $E \subset V \times V$ a set of edges between these nodes. It is minimal in the sense that there is no regular node: each node has either one or at least 3 incident edges. Moreover, nodes do not have 3D coordinates. Thus, in order to construct a skeleton which is suitable for animation from such a graph, we must *embed* it in \mathbb{R}^3 , that is to say link each node with 3D coordinates. Thus, our method first computes a Reeb graph, then embeds it in \mathbb{R}^3 . In the following, *node*

will refer to the graph while *joint* will refer to the skeleton and *vertex* to the mesh.

Our algorithm runs in seven successive stages:

1. the endpoints of the desired skeleton are chosen by the user or computed (however at least one of them, called the *source node/joint*, must be manually chosen on the head of the character);
2. the harmonic function f solving Laplace’s equation with non-homogeneous Dirichlet boundary conditions is computed;
3. the Reeb graph of f is computed with the algorithm described in [CMEH*03];
4. this graph, which we call a *harmonic graph* since f is a harmonic function, is subsequently filtered to recover the symmetry of the character’s morphology (i.e., overall structure);
5. starting from the source node, the symmetry axis of the graph is detected;
6. the harmonic graph is refined by inserting regular nodes and embedded in \mathbb{R}^3 : this gives us the *harmonic skeleton*, which carries anatomical information about the input model (such as “this joint corresponds to the tail”);
7. additional heuristics are used in case the model is detected to be a biped or quadruped with sagittally oriented legs (this excludes amphibians, but includes most mammals), in order to fit the IK skeleton that would be manually created by an expert. Although not presented in this paper, equivalent heuristics can be defined for other kinds of characters, such as birds or insects.

The contributions of this paper are the following:

- the computed skeleton is robust: endpoints are exactly the ones that have been chosen, and two meshes representing the same model under two different postures generate equivalent skeletons;
- our algorithm is fast and does not need user intervention, except for the selection of the source joint at the very beginning. However, controlling the skeleton generation is possible, by manually choosing its endpoints or tuning some parameters;
- our method gives a semantic decomposition of the shape (which is used for the embedding process): we know which part of the mesh corresponds to the head, the legs, the trunk and the tail of the character;
- we propose standard skeletons (graphs and their embeddings) for bipeds and quadrupeds with sagittally oriented limbs.

Moreover, in the case of quadrupeds, we have validated our results not only visually but also by comparing parameters with handmade animation skeletons. To our knowledge, this is the first time a quantitative validation is proposed.

The organization of this paper is as follows: section 2 describes stages 1 to 4 of our algorithm, that is to say the computation of the harmonic graph; section 3 explains the construction of the harmonic skeleton from the harmonic graph,

that is to say stages 5 and 6; in section 4, we detail the proposed skeletons for bipeds and quadrupeds; we give results and discuss them in section 5; finally, we conclude in section 6.

2. Harmonic graph

2.1. Graph computation

2.1.1. Finding extrema

The first stage of our algorithm is to choose the endpoints of the skeleton; they will correspond to extremal joints. The user must select one source vertex x_{source} on the head of the character, which will give the source node of the graph. We set $f(x_{source}) = 0$. Other endpoints should match relevant anatomical features of the character that the user wants to animate: hands, feet and possibly tail, ears, etc. These endpoints can be either selected manually, or computed. In the latter case, we try to find vertices x such that the distance $d(x_{source}, x)$ on the mesh is locally maximum. Several methods have been proposed to solve this problem: for instance, Dong et al. [DKG05] choose to solve the Poisson equation $\Delta f = -\|\Delta x\|$; the algorithm proposed by Tierny et al. [TVD06] can also be applied, but it does not use the source vertex, which should be selected afterwards among the detected feature vertices, hence it does not ensure this vertex will be on the head of the character. The same problem arises when computing the average geodesic distance function over the mesh, as did Zhang et al. [ZMT05]. In our implementation, we use a fast and more straightforward solution: g is defined as a geodesic distance to x_{source} ; we use Dijkstra’s algorithm to compute shortest paths on the mesh from the source vertex to all other vertices, as proposed by Lazarus and Verroust [LV99]. This method, as Dong’s, has one drawback: multiple neighboring local extrema can be found in almost flat regions. We propose a solution to cluster these extrema, which will be discussed in section 2.2. For each extremum vertex x (be it manually or automatically chosen), the value $f(x)$ is set to the length of the shortest path from the source vertex, as computed by Dijkstra’s algorithm (it could also be set to the value given by Dong’s method when using this algorithm). Doing so, the harmonic function f can be seen as a smooth approximated distance to the source vertex over the mesh.

2.1.2. Solving Laplace’s equation

Once the boundary conditions to Laplace’s equation are set, the system (1) is solved using a classical finite elements method of $P1$ type (the function f , defined for each vertex, is linearly interpolated inside each triangle). Since the assembled matrix is very sparse, computation can be done very efficiently (e.g. using the SuperLU solver [DEG*99]).

2.1.3. Generating the graph

The Reeb graph of f is then computed using Cole-McLaughlin’s algorithm [CMEH*03]. This algorithm requires f to be a Morse function: this basically means that two

neighboring critical points should have two different values for f . To ensure this property, we check if all vertices on the mesh have different values. If several vertices x_1, \dots, x_k have the same value $f(x_1) = \dots = f(x_k)$, we order them and change their values slightly.

2.2. Graph filtering

2.2.1. Recovering the shape’s symmetries

Even if the model is symmetric, Cole-McLaughlin’s algorithm may generate a non-symmetric graph, because the source vertex may not be located exactly on the symmetry plane or axis. We propose here a simple way to recover these symmetries.

Each node n of the graph G is assigned with the value $f(x)$, where x is the critical vertex on the surface corresponding to n . Now, let us give weights to the edges of G . Let (n_1, n_2) be an edge of G . (n_1, n_2) is balanced by the following weight:

$$w(n_1, n_2) = \frac{|f(n_1) - f(n_2)|}{|\max_{n \in G} f(n) - \min_{n \in G} f(n)|} \quad (2)$$

Considering f as an approximated distance to the source vertex over the mesh (see section 2.1.1), $w(n_1, n_2)$ represents the normalized difference between the distance to the source vertex of two “topologically close” vertices. If $w(n_1, n_2)$ is small, this means that the corresponding vertices x_1 and x_2 are approximately at the same distance to the source vertex, and are also located in the same topological area (they are not necessarily geometrically close to each other). Thus, in order to recover the shape’s symmetries, we propose to filter the graph by collapsing every *internal* edge with a weight lower than a given threshold t_1 . We do not collapse edges containing a leaf node, since this could remove small features.

Notice that we can recover not only *geometrical* symmetries of the model, but also *morphological* ones: for instance, the octopus model of Figure 4 is not symmetric, geometrically speaking, because its tentacles are not in the same position; it can however be regarded as morphologically symmetric, because these tentacles have the same size and are regularly placed around a symmetry axis. As shown on the same model, we can recover not only symmetries *w.r.t.* a plane but also symmetries *w.r.t.* an axis.

2.2.2. Removing irrelevant extrema

As explained in section 2.1.1, it may happen that too many extremum vertices are computed. In order to remove irrelevant extrema, since extrema correspond exactly to the leaf nodes of the graph, we propose to remove the *external* edges (that is to say edges containing a leaf node) with a weight lower than a given threshold t_2 , *together with their nodes*. However, these edges should be removed carefully (see Figure 5): in order to avoid extra deletion of edges, they should first be ordered by increasing weight.

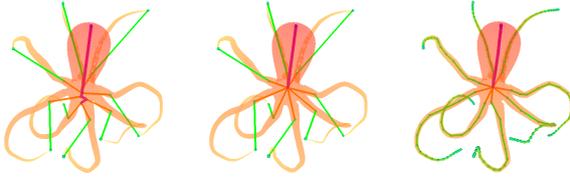


Figure 4: Left: non-symmetric graph obtained from a model containing a symmetry. Middle: the same graph after filtering ($t_1 = 0.007$). Right: refined harmonic skeleton.

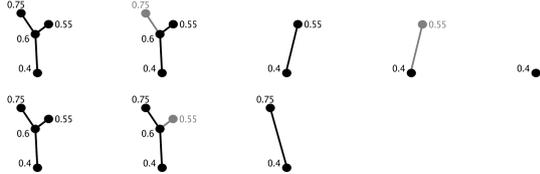


Figure 5: Deletion of edges whose weight is lower than or equal to $t_2 = 0.15$. Top: without weight ordering. Bottom: with weight ordering.

Both thresholds t_1 and t_2 can be set by the user, but they can also be computed. Indeed, unwanted edges usually have very small weights compared to the others', since they can be seen as noise while the others are associated with feature sizes. Thus, a statistical analysis upon all the edge's weights can help to set these parameters.

3. Harmonic skeleton

The harmonic graph gives the topological structure of the model. This is not enough to get an animation skeleton: we need to add 3D coordinates to its nodes, which will represent the joints of the skeleton; we may also need to refine the graph. Previous methods constructed the skeleton from a Reeb graph using only topological and geometrical information from the model, which is often not sufficient for realistic animation. We propose to take benefit from anatomical information to design the skeleton; this information will be recovered from the harmonic graph, knowing that the source vertex was chosen on the head of the character. In this section, we explain how to detect symmetries of the model's morphology on the graph and propose a skeleton in general case. In the next section, we show how to improve this general skeleton in the case of biped and quadruped characters.

3.1. Symmetry axis detection

We suppose here that the character's morphology is symmetric. This is often the case: typically, the model has two or four legs, two ears, and the head and the tail (if it exists) are centered with respect to the legs. Thus, the harmonic graph should also be symmetric with respect to an axis (or a node, but a node can be considered as a degenerate case of an axis).

We propose here a heuristic to recover this symmetry axis starting from the source node, which is located on this axis since its corresponding vertex is supposed to be on the head of the character.

Finding symmetries on a graph is a NP-complete problem; that is why we must make some hypotheses about the graph to get an efficient algorithm. Several restrictions have been proposed in the graph theory community [DeF99]; we describe here a simple iterative algorithm based on the 3 following assumptions:

1. the source node is located on the symmetry axis;
2. the harmonic graph is actually a tree, i.e. it does not contain any cycle;
3. two subtrees are isomorphic if they have the same depth and if their root nodes have the same degree (that is to say, the same number of child nodes).

The two last hypotheses are relevant for our application, since harmonic graphs are usually simple: they are made of one node for the head, one node for each leg, possibly one extremal node for the tail, for each ear and/or each wing and/or each finger, and that is usually all.

We use n_0 to denote the source node of the harmonic graph, and $(n_0, n_1) = e_0$ as its incident edge: e_0 is on the symmetry axis. n and n' denote nodes of the harmonic graph, whereas e denotes an edge. Our algorithm proceeds as follows:

- $e = (n, n') \leftarrow e_0 = (n_0, n_1)$
- **while** $e \neq NULL$ **loop**
 - add e to the symmetry axis;
 - let $e_1 = (n', n'_1), e_2 = (n', n'_2), \dots, e_k = (n', n'_k)$ be the incident edges to n' , excepting e ;
 - for each node n'_j , let T_j be the subtree of G whose root node is n'_j and which does not contain n' ;
 - store the T_j into sets $\mathcal{S}_1, \dots, \mathcal{S}_l$ of isomorphic trees, according to assumption number 3;
 - **if** $\exists! \mathcal{S}_i$ which contains only one tree T_i **then** $e \leftarrow e_i = (n', n'_i)$
 - **else** $e \leftarrow NULL$
 - **end if**
- **end loop**

Figure 6 shows the successive steps of the algorithm on an example. It adds edges to the symmetry axis iteratively, discarding subtrees of the harmonic graph that are symmetric *w.r.t.* the computed part of the axis. Note that if several sets \mathcal{S}_i containing one tree exist at the same time the algorithm stops, because it cannot tell which tree has its root on the axis. This algorithm can be applied not only to the harmonic graph G , but also to subtrees of G , in order to find non-principal symmetries. We can thereby obtain a hierarchy of symmetries, like [SKS06].

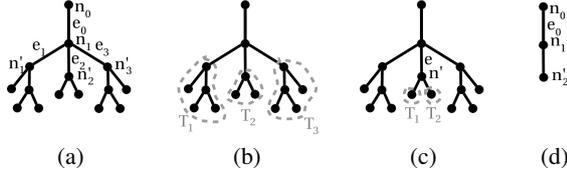


Figure 6: Symmetry axis detection. (a) Initialization (b) First step: T_1 and T_3 are isomorphic trees, and T_2 is not isomorphic to any other known tree: $e_2 = (n_1, n_2)$ is on the symmetry axis. (c) Second step: T_1 and T_2 are isomorphic, there is no candidate tree to process further on, so the algorithm stops. (d) Detected symmetry axis.

3.2. Simple embedding

Finding an appropriate embedding for each node of the harmonic graph is not a trivial task: even if each extremal node can be embedded onto the corresponding vertex on the mesh, this is not always possible for internal nodes, since they may have more than one corresponding vertex (Figure 7 (a)). Moreover, it is often more relevant to embed an internal node *inside* the model than *on* the surface. Before giving details about how internal nodes will be embedded in \mathbb{R}^3 , we should explain how regular nodes (nodes with exactly two incident edges) that will be inserted to the graph will be embedded.

Let u be a regular value of f (that is to say a non-critical value), let $f^{-1}(u)$ be its level set, and let C be a connected component of $f^{-1}(u)$. C is a simple closed curve made of segments whose endpoints $p_1, p_2, \dots, p_k, p_{k+1} = p_1$ intersect the edges of the mesh. We define the *center of mass* of C as the center of mass of these segments [LV99]:

$$\text{center}(C) = \frac{\sum_{i=1}^k \|p_i p_{i+1}\| \frac{p_i + p_{i+1}}{2}}{\sum_{i=1}^k \|p_i p_{i+1}\|} \quad (3)$$

We embed a regular node with value $f(u)$ onto the center of its associated connected component C . This choice is more relevant than the center of mass of the points p_i , since the result is less dependent on the surface's discretization level.

Now, here is the algorithm we propose in order to embed an internal node n :

1. split each incident edge (n, n_i) to n in two, by inserting a new node n'_i ;
2. assign the value $f(n) + \varepsilon$ or $f(n) - \varepsilon$ to each n'_i , depending whether $f(n) < f(n_i)$ or $f(n) > f(n_i)$ (ε should be a small scalar value, lower than the lowest weight among the graph's edges);
3. since each node n'_i is a regular node, embed it as explained before;
4. determine which nodes among these are on the symmetry axis:

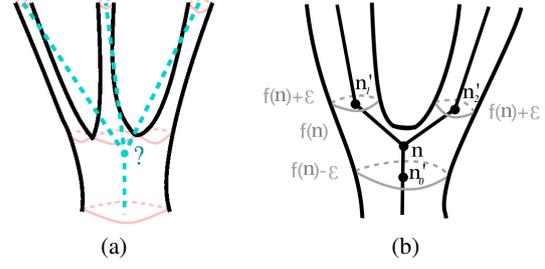


Figure 7: (a) Some nodes may have more than one corresponding vertex on the mesh. (b) Added regular nodes and possible embedding for internal nodes with 3 incident edges.

- if there is none, embed n onto the center of mass of the n'_i 's embeddings;
- if there is one, embed n onto the embedding of this node n'_k ;
- if there are two (three or more is not possible), choose one of them, embed n onto its embedding and remove the other node from the graph;

5. finally, freeze the new edges (n, n'_i) : this means that if one node's embedding is subsequently modified, the other should be modified the same way.

Figure 7 (b) shows the possible embeddings for internal nodes with 3 incident edges. Freezing edges has an important meaning: some degrees of freedom are removed for some joints of our animation skeleton, and freezing allows us to mirror the effect of bones such as the clavicle or the pelvis.

3.3. Joint hierarchy

Embedding the graph's nodes in \mathbb{R}^3 is not sufficient to get an applicable animation skeleton: we should also define a joint hierarchy. This can easily be done using the detected symmetry axis on the harmonic graph: the base joint can correspond to any node on this axis, then other joints recursively come from it. Common base joint choices include the head, that is to say the source joint, a node on the symmetry axis with a mean value for f , or the pelvis, which is the last node on the symmetry axis with at least three incident edges.

Once we have set up this hierarchy, we can use our embedded and augmented harmonic graph, which we call *harmonic skeleton*, as animation skeleton: nodes will be used as joints. The direction of the symmetry axis (or more precisely, of its embedding) can be used to set up the initial orientation of each joint. Moreover, additional joints can be added in a very simple way as regular nodes on the graph, with the embedding described in section 3.2. The value for f corresponding to a new joint, and hence its exact location, can be either set up by the user, or computed as the mean value between the two values of the edge's nodes (this is our default choice), or even computed so that the joint fits some geometrical feature

(e.g. local minimum of the gaussian curvature, as proposed by [TVDO6]).

4. Adapted embedding for bipeds and quadrupeds

In this section, we explain how the previously computed skeleton can be modified in order to better fit biped or quadruped mammals. Equivalent heuristics can be developed for other kinds of characters. These heuristics rely on semantic information about the model's anatomy associated to each joint of the skeleton, which can be recovered since the source joint corresponds to the head of the character and all skeleton extrema are known (see Figure 8 (a)). First, we propose a heuristic to check if the skeleton corresponds to a biped or a quadruped model.

4.1. Biped/quadruped discrimination

In the case of a biped or quadruped character, the computed harmonic graph should be as described in Figure 8 (a): the symmetry axis should have at least 2 nodes with at least 3 incident edges. The last of these nodes P matches the pelvis, and the previous one S matches the shoulders (we can have others, matching for example the ears). Since P and S have 3 or 4 incident edges, we know from section 3.2 that the ones not on the symmetry axis have been frozen: let P_1, P_2, S_1 and S_2 be their other endpoints; these nodes correspond to the beginning of the leg bones (when the subtree corresponding to the tail is isomorphic to the back legs, P_1 and P_2 are chosen among the three children of P so that $|SP.(PP_1 \times PP_2)|$ is maximum). We can now define 3 unit vectors: the spine direction $Spine = \frac{SP}{\|SP\|}$, a unit vector N_P normal to the triangle PP_1P_2 and a unit vector N_S normal to the triangle SS_1S_2 . Since edges PP_1, PP_2, SS_1 and SS_2 are frozen with $f(P_1) \approx f(P_2) \approx f(P)$ and $f(S_1) \approx f(S_2) \approx f(S)$, we say that the model is a quadruped if $|Spine.N_P| \approx 1$ and $|Spine.N_S| \approx 1$, and a biped if $|Spine.N_P| \approx 0$ and $|Spine.N_S| \approx 0$ (see Figure 8 (b) and (c)). In the other cases, we cannot conclude.

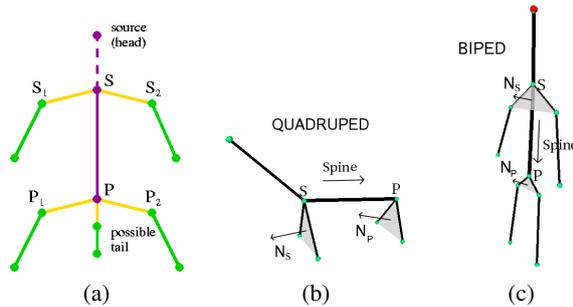


Figure 8: (a) Minimal harmonic skeleton for a biped or a quadruped model. The symmetry axis is colored in purple and frozen edges are colored in orange. (b,c) Spine, N_P and N_S vectors for quadrupeds and bipeds.

Actually, this heuristic is well-adapted for most quadrupeds, but not all. Indeed, vertebrate terrestrial

quadrupeds can be classified into two groups, according to the orientation of their leg bones (see Figure 9): in the case of amphibians these bones approximately lie in a transversal plane (plane with constant altitude), while in the case of most mammals they lie in a sagittal plane (orthogonal to S_1S_2 and P_1P_2). While our test is adequate for “sagittally oriented” quadrupeds, it can fail for amphibians, for which the result can be the same than for bipeds: $|Spine.N_P| \approx 0$ and $|Spine.N_S| \approx 0$.

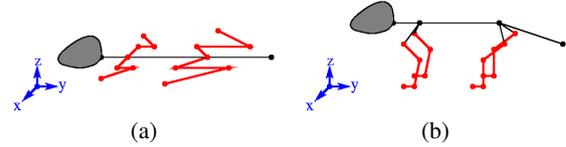


Figure 9: (a) Schematic skeleton of an amphibian: the leg bones are in a transversal plane ($z = cst$). (b) Mammal case: they are in a sagittal plane ($x = cst$).

4.2. Biped embedding

If the character has been detected as a biped, we propose a special refinement of the harmonic skeleton. This refinement starts with the addition of several nodes to the graph:

- the spine, that is to say the edge SP , is subdivided into 4;
- a new node N is inserted on the symmetry axis before S ;
- a new node J is inserted before N , and a new edge JM is added from J (M is a new extremum of the graph);
- each arm and each leg is subdivided into 3 edges;
- if there is a tail, it is subdivided into 4 edges.

The goal of this refinement is to match what would have created an artist. The nodes added to each arm will match elbows and wrists, while the nodes added to each leg will match knees and ankles; N will match the base of the neck, J the jaw and M the mouth. Notice that the source node and M do not match any real joint: these are in fact useful to better control the movement of the head and its size. We choose not to add edges for the rib cage, as it is not usually modeled for IK skeletons.

In order to shift some node embeddings and to embed the newly inserted nodes, we first give a reference frame to the model. This reference frame is defined by the previously introduced unit vector $Spine$, the unit vector $\frac{P_1P_2}{\|P_1P_2\|}$ and the unit vector $Spine \times \frac{P_1P_2}{\|P_1P_2\|}$, which gives the front-to-back (or back-to-front) direction. We can then embed newly inserted nodes, such as the nodes of the spine which can be slightly moved backward. To mimick what an artist would do, we have also chosen to unfreeze the SS_1 and SS_2 edges, and to embed S_1 and S_2 ahead of the embedding of S , in order to match clavicles. Regular nodes can be embedded either using a mean Euclidean position or a mean value for f w.r.t. the embeddings of their edge's endpoints, or fitting some geometric criterion, such as proposed by [TVDO6]. The last solution can be particularly adapted for neck and wrists, which match constrictions of the shape.

4.3. Quadruped embedding

Automatic animation skeleton generation is much less developed for four-footed animals than for bipeds. In order to refine the harmonic skeleton for parasagittally oriented quadrupeds, we based our work on the reference animation skeletons proposed by [RFDC05]. These IK skeletons were constructed by hand, from anatomical references [Cal75]. We add the same nodes to the harmonic graph as for bipeds, except that each front leg is subdivided into 5 edges, each back leg into 4 edges, and instead of having 2 edges between J and S (JN and NS), we have 5: the 4 added nodes will match the first, the second, the fourth and the seventh (which is the last) cervical vertebrae. We also subdivide the edge starting from the source node in 3; the first inserted node J' will match the jaw, while this time J will match the cranium. As for bipeds, M does not match any real joint and is useful to control the head's size and its movement. It will be put on top of the head of the character. We use the same reference frame as for biped embedding; here is how some of the joints are embedded: P is lifted up along the $Spine \times \frac{P_1 P_2}{\|P_1 P_2\|}$ direction from the simple embedding position (the center of its connected component for $f^{-1}(f(P))$) in order to be close to the back; nodes on SP are also lifted up, and so are S_1 , S_2 , P_1 and P_2 ; S is lifted up in order to match the pelvis' height; the first inserted nodes on each leg are moved along the $-Spine$ direction. We found that the best choice to embed the node J was near the neck constriction (actually a bit closer to the source joint); its value for f and exact location depends on the neck length. Finally, a simple solution for J' is along the $-Spine \times \frac{P_1 P_2}{\|P_1 P_2\|}$ direction from J , close to the chin.

5. Results and validation

Figures 2 and 10 to 12 show harmonic skeletons computed with our method. In these cases extrema have been selected by hand, because automatic computation of the extremal features can be quite slow. Thus, the threshold t_2 has not been used (it has been set to zero). No fine tuning of t_1 has been necessary: for almost all models, setting t_1 between 0.001 and 0.150 is sufficient. Except the selection of the extrema and t_1 , the entire process is automatic; no post-processing has been applied.

5.1. Biped and quadruped embeddings

Figure 11 shows the harmonic skeleton computed from a biped model, compared with a standard handmade skeleton (from Autodesk's Maya software). We have not modeled the rib cage, as explained before. As for the other models, the symmetry axis is colored in purple and frozen edges are colored in orange. Even though the graph is more complex than the minimal harmonic graph for a biped (Figure 8 (a)) because we decided to model the fingers, the symmetry axis has been correctly detected. Another biped skeleton is shown on the right of the figure. We have chosen to embed extremal nodes onto corresponding vertices on the mesh, but we could have easily embedded them inside the model instead, using

a close but regular value for f and the definition (3) of the center of a connected component.

Results on two quadruped models are shown on figures 2 and 12. The cat's tail is not considered as part of the symmetry axis, since its corresponding subtree on the harmonic graph is isomorphic to the back legs. Our algorithm provides animation skeletons close to the model's anatomy and to traditional IK skeletons. Nevertheless, some joints may need to be slightly displaced for better animation, particularly in the head. It is also noticeable that the very beginning of the tail is actually included in a frozen edge; this is correct since it corresponds to the first coccygeal vertebrae which are indeed attached to the sacrum [Cal75].

Our harmonic skeletons have been used for animation, as can be seen on Figure 1 and on the accompanying video.

5.2. Robustness

Figure 10 shows the robustness of the skeleton generation *w.r.t* the pose, mesh deformation and source vertex location.

Two different poses of the same character generate the same graph, with approximately the same values for f on each node, as long as the model is not stretched from one to the other. The reason is twofold: we are guaranteed that the extremal nodes correspond to the selected or computed extremal vertices, and f can be approximated as a distance *over the mesh* to the source vertex. Then, the embedding is most often the same since it does not depend on the leg orientation, for instance: it depends mostly on the computed reference frame, which is the same except if the back has been bended. It can also depends on the surface's local geometry, if we use constrictions to fix some joints such as the neck and wrists.

If the pose deformation is not isometric, we cannot be sure to get the same harmonic graph, from a theoretical point of view. However, stretching or shortening one leg in a homogeneous way does not change neither the graph nor its embedding, since for instance the ratio forearm length over arm length is not modified.

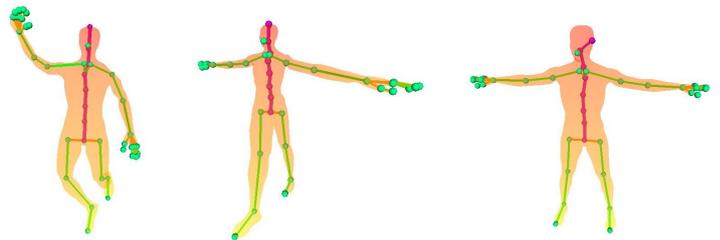


Figure 10: Robustness of the skeleton generation *w.r.t* the pose (left), mesh deformation (middle) and the source vertex location (right). Compare to Figure 11.

Our skeleton computation is also very robust *w.r.t* the

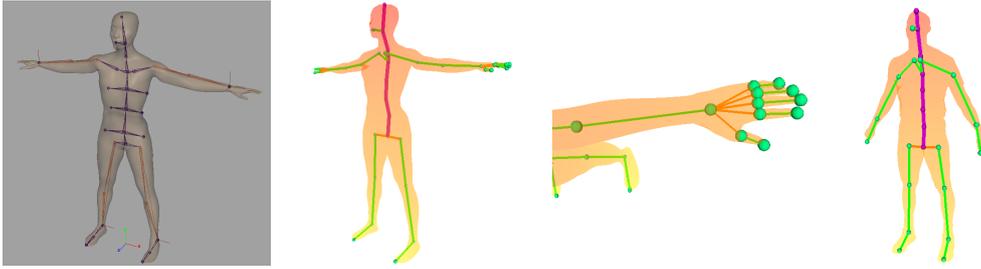


Figure 11: Comparison on a standard biped model, *MayaHuman*, between a standard IK skeleton (left) and our harmonic skeleton (middle left). Middle right: hand close-up; right: harmonic skeleton for another biped model, *MaleWB*.



Figure 12: Comparison on a horse model between several methods. Images are taken from the papers; the right image is taken from Wikipedia.

source vertex location, as long as it is chosen on the head: even if it is not on the character’s symmetry plane, the symmetry axis of the harmonic graph is recovered; then, since the embedding we propose does not depend on the source vertex location, it does not change.

5.3. Quantitative validation

To prove that our approach is useful, we have carried out a quantitative validation of our results: since [RFDC05] introduced parameters to define quadruped’s skeletons (back and front leg height – or similarly spine tilt – and neck length, normalized by the spine length), we compared their values between our skeletons and IK skeletons, handmade from anatomical reference. Results for 6 models are provided in Table 1; in most cases our embedding of nodes S and J is correct, resulting in similar values between harmonic skeletons and IK skeletons for front leg height and neck length. The location of the pelvis is sometimes low in our harmonic skeletons, which explains the greater difference for back leg height.

5.4. Computation time

The Table 2 gives computation times for 5 models on a standard PC with a 2.4 GHz Pentium 4 processor. Even for a dense mesh, our algorithm generates the skeleton in less than 1 minute. The memory requirement is also low: at most 350 MB for a model made of 300,000 faces, 1.5 MB for a model with 15,000 faces (including the storage of the mesh). Most of the time is spent on the harmonic function computation;

graph computation is then done in $O(n \log n)$ time for a mesh with n faces [CMEH*03], and embedding is done in nearly linear time because we only compute ray/mesh intersections for some joints in order to get their distance to the mesh, and the number of joints does not depend on the mesh’s complexity.

Mesh	Back leg		Front leg		Neck	
	Harmo.	IK	Harmo.	IK	Harmo.	IK
Cat	1.2	1.3	1.2	1.2	0.4	0.4
Cow	1.0	1.1	0.9	0.9	0.3	0.4
Dog	1.3	1.3	1.1	1.2	0.5	0.4
Elephant	1.4	1.6	1.4	1.4	0.3	0.3
Horse	1.3	1.7	1.4	1.6	0.7	1.0
Panther	1.0	1.1	0.9	1.0	0.4	0.5

Table 1: Parameter comparison between our harmonic skeletons and hand-built IK skeletons.

Mesh	Nb. faces	Graph	Embedding	Total
Cat	2,566	0.085	0.108	0.193
MayaHuman	14,118	0.634	0.139	0.773
Octopus	33,058	1.393	0.061	1.454
Horse	96,966	6.268	3.525	9.793
MaleWB	296,272	30.816	5.230	36.046

Table 2: Computation time (in seconds) for some meshes.

6. Conclusion

In this paper we have presented a fully automatic method to compute an animation skeleton from a 3D meshed model in

a few seconds after the selection of an initial point. In the case of most bipeds or quadrupeds, this skeleton fits the animation skeleton that would be hand-built by an expert starting from anatomical boards, and is thus adapted for realistic animation. The main idea is to construct the Reeb graph of a harmonic function, which gives the overall morphological structure of the model (especially its symmetry axis), then to refine and embed it using anatomical information. There are two main restrictions on the input mesh: it should be a triangulated 2-manifold (with or without boundary), and, in order to recover the symmetry axis of the shape's morphology, it should not have handles (otherwise the Reeb graph contains cycles). Although the method is fully automatic, the user can control the skeleton generation by tuning a few optional parameters. This tool has been designed both to help artists and to allow non-experts to quickly generate skeletons which can be used for realistic character animation. Computed skeletons can be edited and refined, for instance to add joints that correspond to wings or to the trunk of an elephant.

Given this skeleton generation process, we see three promising research directions. First, each vertex of the mesh is related to the joints of the skeleton, since we have given values for the harmonic function to the graph's nodes, and hence the skeleton's joints; these relations may be used to enhance skinning weights. Second, our semantic decomposition of the graph may also be used to define heuristics that give adapted skinning weights: weights may vary according to the meaning of neighboring joints. It may also help for automatic mesh segmentation into anatomically meaningful regions. Finally, even if not embedded to match the model's anatomy, the harmonic graph may be useful for other applications (e.g. shape matching), since its construction is robust and does not create unnecessary nodes.

Acknowledgments

The authors would like to thank Lionel Revéret for interesting discussions at the beginning of this work. The horse and MaleWB models are courtesy of Cyberware. The MayaHuman model is courtesy of Autodesk.

References

- [BL99] BLOOMENTHAL J., LIM C.: Skeletal methods of shape manipulation. In *Shape Modeling International* (1999).
- [Blu67] BLUM H.: A transformation for extracting new descriptors of shape. In *Symposium on Models for the Perception of Speech and Visual Form* (1967), pp. 362–380.
- [Cal75] CALDERON W.: *Animal Painting and Anatomy*. Dover, 1975.
- [CMEH*03] COLE-MCLAUGHLIN K., EDELSBRUNNER H., HARER J., NATARAJAN V., PASCUCCI V.: Loops in reeb graphs of 2-manifolds. In *Symposium on Computational Geometry* (2003), pp. 344–350.
- [CSYB05] CORNEA N., SILVER D., YUAN X., BALASUBRAMANIAN R.: Computing hierarchical curve-skeletons of 3d objects. *The Visual Computer* 21, 11 (2005), 945–955.
- [DeF99] DEFRAYSSEIX H.: An heuristic for graph symmetry detection. In *Symposium on Graph Drawing* (1999), pp. 276–285.
- [DEG*99] DEMMEL J., EISENSTAT S., GILBERT J., LI X., LIU J.: A supernodal approach to sparse partial pivoting. *SIAM Journal on Matrix Analysis and Applications* 20, 3 (1999), 720–755.
- [DKG05] DONG S., KIRCHNER S., GARLAND M.: Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Computer Aided Geometric Design, Special issue on Geometry Processing* 22, 5 (2005), 392–423.
- [DS06] DEY T., SUN J.: Defining and computing curve-skeletons with medial geodesic function. In *Symposium on Geometry Processing* (2006), pp. 143–152.
- [FK97] FOMENKO A., KUNII T.: *Topological Modeling for Visualization*. Springer-Verlag, 1997.
- [GS01] GAGVANI N., SILVER D.: Animating volumetric models. *Graphical Models* 63, 6 (2001), 443–458.
- [HKK01] HILAGA M., SHINAGAWA Y., KOMURA T., KUNII T.: Topology matching for fully automatic similarity estimation of 3d shapes. In *SIGGRAPH* (2001), pp. 203–212.
- [KT03] KATZ S., TAL A.: Hierarchical mesh decomposition using fuzzy clustering and cuts. In *SIGGRAPH* (2003).
- [LKA06] LIEN J., KEYSER J., AMATO N.: Simultaneous shape decomposition and skeletonization. In *ACM Symposium on Solid and Physical Modeling* (2006), pp. 219–228.
- [LV99] LAZARUS F., VERROUST A.: Level set diagrams of polyhedral objects. In *ACM Symposium on Solid Modeling* (1999).
- [LWM*03] LIU P., WU F., MA W., LIANG R., OUHYOUNG M.: Automatic animation skeleton construction using repulsive force field. In *Pacific Graphics* (2003), pp. 409–413.
- [NGH04] NI X., GARLAND M., HART J.: Fair morse functions for extracting the topological structure of a surface mesh. In *SIGGRAPH* (2004), pp. 613–622.
- [Ree46] REEB G.: Sur les points singuliers d'une forme de pfaff complètement intégrable ou d'une fonction numérique. *Comptes-Rendus de l'Académie des Sciences* 222 (1946), 847–849.
- [RFDC05] REVÉRET L., FAVREAU L., DEPRAZ C., CANI M.: Morphable model of quadruped skeletons for animating 3d animals. In *Symposium on Computer Animation* (2005).
- [SF01] STEINER D., FISCHER A.: Topology recognition of 3d closed freeform objects based on topological graphs. In *Pacific Graphics* (2001), pp. 82–88.
- [SKK91] SHINAGAWA Y., KUNII T., KERGOSIEN Y.: Surface coding based on morse theory. *IEEE Computer Graphics and Applications* 11, 5 (1991), 66–78.
- [SKS06] SIMARI P., KALOGERAKIS E., SINGH K.: Folding meshes: Hierarchical mesh segmentation based on planar symmetry. In *Symposium on Geometry Processing* (2006).
- [TVD06] TIERNY J., VANDEBORRE J., DAUDI M.: 3d mesh skeleton extraction using topological and geometrical analyses. In *Pacific Graphics* (2006), pp. 409–413.
- [WP02] WADE L., PARENT R.: Automated generation of control skeletons for use in animation. *The Visual Computer* 18 (2002).
- [ZMT05] ZHANG E., MISCHAIKOW K., TURK G.: Feature-based surface parameterization and texture mapping. *ACM Transactions on Graphics* 24, 1 (2005), 1–27.