Applications of digital topology for real-time markerless motion capture

Benjamin Raynal

UNIVERSITÉ PARIS-EST

UNIVERSITÉ PARIS-EST

ECOLE DOCTORALE MSTIC

Thèse pour obtenir le titre de **docteur de l'Université Paris-Est**
Spécialité : Informatique

Soutenue et présentée publiquement par

Benjamin RAYNAL

Sous la direction de Michel COUPRIE

Applications de la Topologie Discrète pour la Captation de Mouvement en Temps Réel et Sans Marqueurs

7 December 2010

Composition du jury :

Rapporteurs : Edmond BOYER
Luc BRUN

Examinateurs : Kálmán PALÁGYI
Hideo SAITO
Michel COUPRIE
Vincent NOZICK

To my beloved parents,
who always believed in me.

Title:

Applications of Digital Topology
For Real-Time Markerless Motion Capture

Abstract

This manuscript deals with the problem of markerless motion capture. An approach to this problem is model-based and is divided into two steps: an initialization step in which the initial pose is estimated, and a tracking which computes the current pose of the subject using information of previous ones. Classically, the initialization step is done manually, forbidding the possibility to be used online, or requires constraining actions of the subject.

We propose an automatic real-time markerless initialization step, that relies on topological information provided by skeletonization of a 3D reconstruction of the subject. This topological information is then represented as a tree, which is matched with another tree used as model description, in order to identify the different parts of the subject. In order to provide such a method, we propose some contributions in both digital topology and graph theory research fields.

As our method requires real-time computation, we first focus on the speed optimization of skeletonization methods, and on the design of new fast skeletonization schemes providing good results.

In order to efficiently match the tree representing the topological information with the tree describing the model, we propose new matching definitions and associated algorithms.

Finally, we study how to improve the robustness of our method by the use of innovative constraints in the model.

This manuscript ends by a study of the application of our method on several data sets, demonstrating its interesting properties: fast computation, robustness, and adaptability to any kind of subjects.

Keywords

markerless; **motion capture**; **digital topology**; **thinning**; **alignment**; **homeomorphism**; **betweenness**.

Titre:

Applications de la topologie discrète pour
la captation de mouvement en temps réel et sans marqueurs.

Résumé

Durant cette thèse, nous nous sommes intéressés à la problématique de la captation de mouvement sans marqueurs. Une approche classique est basée sur l'utilisation d'un modèle prédéfini du sujet, et est divisée en deux phases: celle d'initialisation, où la pose initiale du sujet est estimée, et celle de suivi, où la pose actuelle du sujet est estimée à partir des précédentes. Souvent, la phase d'initialisation est faite manuellement, rendant impossible l'utilisation en direct, ou nécessite des actions spécifiques du sujet.

Nous proposons une phase d'initialisation automatique et temps-réel, utilisant l'information topologique extraite par squelettisation d'une reconstruction 3D du sujet. Cette information est représentée sous forme d'arbre (arbre de données), qui est mis en correspondance avec un arbre utilisé comme modèle, afin d'identifier les différentes parties du sujet. Pour obtenir une telle méthode, nous apportons des contributions dans les domaines de la topologie discrète et de la théorie des graphes.

Comme notre méthode requiert le temps réel, nous nous intéressons d'abord à l'optimisation du temps de calcul des méthodes de squelettisation, ainsi qu'à l'élaboration de nouveaux algorithmes rapides fournissant de bons résultats.

Nous nous intéressons ensuite à la définition d'une mise en correspondance efficace entre l'arbre de données et celui décrivant le modèle.

Enfin, nous améliorons la robustesse de notre méthode en ajoutant des contraintes novatrices au modèle.

Nous terminons par l'application de notre méthode sur différents jeux de données, démontrant ses propriétés: rapidité, robustesse et adaptabilité à différents types de sujet.

Mots Clés

captation de mouvement sans marqueurs; **topologie discrète**; **squelettisation**; **alignement**; **homeomorphisme**; **interposition**.

Acknowledgements

First and foremost, I wish to thank my supervisors: Michel Couprie for his numerous and precious advices, his patience and all the other things which are too long to enumerate; Vincent Nozick, especially for his infectious enthusiasm, his support and his friendship, and many other things.

Over the three past years I have enjoyed working in the A3SI team, in a good mood and with very competent researchers. I wish to thank Venceslas, Gilles, Jean, Denis, Yukiko, Hugues, Laurent and Dror for their good advices and their attention. I specially thank my fellow PhD students, Franois-senpai, Patrice-senpai, Yohann, John, Fabrice, Anthony, Adrien, Camille, Olena (special thanks for you, for all the time you spent to read my thesis ;)) and Nadine for all the time spend together, for their support and their friendship.

From my point of view, the LIGM is a great lab, thanks to its awesome members, who taught me almost all I know concerning computer science, from the basics to my actual level. I wish to specially thanks Marc, Nicolas, Marie-Pierre, Eric, Etienne, Cyril, and Jean-Christophe for this reason, and Julien, Elsa and Nelly for their friendship.

In a more general way, I would like to thanks my Dad who transmits me his passion for the computers, my Mom who always encourages me to do what I wanted, my big brother who shows me the way of the research and my little sister, well, you know... for being my little sister. Special thanks to Florian and Damien, who believe in me and are my friends since more than twenty years.

Finally, I would like to thank my fiancée Celine, for her patience, her kind, her tenderness, and her love.

Contents

List of Figures

List of Tables

List of Algorithms

Part I

Introduction

Chapter 1

Global Introduction

"Notre nature est dans le mouvement, le repos entier est la mort."
("Our nature consists in motion; complete rest is death.")

Blaise Pascal.

1.1 Background

Motion capture consists in automatic estimation of the pose (i.e. relative position and orientation of each part of the subject), motion and actions of a subject, usually a full human being. It is an highly active research field since more than twenty years, notably due to its industrial applications in computer-animated feature films and Human-Computer Interfaces (HCI). For the latest, some specific properties are required:

- *Real-time computation*, in order to provide good interaction.
- The subject must not need to wear specific equipment (markers).

Usual markerless motion capture systems are composed by a set of cameras, providing the input information of the subject. Numerous markerless methods of motion capture use an *a priori model* describing the structure and shape of the subject whom motion has to be captured. Such methods are composed of at least two steps: the *initialization step*, where the initial pose and the parameters of the model are detected, and the *tracking step*, consisting of finding the current pose in function of the previous ones.

In many of such methods, the initialization step is done manually by the user, forbidding the possibility to be used for HCI purpose, or requires constraining actions or poses of the subject.

1.2 Contribution

In this thesis, we propose an automatic real-time markerless initialization step using a very simple a priori model, described by a tree (called *model tree*), which can be easily designed in order to use the method for any kind of subject, e.g. full human being, hand, or dog.

Our method uses topological information (the *skeleton*) extracted from a 3D reconstruction of the subject, obtained by using a multi-camera system. This topological information is then represented by a tree (called *data tree*), which will be matched with the model, in order to find the position of specific parts, e.g. head, torso, crotch, arms and legs in the case of full human being.

In order to reach our goal, we have done some contributions in different fields of research:

Digital Topology. As our method requires real-time computation, we have worked on optimizations of the speed of skeletonization algorithms and on design of new fast and efficient skeletonization schemes.

Graph Theory. The matching between the model tree and the data tree has to take into consideration some specific noises on the data tree, due to its acquisition. We have proposed a definition of matching that is adapted to our needs, and associated computationally efficient algorithms.

Motion Capture. Merging the results of our previous works, we provided an efficient automatic real-time markerless initialization step. We also introduced novel intuitive constraint definitions, which can be optionally added to the model definition, in order to improve the robustness of the method.

The structure of this manuscript is in four parts: the first one introduces the background and the backbone of our method, and the three others deal with our contributions and results in the three fields of research enumerated above, respectively.

Chapter 2

Motion Capture Overview

Automatic capture and analysis of articulated 3D subjects (e.g. humans, hands or animals) motion is an highly active research area since more than twenty years.

Motion capture presents issues in several kinds of applications:

3D models animation: motion capture is widely used for movies and video games characters animation. Usage of real actors movements to animate virtual avatars is faster, more accurate and realistic than doing it manually.

Human-computer interfaces (HCI): several usages can be done of motion capture in the field of HCIs, including domotic (control of TV or HIFI by hand moves), sign language understanding, or virtual reality system interactions.

Surveillance: motion capture can be used in surveillance field in order to detect suspect behaviors, e.g. fights or theft, or crowd analysis.

Medical analysis: motion capture can be used for medical purpose, in order to analyze gait problems or high level sportsmen performances.

Of course, all these applications do not require the same properties for motion capture. Animation and medical analysis request a very accurate motion capture and analysis in order to be usable, but it can be done offline (i.e. after the performance of the subject). On the other hand, HCI and surveillance applications need a real-time motion capture, in order to allow an immediate response for some given situations but the accuracy is less essential, and only requires to allow the understanding of the action or the approximate location of the subject parts.

There exists numerous different methods to perform motion capture. In the sequel of this chapter, we propose a non exhaustive state of the art on motion capture, focusing on the class of methods used in the thesis, and on their specificities. For exhaustive studies, the reader can refer to [116, 117, 140].

2.1 Overview of Motion Captures Systems

2.1.1 Acquisition Systems Using Markers

Several acquisition systems estimate motion of a subject by measuring the position and/or the orientation of physical objects (called *markers*), which are fixed on the subject at specific locations. Acquisition systems with markers are widely proposed as commercial solutions. Different classes of such system can be defined, in regard of the method used to measure marker positions:

- Mechanical systems (e.g. *Gypsy*™ system, see Figure 2.1a) estimate the motion by direct measurement of articulation angles, via potentiometers fixed on an exoskeleton. The main problem of such systems is the fragility of the skeleton, which can be easily broken.
- Magnetic systems (e.g. *MotionStar*™ and *Fastrak*™ systems) consist in generating an electromagnetic field then finding the position and orientation of markers (coils of electric wire) by their perturbation of the field. The main problem of such system is the sensitivity to interferences from metal objects of the environment.
- Inertial systems (e.g. *Colibri*™ and *Xsens*™ systems, see Figure 2.1b) use markers composed by accelerometers and gyroscopes. These sensors provide motion informations, as speed and rotation of the marker. Then, softwares are used in order to find the pose of the subject from these informations. A very famous inertial motion capture system is the *Wiimote*™. The main problem of such system is the lack of accuracy, in regard of other methods.
- Optical Systems can be classified in two categories:
  - Passive optical systems (e.g. *OptiTrack*™, see Figure 2.1c, and *Qualisys*™ systems) use cameras in order to track markers coated with a retroreflective material reflecting light generated near the cameras lens.
  - Active optical systems (e.g. *PhaseSpace*™ system) use markers powered to emit their own light (via a LED).

  The main problem of such systems is the sensibility to self occlusions: all the markers are not detected by all cameras, some of them being hidden by parts of the body.

These acquisition systems using markers have in common to be very expensive, due to the requested specific hardware.

2.1.2 Marker Free Optical Acquisition Systems

Contrary to other acquisition systems, marker free optical acquisition systems only require cameras. We can classify marker free methods in regard of the requested number of cameras:

FIGURE 2.1: Examples of systems using markers. (a) exoskeleton of *Gypsy*™ mechanical system. (b) markers of *Xsens*™ inertial system. (c) markers of *OptiTrack*™ passive optical system.

Monocular Methods. Pose estimation from only one video stream is a very complex problem, notably due to self occlusions of subject parts. This kind of method is mainly used for surveillance purpose, crowd study and hand gesture recognition.

Multi-view Methods. During the last twenty years, the increase of computer power has allowed the management of multiple synchronized cameras and fast computation of 3D representation of the subject. It led to the development of numerous multi-view methods, freed from the problem of self occlusions. A majority of these methods requires a static background and a prior camera calibration.

According to Menier et al. [111], most of the markerless methods fall into three categories:

- *Learning-based methods* [1, 6, 9, 58, 119, 156, 163, 171, 174] store a set of training examples whose poses are known and estimate pose by searching for training images similar to the input images. This category of methods requires larger storage memory space than others.
- *Model-free methods* [27, 36, 42, 43, 48, 104, 105, 161, 176] do not require any a priori knowledge and automatically recover articulated structures.
- *Model-based methods* use an a priori model which is tracked using image information. Moeslund et al. [116, 117] defined a general structure of such methods in four steps:
    1. *Initialization.* Ensuring that the system starts motion capture with a correct interpretation of the scene: initial subject pose, background images, model parameters are some of the possible informations acquired during this step.

2. *Tracking.* Using temporal information in order to find the current parameters of the model (e.g. position and orientation of the different parts) from the previous ones.
3. *Pose estimation.* Estimating the pose of the subject in function of the model parameters.
4. *Recognition.* Recognizing the actions, activities and behaviors of the subject.

Classically, tracking and pose estimation steps are merged in one step. Here, we consider that recognition step is a problem distinct from motion capture.

Since in this thesis, we exclusively deal with markerless multi-view model-based methods, we will focus on the main characteristics of this kind of method.

2.2 A priori Model Definitions

Numerous model definitions have been proposed in the literature. A wide majority of them includes a *kinematic structure* (also called *kinematic skeleton*) defined by:

- a set of segments with specified lengths, representing limbs,
- a set of joints linking segments, with specified degrees-of-freedom, representing articulations.

Although some methods only require a kinematic structure [35, 111], most of them require to link it with shape information.

A usual way to provide shape information is the use of geometric primitives, like ellipsoids [31], truncated elliptical cones [79–81], super-quadric [166, 167], or combinations of several kinds [45, 112–115].

Another usual way to provide shape information is to use a surface, which can be generated from geometric primitives [66, 139] (implicit surface), or an accurate mesh acquired in a prior step by a specific device [51, 76]. Notice that in some cases [3, 50], such meshes are used without kinematic skeletons.

In addition to kinematic and shape information, color information can be used, by texturing the mesh [11], or giving some additional color priors [112] (e.g. noticing that hands and head are the only parts to have the same color).

Figure 2.2 shows examples of the different model definitions.

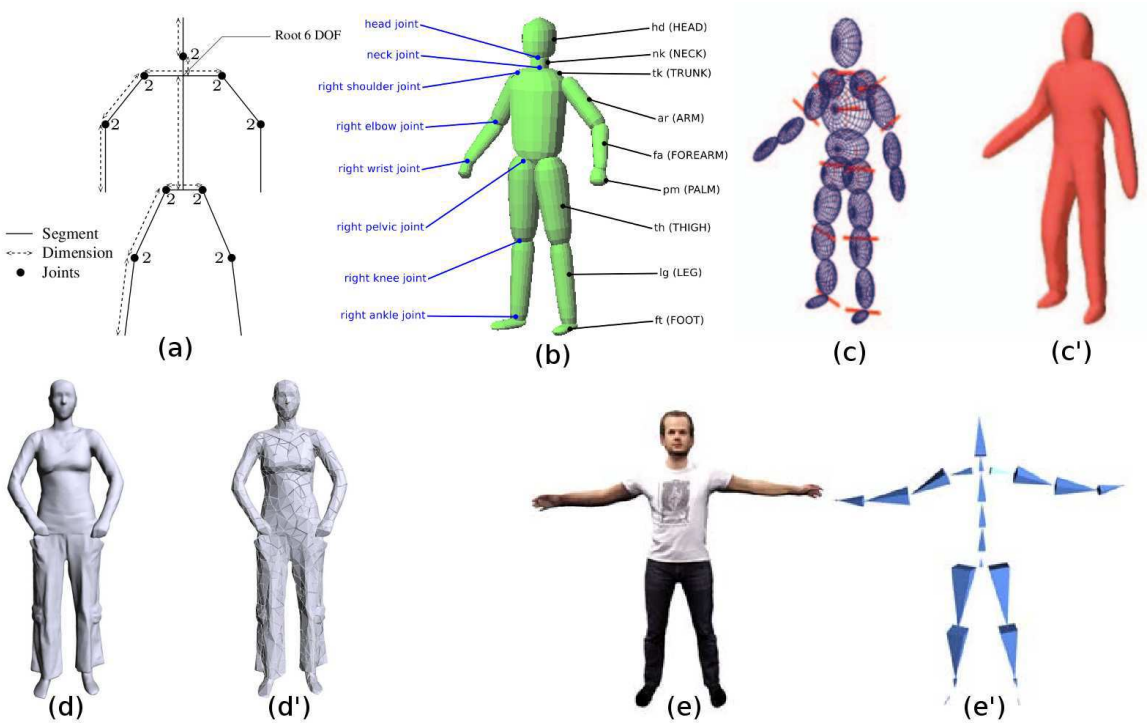


FIGURE 2.2: Examples of model definitions. (a) Kinematic model used by Menier et al. [111]; (b) Super-quadratic representation used by Sundaresan and Chellappa [166]; (c) Ellipsoid representation used by Horaud et al. [66] and (c') its associated implicit surface; (d) A surface scan of an actress and (d') corresponding mesh used by De Aguiar et al. [3]; (e) Textured mesh and (e') its underlying skeleton used by Ballan and Cortelazzo [11].

2.2.1 Adaptability of the Models

The *adaptability* of the method, i.e. the simplicity to modify the method in order to use it for another subject than the one initially targeted, obviously depends on the model definition. We define two levels of adaptability:

- *inner-class adaptability* is the simplicity to modify the model in order to use it for another subject of the same kind (e.g. two different human beings)
- *trans-class adaptability* is the simplicity to modify the model in order to use it for a subject of different kind (e.g. from full human body to hand motion capture)

Models using only kinematic structure, geometric primitives and implicit surfaces have a high inner-classes adaptability, model parameters being adjusted during initialization step, contrary to meshes, which have to be rebuild in a prior step, using specific devices and methods.

Concerning the trans-classes adaptability, almost all the methods can be used for any kind of subject, if an adapted model is provided (such methods are called *generic*). The exception is the case of color priors [112] and methods requesting a specific detection for each part of the

subject [112, 114, 115]. For all the others, the trans-classes adaptability is inversely proportional to the complexity of the model.

2.3 On the Use of Multi Camera Systems

Motion capture methods use information provided by multi-camera systems in many different ways, and can be classified into two categories: methods using 3D reconstruction of the subject and methods performing projection of the model in the images.

2.3.1 Projection in Images

Projection of the model in images can be performed by several methods, in order to use specific information which is easier and faster to extract in 2D than in 3D. Examples of information are:

- *silhouette* [50] and *contours of the subject* [45, 79–81], methods searching for a optimal fitting of the model in each of them.
- *connected components of a same color* [2], methods using them to facilitate the fitting.
- *optical flow* of contours [167], specific projected points [11] or shape descriptors [3, 51] (e.g.*SIFT* [47]) can be performed in order to track the motion in all the images independently, then reconstruct the 3D motion by triangulation.

2.3.2 3D Reconstruction of the Subject

Another approach consists of directly using 3D clues, considering 3D data resulting from multi-view modeling methods.

2.3.2.1 Stereo-Correlation

A famous binocular modeling method is the stereo-correlation, used in some methods [45, 139]: when cameras have a small baseline and are pointing in about the same direction, a part of the surface of the subject is visible in both images. It is thus possible to estimate the depth of this surface part by studying the disparity between corresponding points in the two images.

2.3.2.2 Visual Hulls

The most used multi-view modeling method is the visual hull, introduced by Laurentini [89]: intuitively, an object lies inside the volume generated by back-projecting its silhouette through the camera center (called *silhouette cone*). The intersection of the silhouette cones of all the cameras results in a volume called the *visual hull* (see Figure 2.3 for some illustrations), which is guaranteed to contain the object. Two kind of visual hulls can be distinguished:

- *surface visual hull*, used in [111], consists in only representing the surface of the visual hull by a mesh. Some methods, e.g. [66], use in addition the information from normals to surface elements.
- *volumetric visual hull*, used in [114, 115], consists in representing the visual hull volume by a subset of a voxel grid positioned at the intersection of all images cones: a voxel belongs to the visual hull if and only if the projection of it center in each image belong the corresponding silhouette. The voxels can be colorized [31, 35, 76, 112, 113] using ray tracing from the cameras, providing additional information. Volumetric visual hull can be computed in real-time, using GPU [63], or hierarchical carving [31].

Notice that some methods [166] use both visual hull and projection in images.

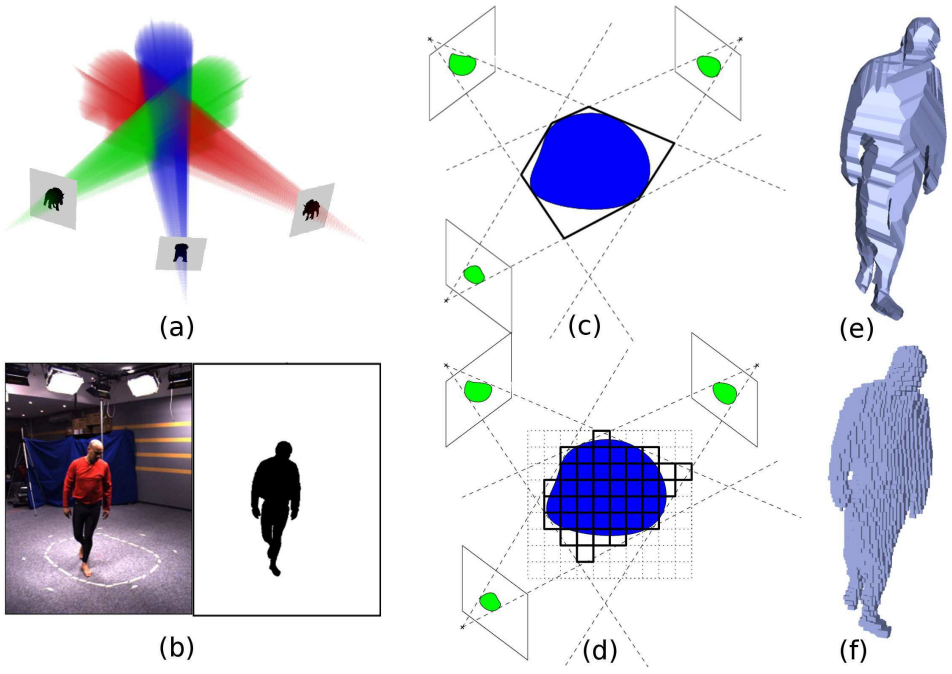


FIGURE 2.3: (a) Illustration of silhouette cones, reprinted from Matusick et al. paper [109]. (b) Example of input camera image and associated silhouette. (c) and (d) Illustrations of surface and volumetric visual hulls reconstructions, respectively: object is represented in blue, its silhouettes in green, silhouette cones by dashed lines, and result by bold lines. (e) and (f) surface and volumetric visual hulls of a same object, respectively. Figures (b), (e) and (f) are reprinted from Menier thesis [110].

2.4 Initialization Step of Model-Based Methods

A wide part of model-based motion capture methods found in the literature focus on the tracking and pose estimation step, and often omit the initialization step.

The initialization step is though essential, as providing the parameters of the model (limb lengths, dimensions of geometric primitives, color information, initial position and orientation of the different parts) which are primordial for an efficient tracking.

Different solutions have been proposed in the literature:

1. *manual initialization* [111] consists of manually setting the initial values of the model.
2. *limb by limb identification* [35] consists of asking to the subject to successively move only each of its limbs, in a specific order, in order to easily identify which part of subject information (in the case of Cheung et al. [35], a visual hull) is associated to each part of the model.
3. some methods require a *specific initial pose* [31, 35, 76], most often an "X"-pose, where the subject have spread legs and arms. It allows a fast identification of the model parameters, as the limb positions are already approximately set.
4. the initialization can be automatically done, using a *hierarchic fitting* [112–114]: due to its unique shape and size, the head is the easiest part to find and is identified first. Then, the torso is located using head position information, and finally limbs are detected using torso position and orientation information.

The method 1 is difficult to use online, for HCI purpose for example. It is even more a shame, as some methods using it (e.g [111]) are able to perform real-time tracking.

On the other hand, methods 2 and 3 can be used online, but such methods involve some constraints for the subject, which have to perform special actions in order to be tracked.

Finally, method 4 can be used online and does not require any specific action or pose from the subject, but is not generic: if the subject does not contain a part which is easy to detect, the initialization cannot be performed.

For our best knowledge, no real-time generic initialization step method has been proposed in the literature.

Chapter 3

Motivation

As seen in the previous chapter, numerous methods of marker-free 3D motion capture require a manual initialization of the subject pose or a constraining one, e.g. needing a special pose of the subject.

The aim of our work is to provide a pose initialization method (i.e. resulting only in the positions of specific parts of the subject) with several properties. First, our method has to be **generic**, in the sense it must be adaptable for a large set of subjects (full human body, hands, upper part of the human body, animals...) only by creating a very simple a priori model. Our method has to be **real-time**, in order to be usable with real-time tracking for online applications. Furthermore, we request our method to be **robust**, i.e. to work for a large set of subject poses, avoiding constraints for the subject. Finally, we consider that the input of the method is only a volumetric visual hull of the subject.

This chapter is composed of two sections. In the first one, we discuss the definition of the a priori model required by our method. Then, in the second section, we propose a brief overview of the pipeline of our method.

3.1 Model Definition

3.1.1 Constraints of Descriptors

The choice of the descriptors used for our a priori model definition is linked to different observations, according to the nature of the application and to our philosophy.

As we consider only the visual hull in input, we do not use any color information, contained in the camera images. No color prior is used in our method. Furthermore, some models have homogeneous color, e.g. hands, thus such information is not enough generic.

The quality of the 3D shape depends of the number of views for the visual hull reconstruction, and the respective positions of the cameras. In a multi-view system without enough cameras, or not well positioned ones, the visual hull can present important variations of limbs thickness in regard of the subject. Due to this fact, we do not use accurate shape representations, like geometric features.

As we want to create a method that should be usable with any kind of subject, we have to propose a representation allowing to easily describe the subject. We thus avoid the use of shape primitive description and laser scanned models.

Our method has to be able to find the pose of the subject in a maximum number of situations, in order to be able to perform the initialization step as often as possible. By this way, we can use our method even when the subject does not perform a specific action or pose (like it is usually the case), and we can check the tracking at closer intervals, increasing the accuracy.

The weakest constraint we found is that the topology of the shape and its salient geometric features have to be the same as for the model.

3.1.2 Model Description

Taking into consideration all the constraints enumerated in the previous section, we propose an a priori model providing a description of the topology and of the geometric features of the subject, using a tree representation.

The model is built as follows:

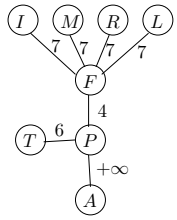
- Each vertex represents a characteristic part of the model. For example:
  - for the full human body: hands, feet, head, torso and crotch.
  - for the hand: the arm, the palm (in two parts), the end of the fingers.
- Two vertices are linked if the associated parts are adjacent in the model.
- To each edge is associated a weight, corresponding to the approximative distance (for a given distance unit used for all the weights in the graph) between the two linked parts.

Two kinds of models can be considered (see Figure 3.1 for examples):

- the *complete models*, for subjects fully contained in the 3D acquisition space, as in the case of full human body motion capture, for example.

- the *incomplete models*, which are part of a biggest shape, as in the case of hand pose estimation. In this case, a part of the shape intersects the border of the 3D acquisition space (e.g. the arm, in case of hand motion capture), and we represent this part in the model as an *infinite limb*, as we do not know it length.

Vertices:	Edges:
A // the arm	A-P=infinity
P // the palm	P-F=4
F // the finger root	P-T=6
T // the thumb	F-I=7
I // the index	F-M=7
M// the middle finger	F-R=7
R // the ring finger	F-L=7
L // the little finger	



Vertices:	Edges:
H // the head	H-T=4
T // the torso	T-C=10
C // the crotch	T-H1=12
A1// the first arm	T-H2=12
A2// the second arm	C-F1=16
F1// the first foot	C-F2=16
F2// the second foot	

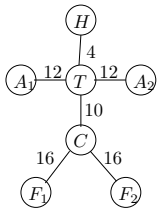


FIGURE 3.1: Description of two different models. On the Left, the hand model (incomplete model): notice that the edge between the arm and the palm is weighted by "infinity", as we do not know its length. On the right, the full human body model (complete model).

Notice that the use of a tree, instead of a graph, for the model description, forbid the representation of models containing holes (represented by a loop in a graph), but allow faster algorithms for the matching with data.

3.2 Initialization Method Overview

Now we have defined the inputs of our method: a visual hull representation of the subject and the a priori model.

Our method consists in extracting a descriptor from the visual hull (called the *data descriptor*) which is similar to the one used in the a priori model (called the *model descriptor*). The main difference is that the data descriptor has to contain some spatial information, where as that the model descriptor contains informations about the nature of the parts of the model.

Finding a matching between the two descriptors results in finding spatial information for each part of the model and by this way, the pose of the subject.

3.2.1 Extraction of Data Tree

First we have to find a way to extract topological and geometrical information from the visual hull, and to represent it by a tree. We solve this problem in two steps: first, we consider the skeleton of the visual hull. The skeleton is an efficient shape descriptor, representing in a compact form the topology and the main geometric features of a shape. It can be obtained by iteratively removing voxels without changing the topology (see Figure 3.2 for an example).

Then, we "translate" the skeleton into a tree:

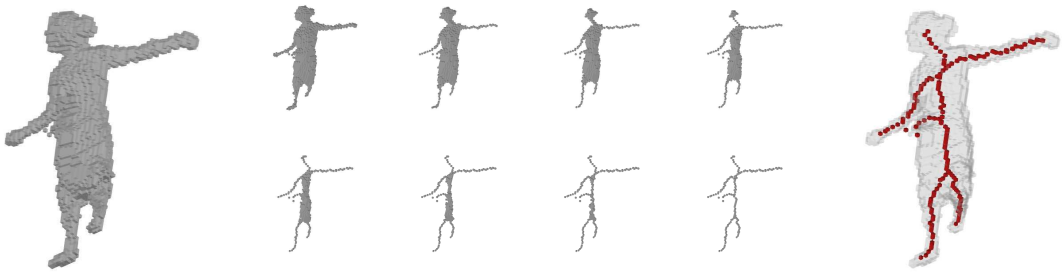


FIGURE 3.2: Example of skeletonization process. Left: a visual hull. Middle: different steps of skeletonization process. Right: resulting skeleton (in red).

- Each vertex represents a characteristic point of the skeleton: ending point or intersection point.
- Two vertices are linked if the associated points are adjacent to a same curve in the skeleton.
- To each edge is associated a weight, which is the number of voxels in the corresponding curve.

In the case of incomplete model, we have to consider that a branch ended by a point touching the border of acquisition space have an infinite length. See Figure 3.3 for some examples of data tree extractions.

More details about data tree extraction are given in Chapter 7.

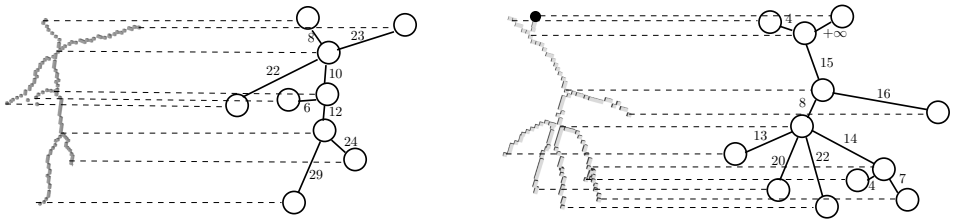


FIGURE 3.3: Examples of data tree extraction. Left: for a full human body subject. Right: for a hand subject. The black disk represents a point of the skeleton touching the border of the acquisition space.

3.2.2 Matching Data Tree with Model Tree

Once we have a tree description of the subject, we have to find a way to match it with the model tree. In the literature, numerous methods are proposed to find a matching between two trees. However, in our case we have to take into consideration several kind of noises in the data tree. Furthermore, in the literature, proposed methods are designed for special kind of trees, usually rooted, oriented, and with information on vertices instead of on edges, as in our case.

We have to define a new way to match our data tree with our model tree, corresponding to our constraints.

3.2.3 Pipeline and Main Difficulties of Our Method

The pipeline of our method is summarized in Figure 3.4.

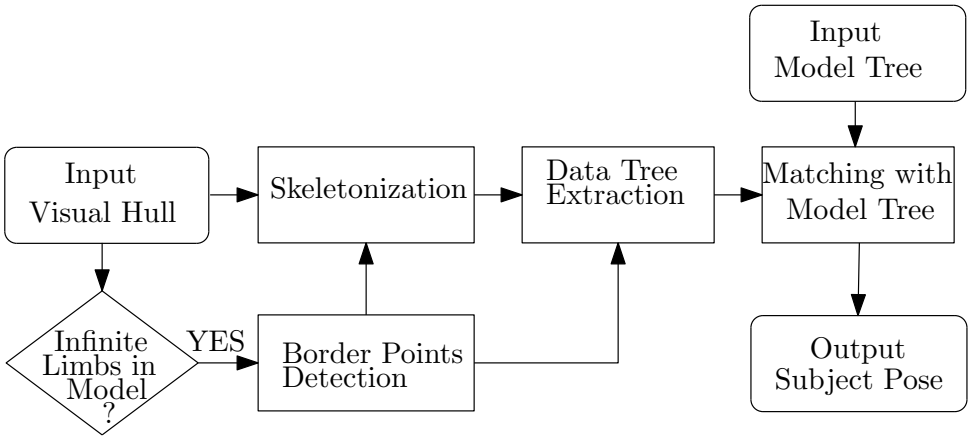


FIGURE 3.4: Pipeline of our method. Rounded rectangles represent input and output data. Other rectangles represent steps of the method.

We have now defined the different steps of our method, however, two main difficulties remain.

First, the skeletonization process has to be both speed efficient and robust to visual hull surface noise. To our best knowledge, no skeletonization algorithm satisfies these two properties. In the second part of this manuscript, we propose some solutions to speed up existent algorithms and a new algorithm providing good skeletons in real time.

The second problem is the matching with the model tree. As explained in Section 3.2.2, no method found in the literature is adapted to our special case of unrooted edge weighted trees, with special noises on the data tree. In the third part of this manuscript, we propose new kind of matching specially designed for our problem, and efficient algorithms for their computation.

Finally, after solving these two main difficulties, we propose in the last part of the manuscript to improve the robustness of the result by addition of intuitive constraints, and we study the results and the speed of our method.

Part II

Skeletonization Optimizations

Chapter 4

State of the Art of Skeletonization

The skeleton was originally defined by Blum [22] based on a " grass fire" analogy. Imagine a shape as a field covered by dry grass; if you set on fire the contour of the field, then the meeting points of the flame fronts would constitute the skeleton of the shape. In the continuous framework, this definition is equivalent to saying that the skeleton is the set of points which are centers of maximal balls (balls included in the object, and not strictly included in any other such ball) ([32]). Figure 4.1 shows an example of skeleton in continuous framework.

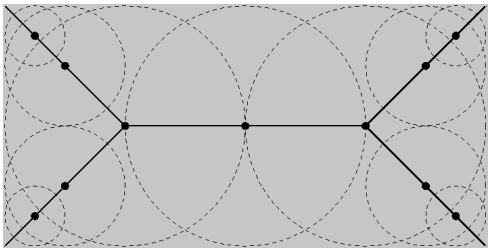


FIGURE 4.1: Example of skeleton in continuous framework. The shape (a rectangle) is represented in grey. The skeleton is represented by bold lines. Some maximal balls are represented by dashed lines, with their center represented by black points.

In 1969, Hilditch gave four properties that a skeleton in a bi-dimensional space should possess [64]. Adapted to the general case of n-dimensional skeletons, these properties are:

1. a skeleton should be homotopic to the original object,
2. a skeleton should be thin (should have lower dimension than the object),
3. a skeleton should be centered in the original object,
4. skeletonizing a skeleton should not change anything.

In the continuous framework, the set of centers of maximal balls, called the *medial axis*, satisfies these properties [91, 106]. In the discrete framework $\mathbb{Z}^n$, the discrete medial axis does not satisfy

two of these properties: it is not always homotopic to the original object, and it is not always thin (See Figure 4.2 for an example).

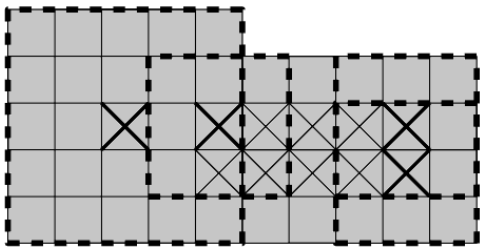


FIGURE 4.2: Example of medial axis in discrete framework. The shape is represented in grey. The medial axis is represented by crosses. Some maximal balls are represented by dashed lines, with their centers represented by bold crosses.

Various methods have now been developed for performing skeletonization of a discrete object. According to Palágyi [129], discrete skeletons can be computed using four types of methods: Voronoi-based transformations [26, 122], distance-based transformations [23, 172], general-field methods [5, 149] and thinning. In this thesis, we will focus only on thinning methods, as they are the most studied and well defined.

4.1 Thinning Theory

Thinning consists of iteratively removing points of the object without changing its topology. These points have specific characteristics and are called *simple points*. In this section, we will first provide usual definitions and notations of digital topology. Using these definitions, we will in a second time describe the characteristics of simple points.

4.1.1 Definitions and Notations

First we have to introduce some fundamental definitions and notations of digital topology.

In digital topology, the framework is the *discrete grid* $\mathbb{Z}^n$. In our case, we will only consider the cases $n = 2$ (bi-dimensional) and $n = 3$ (tri-dimensional). The *object* is represented by $X \subset \mathbb{Z}^n$, and its *complementary* $\mathbb{Z}^n \setminus X$ is denoted by $\bar{X}$.

A *point* $p \in \mathbb{Z}^n$ is defined by $(p_1, ..., p_n)$, with $p_i \in \mathbb{Z}$.

4.1.1.1 Neighborhood

The notion of *neighborhood* is central for digital topology. In the 2D case, two neighborhoods (see Figure 4.3) are considered:

- the *4-neighborhood* of p is the set $N_4(p) = \{q \in \mathbb{Z}^2; |q_1 - p_1| + |q_2 - p_2| \leq 1\}$.
- the *8-neighborhood* of p is the set $N_8(p) = \{q \in \mathbb{Z}^2; max(|q_1 - p_1|, |q_2 - p_2|) \leq 1\}$.

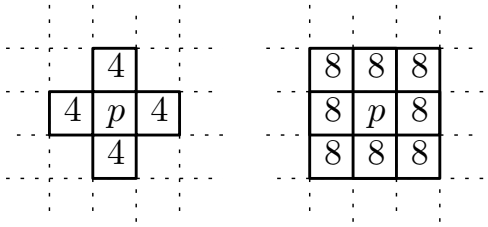


FIGURE 4.3: Left: 4-neighborhood of p represented by boxes labeled by 4 or p. Right: 8-neighborhood of p represented by boxes labeled by 8 or p.

In 3D case, three neighborhoods (see Figure 4.4) are considered:

- the *6-neighborhood* of p is the set $N_6(p) = \{q \in \mathbb{Z}^3; |q_1 - p_1| + |q_2 - p_2| + |q_3 - p_3| \leq 1\}$.
- the *26-neighborhood* of p is the set $N_{26}(p) = \{q \in \mathbb{Z}^3; max(|q_1 - p_1|, |q_2 - p_2|, |q_3 - p_3|) \leq 1\}$.
- the *18-neighborhood* of p is the set $N_{18}(p) = \{q \in N_{26}(p); |q_1 - p_1| + |q_2 - p_2| + |q_3 - p_3| \leq 2\}$.

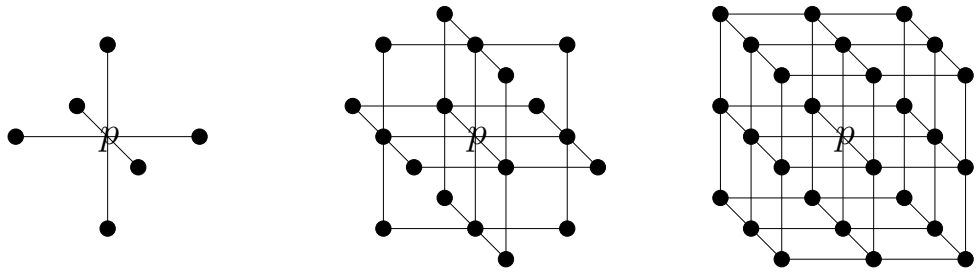


FIGURE 4.4: From the left to the right: 6-neighborhood, 18-neighborhood and 26-neighborhood of point p, represented by black disks and p.

For a k-neighborhood, we define $N_k^*(p) = N_k(p) \setminus \{p\}$.

4.1.1.2 Connectivity

From the definition of neighborhood we can propose the definition of connectivity.

Let p be an element of X, we define the *k-connected component of X containing p*, denoted by $C_k(p, X)$, as the maximal subset of X containing p, such that for all points $a \in C_k(p, X)$, there exists a sequence of points of $\langle p_0, ..., p_n \rangle$ such:

- $p_0 = p$ and $p_n = a$,
- $\forall i \in \{0, ..., n\}, p_i \in C_k(p, X)$,

- $\forall i \in \{1, ..., n\}, p_{i-1}$ is in the k-neighborhood of p_i.

The set of all k-connected components of X is denoted by $C_k(X)$. We can notice that the union of all the elements of $C_k(X)$ is equal to X, and the intersection of two different elements of $C_k(X)$ is always the empty set.

A subset Y of $\mathbb{Z}^n$ is *k-adjacent* to a point $p \in \mathbb{Z}^n$ if $Y \cap N_k^*(p) \neq \emptyset$. The set of all k-connected components of X which are k-adjacent to a point p is denoted by $C_k^p(X)$.

When we consider the k-connectivity of X, it is mandatory to consider a different $\bar{k}$-connectivity for $\bar{X}$ [82]. In bi-dimensional space, if $k = 4$, then $\bar{k} = 8$, and inversely. In tri-dimensional space, if $k = 6$, then $\bar{k} = 26$ and inversely. This rule is necessary in order to retrieve some important topological properties such as the Jordan theorem.

4.1.1.3 Connectivity Numbers

In order to provide local description of simple points and other characteristic points, we have to introduce the notion of connectivity numbers.

In the 2D case, let $X \subseteq \mathbb{Z}^2$ and $p \in \mathbb{Z}^2$. For $k \in \{4, 8\}$ the connectivity number $T_k(p, X)$ is defined by:

$$T_k(p, X) = |C_k^p(N_8^*(p) \cap X)|$$

In the 3D case, the definition of connectivity numbers lies on the notion of *geodesic neighborhood*. Let $X \subseteq \mathbb{Z}^3$ and $p \in \mathbb{Z}^3$. The *t-order k-geodesic neighborhood of p in X* is the set $N_k^t(p, X)$ recursively defined by:

- $N_k^1(p, X) = N_k^*(p) \cap X$
- $N_k^t(p, X) = \bigcup\{N_k(q) \cap N_{26}^*(p) \cap X, q \in N_k^{t-1}(p, X)\}$

The geodesic neighborhoods $G_k(p, X)$ are defined by: $G_6(p, X) = N_6^2(p, X)$ and $G_{26}(p, X) = N_{26}^1(p, X)$.

We can now define the connectivity numbers in 3D, for $k \in \{6, 26\}$ as:

$$T_k(p, X) = |C_k(G_k(p, X))|$$

.

4.1.2 Simple points

4.1.2.1 Simple Points in 2D

Intuitively, a point is simple if it can be removed from an object without changing its topology. In the digital topology framework, the topology of an object depends on the chosen connectivity; for this reason, when considering a k-connected object, we will talk about k-simple points. The notion of simple point is central for homotopic thinning in the digital framework: a skeleton is obtained by iteratively removing simple points from an object.

According to [88], in the 60s, 2D simple points were characterized based on connectivity: a point p is k-simple for an object X if the removal of p does not change the number of k-connected components of X nor the number of $\overline{k}$-connected components of $\overline{X}$ [25, 56]. This definition does not lead to efficient algorithms : indeed, in order to test if a single point is simple, it requires to scan the whole object in order to enumerate its connected components. Fortunately, local characterization of deletable points in 2D began to appear in the mid 60s [64, 147, 150, 184].

All these works established that, in order to decide whether a point is deletable or not, it is only necessary to look at the configuration of the point's neighbourhood (no need to count the number of connected components of the whole object). Consequently, in 2D, deciding if a point is simple can be done in constant time.

Proposition 1. *Let $X \subset \mathbb{Z}^2$, and $p \in X$. If $T_k(x, X) = 1$ and $T_{\overline{k}}(x, \overline{X}) = 1$, then p is k-simple for X.*

4.1.2.2 Simple Points in 3D

In 3D, the removal of a point may not only change the number of connected components of the object, but may also change the tunnels of the object (see Figure 4.5 for an example). As in the 2D case, 3D simple points can be locally characterized [118]. Further work on 3D simple points have established that only connectivity of X and $\overline{X}$ is sufficient in order to characterize 3D simple points [15, 19, 102, 151, 152]. As in 2D, deciding if a point is simple can be done in constant time in 3D. Bertrand and Malandain propose a local definition of simple points using connectivity numbers T_6 and T_{26}:

Proposition 2. *[19] Let $X \subseteq \mathbb{Z}^3$ and $x \in X$. The point x is k-simple for X iff $T_k(x, X) = 1$ and $T_{\overline{k}}(x, \overline{X}) = 1$.*

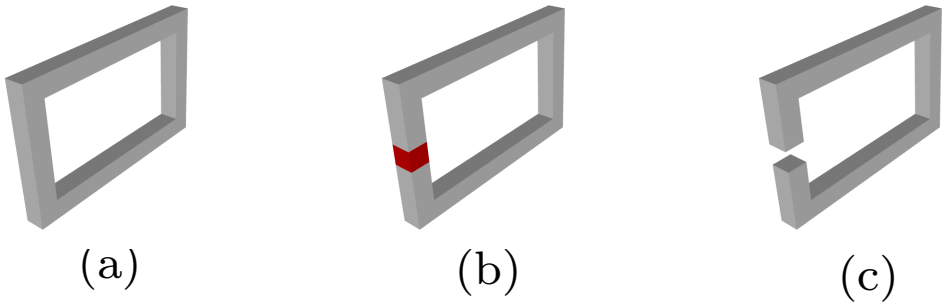


FIGURE 4.5: (a) A shape with a tunnel. (b) Deletion of the red point. (c) The shape has no tunnel.

4.1.2.3 Simple Points in Higher Dimensions

Studies of simple points in 4-dimensions have also been achieved, leading once more to a local characterization of such points [40, 85]. Thanks to these works, characterization of simple points in 4D can be done again in constant time.

4.2 Different Kind of Skeletons

In this section, we will review the different kinds of results which can be expected from a thinning process. The design and the constraints of the thinning algorithm will obviously depend on the desired result.

4.2.1 Ultimate Skeleton

Let $X \in \mathbb{Z}^n$ and $Y \in \mathbb{Z}^n$ be two objects. We say that X and Y are *homotopic* if we can transform X in Y by iteratively adding and removing simple points. We say that Y is *lower homotopic* to X if Y can be obtained from X by iteratively removing simple points.

Let $X \in \mathbb{Z}^n$ be an object. A subset $Y \subseteq X$ is an *ultimate skeleton* of X if:

- Y is lower homotopic to X
- for any point p of Y, p is not simple for Y.

Ultimate skeleton can be used to study the topology of the object, but possibly important geometric features informations are lost (see Fig. 4.6,(a) and (c)).

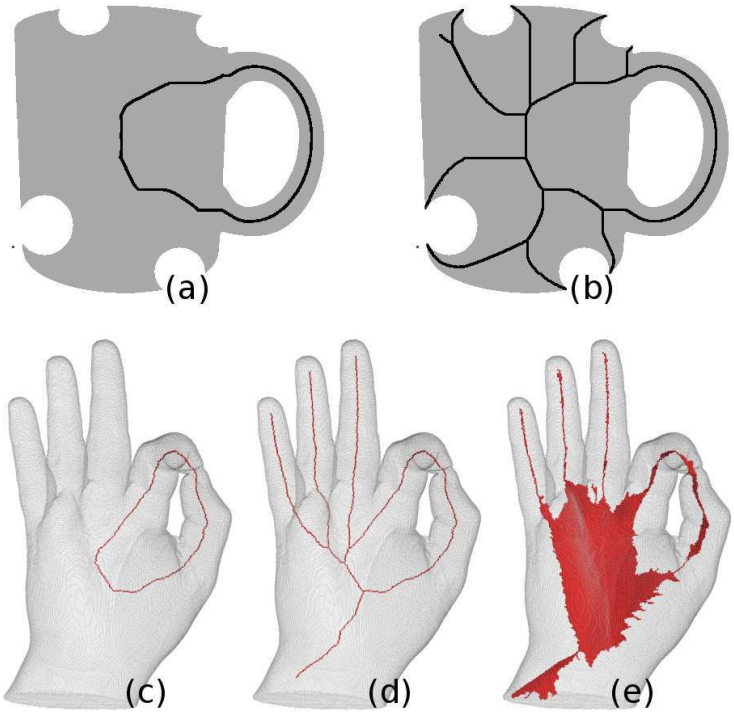


FIGURE 4.6: Examples of the different kinds of skeleton. Top, 2D skeletons: (a) ultimate skeleton, (b) curvilinear skeleton. Bottom, 3D skeletons: (c) ultimate skeleton, (d) curvilinear skeleton, (e) surface skeleton.

4.2.2 Curvilinear and Surface Skeletons

Ultimate skeleton, providing only topological information about the shape, is not efficient as a shape descriptor. Some geometric information, like salient parts, have to be kept in the result of the thinning algorithm.

In 2D case, the skeleton has to be 1D or 0D, according to the Hilditch's criteria. A 1-dimensional skeleton, regardless of the dimension of the original object, is called a *curvilinear skeleton* (see Fig. 4.6,(b) and (d)).

In the 3D case, always according to the Hilditch's criteria, the skeleton can be 0D, 1D or 2D. A 2-dimensional skeleton is called a *surface skeleton*(see Fig. 4.6(e)).

These skeletons provide both topological and geometrical information about the initial shape, which summarize the general form of the object.

The choice of the skeleton dimension depends on the application. We can notice that a surface skeleton corresponds to the analogy of grass fire in 3D, and provides a more accurate descriptor of the initial object. On the other hand, the curvilinear skeleton is smaller than the surface skeleton, and the provided geometric information can be enough for some applications.

In order to obtain curvilinear or surface skeletons instead of ultimate skeleton, several approaches exist: a first one consists of forcing the preservation of medial axis points, another one consists of preserving some locally characterized points, as extremity points or isthmuses.

4.2.2.1 Extremity Points

Extremity points are simple points which have to be kept in the skeleton in order to preserve elongated geometric features.

In the case of curvilinear skeletons, the considered points are those which end a curve, and are called *curve extremities*. These points can be locally described:

- Let $X \subseteq \mathbb{Z}^n, p \in X$, p is a k-curve extremity iff $|N_k^* \cap X| = 1$.

In the case of surface skeletons, the considered points are those that can be found on "*surface borders*". To our knowledge, there is no consensus on the definition of surface border in digital topology.

In addition to this problem, the other one is the sensitivity of this approach to small variations. It results in skeletons containing more branches than needed to represent the geometric features. These additional uninteresting branches are called *spurious branches*.

4.2.2.2 Isthmus

In order to solve the problems mentioned above, another strategy has been recently developed by Bertrand and Couprie [18]. Instead of searching for simple points which have to be kept in order to preserve geometric features, we can search for non simple points having certain topological characteristics. These points are called *isthmuses*.

For a curvilinear skeleton, the considered isthmuses are the $1D - isthmuses$ (see Figure 4.7 for some examples). In the case of surface skeleton, the considered isthmuses are the $2D-isthmuses$.

The main advantage in regard of surface borders is that $2D$-isthmus are locally well defined.

In 2D, let $X \subseteq \mathbb{Z}^2, p \in X$, p is $1D$-isthmus iff $T_k(p, X) \geq 2$.

In 3D, let $X \subseteq \mathbb{Z}^3, p \in X$, p is $1D$-isthmus iff $T_k(p, X) \geq 2$. The point p is $2D$-isthmus iff $T_{\bar{k}}(p, \bar{X}) \geq 2$.

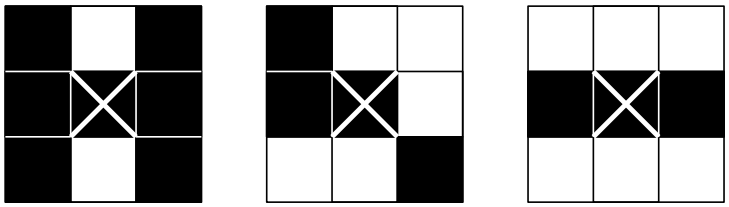


FIGURE 4.7: Examples of 1D isthmuses.

4.2.2.3 Symmetric versus Asymmetric Skeletons

Another distinction has to be done between the different kind of skeletons: symmetric and asymmetric ones. This notion of symmetry is due to interpretation in the digital framework of two of the four properties of the skeleton proposed by Blum: the centering and the thinness properties.

Assume that we have an efficient thinning process which thins an object layer by layer. It can lead to sets of simple points which cannot be all removed together without breaking the topology. We have then two choices: we can keep all these points or we can arbitrarily remove a subset of them in order to have a thin skeleton.

The first choice leads to a *symmetric skeleton*: it is well centered but it is not thin. A symmetric skeleton is closer to the medial axis as it preserves more maximal ball centers.

The second choice leads to an *asymmetric skeleton*: it is thin but not well centered, due to arbitrary choices.

4.3 Thinning Algorithms

Homotopic thinning in the digital framework consists of removing simple points from an object, until either no more simple point can be found (resulting in an ultimate skeleton), or a satisfactory subset of voxels has been reached (resulting in a surface or curvilinear skeleton).

Two main strategies are possible for removing simple points: sequential removal and parallel removal.

4.3.1 Sequential algorithms

Sequential removal of simple points can be achieved by detecting simple points in an object, and removing them one after the other, until no more simple point can be found. After removing a simple point, the new set of simple points of the object must be computed. Such basic strategy does not guarantee the result, which is an ultimate skeleton, to be centered in the original object.

It is important, when designing a sequential thinning algorithm, to decide of a removal order of simple points, and of a strategy for preserving interesting visual features of the object (as explained in Section 4.2).

In order to obtain a centered skeleton, one must define a precise order of removal of simple points. Usually, in order to get a centered skeleton, it is necessary to delete simple points "layer by layer", from the outer layer to the inner one. Many strategies have been proposed in 2D for choosing of a removal order (a very exhaustive survey of thinning methods in 2D before 1992 can be found in [88]) : for example, in the 80s, it was proposed to follow an object's contour in order to find and remove simple points "layer by layer" [8, 134]. However, this thinning scheme (as many others) hardly generalizes to 3D.

A widely used strategy to obtain a centered skeleton with a sequential thinning process consists of computing a priority function on the object and removing the simple points of X according to the value of this function [44] : at each step, the simple point that is removed is one with the lowest possible value. The Euclidean distance map (storing distances to the complementary) is widely used as priority function [41, 103, 169]. Other works use discrete distances, such as the chamfer distance [141], to decide of a removal order.

Algorithm 1: Basic Thinning(X, W, D, k)

Data: A k-connected shape X, a priority function D and a subset $W \subseteq X$
Result: A skeleton of X

while *there exists a k-simple point in* $X \backslash W$ **do**
  $A = \{y \in X \backslash W | y$ is k-simple for $X\}$
  $B = \{x \in A |$ for all $y \in A, D(x) \leq D(y)\}$
  Let $z \in B$
  $X = X \backslash \{z\}$
return X

Algorithm 1 shows the basic thinning scheme based on a priority function. The set W, called *inhibitor set*, is a set of points of the input object which must be in the resulting skeleton, and D is the priority function used to decide an order of points removal (here, the lower priority means faster removal, so it is possible to use a distance map as priority function).

An inhibitor set allows to choose "anchor points" for the skeleton, and therefore preserve the visual aspect of the original object in the skeleton. However, when performing a thinning guided by an Euclidean distance map, the points of the inhibitor set and the directions of thinning followed by the algorithm are not always "compatible". In [169], the authors use a thinning algorithm where the slope of the priority function is used to dynamically add points to the constraint set. In [41], the authors propose to merge the slope calculation into the priority function, leading to a new priority function and a new thinning algorithm which works in 2D and 3D.

4.3.2 Parallel algorithms

Whereas iterative algorithms remove only one simple point at a time, parallel algorithms consist of removing a set of simple points simultaneously. Algorithm 2 illustrate this concept. In

Algorithm 2: Parallel Thinning(X)

Data: A connected shape X
Result: A skeleton of X

```
repeat
    S = {x ∈ X | x satisfies deletion conditions}
    X = X\S
until stability ;
return X
```

general, an object possesses more than one simple point. When a simple point is removed from an object, three events can take place: non simple points can become simple, simple points can become non-simple, or nothing changes. It can be easily seen that removing two or more simple points simultaneously from an object may lead to obtaining a set that is not homotopic to the original object. Parallel thinning (simultaneously removing several simple points) is possible but must be performed under certain conditions.

The main problem of parallel thinning is that removing simple points simultaneously from an object usually "breaks" the topology. Thus, additional conditions must be introduced in order to solve this problem. In order to describe the different strategies, we introduce some vocabulary found in [129].

A *thinning operator* transforms an object X only by removing some points, without changing the topology of the shape. A *parallel thinning operator* simultaneously deletes all points of the object X satisfying some conditions (called the *deletion conditions*). The *support* of a thinning operator is the minimal set of points whose values must be examined to verify the deletion conditions for a point $p \in X$.

According to Hall [62], parallel thinning algorithms can be divided into three categories:

- In *directional algorithms*, the main loop is divided into sub-iterations, and the deletion conditions are changed from one sub-iteration to another.
- In *subfield-based algorithms*, the points of the object are decomposed into subsets, and at a given iteration of the algorithm, only points in a given subset are studied.
- In *fully parallel algorithms*, no sub-iteration takes place : the same thinning operator is used at each iteration of the main loop. However, if for the other strategies the usual support is a $3 \times 3 \times 3$ neighborhood (i.e. N_{26}), fully parallel operators require an extended support (usually, $5 \times 5 \times 5$).

4.3.2.1 Directional Algorithms

A directional algorithm is a parallel algorithm where each iteration is divided in sub-iterations using different thinning operators. Each thinning operator can only delete points belonging to a specific *border orientation*. This border orientation classification is performed regarding the belonging to complementary of points in a given neighborhood.

The first directional thinning algorithm has been proposed in 1971 by Stefanelli and Rosenfeld [162] in the $\mathbb{Z}^2$ framework. The authors used the 4-neighborhood to classify the borders points (see Figure 4.8), leading to a 4 sub-iteration algorithm.

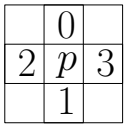


FIGURE 4.8: Each number i represents a pixel which has to be out of the object to allow the deletion of p in the ith sub-iteration.

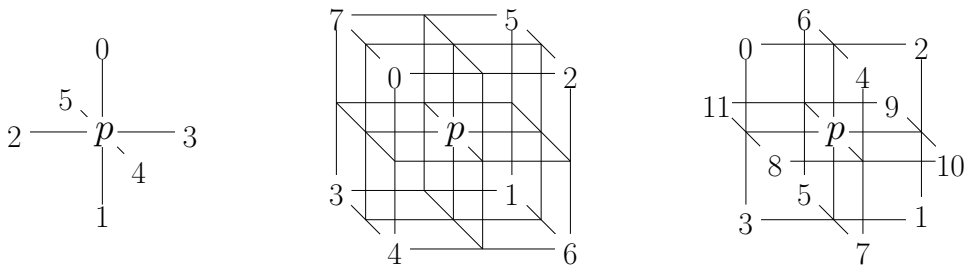


FIGURE 4.9: From the left to the right: Directions used for directional thinning algorithms with 6, 8 and 12 sub-iterations. Each number i represents a voxel which have to be out of the object to allow the deletion of p in the ith sub iteration.

In the $\mathbb{Z}^3$ framework, the most usual kind of directional algorithms use a classification in regard of the 6-neighborhood, leading to algorithms with 6 sub-iterations [13, 57, 90, 93, 96, 120, 130, 182] or 3 sub-iterations [126–128], considering in one sub-iteration a class and its opposite.

Directional algorithms with 8 sub-iterations [132] (classification in regard of $N_{26} \setminus N_{18}$ neighborhood) and 12 sub-iterations [92, 131] (classification in regard of $N_{18} \setminus N_6$ neighborhood) have also been proposed.

Generally, a directional algorithm is designed on the model of Algorithm 3, using classification of the directions in the order proposed in Figure 4.9 for 3D, or Figure 4.8 for 2D.

Algorithm 3: Directional Thinning(X)

Data: A connected shape X
Result: A skeleton of X

```
repeat
    foreach direction i do
        S = {x ∈ X | x satisfies deletion conditions of the ith thinning operator}
        X = X\S
until stability ;
return X
```

4.3.2.2 Subfield-Based Algorithms

In subfield-based algorithms, $\mathbb{Z}^n$ is divided in $m \geq 2$ disjoint subfields $\mathbb{Z}_0^n, ..., \mathbb{Z}_{m-1}^n$, such $\bigcup_{i=0}^{m-1} \mathbb{Z}_i^n = \mathbb{Z}^n$.

An iteration of subfield-based algorithm is divided in sub iterations corresponding to the activation of each subfield. A point can be deleted only when the subfield it belongs to is activated.

The first subfield-based algorithm, at our knowledge, has been proposed in 1984 by Groen and Foster [60] in the $\mathbb{Z}^2$ framework.

In the $\mathbb{Z}^3$, algorithms using 2 subfields [72, 97, 98], 4 subfields [99, 123] and 8 subfields [123] have been proposed.

The subfields are designed in sort that if 2 points p, q belong to the same subfield, q is not in a given neighborhood of p (see Figure 4.10):

- for $m = 2, \forall 0 \leq i < m, \forall (p, q) \in \mathbb{Z}_i^3 \times \mathbb{Z}_i^3, q \notin N_6(p)$
- for $m = 4, \forall 0 \leq i < m, \forall (p, q) \in \mathbb{Z}_i^3 \times \mathbb{Z}_i^3, q \notin N_{18}(p)$
- for $m = 8, \forall 0 \leq i < m, \forall (p, q) \in \mathbb{Z}_i^3 \times \mathbb{Z}_i^3, q \notin N_{26}(p)$

It is this property which allow the preservation of the topology, the deletion of a point having a controlled impact on its neighborhood.

Generally, a subfield-based algorithm is designed on the model of Algorithm 4, using subfields proposed in Figure 4.10.

Notice that this strategy is known to usually provide bad quality skeletons.

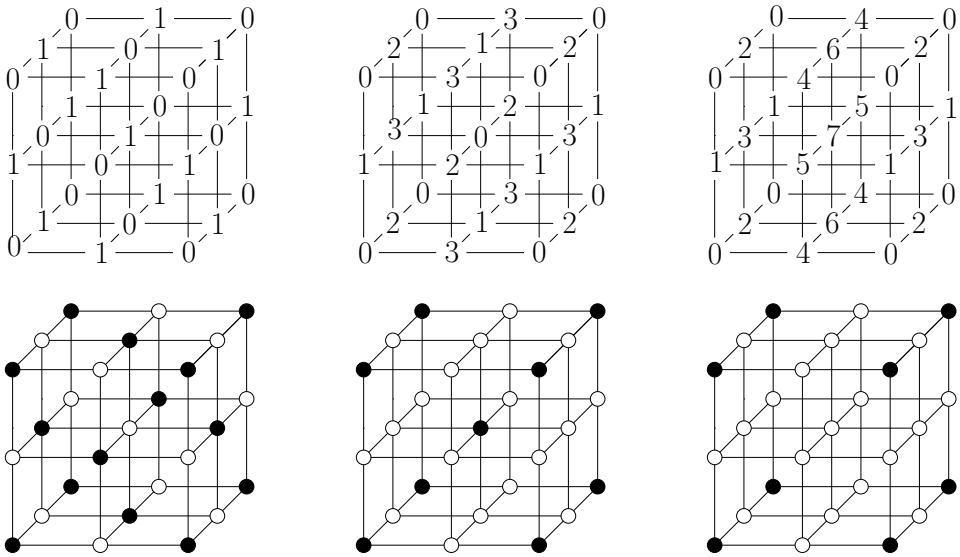


FIGURE 4.10: Top, from the left to the right: Divisions of $\mathbb{Z}^3$ into 2, 4 and 8 subfields. For an algorithm using k subfields, in a ith sub iteration, only points marked i will be considered. Bottom, from the left to the right: black points represent considered points in the first iteration of a k-subfield based algorithm, for $k = 2, 4, 8$.

Algorithm 4: Subfield Based Thinning(X)

Data: A connected shape X
Result: A skeleton of X

```
repeat
    foreach subfield F_i do
        S = {x ∈ X ∩ F_i | x satisfies deletion conditions }
        X = X \ S
until stability ;
return X
```

4.3.2.3 Fully Parallel Algorithms

Fully parallel algorithms use only one thinning operator in the full space of the same iteration. In order to do it without breaking the topology, the support has to be bigger than for other classes of thinning algorithms.

Rutovitz was the first to propose a (fully) parallel thinning algorithm, in 1966 ([150]). However, it has been proved that Rutovitz's algorithm does not always preserve topology and "patches" exist in order to correct it ([38]). It is in 1981 that Pavlidis published the first fully parallel homotopic thinning algorithm ([135]), in 2D for 8-connected objects, that was later proved to preserve topology ([38]).

In the 3D digital topology framework, the problem is very complex, and several theories have been developed in order to help to design correct fully parallel thinning algorithms.

Minimal non-deletable sets Minimal non-deletable sets were introduced by C. Ronse ([146]) in order to characterize under which conditions simple points could not be removed simultaneously from a 2D object without "breaking" the topology.

Minimal non-deletable sets define "forbidden features" that should not appear in a set of pixels in order for it to be simple. Minimal non-deletable sets were designed in order to prove that 2D parallel thinning algorithms were topology preserving (and therefore valid) by testing only a small number of configurations of points. A computer-based implementation of these tests was later proposed in [61], a 3D implementation of these sets was proposed in [83], [95] and [84], and a 4D implementation was proposed in [53] and [86].

***P*-simple points** In 1995, Bertrand introduced the P-simple points in order to characterize, in 3D, which simple points could be removed simultaneously ([13]).

As with the minimal non-deletable sets, the P-simple points allow to check if existing parallel 3D thinning algorithms work: indeed, in [14], the author gives a method for checking, based on the P-simple points framework, the topological validity of thinning algorithms. Moreover, P-simple points were widely used in order to propose new parallel 3D thinning algorithms (an example of a new algorithm is given in [14]).

Critical kernels The recent critical kernels, introduced in [16], present a new framework for performing parallel thinning in 2D, 3D and 4D (see [17] and [11]). In relation to critical kernels, a new definition and new characterizations of simple points in 2D, 3D and 4D (see [40]) have been proposed, and links between this framework, P-simple points and minimal non-deletable sets were established in [39]. More precisely, critical kernel theory is a generalization of the two other theories. Critical kernels were also used to prove that some thinning algorithms were valid, while others were not correct ([38]).

Chapter 5

Faster Implementation of Thinning Schemes

In our context of real time motion capture, the considered 3d objects contain between 50 000 and 1 000 000 voxels, and have to be skeletonized in less than 40 milliseconds (remember that skeletonization is only a part of the process which has to be "real time"). No direct implementation of existing thinning algorithms running on an average computer (with CPU frequency lower than 4GHz) can reach this time for these sizes of object. In this chapter we will first discuss speed optimizations schemes found in the literature, then propose some new ones.

5.1 Classical Optimizations

In the literature, the thinning algorithms are most usually described using pseudo code, only describing the main lines of the real implementation, and skipping details about, e.g. how to reach low execution time. However, thinning algorithms used in full applications (like medical imaging or material studies) are implemented using some common optimizations which can be called "classical". In this section, we will review the most usual of them.

5.1.1 Restriction of Points to Check

The first and the most common optimization method consists of a restriction of points to test for deletion conditions at each iteration. If we implement directly the algorithms described in pseudo code in the majority of the literature, we have to check all the points in the image grid at each iteration, leading to a time complexity in $O(|G| \times I)$, with I being the number of iterations and $|G|$ being the size of the image grid.

First, as it is a *deletion* condition test, we can test only the points which can be deleted (i.e. the points belonging to the object). Furthermore, in order to be deletable in a thinning process, a point has to be simple. Considering this fact, we can test only the points at the border of the object at each iteration.

This is naturally done by sequential thinning using distance transform as a priority function [41, 103, 141, 169], by sorting the points in regard of their distance to the border.

In the case of parallel algorithms, a list can be used in order to store the border points [129, 133]. Before the iterative part of the algorithm, the list is initialized by adding all the border points of the image. During the iterative process, only points of the list are tested: if a point has to be deleted, we remove it from the list and we add its neighbors belonging to the object which are not already in the list.

Using this optimization, the time complexity is linear with regard to the size of the image.

5.1.2 Optimization of Deletion Conditions Tests

Depending on the considered algorithm, the tests of deletion conditions can be very lengthy.

Assuming that deletion tests take into consideration only a determined finite set of neighbors (the support of the thinning operator), pre-computation methods can be used in order to obtain quicker and constant time.

5.1.2.1 Usage of Look-Up Tables

In thinning algorithms, a widely used tool(explicitly in [126–129], for example) to reduce computation time is the *look-up table* (usually shortened in LUT).

A look-up table is an array containing the precomputed results of some time-consuming problem P for a given set of input values V. In order to find the index in the look-up table storing the result $P(v)$, $v \in V$, we have to define a bijective function $\mathcal{K} : V \to \{0, ..., S-1\}$, with S being the size of the look-up table.

In the case of thinning algorithms, the problem P is the deletion condition test returning a boolean (true if the point can be deleted, false otherwise) and the set of input values is the set of all possible configurations of points in the support.

For a $3 \times 3 \times 3$ support, we have to consider that each point in the 26 neighborhood may or may not belong to the object, leading to a look-up table size of $2^{26} = 67\ 108\ 864$ values. Since each result is a boolean, the look-up table uses 4 megabytes in memory.

The bijective function $\mathcal{K}$ is defined using some properties of the possible configurations of points in the support. Assuming that each point in the support has a binary status (1 if the point belongs to the object, 0 otherwise), each configuration can be represented by a vector $v = (v_0, ..., v_{25}) \in \{0,1\}^{26}$, were each v_i represents the status of the ith point for a given navigation order of the support. We define the bijective function providing an index to each configuration by:

$$\mathcal{K}(v) = \sum_{i=0}^{26} 2^i * v_i$$

The navigation order of the support depends on the algorithm: if the algorithm needs only one look-up table, the order will be the same as the order of the image representation in memory, in order to minimize transfer between RAM and CPU cache. If the algorithm needs several look-up tables based on different supports for the same point, the order will place first the points of the 6-neighborhood, then those of the 18-neighborhood and finally those of 26-neighborhood. Figure 5.1 shows two examples of these orders.

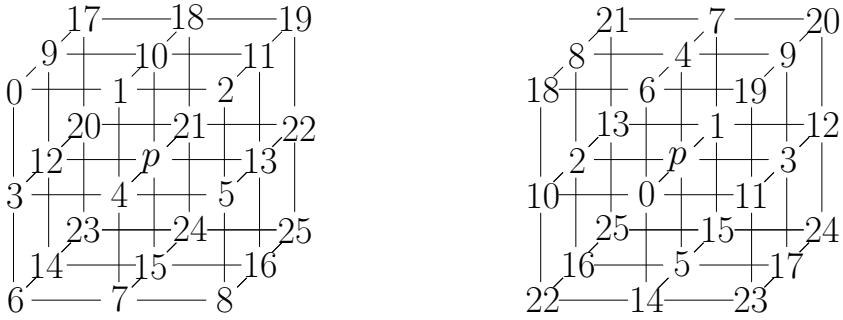


FIGURE 5.1: Left: an order optimized for reducing amount of cache miss. Right: an order useful for algorithms using several look-up tables for different neighborhoods.

5.1.2.2 Usage of Binary Decision Diagrams

Look up tables are not the only way to store and access pre-computed results of a function. For functions taking as input a vector of booleans, as in the case of deletion condition tests, specific data structures have been designed: the *binary decision trees*, and their compact representation, the *binary decision diagrams* [29] (usually shortened in *BDD*). The use of binary decision diagrams is proposed in some thinning literature papers [92, 93].

A binary decision tree is a binary tree with several specificities:

- each leaf represents a result;
- if the input vector contains n values, the tree has n levels of interior nodes;
- all the interior nodes of the ith level of the tree represent the ith value of the input vector;

- the leaves of the left (respectively, right) subtree of an interior node at level i contains results obtained if the ith value equals 0 (respectively 1).

Finding the result for a given boolean vector is done using Algorithm 5.

Algorithm 5: Result Retrieval in Binary Decision Tree

Data: Binary Decision Tree T, input vector $v = (b_0, ..., b_n)$
Result: Value of function for v in input

```
Node cur = root of T
while cur is not a leaf do
    if value represented by cur equals 0 then
        cur = left child of cur
    else
        cur = right child of cur
return value contained in cur
```

Example Let consider the function $f : \{0,1\}^4 \to \{0,1\}$ such $f(b_0, b_1, b_2, b_3) = (b_0\ AND\ b_1)\ OR\ (b_2\ AND\ b_3)$. The results of f are represented by the binary decision tree in Figure 5.2. If we are searching for the value of $f(0,0,1,0)$, from the root, we go down twice in the left child ($b_0 = 0$ and $b_1 = 0$), then in the right child ($b_2 = 1$) and finally in the left child again ($b_3 = 1$): the result (0) is contained in the leaf where we are.

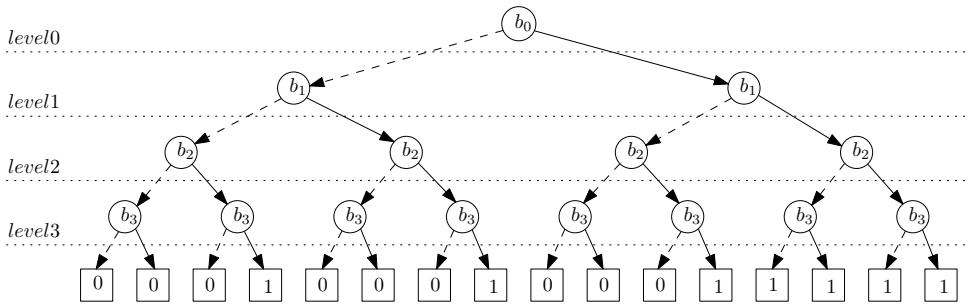


FIGURE 5.2: A binary decision tree containing the results of the function f.

A binary decision diagram is obtained from a binary decision tree by applying two operations:

1. all subtrees which are equal are merged;
2. remove nodes with two subtrees which are equal.

Figure 5.3 shows an example of transformation of a binary decision tree into a binary decision diagram.

Notice that the size of the binary decision diagram depends on both the considered function and the order in which are considered the values.

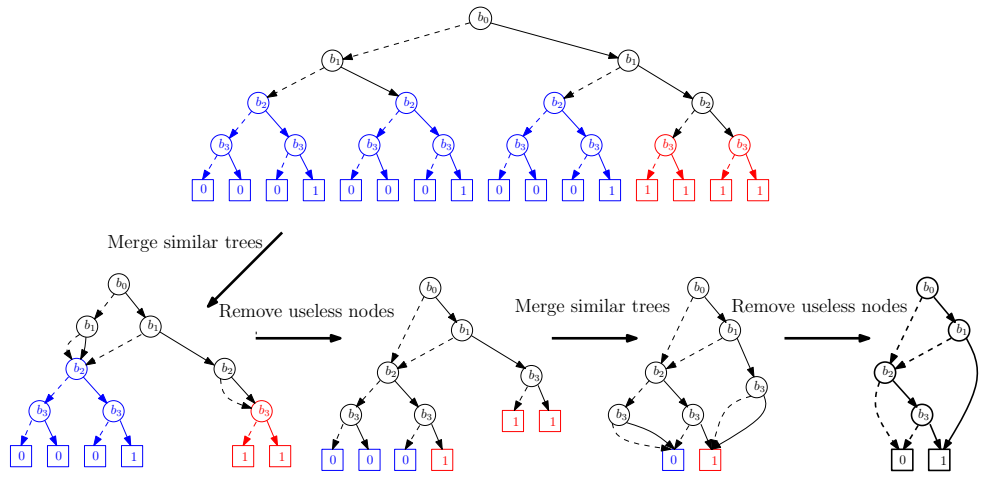


FIGURE 5.3: Transformation of a binary decision tree into a binary decision diagram.

5.1.2.3 Look Up Tables versus Binary Decision Diagrams

A good point for the binary decision diagram is the potentially lower space needed to store the results. Concerning the speed-up, contrary to the LUT which provides an access in constant time, access with BDD depends both on the considered deletion conditions and the order in which the values of the configuration are considered. Furthermore, even if the number of points to consider is reduced, navigation in the binary decision diagram can be more time consuming than the direct access to an array as required by LUT methods.

5.2 Look Up Tables for Critical Kernel Based Algorithms

Look up tables can not be directly used for algorithms using extended $5 \times 5 \times 5$ support: it implies more than 2×10^{37} values, and storing a such look-up table is not possible, even using all the hard drives existing in the world. Furthermore, the computation of values of the whole look-up table with the fastest super-computer would take more time than the age of the universe.

In 2009, Palágyi and Németh [133] proposed fully parallel algorithms using a thinning operator with $5 \times 5 \times 5$ support, the deletion conditions taking into account not only the configuration of the 26-neighborhood, but also the simplicity of the points in this neighborhood. However, the authors proposed algorithms using tests only in the $3 \times 3 \times 3$ neighborhoods, using a decomposition of the tests, and allowing for the usage of look-up tables.

In 2006, Bertrand and Couprie [17] proposed a new fully parallel thinning scheme based on critical kernels, allowing to easily design thinning algorithms for any kind of skeletons (homotopic kernel, curvilinear or surfacic, asymmetric as well as symmetric). Even if the support of the thinning operator is $5 \times 5 \times 5$, we will propose in this section a way to decompose the tests as

evoked above and to use look-up tables for critical kernel based algorithms, in order to improve their speed.

5.2.1 Critical Kernel Based Algorithms

First we briefly recall the scheme of critical kernel based algorithms. These algorithms use the notion of crucial cliques, which are set of points which can not be all removed in order to preserve topology. The crucial points are classified in several subtypes : 3-crucial, 2-crucial, 1-crucial and 0-crucial. An iteration of the algorithm consist in keeping only points of the shape which are crucial.

The belonging to a crucial clique is detected using a set of templates (see Figure 5.4).

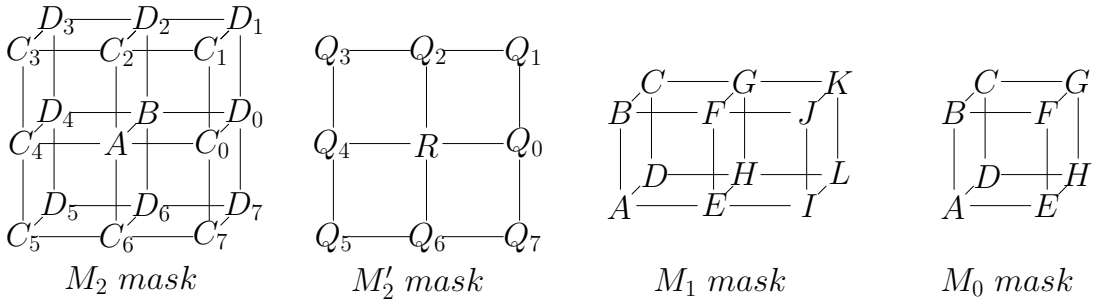


FIGURE 5.4: Masks used for crucial cliques detection.

Definition 3. *[17] Let $X \subseteq \mathbb{Z}^3$ the object, and let M be a set of voxels of X.*

1) The set M matches the mask M_2 if the three following conditions hold:

- *i) the voxels in M are simple for X.*
- *ii) $M = \{A, B\}$.*
- *iii) the 2D configuration M_2' obtained by setting $R \in M_2'$ and setting $Q_i \in M_2'$ if $\{C_i, D_i\} \cap X \neq \emptyset$, with $i \in [0, ..., 7]$, is such that R is non-simple in the 2D sense.*

2) The set M matches the mask M_1 if the four following conditions hold:

- *i) the voxels in M are simple and not 2-crucial for X.*
- *ii) $M = \{E, F, G, H\} \cap X$.*
- *iii) the set $\{E, G\}$ or the set $\{F, H\}$ (or both) is included in M.*
- *iv) we have either $[U \cap X \neq \emptyset$ and $V \cap X \neq \emptyset]$ or $[U \cap X = \emptyset$ and $V \cap X = \emptyset]$, with $U = \{A, B, C, D\}$ and $V = \{I, J, K, L\}$.*

3) The set M matches the mask M_0 if the three following conditions hold:

- *i) the voxels in M are simple and neither 2-crucial nor 1-crucial for X.*

ii) $M = \{A, B, C, D, E, F, G, H\}$.

iii) at least one of the sets $\{A, G\}, \{B, H\}, \{C, E\}, \{D, F\}$ *is a subset of* M.

Definition 4. *[17] Let* $X \subseteq \mathbb{Z}^3$ *and let* M *be a set of voxels of* X.

i) M *is a* 2*-crucial clique for* X *if and only if* M *matches the mask* M_2 *or a mask obtained from it by* $\frac{\pi}{2}$ *rotations.*

ii) M *is a* 1*-crucial clique for* X *if and only if* M *matches the mask* M_1 *or a mask obtained from it by* $\frac{\pi}{2}$ *rotations.*

iii) M *is a* 0*-crucial clique for* X *if and only if* M *matches the mask* M_0.

Using these definitions, Bertrand and Couprie proposed several generic fully parallel thinning algorithms, like Algorithm 6 for symmetric skeletons and Algorithm 7 for asymmetric ones.

Note that these algorithms ensure the preservation of some points defined by the set K and the use of any kind of extremity point or isthmus, defined by the function Ψ.

Function Ψ will return the subset of the input set $S \in \mathbb{Z}^3$ which has to be preserved in the skeleton (for an homotopic kernel, Ψ always returns $\emptyset$).

Algorithm 6: CK Thinning

Data: $X \subseteq \mathbb{Z}^3$, $K \subseteq X$, $\Psi : X \to \Psi(X)$
Result: A skeleton of X

repeat
  $K = K \cup \Psi(X)$
  $R =$ set of non-simple points for X or which are in K
  foreach *possible position of mask* M_2 *(and any of its* $\frac{\pi}{2}$ *rotations)* **do**
    if M_2 *matches* **then** $R = R \cup \{A, B\}$
  foreach *possible position of mask* M_1 *(and any of its* $\frac{\pi}{2}$ *rotations)* **do**
    if M_1 *matches* **then** $R = R \cup [\{E, F, G, H\} \cap X]$
  foreach *possible position of mask* M_0 **do**
    if M_0 *matches* **then** $R = R \cup [\{A, B, C, D, E, F, G, H\} \cap X]$
  $X = R$
until *stability* ;
return X

Algorithm 7: ACK Thinning

Data: $X \subseteq \mathbb{Z}^3$, $K \subseteq X$, $\Psi : X \to \Psi(X)$
Result: A skeleton of X

repeat
  $K = K \cup \Psi(X)$
  $R =$ set of non-simple points for X or which are in K
  foreach *possible position of mask* M_2 *(and any of its* $\frac{\pi}{2}$ *rotations)* **do**
    if M_2 *matches and* $\{A, B\} \cap R = \emptyset$ **then**
      $R = R \cup \{A\}$
  foreach *possible position of mask* M_1 *(and any of its* $\frac{\pi}{2}$ *rotations)* **do**
    if M_1 *matches and* $\{E, F, G, H\} \cap R = \emptyset$ **then**
      $P =$ first voxel in $\{E, F, G, H\} \cap X$
      $R = R \cup \{P\}$
  foreach *possible position of mask* M_0 **do**
    if M_0 *matches and* $\{A, B, C, D, E, F, G, H\} \cap R = \emptyset$ **then**
      $P =$ first voxel in $\{A, B, C, D, E, F, G, H\} \cap X$
      $R = R \cup \{P\}$
  $X = R$
until *stability* ;
return X

5.2.2 Reformulation of templates

In order to be compatible with LUT usage, a mask has to take into consideration configurations of points with binary state.

Some observations can be done on the used templates:

- In all cases two different informations are taken into consideration:
  1. The configuration of the points belonging to the object (we denote this configuration by C_X).
  2. The configuration of the points belonging to the object and holding the same set of additional properties, e.g. being non-simple (we denote this configuration by C_S).

  Notice that these two configurations are contained in a $3 \times 3 \times 3$ support.
- Due to both symmetries of the templates and their use in the scheme, it is possible to consider only deletion conditions on one voxel, instead of a set, the other voxels of the set being tested for the same neighborhood at a different step of the same iteration.

Using these two observations, we can redefine the masks M_i, $i \in \{0, 1, 2\}$, using for each of them two templates: one providing conditions for C_X and the other one for C_S. This separation in two masks allows for the use of look-up tables.

In the sequel of this section we say that the configuration C_S is compatible with the mask M_{iS}, $i \in \{0, 1, 2\}$, if and only if the points represented by black disks in the mask belong to the set S and the points represented by white disks in the mask do not belong to the set S.

5.2.2.1 Reformulation of M_2

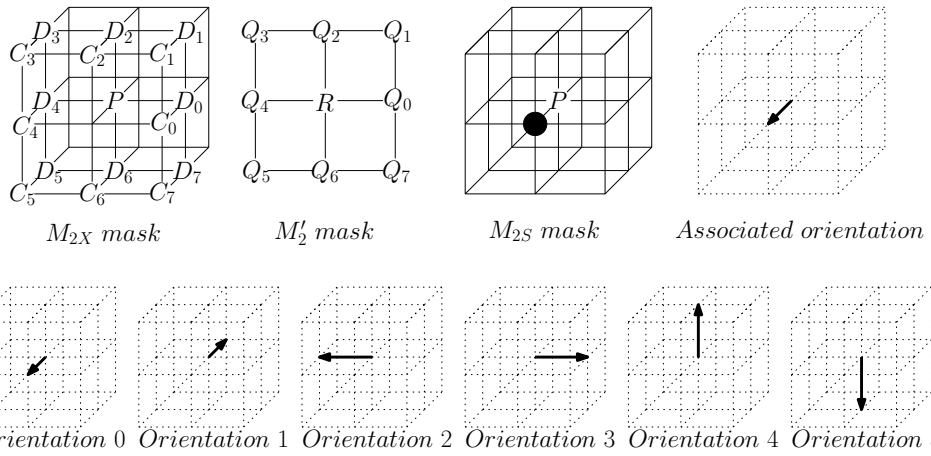


FIGURE 5.5: New masks for 2-crucial points detection, and all possible orientations of these masks.

A detection of 2-crucial points can be done using masks shown in Figure 5.5.

Definition 5. *Let $X \subseteq \mathbb{Z}^3$.*

A) A point P matches the mask M_{2X} if the 2D configuration M'_2 obtained by setting $R \in M'_2$ and setting $Q_i \in M'_2$ if $\{C_i, D_i\} \neq \emptyset$, with $i \in [0, ..., 7]$, is such that R is non-simple in the 2D sense (condition on C_X).

B) A point P matches the mask M_{2S} if the configuration of the set of simple points is compatible with M_{2S}.

C) A simple point P is 2-crucial if and only if it matches the masks M_{2X} and M_{2S}, or the masks obtained by a same π or $\frac{\pi}{2}$ rotation of these two masks.

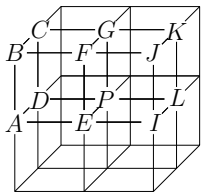
5.2.2.2 Reformulation of M_1

A detection of 1-crucial points can be done using masks shown in Figure 5.6.

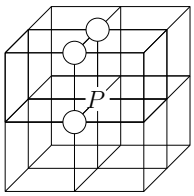
Definition 6. *Let $X \subseteq \mathbb{Z}^3$.*

A) A point P matches the mask M_{1X} if the two following conditions hold:

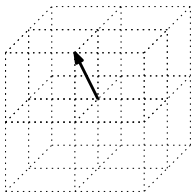
i) the set $\{E, G\}$ or the set $\{F\}$ (or both) is included in X.



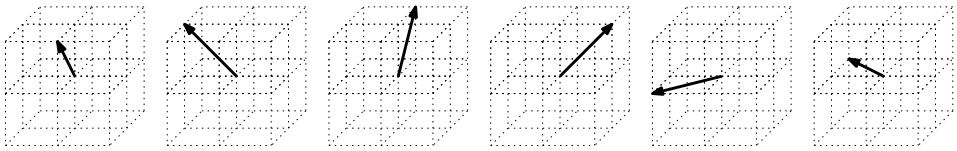
M_{1X} *mask*



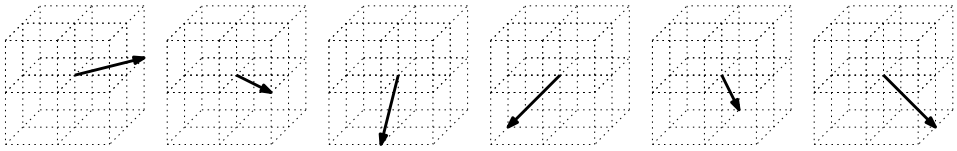
M_{1S} *mask*



Associated orientation



Orientation 0 *Orientation* 1 *Orientation* 2 *Orientation* 3 *Orientation* 4 *Orientation* 5



Orientation 6 *Orientation* 7 *Orientation* 8 *Orientation* 9 *Orientation* 10 *Orientation* 11

FIGURE 5.6: New masks for 1-crucial points detection, and all possible orientations of these masks.

ii) we have either $[U \cap X \neq \emptyset$ *and* $V \cap X \neq \emptyset]$ *or* $[U \cap X = \emptyset$ *and* $V \cap X = \emptyset]$*, with* $U = \{A, B, C, D\}$ *and* $V = \{I, J, K, L\}$*.*

B) A point P *matches the mask* M_{1S} *if the configuration of the set of 2-crucial or not simple points belonging to* X *is compatible with* M_{1S}*.*

C) A simple and non 2-crucial point P *is* 1*-crucial if and only if it matches the masks* M_{1X} *and* M_{1S}*, or the masks obtained by the same combination of* $\frac{\pi}{2}$ *rotations of these two masks.*

5.2.2.3 Reformulation of M_0

A detection of 0-crucial points can be done using masks shown in Figure 5.7.

Definition 7. *Let* $X \subseteq \mathbb{Z}^3$*.*

A) A point P *matches the mask* M_{0X} *if at least one of the sets* $\{A, G\}, \{B\}, \{C, E\}, \{D, F\}$ *is a subset of* X*.*

B) A point P *matches the mask* M_{0S} *if the configuration of the set of 2-crucial or 1-crucial or non simple points belonging to* X *is compatible with* M_{0S}*.*

C) A simple and neither 2-crucial nor 1-crucial point P *is* 0*-crucial if and only if it matches the masks* M_{0X} *and* M_{0S}*, or the masks obtained by the same combination of* $\frac{\pi}{2}$ *rotations of these two masks.*

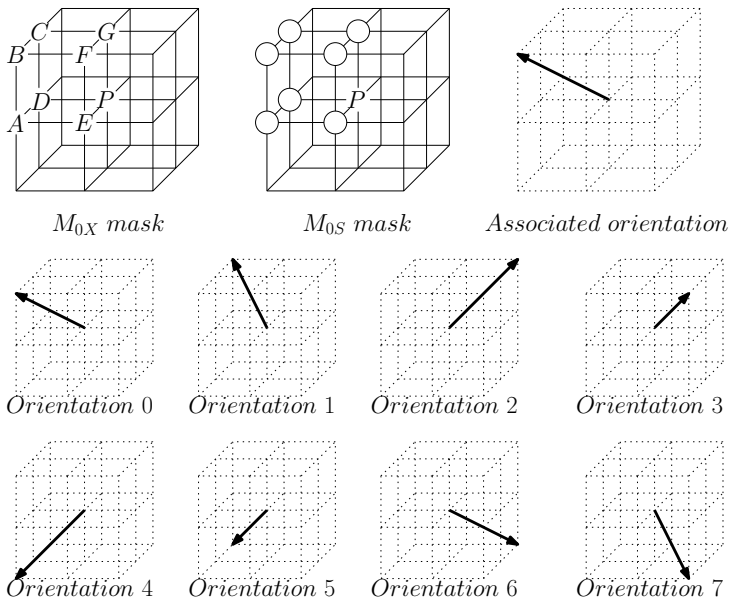


FIGURE 5.7: New masks for 0-crucial points detection, and all possible orientations of these masks.

5.2.3 Look Up Tables for New Masks

Now we have masks taking into consideration the configuration of points sharing the same properties, so we can use look-up tables.

We can first observe that masks M_{2S} and M_{1S} do not take into consideration all the points in N_{26}, but only points in N_6 and N_{18}, respectively. For these masks we can use look-up tables with size reduced to 2^6 and 2^{18}, respectively.

As each couple of masks has several orientations and as the properties checking requires the matching of the two masks for a same orientation, we have to store separately the result of the matching of each mask for each orientation.

Due to this fact, in each of the six look-up tables (one per mask) the result for a configuration C will be a vector of bits $v = v_0, ..., v_{D-1}$, such v_i, $i \in [0, ...D-1]$ indicate if C matches ($v_i = 1$) or not ($v_i = 0$) the mask in its ith orientation.

The properties checking is then obtained as follows. Let LUT_{iS} and LUT_{iX}, $i \in \{0, 1, 2\}$, the look-up tables storing the results of matching masks M_{iS} and M_{iX}, respectively. Let $index\ of\ C$ be the index of the configuration C computed as explained in Section 5.1.2.1. For a point p with configurations $C_X(p)$ and $C_S(p)$, p will match M_{iS} and M_{iX} if and only if $LUT_{iS}(index\ of\ C_S(p))$ and $LUT_{iX}(index\ of\ C_X(p))$ have a common bit to 1 in their vector.

This verification can be done using the bitwise AND operator: p will match M_{iS} and M_{iX} if and only if $LUT_{iS}(index\ of\ C_S(p))\ AND\ LUT_{iX}(index\ of\ C_X(p)) \neq 0$.

To sum up, we use six look-up tables for checking of crucial points, as summarized in the Table 5.1.

TABLE 5.1: Look Up Tables for Crucial Point Detection

	related mask	**number of values**	**value length**		**Total size**
			in bits	**in bytes**	
LUT_{2X}	M_{2X}	2^{26}	6	1	64 Mb
LUT_{2S}	M_{2S}	2^{6}	6	1	64 b
LUT_{1X}	M_{1X}	2^{26}	12	2	128 Mb
LUT_{1S}	M_{1S}	2^{18}	12	2	512 Kb
LUT_{0X}	M_{0X}	2^{26}	8	1	64 Mb
LUT_{0S}	M_{0S}	2^{26}	8	1	64 Mb

Notice that even if the masks proposed in the Figures 5.5, 5.6 and 5.7 do not require the consideration of more than 17 neighbors, the number of values of the LUT corresponds to full 26-neighborhood. It is due to the fact that LUT stores the results for each orientation of the masks, thus we have to consider not only the points considered for one orientation, but for all orientations at the same time.

5.2.4 Reformulation of the Thinning Schemes

Using the look-up tables described above, we can now reformulate the thinning schemes.

Algorithm 8: Optimized ACK Thinning

Data: $X \subseteq \mathbb{Z}^3$, $K \subseteq X$, $\Psi : X \to \Psi(X)$
Result: A skeleton of X

repeat
  $K = K \cup \Psi(X)$
  $S =$ set of simple points for X which are not in K
  foreach *point* $p \in S$ **do**
    if $LUT_{2X}(index_{26}(p, X))$ *AND* $LUT_{2S}(index_{6}(p, S)) \neq 0$ **then**
      $S = S \setminus \{p\}$
  foreach *point* $p \in S$ **do**
    if $LUT_{1X}(index_{26}(p, X))$ *AND* $LUT_{1S}(index_{18}(p, X \setminus S)) \neq 0$ **then**
      $S = S \setminus \{p\}$
  foreach *possible position of mask* M_0 **do**
    if $LUT_{0X}(index_{26}(p, X))$ *AND* $LUT_{0S}(index_{26}(p, X \setminus S)) \neq 0$ **then**
      $S = S \setminus \{p\}$
  $X = X \setminus S$
until *stability* ;
return X

We denote by $index_n(p, S)$ the index in the look-up table of the configuration of the set S in the n-neighborhood of a point p. Algorithms 9 and 8 are the LUT-optimized reformulations of the thinning Algorithms 6 and 7, respectively.

Algorithm 9: Optimized CK Thinning

Data: $X \subseteq \mathbb{Z}^3$, $K \subseteq X$, $\Psi : X \to \Psi(X)$
Result: A skeleton of X

```
repeat
    K = K ∪ Ψ(X)
    S = set of simple points for X which are not in K
    R = ∅
    foreach point p ∈ S do
        if LUT_2X(index_26(p, X)) AND LUT_2S(index_6(p, S)) ≠ 0 then
            R = R ∪ {p}
    S = S \ R
    R = ∅
    foreach point p ∈ S do
        if LUT_1X(index_26(p, X)) AND LUT_1S(index_18(p, X \ S)) ≠ 0 then
            R = R ∪ {p}
    S = S \ R
    R = ∅
    foreach possible position of mask M_0 do
        if LUT_0X(index_26(p, X)) AND LUT_0S(index_26(p, X \ S)) ≠ 0 then
            R = R ∪ {p}
    S = S \ R
    X = X \ S
until stability ;
return X
```

5.2.4.1 Speed up

The use of look-up tables on critical kernel based thinning (already using border list defined in Section 5.1.1) proposed in this section allows a speed-up yielding a division by ten of the computation time for the thinning on some objects. Please refer to the Section 5.4 for more details.

5.3 Configurations Computation Speed-Up

As explained in Section 5.1.2.1, in order to access to the results stored in a look-up table, we have first to compute the index corresponding to the configuration of the point. Computing this index is time consuming, due to the fact that values of the neighborhood are spaced in

the memory, whatever the used representation of the image, involving cache misses (transfers between CPU cache and main memory).

Furthermore, for the same point, several conditions must be checked, depending on the thinning method: in directional thinning, the same point must be checked for all directions; in critical kernel based thinning, points are checked for simplicity, then for the different crucial properties.

As each check is done in a different step of the iteration, the configuration index is recomputed for each one, involving a waste of time.

We can notice that in a thinning algorithm, the configuration of a point changes only when one of its neighbors is deleted, thus the configuration changes a limited number of times. In this section, we propose a method for avoiding multiple useless computations of the configuration index of the same point.

At the beginning of the algorithm, we build a map M with the same dimensions as the image I, such that for each point $p \in I$, $M(p)$ contains the index of the configuration of p in I.

During the iterative process, when a point p is deleted, we update the configuration index of its neighbors in M, by putting to 0 the bit representing p in their configuration.

Using this method, the configurations are computed only when a point is deleted, instead of each time a point is checked.

We call this optimization the *Configuration Computation Speed-Up*, abbreviated as *CCSU*.

5.3.0.2 Speed up

The use of this configuration computation speed-up divides by two (in average) the computation time of a thinning. Please refer to Section 5.4 for more details.

5.4 Benchmarks

In this section, we study the speed gain of the proposed optimizations.

5.4.1 Methodology

For this purpose, we implemented several versions of the same algorithm with different levels of optimization: a first version without the use of LUT, a second version using LUT, and a third one using both LUT and CCSU. All the versions are optimized using border lists, as explained in Section 5.1.1. Then, we applied the different algorithms on a set of various images, and we compared their computation speed.

5.4.1.1 Implemented Algorithms

For the evaluation of LUT usage on critical kernel based methods, we implemented the asymmetric scheme (see Algorithm 7), using a constraint function Ψ returning the set of 1D isthmus of the shape. We denote this algorithm by *ACK3*.

In order to observe the impact of CCSU (in addition to LUT) on different thinning strategies, we also implemented the 6-directional thinning algorithm proposed by Palágyi and Kuba in 1998 [130]. We denote this algorithm by *PKD6*.

5.4.1.2 Tested Images

The algorithms are tested on a set of images of different sizes, containing various objects. Our image set is represented in Figure 5.8.

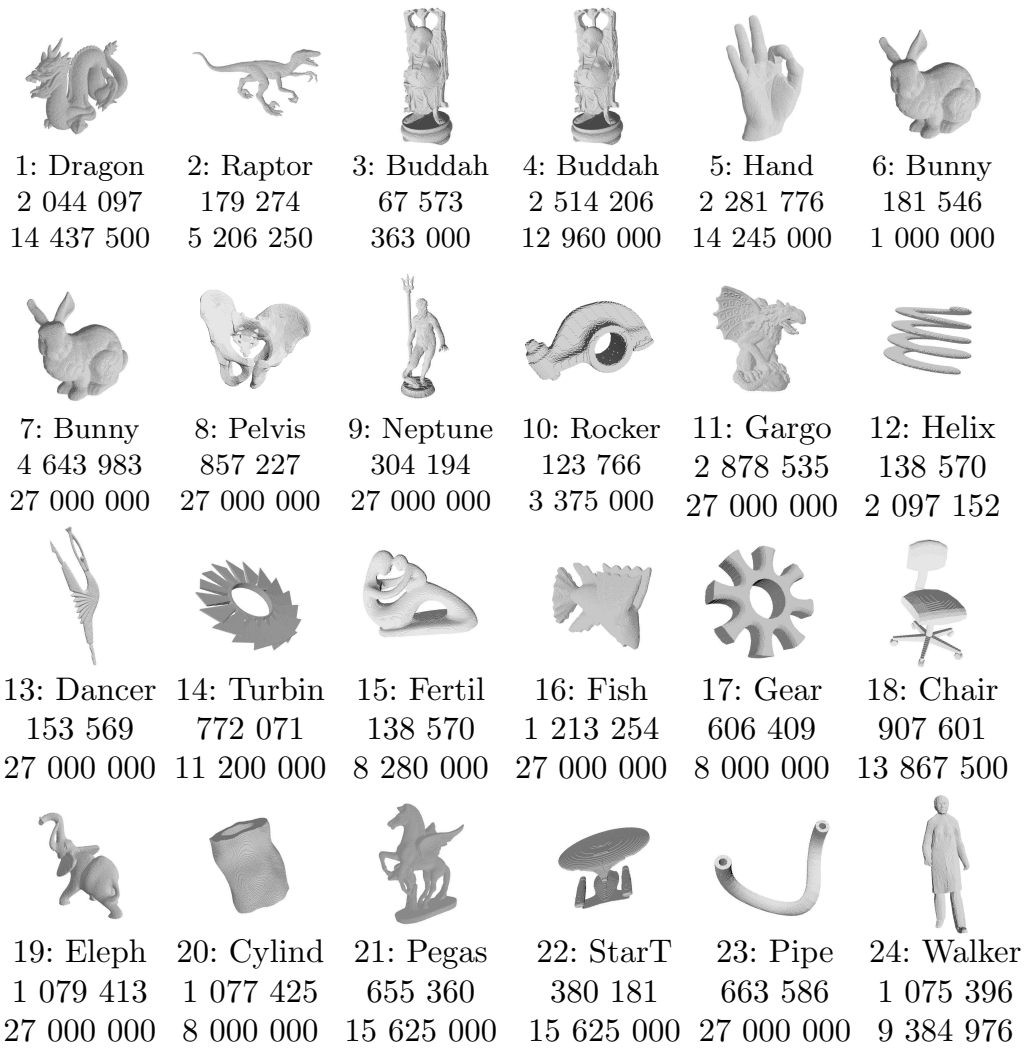


FIGURE 5.8: Images used for the benchmark. Above each image, the id number, the name, the size of the object and the size of the image.

5.4.2 Results

The results are obtained by averaging times of ten thinnings for each image. All the experiments have been done on a computer with a processor Intel(R) Q8200 at 2.33 GHz and 3 gigabytes of RAM.

Figures 5.9 and 5.10 show the computation speed results of ACK3 and PKD6, respectively.

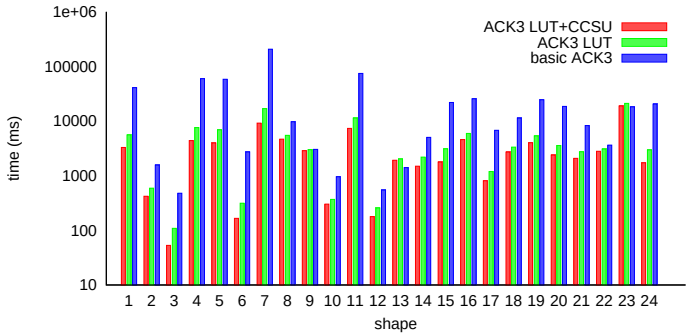


FIGURE 5.9: Computation speed results for ACK3 implemented with different levels of optimization, for each image of our test set.

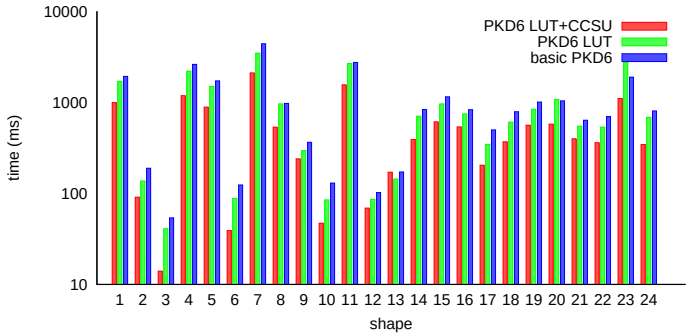


FIGURE 5.10: Computation speed results for PKD6 implemented with different levels of optimization, for each image of our test set.

In order to observe better the speed gain of each optimization, we display the ratio of computation time of each version versus the computation time of a more optimized version. Using this method, we obtain three values:

LUT gain: $\frac{time\ for\ basic\ version}{time\ for\ LUT\ optimized\ version}$

CCSU gain: $\frac{time\ for\ LUT\ version}{time\ for\ LUT{+}CCSU\ optimized\ version}$

Full gain: $\frac{time\ for\ basic\ version}{time\ for\ LUT{+}CCSU\ optimized\ version}$

Figures 5.11 and 5.12 show these ratios for ACK3 and PKD6, respectively. Results are summarized in Table 5.2.

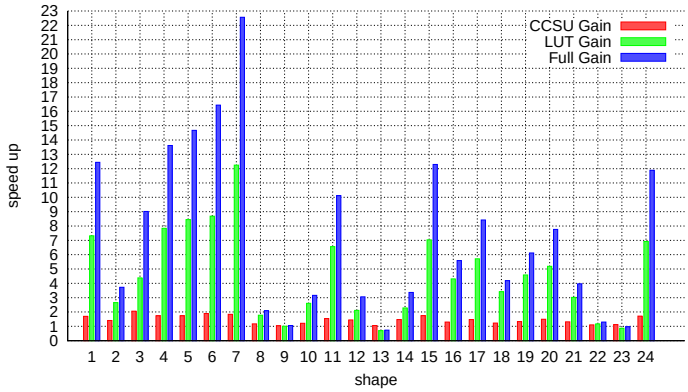


FIGURE 5.11: Speed gain obtained by different optimizations on ACK3.

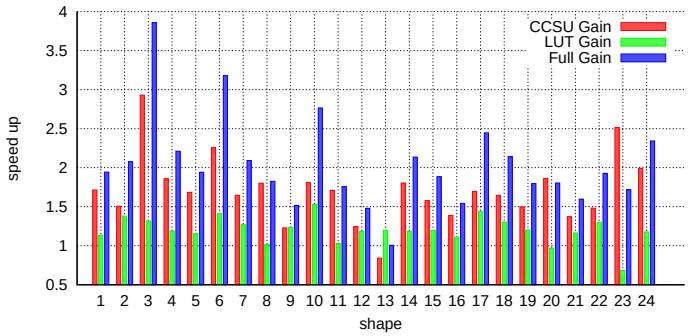


FIGURE 5.12: Speed gain obtained by different optimizations on PKD6.

TABLE 5.2: Summarize of Optimizations Speed Gain
(results expressed as "average [min, max]")

	CCSU Speed Gain	**LUT Speed Gain**	**CCSU+LUT Speed Gain**
ACK3	1.46 [1.04, 2.06]	4.62 [0.69, 12.25]	7.44 [0.73, 22.56]
PKD6	1.71 [0.84, 2.93]	1.20 [0.68, 1.53]	2.04 [1.01, 3.86]

5.4.3 Discussion

5.4.3.1 CCSU Speed Gain

An observation of Figures 5.11 and 5.12 shows some interesting characteristics about CCSU.

First we can notice that the speed gain is higher for PKD6 than for ACK3: it is due to the larger amount of deletion condition tests for the same point in this directional algorithm than in the fully parallel algorithm ACK3.

We also observe the quasi inexistent impact of CCSU optimization on ACK3 for some shapes (9, 13, 22, 23). We can see that the LUT optimization is inefficient too for these shapes. We

can observe that these shapes are thinner than the other ones, and thus require less iterations for the same image size.

In the case of shape 13, the dancer, we can notice that CCSU slows down PKD6. It is due to the higher time consuming pre-iteration step, which cannot be offset by the gain during the iterative process, due to the low amount of iterations.

To sum up, CCSU is an optimization that is easy to set up, and which provides a sensible speed-up in a majority of cases. However, in the case of shapes close to their skeleton, it is inefficient.

5.4.3.2 LUT Speed Gain for Critical Kernel Based Thinning

Observing Figure 5.11, we can notice pretty spectacular speed-up by using LUT on ACK3 (more than 12 times faster in the case of shape 7). However, we can notice that the effect is inexistent for some shapes (already mentioned in the previous section).

Combining both LUT and CCSU highly improve the speed of critical based algorithms, but the speed gain highly depends to the shape in input.

5.5 Application in Motion Capture Framework

Now we have optimized the speed of thinning algorithms, we can try to use them in our context of motion capture. In order to test their speed in a real context, we propose to use two data sets: the dancer data set (height views) from *4Drepository* from INRIA and a home made data set with four views recording gesture of a hand.

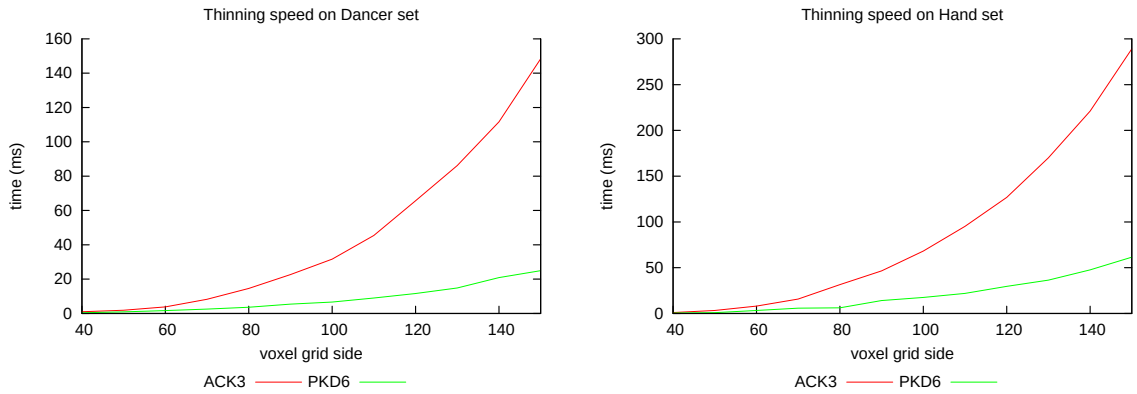


FIGURE 5.13: Thinning speed of fully optimized ACK3 and PKD6 on two data sets.

For each frame of each data set, we compute the visual hull at different voxel grid resolutions and we measure the time to skeletonize the obtained object using our two optimized algorithms. The average computation time for each resolution on each data set are reported in Figure 5.13.

We can observe that despite of our efforts for speed-up critical kernel based thinning, ACK3 is still five times slower than directional thinning, and is not fast enough for its use in real time applications. However, we can notice that directional algorithms like PKD6 are very fast and reach the real time even for high resolutions of the voxel grid.

Chapter 6

Isthmus Based Directional Thinning

In our context of motion capture, we have to consider both the speed of the thinning process (see Chapter 5), and the quality of the produced skeletons. The amount of noise in the resulting skeleton has an impact on the good matching with the model. Due to the use of 1D-isthmus, ACK3 provides curvilinear skeleton with very low amount of spurious branches. However, as seen in the end of the previous chapter, this algorithm is not the fastest one.

On the other hand, faster algorithms, like directional thinning or subfield based algorithms, provide noisier skeletons. It is mainly due to the consideration of curve end points as constraint points, which are more likely to appear during thinning than isthmuses.

In this chapter, we design a directional thinning algorithm considering isthmuses as constraints. Furthermore, we propose a constraint check at the iteration level, instead of sub-iteration level, in order to consider less constraint points than usually in this kind of strategy.

6.1 Background

Our method is based on two different works. We use constraints that were originally introduced by Bertrand and Couprie [18] in the context of sequential thinning, and the design of the masks used in our parallel directional algorithm is based on those defined by Palágyi and Kuba [130].

6.1.1 Bertrand and Couprie Isthmus Based Thinning

In 2007, Bertrand and Couprie [18] proposed a sequential thinning based on isthmuses (see Section 4.2.2.2). Contrary to extremity points, isthmuses are not simple points. Due to this fact, isthmuses cannot be used in classical strategy in which a point is removable if and only if it is simple and does not match certain conditions.

The strategy used by Bertrand and Couprie consists in dynamically detecting the considered isthmuses (1D or 2D) and accumulating them in a constraint set. A point can be deleted if it is simple and not in the constraint set. By this way, a point detected as an isthmus in a given iteration cannot be removed later. Depending on the considered isthmuses, the method provides curvilinear skeletons (in case of 1D isthmuses) or surface skeletons (in case of 2D isthmuses). In both cases, this strategy provides skeletons with very few spurious elements.

The consideration of isthmuses instead of extremities points is interesting for two reasons: isthmuses can be easily locally defined and detected (it is not the case of surface end points), and appear less often than extremities during thinning process, leading to less noisy skeletons. However, the sequential approach proposed is less adapted than parallel ones for this purpose, and provides skeletons of lower quality.

6.1.2 Palágyi and Kuba 6-directional Thinning

In 1998, Palágyi and Kuba [130] proposed a 6-directional thinning algorithm producing curvilinear skeletons, which we denote by PKD6.

Each iteration of PKD6 consists of 6 sub-iterations in which points are removed from the shape if and only if the configuration of their 26-neighborhood matches at least one mask of the mask set $\mathcal{M}_d$ for the given orientation d. The mask set $\mathcal{M}_U$ used for the *UP* direction is presented in Figure 6.1. Mask sets $\mathcal{M}_D$,$\mathcal{M}_N$,$\mathcal{M}_S$,$\mathcal{M}_E$,$\mathcal{M}_W$ for the other directions (*DOWN*, *NORTH*, *SOUTH*, *EAST* and *WEST*) are obtained by appropriate rotations and reflections. The algorithm is summarized in Algorithm 10.

Algorithm 10: PKD6 [130]

Data: $X \subseteq \mathbb{Z}^3$
Result: A skeleton of X

$Y = X$ **repeat**
  $Y = Y \setminus \{p \in Y ; p \text{ matching one mask of } \mathcal{M}_U\}$
  $Y = Y \setminus \{p \in Y ; p \text{ matching one mask of } \mathcal{M}_D\}$
  $Y = Y \setminus \{p \in Y ; p \text{ matching one mask of } \mathcal{M}_N\}$
  $Y = Y \setminus \{p \in Y ; p \text{ matching one mask of } \mathcal{M}_S\}$
  $Y = Y \setminus \{p \in Y ; p \text{ matching one mask of } \mathcal{M}_E\}$
  $Y = Y \setminus \{p \in Y ; p \text{ matching one mask of } \mathcal{M}_W\}$
until *stability* ;
return Y

Palágyi and Kuba proved that their algorithm preserves topology, using the following theorem proved by Ma in [95].

Definition 8. *[95] Let $X \subset \mathbb{Z}^3$ be an object. The set $D = \{d_1, ..., d_k\} \subseteq X$ is called a simple set of X if D can be arranged in a sequence $\langle d_{i_1}, ..., d_{i_k} \rangle$ in which each d_{i_j} is simple for $X \setminus \{d_{i_1}, ..., d_{i_{j-1}}\}$ for $j = 1, ..., k$. (By definition, let the empty set be simple.)*

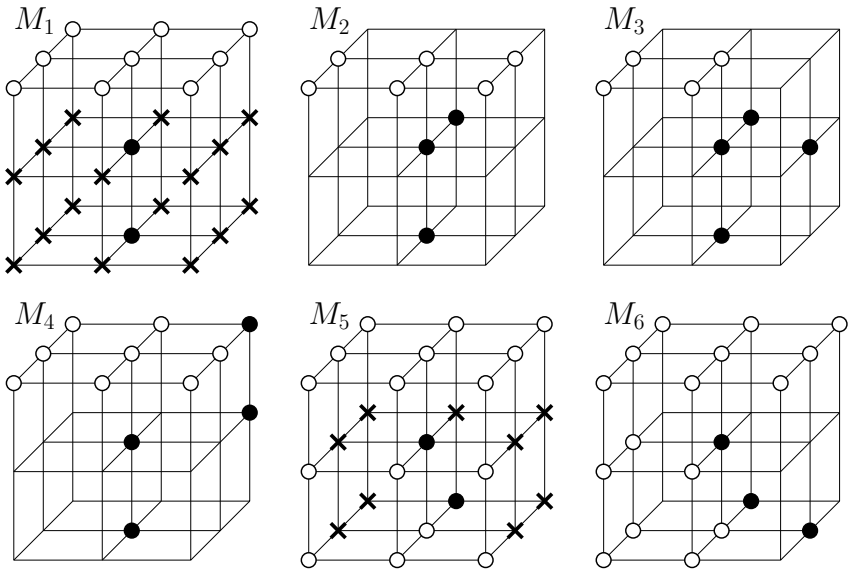


FIGURE 6.1: Base masks M_1-M_6 and their rotations around the vertical axis form the set of masks $\mathcal{M}_U$ for direction *UP*. Points marked by a black disk have to belong to the object, points marked by a white disk have to be out of the object, and at least one point marked by a cross in masks M_1 and M_5 has to belong to the object.

Theorem 9. *[95] A 3D parallel reduction operation preserves topology if all of the following conditions hold:*

1. *Only simple points are deleted.*
2. *If two corners belonging to the object, p and q, of a 2×2 square in $\mathbb{Z}^3$ are deleted, then the set $\{p, q\}$ is simple.*
3. *If three corners belonging to the object, p, q and r, of a 2×2 square in $\mathbb{Z}^3$ are deleted, then the set $\{p, q, r\}$ is simple.*
4. *If four corners belonging to the object, p,q,r and s, of a 2×2 square in $\mathbb{Z}^3$ are deleted, then the set $\{p, q, r, s\}$ is simple.*
5. *No component of the object contained in a $2 \times 2 \times 2$ cube in $\mathbb{Z}^3$ can be deleted completely.*

Theorem 10. *[130] The thinning algorithm PKD6 preserves topology, in the sense of Theorem 9.*

6.2 New Method

We propose a 6-directional thinning algorithm based on isthmuses. Our algorithm is designed in two steps, separating the thinning process (using a 6-directional approach) and the isthmuses detection. Each iteration consists of:

1. updating the constraint set, by adding points of the object detected as isthmuses,
2. removing deletable points of the object which are not in the constraint set (see Algorithm 11).

Notice that this design is inspired by the one of critical kernel based methods (see Algorithms 6 and 7) reported in the previous chapter.

Algorithm 11: D6I1D

Data: $X \subseteq \mathbb{Z}^3$, $K \subseteq X$
Result: A skeleton of X

```
repeat
    K = K ∪ Ψ_I1D(X)
    foreach direction d in (U, D, N, S, E, W) do
        X = τ_d(X, K)
until stability ;
return X
```

This design allows for the preservation of some points defined a priori by the initial set K (which may be empty).

The function $\Psi_{I1D}(X)$ returns the set of all the 1D-isthmuses of X (see Section 4.2.2.2).

The function $\tau_d(X, K)$ returns the set $X \setminus S$, S being the set of all points from $X \setminus K$ respecting deletion conditions in X for direction d. This function is defined in the sequel of this section.

6.2.1 Design of Deletion Condition Masks

Contrary to deletion conditions used in directional algorithms found in the literature, those of τ_d permit the deletion of curve end points, the constraint points being considered separately.

Deletion conditions can be represented using a set of masks: a point respects deletion conditions if and only if it matches at least one of the masks. In this section, we present the masks used by τ_d, obtained by modifying those used by PKD6.

In order to separate geometrical and topological conditions, we have to allow in τ_d the deletion of ending points (i.e. points with only one neighbor in the object). For this purpose, we add three masks to $\mathcal{M}_d$, representing ending points which have to be removed for the direction d. These masks are shown in Figure 6.2 for the case $d = U$.

Proving that our algorithms preserve topology (Proposition 11 and Proposition 12) can be done using the framework of critical kernels, introduced by Bertrand [16]. This framework is indeed the most powerful one for both proving and designing n-dimensional homotopic thinning

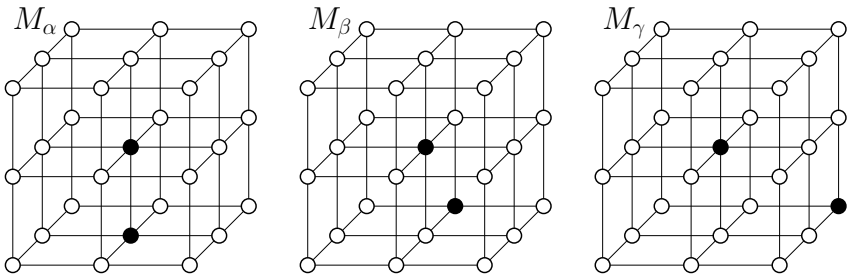


FIGURE 6.2: Masks M_α-M_γ and their rotations around the vertical axis are added to the set of masks $\mathcal{M}_U$ for direction *UP* in order to remove ending points.

algorithms. However, in this particular case it is simpler to build on the proof that was given by Palágyi and Kuba for PKD6, as our algorithm only slightly differs from the latter.

Proposition 11. *Algorithm PKD6 with mask sets $\mathcal{M}_d$ extended by adding convenient rotations of M_α, M_β and M_γ preserves topology.*

Proof. We prove that all conditions of Theorem 9 hold.

Let $X \subseteq \mathbb{Z}^3$ and $p \in X$. If p matches a mask in $\mathcal{M}_U$, then we know that it is simple from Theorem 10. If p matches a mask in $\{M_\alpha, M_\beta, M_\gamma\}$, then p is obviously simple.

Let Z be a set of object points contained in a 2×2 square in $\mathbb{Z}^3$, or being an object component contained in a $2 \times 2 \times 2$ cube in $\mathbb{Z}^3$, with $|Z| \geq 2$.

If all points of Z match masks of $\mathcal{M}_U$, then we know that all conditions of Theorem 9 hold, from Theorem 10.

Otherwise, if one point p of Z matches a mask in $\{M_\alpha, M_\beta, M_\gamma\}$, then clearly $|Z| = 2$. Let q be the other point in Z. By examination of all configurations, it can be seen that any mask in $\mathcal{M}_U \cup \{M_\alpha, M_\beta, M_\gamma\}$ positioned on q does not match, thus Z cannot be deleted. □

6.2.2 Mask Set Reduction

The set of masks $\mathcal{M}_U \cup \{M_\alpha, M_\beta, M_\gamma\}$ can be compacted as proposed in Figure 6.3. We can observe that mask M'_1 matches exactly the same configurations as those matched by masks M_1 and M_α. In the same way, mask M'_5 matches exactly the same configurations as those matched by masks M_5 and M_β. Masks M'_2,M'_3,M'_4,M'_6 and M'_7 are respectively the same as M_2,M_3,M_4,M_6 and M_γ.

Using the sets $\mathcal{M}'_d$ with $d \in \{U, D, N, S, E, W\}$, we can now define τ_d, as proposed in Algorithm 12.

Algorithm 12: τ_d

Data: $X \subseteq \mathbb{Z}^3$, $K \subseteq X$
Result: A subset of X
1 $R =$ set of points matching a mask belonging to $\mathcal{M'}_d$
2 $R = R \setminus K$
3 **return** $X \setminus R$

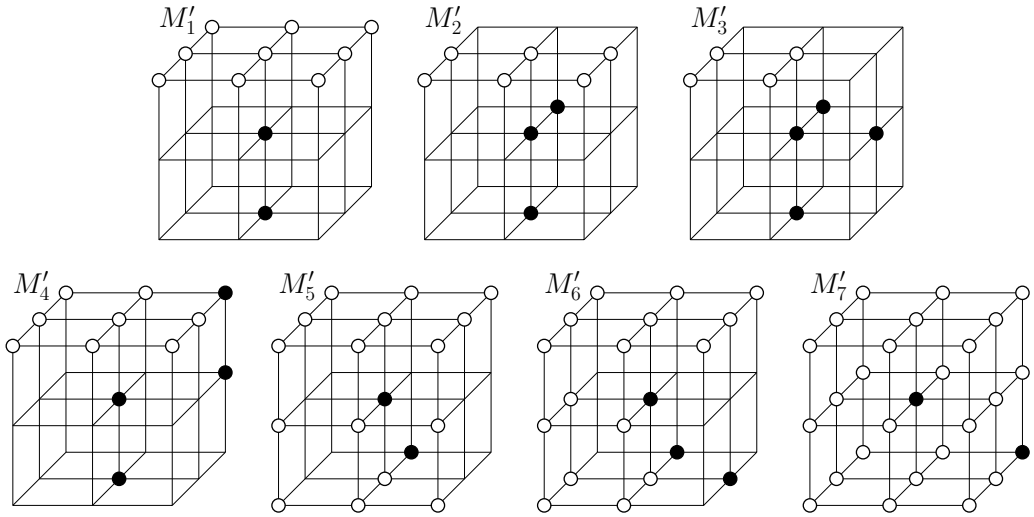


FIGURE 6.3: Final set of masks $\mathcal{M}'_U$ for direction *UP*.

Proposition 12. *Algorithm D6I1D preserves topology.*

Proof. From Proposition 11, algorithm PKD6 with mask sets $\mathcal{M}_d$ extended by adding convenient rotations of M_α, M_β and M_γ preserves topology. We observe that:

- D6I1D differs only from this algorithm by constraining some points to be preserved;
- if Theorem 9 holds for an operation A, then it also holds for an operation B that removes only a subset of the points removed by A.

These observations lead to the conclusion that D6I1D preserves topology. □

6.3 Comparative Results

We implemented *D6I1D* with all the optimizations proposed in the previous chapter: border list, LUT and CCSU. We apply the same procedure as proposed in Section 5.5 to observe the usability of our algorithm in a real context of motion capture. The results are shown in Figure 6.4.

We observe that the computation speed of *D6I1D* is very close to the one of *PKD6*, and can be used in real time.

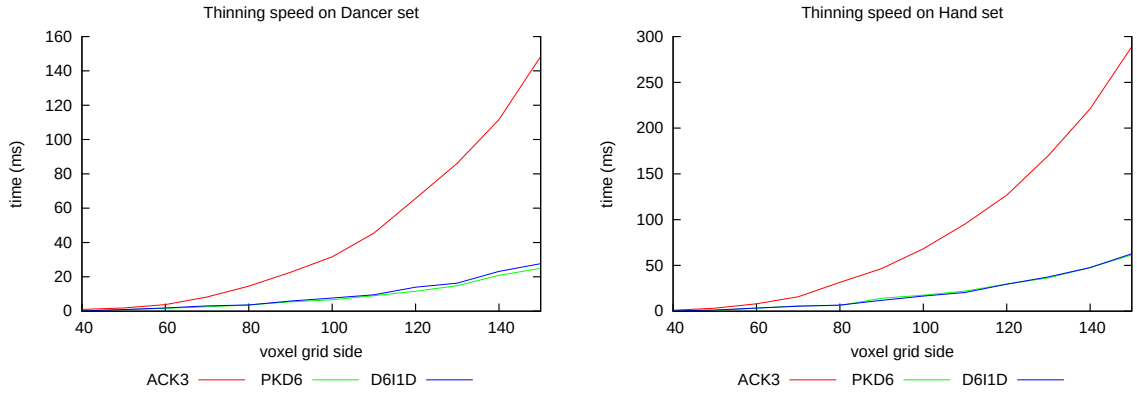


FIGURE 6.4: Thinning speed of fully optimized ACK3, PKD6 and D6I1D on two data sets.

Concerning the quality of the resulting skeleton, as no general quality measurement exists (to our knowledge), we propose to observe the results of the three studied thinning, presented in Figures 6.5 and 6.6. It can be observed that our algorithm provides results with less spurious branches than PKD6, and in certain cases less than ACK3 (see images 20 and 23, for example). However, ACK3 provide better results for flat shape parts; like the chair (image 18) or the turbine (image 14).

6.4 Other Kinds of Skeletons

In fact, our algorithm offers great flexibility, due to the fact that constraint set detection (function Ψ) can be changed to implement other conditions. We provide two examples of such variations.

6.4.1 Ultimate Skeletons

In order to obtain ultimate skeletons instead of curvilinear ones, we propose a very simple variation of D6I1D, consisting in replacing Ψ_{I1D} by the function returning empty set (K is unchanged). We call this variation D6U. Figure 6.7 shows some results obtained by using this algorithm.

6.4.2 Surface Skeletons

In order to obtain surface skeletons instead of curvilinear ones, we propose a very simple variation of D6I1D, consisting in replacing Ψ_{I1D} by a function Ψ_{I2D} returning the set of 2D-isthmuses (see Section 4.2.2.2), which can be locally detected. We call this variation D6I2D. Figure 6.8 shows some results obtained by using this algorithm.

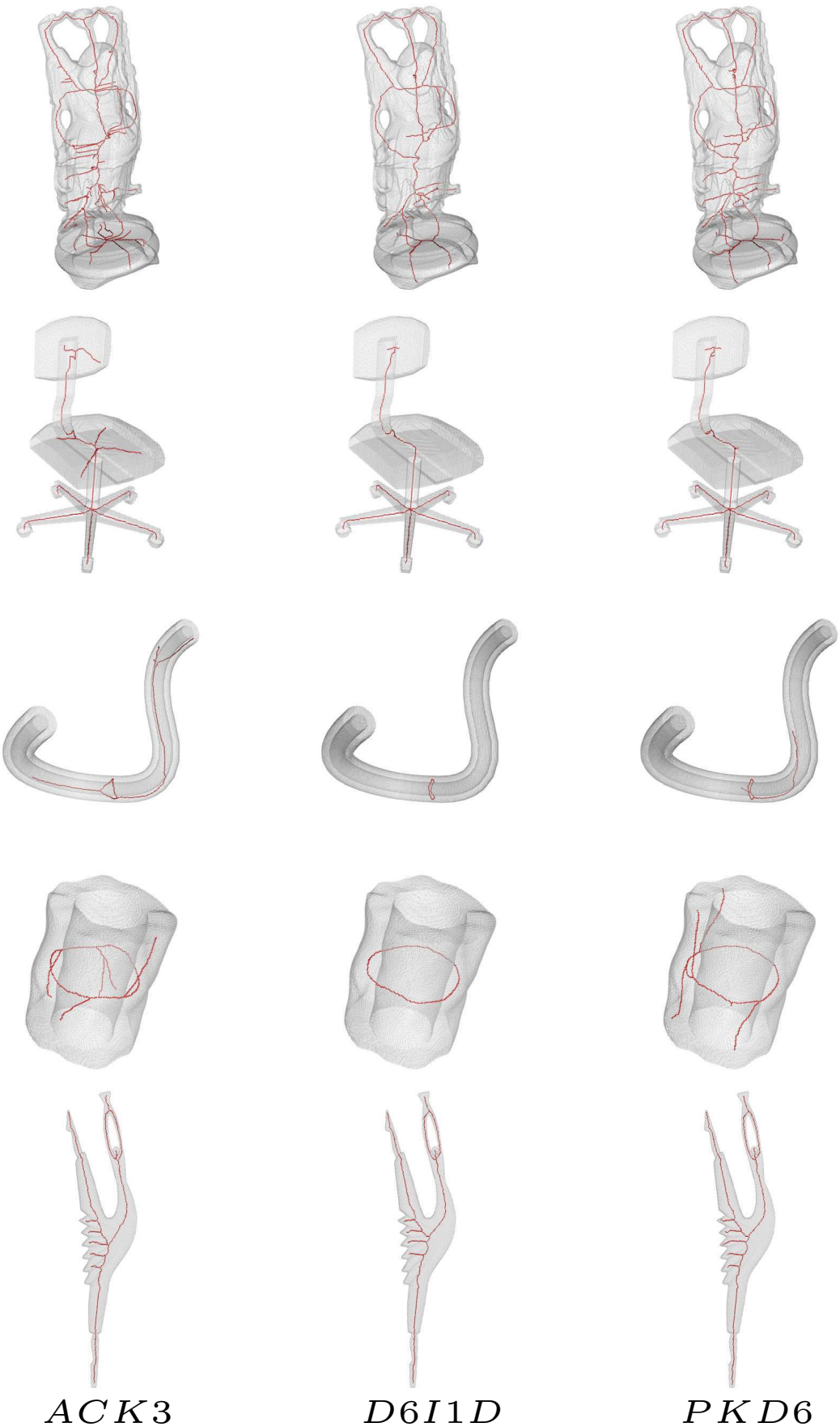


FIGURE 6.5: Skeletons resulting of different algorithms for several shapes. From the left to the right: algorithms ACK3, D6I1D, and PKD6. From the top to the bottom: shapes 4, 18, 23, 20 and 13.

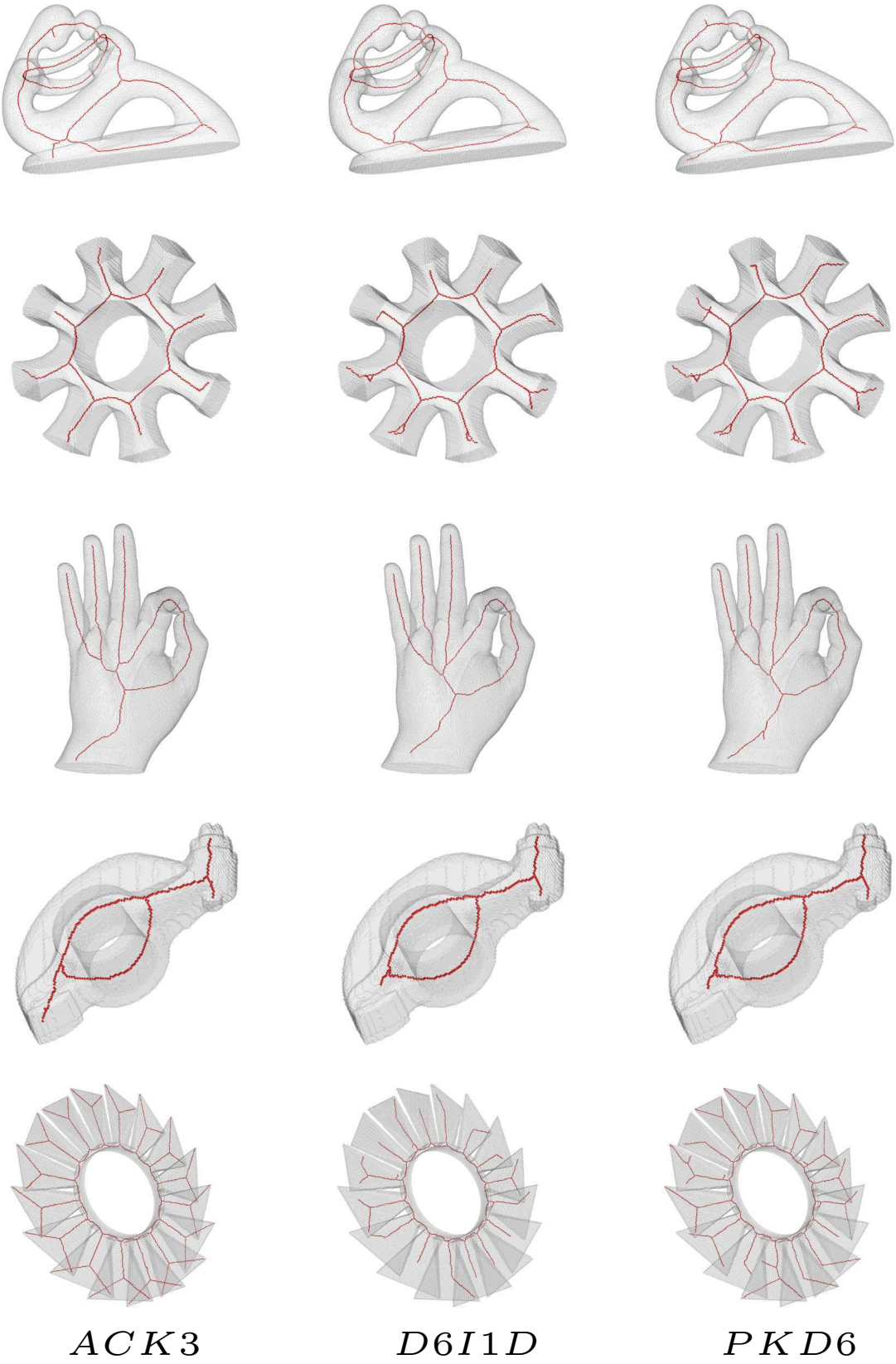


FIGURE 6.6: Skeletons resulting of different algorithms for several shapes. From the left to the right: algorithms ACK3, D6I1D, and PKD6. From the top to the bottom: shapes 15, 17, 5, 10 and 14.

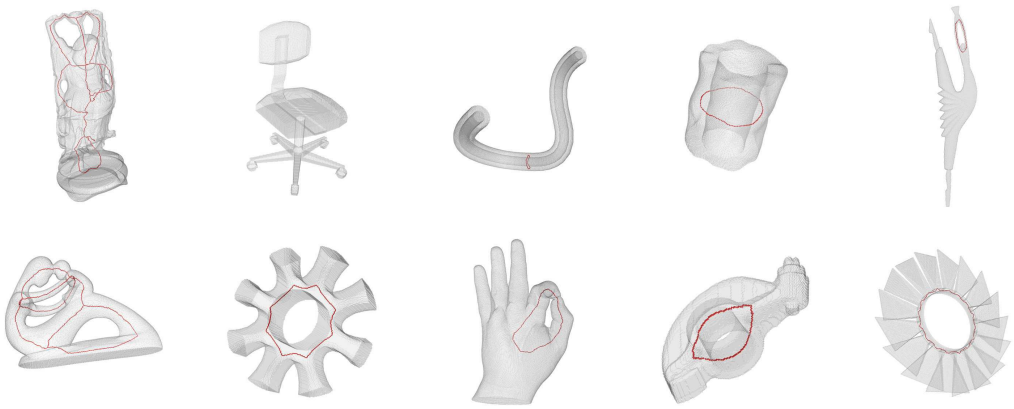


FIGURE 6.7: Examples of homotopic kernels obtained using D6U.

FIGURE 6.8: Surface skeletons obtained using D6I2D.

Part III

Matching to A Priori Model

Chapter 7

Problematics and Usual Definitions

In our motion capture method, we propose to transpose the 3d shape of the subject in a graph. Then, we will perform a matching between this graph and the model graph in order to identify the 3d position of each part described in the model.

We have two problems to solve:

1. find a way to transpose the 3d shape to a usable graph
2. find an appropriate method to match this graph with the model graph

In this chapter, we first provide usual definitions and notations about graph theory which will be used in the following. Then, we explain how we extract a data tree from the shape and what are the possible noises occurring on the data tree which have to be taken into account during the matching.

7.1 Representation of the Shape

In order to perform a fast and efficient matching between two shapes, it is very important to describe them in a concise way. Numerous shape descriptors can be found in the literature. In this section, we propose a quick overview of them.

7.1.1 Global Descriptors

A common class of descriptors is the one of *global descriptors*. A global descriptor represents the whole object, often by a vector of numbers. Global descriptors include:

- classical measurements: perimeter (in 2D), area, volume (in 3D), area and volume of the convex hull
- ratios between measurements
- Fourrier transform
- moments

Global descriptors are very useful for a quick similarity measurement between two shapes, but not suitable for matching.

7.1.1.1 Moments

Moments represent the shape by a numerical sequence (finite or not). The longer the sequence, the more precise the description. If the Cartesian moments are not used as a shape descriptor because non-orthogonal, it is not the case of orthogonal ones, like Legendre moments, used by Mukundan and Ramakrishnan in [121] and Zernike moments used by Novotni and Klein in [124].

7.1.2 Features Distribution

An alternative to global descriptors is the consideration of the distribution of simple features of a shape. Shape distributions have been introduced by Osada et al. in [125]. The authors propose to compute the distribution of simple features of random surface points of the shape: distance between two random points, angles and triangle area for three random points, etc.... The similarity between two shapes is then obtained by computing an appropriate distance between their feature distribution. According to the authors, the feature distribution based on distance between two random points (called "D2 shape distribution") is the one providing experimentally the best results.

Ip et al. proposed in [69] to improve the D2 shape distribution, by separating the distributions of distances for two points, depending on whether the segment between them is fully in the shape ("IN shape distribution"), fully out of it ("OUT shape distribution"), or a mix ("MIXED shape distribution"). The similarity between two shapes is here provided by a weighted combination of the similarities of the four distributions (D2, IN, OUT et MIXED).

7.1.3 Contours

Contours constitute a very intuitive way to describe a 2D shape. In the literature, we can distinguish two different ways to use the contours as a descriptor. Describing the contours as a curve (Bunke and Bühler [30], Gdalyahu and Weinshall [54], Kaygin and Bulut [74]), it is possible to measure the similarity of two shapes by searching the minimal transformation changing one curve to the other. Another way to use the contours is to code it by coefficients in an appropriate base and the similarity is computed by comparing the coefficients (Brun and Pruvot [28], Shen and Shen [159]). Notice that obviously contours description is only usable in 2D.

7.1.4 Spatial Maps

Spatial maps are a way to represent shapes by their spatial distribution. Ankerst et al. [7] propose to segment the space containing the shape in sectors and to compute the histogram of the distribution of the shape in these sectors (a bin representing the filling of the corresponding sector by the shape). The similarity measure is obtained by similarity computation between the histograms representing the shapes, taking into account the distance between the sectors represented by the bins. Such a method is not affine invariant and requires a pose normalization. Kazhdan et al. [75] propose a decomposition in spherical harmonics, providing a rotation invariant descriptor.

7.1.5 Skeleton Based Approaches

Skeleton is an interesting descriptor for several reasons. First, it describes efficiently and concisely the shape and preserves its topology. Secondly, it can be simplified, e.g. by direct graph representation preserving only the shape structure information, or on the opposite can be enriched by adding information on branches, e.g. distance to the complementary. Finally, it yields a lot of different approaches, providing similarity measurement, parts matching, or both.

7.1.5.1 Shock Graphs

From our best knowledge, shock graphs are the most famous skeleton-based approach for shape matching. Siddiqi et Kimia introduced the shock graphs in [160] in 1996: considering the skeleton of a 2D shape, the authors propose to segment its branches in regard to the evolution of radius of the maximal balls along them and for this purpose define four types of evolution:

- *type 1*: continuous and monotone evolution of the radius
- *type 2*: radius is locally minimal

- *type 3*: radius is constant
- *type 4*: radius is locally maximal

Then, a graph is built from these segments, considering types 1 and 3 segments as edges and types 2 and 4 as vertices. Such a graph can be reduced to a tree. The matching and similarity measurement between the shapes are computed from the trees representing them.

7.1.5.2 Skeleton Graph

In 2004, Di Ruberto [148] proposed another representation of the skeleton by a graph: the skeletal graph. Each vertex of a skeletal graph corresponds to an ending point (point with one and only one neighbor) of the skeleton, or to an intersection of branches (point with more than two neighbors). Each edge corresponds to the branch linking such points. Different informations are linked to both vertices (radius of maximal ball) and edges (curvature variation of the corresponding branch, length, orientation, ratio between the size of the branch and the size of the skeleton, etc...). Once again, the matching and the similarity measurement are done via the graphs, using a gradual assignment.

7.1.5.3 Skeletal Shape-Measure

Torsello and Hancock propose in [173] to assign to each branch of the skeleton two segments of the contours, bounded by the points common to the bi-tangents circles centered at the extremities of the branch. Several informations are extracted from these contour segments: curvature evolution, area of the shape part associated to the branch. A measure is computed using these informations, with interesting properties:

- a branch representing a small part of the contour will have a measure smaller than the one of an "important" branch
- the measure homogeneously variates in function of the deformations of the object
- the measure is bending invariant, thus interesting for the matching of articulated shapes

The authors use this measure to improve shock graphs. Notice that, in reason of contours consideration, such measure cannot be directly used for 3D shapes.

7.1.5.4 Skeletal Segments

In 2008, Goh [55] proposed a new way to consider the skeleton, by dividing it into "primary skeletons" in regard of different criteria. These primary skeletons can then be merged depending

to their visual continuity into "secondary skeletons". Several measures (local and global) are linked to these two kinds of skeletons and their combination provides them a weight. The matching of the skeleton is then done by a variation of the Hungarian algorithm on the matrix of these weights.

7.1.5.5 Skeleton Paths

Bai and Latecki propose in [10] a new way to find a matching between two skeletons, considering all the skeleton paths between ending points. Each path is segmented in M equidistant points, and for each point, the radius of the maximal ball centered in this point is considered. Each path is thus represented by a sequence of M radii.

The authors provide a matching approach using this path representation. A distance is computed for each couple of paths (one in each skeleton). For each couple of ending points (one in each skeleton), a distance can be computed, using a new method (called "Optimal Subsequence Bijection") on the set of the distances between all the paths starting by one of the considered points. Finally, the distance between the skeletons is computed using the Hungarian algorithm on the matrix of the distances between each couple of ending points. The Hungarian algorithm provides optimal correspondences for each point.

7.1.6 Choice of the Description

In the case of our application, the problem is slightly different: the matching and similarity computation are not between two shapes, but between a manually described model and a shape. Furthermore, the model has to be simple to express in order to allow for an easy use of our method. Finally, the acquisition of the shape implies different kinds of noise which have to be taken into consideration, and prevent the use of some informations.

Skeletons are interesting in our case, as they allow for both a similarity measurement and a matching between two shapes. Considering the approaches previously described, the skeletal graphs proposed by Di Ruberto [148] are the most adapted, as their definition is intuitive and a similar model is easy to define by the user. Concerning the stored informations on the edges, the length of the branches (and then, the ratio of their length to the size of the skeleton) is the only information which is not noised by the acquisition. We propose to use such description for our method, and explain in the sequel of this chapter how to define it and what are the possible noises encountered which have to be taken into account.

7.2 Usual Definitions on Graphs

As our method will be based on graphs, we propose in this section a brief recall on the basic graph definitions and the notations we will use in this part.

7.2.1 Undirected graphs definitions

7.2.1.1 Undirected graphs

An *undirected graph* is a pair (V, E), where V is a finite set, and E a subset of $\{\{x, y\}, x \in V, y \in V, x \neq y\}$. An element of E is called an *edge*, an element of V is called a *vertex*.

If $\{x, y\} \in E$, then x and y are said to be *adjacent* or *neighbours*. The set of all neighbours of x is denoted by $\mathcal{N}(x)$. The number of vertices adjacent to a vertex v is called the *degree* of v, and is denoted by $deg(v)$. For example, in Figure 7.1, $\mathcal{N}(c) = \{a, d, e\}$ and $deg(c) = 3$.

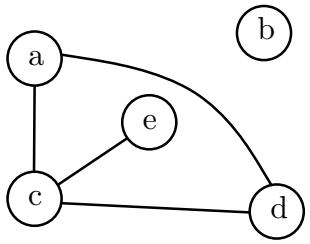


FIGURE 7.1: Example of undirected graph representation. The circles represent the vertices, the letter in each circle is just used in a purpose of easily identify the associated vertex, and the lines represent the edges. The represented graph is $(V = \{a, b, c, d, e\}, E = \{\{a, c\}, \{a, d\}, \{c, d\}, \{c, e\}\})$.

7.2.1.2 Paths and cycles

Let $G = (V, E)$ be an undirected graph, and let x, y be in V, a *walk* from x to y in G is a sequence of vertices $v_0, ..., v_k$ such that $x = v_0$, $y = v_k$ and $\{v_{i-1}, v_i\} \in E, 1 \leq i \leq k$. The number k is called the *length* of the walk. If $k = 0$ the walk is called a *trivial* walk. The walk is *closed* if $x = y$. A *path* is a walk in which no vertex occurs more than once in the sequence of vertices of the path (except possibly $x = y$). A non-trivial closed path in which all edges are distinct is called a *cycle*.

7.2.1.3 Trees and forests

A graph is *connected* if for all $\{x, y\} \subset V$, a path from x to y exists in G. A *tree* is a connected graph with no cycles. A simple path from x to y in a tree is unique and is denoted by $\pi(x, y)$. A graph with no cycles is called a *forest*, each of its connected components being a tree.

Figure 7.2 illustrates the above definitions.

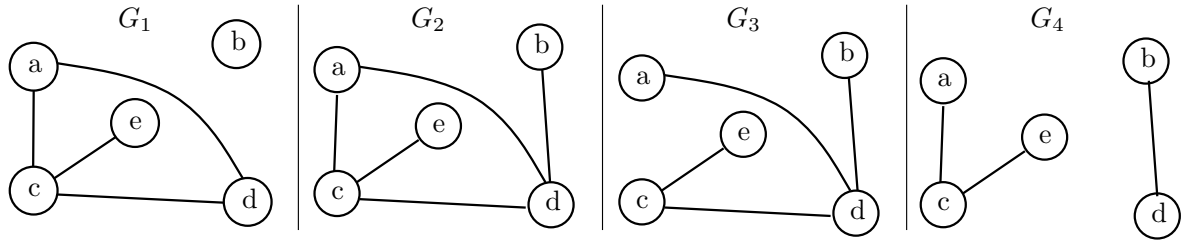


FIGURE 7.2: From the left to the right, four undirected graphs: G_1, G_2, G_3 and G_4. G_2 and G_3 are connected, contrary to G_1 and G_4. G_3 is a tree, contrary to the three others. G_3 and G_4 are forests, composed respectively by one and two trees.

7.2.2 Directed graphs definitions

7.2.2.1 Directed graphs

A *directed graph* is a pair (V, A), where V is a finite set, and A is a subset of $V \times V$. An element of A is called an *arc*, an element of V is called a *vertex*. Figure 7.3 shows a possible representation of directed graphs.

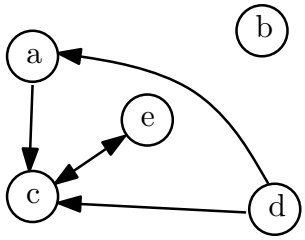


FIGURE 7.3: Example of directed graph representation. The circles represent the vertices, and the arrows represent the arcs. The represented graph is $(V = \{a, b, c, d, e\}, E = \{(a, c), (d, a), (d, c), (c, e), (e, c)\})$.

Let $G = (V, A)$ be a directed graph, and let x, y be in V, a *path* from x to y in G is a sequence of vertices $s_0, ..., s_k$ such that $x = v_0$, $y = v_k$ and $(v_{i-1}, v_i) \in A, 1 \leq i \leq k$.

7.2.2.2 Associated undirected graphs

The *undirected graph associated* to G is the undirected graph $G' = (V, E)$, such that $\{x, y\} \in E$ if and only if $(x, y) \in A$ or $(y, x) \in A$ (see Figure 7.4 for an example).

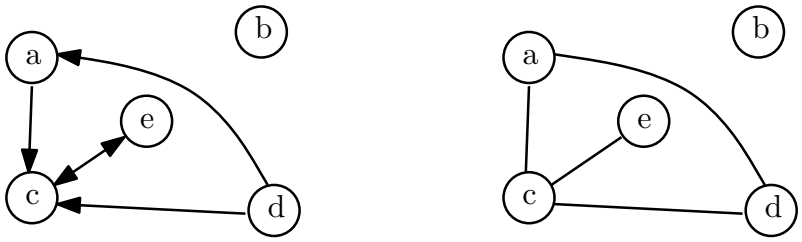


FIGURE 7.4: From the left to the right, a directed graph and its associated undirected graph.

7.2.2.3 Rooted trees and rooted forests

Let G be a directed graph. A vertex $r \in V$ is a root of G if for all $x \in V \setminus \{r\}$, a path from r to x in G exists. The directed graph G is *antisymmetric* if for all $(x, y) \in A$ such that $x \neq y$, $(y, x) \notin A$.

The directed graph G is a *rooted tree* (with root r) if r is the unique root of G, G is antisymmetric and if the undirected graph associated to G is a tree.

A directed graph, where each of its connected components is a rooted tree, is called a *rooted forest*.

Remark 13. in order to avoid possible confusion, trees and forests for undirected case will be respectively called *unrooted trees* and *unrooted forest*.

Let $G = (V, A)$ be a rooted tree. If $(y, x) \in A$, we say that y is the *parent* of x (denoted by $par(x)$), and that x is a *child* of y. The set of all children of y is denoted by $\mathcal{C}(y)$. A vertex with no children is called a *leaf*.

A rooted forest is *ordered* if for all of its vertices, the children are ordered.

The length of a path between the root and a vertex v is called the *depth* of v. The maximum depth of all the vertices is called the *height* of the tree.

The vertices on the path from the root to a vertex x are called the *ancestors* of x. We denote the set of the ancestors of x by $anc(x)$.

Figure 7.5 illustrates the above definitions.

7.2.3 Common definitions

Unless otherwise indicated, all the other definitions and notations are similar for both directed and undirected graphs. We present these definitions for directed graphs, the versions for undirected graphs can be obtained by replacing arcs by edges.

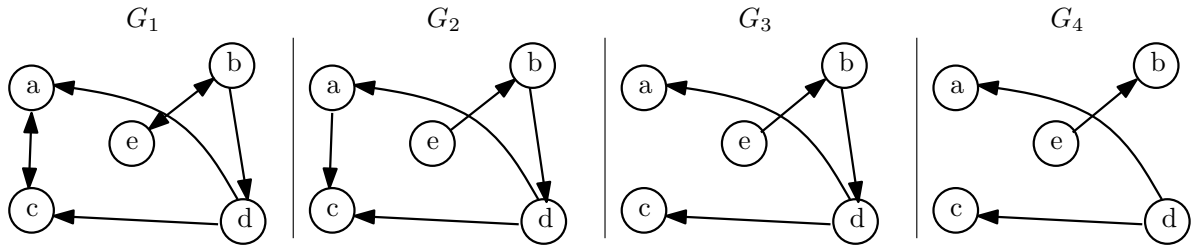


FIGURE 7.5: From the left to the right, four directed graphs: G_1, G_2, G_3 and G_4. G_1 have two roots(b and e), G_2 and G_3 have only one root (e) and G_4 have no root. G_2, G_3 and G_4 are antisymmetric, contrary to G_1. G_3 is a rooted tree, contrary to the three others. G_3 and G_4 are rooted forests, composed respectively by one and two rooted trees.

7.2.3.1 Isomorphism

Two graphs $G = (V_G, A_G)$ and $G' = (V_{G'}, A_{G'})$ are said to be *isomorphic* if there exists a bijection $f : V_G \to V_{G'}$, such as for any pair $(x, y) \in V_G \times V_G$, $(x, y) \in A_G$ if and only if $(f(x), f(y)) \in A_{G'}$ (see Figure 7.6 for some examples).

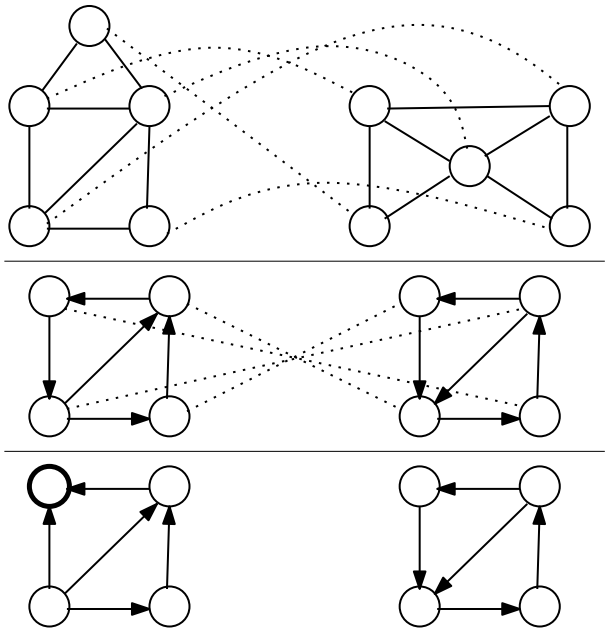


FIGURE 7.6: From the top to the bottom: an example of isomorphism between undirected graphs (dotted lines represent the bijection), an example of isomorphism between directed graphs, and an example of non isomorphism between directed trees (bold vertex cannot find any correspondence respecting the definition).

7.2.3.2 Attributed graphs

An *attributed graph* is a quadruplet $(V, A, \upsilon_V, \upsilon_A)$, where V is a finite set, A a subset of $V \times V$, and υ_V (respectively υ_A) a mapping which associate each element of V (respectively A) to an attribute.

An attribute can be a label (an element of a discrete finite set), a real value, or any other information.

A *vertex-attributed graph* is a triplet (V, A, υ_V) and an *edge-attributed graph* is a triplet (V, A, υ_A), in regard of the above notations.

7.2.3.3 Labeled graphs

A *labeled graph* is a vertex-attributed graph where all attributes are labels (see Figure 7.7 for an example).

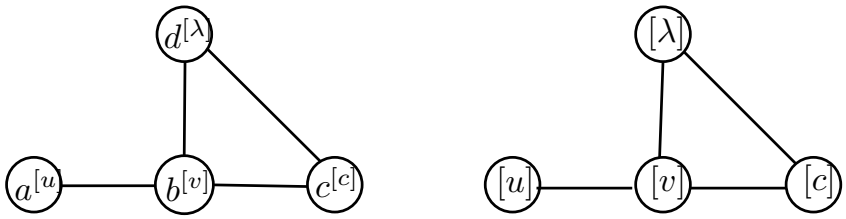


FIGURE 7.7: Example of labeled graph representation. In order to avoid confusion with vertex identifier, the label associated to each vertex is write between brackets. In the case where the identifier must be represented, we use the representation shown on the left graph: the identifier with the label between brackets in exponent. In other case, we just write the label between brackets, like in the right graph.

7.2.3.4 Weighted graphs

A *weighted graph* is an edge-attributed graph where all attributes are non negative reals (see Figure 7.8 for an example).

In a weighted tree, the weight of the unique path from x to y, denoted by $\omega(x, y)$, is the sum of the weights of all arcs traversed in the path.

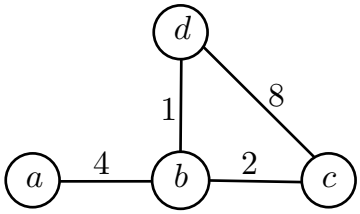


FIGURE 7.8: Example of weighted graph representation. Each number represent the weight of the closest edge.

7.3 Data Tree Extraction

In this section, we will first present the required properties of the graph representation of the shape (called *data tree*). Then, we propose a method to construct the data tree from the skeleton of the shape. Finally we enumerate the different types of noise occurring in the data tree, which have to be taken into consideration for an efficient matching.

7.3.1 Data Tree Specificities

From the definitions presented in the above parts, we can now define formally the graph used in a priori model definition (called the *model tree* or *pattern tree*) as an attributed unrooted tree $M = (V_M, E_M, \Sigma_M, \omega_M)$, with Σ_M a mapping from vertices to labels representing parts of the model and ω_M a mapping from edges to non negative reals representing the distances between the parts.

We can make two observations:

- To match the model tree, the data tree must have approximately the same topology than the model tree, and edge or vertex similar information.
- As the final aim is to obtain spatial positions of the model parts, the data tree have to store spatial informations in the same way that the parts are stored in the model.

The above observations involve that the data tree have to be an attributed unrooted tree $D = (V_D, E_D, \Xi_D, \omega_D)$, with Ξ_D a mapping from vertices to 3d positions of parts of the shape and ω_D a mapping from edges to non negative reals representing the distances between the 3d positions.

Then, the matching can be performed between the weighted unrooted tree (V_M, E_M, ω_M) associated to the model tree and the weighted unrooted tree (V_D, E_D, ω_D) associated to the data tree.

The obtained matching give, for any parts represented by vertices in V_M, a spatial position represented by vertices in V_D.

7.3.2 Data Tree Construction Design

Now we have defined the specificities of the data tree, we have to find a way to build it from the shape. We consider two steps in the design of the building:

1. find a method which automatically provides relevant 3d positions in the shape.

2. find a method which automatically provides both the links between these 3d positions and their appropriate distance.

Figure 7.9 shows areas where we have to find usable 3d positions in the case of hand model (at left) and full human body model (at right). We can observe there exists in all areas at least one intersection or end of branches of curvilinear skeleton of the shape (represented in orange). Furthermore, we can see that areas are linked in a similar way than in the model by branches of the skeleton.

From these observations, we can deduce an efficient method to build the data tree, following the two steps described above:

1. we use intersections and ending points of the skeleton as 3d positions, and represent each of them by a vertex
2. we link two vertices if the associated points are linked by a branch of the skeleton, and the associated weight is the length of the branch

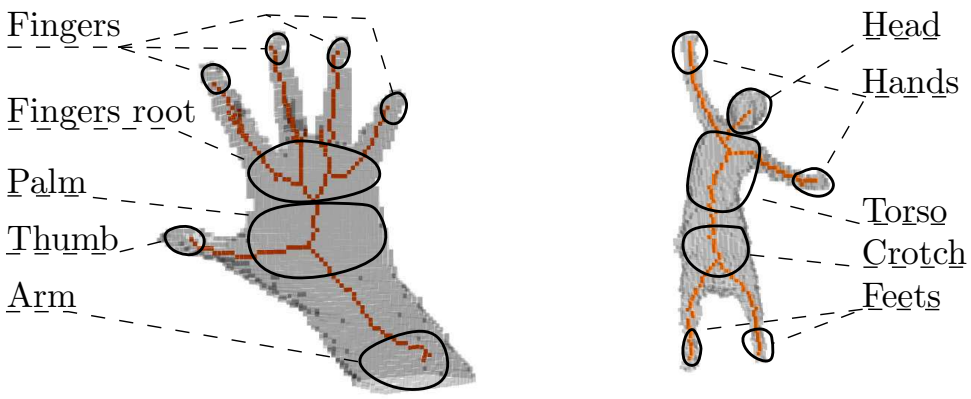


FIGURE 7.9: Areas in the shape where we have to find usable spatial positions to represent in the data tree.

7.3.3 From Skeleton to Data Tree

The skeleton points can be classified into three classes, in regard of their number of neighbors included in the skeleton:

1. zero or one neighbor $\rightarrow$ E-points class (ending points)
2. exactly two neighbors $\rightarrow$ B-points class (branch points)
3. strictly more than two neighbors $\rightarrow$ I-points class (intersection points)

Figure 7.10 shows the distribution of the tree classes in both symmetric and asymmetric skeletons. We can observe that for the asymmetric skeleton, the branches are composed by 2-points and extremities are always E-points. Furthermore, I-points are present only at intersections of branches.

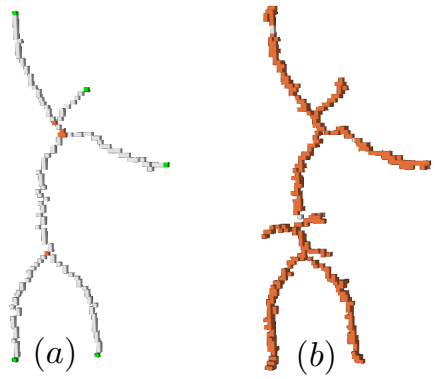


FIGURE 7.10: Classes of skeleton points in asymmetric skeleton (a) and symmetric skeleton (b): E-points are in green, B-points in light grey, and I-points in orange.

In order to construct the data tree of the shape, we will use its asymmetric skeleton, in reason of the above observation. The construction can be done using Algorithm 13, based on detection of connected components of points of a same class (see Figure 7.11 for an example). Empirically, we can considered that data trees obtained by this method have a maximal degree bounded by 6. It is interesting and important to notice it, as a bounded maximal degree allows a better complexity for some matching algorithms.

Algorithm 13: Data tree construction

Data: skeleton S

Result: data tree D

```
1  begin
2      data tree D ← ∅
3      E-CCset ← set of connected components of E-points in S
4      B-CCset ← set of connected components of B-points in S
5      I-CCset ← set of connected components of I-points in S
6      foreach c ∈ E-CCset do
7          add a vertex in D with centroid of c for attribute
8      foreach c ∈ I-CCset do
9          add a vertex in D with centroid of c for attribute
10     foreach c ∈ B-CCset do
11         add an edge between associated vertices of CC touching c, weighted by the size of c
12 end
```

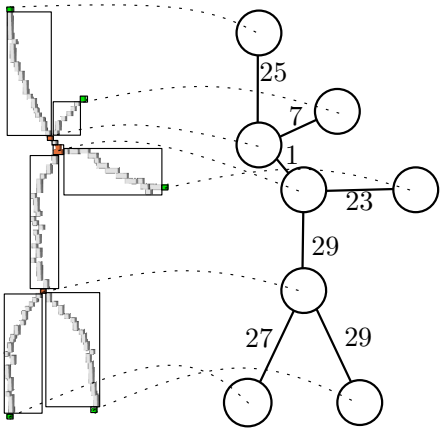


FIGURE 7.11: Left: example of connected components, each rectangle representing a connected component (CC). In this example, there is five E-points CC, seven B-points CC and two I-points CC. Right: data tree constructed from the skeleton at left. Dot lines show the CC associate to each vertex.

7.3.3.1 Incomplete Model Special Case

In the case of an incomplete model, the skeleton is tied to contain at least one border point (point on the boundaries of the grid) by constraining the thinning algorithm. It implies that at least one E-points connected component contains a border point. We consider that all the edges having a bounding vertex associated with such connected component have infinite weight (see Figure 7.12 for an example).

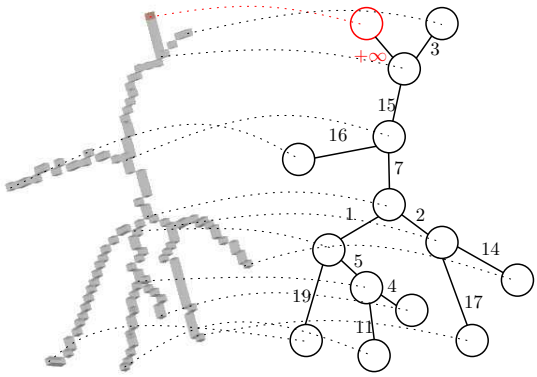


FIGURE 7.12: Example of data tree construction for incomplete model. Left: hand skeleton, with in orange, a border point. Right, the data tree constructed from the skeleton.

7.3.4 Data Tree Noises

Due to the acquisition method, several kinds of noise and deformities can appear in the data tree.

7.3.4.1 Ghost limbs and Spurious branches

Due to the reconstruction from silhouettes, parts of space cannot be carved, resulting in "limbs" of the object which do not exist on the real model. Due to the skeletonization algorithm and to the amount of noise of the shape surface, spurious branches without important topological signification can appear on skeleton. These noises are illustrated in Figure 7.13.

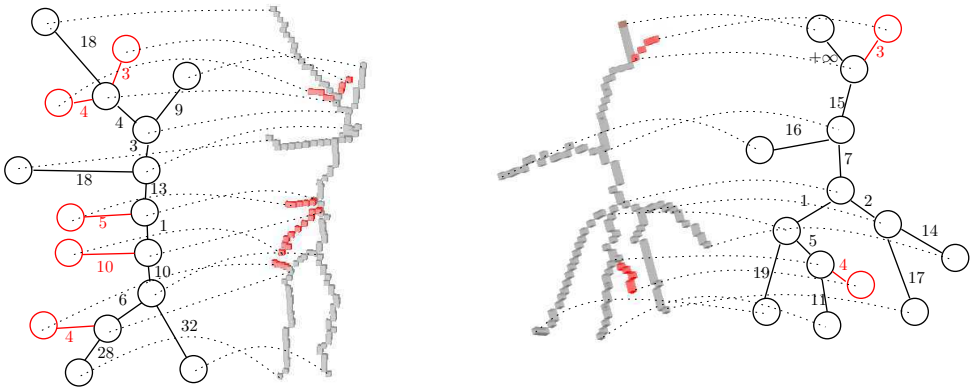


FIGURE 7.13: Examples of spurious branches and ghost limbs(in red) for a full human body shape(left) and a hand shape(right).

7.3.4.2 Useless 2-degree vertices

Vertices with exactly two neighbors are not useful to describe the topology of a shape, and then uselessly split an edge (and its weight) into two parts, making difficult a good matching. This kind of vertices can appear when removing spurious branches or ghosts limbs. See Figure 7.14 for an example.

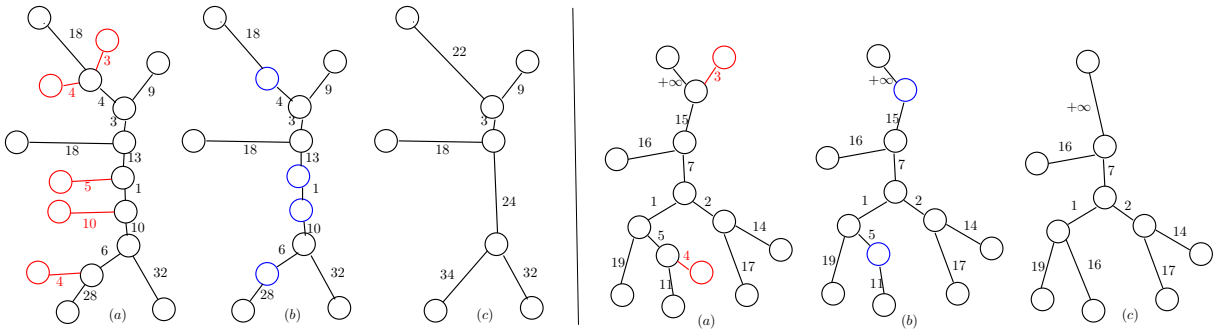


FIGURE 7.14: (a): initial data tree (spurious branches in red). (b): data tree after spurious branches removal (in blue, resulting useless 2-degree vertices). (c): data trees we want to take into consideration during the matching.

7.3.4.3 Splitted vertices

Vertices with more than three neighbors in the model tree can correspond to a cluster of vertices linked by weakly weighted edges in the data tree, due to the skeletonization algorithm. See Figure 7.15 for an example.

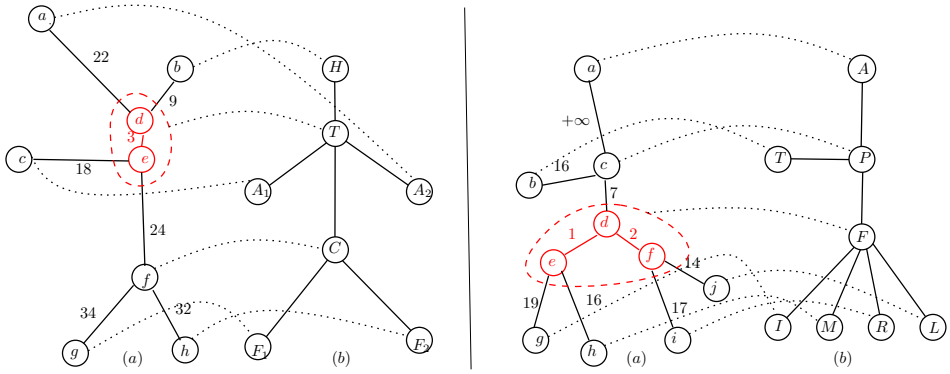


FIGURE 7.15: (a): data tree with splitted vertices (in red). (b): model tree to match. Dotted lines represent the expected matching.

Chapter 8

State of the Art of Edit-Based Matching

Graphs are among the most common and well-studied combinatorial structures in computer science. In particular, the problems of matching graphs and measure the similarity between them occurs in several diverse areas such as compiler design [4], image processing [153], signal processing [34], molecular biology [158], chemistry [164], and many others.

8.1 Similarity and Matching between Graphs

In the case of shape recognition, two graph problems are considered:

- measuring the similarity between two graphs
- matching two graphs

For our application, we request both matching (to find associated positions for each part of the subject) and similarity measurement (to evaluate the significance of the matching). In our context, the graphs to match are acyclic, it is interesting as specific solutions with better complexities have been proposed for such problems on rooted and unrooted trees.

8.1.1 Graph Matching

Graph matching is a very difficult problem and the algorithms providing them are very time consuming. In the field of shape matching, the representations are most often acyclic graphs or rooted trees, which are easier to match.

Tree matching problems can be separate in different classes of problems:

- exact matching problems consist in finding exact patterns in a tree, or finding common subparts of two trees. Examples of exact matching problems are:
  - subtree isomorphism problem [107, 108, 157]: given a pattern tree P and a data tree D, find a subtree of D which is isomorphic to P or decide that there is no such tree.
  - subtree homeomorphism problem [37, 154]: a variant of the subtree isomorphism problem, where 2-degree vertices can be deleted from the data tree.
  - maximal subtree isomorphism [138]: given a pattern tree P and a data tree D, find the biggest subtree of D which is isomorphic to a subtree of P.
- inexact matching problems consist in finding the best approximation of a given pattern in a data tree, or finding the mapping from a tree to another which is the closest to the isomorphism. Different classes of inexact matching exist:
  - one-to-one matching requires that for two trees, each vertex in both trees match with at most one vertex of the other tree.
  - one-to-many and many-to-one matchings allow multiple vertices of only one tree to match with a same vertex of the other tree.
  - many-to-many matching allow clusters of vertices in each tree to match together.

Inexact graph and trees matching are well suited for shape matching and several approaches have been developed to compute them efficiently.

8.1.1.1 Association Graph

Association graphs [12, 136–138] are an interesting tool for both exact and inexact graphs matching. Let consider two graphs $G = (V, E)$ and $G' = (V', E')$, an association graph of G and G' is a graph where the vertex set is $V \times V'$ and an edge link two vertices (v_1, v'_1) and (v_2, v'_2) if and only if a same relation exists between v_1 and v_2 and between v'_1 and v'_2.

The matching between G and G' is then computed by searching the maximal clique in their association graph. The kind of matching will depend of the relation considered for the linking in the association graph.

8.1.1.2 Graph Eigenspace

Another way to obtain an inexact graph matching is to use a graph eigenspace [87], where each vertex of both graphs is represented by an eigenvector, which can be seen as a point in

n-dimensions. Then, the computation of the clusters of eigenvectors in this eigenspace provide the correspondences between the two sets of vertices.

8.1.2 Similarity Measurement

From our best knowledge, the only methods providing a similarity measurement without providing a matching are those based on graph kernels. Graphs kernels are very useful, as they provide an easy way to classify and compare graphs. For two graphs, computing their kernel consists in projecting them in a Hilbert space and to compute the scalar product of these projections. Different projection functions have been proposed, based on random walks [52, 73, 100, 177], tree patterns [101, 142], or bags of paths [24, 165]. Some kernels have also been proposed for trees [178].

8.1.3 Methods Providing both Graph Similarity and Matching

In the trees context, an approach widely used for the matching and similarity measurement is the one using edit-based distances. Edit-based distances are a very powerful and highly studied tool to solve inexact matching problems. It consists in finding the cheapest sequence of operations which transforms the pattern tree into the data tree. In order to solve our specific problem of tree matching, we have developed a new edit-based distance and its associated matching.

In this chapter, we propose a state of art of edit-based distances, in order to position our method.

8.2 Edit-based Distance Basics

In this section, we introduce the important notions necessary to easily understand the following state of art of edit-based distances.

8.2.1 Edit Operations

The most fundamental notion is obviously the notion of edit operation. An edit operation is an atomic treatment which will modify the tree on which it is applied. Depending from both the method and the kind of trees, the defined edit operations will differ. We propose to express only the classical ones on labeled rooted trees, widely used in the literature, and those on weighted trees, as our method has to use it.

Labeled rooted tree edit operations For a labeled rooted tree $G = (V, A, \Sigma)$, classical edit operations are:

Relabel: Change the label of a vertex $v \in V$.

Delete: Delete a non-root vertex $v \in V$ with parent v', making the children of v become the children of v'.

Insert: The dual of delete. Insert a new vertex v as a child of v', making v the parent of a subset of the children of v'.

Examples of these operations are shown in Figure 8.1.

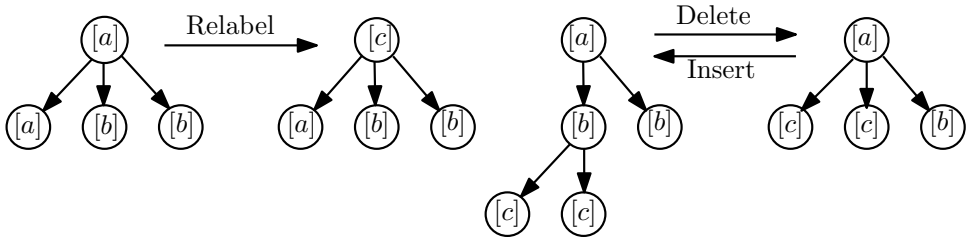


FIGURE 8.1: Examples of edit operations for labeled rooted trees.

Weighted tree edit operations For a graph $G = (V, A, \omega)$, we can translate the edit operations for labelled trees as following:

Resize: Change the weight of an arc $a = (u, v) \in A$.

Delete: Delete an arc $a = (u, v) \in A$ and merge u and v into one vertex.

Insert: Split a vertex in two vertices, and link them by a new arc.

Examples of these operations are shown in Figure 8.2.

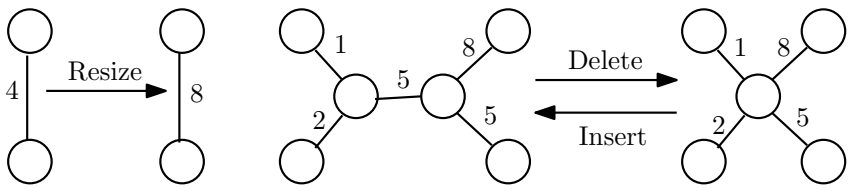


FIGURE 8.2: Examples of edit operations for weighted trees.

8.2.2 Cost Function

A *cost function* is associated to the edit operations. In the literature about labeled trees, it is usual to consider that a relabeling with a different label, the deletion and the insertion have a cost of 1, and the relabeling with the same label have a cost of 0.

For our weighted graph purpose, we use a function $\gamma : \mathbb{R}^+ \times \mathbb{R}^+ \to \mathbb{R}^+$. For a given operation, its cost is $\gamma(w, w')$, where w (respectively w') is the total weight of the arcs involved in the operation before (respectively, after) its application.

As a consequence, the cost of a resizement can be denoted by $\gamma(w, w')$, where w is the former weight of the resized arc and w' is the new weight, the cost of a deletion can be denoted by $\gamma(w, 0)$, where w is the weight of the deleted arc, and the cost of an insertion, $\gamma(0, w)$, where w is the weight of the created arc.

We choose γ to be a *translation invariant metric*, and is required to satisfy the following constraints:

- $\gamma(w, w') \geq 0$ (non-negativity constraint)
- $\gamma(w, w') = 0$ if and only if $w = w'$ (identity of indiscernible constraint)
- $\gamma(w, w') = \gamma(w', w)$ (symmetry constraint)
- $\gamma(w, w') \leq \gamma(w, x) + \gamma(x, w')$ (triangle inequality constraint)
- $\gamma(w, w') = \gamma(w + x, w' + x)$ (translation invariance constraint)

Proposition 14. $\gamma(a + b, c) \leq \gamma(a, 0) + \gamma(b, c)$

Proof. Due to triangle inequality, we have $\gamma(a + b, c) \leq \gamma(a + b, b) + \gamma(b, c)$. Due to translation invariance, we have $\gamma(a, 0) = \gamma(a + b, b)$. Thus, $\gamma(a + b, c) \leq \gamma(a, 0) + \gamma(b, c)$. □

8.2.3 Tree Edit Distance

An *edit script* S between two graphs G_1 and G_2 is a sequence of edit operations turning G_1 into G_2. The cost of S is the sum of the costs of the operations in S. An *optimal edit script* between G_1 and G_2 is an edit script between G_1 and G_2 of minimum cost and this cost is the *tree edit distance*.

8.2.4 Mapping

An *edit distance mapping* (also just called a *mapping*) between two graphs G_1 and G_2 is a representation of the edit script.

Mapping for labeled rooted trees In the literature, a mapping between two labeled ordered rooted trees $T_1 = (V_1, A_1, \Sigma_1)$ and $T_2 = (V_2, A_2, \Sigma_2)$ can be formally defined as the triplet (M, T_1, T_2), with $M \subseteq V_1 \times V_2$ and for any pair $(v_1, w_1), (v_2, w_2) \in M$:

- $v_1 = v_2$ if and only if $w_1 = w_2$ (one-to-one condition)
- $v_1 \in anc(v_2)$ if and only if $w_1 \in anc(w_2)$ (ancestor condition)
- v_1 is to the left of v_2 if and only if w_1 is to the left of w_2 (sibling condition)

An element of M is called a *mapping line*.

The mapping for labeled unordered rooted tree is defined in the same way, but without the sibling condition.

An example of mapping between labeled rooted trees is shown in Figure 8.3.

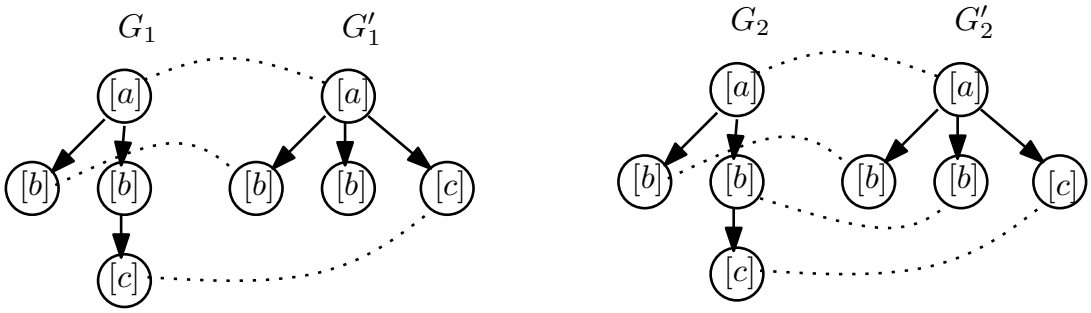


FIGURE 8.3: Left: A mapping from G_1 to G'_1. The dotted lines represent the mapping. Vertices in G_1 not touched by a mapping line have to be deleted. Vertices in G'_1 not touched by a mapping line have to be inserted. The mapping shows a way to transform G_1 to G'_1. Right: An invalidate mapping from G_2 to G'_2: the ancestor condition is not respected.

Remark 15. In the sequel, we will present some mappings with no mapping line between the roots. As it is in contradiction with the definition of delete operation, which occurs only on non-rooted vertices, we have to consider that the deletion of the root r_1 of T_1, which split T_1 in the forest of all subtrees rooted in a vertex $\in \mathcal{C}(r_1)$, is possible in two cases:

1. the root r_2 of T_2 is deleted too, then the mapping occurs on the forests of both trees.
2. the root r_2 of T_2 is mapped to a vertex v in T_1, such that v is an ancestor of all vertices mapped with vertices of T_2 (due to ancestor condition).

Mapping for weighted trees In the case of weighted trees, we can adapt the above definitions of mapping.

A mapping between two weighted rooted trees $T_1 = (V_1, A_1, \omega_1)$ and $T_2 = (V_2, A_2, \omega_2)$ can be formally define as the triplet (M, T_1, T_2), with $M \subseteq A_1 \times A_2$ and for any pair $(a_1 = (v_1, v_1'), b_1 = (w_1, w_1')), (a_2 = (v_2, v_2'), b_2 = (w_2, w_2')) \in M$:

- $a_1 = a_2$ if and only if $b_1 = b_2$ (one-to-one condition)
- $v_1' \in anc(v_2')$ if and only if $w_1' \in anc(w_2')$ (ancestor condition)

An example of invalidate mapping between weighted trees is shown in Figure 8.4.

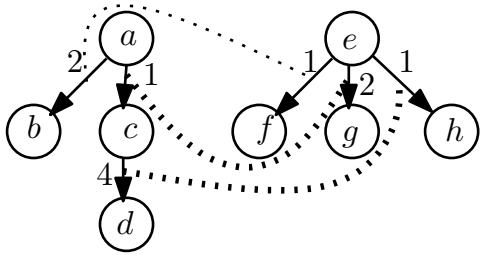


FIGURE 8.4: An invalidate mapping: the ancestor condition is not respected (due to bold pair: c is an ancestor of d, contrary to g which is not an ancestor of h).

Mapping Cost By definition, the cost of a mapping is equal to the cost of the associated edit script.

8.2.5 Complexities notations

In the sequel of the chapter, we will give the complexity of algorithms computing edit distances between two trees T_1 and T_2, mainly in function of their size, their number of leaves, their height and their maximal degree. For $i \in \{1, 2\}$:

- the size of T_i is denoted by S_i
- the number of leaves in T_i is denoted by L_i
- the height of T_i is denoted by H_i
- the maximum degree of T_i is denoted by D_i

8.3 General Edit Distance

The edit distance problem between labeled ordered rooted trees was introduced by Tai in [168]. It consists in finding the optimal edit script and the associated mapping (called the *optimal mapping*) between labeled ordered rooted trees. Notice that the optimal mapping is not necessary unique. In [187], Zhang and Shasha proposed an algorithm with an $O(S_1 S_2 min(L_1, H_1) min(L_2, H_2))$ time complexity. This algorithm was modified by Klein in [77] to get a better worst case time complexity in $O(S_1^2 S_2 log S_2)$.

In 2001, Chen [33] proposed an algorithm with $O(S_1 S_2 + L_1^2 S_2 + L_1^{2.5} L_2)$, which improves previous bounds for certain kinds of trees (for example, if $L_i < \sqrt{S_i}, i = 1, 2$). More recently, Demaine et al. [46] proposed an algorithm in $O((S_1 + S_2)^3)$ time complexity.

For the edit distance problem between labeled unordered rooted trees, Zhang et al. [188] shown that the problem was NP-complete, and proposed an algorithm with an $O(S_1 S_2 + L_1! 2^{L_1} (L_1^3 + H_2^2) S_2)$ time complexity.

8.4 Other Edit Distances

Numerous other edit distances have been proposed in order to reduce the complexity of the general edit distance problem or to provide special characteristics to the associated mapping. These edit distances are derived from the general edit distance by adding constraints on the mapping, on the edit operations or on the edit script. We will here propose a quick overview of these edit distances.

8.4.1 Alignment Distance

Jiang et al. proposed in [71] the notion of *alignment of trees*. Let $T_1 = (V_1, A_1, \Sigma_1)$ and $T_2 = (V_2, A_2, \Sigma_2)$ be two labeled rooted trees. We denote by λ a *space*, which is a label not occurring in T_1 and T_2.

An alignment of T_1 and T_2 is obtained by first inserting vertices labeled by λ in T_1 and T_2 such that the two resulting trees T_1' and T_2' are isomorphic, then drawing a mapping line from each vertex in T_1' to its corresponding vertex in T_2'. See left part of Figure 8.5 for an example.

An alignment mapping can be provided between T_1 and T_2 by drawing a mapping line between each pair $(v_1, w_1) \in V_1 \times V_2$, if there is a mapping line between them in the alignment. See right part of Figure 8.5 for an example.

Jiang et al. provided an algorithm for the alignment between labeled ordered rooted trees with $O(S_1S_2(D_1 + D_2)^2)$ time complexity. In the unordered case, the problem is NP-complete, excepted if D_1 and D_2 are bounded. In this case, the complexity is in $O(S_1S_2)$.

In 2001, Jansson and Lingas [70] proposed an algorithm to compute the optimal alignment between *similar* trees. Two trees are similar if there is an optimal alignment between them using at most s spaces. Their algorithm had a $O(S_1 + S_2)log(S_1 + S_2)(D_1 + D_2)^4s^2)$ time complexity.

Remark 16. In his thesis [65], Höchsmann shown that an alignment distance is not a distance, due to non respect of triangle inequality constraint.

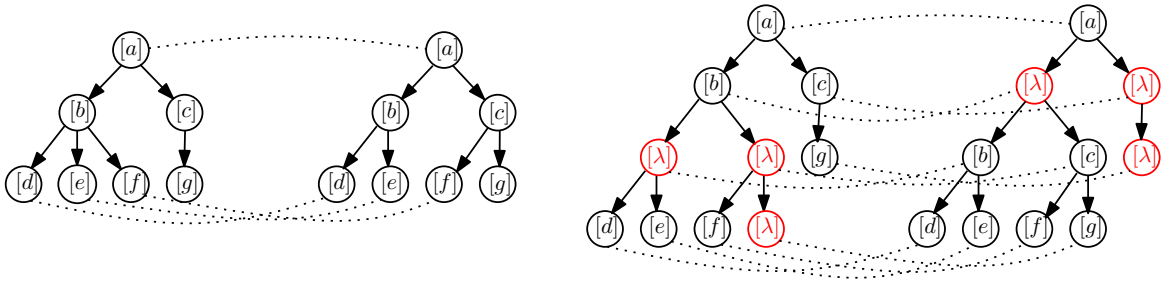


FIGURE 8.5: Left: an alignment mapping. Right: the associated alignment.

8.4.2 Constrained/Isolated Subtree Edit Distance

According to Wang and Zhang [179], the *isolated subtree edit distance* (also called the *constrained edit distance* in the litterature [21]) was first proposed in 1988 by Tanaka and Tanaka [170] and involves that disjoint subtrees are mapped with disjoint subtrees.

A constrained mapping can be obtained by adding a condition on mapping. In a rooted tree $T = (V, A)$, we denote by $nca(x, y)$ the nearest common ancestor of x and y in T, which is the vertex in $anc(x) \cap anc(y)$ with maximal height. The additional condition for the mapping (M, T_1, T_2) is, for any triplet (v_1, w_1), (v_2, w_2), $(v_3, w_3) \in M$:

- $nca(v_1, v_2) \in anc(v_3) \setminus \{v_3\}$ if and only if $nca(w_1, w_2) \in anc(w_3) \setminus \{w_3\}$.

An example of constrained mapping is shown in Figure 8.6.

For the application between labeled ordered rooted trees, Zhang [185] gives an algorithm with a time complexity in $O(S_1S_2)$. In [186], he proposed an algorithm for the unordered case with a $O(S_1S_2(D_1 + D_2)log(D_1 + D_2))$ time complexity.

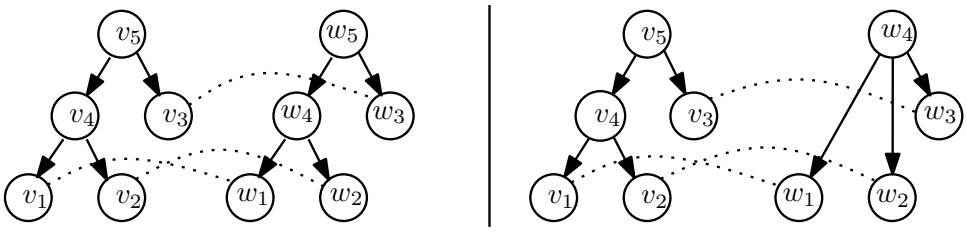


FIGURE 8.6: Left: a mapping which is constrained. Right: a mapping which is not constrained.

In 2003, Ferraro and Godin [49] proposed an algorithm to compute the optimal constrained edit mapping with a minimal number of connected components in both trees, with the same complexity than Zhang.

Constrained Mapping for labeled unrooted trees In [189], Zhang et al. proposed a definition of constrained mapping for labeled unrooted trees, using the definition of the center of three vertices.

Let u, v, w be three vertices in a labeled unrooted tree T, the *center* of the three vertices u, v, w, denoted by $center(u, v, w)$, is the only vertex belonging to the three paths $\pi(u, v)$, $\pi(u, w)$ and $\pi(v, w)$ (see Figure 8.7 for an example).

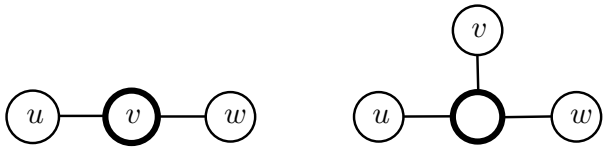


FIGURE 8.7: Illustrations of the center, represented by the bold circle.

A constrained mapping between two labeled unrooted trees $T_1 = (V_1, E_1, \Sigma_1)$ and $T_2 = (V_2, E_2, \Sigma_2)$ can be formally defined as the triplet (M, T_1, T_2), with $M \subseteq V_1 \times V_2$ and for any triplet (v_1, w_1), (v_2, w_2), $(v_3, w_3) \in M$:

- $v_1 = v_2$ if and only if $w_1 = w_2$ (one-to-one condition)
- $(c, c') \in M$, with $c = center(v_1, v_2, v_3)$ and $c' = center(w_1, w_2, w_3)$ (center relationship preservation condition)

An example of invalidate constrained mapping is shown in Figure 8.8.

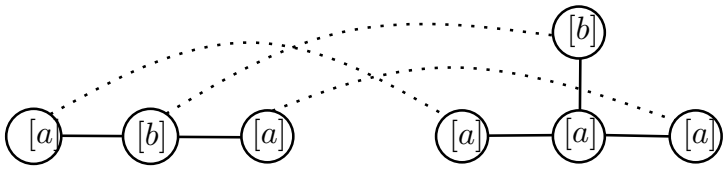


FIGURE 8.8: An invalidate constrained mapping: the center relationship preservation condition is not respected.

8.4.3 Less Constrained Edit Distance

In 2001, Lu et al. [94] introduced the *less-constrained edit distance*, which relaxes the constrained mapping. Here, the additional condition for the mapping (M, T_1, T_2) is, for any triplet $(v_1, w_1), (v_2, w_2), (v_3, w_3) \in M$, such none of $v_1, v2$ and v_3 is an ancestor of the others:

- *depth of* $nca(v_1, v_2) \geq$ *depth of* $nca(v_1, v_3)$ and also $nca(v_1, v_3) = nca(v_2, v_3)$ if and only if *depth of* $nca(w_1, w_2) \geq$ *depth of* $nca(w_1, w_3)$ and also $nca(w_1, w_3) = nca(w_2, w_3)$

The authors proposed an algorithm for this edit distance between labeled ordered rooted trees with a $O(S_1 S_2 D_1^3 D_2^3 (D_1 + D_2))$ time complexity. In the unordered case, the problem is NP-complete. An example of less constrained mapping is shown in Figure 8.9.

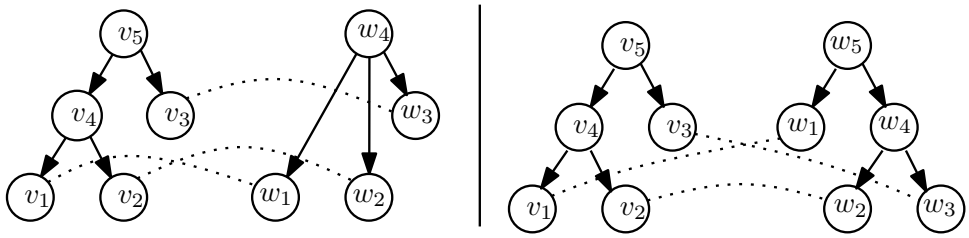


FIGURE 8.9: Left: a mapping which is not constrained but less constrained. Right: a mapping which is not less constrained.

8.4.4 1-Degree/Top-Down Edit Distance

In [155], Selkow proposed an edit distance with for additional constraint that insertions and deletions are restricted to leaves of the tree. This edit distance is called *1-degree edit distance* or *top-down edit distance*, due to the translation of this constraint in the mapping.

Let $T_1 = (V_1, A_1, \Sigma_1)$ and $T_2 = (V_2, A_2, \Sigma_2)$ be two labeled trees rooted respectively in r_1 and r_2. A mapping (M, T_1, T_2) is a top-down mapping if for all couple $(v_1, w_1) \in (V_1 \setminus \{r_1\}) \times (V_2 \setminus \{r_2\})$:

- $(v_1, w_1) \in M$ if and only if $(par(v_1), par(w_1)) \in M$

An example of top-down mapping is shown in Figure 8.10.

Yang proposed in [183] an algorithm which computes the top-down edit distance between labeled ordered rooted trees with a $O(S_1 S_2)$ time complexity. For our best knowledge, there does not exist algorithm for the unordered case.

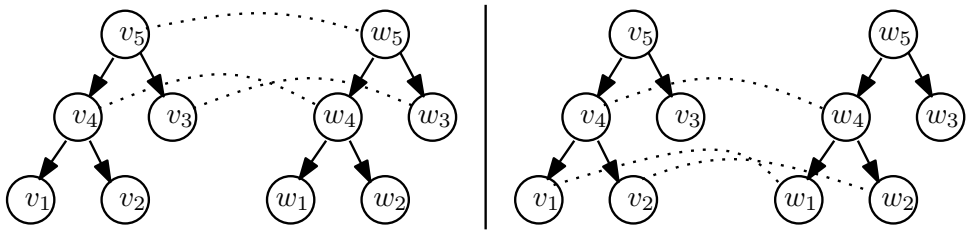


FIGURE 8.10: At left: a mapping which is top down. At right: a mapping which is not top down.

8.4.5 2-Degree Edit Distance

Zhang et al. proposed in [189] a restricted form of the constraint edit mapping, with for additional constraint that insertions and deletions are restricted to vertices with no more than two neighbors.

The authors provide algorithms for labeled unordered rooted trees with $O(S_1 S_2 \sqrt{d}\ log\ d)$ time complexity (with $d = min(D_1 D_2)$), and for labeled unrooted trees with $O(S_1 S_2 d \sqrt{d}\ log\ d)$ time complexity.

8.4.6 Bottom-Up Edit Distance

Valiente proposed in [175] a bottom up mapping defined by adding the following constraint.

Let $T_1 = (V_1, A_1, \Sigma_1)$ and $T_2 = (V_2, A_2, \Sigma_2)$ be two labeled trees rooted respectively in r_1 and r_2. A mapping (M, T_1, T_2) is a bottom up mapping if for all couple $(v_1, w_1) \in M$:

- $(v_1, w_1) \in M$ then $\forall c_i \in \mathcal{C}(v_1), \exists c_j \in \mathcal{C}(w_1)$ such $(c_i, c_j) \in M$ and $\forall c_j \in \mathcal{C}(w_1), \exists c_i \in \mathcal{C}(v_1)$ such $(c_i, c_j) \in M$.

An example of bottom-up mapping is shown in Figure 8.11.

The algorithm provided by Valiente to compute the bottom up distance have a complexity in $O(S_1 + S_2)$ for both ordered and unordered cases.

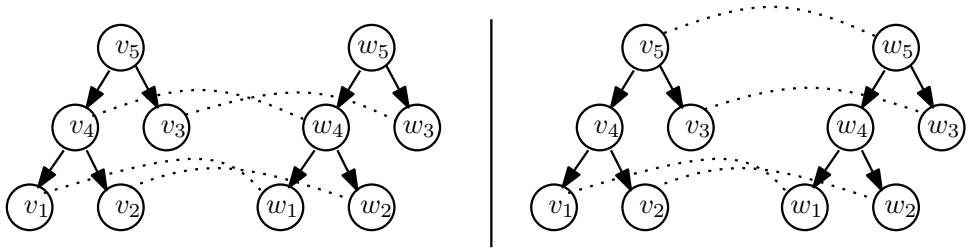


FIGURE 8.11: At left: a mapping which is bottom up. At right: a mapping which is not bottom up.

8.5 Inclusion of Mappings

Mappings based on general edit mapping can be hierarchically classified in regard to their constraints. For example, we can assume that all mappings derived from the general edit mapping by adding restriction are general edit mappings. Some of these relations are directly derived from the mappings definitions:

- all constrained (or isolated subtree) mappings are less constrained mappings.
- all 1-degree (or top down) mappings are 2-degree mappings.
- all 2-degree mappings are isolated subtree mappings.
- all bottom-up mappings are isolated subtree mappings.

In 2001, Wang and Zhang [179] demonstrated that all isolated subtree mappings are alignment mappings. From all these relations, we can established a hierarchy on all the mappings described above. This hierarchy is shown on Figure 8.12.

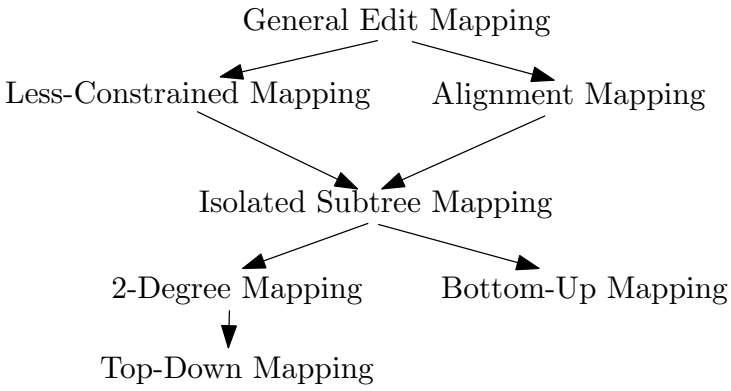


FIGURE 8.12: Hierarchy of mappings.

8.6 Edit Distances with Extended Set of Operations

Edit distances using additional operations have also been proposed, in order to treat specific problems, as shape matching for example.

8.6.1 Edge Merging and Edge Pruning

Klein et al. [78] proposed a modified edit distance on edge-attributed ordered rooted trees using two new operations:

edge merging Merges two edges sharing a 2-degree vertex. The resulting attribute is derived from the attributes of involved edges.

edge pruning Removes an edge ended by a 1-degree vertex at one side and a 3-degree vertex v at the other side. Then, the two edges sharing v are merged.

8.6.2 Cut Operation

In [180], Wang et al. proposed to add the *cut operation*, which consist in removing complete subtrees in one of the two trees for free. It allows to use edit distance for inexact subtree matching problem.

8.6.3 Horizontal/Vertical Merge and Split

In 2003, Bille [20] proposed to extend the general edit distance for labeled ordered rooted trees using *merge and split operations*:

horizontal-merge Merges a consecutive subsequence of siblings $v_1, ..., v_s$ into a single vertex v. The children of $v_1, ..., v_s$ become the children of v.

horizontal-split The complement of horizontal-merge. Splits a vertex v into a consecutive sequence of siblings $v_1, ..., v_s$. The children of v become children of $v_1, ..., v_s$.

vertical-merge Merge a sequence of vertices $v_1, ..., v_s$, where $par(v_{i+1}) = v_i, 1 \leq i < s$, into a single vertex v. The children of $v_1, ..., v_s$ not in the sequence become the children of v.

vertical-split The complement of vertical-merge. Split a vertex v into a sequence of vertices $v_1, ..., v_s$, where $par(v_{i+1}) = v_i, 1 \leq i < s$. The children of v become the children of the sequence $v_1, ..., v_s$.

Bille proposed two new edit distances using these operations. The *k-way horizontal edit distance* consists in allowing horizontal merges and splits in the edit script, with the restriction to merge less than k vertices together. The author provides an algorithm that reaches $O(S^{2k+2})$ time complexity.

The *vertical edit distance* consists in allowing vertical merges and splits in the edit script. An algorithm is proposed with a $O(S^7)$ time complexity.

We can notice that the use of merge operations involve to break the one-to-one constraint of edit mapping: this kind of mapping is *many-to-many*.

8.7 Discussion for our Purpose

In our purpose, none edit mapping is well adapted to take into consideration the noises, especially the 2-degree useless vertex noises and the spurious branches. However, we can consider that these two problems can be solved by adding respectively merge operation of Klein et al. [78] and the cut operation of Wang et al. [180].

The last remaining problem, the splitted vertex problem, can be taken into consideration by edit mappings, using the edge deletion. Among the edit mappings, we will use the one which well preserves the topology and which has a good complexity. Isolated subtree mapping and its sub cases (regarding the mapping hierarchy previously established in Figure 8.12) are too restrictive. The alignment mapping seems a good compromise between topology preservation and computation speed.

8.8 Alignment Distance for Weighted Trees

In [71], Jiang et al. proposed the alignment between vertex-labeled trees, that we transpose here for edge-weighted graphs.

Let $G_1 = (V_1, A_1, \omega_1)$ and $G_2 = (V_2, A_2, \omega_2)$ be two weighted graphs. Let $G'_1 = (V'_1, A'_1, \omega'_1)$ and $G'_2 = (V'_2, A'_2, \omega'_2)$ be weighted graphs obtained by inserting arcs weighted by 0 in G_1 and G_2, such that there exists an isomorphism $\mathcal{I}$ between (V'_1, A'_1) and (V'_2, A'_2). The set of all couples of arcs $\mathcal{A} = \{(a_1, a_2); a_1 \in A'_1, a_2 \in A'_2, a_2 = \mathcal{I}(a_1)\}$ is called an alignment of G_1 and G_2. The *cost* $C_\mathcal{A}$ of $\mathcal{A}$ is given by

$$C_\mathcal{A} = \sum_{(a_1,a_2)\in\mathcal{A}} \gamma(\omega'_1(a_1), \omega'_2(a_2)) \ . \tag{8.1}$$

The minimal cost of all alignments from G_1 and G_2, called the *alignment distance*, is denoted by $\alpha(G_1, G_2)$.

An example of alignment and alignment mapping between weighted trees is shown in Figure 8.13.

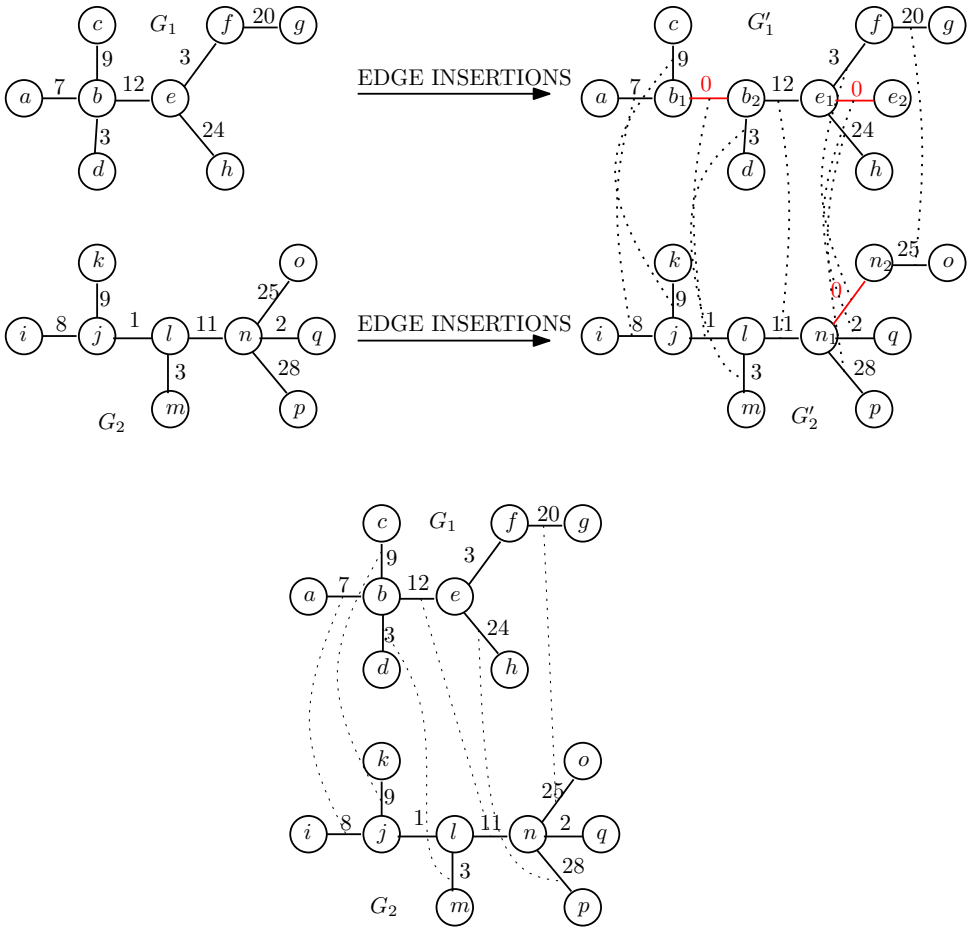


FIGURE 8.13: Example of alignment for weighted trees. Top left: initial trees G_1 and G_2. Top right: trees after edge insertions G_1' and G_2', dotted lines representing the alignment. Bottom: associated alignment mapping of G_1 and G_2.

Chapter 9

Homeomorphic Alignment

As shown in the previous chapter, there is no matching method fitted for our need in the literature, as far as we know. However, we have seen that solutions exist for the three kind of noises, considered separately:

- alignment mapping seems a good compromise between time complexity and topology preservation, and solve the problem of splitted vertices
- merging operations on edges can be a solution for useless 2-degree vertices
- cut operation is an interesting tool for spurious branches removing

In this chapter we propose a new kind of alignment, specially designed to take into account the problem of useless 2-degree vertices, by extending the set of edit operations. First, we give a formal definition of this alignment. Then, we show how to compute it efficiently for both rooted trees and unrooted trees, using bottom-up based algorithms. Finally, we show how to integrate the cut operation in this alignment, in order to solve the problem of spurious branches.

9.1 Preliminar Definitions

9.1.1 Merging Operation

The *merging* is an operation that can be applied only on arcs sharing a 2-degree vertex (for directed graphs, the vertex have to be the end of one arc and the start of the other). The merging of two arcs (u, v) and (v, w) (we say that we *merge on vertex* v) in a weighted graph $G = (V, A, \omega)$ consists of removing v in V, replacing (u, v) and (v, w) by (u, w) in A, weighted by $\omega((u, w)) = \omega((u, v)) + \omega((v, w))$. Notice that the cost of a merging is equal to 0, as no weight variation occurs. Examples of merging for directed and undirected graphs are shown in Figures 9.1 and 9.2, respectively.

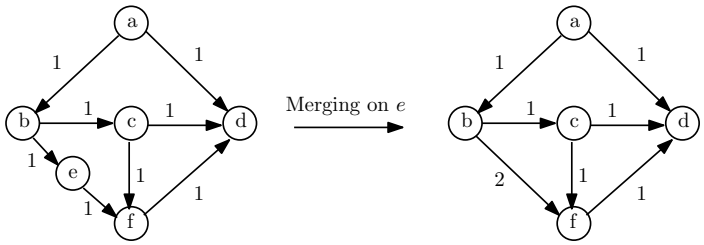


FIGURE 9.1: Example of merging on directed graph.

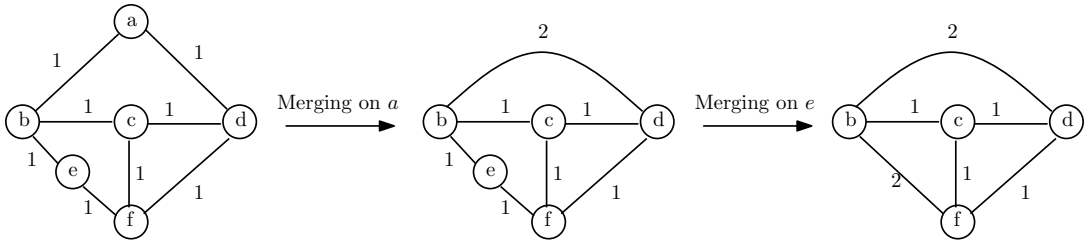


FIGURE 9.2: Examples of merging on undirected graph.

9.1.2 Homeomorphism

Two weighted graphs $G = (V_G, A_G, \omega_G)$ and $G' = (V_{G'}, A_{G'}, \omega_{G'})$ are *homeomorphic* [181] if and only if there exists an isomorphism between a graph obtained by mergings on G and a graph obtained by mergings on G'. Some examples of homeomorphism are shown in Figure 9.3.

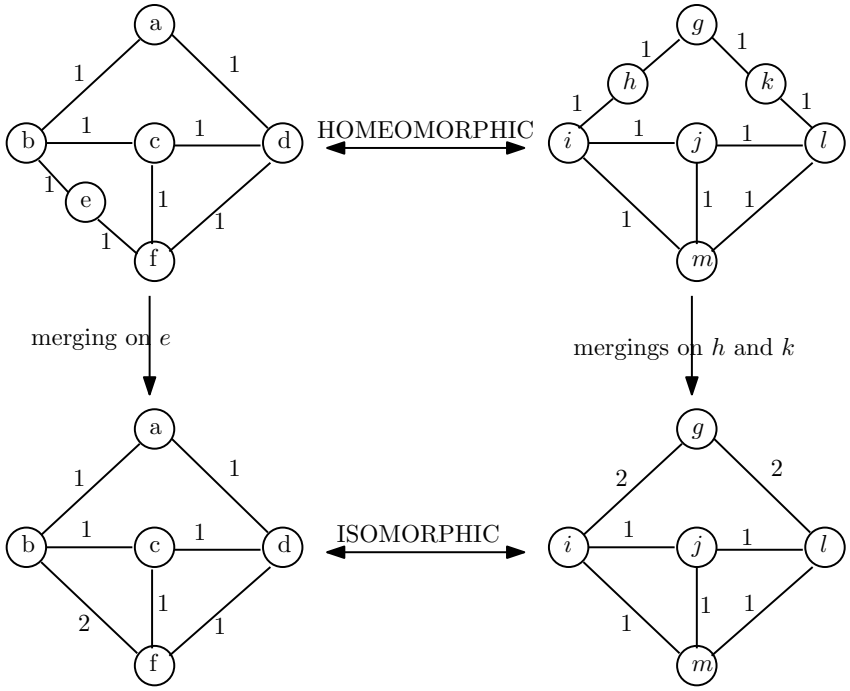


FIGURE 9.3: Examples of homeomorphism.

9.1.3 Merging Kernel

Considering that a merging on a vertex v on the graph $G = (V, A, \omega)$ does not affect the degree of any vertex in $V \setminus \{v\}$ (by definition of merging operation) and therefore the possibility of merging this vertex, the number of possible mergings decreases by one after each merging.

In consequence, the maximal size of a sequence of merging operations, transforming G into another graph $G' = (V', A', \omega')$ is equal to the initial number of possible mergings in G.

It can be remarked that any sequence of merging operations of maximal size yields the same result. The graph resulting of such a sequence on G is called the *merging kernel* of G, and is denoted by $\mathcal{MK}(G)$.

The following definition is straightforward:

Definition 17. *Two graphs $G_1 = (V_1, A_1, \omega_1)$ and $G_2 = (V_2, G_2, \omega_2)$ are homeomorphic iff $\mathcal{MK}(G_1)$ and $\mathcal{MK}(G_2)$ are isomorphic.*

Examples of merging kernels of directed and undirected graphs are shown in Figures 9.5 and 9.4, respectively.

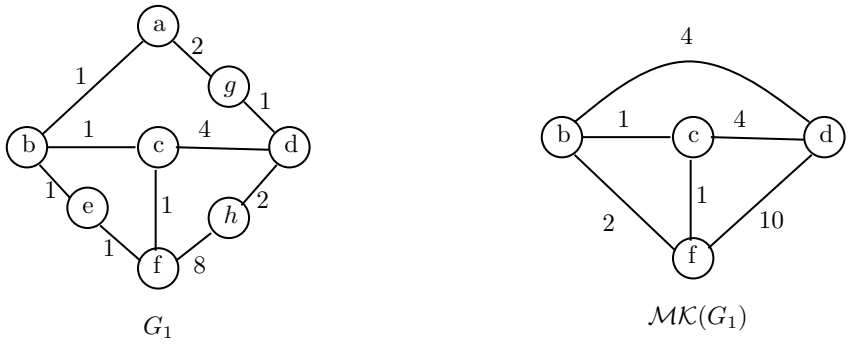


FIGURE 9.4: Example of merging kernel of an undirected graph.

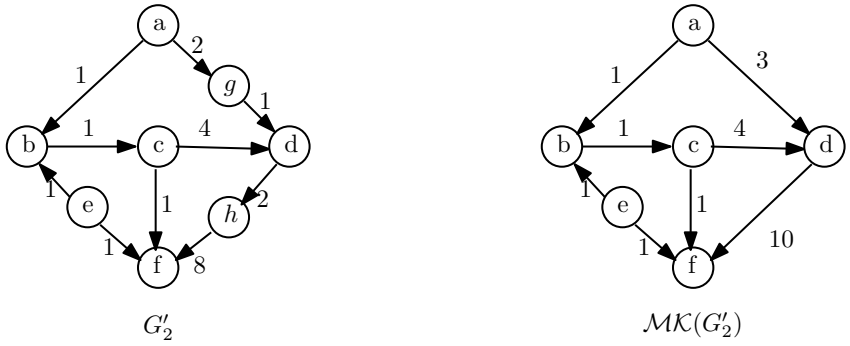


FIGURE 9.5: Example of merging kernel of a directed graph.

9.2 Homeomorphic Alignment Definition

Let $G_1 = (V_1, A_1, \omega_1)$ and $G_2 = (V_2, A_2, \omega_2)$ be two weighted graphs. Let $G'_1 = (V'_1, A'_1, \omega'_1)$ and $G'_2 = (V'_2, A'_2, \omega'_2)$ be weighted graphs obtained by deleting arcs in G_1 and G_2, such that there exists an homeomorphism between G'_1 and G'_2 (not necessarily unique). Let $G''_1 = (V''_1, A''_1, \omega''_1)$ and $G''_2 = (V''_2, A''_2, \omega''_2)$ be the merging kernel of G'_1 and G'_2, respectively. From definition 17, there exists an isomorphism $\mathcal{I}$ between G''_1 and G''_2.

The set of all couples of arcs $\mathcal{H} = \{(a, a'); a \in A''_1, a' \in A''_2, a' = \mathcal{I}(a)\}$ is called an *homeomorphic alignment* of G_1 with G_2 (see Figure 9.6). The graph G''_1 is called the *left graph* of $\mathcal{H}$. The graph G''_2 is called the *right graph* of $\mathcal{H}$.

The *cost* $C_{\mathcal{H}}$ of $\mathcal{H}$ is the sum of the costs of all operations used to homeomorphically align G_1 and G_2: the deletion of arcs in G_1 and G_2, to obtain G'_1 and G'_2 respectively, and the resizement for each arc $a_1 \in A''_1$ to the weight of $\mathcal{H}(a_1)$. More formally :

$$C_{\mathcal{H}} = \sum_{(a,a') \in \mathcal{H}} \gamma(\omega''_1(a), \omega''_2(a')) + \sum_{a_d \in A_1 \setminus A'_1} \gamma(\omega_1(a_d), 0) + \sum_{a'_d \in A_2 \setminus A'_2} \gamma(0, \omega_2(a'_d)) \ .$$

This minimal cost of all homeomorphic alignments between G_1 and G_2, called the *homeomorphic alignment distance*, is denoted by $\eta(G_1, G_2)$.

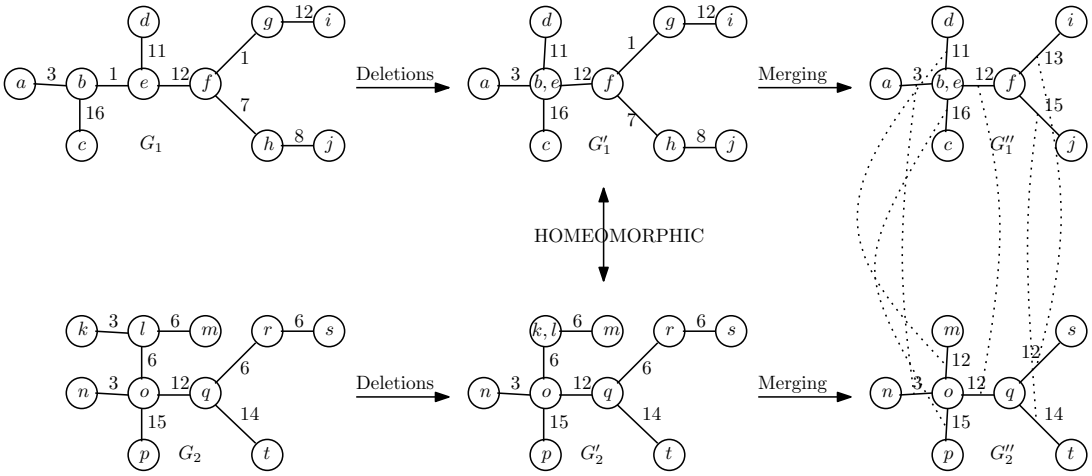


FIGURE 9.6: Example of homeomorphic alignment.

9.3 Algorithm for rooted trees

In order to compute homeomorphic alignment distance with good complexity, we will use a bottom-up approach. This approach, widely used in the literature on edit-based distances, is based on a reformulation of the distance between trees in terms of both the distance between

their subtrees and the matching of their roots. Then, using this reformulation, the computation starts by the distances between the simplest subtrees (i.e the leafs), and goes up to the root. The distances between subtrees are kept in memory, avoiding redundancy of computation. However, there are two main differences to take into account in the case of homeomorphic alignment distance:

1. Information is on arcs, rather than on vertices. It involves to take into consideration several arc weights in the reformulation (one for each arc between the root and its children), instead of the unique value linked to the root, as in the literature.
2. An arc can be matched with a set of arcs merged together (due to merging kernel application). It involves to take into account not only the subtrees of the tree, but also those which can be generated by some merging.

Our approach to solve these two problems is to use a special kind of tree, the root of which has only one child. Then, we make a reformulation of the homeomorphic alignment distance between this kind of trees, in function of subtrees with the same property. By this way, there is only one new arc to take into consideration at each step, the one between the root and its child. Moreover, we consider that this arc can be the result of a merging. In combination with the recursive nature of the bottom-up approach, it will lead to consider subtrees with all possible mergings. Finally, we also need a reformulation of the homeomorphic alignment distance between "classic" trees in function of the homeomorphic alignment distances between such subtrees.

9.3.1 Definitions and notations

First, we need to define more formally the particular kind of tree described above. Let $T = (V, A, \omega)$ be a weighted tree rooted in r_T. We denote by $T(v), v \in V$, the subtree of T rooted in v (see Figure 9.7). Let v_a be an ancestor of v, we denote by $T(v, v_a)$ the subgraph of T obtained from $T(v_a)$ by removing all complete subtrees which do not contain vertices of $T(v)$, and by merging on each vertex $n \in anc(v) \setminus \{v_a, v\}$. We say that $T(v, v_a)$ is the subtree of T rooted in v_a *pruned in* v, and we call this kind of trees a *pruned tree* (see Figure 9.7 for an example). We denote by $\mathcal{F}(T, v)$ the *forest*, the connected components of which are the trees $T(p, v)$, for all $p \in \mathcal{C}(v)$ (see Figure 9.7 for an example). By abuse of notation we also denote by $\mathcal{F}(T, v)$ the set of all connected components of this forest.

9.3.2 Reformulations

In order to use a bottom-up approach, we have to express the homeomorphic alignment distance:

- between trees in function of the distances between their forests (Proposition 18).

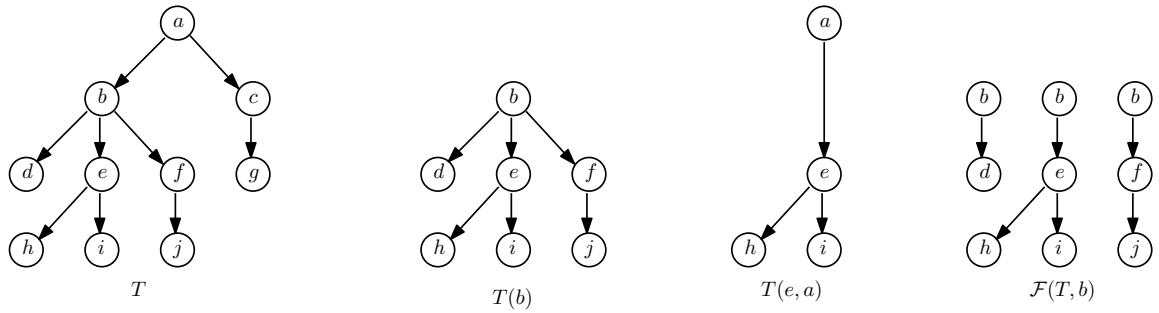


FIGURE 9.7: From the left to the right: a tree T, a subtree of T rooted in b, a subtree of T rooted in a and pruned in e, and a forest of T with origin b.

- between pruned trees in function of the distances between their pruned subtrees (Proposition 20).
- between forests in function of the distances between their subforests (Proposition 21).

In addition, we have to reformulate these distances in the special cases where at least one of the trees is empty (Proposition 19).

In the sequel we consider two weighted trees $P = (V_P, A_P, \omega_P)$ and $D = (V_D, A_D, \omega_D)$, rooted respectively in r_P and r_D.

Proposition 18. *We have :*

$$\eta(P, D) = \eta(\mathcal{F}(P, r_P), \mathcal{F}(D, r_D)).$$

Proof. Since the root of a tree cannot be eliminated by any operation (merging, deletion) involved in the definition of η, it may be seen that any homeomorphic alignment $\mathcal{H}$ of P with D has a corresponding alignment $\mathcal{H}'$ of $\mathcal{F}(P, r_P)$ with $\mathcal{F}(D, r_D)$ of same cost, and the converse also holds. □

Proposition 19. *Let $i \in V_P \setminus \{r_P\}, j \in V_D \setminus \{r_D\}, i_a \in anc(i), j_a \in anc(j)$,*

$$\begin{aligned}
&\eta(\emptyset, \emptyset) = 0 \\
&\eta(P(i, i_a), \emptyset) = \eta(\mathcal{F}(P, i), \emptyset) + \gamma(\omega_P(i_a, i), 0) \\
&\eta(\mathcal{F}(P, i), \emptyset) = \sum_{i' \in \mathcal{C}(i)} \eta(P(i', i), \emptyset) \\
&\eta(\emptyset, D(j, j_a)) = \eta(\emptyset, \mathcal{F}(D, j)) + \gamma(0, \omega_D(j_a, j)) \\
&\eta(\emptyset, \mathcal{F}(D, j)) = \sum_{j' \in \mathcal{C}(j)} \eta(\emptyset, D(j', j)).
\end{aligned}$$

Proof. Straightforward. □

Proposition 20. *Let $i \in V_P \setminus \{p\}, j \in V_D \setminus \{d\}, i_a \in anc(i), j_a \in anc(j)$.*

$$\eta(P(i,i_a),D(j,j_a)) = \\ min \begin{cases} \eta(P(i,i_a),\emptyset) + \eta(\emptyset,D(j,j_a)), \\ \gamma(\omega_P(i_a,i),\omega_D(j_a,j)) + \eta(\mathcal{F}(P,i),\mathcal{F}(D,j)), \\ min_{j_c \in \mathcal{C}(j)}\{\eta(P(i,i_a),D(j_c,j_a)) + \eta(\emptyset,\mathcal{F}(D,j) \setminus D(j_c,j))\}, \\ min_{i_c \in \mathcal{C}(i)}\{\eta(P(i_c,i_a),D(j,j_a)) + \eta(\mathcal{F}(P,i) \setminus P(i_c,i),\emptyset)\}. \end{cases}$$

Proof. Let $\mathcal{H}$ be an homeomorphic alignment of $P(i,i_a)$ with $D(j,j_a)$, and let $\mathcal{H}_\mathcal{L} = (V_L, A_L, \omega_L)$ and $\mathcal{H}_\mathcal{R} = (V_R, A_R, \omega_R)$ the left and right graphs of $\mathcal{H}$, respectively.

There are seven possible cases:

1. $\mathcal{H}$ is an empty set (it is cheaper to remove both $P(i,i_a)$ and $D(j,j_a)$ than to align them).
2. $\{(i_a,i),(j_a,j)\} \in \mathcal{H}$.
3. $\exists f \in A_R$, f being obtained by merging (j_a,j) with other arcs. In this case, there is one and only one child j_c of j, such (j,j_c) is merged with (j_a,j) in f, and then all $D(j'_c,j), j'_c \in \mathcal{C}(j) \setminus \{j_c\}$ are deleted.
4. $\exists f \in A_L$, f being obtained by merging (i_a,i) with other arcs. In this case, there is one and only one child i_c of i, such (i,i_c) is merged with (i_a,i) in f, and then all $P(i'_c,i), i'_c \in \mathcal{C}(i) \setminus \{i_c\}$ are deleted.

Cases 1,2,3,4 justify, respectively, the lines 1,2,3,4 of the expression of $\eta(P(i,i_a),D(j,j_a))$ in the proposition.

The three last cases cannot lead to a better homeomorphic alignment:

5. The deletion of (i_a,i) and (j_a,j) cannot be preferred to the resizement (possible case 2), because $\gamma(\omega_P(i_a,i),0) + \gamma(0,\omega_D(j_a,j)) \geq \gamma(\omega_P(i_a,i),\omega_D(j_a,j))$.
6. If (i_a,i) was deleted, and not (j_a,j), then only one $P(i_c,i), i_c \in \mathcal{C}(i)$ is aligned with $D(j,j_a)$, the other being removed. It is less expensive to merge (i_a,i) with (i,i_c) (possible case 3), because $\gamma(\omega_P(i_a,i),0) + \gamma(\omega_P(i,i_c),\omega_D(j_a,j)) \geq \gamma(\omega_P(i_a,i_c),\omega_D(j_a,j))$ (Prop. 14).
7. The deletion of (j_a,j) is more expensive than the merging of (j_a,j) (possible case 4), for the same reasons as above.

□

Proposition 21. $\forall A \subseteq \mathcal{F}(P,i), B \subseteq \mathcal{F}(D,j)$,

$$\eta(A,B) = \\ min \begin{cases} min_{D(j',j)\in B} & \{\eta(A, B \setminus \{D(j',j)\}) + \eta(\emptyset, D(j',j))\}, \\ min_{P(i',i)\in A} & \{\eta(A \setminus \{P(i',i)\}, B) + \eta(P(i',i), \emptyset)\}, \\ min_{P(i',i)\in A, D(j',j)\in B} & \{\eta(A \setminus \{P(i',i)\}, B \setminus \{D(j',j)\}) \\ & +\eta(P(i',i), D(j',j))\}, \\ min_{P(i',i)\in A, B'\subseteq B} & \{\eta(A \setminus \{P(i',i)\}, B \setminus B'), \\ & +\eta(\mathcal{F}(P,i'), B') + \gamma(\omega_P(i,i'), 0)\}, \\ min_{A'\subseteq A, D(j',j)\in B} & \{\eta(A \setminus A', B \setminus \{D(j',j)\}) + \\ & \eta(A', \mathcal{F}(D,j')j) + \gamma(0, \omega_D(j,j'))\}. \end{cases}$$

Proof. Let $\mathcal{H}$ be an homeomorphic alignment of $A \subseteq \mathcal{F}(P,i)$ with $B \subseteq \mathcal{F}(D,j)$, and let $P(i',i) \in A$ and $D(j',j) \in B$. There are five possible cases:

1. $D(j',j)$ is not aligned with element of A,
2. $P(i',i)$ is not aligned with element of B,
3. $P(i',i)$ is aligned with $D(j',j)$,
4. disjoint subparts of $P(i',i)$ are aligned with elements of B,
5. disjoint subparts of $D(j',j)$ are aligned with elements of A.

Cases 1, 2, 3, 4, 5 justify, respectively, the lines 1, 2, 3, 4, 5 of the expression of $\eta(A,B)$ in the proposition. □

9.3.3 Algorithm

Using the reformulations defined above, we can now design the Algorithm 14 following a bottom-up approach.

Algorithm 14: Homeomorphic Alignment Distance for Rooted Trees

Data: pattern rooted tree P, data rooted tree D

Result: $\eta(P, D) = \eta(\mathcal{F}(P, r_P), \mathcal{F}(D, r_D))$; `// Prop.18`

```
1  begin
2      foreach p ∈ V_P, in suffix order do
3          foreach A ⊆ F(P, p) do
4              Compute η(A, ∅); // Prop.19
5          foreach p_a ∈ anc(p) \ {p} do
6              Compute η(P(p, p_a), ∅); // Prop.19
7      foreach d ∈ V_D, in suffix order do
8          foreach B ⊆ F(D, d) do
9              Compute η(∅, B); // Prop.19
10         foreach d_a ∈ anc(d) \ {d} do
11             Compute η(∅, D(d, d_a)); // Prop.19
12     foreach p ∈ V_P, in suffix order do
13         foreach d ∈ V_D, in suffix order do
14             foreach A ⊆ F(P, p) do
15                 foreach B ⊆ F(D, d) do
16                     Compute η(A, B); // Prop.21
17             foreach p_a ∈ anc(p) \ {p} do
18                 foreach d_a ∈ anc(d) \ {d} do
19                     Compute η(P(p, p_a), D(d, d_a)); // Prop.20
20 end
```

For a tree $T = (V, A, \omega)$, we define a *suffix order* for the vertices in V, such that for all $v \in V$, all the children of v are before v.

9.3.4 Complexity

We denote by N the maximum degree of a vertex in the pattern tree or in the data tree, H the maximal height of both trees and S the maximal size (number of vertices) of both trees. We recall that a set of size n has 2^n subsets.

The time complexity of computing each reformulation used in the algorithm is:

- $\eta(P(i, i_a), \emptyset)$ and $\eta(\emptyset, D(j, j_a)) \rightarrow O(1)$.
- $\eta(A, \emptyset)$ and $\eta(\emptyset, B) \rightarrow O(N)$.
- $\eta(P(i, i_a), D(j, j_a)) \rightarrow O(N)$.
- $\eta(A, B) \rightarrow O(N2^N)$.

Combining these complexities with the different loops of the algorithm, we obtain that the total computation is in $O(S^2(N2^{3N} + H^2N))$ time complexity.

If the maximal degree is bounded, the total computation is in $O(S^2H^2)$ time complexity.

9.4 Algorithm for unrooted trees

First, let us give an expression of the homeomorphic alignment distance between unrooted trees, in function of the distances between all their possible rooted versions.

Let $G = (V, E, \omega)$ be a weighted tree, let $r \in V$, we denote by G^r, the directed weighted tree rooted in r, such that G is the undirected graph associated to G^r (see Figure 9.8).

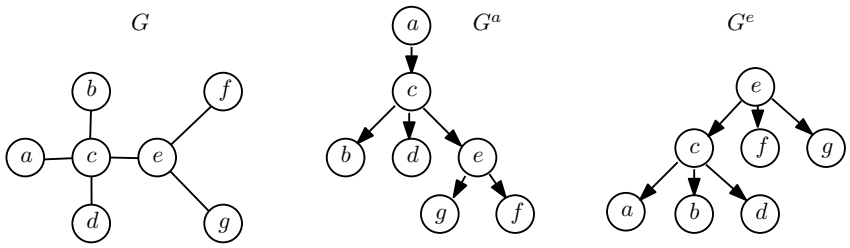


FIGURE 9.8: A tree G and the rooted trees G^a and G^e.

Proposition 22. *Let $P = (V_P, E_P, \omega_P)$ and $D = (V_D, E_D, \omega_D)$ be two weighted trees. We have:*

$$\eta(P, D) = min_{i \in V_P, j \in V_D}\{\eta(P^i, D^j)\}.$$

Proof. Let $G = (V, E, \omega)$ be a graph, and $r \in V$ a vertex of G. Notice that:

- a merging occurring on a vertex $v \in V \setminus \{r\}$ in G can occur in G^r,
- deletion, insertion, resizement, and division occurring in G can occur in G^r.

On the other hand, for each optimal homeomorphic alignment $\mathcal{H}$ of P in D, it is easy to see that there exists $p \in V_P$ and $d \in V_D$, such that p and d are not affected by a merging. For example, if $D' = (V'_D, E'_D, \omega'_D)$ is a subgraph such as $\eta(P, D) = \eta(P, D')$, p and d can be chosen as 1-degree vertices of V_P and V'_D, respectively. As a result, $\eta(P, D) = \eta(P^p, D^d)$. Since the homeomorphic alignment is more constrained in the case of rooted trees than in the case of unrooted trees, we can assure than $\eta(P, D) \leq \eta(P^a, D^b), a \in V_P, b \in V_D$. To sum up, knowing that $\eta(P, D) \leq \eta(P^a, D^b), a \in V_P, b \in V_D$, and that there exists $p \in V_P$ and $d \in V_D$, such that $\eta(P, D) = \eta(P^p, D^d)$, we conclude that $\eta(P, D) = min_{i \in V_P, j \in V_D} \eta(P^i, D^j)$. □

9.4.1 Naive algorithm

Let $P = (V_P, E_P, \omega_P)$ and $D = (V_D, E_D, \omega_D)$ be two weighted trees. From Proposition 22, we propose a first, naive algorithm to compute $\eta(P, D)$, consisting of computing the homeomorphic alignment distance for all couples of weighted rooted trees we can obtain from P and D, and keeping the minimum reached.

9.4.1.1 Complexity

Let $P = (V_P, E_P, \omega_P)$ and $D = (V_D, E_D, \omega_D)$ be two weighted trees. As the number of rooted trees we can obtain from an undirected tree is equal to the number of vertices of this graph, the number of couples of weighted rooted trees we can obtain from P and D is equal to $|V_P| * |V_D|$.

The total computation is then in $O(S^4(N2^{3N} + \hat{H}^2 N))$ time complexity, with $\hat{H}$ representing the maximal height of a rooted tree obtained from P or D (N and S are already defined in Section 9.3.4).

If the maximal degree is bounded, the total computation is in $O(S^4 \hat{H}^2)$ time complexity.

9.4.2 Optimized algorithm

It is easy to see that the above algorithm computes the alignment of subparts of P and D more than one time. Using dynamic programming and an adapted order of navigation in the tree, we can avoid useless computation.

Let $P = (V_P, E_P, \omega_P)$ and $D = (V_D, E_D, \omega_D)$ be two undirected weighted trees.

We denote by $\mathcal{F}(P, a, b), a, b \in V_P$ the set of rooted trees P^r, $r \in V_P$, such that b is an ancestor of a in P^r.

We denote by $anc(P, a, b)$, $a, b \in V_P$ the set of vertices $x \in V_P$ such that x is an ancestor of a in at least one rooted tree in $\mathcal{F}(P, a, b)$.

We denote by $\mathcal{C}(P, a, b)$, $a, b \in V_P$ the set of of vertices $x \in V_P$ such that x is a child of a in at least one rooted tree in $\mathcal{F}(P, a, b)$.

For computing $\eta(P^p(i, i_a), D^d(j, j_a))$ and $\eta(\mathcal{F}(P^p, i), \mathcal{F}(D^d, j))$, we need to know $\eta(P^p(i_c, i), D^d(j_c, j))$ for all $i_c \in \mathcal{C}(P, i, p)$, $j_c \in \mathcal{C}(D, j, d)$.

We can start by computing $\eta(P^p(i, i_a), D^d(j, j_a))$ and $\eta(\mathcal{F}(P^p, i), \mathcal{F}(D^d, j))$, for all i (respectively, j) being a leaf of P^p (respectively, D^d), which have no child, by definition, and continue iteratively with all vertices which have all their children already computed.

9.4.2.1 Adapted order of navigation

An adapted order of navigation for a tree $T = (V, E, \omega)$ can be obtained by the Algorithm 15.

Algorithm 15: Order of navigation computation algorithm (computeOrder)

Data: unrooted tree T

Result: list of couples of vertices L

```
1  begin
2      list of couples of vertices L ← ∅
3      FIFO queue of vertices Q ← ∅
4      foreach v ∈ V do
5          if deg(v) = 1 then
6              push v in Q
7      while Q ≠ ∅ do
8          pop v of Q
9          foreach w ∈ N(v) do
10             if (v, w) ∉ L then
11                 if ∀x ∈ N(v) \ {w}, (x, v) ∈ L then
12                     push w in Q
13                     add (v, w) at the end of L
14 end
```

9.4.2.2 Final algorithm

We can compute Homeomorphic Alignment for unrooted trees (see Algorithm 16) with improved complexity, using this order of navigation.

Algorithm 16: Homeomorphic Alignment Distance for Unrooted Trees

Data: pattern rooted tree P,
data rooted tree D
Result: $\eta(P, D)$

```
1  begin
2      list of couples of vertices L_P ← computeOrder(P)
3      list of couples of vertices L_D ← computeOrder(D)
4      foreach (p, p') ∈ L_P do
5          foreach A ⊆ F(P^{p'}, p) do
6              Compute η(A, ∅); // Prop.19
7          foreach p_a ∈ anc(P, p, p') \ {p} do
8              Compute η(P^{p_a}(p, p_a), ∅); // Prop.19
9      foreach (d, d') ∈ L_D do
10         foreach B ⊆ F(D^{d'}, d) do
11             Compute η(∅, B); // Prop.19
12         foreach d_a ∈ anc(D, d, d') \ {d} do
13             Compute η(∅, D^{d_a}(d, d_a)); // Prop.19
14     foreach (p, p') ∈ L_P do
15         foreach (d, d') ∈ L_D do
16             foreach A ⊆ F(P^{p'}, p) do
17                 foreach B ⊆ F(D^{d'}, d) do
18                     Compute η(A, B); // Prop.21
19             foreach p_a ∈ anc(P, p, p') \ {p} do
20                 foreach d_a ∈ anc(D, d, d') \ {d} do
21                     Compute η(P^{p_a}(p, p_a), D^{d_a}(d, d_a)); // Prop.20
22     Compute η(P, D); // Prop.18 and Prop.22
23 end
```

Complexity Let $T = (V, E, \omega)$ be an unrooted tree. Each vertex of degree 1 (one) will be put in the queue, then, for each edge, each of its vertices will be put twice in the queue. As $|V| = |E| + 1$, the complexity of computeOrder is in $O(|V|)$.

Let $P = (V_P, E_P, \omega_P)$ and $D = (V_D, E_D, \omega_D)$ be two weighted trees. The initialization of L_P and L_D is in $O(|V_P| + |V_D|)$ time complexity.

Observe that, for $p \in V_P$,

$$\sum_{p' \in \mathcal{N}(p)} |anc(P, p, p') \setminus \{p\}| = |V_P| - 1 \ .$$

As a result,

$$\sum_{(p,p') \in L_P} |anc(P, p, p') \setminus \{p\}| = (|V_P| - 1) \times |V_P| \ .$$

Combining the time complexities of computing the reformulations (given in the Section 9.3.4) with the loops of the algorithm, and taking into account the above observation, we obtain that the total computation time of this algorithm is in $O(S^2 N(2^{3N} + S^2))$ complexity, with N and S defined as in Section 9.3.4.

If the maximal degree is bounded, the total computation is in $O(S^4)$ time complexity.

9.5 Algorithm for rooted tree with unrooted tree

As seen in the previous section, the optimal homeomorphic alignment between two unrooted trees P and D is equal to the minimal optimal homeomorphic alignment between all possible rooted versions of P and D.

From this fact, we can define the optimal homeomorphic alignment between a tree P^r rooted in r and an unrooted tree D as the minimal optimal homeomorphic alignment between P^r and all possible rooted versions of D.

From Proposition 22 we can propose Proposition 23, which is straightforward.

Proposition 23. *Let $P^r = (V_P, A_P, \omega_P)$ a weighted tree rooted in $r \in V_P$ and $D = (V_D, E_D, \omega_D)$ an unrooted weighted tree. We have:*

$$\eta(P^r, D) = min_{j \in V_D}\{\eta(P^r, D^j)\}.$$

The only difference between matching a rooted tree P^r with an unrooted tree D and matching two unrooted trees P and D is that no merging can occur on the root r of P^r, by definition. It is interesting if we want to be sure that a vertex of P is preserved by the matching.

9.5.1 Algorithm

The algorithm to compute the homeomorphic alignment between rooted tree and unrooted tree (see Algorithm 17) can be obtained by crossing Algorithm 14 and Algorithm 16.

Algorithm 17: Homeomorphic Alignment Distance between Rooted and Unrooted Trees

Data: pattern rooted tree P,
data unrooted tree D
Result: $\eta(P, D)$

```
1  begin
2      list of couples of vertices L_D ← computeOrder(D)
3      foreach p ∈ V_P, in suffix order do
4          foreach A ⊆ F(P, p) do
5              Compute η(A, ∅); // Prop.19
6          foreach p_a ∈ anc(p) \ {p} do
7              Compute η(P(p, p_a), ∅); // Prop.19
8      foreach (d, d') ∈ L_D do
9          foreach B ⊆ F(D^{d'}, d) do
10             Compute η(∅, B); // Prop.19
11         foreach d_a ∈ anc(D, d, d') \ {d} do
12             Compute η(∅, D^{d_a}(d, d_a)); // Prop.19
13     foreach p ∈ V_P, in suffix order do
14         foreach (d, d') ∈ L_D do
15             foreach A ⊆ F(P, p) do
16                 foreach B ⊆ F(D^{d'}, d) do
17                     Compute η(A, B); // Prop.21
18             foreach p_a ∈ anc(p) \ {p} do
19                 foreach d_a ∈ anc(D, d, d') \ {d} do
20                     Compute η(P(p, p_a), D^{d_a}(d, d_a)); // Prop.20
21     Compute η(P, D); // Prop.18 and Prop.23
22 end
```

9.5.1.1 Complexity

We denote by N the maximal degree of both trees, S their maximum size, and H the height of the rooted tree. The total computation time of this algorithm is in $O(S^2N(2^{3N} + SH))$

complexity.

If the maximal degree is bounded, the total computation is in $O(HS^3)$ time complexity.

9.6 Usage of Cut Operation

For the purpose of removing spurious branches without any cost, we propose to integrate the cut operation in our alignment.

In [180], Wang et al. propose a new operation allowing to consider only a part of a tree. Let $G = (V, A, \omega)$ be a weighted tree. *Cutting* G at an arc $a \in A$, means removing a, thus dividing G into two subtrees G_1 and G_2. The *cut operation* consists of cutting G at an arc $a \in A$, then considering only one of the two subtrees. Let K a subset of A. We use $Cut(G, K, v)$ to denote the subtree of G containing v and resulting from cutting G at all arcs in K. In the case of a rooted tree, we consider that the root r_G of G cannot be removed by the cut operation, and then we use the notation $Cut(G, K) = Cut(G, K, r_G)$. In the case of a rooted forest, we consider that the root of each rooted tree composing the rooted forest cannot be removed by the cut operation, and then we use the same notation than above: $Cut(G, K)$.

See Figure 9.9 for some examples of cut operation.

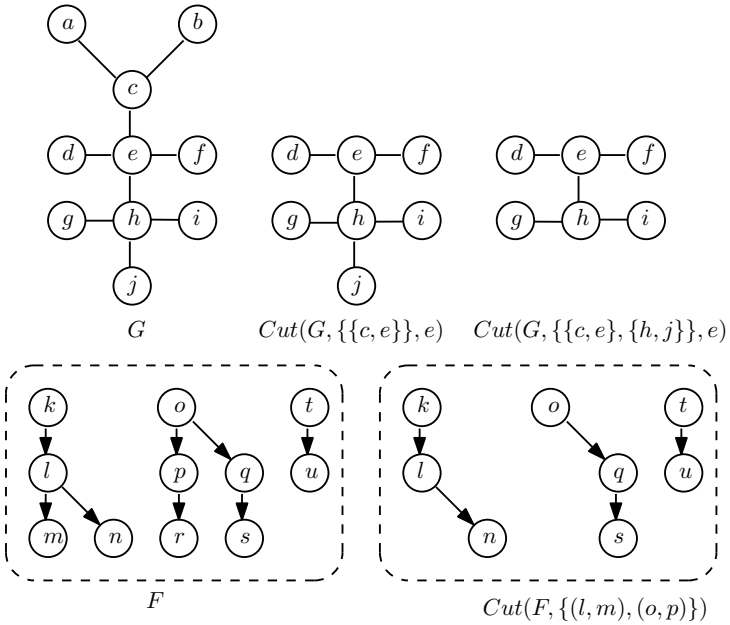


FIGURE 9.9: Examples of cut operations. Top row: on an unrooted tree. Bottom row: on a rooted forest.

Our main problem can be stated as follows: Given a weighted tree $P = (V_P, A_P, \omega_P)$ (the pattern tree) and a weighted tree $G_D = (V_D, A_D, \omega_D)$ (the data tree), find

$$\eta_{cut}(P, D) = min_{K \subseteq A_D, v \in V_D}\{\eta(P, Cut(D, K, v)\}$$

and the associated homeomorphic alignment. In the case of rooted trees and rooted forests, $\eta_{cut}(P, D) = min_{K \subseteq A_D}\{\eta(P, Cut(D, K))\}$.

9.6.1 Integration of cut operation in our algorithm

It can be seen that the cut operation can be integrated in the homeomorphic alignment by replacing the deletion of complete subtrees by their cut.

In order to obtain the algorithms for the computation of η_{cut}, we just need to replace:

- η by η_{cut} in the reformulations and algorithms of the Sections 9.3 and 9.4
- Proposition 19 by the following:

  Proposition 24. *Let $i \in V_P \setminus \{p\}, j \in V_D \setminus \{d\}, i_a \in anc(i), j_a \in anc(j)$,*

$$\begin{aligned}
&\eta_{cut}(\emptyset, \emptyset) = 0\\
&\eta_{cut}(P(i, i_a), \emptyset) = \eta_{cut}(\mathcal{F}(P, i), \emptyset) + \gamma(\omega_P(i_a, i), 0)\\
&\eta_{cut}(\mathcal{F}(P, i_a), \emptyset) = \sum_{i' \in \mathcal{C}(i_a)} \eta_{cut}(P(i', i_a), \emptyset)\\
&\eta_{cut}(\emptyset, D(j, j_a)) = 0\\
&\eta_{cut}(\emptyset, \mathcal{F}(D, j_a)) = 0.
\end{aligned}$$

It is interesting to remark that the complexity is not modified.

9.7 Limitations

The homeomorphic alignment is an efficient tool for our purpose of motion capture: it affords to match our model tree with data tree, taking into account the possible noises occurring in our context. However, we can observe some limitations.

The merging operation have to be more restricted. For our application, merging in the model tree are useless, the problem of useless 2-degree vertices occurring only in the data tree. Furthermore, merging operations in model tree can lead to bad matchings, as shown in Figure 9.10.

Restricting the merging operation on the data tree could lead to both optimize the computation speed and increase the accuracy of results.

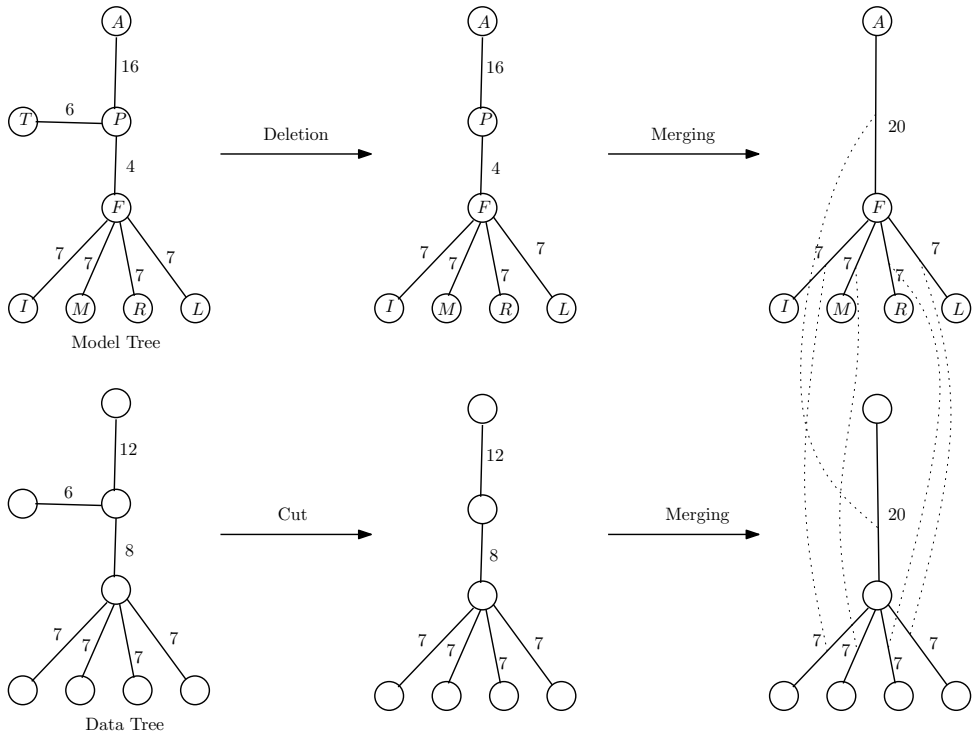


FIGURE 9.10: Example of bad homeomorphic alignment matching due to merging on model tree: the "thumb" edge is removed for a cost of 6, as the merging of "arm" edge and "palm" edge in both trees reduce the matching cost by 8.

Chapter 10

Asymmetric Homeomorphic Alignment

In the previous chapter, we proposed a first solution to our problem of matching by the way of homeomorphic alignment. We have closed this chapter by showing the limitations of this method, due to non restrictions on merging operation.

The alignment and the homeomorphic alignment allow to find the best way to modify two trees in order to reach a specific relation between them (isomorphism in the first case, homeomorphism in the second). Isomorphism is a sub case of homeomorphism (i.e. two trees isomorphic are homeomorphic) and by extension, an alignment between two trees is an homeomorphic alignment as well.

In this chapter, we first define an asymmetric relation being sub case of homeomorphism and for which isomorphism is a sub case. Then, we propose new alignment allowing to find the best way to modify two trees in order to reach this relations. Finally, we design algorithms in order to compute it efficiently.

10.1 Asymmetric Homeomorphism

10.1.1 Definition

Let $G = (V, E, \omega)$ and $G' = (V', E', \omega')$ be two weighted graphs, G is *homeomorphically greater than or equal* to G' if and only if there exists an isomorphism between G' and a graph obtained by mergings on G. We denote this relation by $G \overset{\mathcal{H}}{\geq} G'$.

10.1.2 Properties

In this section we denote by $\overset{\mathcal{H}}{=}$ the homeomorphism relation, and by $\overset{\mathcal{I}}{=}$ the isomorphism relation.

10.1.2.1 Usual definitions on binary relations

A *binary relation* $\mathcal{R}$ on a set S is a collection of ordered pairs of elements of S. Let a and b two elements of S, the statement $(a, b) \in \mathcal{R}$ is denoted by $a\mathcal{R}b$.

The binary relation $\mathcal{R}$ is an *equivalence relation* if and only if for all a, b and c in S:

- $a\ \mathcal{R}\ a$ (reflexivity)
- if $a\ \mathcal{R}\ b$ then $b\ \mathcal{R}\ a$ (symmetry)
- if $a\ \mathcal{R}\ b$ and $b\ \mathcal{R}\ c$ then $a\ \mathcal{R}\ b$ (transitivity)

The *equivalence class* of an element s of S by $\mathcal{R}$ is defined by $\mathcal{R}(s) = \{x \in S | x\mathcal{R}s\}$.

The binary relation $\mathcal{R}$ is an *order relation* if and only if for all a, b and c in S:

- $a\ \mathcal{R}\ a$ (reflexivity)
- if $a\ \mathcal{R}\ b$ and $b\ \mathcal{R}\ a$ then $a = b$ (antisymmetry)
- if $a\ \mathcal{R}\ b$ and $b\ \mathcal{R}\ c$ then $a\ \mathcal{R}\ b$ (transitivity)

An order relation $\mathcal{R}$ is *total* if for all a and b in S, $a\ \mathcal{R}\ b$ or $b\ \mathcal{R}\ a$. In the other case, $\mathcal{R}$ is a *partial order relation*.

10.1.2.2 Properties of asymmetric homeomorphism

Contrary to isomorphism and homeomorphism which are equivalence relations on the set of graphs $\mathbb{G}$, the asymmetric homeomorphism is an partial order relation on $\mathbb{G}$.

More precisely, as the asymmetric homeomorphism is a sub case of homeomorphism, there is no asymmetric homeomorphism relations between elements of different equivalence class by homeomorphism on $\mathbb{G}$.

If we consider a graph G, the equivalence class $\overset{\mathcal{H}}{=}(G)$ is partially ordered by $\overset{\mathcal{H}}{\geq}$ (see Figure 10.1 for an example) and have a *least element* equal to $\mathcal{MK}(G)$: for all $G' \in \overset{\mathcal{H}}{=}(G)$, $G' \overset{\mathcal{H}}{\geq} \mathcal{MK}(G)$.

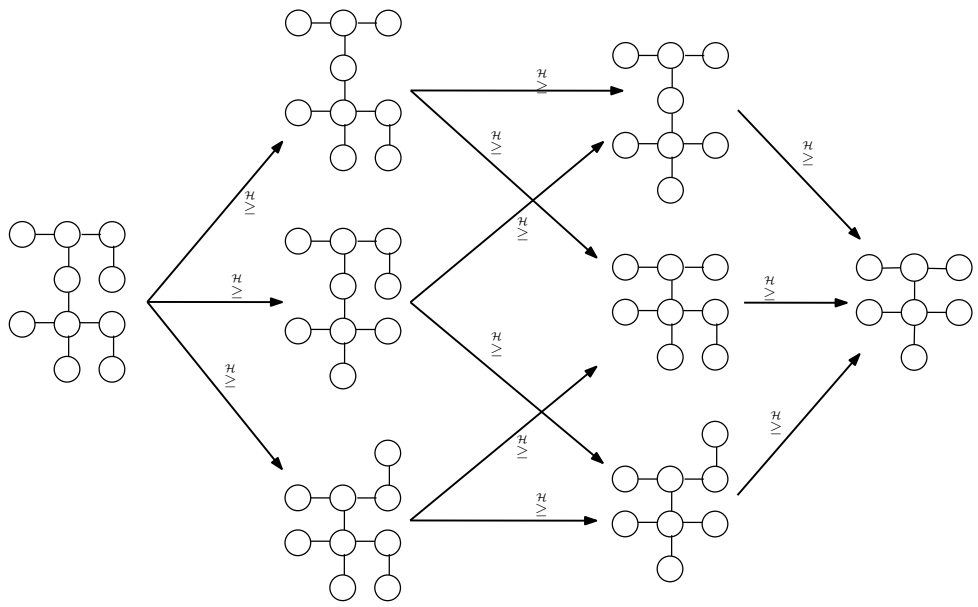


FIGURE 10.1: Example of asymmetric homeomorphic relations on a set of homeomorphic trees.

10.2 Asymmetric Homeomorphic Alignment

Let $G_1 = (V_1, A_1, \omega_1)$ and $G_2 = (V_2, A_2, \omega_2)$ be two weighted graphs. Let $G'_1 = (V'_1, A'_1, \omega'_1)$ and $G'_2 = (V'_2, A'_2, \omega'_2)$ be weighted graphs obtained by deleting arcs in G_1 and G_2, such that $G'_2 \overset{\mathcal{H}}{\geq} G'_1$. Let $G''_2 = (V''_2, A''_2, \omega''_2)$ be a graph obtained from G'_2 by a sequence of merging, such there exists an isomorphism $\mathcal{I}$ between G'_1 and G''_2.

The set of all couples of arcs $\vec{\mathcal{H}} = \{(a, a'); a \in A'_1, a' \in A''_2, a' = \mathcal{I}(a)\}$ is called an *asymmetric homeomorphic alignment* of G_1 with G_2.

The *cost* $C_{\vec{\mathcal{H}}}$ of $\vec{\mathcal{H}}$ is the sum of the costs of all operations used to asymmetric homeomorphically align G_1 and G_2: the deletion of arcs in G_1 and G_2, to obtain G'_1 and G'_2 respectively, and the resizement for each arc $a_1 \in A'_1$ to the weight of $\vec{\mathcal{H}}(a_1)$. More formally :

$$C_{\vec{\mathcal{H}}} = \sum_{(a,a') \in \vec{\mathcal{H}}} \gamma(\omega'_1(a), \omega''_2(a')) + \sum_{a_d \in A_1 \setminus A'_1} \gamma(\omega_1(a_d), 0) + \sum_{a'_d \in A_2 \setminus A'_2} \gamma(0, \omega_2(a'_d)) \ .$$

This minimal cost of all homeomorphic alignments between G_1 and G_2, called the *asymmetric homeomorphic alignment distance*, is denoted by $\vec{\eta}(G_1, G_2)$.

10.3 Algorithm for Rooted Trees

The algorithms for compute the asymmetric homeomorphic alignment distance is designed in a similar way that those we described for compute the homeomorphic alignment distance: we use

again a bottom-up approach, but here we consider only mergings on data tree. Due to this fact, we consider pruned trees only on the data tree side.

10.3.1 Reformulations

In the sequel we consider two weighted trees $P = (V_P, A_P, \omega_P)$ and $D = (V_D, A_D, \omega_D)$, rooted respectively in r_P and r_D.

As merging is not used in the reformulations 25, 26 and 28, the proofs are the same as for Propositions 18, 19 and 21, respectively.

Proposition 25. *We have :*

$$\vec{\eta}(P, D) = \vec{\eta}(\mathcal{F}(P, r_P), \mathcal{F}(D, r_D)).$$

Proposition 26. *Let $i \in V_P \setminus \{r_P\}, j \in V_D \setminus \{r_D\}, i_a = par(i), j_a \in anc(j)$,*

$$\begin{aligned}
&\vec{\eta}(\emptyset, \emptyset) = 0\\
&\vec{\eta}(P(i, i_a), \emptyset) = \vec{\eta}(\mathcal{F}(P, i), \emptyset) + \gamma(\omega_P(i_a, i), 0)\\
&\vec{\eta}(\mathcal{F}(P, i_a), \emptyset) = \sum_{i' \in \mathcal{C}(i_a)} \vec{\eta}(P(i', i_a), \emptyset)\\
&\vec{\eta}(\emptyset, D(j, j_a)) = \vec{\eta}(\emptyset, \mathcal{F}(D, j)) + \gamma(0, \omega_D(j_a, j))\\
&\vec{\eta}(\emptyset, \mathcal{F}(D, j_a)) = \sum_{j' \in \mathcal{C}(j_a)} \vec{\eta}(\emptyset, D(j', j_a)).
\end{aligned}$$

Proposition 27. *Let $i \in V_P \setminus \{r_P\}, j \in V_D \setminus \{r_D\}, i_a = par(i), j_a \in anc(j)$.*

$$\vec{\eta}(P(i, i_a), D(j, j_a)) = \\ min \begin{cases} \vec{\eta}(P(i, i_a), \emptyset) + \vec{\eta}(\emptyset, D(j, j_a)), \\ \vec{\eta}(\gamma(\omega_P(i_a, i), \omega_D(j_a, j)) + \vec{\eta}(\mathcal{F}(P, i), \mathcal{F}(D, j)), \\ min_{j_c \in \mathcal{C}(j)}\{\vec{\eta}(P(i, i_a), D(j_c, j_a)) + \vec{\eta}(\emptyset, \mathcal{F}(D, j) \setminus D(j_c, j))\}, \\ \vec{\eta}(\gamma(\omega_P(i_a, i), 0) + min_{i_c \in \mathcal{C}(i)}\{\vec{\eta}(P(i_c, i), D(j, j_a)) + \vec{\eta}(\mathcal{F}(P, i) \setminus P(i_c, i), \emptyset)\}. \end{cases}$$

Proof. Let $\vec{\mathcal{H}}$ be an asymmetric homeomorphic alignment of $P(i, i_a)$ with $D(j, j_a)$, and let $\vec{\mathcal{H}}_{\mathcal{L}} = (V_L, A_L, \omega_L)$ and $\vec{\mathcal{H}}_{\mathcal{R}} = (V_R, A_R, \omega_R)$ the left and right graphs of $\vec{\mathcal{H}}$, respectively.

There are six possible cases:

1. $\vec{\mathcal{H}}$ is an empty set (it is cheaper to remove both $P(i, i_a)$ and $D(j, j_a)$ than to align them).
2. $\{(i_a, i), (j_a, j)\} \in \vec{\mathcal{H}}$.

3. $\exists f \in A_R$, f being obtained by merging (j_a, j) with other arcs. In this case, there is one and only one child j_c of j, such (j, j_c) is merged with (j_a, j) in f, and then all $D(j'_c, j), j'_c \in \mathcal{C}(j) \setminus \{j_c\}$ are deleted.

4. (i_a, i) is deleted, then only one $P(i_c, i), i_c \in \mathcal{C}(i)$ is aligned with $D(j, j_a)$, the other being removed.

Cases 1,2,3,4 justify, respectively, the lines 1,2,3,4 of the expression of $\vec{\eta}(P(i, i_a), D(j, j_a))$ in the proposition.

The two last cases cannot lead to a better homeomorphic alignment:

5. The deletion of (i_a, i) and (j_a, j) cannot be preferred to the resizement (possible case 2), because $\gamma(\omega_P(i_a, i), 0) + \gamma(0, \omega_D(j_a, j)) \geq \gamma(\omega_P(i_a, i), \omega_D(j_a, j))$.

6. If (j_a, j) was deleted, and not (i_a, i), then only one $D(j_c, j), j_c \in \mathcal{C}(j)$ is aligned with $P(i, i_a)$, the other being removed. It is less expensive to merge (j_a, j) with (j, j_c) (possible case 3), because $\gamma(0, \omega_D(j_a, j)) + \gamma(\omega_P(i_a, i), \omega_D(j, j_c)) \geq \gamma(\omega_P(i_a, i), \omega_D(j_a, j_c))$.

□

Proposition 28. $\forall A \subseteq \mathcal{F}(P, i), B \subseteq \mathcal{F}(D, j)$,

$$\vec{\eta}(A, B) = \min \begin{cases} min_{D(j',j)\in B} & \{\vec{\eta}(A, B \setminus \{D(j', j)\}) + \vec{\eta}(\emptyset, D(j', j))\}, \\ min_{P(i',i)\in A} & \{\vec{\eta}(A \setminus \{P(i', i)\}, B) + \vec{\eta}(P(i', i), \emptyset)\}, \\ min_{P(i',i)\in A, D(j',j)\in B} & \{\vec{\eta}(A \setminus \{P(i', i)\}, B \setminus \{D(j', j)\}) \\ & +\vec{\eta}(P(i', i), D(j', j))\}, \\ min_{P(i',i)\in A, B'\subseteq B} & \{\vec{\eta}(A \setminus \{P(i', i)\}, B \setminus B'), \\ & +\vec{\eta}(\mathcal{F}(P, i'), B') + \gamma(\omega_P(i, i'), 0)\}, \\ min_{A'\subseteq A, D(j',j)\in B} & \{\vec{\eta}(A \setminus A', B \setminus \{D(j', j)\}) + \\ & \vec{\eta}(A', \mathcal{F}(D, j')) + \gamma(0, \omega_D(j, j'))\}. \end{cases}$$

10.3.2 Algorithm

Using the reformulations defined above, we can now design our algorithm following the bottom-up approach (see Algorithm 18).

Algorithm 18: Asymmetric Homeomorphic Alignment Distance for Rooted Trees

Data: pattern rooted tree P, data rooted tree D

Result: $\vec{\eta}(P, D) = \vec{\eta}(\mathcal{F}(P, r_P), \mathcal{F}(D, r_D))$; `// Prop.25`

```
1  begin
2      foreach p ∈ V_P, in suffix order do
3          foreach A ⊆ F(P, p) do
4              Compute η⃗(A, ∅); // Prop.26
5          if p ≠ r_P then
6              Compute η⃗(P(p, par(p)), ∅); // Prop.26
7      foreach d ∈ V_D, in suffix order do
8          foreach B ⊆ F(D, d) do
9              Compute η⃗(∅, B); // Prop.26
10         foreach d_a ∈ anc(d) \ {d} do
11             Compute η⃗(∅, D(d, d_a)); // Prop.26
12     foreach p ∈ V_P, in suffix order do
13         foreach d ∈ V_D, in suffix order do
14             foreach A ⊆ F(P, p) do
15                 foreach B ⊆ F(D, d) do
16                     Compute η⃗(A, B); // Prop.28
17             if p ≠ r_P then
18                 foreach d_a ∈ anc(d) \ {d} do
19                     Compute η⃗(P(p, par(p)), D(d, d_a)); // Prop.27
20 end
```

10.3.3 Complexity

We denote by N the maximum degree of a vertex in the pattern tree or in the data tree, H_P and H_D the maximal height of both trees and S the maximal size (number of vertices) of both trees. We recall that a set of size n has 2^n subsets. The computation is in $O(S^2 * (N * 2^{3*N} + H_D * N))$ time complexity. If the maximal degree is bounded, the total computation is in $O(S^2 * H_D)$ time complexity.

10.4 Algorithm for Unrooted Trees and for Rooted Tree with Unrooted Tree

Algorithms for unrooted trees and for rooted tree with unrooted tree can be obtained using the same method that for homeomorphic alignment. In case of asymmetric homeomorphic alignment, the algorithm for unrooted trees is in $O(S^2 * (N * 2^{3*N} + S_D * N))$ time complexity ($O(S^3)$ if the maximal degree is bounded).

The algorithm for rooted tree with unrooted tree is in $O(S^2 * (N * 2^{3*N} + H_D * N))$ time complexity ($O(S^2 H_D)$ if the maximal degree is bounded).

Chapter 11

Alignments Comparison

All the experiments in this section have been done on a computer with a processor Intel(R) Q8200 at 2.33 GHz and 3 Gio of ram.

11.1 Complexities Comparison

In our method, we assume that both pattern tree and data tree have a bounded maximal degree. Table 11.1 summarizes the complexities of classic Jiang et al. alignment(α), homeomorphic alignment(η), and asymmetric homeomorphic alignment($\vec{\eta}$) for trees with bounded degree.

We denote by S_P and S_D the size of the pattern tree and the size of the data tree, respectively. We denote by H_P and H_D the height of the pattern tree and the size of the data tree, respectively.

11.2 Accuracy

In this section we propose to study the robustness of the different alignments in regard of the noises occurring in our motion capture framework.

TABLE 11.1: Time complexities of the different alignments.

	α	η	$\vec{\eta}$
rooted trees	$O(S_P S_D)$	$O(S_P S_D H_P H_D)$	$O(S_P S_D H_D)$
unrooted trees	$O(S_P S_D)$	$O(S_P^2 S_D^2)$	$O(S_P S_D^2)$
rooted tree with unrooted tree	$O(S_P S_D)$	$O(S_P H_P S_D^2)$	$O(S_P S_D^2)$

11.2.1 Protocol

Each experiment consists of the following:

1. We randomly generate a pattern tree P.
2. We generate a data tree D from P in two steps:

   weight variation: we randomly alter the edge weights by a given amount of variation.

   structural noise: we randomly add new vertices by three ways, corresponding to the different types of noise: splitting an existing vertex, and linking the two parts by a 0-weighted edge (*splitted vertices*), adding a new 2-degree vertex by the split of an edge (*useless 2-degree vertices*), and adding a new 1-degree vertex, linked by an randomly weighted edge (*spurious branches* and *ghost limbs*). Figure 11.1 illustrates this process.
3. We compute the different alignments between P and D.
4. The precision is given by the percentage of good matchings, that is the percentage of pattern tree vertices that match with their equivalent in the data tree (which is known, according to the above protocol to generate the data tree).

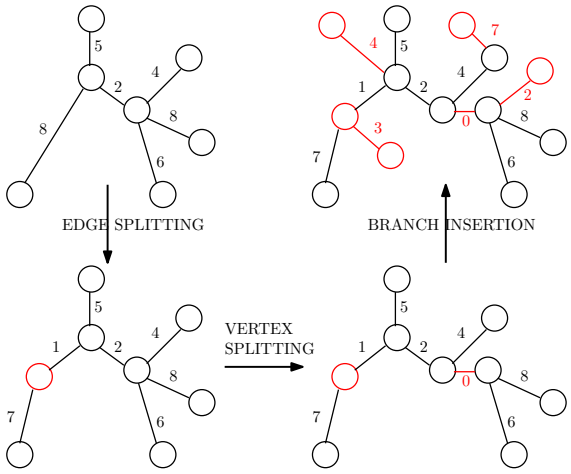


FIGURE 11.1: Illustration of structural noise generation.

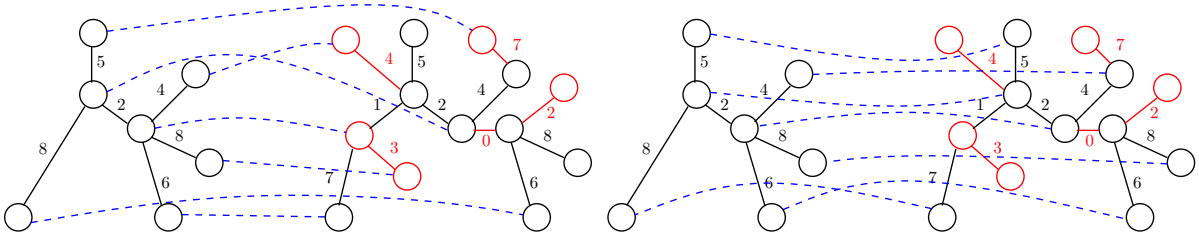


FIGURE 11.2: Illustration of precision measurement. Left: 0% of good matchings. Right: 100% of good matchings.

11.2.2 Results

The results, obtained by averaging precisions over 1000 experiments, are shown on Figures 11.3, 11.4 and 11.5.

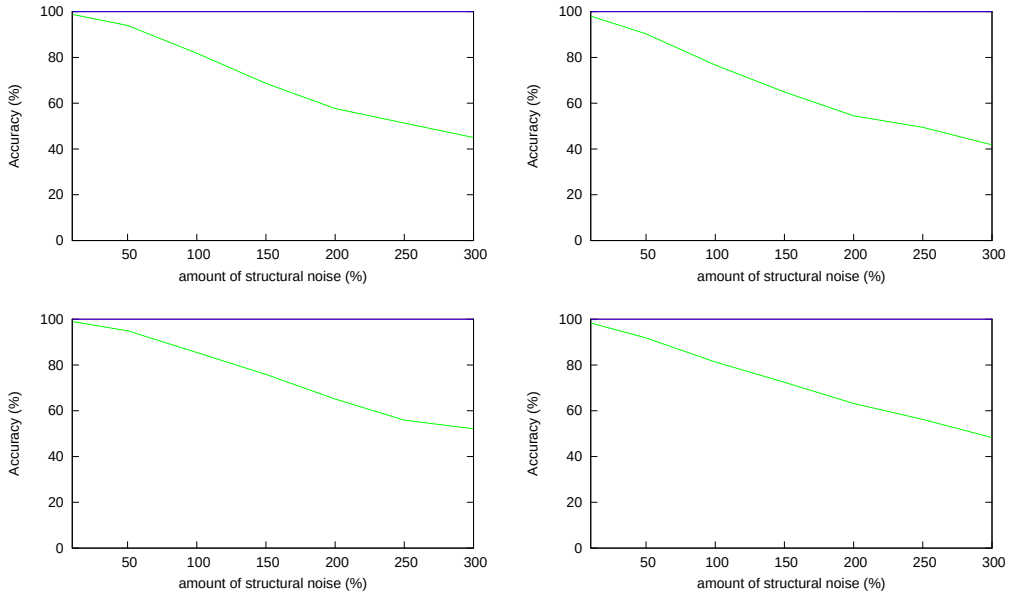


FIGURE 11.3: Precision without weight variation. Top: pattern tree with 10 vertices. Bottom: pattern tree with 20 vertices. Left: without cut operation. Right: with cut operation.

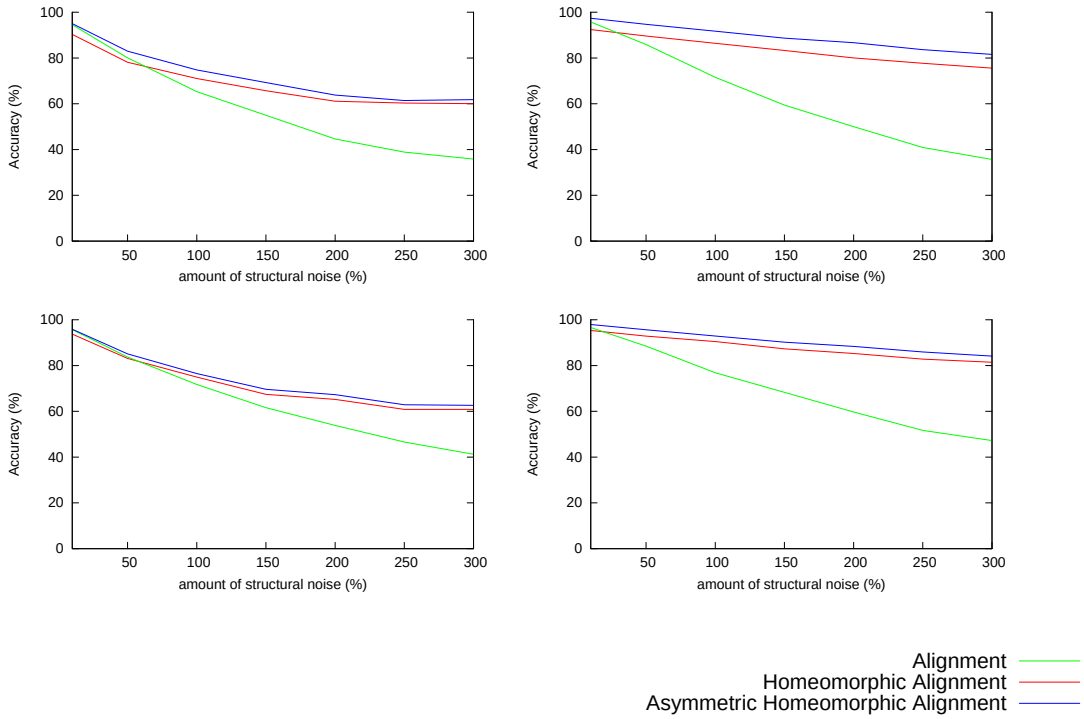


FIGURE 11.4: Precision with 10 percent of weight variation. Top: pattern tree with 10 vertices. Bottom: pattern tree with 20 vertices. Left: without cut operation. Right: with cut operation.

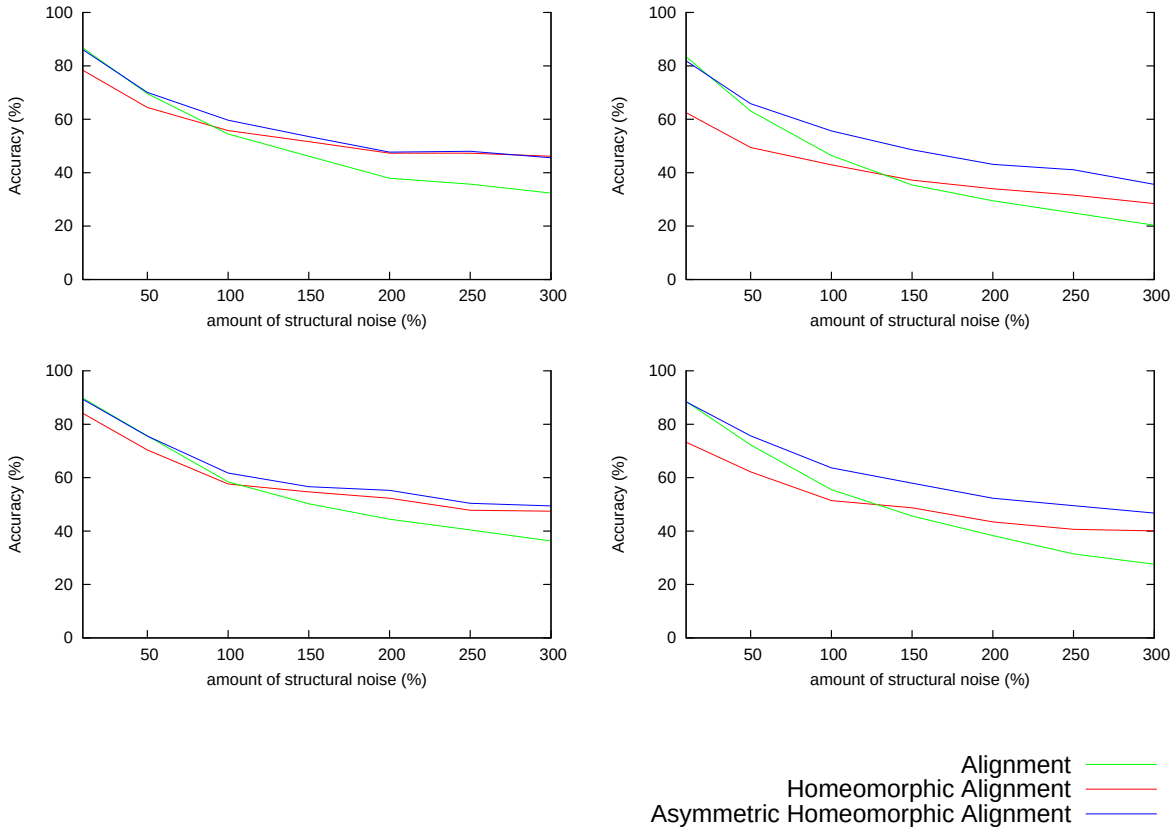


FIGURE 11.5: Precision with 50 percent of weight variation. Top: pattern tree with 10 vertices. Bottom: pattern tree with 20 vertices. Left: without cut operation. Right: with cut operation.

11.2.3 Discussion

The first observation is that the homeomorphic alignment and asymmetric homeomorphic alignment provide a perfect matching if no weight variation occurs, even for large amount of structural noise.

On the other side, for high weight variation and few structural noises, we can observe that the percentage of good matchings for homeomorphic alignment is lower than those for classic alignment. It is due to the use of mergings on both trees, as shown on Figure 11.6: as the vertex d disappear by merging in case of homeomorphic alignment, the percentage of good matchings decrease for this alignment. The asymmetric homeomorphic alignment allow the merging only on data tree, this special case is treated in the same way than for classic alignment.

Concerning the cut operation, we can observe that using it for small weight variations increase the percentage of good matching, but for large weight variations, the results are worst than without cut operation. Then the choice of using cut operation or not will depend to the a priori knowledge of amount of weight variation and structural noise in the data tree and to the presence or not of others mechanisms increasing the robustness.

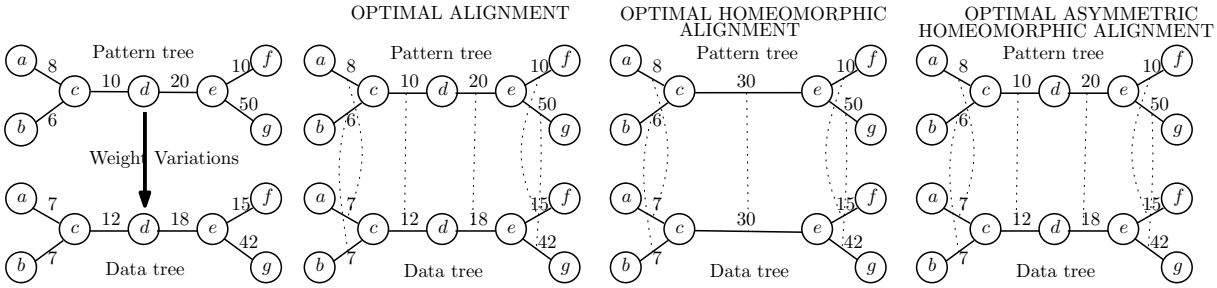


FIGURE 11.6: Example where alignment provide better result than homeomorphic alignment.

Finally, it is interesting to remark than in all the cases, the asymmetric homeomorphic alignment provide more good matchings than the other alignments, and then is the most adapted in our context of motion capture.

11.3 Computation Speed

In this section we propose to study computation speed of the different alignments in a general purpose.

11.3.1 Protocol

Each experiment consists of the following:

1. We randomly generate n trees $T_0, T_1, ..., T_{n-1}$
2. For each tested algorithm:
   - we record the starting time t_s
   - we apply the algorithm on each possible couple (T_i, T_j)
   - we record the ending time t_e
   - the average speed of the tested algorithm is equal to $\frac{(t_e - t_s)}{n^2}$

11.3.2 Results

The results, obtained for $n = 32$, are shown on Figure 11.7.

11.3.3 Discussion

The speed difference of the algorithms is well correlated with their complexities. However, we can observe a variation of speed between the classic alignment for rooted trees and the classic

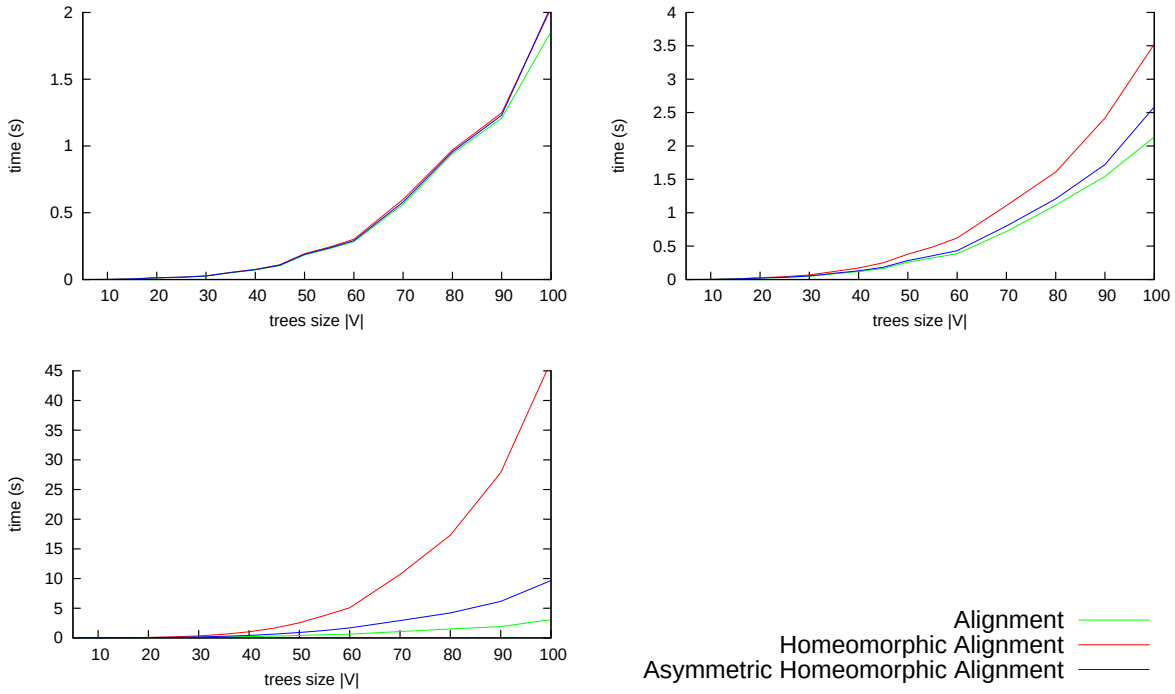


FIGURE 11.7: Computation speed of different alignments for different sizes of trees. Top left: between two rooted trees. Top right: between a rooted tree and an unrooted tree. Bottom left: between two unrooted trees.

TABLE 11.2: Time complexities of the different alignments, taking into account the bounded maximal degree.

	α	η	$\vec{\eta}$
rooted trees	$O(S_P S_D C)$	$O(S_P S_D (C + H_P H_D))$	$O(S_P S_D (C + H_D))$
unrooted trees	$O(S_P S_D C)$	$O(S_P S_D (C + S_P S_D))$	$O(S_P S_D (C + S_D))$
rooted tree with unrooted tree	$O(S_P S_D C)$	$O(S_P S_D (C + H_P S_D))$	$O(S_P S_D (C + S_D))$

alignment for unrooted trees, even if they have the same complexity. This variation is due to the maximal degree observed: in the rooted tree, the degree considered is in fact the number of children. Assuming that the maximal number of children is equal to the maximal degree decreased by one, the constant linked to the computation of forests alignments is divide per two for the alignment between rooted trees.

The constant C linked to the computation of forests alignments (due to bounded maximal degree) have to be take into consideration when we observe the computation speed. Table 11.2 summarizes the time complexities of the different alignments, taking into account the bounded maximal degree.

If $C >> H_P + H_D$, the three alignments between rooted trees will have closed speed, as shown in Figure 11.7.

11.4 Frequencies in Motion Capture Context

In this section we propose to study frequencies (number of iterations per second) of the alignments applied in our motion capture context. We first give some statistics on trees for different subjects and different sizes of grid, then, we show the frequencies results in regard of these statistics.

11.4.1 Statistics on data trees

We propose to study two models: the full human body model and the hand model. Full human body model tree have seven vertices and a maximal degree equal to four. Hand model tree have heights vertices and a maximal degree equal to five.

For the data, we use here the dancer data set (height views) from *4Drepository* from INRIA for full human body application, and a home made data set with four views for hand application. We use these sets on voxels grids with side 40,80,120. We show the results of distributions of tree sizes in Figures 11.8,11.9,11.10 and 11.11.

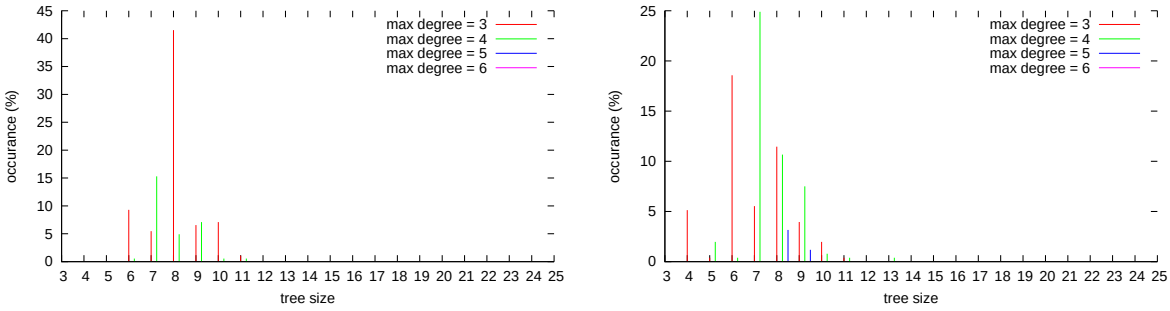


FIGURE 11.8: Distributions for a grid sided by 40. Left: dancer data set. Right: hand data set.

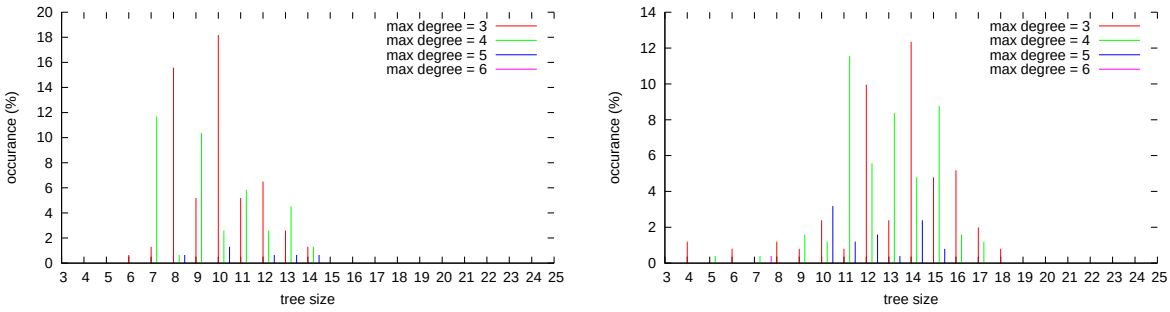


FIGURE 11.9: Distributions for a grid sided by 80. Left: dancer data set. Right: hand data set.

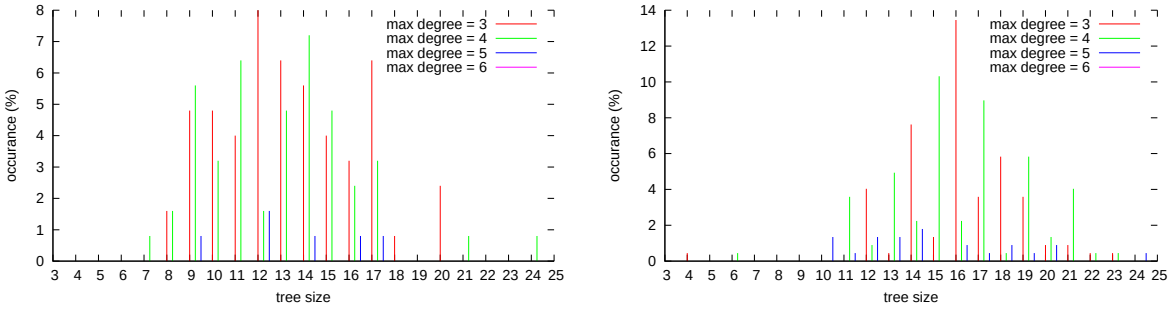


FIGURE 11.10: Distributions for a grid sided by 120. Left: dancer data set. Right: hand data set.

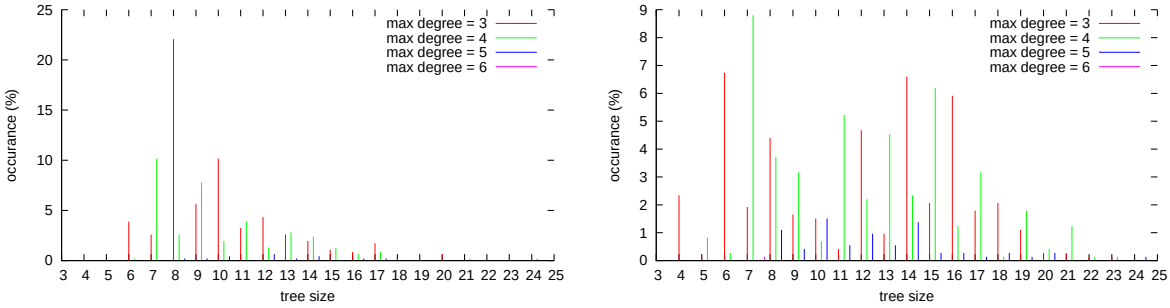


FIGURE 11.11: Average distributions. Left: dancer data set. Right: hand data set.

11.4.2 Choice of algorithm

First, we have to determine which alignment is the most adapted: as we want preserve all vertices in the model tree, and take into consideration useless 2-degree vertices in the data tree, we can use the asymmetric homeomorphic alignment in our application.

In a second time, we have to specify if both the model tree and the data tree can be rooted or not. If the both trees are rooted, it involves that the roots will be matched together. If only one tree is rooted, it involves that no merging can occurs on the root, then the root will already be matched.

In the case of full human body, and for all complete models, the only assumption we can do is that all vertices have to be matched in the model, but their is no particular vertex in the data tree. We can use a rooted version of the model tree, with an unrooted version of the data tree. The choice of the root have consequences on the speed results: if the root of full human body model tree is the torso, the rooted tree have a max number of children equals to four, and only three if the root is an other vertex.

In the case of hand, and for all incomplete models, we can consider that the 1-degree vertex which is an extremity of the infinite weighted edge in the model tree will always match with the vertex associated with the grid border point (supposed unique) in the data tree. Considering this fact, both trees can be rooted on the vertices described above.

11.4.3 Protocol

Each experiment consists of the following:

1. We randomly generate n data trees
2. we record the starting time t_s
3. we compute the asymmetric homeomorphic alignment between the model tree and each data tree
4. we record the ending time t_e
5. the frequency is equal to $\frac{n}{(t_e - t_s)}$

11.4.4 Results

The results, obtained for $n = 10000$, are shown on Figure 11.12 for hand application, and on Figure 11.13 for full human body application.

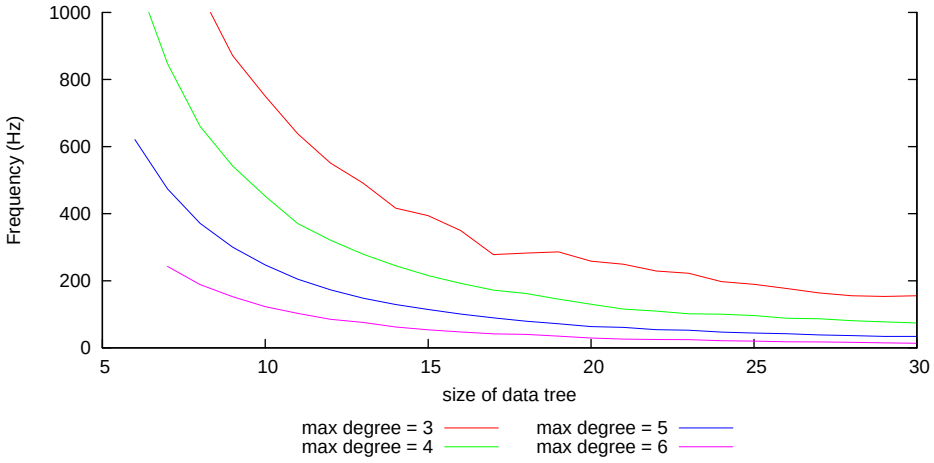


FIGURE 11.12: Frequencies of asymmetric homeomorphic alignment of hand rooted model tree with different rooted data trees.

11.4.5 Discussion

The average frequency can be done by crossing the graphs statistics and the frequencies using the following formula:

$$\sum_{d=3}^{7}\sum_{s=1}^{25} dist(d,s) * freq(d,s)$$

with $dist(d,s)$ representing the percentage of graphs with s vertices and a max degree equal to d, and $freq(d,s)$ representing the average frequency of the matching between the model and a

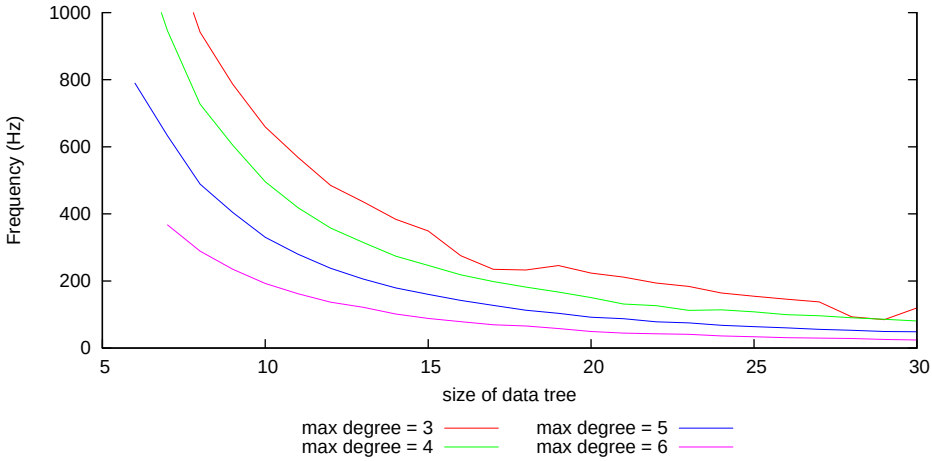


FIGURE 11.13: Frequencies of asymmetric homeomorphic alignment of full human body rooted model tree with different unrooted data trees.

data tree with s vertices and a max degree equal to d. The frequencies for different sizes of grid are shown in Figure 11.14.

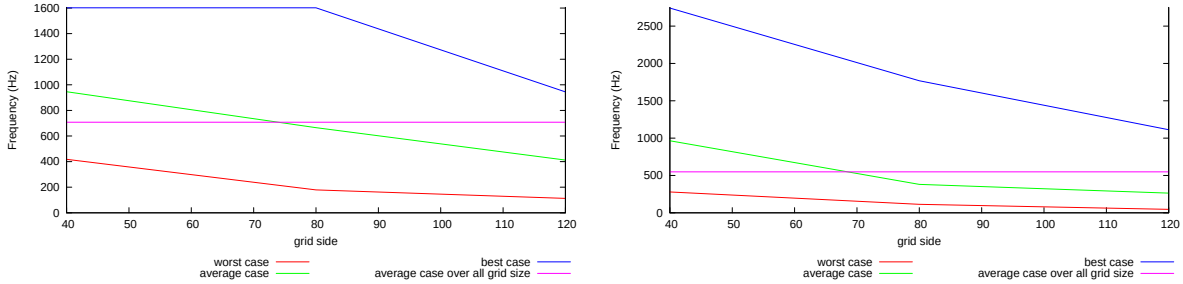


FIGURE 11.14: Average frequencies of asymmetric homeomorphic alignment for different sizes of grid. Left: in full human body context. Right: in hand context.

Then, in the average case:

- for the dancer data set, we can perform more than **708** alignments per second
- for the hand data set, we can perform more than **548** alignments per second

We can consider that the matching step of our method will be performed very fast.

In the worst case:

- for the dancer data set (data tree with 17 vertices and a degree max equal to 5), we can perform more than **89** alignments per second
- for the hand data set (data tree with 24 vertices and a degree max equal to 5), we can perform more than **46** alignments per second

It involve that even if the worst case occurs all the time, the matching with model tree can be performed in real time.

Of course, as the size and the maximal degree of data trees is correlated with the size of the grid, we can choose this size in order to obtain a good compromise between accuracy of positions and computation speed.

Part IV

Motion Capture Applications

Chapter 12

Generic Pose Initialization Step

In the two previous parts of this manuscript, we proposed solutions for both fast skeletonization and adapted matching.

In this chapter, we first describe in details the pipeline of our method, using the works described in the previous parts. Then, in order to improve the robustness of our method, we propose the optional addition of intuitive constraints in the model definition.

12.1 Summary of our Method

Now, as we have found efficient solutions for skeletonization and thinning, we can integrate them in our pipeline. For the skeletonization, the use of our $D6I1D$ thinning algorithm (described in Chapter 6) is interesting, in regard of its speed and it resulting skeleton, which has few spurious branches. In the case of incomplete model, we constrain the thinning process to preserve the centroid of the set of points on the boundaries of the grid, belonging to the shape. For the data tree extraction from the skeleton, we use the Algorithm 13 (defined in Section 7.3).

12.1.1 Use of Asymmetric Homeomorphic Alignment

The matching of the model tree with the data tree is done by asymmetric homeomorphic alignment (described in Chapter 10), specially designed for this purpose. Three different algorithms have been proposed for the computation of asymmetric homeomorphic alignment, depending on the kind of trees to align:

a) two unrooted trees: this algorithm can be used for matching model tree and data tree without constraints.

b) one rooted tree and one unrooted tree: applying this algorithm involves that the root of the rooted tree matches with a vertex of the unrooted tree, and cannot be removed by a merging.

c) two rooted trees: applying this algorithm involves that the two roots match together.

When applied to the same data, algorithm *c*) is faster than algorithm *b*), which is itself faster than algorithm *a*). In order to use the full potential of the proposed algorithms, we have to consider different cases, in regard of the class of model.

In case of incomplete model, it is easy to see that infinity weighted edges in both model tree and data tree have to match. Rooting the trees in the two unique 1-degree vertices bounding these edges involves matching them together, and by this way, to match together the unique edges linked to them.

In case of complete model, as we are mostly interested in matchings that preserve a large part of the model tree vertices, we can root the model in one of the vertices which have less chances to be removed, a high degree vertex for example. As we have no information about the importance of data tree vertices, this tree is considered as unrooted.

For the purpose of scale invariance, we start by normalizing the weights of the two trees, so that the sum of all non-infinite weights in a tree is equal to 1. Then, we compute the asymmetric homeomorphic alignment (and the associated distance) between the trees. If the distance is greater than a given threshold T_d, which is defined by the user, we assume that the data tree is not enough similar to the model tree to provide a good matching, and we stop the pipeline for this frame. Otherwise, we use the optimal asymmetric homeomorphic alignment to match the labels associated with the model vertices, with the 3D positions associated to the data vertices.

The number of non-aborted matchings, and their quality, obviously depend on the choice of the threshold value T_d : as both trees are normalized, $T_d = 2$ means that no matching will be aborted (both trees can be deleted), but the resulting matching can be poor. On the other hand, a lower value yields a greater amount of aborted matchings, but with better probability of good matching.

12.1.2 Pipeline

A detailed view of the pipeline of our method is shown in Figure 12.1.

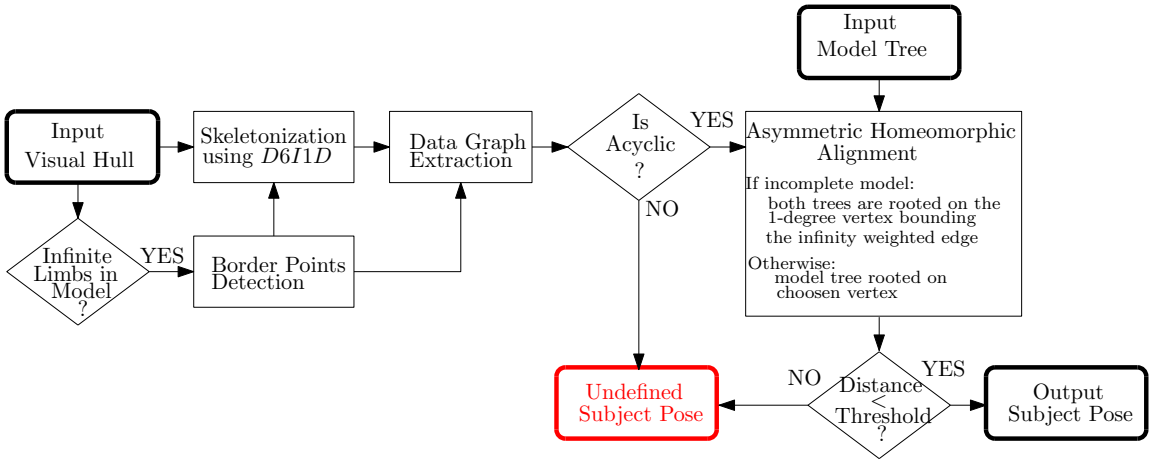


FIGURE 12.1: Detailed pipeline of our method. Rounded rectangles represent input and output data. Other rectangles represent steps of the method.

12.2 Optional Constraints

The method proposed in the previous section already provides some good results. However there are still some special cases which can not be solved by our method, even with sufficient topological information.

The first special case occurs when the different limbs of the subject are represented with wrong lengths in the skeleton. It can be due to the clothes of the subject. For example, if the subject wears a dress, the legs seem small in the visual hull, and the lengths of the branches representing them in the skeleton can be smaller than those representing the arms. The alignment algorithm will then propose a matching of the arms of the model with the legs of the subject, and inversely. Figure 12.2 illustrates this kind of cases.

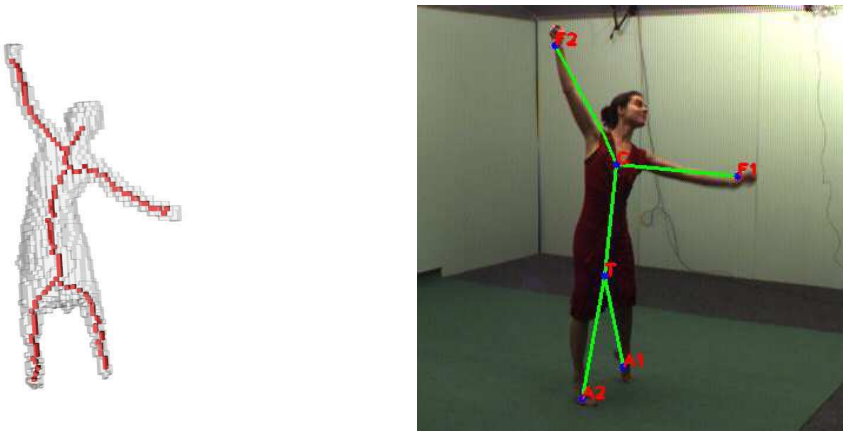


FIGURE 12.2: On the left, a visual hull of full human body where the skeleton legs branches are shorter than arms branches. On the right, the incorrect pose found by the method, due to the length variations.

The second special case occurs when different limbs of a model are represented by edges with similar lengths in data tree. In the case of the hand for example, we can consider the case of four fingers: index, middle finger, ring finger and little finger.

In the reality, little finger is always (or at most in a wide majority of case) smaller than the three others, and the middle finger is longer than the other fingers. It seems natural to represent them in this way in the model, by giving a higher value to the edge representing the middle finger, and a smaller to the little finger, in order to differentiate them.

However, the branches of the skeleton representing these fingers can have very variating lengths, due to the visual hull reconstruction, for example. The branch representing little finger can be longer than the one representing the middle finger, as shown in Figure 12.3. To solve this problem, we have to represent the four fingers by edges with same length, and to find an other way to differentiate them.

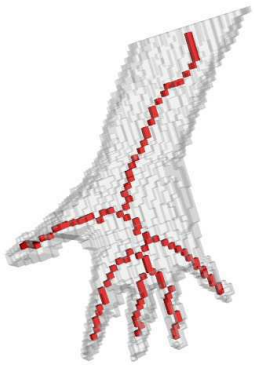


FIGURE 12.3: A visual hull of hand where the skeleton finger branches are not proportional to the real lengths of the fingers.

In order to solve this kind of problem, we propose to add optional constraints in our method. These constraints have to be intuitive and easy to describe, in order to preserve the philosophy of our method ("keep it simple"). We propose two different kinds of constraints, both requesting a specific positioning of a subject part relative to others.

A coordinate constraint corresponds to constraining the position of a subject part in regard of another linked part, for a given axis (for example, constrain the head to be above the torso).

A betweenness constraint corresponds to constraining the position of a subject part in regard of two others, requesting to be between them (for example, constrain the middle finger to be between the index and the ring finger).

In the sequel of this section, we propose a way to integrate these optional constraints in the model definition and in the pipeline of the method.

12.2.1 Coordinate Constraints

A coordinate constraint consists of forcing a part of the model to be positioned in a subspace defined by one of its linked parts. For example, in the case of full human body model, we can constrain the head to be above the torso: let $P_H = (x_H, y_H, z_H)$ be the 3d position of the head and $P_T = (x_T, y_T, z_T)$ the one of the torso, the constraint is $P_H \in \{P = (x, y, z) \in \mathbb{R}^3; y > y_T\}$.

Considering any rooted version of both the model tree and the data tree, and using the property that coordinate constraint is defined on linked parts only, we can translate coordinate constraint into a constraint on the tree matching.

Let v_m and w_m be the two vertices associated to the constraint in the model tree. For example, for the head-torso constraint, v_m (respectively, w_m) represents the torso (respectively, the head). By definition of the constraint (the two parts have to be linked), there exists an arc $a_m = (v_m, w_m)$ belonging to the model tree. Let v be any vertex of the data tree, we denote by $P(v) = (P_x(v), P_y(v), P_z(v))$ the 3d point associated to this vertex.

The coordinate constraint on a_m can be expressed as follows: let $a_d = (v_d, w_d)$ be an arc of the data tree (possibly obtained by some merging), a_d can match with a_m if and only if $P(v_d)$ and $P(w_d)$ have relative positions satisfying the constraint. For example, for the head-torso constraint, a_d can match with a_m if and only if $P_y(w_d) > P_y(v_d)$.

We propose to integrate coordinate constraints in the alignment computation, by modifying the resizement cost formulation for constrained edges. We consider the case of asymmetric homeomorphic alignment for rooted trees (AHA between unrooted trees just consists in searching the best AHA for all rooted versions of the trees, please refer to Chapter 10 for more details).

The cost of the resizement of a model arc $a_m = (v_m, w_m)$, weighted by W_m, to the weight W_d of a data arc $a_d = (v_d, w_d)$ is equal to $+\infty$ if there is a coordinate constraint C associated to a_m and the relative positions of $P(v_d)$ and $P(w_d)$ are not compatible with C. Otherwise, the cost is still $\gamma(W_m, W_d)$.

12.2.2 Betweenness Constraints

In case of similar limbs, the matching with the model tree can generate multiple solutions. In the case of our model of hand, as the descriptions of index, middle finger, ring finger and little finger are identical, the matching will give a set of four possible positions for each finger. To

solve this problem, we introduce the betweenness constraints. It consists of constraining the position of some limb to lie between two other ones.

An usual definition of a betweenness ternary relation is based on collinearity [59] : a point P_3 is said to be between two other points P_1 and P_2 (denoted by $R_c(P_1, P_2, P_3)$) if it belongs to the segment P_1P_2. However, this definition is too restrictive.

On the other hand, we could say that a point P_3 is between P_1 and P_2 (denoted by $R_p(P_1, P_2, P_3)$) if its orthogonal projection on the line (P_1P_2) is between P_1 and P_2. In this case, the problem is that there exist triplets (P_1, P_2, P_3) such that each point is between the two other ones (for example, the three vertices of an equilateral triangle).

These two relations are illustrated in Figure 12.4.

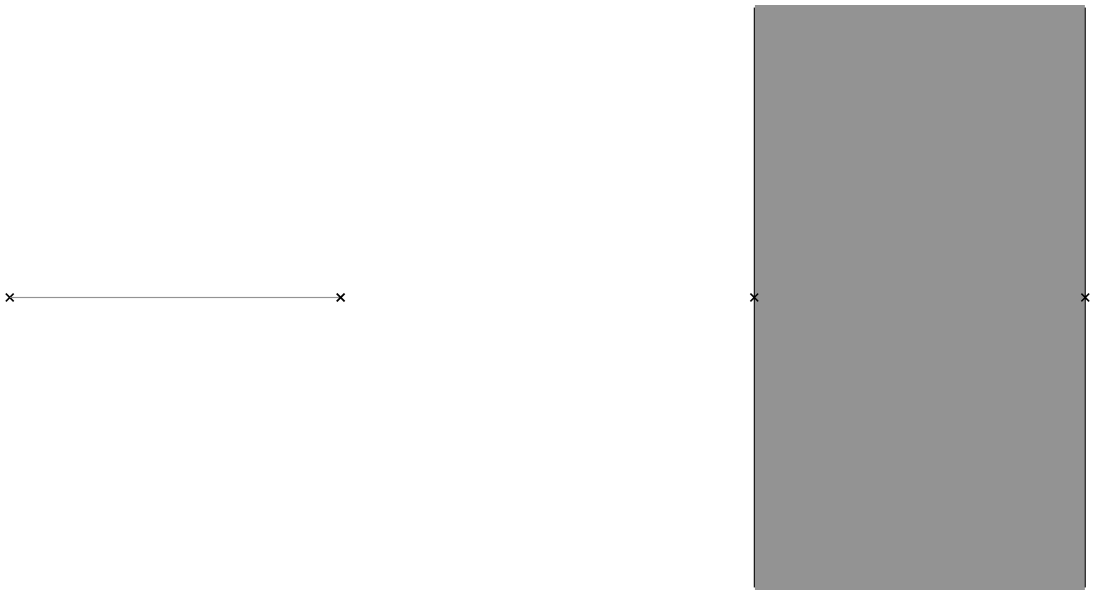


FIGURE 12.4: All points in the grey area are between the two points represented by black crosses: at left, in the sense of R_c, at right, in the sense of R_p.

12.2.2.1 Design of Intuitive Betweenness Relation

We propose a definition which is not too restrictive, and which gives at most one possibility of "betweenness" for three points P_1, P_2, P_3 : consider $\mathcal{B}$ the unique ball with diameter P_1P_2 minus the unique sphere with diameter P_1P_2. The point P_3 is between P_1 and P_2 (denoted by $R_b(P_1, P_2, P_3)$) iff $P_3 \in \mathcal{B}$ (see Figure 12.5 for an illustration).

An other formulation is that P_3 is between two points P_1 and P_2 iff the angle between $\overrightarrow{P_3P_1}$ and $\overrightarrow{P_3P_2}$ is greater than $\frac{\pi}{2}$. This definition is easy to use in our context, and provides a unique result for a wide set of configurations (all the configurations where the 3 points are the vertices of obtuse or right triangles). However, in order to cover all possible configurations, we have to consider other definitions.

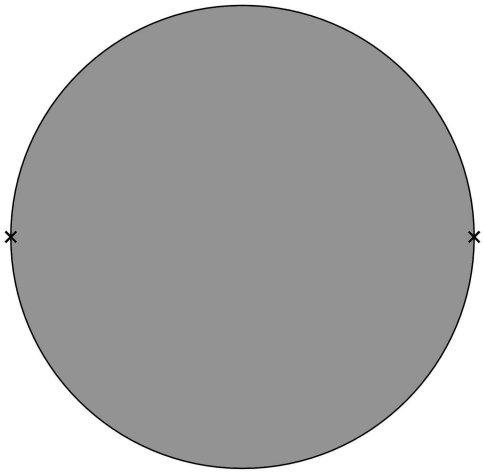


FIGURE 12.5: All points in the grey area are between the two points represented by black crosses, in the sense of R_b.

12.2.2.2 Design of Mathematically Efficient Betweenness Relation

In order to design a betweenness relation on 3D points adapted to our purpose, we first need to define its required properties. A well adapted set of properties in our case can be based on the postulates of betweenness proposed by Huntington and Kline in 1917 [68] and refined by Huntington [67].

Definition 29. *[67, 68] Let S be a set and R a ternary relation on S. The notation abc indicates that three given elements a,b and c of S, in the order stated, satisfy the relation R. The notation $\overline{abc}$ indicates that a,b and c do not satisfy the relation R. The couple (S, R) is a betweenness system if:*

Postulate A: $abc \Rightarrow cba$.

Postulate B: $a \neq b$ *and* $a \neq c$ *and* $b \neq c \Rightarrow bac$ *or* cab *or* abc *or* cba *or* acb *or* bca.

Postulate C: $a \neq b$ *and* $a \neq c$ *and* $b \neq c$ *and* $abc \Rightarrow \overline{acb}$.

Postulate D: $abc \Rightarrow a \neq b$ *and* $a \neq c$ *and* $b \neq c$.

Postulate 9: $a \neq b$ *and* $a \neq c$ *and* $b \neq c$ *and* $x \neq a$ *and* $abc \Rightarrow abx$ *or* xbc.

Definition 30. *[67] A system is a near-betweenness system if it possess all but one of the five properties of a betweenness system.*

In our case, $S = \mathbb{R}^3$. *Postulate A* involves a symmetry in the relation. *Postulate B* involves that among all configurations of three distinct points, at least one satisfy R. *Postulate C* involves that for three distinct points, only one is between the two others. *Postulate D* involves that the

relation is only satisfied on distinct points. Finally, *Postulate 9* is required to derive an order on S from the ternary relation. In our case, it is not necessary.

These postulates have been proposed in a framework considering only 1D sets, like numbers or points of a line. In higher dimensions, some configurations are unsolvable: if three points are equidistant, it is impossible to find a relation such that one and only one of the points is between the others. We propose to modify the postulates to take this into consideration.

Postulate B': $a \neq b$ and $a \neq c$ and $b \neq c$ and a, b, c not equidistant $\Rightarrow$ bac or cab or abc or cba or acb or bca.

Postulate D': $abc \Rightarrow a \neq b$ and $a \neq c$ and $b \neq c$ and a, b, c not equidistant.

To sum up, we have to design a ternary relation B such the system $(\mathbb{R}^3, B)$ is a near-betweenness system satisfying the postulates A,B',C and D'.

However, we have to notice that these postulates are necessary but not sufficient for our context. In order to show it, we propose a betweenness relation satisfying the four postulates but which is unusable in our case.

Unusable Betweenness Relation By definition, in any non-equilateral triangle, there is at least one edge with a length different from those of the other edges. We denote by $d(a, b)$ the Euclidian distance between two points a and b. We denote by $\delta : \mathbb{R}^3 \times \mathbb{R}^3 \times \mathbb{R}^3 \rightarrow \mathbb{R}$ the function:

$$\delta(a, b, c) = |d(a, b) - d(b, c)|$$

An interesting property is that if a,b and c are the three distinct vertices of a non-equilateral triangle, one and only one of the three values $\delta(a, b, c), \delta(a, c, b)$ and $\delta(c, a, b)$ is strictly less than the two others. This can easily be proved by showing that the only case where two of the three values are equal is for isosceles triangle, and in that case, these values are strictly greater than the last one, which is equal to zero.

Using this property, we can define a betweenness relation R_u satisfying the four postulates. Let a, b and c be three distinct and non-equidistant points, b is between a and c ($R_u(a, b, c)$ is true) if and only if $\delta(a, b, c) \leq \delta(b, c, a)$ and $\delta(a, b, c) \leq \delta(b, a, c)$.

However, even if this relation satisfies the four postulates, it is not usable in our case, due to the fact that it is not intuitive for the common conception of betweenness (see Figure 12.6 for some examples). Due to this fact, it is very difficult to propose betweenness constraints using this relation.

As our model description has to be simple, we have to find a betweenness relation which satisfies the four postulates and which is intuitive enough for an easy usage.

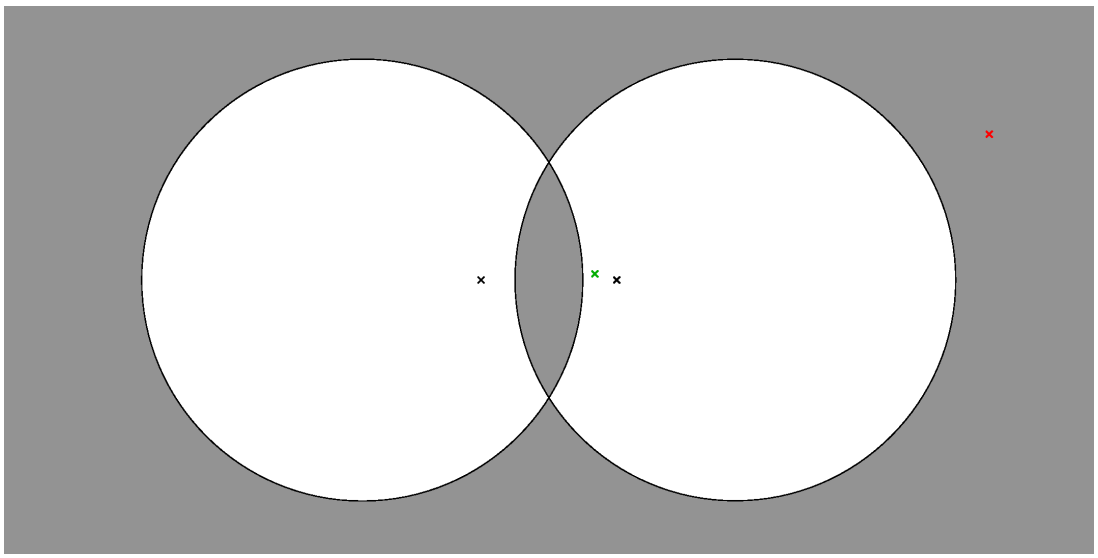


FIGURE 12.6: All points in the grey area are between the two points represented by black crosses(in the sense of R_u). We can notice that the green cross is between the black ones, in the common conception of betweenness, but is not in the sense of R_u. In the same way, the red cross is not between the black ones, in the common conception of betweenness, but is so in the sense of R_u.

More Intuitive Betweenness Relation First we have to introduce a new postulate providing concept of intuition for the common conception of betweenness.

Definition 31. *Let R a ternary relation on S. The couple $(\mathbb{R}^3, R)$ is an intuitive betweenness system if:*

- *it satisfies the postulates A,B',C and D'.*
- ***Postulate I:** $abc \Rightarrow$ orthogonal projection a' of a on the line (bc) is between b and c (in the sense, $d(a', b) + d(a', c) = d(b, c)$).*

We propose now a definition of intuitive betweenness relation, in the sense of the above definition. The non-equilateral triangles can be classified in two categories: isosceles triangles and non-isosceles triangles.

In the case of isosceles triangles, one of the three edges has a length different to those of the two others, by definition. The vertex at the opposite of this edge can be distinguished from the two others by this way. It is interesting to notice that the orthogonal projection of this vertex on its opposite edge is always between its boundaries.

In the case of non isosceles triangles, one of the three edges is greater than the two others. This way, the vertex at the opposite of this edge can be distinguished from the two others. It is interesting to notice that the orthogonal projection of this vertex on its opposite edge is always between its boundaries.

Using these properties, we can define a betweenness relation R_i satisfying the four postulates and which is intuitive. Let a, b and c be three distinct and non-equidistant points, b is between a and c ($R_i(a, b, c)$ is true) if and only if one of the following conditions holds:

1. $d(a, c) > d(b, c)$ and $d(a, c) > d(b, a)$.
2. $d(b, c) = d(b, a)$ and $d(a, c) \neq d(b, c)$.

This relation is illustrated in Figure 12.7.

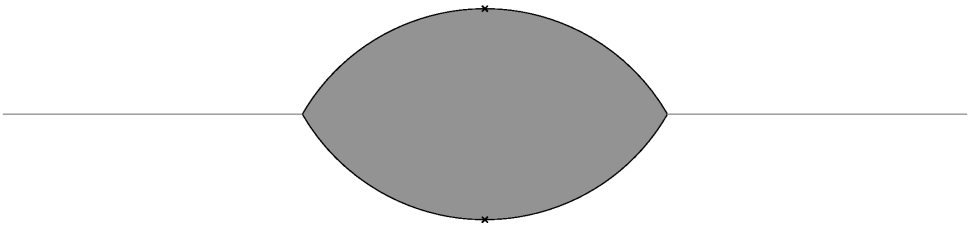


FIGURE 12.7: All points in the grey area are between the two points represented by black crosses(in the sense of R_i).

12.2.2.3 Application in our Method

In our model, a betweenness constraint is represented by a sequence of at least three model tree vertices. For the example of hand fingers, the constraint is formulated as "betweenness: thumb, index, middleFinger, ringFinger, littleFinger".

The application of this constraint is done on the output of the tree matching step, which provides multiple matching possibilities for some vertices. Then, if there exists a betweenness constraint on model vertices, defined by a sequence $m_0, ..., m_n$, the corresponding data vertices $d_0, ..., d_n$, with associated positions $P_0, ..., P_n$ must be chosen such as, for a given betweenness relation R, for each $i \in [1, n-1]$, $R(P_{i-1}, P_i, P_{i+1})$.

The choice of the best betweenness relation will be empirically discussed in the next chapter.

12.3 Complete Method and Model Definition

Now we have solutions for problematic cases, thanks to the new optional constraints. The final pipeline of our method, including the management of these constraints, is shown in Figure 12.8.

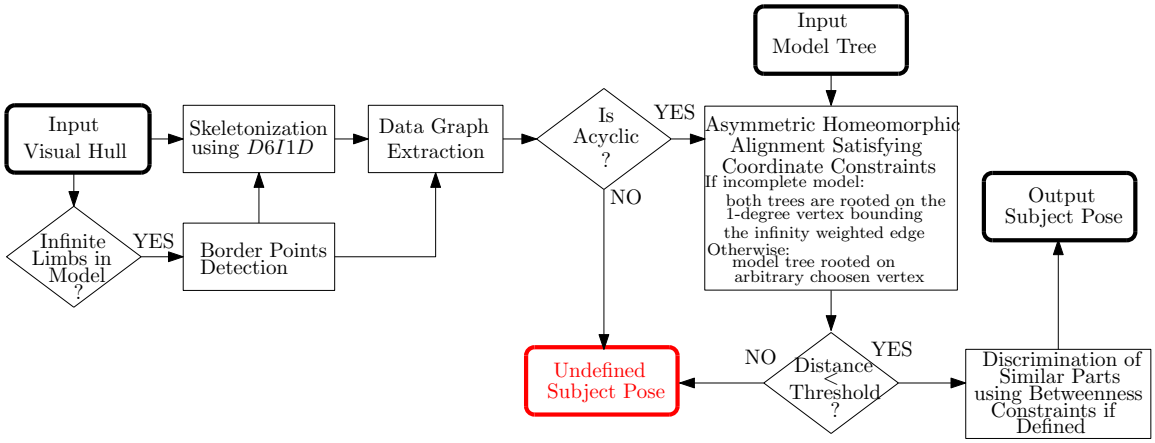


FIGURE 12.8: Complete pipeline of our method. Rounded rectangles represent input and output data. Other rectangles represent steps of the method.

The new constraints can be integrated in our a priori models in order to solve the problematic cases, as shown in Figure 12.9.

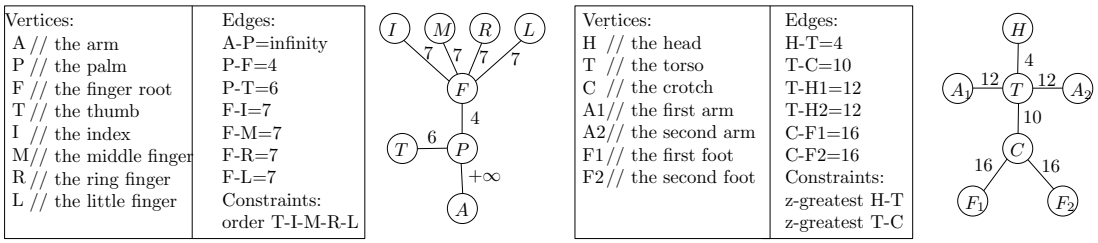


FIGURE 12.9: Full description of two different subjects. On the left, the hand model description (incomplete model). On the right, the full human body model description (complete model).

For the hand model, a betweenness constraint is set on the five fingers, in order to allow the pose estimation for a natural initial pose of the hand, for example when the hand is wide open.

For the full human body model, coordinate constraints are set on couples head-torso, and torso-crotch, allowing a high variety of initial poses, the only requirement being that the subject stands up and that the different parts of the body are not too close to each other.

Chapter 13

Results and Discussion

In this chapter, we present the results obtained with our pose initialization method, by considering the speed of our method in different situations (voxel grid size, subject), and by showing some samples of output poses.

Then, using empirical data, we discuss the choice of the good threshold to use for the acyclic homeomorphic alignment distance, and of the most adapted betweenness relation to use for betweenness constraints.

Finally, we propose some ways to use our method, in combination with other works found in the literature.

13.1 Implementation and Data Sets

13.1.1 Implementation

Our method has been tested on a computer with a processor Intel(R) Core(TM) 2 Quad Q8200 at 2.33 GHz, a GPU Nvidia(R) Geforce(TM) 9800 GT and 3 Go of RAM. Our program is implemented in C++, and the visual hull is computed on GPU, using GLSL.

The program has two modes:

- In *online mode*, input images are acquired from calibrated cameras (calibration can be done in a prior step, or can be loaded using previously saved data), and silhouettes are extracted using background extraction.
- In *offline mode*, input images are loaded from a data set directory, containing input images, silhouette images (or background images used to extract them), and all required parameter files (e.g. for camera calibration).

In both cases, the program computes our pose initialization method in the following loop that consists of four steps:

1. loading of images and silhouette extraction (or direct silhouette loading when available)
2. visual hull computation
3. pose initialization step application
4. display of results

Several options can be set using command line arguments, e.g. allowing to only perform the two or three first steps.

13.1.2 Used Data Sets

For our tests, we used several data sets provided by INRIA Perception Group [1]:

- for the application to full human body, we used the *dancer* data set (see Figure 13.1a), showing a professional dancer wearing a dress doing some moves. This data set is composed of the synchronized images from 8 calibrated cameras and also provides the white on black silhouette of the subject for each image. This data set is interesting for several reasons:
  - as the subject wears a dress, it allows us to observe the impact of length variation of observed limbs;
  - the subject does different poses, not conventional for usual initialization steps, but with all limbs separated from the others, allowing the study of the robustness of our method;
  - the dress is attached to a floating piece of fabric, which is difficult to differentiate from a limb in visual hull. It is interesting to observe the robustness of our method in regard of this additional "noise".
- also for the application to full human body, we used the *children* data set (see Figure 13.1b), showing a young boy performing some cartwheels. This data set is composed of the synchronized images (and their silhouettes) from 16 calibrated cameras (only 8 are used in our program). This data set is interesting as it forbids the use of coordinate constraints (the torso is above the head in a large amount of frames).
- for the application to animals, we used the *dog* data set (see Figure 13.1c), showing a dog trying to catch a ball held by a man. This data set is composed of synchronized images from 16 calibrated cameras (only 8 are used in our program). This data set is interesting for

[1] http://4drepository.inrialpes.fr/

showing the trans classes adaptability of our method. However, the automatic silhouette extraction is very difficult on this data set, due to the fact that dog legs have the same color as the shadows. For our needs, we did manually extract silhouettes for a small set of frames.

In addition, we used a home made *hand* data set (see Figure 13.1d) for hand pose estimation. This data set is composed of the synchronized images from 4 calibrated cameras. As camera positions and silhouette extractions are not performed with a good quality, the visual hull is very noisy: fingers are sometimes cut. However, this data set is interesting for the observation of the robustness in case of bad quality 3D reconstruction.

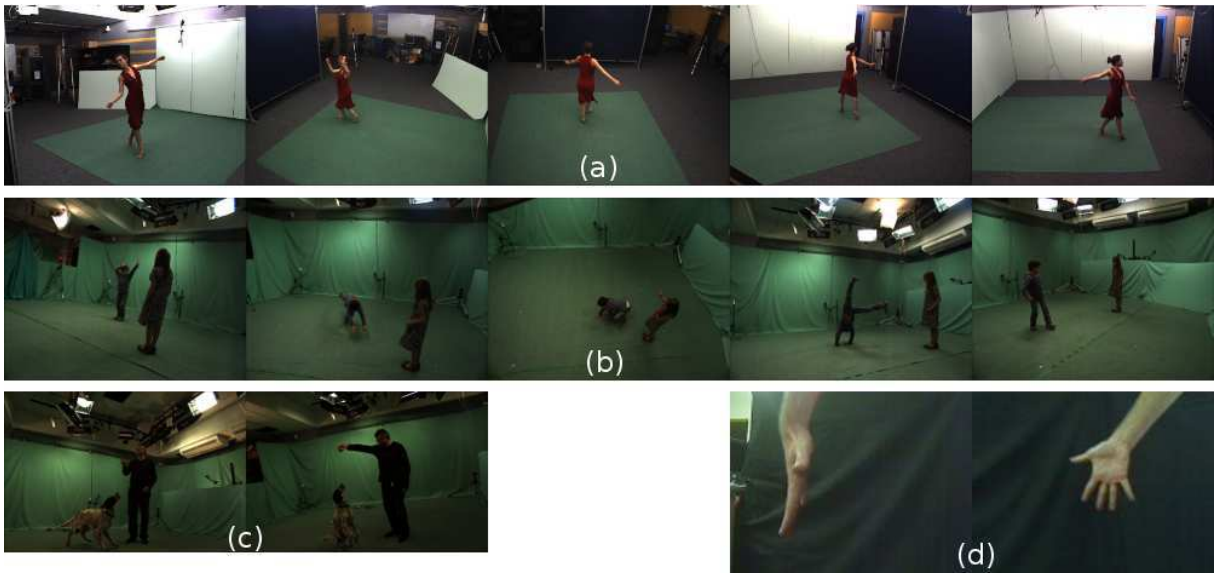


FIGURE 13.1: Samples of the different used data sets: (a) dancer, (b) children, (c) dog and (d) hand.

The visual hull is computed for a cubic voxel grid with parameterizable resolution positioned in such a way that the whole set of actions performed by the subject is covered as tightly as possible.

13.2 Results

13.2.1 Speed

In order to measure the computation speed of our method, we use the following methodology: for each data set (*dancer*, *hand* and *children*) and for each resolution of voxel grid, we measure the computation times of the application performing only its two first steps (image loading and visual hull computation) and its three first steps (the third being initialization step application). We call these two computation times $CT2$ and $CT3$, respectively. Let Nb_{frames} be the number

of frames of the current data set. The computation time of our method is thus obtained using the following formula: $\frac{CT3 - CT2}{Nb_{frames}}$. The average frame rate of our method can be obtained by inversing the previous formula.

Figure 13.2 shows computation time and frame rate for different data sets and different voxel grid resolutions. It can be observed that for each data set, real-time (40 milliseconds or 25 iterations per second) is reached for voxel grid resolutions lower than $100 \times 100 \times 100$. It is

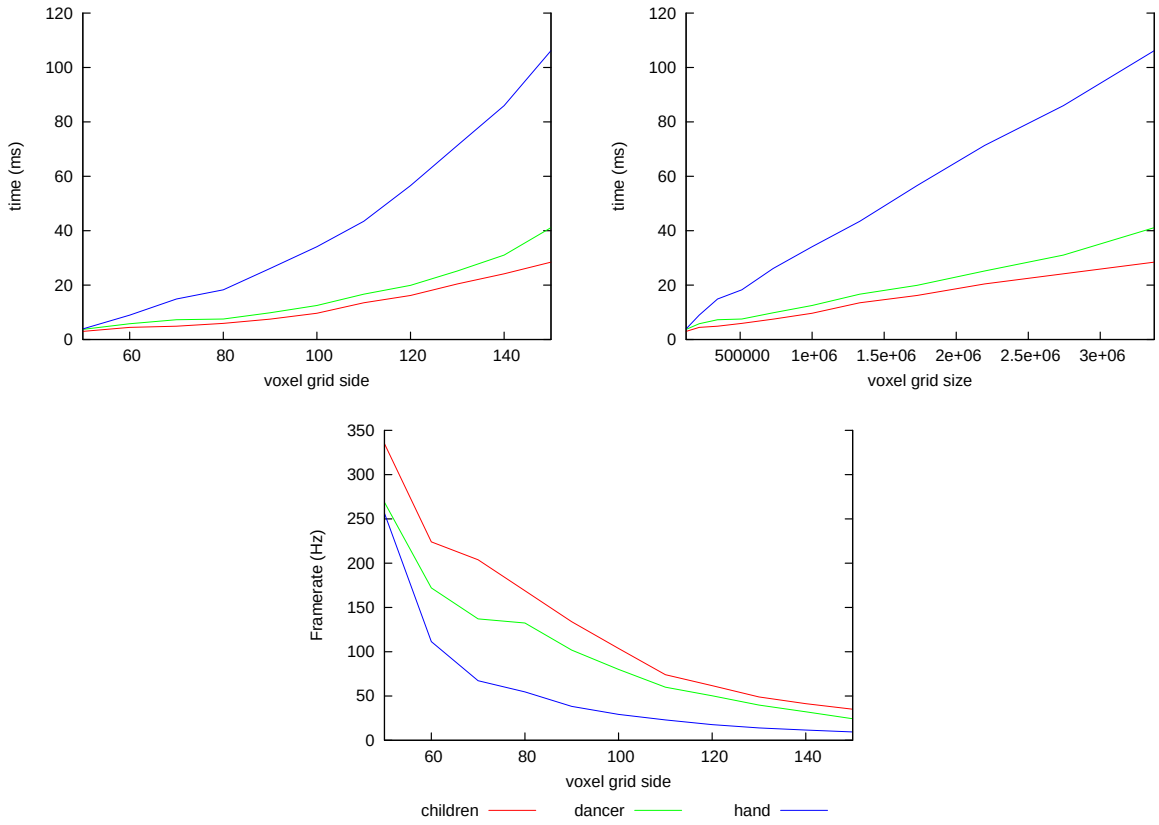


FIGURE 13.2: Speed results for three different data sets: *children*, *dancer* and *hand*. On top, average speed computation for one frame, for different sizes of voxel grid: on the left, in regard of the number of voxels per side of the voxel grid; on the right, in regard of the number of voxels of voxel grid. On bottom: average frame rate in regard of the number of voxels per side of the voxel grid.

interesting to remark that method has a linear time complexity in regard of the voxel grid size (see Figure 13.2 top right plot).

13.2.2 Proportion of False Positives

A critical point in initialization step methods is the proportion of false positives, i.e. the proportion of invalid output poses. In order to appreciate the quality of an output pose for our method, as no ground truth data is available, we propose to classify them in regard of their usability, which can be visually measured:

1. an output pose is a *good matching* (see Figure 13.3a for an example) if all parts are positioned at appropriate locations;

2. an output pose is a *missing limb matching* (see Figure 13.3b for an example) if one part has no position, other parts being positioned at appropriate locations;

3. an output pose is a *partial matching* (see Figure 13.3c for an example) if several parts have no position, other parts being positioned at appropriate locations;

4. an output pose is a *misplaced limb matching* (see Figure 13.3d for an example) if at least one part is positioned at inappropriate location;

5. an output pose is a *bad matching* (see Figure 13.3e for an example) if almost all parts are positioned at inappropriate locations;

6. an output pose is *not a matching* if at most one part is matched.

Classes 3 and 6 can be easily detected and rejected by our method, due to the lack of part positions. Classes 1 and 2 (good matchings and missing limb matchings) are considered as usable for initialization (we call them True Positive, shorted in TP), and classes 4 and 5 are considered as unusable for initialization (we call them False Positive, shorted in FP).

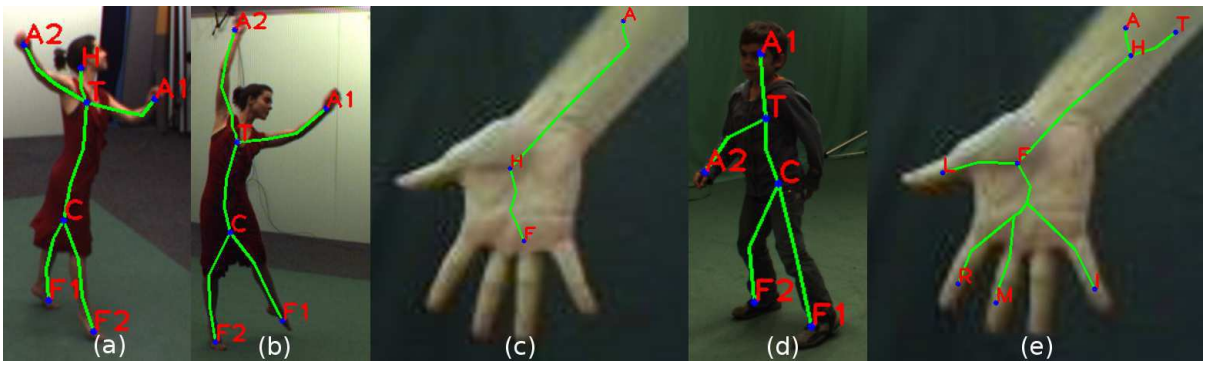


FIGURE 13.3: Examples for the different classes of matchings: (a) good matching, (b) missing limb matching, (c) partial matching, (d) misplaced limb matching, (e) bad matching.

We propose to study the quality of the output pose in regard of the Asymmetric Homeomorphic Alignment (AHA) distance from the model tree to the data tree, in order to highlight the correlation between them.

For this purpose, we measure both the AHA distance and the quality of output pose (manually estimated) for each frame of different data sets, and for different grid resolutions. From these measures, we compute, for each AHA distance threshold, three values:

- the ratio of the number of true positives over the number of frames of the data set;

- the ratio of the number of false positives over the number of frames of the data set;
- the ratio of the number of false positives over the number of matchings (true positives plus false positives) .

The study of these data provides a lot of interesting information, as the detection of an efficient AHA distance threshold, the impact of grid resolution on the validity of the results, or the choice of the most adapted betweenness relation.

13.2.2.1 Study of Full Human Body Matchings

Results for dancer data set and children data set are shown in Figure 13.4 and Figure 13.5, respectively. We remind that the main difference between these data sets is that no coordinate constraint can be set in the case of children data set, due to the performed action (cartwheel). Due to this fact, we obviously expect better results for the dancer data set.

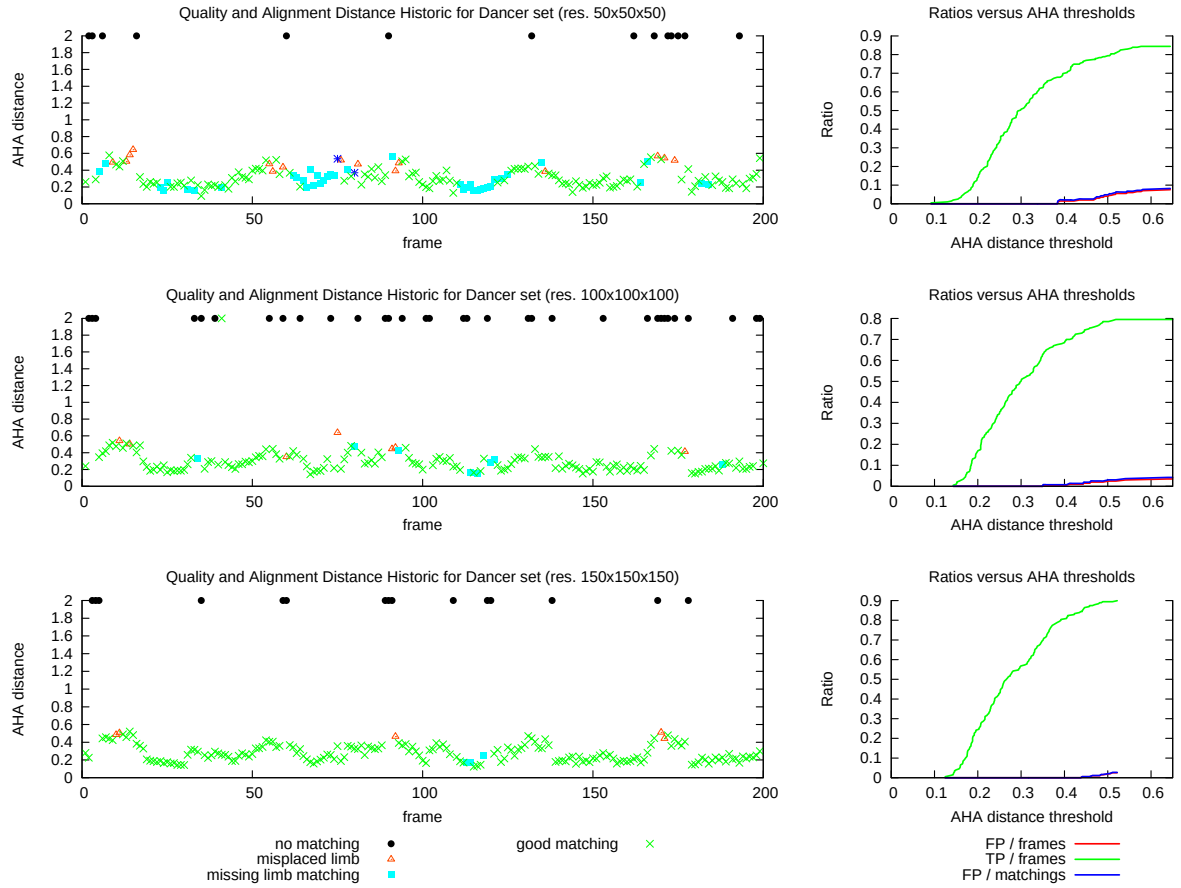


FIGURE 13.4: Study of correlations between AHA distance and matching quality for dancer data set with different grid resolutions. Left: historic of AHA distances and matching qualities all along the sequence. Right: different ratios in regard of given AHA distance threshold. From top to bottom: for resolutions 50^3, 100^3 and 150^3.

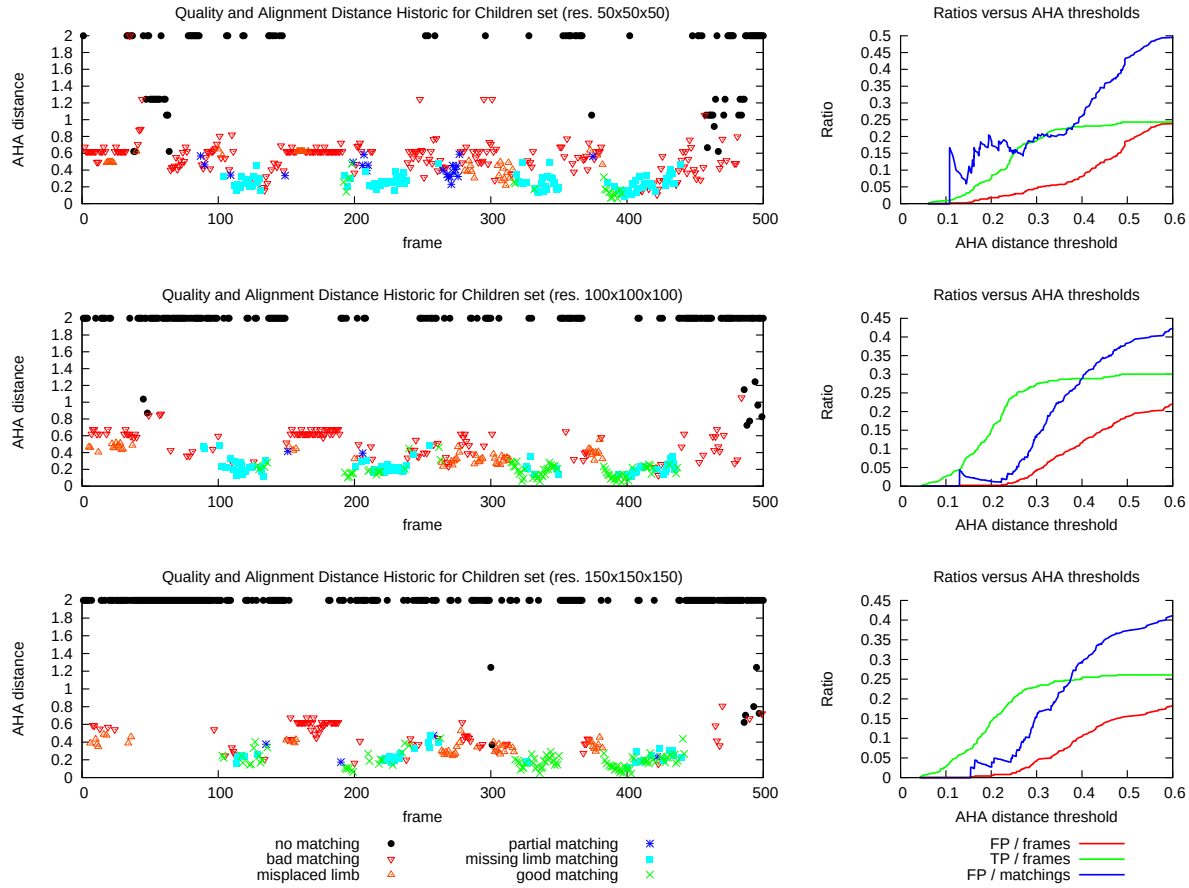


FIGURE 13.5: Study of correlations between AHA distance and matching quality for children data set with different grid resolutions. Left: historic of AHA distances and matching qualities all along the sequence. Right: different ratios in regard of given AHA distance threshold. From top to bottom: for resolutions 50^3, 100^3 and 150^3.

First, we observe from measures for both data sets that a higher grid resolution provides better quality (more true positives, missing limbs matching being replaced by good matchings). However, this improvement of the quality is more sensible from 50^3 resolution to 100^3 resolution than from 100^3 resolution to 150^3 resolution. Thus, we can assume that for all grid resolutions precise enough for a good visual hull definition, where each limb is clearly separated from others, and allowing a good thinning, the output quality will be about the same. In the same way, we can observe that false positives are fewer for higher grid resolutions.

Concerning the dancer data set, the study of ratios shows that very few false positives are associated to AHA distance lower than 0.4. The use of our method with AHA distance threshold set at this value provides a good matching for between 70 and 90 percent of the frames, with a fail rate lower than five percent.

Concerning the children data set, the study of ratios shows that for 50^3 grid resolution, no AHA distance threshold can be found in order to provide acceptable amount of matching with low

rate of false positives. However, for higher resolutions, setting threshold to 0.2 provides a good matching for between 15 and 20 percent of the frames, with a fail rate lower than five percent.

The impossibility to use our method for 50^3 grid resolution for children data set is partially due to the fact that grid volume is higher for children data set than for dancer data set, as the subject is more mobile. A solution should be to refocus the grid on the subject at each frame.

13.2.2.2 Study of Hand Matchings

The main goal of our study of false positive for hand pose estimation is to find the most adapted betweenness relation to use for the betweenness constraints solving. For this purpose, we propose to compare the results for the hand data set, using on one hand the "ball-based" intuitive betweenness relation R_b defined in Section 12.2.2.1, and on the other hand the "mathematically efficient" intuitive betweenness relation R_i defined in Section 12.2.2.2. The measures are shown in Figure 13.6 and Figure 13.7, respectively.

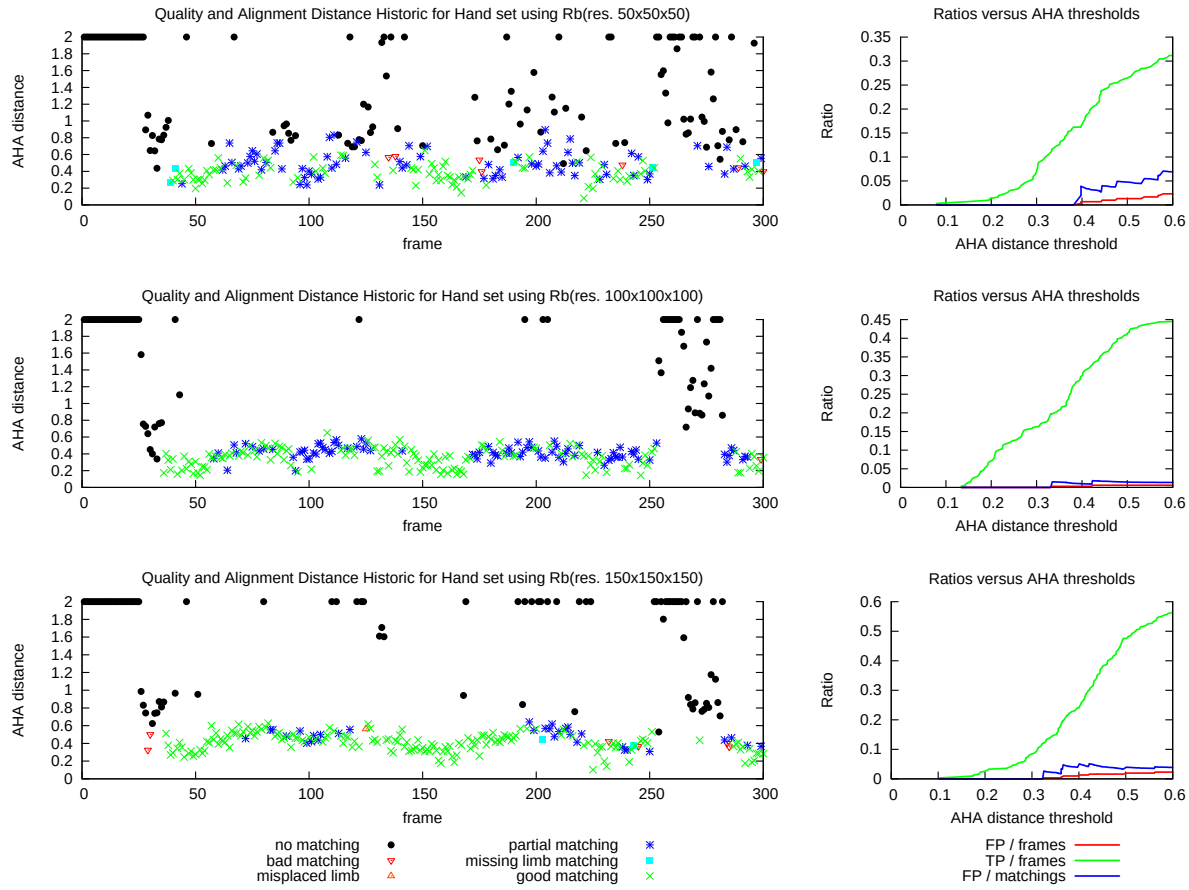


FIGURE 13.6: Study of correlations between AHA distance and matching quality for hand data set with different grid resolutions, using betweenness relation R_b. Left: historic of AHA distances and matching qualities all along the sequence. Right: different ratios in regard of given AHA distance threshold. From top to bottom: for resolutions 50^3, 100^3 and 150^3.

We can observe that the rate of true positives for a given AHA distance threshold is nearly invariant in regard of the chosen betweenness relation. However, the rate of false positives is sensibly higher when the R_i betweenness is used. A quick observation of the measures allows us to see that the use of R_b provides a larger amount of partial matchings than the use of R_i, and a fewer amount of bad matchings, especially in case of low grid resolutions.

This fact can be easily explained by analyzing the differences between these two relations: for three distinct and non-equidistant points, there is always one point which is between the two others, in the sense of R_i, even if the points are close to be equidistant. It is not the case for R_b: when three points are close to be equidistant, none is between the others.

For low grid resolutions, positions of fingers are more approximative, leading to ambiguous cases (where three fingers are equidistant). Using R_i, the method will always provide an order, even for these ambiguous cases, possibly leading to a bad matching. Using R_b the method will provide an order only for non-ambiguous cases, the other leading to partial matchings, which can be detected and rejected.

The most adapted betweenness relation for our purpose is thus the "ball-based" betweenness relation R_b, which is also the fastest to compute.

Concerning the AHA distance threshold, a good value seems to be 3.5, providing a good matching for between 10 and 20 percent of the frames, with a fail rate lower than five percent. Please remember that this data set is of very bad quality, with only four non-optimally placed cameras, and basic silhouette extraction. Considering this fact, the results are rather good.

13.2.3 Output Pose Samples

In order to show the quality of the output poses, the large amount of poses supported by our method and its adaptability, we recorded several outputs of our application for the different data sets, and present them in Figure 13.8.

13.2.4 Discussion

The choice of the grid resolution will depend on the application: a low grid resolution like 50^3 is suited for a fast computation, but less accurate pose estimations (see Figure 13.9) and higher rate of false positives. On the other hand, high resolution like 150^3 provides accurate good matchings and few false positives, but does not always allow a real-time use.

In our opinion, a 100^3 grid resolution is a good compromise (at least for the studied data sets), the accuracy being sufficient and the matching rates being close to those obtained with higher resolutions. Furthermore, this resolution allows a real-time use, even in the worst observed case.

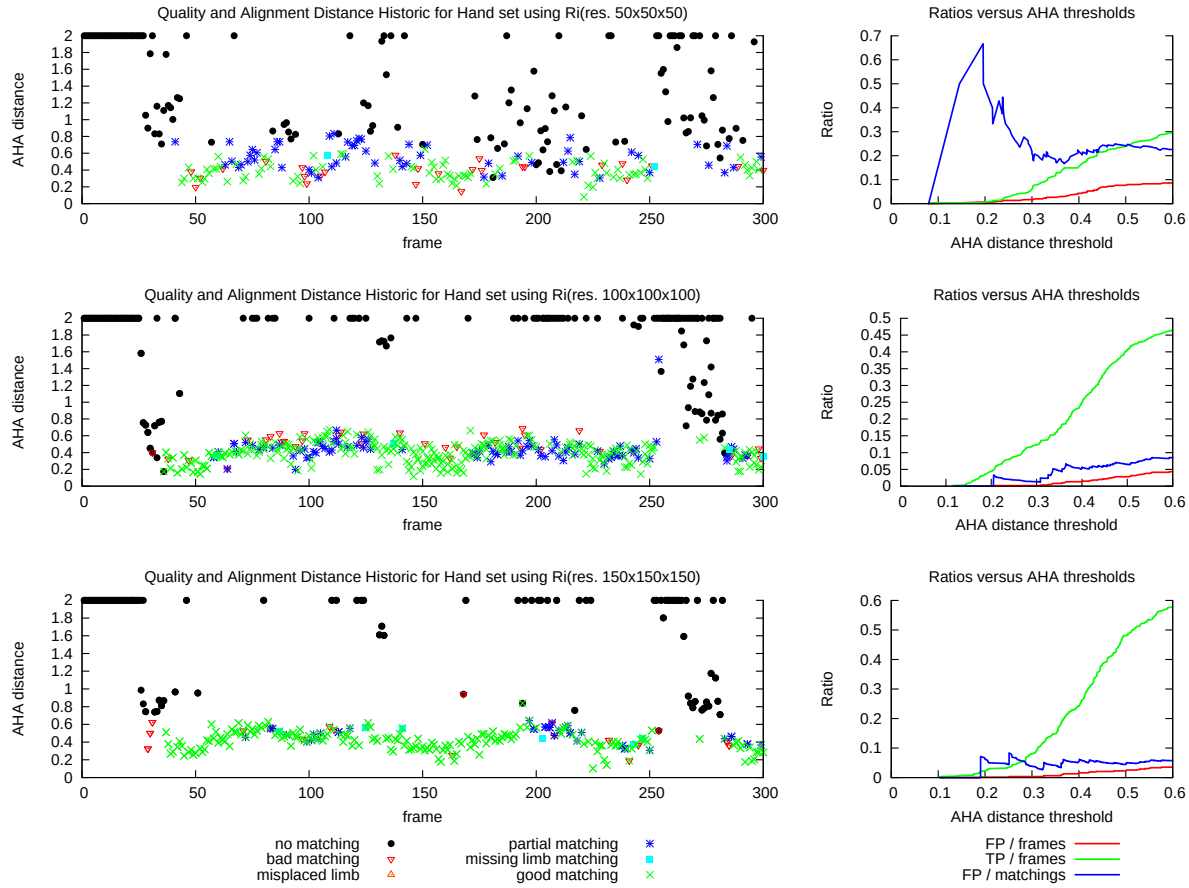


FIGURE 13.7: Study of correlations between AHA distance and matching quality for hand data set with different grid resolutions, using betweenness relation R_i. Left: historic of AHA distances and matching qualities all along the sequence. Right: different ratios in regard of given AHA distance threshold. From top to bottom: for resolutions 50^3, 100^3 and 150^3.

Concerning the model definition, we encourage the use of constraints whenever possible, in order to obtain higher rate of true positives.

13.3 Possible Usages of our Method

Our method can be used in many different ways. In this section, we propose some of the most natural of them.

13.3.1 Combination with Tracking

The more natural usage is obviously to associate it with a tracking method, in order to obtain a full motion capture application.

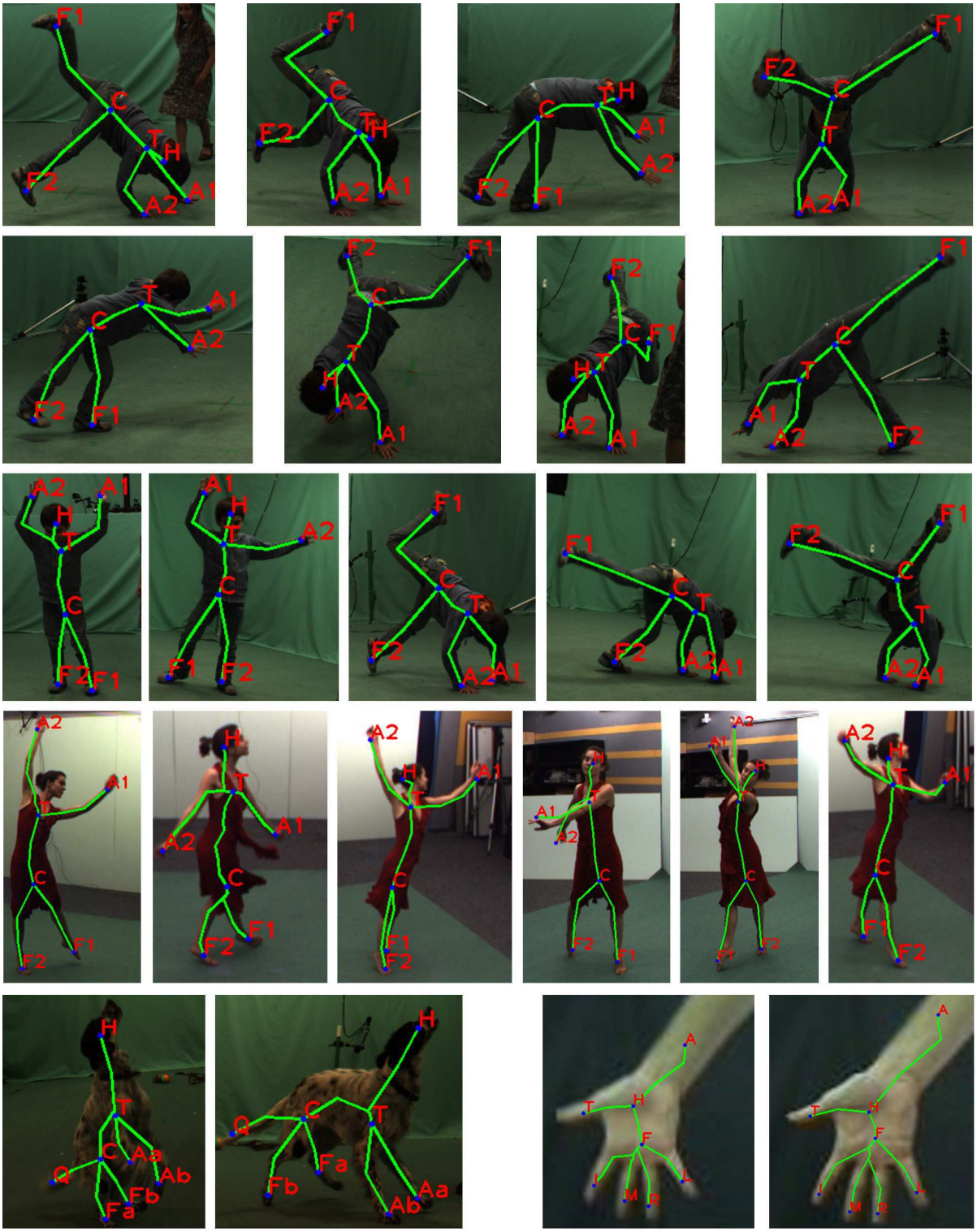


FIGURE 13.8: Samples of output poses for different models. Model parts positions are represented in blue, with the associated label in red. Approximations of skeleton branches between the parts are drawn green.

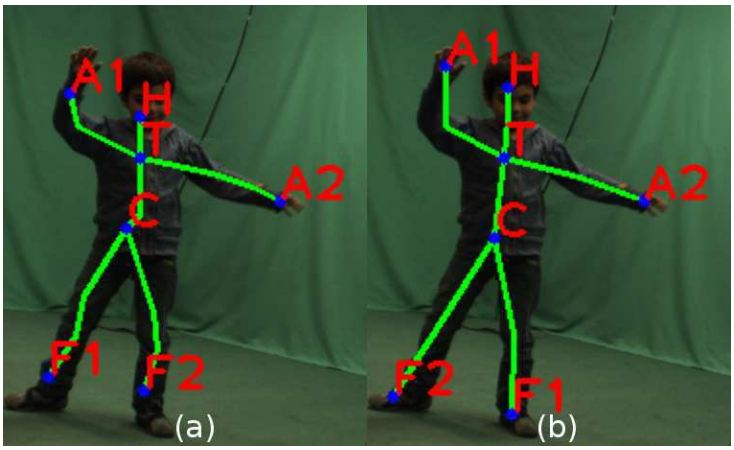


FIGURE 13.9: Sample of accuracy difference of pose estimation for different grid resolutions: (a) $50 \times 50 \times 50$ resolution, (b) $150 \times 150 \times 150$ resolution.

13.3.1.1 Combination with Model Free Tracking

An easy way to use our method with tracking method is to associate it with model-free tracking, using for example set of points matching, like those proposed by Mateus et al. [104, 105]. As those methods do not require any kind of a priori model, the goal of our method will be to perform a body part segmentation of the visual hull.

A simple method to obtain an efficient part labeling of each voxel from our method outputs (i.e. skeleton with matched specific points) consists of two steps:

- color labeling each part of the skeleton bounded by matched points (e.g. voxels of the skeleton branch linking head and torso are colored in red, those of the branch linking the torso and the crotch are colored in cyan, etc).
- flooding the visual hull simultaneously with all the color labeled voxels.

Results obtained with this method are shown in Figure 13.10.

13.3.1.2 Combination with Model-Based Tracking

Model-based tracking methods found in literature requires specific models, e.g. kinematic models or geometric shapes, more complex than ours. However, we can use information provided by our method, in order to initialize them easily.

In the case of kinematic model, our method's output can be used as a guide to provide both position and scale for the kinematic model. A way to do it is to set positions of kinematic model extremities to those of our output, then to perform a minimization step in order to find the pose as close as possible to the skeleton.

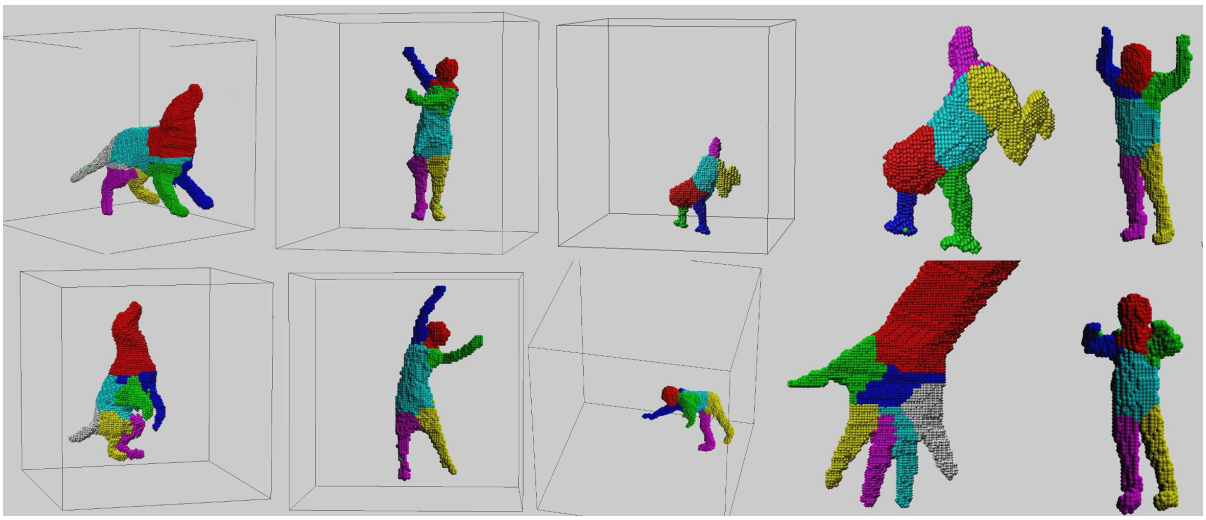


FIGURE 13.10: Examples of visual hull segmentations based on our method.

In the case of geometric shape-based model, we can use the above method, then use a fitting process in order to find parameters for each geometric shape, being the most adapted to the subject.

We can also use our method in order to provide color information on the different parts of the subject, e.g. using the segmentation method proposed in previous section, and looking for the color associated to these parts by projecting them in the input images. This color information can be used to easily track the motion of the subject, as proposed in [31, 35, 76].

13.3.2 Specific Pose Detection and Pose Clustering

Another possible way to use our method is to detect some specific poses, or to cluster similar poses.

13.3.2.1 Specific Pose Detection

For a pose detection purpose, we can describe some poses combining the syntax used to describe constraints and boolean operators, e.g.:

- for a “hands up” pose, it will consist in searching if both hands are above the head, and the description is "$(z - greatest\ A_1 - H)\ AND\ (z - greatest\ A_2 - H)$".
- for an “ask a question” pose, it will consist in searching if one and only one hand is above the head, and the description is "$(z - greatest\ A_1 - H)\ XOR\ (z - greatest\ A_2 - H)$".

- for a “wide spread arms” pose, it will consist in searching if head, torso and crotch are between the two arms, and the description is "$(order\ A_1 - H - A_2)\ AND\ (order\ A_1 - T - A_2)\ AND\ (order\ A_1 - C - A_2)$".
- for an “acrobatic” pose, it will consist in searching if at least one of the feet is above the head, and the description is "$(z - greatest\ F_1 - H)\ OR\ (z - greatest\ F_2 - H)$".

Thus, the current pose is detected as correct if the positions of the parts fit with the description. The good point of this method is the description syntax, which allow the detection of very specific poses, as well as pose of only some parts of the subject.

13.3.2.2 Pose Clustering

For a pose clustering purpose, consisting in regrouping frames where subject is in similar poses, our method can be used as follows: for each frame where an output pose is provided, we compute the set of all betweenness relations between three parts of the subject. Two poses are considered similar if their betweenness relation sets are similar enough (it can be measured using the cardinality of their intersection over the the cardinality of their union).

The two applications described above are interesting as they propose a high level description of the required pose, via understandable syntax (in case of specific pose detection) or an example (in case of pose clustering).

Part V

Conclusion

Chapter 14

Conclusion

In this thesis, we have provided an efficient real-time generic pose estimation step for multi-view marker-free motion capture, with multiple possible applications.

14.1 Contributions

Our contributions have been done in different fields of research:

Digital Topology. In this field, our contributions are:

- An efficient method to optimize the speed of critical kernel-based thinning algorithms, by providing a reformulation of the detection of crucial points, allowing the use of look-up tables.
- An easy-to-implement method to optimize the speed of any thinning algorithm using look-up tables, by reducing the time requested to compute the configuration of the neighborhood of a point.
- A new 6-directional thinning scheme based on isthmuses, speed efficient and providing good quality results.
  We submitted a paper about this method at the *16th IAPR International Conference on Discrete Geometry for Computer Imagery*.

Graph Theory. In this field, our contributions are:

- The *Homeomorphic Alignment*, a new alignment between weighted graphs, based on homeomorphism notion, instead of isomorphism one. We also proposed efficient algorithms computing it for weighted trees, rooted as well as unrooted.
  We presented a paper about Homeomorphic Alignment at the *7th IAPR Workshop on Graph-based Representations in Pattern Recognition* [143], and an extended version has been published in *Pattern Recognition* [144].

- The *Asymmetric Homeomorphism*, a new binary relation on homeomorphic graph sets.
- The *Asymmetric Homeomorphic Alignment*, a new alignment between weighted graphs, based on our new binary relation. We also proposed efficient algorithms computing it for weighted trees, rooted as well as unrooted.

Motion Capture. In this field, our contributions are:

- The introduction of new constraints in a priori model definitions, increasing the robustness of motion capture methods.
- The formalization of betweenness for 3D points, and the proposition and evaluation of several definitions of betweenness relations.
- The definition of a new method for pose estimation step for multi-view marker-free motion capture, usable in real-time context, adaptable to different kinds of subject, and not constraining for the user.
  We presented a paper about this method at the *International Conference on Image Analysis and Recognition 2010* [145].

Papers about our unpublished contributions are in redaction.

14.2 Future Works

The contributions of this thesis can be extended in different ways.

First, concerning motion capture applications, we want now focus on tracking methods, by providing new solutions using our knowledge in digital topology and mathematical morphology. It can also be interesting to develop new methods for pose detection and pose clustering using our method.

Concerning the digital topology field of research, our aim is now to provide new thinning schemes adapted to CPU-parallelization, and to search for new parameterizable constraints more powerful than isthmuses.

Finally, concerning our graph theory contributions, we will try to design efficient algorithms for the computation of both Homeomorphic Alignment and Asymmetric Homeomorphic Alignment between any kinds of graph.

"The hardest thing is to go to sleep at night, when there are so many urgent things needing to be done. A huge gap exists between what we know is possible with today's machines and what we have so far been able to finish."

Donald Knuth.

Bibliography

[1] A. Agarwal and B. Triggs. 3D human pose from silhouettes by relevance vector regression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2004.

[2] H. Aghajan, C. Wu, and R. Kleihorst. Distributed Vision Networks for Human Pose Analysis. *Signal Processing Techniques for Knowledge Extraction and Information Fusion*, pages 181–200, 2008.

[3] E. De Aguiar, C. Stoll, C. Theobalt, N. Ahmed, H.P. Seidel, and S. Thrun. Performance capture from sparse multi-view video. *ACM Transactions on Graphics (TOG)*, 27(3):98, 2008.

[4] A.V. Aho, R. Sethi, and J.D. Ullman. Compilers: principles, techniques, and tools. *Reading, MA,*, 1986.

[5] N. Ahuja and J-H. Chuang. Shape representation using a generalized potential field model. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(2):169–176, 1997.

[6] T.M. Alisi, A. Del Bimbo, F. Pucci, and A. Valli. Motion Capture Based on Color Error Maps in a Distributed Collaborative Environment. *Pattern Recognition*, 4:953–956, 2004.

[7] M. Ankerst, G. Kastenmüller, H.P. Kriegel, and T. Seidl. 3D shape histograms for similarity search and classification in spatial databases. In *Advances in Spatial Databases*, pages 207–226. Springer, 1999.

[8] C. Arcelli. Pattern thinning by contour tracing. *Computer Graphics and Image Processing*, 17(2):130 – 144, 1981.

[9] V. Athitsos and S. Sclaroff. Estimating 3D hand pose from a cluttered image. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2003.

[10] X. Bai and L.J. Latecki. Path similarity skeleton graph matching. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(7):1282–1292, 2008.

[11] L. Ballan and G. Cortelazzo. Marker-less motion capture of skinned models in a four camera set-up using optical flow and silhouettes. *Proceedings of International Symposium on 3D Data Processing, Visualization and Transmission*, 2008.

[12] M. Bartoli, M. Pelillo, K. Siddiqi, and S. Zucker. Attributed tree homomorphism using association graphs. In *INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION*, volume 15, pages 133–136. Citeseer, 2000.

[13] G. Bertrand. On p-simple points. *Comptes rendus de l'Académie des Sciences, Série Math*, I(321):1077–1084, 1995.

[14] G. Bertrand. Sufficient conditions for 3D parallel thinning algorithms. In R. A. Melter, A. Y. Wu, F. L. Bookstein, & W. D. Green, editor, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 2573 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 52–60, August 1995.

[15] G. Bertrand. A boolean characterization of three-dimensional simple points. *Pattern Recognition Letters*, 17(2):115–124, 1996.

[16] G. Bertrand. On critical kernels. *Comptes Rendus de l'Académie des Sciences, Série Math.*, I(345):363–367, 2007.

[17] G. Bertrand and M. Couprie. A new 3d parallel thinning scheme based on critical kernels. In *Discrete Geometry for Computer Imagery*, pages 580–591. Springer, 2006.

[18] G. Bertrand and M. Couprie. Transformations topologiques discrètes. In David Coeurjolly, Annick Montanvert, and Jean-Marc Chassery, editors, *Géométrie discrète et images numériques*, pages 187–209. Hermès, 2007.

[19] G. Bertrand and G. Malandain. A new characterization of three-dimensional simple points. *Pattern Recognition Letters*, 15(2):169–175, 1994.

[20] P. Bille. Ordered Tree Edit Distance with Merge and Split Operations. Technical report, IT University of Copenhagen, 2003.

[21] P. Bille. A survey on tree edit distance and related problems. *Theoretical computer science*, 337(1-3):217–239, 2005.

[22] H. Blum. An associative machine for dealing with the visual field and some of its biological implications. In E.E. Bernard and M.R. Kare, editors, *Biological Prototypes and Synthetic Systems*, volume 1, pages 244–260, NY, 1962. Plenum Press.

[23] G. Borgefors, I. Nystrm, and G. Sanniti Di Baja. Computing skeletons in three dimensions. *Pattern Recognition*, 32(7):1225 – 1236, 1999.

[24] K.M. Borgwardt and H.P. Kriegel. Shortest-path kernels on graphs. 2005.

[25] A.E. Brain, R.O. Duda, and J.H. Munson. Graphical data processing research study and experimental investigation. Technical report, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, Aug 1965.

[26] Jonathan W. Brandt and V. Ralph Algazi. Continuous skeleton computation by voronoi diagram. *CVGIP: Image Underst.*, 55(3):329–338, 1992.

[27] G. Brostow, I. Essa, D. Steedly, and V. Kwatra. Novel skeletal representation for articulated creatures. *Proceedings of the European Conference on Computer Vision*, pages 66–78, 2004.

[28] L. Brun and J.H. Pruvot. Hierarchical Matching Using Combinatorial Pyramid Framework. *Image and Signal Processing*, pages 346–355, 2008.

[29] R.E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *Relation*, 10(1.104):3755, 1986.

[30] H. Bunke and U. Bühler. Applications of approximate string matching to 2D shape recognition. *Pattern recognition*, 26(12):1797–1812, 1993.

[31] F. Caillette and T. Howard. Real-time markerless human body tracking with multi-view 3-d voxel reconstruction. In *Proceedings of the British Machine Vision Conference*, volume 2, pages 597–606, 2004.

[32] L. Calabi. A study of the skeleton of plane figures. Technical Report Technical Report 60429, Parke Mathematical Laboratories, 1965.

[33] W. Chen. New algorithm for ordered tree-to-tree correction problem. *Journal of Algorithms*, 40(2):135–158, 2001.

[34] Y.C. Cheng and S.Y. Lu. Waveform correlation by tree matching. *IEEE transactions on pattern analysis and machine intelligence*, 7(3):299–305, 1985.

[35] K.M.G. Cheung, S. Baker, and T. Kanade. Shape-from-silhouette of articulated objects and its use for human body kinematics estimation and motion capture. In *Proceedings of the Conference on Computer Vision and Pattern Recognition*, volume 1, 2003.

[36] C.W. Chu, O.C. Jenkins, and M.J. Mataric. Markerless kinematic model and motion capture from volume sequences. 2003.

[37] M.J. Chung. $O(n^{2.5})$ time algorithms for the subgraph homeomorphism problem on trees. *Journal of Algorithms*, 8(1):106–112, 1987.

[38] M. Couprie. Note on fifteen 2d parallel thinning algorithms. In *Internal Report, Universit de Marne-laValle*, 2006.

[39] M. Couprie and G. Bertrand. New characterizations of simple points, minimal non-simple sets and p-simple points in 2d, 3d and 4d discrete spaces. In *DGCI'08: Proceedings of the 14th IAPR international conference on Discrete geometry for computer imagery*, pages 105–116, Berlin, Heidelberg, 2008. Springer-Verlag.

[40] M. Couprie and G. Bertrand. New characterizations of simple points in 2d, 3d, and 4d discrete spaces. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(4):637–648, 2009.

[41] M. Couprie, D. Coeurjolly, and R. Zrour. Discrete bisector function and euclidean skeleton in 2d and 3d. *Image Vision Comput.*, 25(10):1543–1556, 2007.

[42] F. Cuzzolin, D. Mateus, E. Boyer, and R. Horaud. Robust spectral 3D-bodypart segmentation along time. *Human Motion–Understanding, Modeling, Capture and Animation*, pages 196–211, 2007.

[43] F. Cuzzolin, D. Mateus, D. Knossow, E. Boyer, and R. Horaud. Coherent Laplacian 3-D protrusion segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2008.

[44] E.R. Davies and A.P.N. Plummer. Thinning algorithms: A critique and a new methodology. *Pattern Recognition*, 14(1-6):53 – 63, 1981.

[45] Q. Delamarre and O. Faugeras. 3D Articulated Models and Multiview Tracking with Physical Forces* 1. *Computer Vision and Image Understanding*, 81(3):328–357, 2001.

[46] E.D. Demaine, S. Mozes, B. Rossman, and O. Weimann. An O (nˆ 3)-Time Algorithm for Tree Edit Distance. *Arxiv preprint cs/0604037*, 2006.

[47] D.G.Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[48] S. Drouin, P. Hebert, and M. Parizeau. Incremental discovery of object parts in video sequences. *Computer vision and image understanding*, 110(1):60–74, 2008.

[49] P. Ferraro and C. Godin. Optimal mappings with minimum number of connected components in tree-to-tree comparison problems. *Journal of Algorithms*, 48(2):385–406, 2003.

[50] J. Gall, B. Rosenhahn, T. Brox, and H.P. Seidel. Optimization and filtering for human motion capture. *International journal of computer vision*, 87(1):75–92, 2010.

[51] J. Gall, C. Stoll, E. de Aguiar, C. Theobalt, B. Rosenhahn, and H.P. Seidel. Motion capture using joint skeleton tracking and surface estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. IEEE, 2009.

[52] T. Gärtner. Exponential and geometric kernels for graphs. In *NIPS Workshop on Unreal Data: Principles of Modeling Nonvectorial Data*, 2002.

[53] C. J. Gau and T.Y. Kong. Minimal non-simple sets in 4d binary images. *Graph. Models*, 65(1-3):112–130, 2003.

[54] Y. Gdalyahu and D. Weinshall. Flexible syntactic matching of curves and its application to automatic hierarchical classification of silhouettes. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(12):1312–1328, 2002.

[55] W.B. Goh. Strategies for shape matching using skeletons. *Computer Vision and Image Understanding*, 110(3):326–345, 2008.

[56] M.J.E. Golay. Hexagonal parallel pattern transformations. *IEEE Trans. Comput.*, 18(8):733–740, 1969.

[57] W. Gong and G. Bertrand. A simple parallel 3D thinning algorithm. In *10th International Conference on Pattern Recognition, 1990. Proceedings.*, pages 188–190, 1990.

[58] K. Grauman, G. Shakhnarovich, and T. Darrell. Inferring 3d structure with a statistical image-based shape model. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, pages 641–647, 2003.

[59] M.J. Greenburg. Euclidean and non-Euclidean geometries: Development and history. *New York: WH Freemand and Compnay*, 1993.

[60] F.C.A. Groen and N.J. Foster. A fast algorithm for cellular logic operations on sequential machines. *Pattern Recognition Letters*, 2(5):333 – 338, 1984.

[61] R.W. Hall. Tests for connectivity preservation for parallel reduction operators. *Topology and its Applications*, 46(3):199 – 217, 1992.

[62] R.W. Hall. Parallel connectivity-preserving thinning algorithms. *Machine Intelligence and Pattern Recognition*, 19:145–179, 1996.

[63] J.M. Hasenfratz, M. Lapierre, J.D. Gascuel, and E. Boyer. Real-time capture, reconstruction and insertion into virtual world of human actors. In *Vision, Video and Graphics*, pages 49–56, 2003.

[64] C.J. Hilditch. Linear skeletons from square cupboards. *Machine Intelligence*, 4:403–420, 1969.

[65] M. Höchsmann. The tree alignment model: algorithms, implementations and applications for the analysis of RNA secondary structures. 2005.

[66] R. Horaud, M. Niskanen, G. Dewaele, and E. Boyer. Human motion tracking by registering an articulated surface to 3D points and normals. *IEEE transactions on pattern analysis and machine intelligence*, 31(1):158–163, 2009.

[67] E.V. Huntington. A new set of postulates for betweenness, with proof of complete independence. *Transactions*, 18:301–325, 1917.

[68] E.V. Huntington and J.R. Kline. Sets of independent postulates for betweenness. *Transactions of the American Mathematical Society*, 18:301–325, 1917.

[69] C.Y. Ip, D. Lapadat, L. Sieger, and W.C. Regli. Using shape distributions to compare solid models. In *Proceedings of the seventh ACM symposium on Solid modeling and applications*, pages 273–280. ACM, 2002.

[70] J. Jansson and A. Lingas. A Fast Algorithm for Optimal Alignment between Similar Ordered Trees. In *Combinatorial Pattern Matching*, pages 232–240. Springer, 2001.

[71] T. Jiang, L. Wang, and K. Zhang. Alignment of trees - an alternative to tree edit. In *CPM '94: Proceedings of the 5th Annual Symposium on Combinatorial Pattern Matching*, pages 75–86, London, UK, 1994. Springer-Verlag.

[72] P. Kardos, G. Németh, and K. Palágyi. Topology preserving 2-subfield 3d thinning algorithms. In *Proceedings of the IASTED International Conference on Signal Processing, Pattern Recognition and Applications*, pages 310–316, Innsbruck, Austria, February 2010. IASTED.

[73] H. Kashima, K. Tsuda, and A. Inokuchi. Marginalized kernels between labeled graphs. In *Machine Learning, Proceedings of the Twentieth International Conference (ICML 2003)*, volume 20, pages 321–328, 2003.

[74] S. Kaygin and M.M. Bulut. Shape recognition using attributed string matching with polygon vertices as the primitives. *Pattern Recognition Letters*, 23(1-3):287–294, 2002.

[75] M. Kazhdan, T. Funkhouser, and S. Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *Proceedings of the 2003 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 156–164. Eurographics Association, 2003.

[76] R. Kehl, M. Bray, and L. Van Gool. Full body tracking from multiple views using stochastic sampling. In *Proceedings of theIEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 129–136, 2005.

[77] P. Klein. Computing the edit-distance between unrooted ordered trees. *AlgorithmsESA98*, pages 1–1.

[78] P. Klein, S. Tirthapura, D. Sharvit, and B. Kimia. A tree-edit-distance algorithm for comparing simple, closed shapes. In *Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 696–704. Society for Industrial and Applied Mathematics, 2000.

[79] D. Knossow, R. Ronfard, and R. Horaud. Human motion tracking with a kinematic parameterization of extremal contours. *International Journal of Computer Vision*, 79(3):247–269, 2008.

[80] D. Knossow, R. Ronfard, R. Horaud, and F. Devernay. Tracking with the kinematics of extremal contours. *Proceedings of the Asian Conference on Computer Vision*, pages 664–673, 2006.

[81] D. Knossow, J. Van De Weijer, R. Horaud, and R. Ronfard. Articulated-Body Tracking Through Anisotropic Edge Detection. *Dynamical Vision*, pages 86–99, 2007.

[82] T. Y. Kong and A. Rosenfeld. Digital topology: introduction and survey. *Comp. Vision, Graphics and Image Proc.*, 48:357–393, 1989.

[83] T.Y. Kong. Problem of determining whether a parallel reduction operator for n-dimensional binary images always preserves topology. In R. A. Melter & A. Y. Wu, editor, *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, volume 2060 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, pages 69–77, December 1993.

[84] T.Y. Kong. On topology preservation in 2-d and 3-d thinning. *International journal of pattern recognition and artificial intelligence*, 9:813–844, 1995.

[85] T.Y. Kong. Topology-preserving deletion of 1's from 2-, 3- and 4-dimensional binary images. In *DGCI '97: Proceedings of the 7th International Workshop on Discrete Geometry for Computer Imagery*, pages 3–18, London, UK, 1997. Springer-Verlag.

[86] T.Y. Kong and C.J. Gau. Minimal non-simple sets in 4-dimensional binary images with (8,80)-adjacency. In *Lecture notes in computer science*, pages 318–333, 2004.

[87] S. Kosinov and T. Caelli. Inexact Multisubgraph matching using Graph Eigenspace and Clustering Models. In *Proceedings of SSPR/SPR*, pages 133–142. Citeseer, 2002.

[88] L. Lam, S.W. Lee, and C.Y. Suen. Thinning methodologies-a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14:869–885, 1992.

[89] A. Laurentini. The visual hull concept for silhouette-based image understanding. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 150–162, 1994.

[90] T.C. Lee, R.L. Kashyap, and C.N. Chu. Building skeleton models via 3-D medial surface/axis thinning algorithms. *Graphical Models and Image Processing*, 56(6):462–478, 1994.

[91] A. Lieutier. Any open bounded subset of $\setminus^n$ has the same homotopy type as its medial axis. In *Proc. 8th ACM Symposium on Solid Modeling Applications*, pages 65–75. Academic Press, 2003.

[92] C. Lohou and G. Bertrand. A 3d 12-subiteration thinning algorithm based on p-simple points. *Discrete Appl. Math.*, 139(1-3):171–195, 2004.

[93] C. Lohou and G. Bertrand. A 3d 6-subiteration curve thinning algorithm based on p-simple points. *Discrete Appl. Math.*, 151(1-3):198–228, 2005.

[94] C. Lu, Z.Y. Su, and C. Tang. A new measure of edit distance between labeled trees. *Computing and Combinatorics*, pages 338–348.

[95] C.M. Ma. On topology preservation in 3d thinning. *CVGIP: Image Understanding*, 59(3):328 – 339, 1994.

[96] C.M. Ma and S.Y. Wan. Parallel thinning algorithms on 3D (18, 6) binary images. *Computer Vision and Image Understanding*, 80(3):364–378, 2000.

[97] C.M. Ma and S.Y. Wan. A medial-surface oriented 3-d two-subfield thinning algorithm. *Pattern Recognition Letters*, 22(13):1439–1446, 2001.

[98] C.M. Ma, S.Y. Wan, and H.K. Chang. Extracting medial curves on 3D images. *Pattern Recognition Letters*, 23(8):895–904, 2002.

[99] C.M. Ma, S.Y. Wan, et al. Three-dimensional topology preserving reduction on the 4-subfields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pages 1594–1605, 2002.

[100] P. Mahé, N. Ueda, T. Akutsu, J.L. Perret, and J.P. Vert. Extensions of marginalized graph kernels. In *Proceedings of the twenty-first international conference on Machine learning*, page 70. ACM, 2004.

[101] P. Mahé and J.P. Vert. Graph kernels based on tree patterns for molecules. *Machine learning*, 75(1):3–35, 2009.

[102] G. Malandain and G. Bertrand. Fast characterization of 3D simple points. In *IAPR 11th International Conference on Pattern Recognition*, The Hague, The Netherlands, August 30 – September 3 1992.

[103] G. Malandain and S. Fernndez-Vidal. Euclidean skeletons. *Image and Vision Computing*, 16(5):317 – 327, 1998.

[104] D. Mateus, F. Cuzzolin, R. Horaud, and E. Boyer. Articulated shape matching using locally linear embedding and orthogonal alignment. In *Proceedings of the International Conference on Computer Vision*, 2007.

[105] D. Mateus, R. Horaud, D. Knossow, F. Cuzzolin, and E. Boyer. Articulated shape matching using laplacian eigenfunctions and unsupervised point registration. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2008.

[106] G. Matheron. *Examples of topological properties of skeletons*, volume 2, pages 217–238. Academic Press, 1988.

[107] D.W. Matula. An algorithm for subtree identification. *SIAM Rev*, 10:273–274, 1968.

[108] D.W. Matula. Subtree Isomorphism in $0(n^{5/2})$. *Algorithmic aspects of combinatorics*, page 91, 1978.

[109] W. Matusik, C. Buehler, R. Raskar, S.J. Gortler, and L. McMillan. Image-based visual hulls. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, pages 369–374. ACM Press/Addison-Wesley Publishing Co., 2000.

[110] C. Menier. *Système de vision temps-réel pour les interactions*. PhD thesis, Institut National Polytechnique de Grenoble, 2007.

[111] C. Menier, E. Boyer, and B. Raffin. 3d skeleton-based body pose recovery. In *Proceedings of the 3rd International Symposium on 3D Data Processing, Visualization and Transmission*. Citeseer, 2006.

[112] B. Michoud, E. Guillou, H. Briceño, and S. Bouakaz. Real-time and marker-free 3D motion capture for home entertainment oriented applications. In *Proceedings of the 8th Asian Conference on Computer vision*, pages 678–687. Springer-Verlag, 2007.

[113] B. Michoud, E. Guillou, H. Briceno, and S. Bouakaz. Real-Time Marker-free Motion Capture from multiple cameras. In *Proceedings of the IEEE 11th International Conference on Computer Vision*, pages 1–7, 2007.

[114] I. Mikić, M. Trivedi, E. Hunter, and P. Cosman. Articulated body posture estimation from multi-camera voxel data. In *IEEE Conference on Computer Vision and Pattern Recognition*, 2001.

[115] I. Mikić, M. Trivedi, E. Hunter, and P. Cosman. Human body model acquisition and tracking using voxel data. *International Journal of Computer Vision*, 53(3):199–223, 2003.

[116] T.B. Moeslund and E. Granum. A survey of computer vision-based human motion capture. *Computer Vision and Image Understanding*, 81(3):231–268, 2001.

[117] T.B. Moeslund, A. Hilton, and V. Krüger. A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding*, 104(2-3):90–126, 2006.

[118] D.G. Morgenthaler. Three-dimensional simple points: serial erosion, parallel thinning and skeletonization. Technical Report TR-1005, Computer Vision Laboratory, Computer Science Center, University of Maryland, College Park, 1981.

[119] G. Mori and J. Malik. Estimating human body configurations using shape context matching. *Computer VisionECCV 2002*, pages 150–180, 2002.

[120] J. Mukherjee, P.P. Das, and B.N. Chatterji. On connectivity issues of ESPTA. *Pattern Recognition Letters*, 11(9):643–648, 1990.

[121] R. Mukundan and K.R. Ramakrishnan. *Moment functions in image analysis: theory and applications*. World Scientific Pub Co Inc, 1998.

[122] M. Näf, G. Székely, R. Kikinis, M. E. Shenton, and O. Kübler. 3d voronoi skeletons and their usage for the characterization and recognition of 3d organ shape. *Comput. Vis. Image Underst.*, 66(2):147–161, 1997.

[123] G. Németh, P. Kardos, and K. Palágyi. Topology Preserving 3D Thinning Algorithms Using Four and Eight Subfields. *Image Analysis and Recognition*, pages 316–325, 2010.

[124] M. Novotni and R. Klein. Shape retrieval using 3D Zernike descriptors. *Computer-Aided Design*, 36(11):1047–1062, 2004.

[125] R. Osada, T. Funkhouser, B. Chazelle, and D. Dobkin. Shape distributions. *ACM Transactions on Graphics (TOG)*, 21(4):807–832, 2002.

[126] K. Palágyi. A 3-subiteration 3D thinning algorithm for extracting medial surfaces. *Pattern Recognition Letters*, 23(6):663–675, 2002.

[127] K. Palágyi. A 3–Subiteration Surface–Thinning Algorithm. In *Computer Analysis of Images and Patterns*, pages 628–635. Springer, 2007.

[128] K. Palágyi. A subiteration-based surface-thinning algorithm with a period of three. *Pattern Recognition*, pages 294–303, 2007.

[129] K. Palágyi. A 3d fully parallel surface-thinning algorithm. *Theor. Comput. Sci.*, 406(1-2):119–135, 2008.

[130] K. Palágyi and A. Kuba. A 3D 6-subiteration thinning algorithm for extracting medial lines. *Pattern Recognition Letters*, 19(7):613–627, 1998.

[131] K. Palágyi and A. Kuba. A parallel 3D 12-subiteration thinning algorithm. *Graphical Models and Image Processing*, 61(4):199–221, 1999.

[132] K. Palágyi and A. Kuba. Directional 3D thinning using 8 subiterations. In *Discrete Geometry for Computer Imagery*, pages 325–336. Springer, 1999.

[133] K. Palágyi and G. Németh. Fully parallel 3D thinning algorithms based on sufficient conditions for topology preservation. In *Proceedings of the 15th IAPR international conference on Discrete geometry for computer imagery*, page 481492. Springer-Verlag, 2009.

[134] T. Pavlidis. A thinning algorithm for discrete binary images. *Computer Graphics and Image Processing*, 13(2):142 – 157, 1980.

[135] T. Pavlidis. A flexible parallel thinning algorithm. In *Pattern Recognition and Image Processing*, pages 162–167, 1981.

[136] M. Pelillo. Matching free trees, maximal cliques, and monotone game dynamics. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 24(11):1535–1541, 2002.

[137] M. Pelillo, K. Siddiqi, and S. Zucker. Many-to-many matching of attributed trees using association graphs and game dynamics. *Visual Form 2001*, pages 583–593, 2001.

[138] M. Pelillo, K. Siddiqi, and S.W. Zucker. Matching hierarchical structures using association graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 21(11):1105–1120, 1999.

[139] R. Plänkers and P. Fua. Articulated soft objects for video-based body modeling. In *International Conference on Computer Vision*, volume 1, page 394, 2001.

[140] R. Poppe. Vision-based human motion analysis: An overview. *Computer Vision and Image Understanding*, 108(1-2):4–18, 2007.

[141] C. Pudney. Distance-ordered homotopic thinning: A skeletonization algorithm for 3d digital images. *Computer Vision and Image Understanding*, 72(3):404 – 413, 1998.

[142] J. Ramon and T. Gärtner. Expressivity versus efficiency of graph kernels. In *First International Workshop on Mining Graphs, Trees and Sequences*, pages 65–74, 2003.

[143] B. Raynal, M. Couprie, and V. Biri. Homeomorphic alignment of edge-weighted trees. In *Graph-Based Representations in Pattern Recognition*, pages 134–143. Springer, may 2009.

[144] B. Raynal, M. Couprie, and V. Biri. Homeomorphic alignment of weighted trees. *Pattern Recognition*, 43(8):2937–2949, 2010.

[145] B. Raynal, M. Couprie, and V. Nozick. Generic initialization for motion capture from 3d shape. In *Proceedings of the International Conference in Image Analysis and Recognition*, pages 306–315. Springer, june 2010.

[146] C. Ronse. Minimal test patterns for connectivity preservation in parallel thinning algorithms for binary digital images. *Discrete Appl. Math.*, 21(1):67–79, 1988.

[147] A. Rosenfeld. Connectivity in digital pictures. *Journal of the ACM*, 17(1):146–160, January 1970.

[148] C. Di Ruberto. Recognition of shapes by attributed skeletal graphs. *Pattern Recognition*, 37(1):21–31, 2004.

[149] M. Rumpf and A. Telea. A continuous skeletonization method based on level sets. In *VISSYM '02: Proceedings of the symposium on Data Visualisation 2002*, pages 151–ff, Aire-la-Ville, Switzerland, Switzerland, 2002. Eurographics Association.

[150] D. Rutovitz. Pattern recognition. *Journal of the Royal Statistical Society. Series A (General)*, 129(4):504–530, 1966.

[151] P.K. Saha and B.B. Chaudhuri. Detection of 3-d simple points for topology preserving transformations with application to thinning. *IEEE Trans. Pattern Anal. Mach. Intell.*, 16(10):1028–1032, 1994.

[152] P.K. Saha, B.B. Chaudhuri, B. Chanda, and D.D. Majumder. Topology preservation in 3d digital space. *Pattern Recognition*, 27(2):295–300, 1994.

[153] H. Samet. Distance Transform for Images Represented by Quadtrees. *IEEE TRANS. PATTERN ANALY. AND MACH. INTELLIG.*, 4(3):298–303, 1982.

[154] T. Schlieder and F. Naumann. Approximate tree embedding for querying XML data. In *ACM SIGIR workshop on XML and information retrieval, Athens, Greece*. Citeseer, 2000.

[155] S.M. Selkow. The tree-to-tree editing problem. *Information processing letters*, 6(6):184–186, 1977.

[156] G. Shakhnarovich, P. Viola, and T. Darrell. Fast pose estimation with parameter-sensitive hashing. 2003.

[157] R. Shamir and D. Tsur. Faster subtree isomorphism. *Journal of Algorithms*, 33(2):267–280, 1999.

[158] B.A. Shapiro and K. Zhang. Comparing multiple RNA secondary structures using tree comparisons. *Bioinformatics*, 6(4):309, 1990.

[159] J. Shen and D. Shen. Orthogonal Legendre moments and their calculation. In *Pattern Recognition, 1996., Proceedings of the 13th International Conference on*, volume 2, pages 241–245. IEEE, 2002.

[160] K. Siddiqi and B.B. Kimia. A shock grammar for recognition. In *1996 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 1996. Proceedings CVPR'96*, pages 507–513, 1996.

[161] J. Starck and A.Hilton. Spherical matching for temporal correspondence of non-rigid surfaces. 2005.

[162] R. Stefanelli and A. Rosenfeld. Some parallel thinning algorithms for digital pictures. *J. ACM*, 18(2):255–264, 1971.

[163] B. Stenger, A. Thayananthan, P.H.S. Torr, and R. Cipolla. Filtering using a tree-based estimator. In *Proceedings of 9th International Conference on Computer Vision*, volume 2, pages 1063–1070, 2003.

[164] R.E. Stobaugh. Chemical substructure searching. *Journal of Chemical Information and Computer Sciences*, 25(3):271–275, 1985.

[165] F. Suard, A. Rakotomamonjy, and A. Bensrhair. Kernel on bag of paths for measuring similarity of shapes. In *European Symposium on Artificial Neural Networks, Bruges-Belgique (April 2007)*.

[166] A. Sundaresan and R. Chellappa. Markerless motion capture using multiple cameras. *Computer Vision for Interactive and Intelligent Environment*, pages 15–26, 2005.

[167] A. Sundaresan and R. Chellappa. Multi-camera tracking of articulated human motion using motion and shape cues. *Proceedings of the Asian Conference on Computer Vision*, pages 131–140, 2006.

[168] K.C. Tai. The tree-to-tree correction problem. *Journal of the ACM (JACM)*, 26(3):433, 1979.

[169] H. Talbot. Euclidean skeletons and conditional bisectors. In *in Visual Communications and Image Processing'92*, pages 862–876, 1992.

[170] E. Tanaka and K. Tanaka. The tree-to-tree editing problem. *INT. J. PATTERN RECOG. ARTIF. INTELL.*, 2(2):221–240, 1988.

[171] H. Tanaka, A. Nakazawa, and H. Takemura. Human pose estimation from volume data and topological graph database. *Proceedings of the Asian Conference on Computer Vision*, pages 618–627, 2007.

[172] J.I. Toriwaki and K. Mori. Distance transformation and skeletonization of 3d pictures and their applications to medical images. In *Digital and Image Geometry, Advanced Lectures*, pages 412–428, London, UK, 2001. Springer-Verlag.

[173] A. Torsello and E.R. Hancock. A skeletal measure of 2D shape similarity. *Computer Vision and Image Understanding*, 95(1):1–29, 2004.

[174] R. Urtasun, D.J. Fleet, and P. Fua. Temporal motion models for monocular and multiview 3D human body tracking. *Computer vision and image understanding*, 104(2-3):157–177, 2006.

[175] G. Valiente. An efficient bottom-up distance between trees. In *String Processing and Information Retrieval, 2001. SPIRE 2001. Proceedings. Eighth International Symposium on*, pages 212–219, 2001.

[176] K. Varanasi, A. Zaharescu, E. Boyer, and R. Horaud. Temporal surface tracking using mesh evolution. *Proceedings of the European Conference on Computer Vision*, pages 30–43, 2008.

[177] S. Vishwanathan, K.M. Borgwardt, I.R. Kondor, and N.N. Schraudolph. Graph kernels. *Journal of Machine Learning Research*, 9:1–41, 2008.

[178] S.V.N. Vishwanathan and A.J. Smola. Fast kernels for string and tree matching. *Kernel methods in computational biology*, pages 113–130, 2004.

[179] J.T.L. Wang and K. Zhang. Finding similar consensus between trees: an algorithm and a distance hierarchy. *Pattern Recognition*, 34(1):127–137, 2001.

[180] J.T.L. Wang, K. Zhang, G. Chang, and D. Shasha. Finding approximate patterns in undirected acyclic graphs. *Pattern Recognition*, 35(2):473–483, 2002.

[181] H. Whitney. A set of topological invariants for graphs. *American Journal of Mathematics*, 55(1):231–235, 1933.

[182] W. Xie, R.P. Thompson, and R. Perucchio. A topology-preserving parallel 3D thinning algorithm for extracting the curve skeleton. *Pattern Recognition*, 36(7):1529–1544, 2003.

[183] W. Yang. Identifying syntactic differences between two programs. *Software - Practice and Experience*, 21(7):739–755, 1991.

[184] S. Yokoi, J-I. Toriwaki, and T. Fukumura. An analysis of topological properties of digitized binary pictures using local features. *Computer Graphics and Image Processing*, 4(1):63 – 73, 1975.

[185] K. Zhang. Algorithms for the constrained editing distance between ordered labeled trees and related problems* 1. *Pattern Recognition*, 28(3):463–474, 1995.

[186] K. Zhang. A constrained edit distance between unordered labeled trees. *Algorithmica*, 15(3):205–222, 1996.

[187] K. Zhang and D. Shasha. Simple fast algorithms for the editing distance between trees and related problems. *SIAM J. Comput.*, 18(6):1245–1262, 1989.

[188] K. Zhang, R. Statman, and D. Shasha. On the editing distance between unordered labeled trees. *Information Processing Letters*, 42(3):133–139, 1992.

[189] K. Zhang, J. Wang, and D. Shasha. On the editing distance between undirected acyclic graphs and related problems. In *Combinatorial Pattern Matching*, pages 395–407. Springer, 1995.