Measurement and synthesis of motion of plants

Julien Diener

INSTITUT POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque

THESE

pour obtenir le grade de

DOCTEUR DE L'INSTITUT POLYTECHNIQUE DE GRENOBLE

Spécialité **Mathématiques - Informatique**

Préparée au Laboratoire Jean Kuntzmann,

dans le cadre de l'**École Doctorale "Mathématiques, Sciences et Technologies de l'Information, Informatique"**

Présenté et soutenu publiquement

par

Julien DIENER

le ... juillet 2009

Acquisition et generation du mouvement de plantes

Directeur de thèse : Fabrice NEYRET
Co-directeur : Lionel REVERET

JURY

J.M. CHASSERY	Directeur de Recherche - CNRS	, Président
M. WIMMER	Associate Professor - Technische Universitat Wien	, Rapporteur
C. GODIN	Directeur de Recherche - INRIA	, Rapporteur
E. GALIN	Professeur des Universités - Université Lumière Lyon 2	, Examinateur
F. NEYRET	Directeur de Recherche - CNRS	, Directeur de thèse
L. REVERET	Chargé de Recherche - INRIA	, Co-directeur de thèse

Contents

1 Introduction **7**
- 1.1 Rationale . . . 7
  - 1.1.1 Trees . . . 7
  - 1.1.2 Tree Animation in Computer Graphics . . . 10
- 1.2 Contribution . . . 11
- 1.3 Thesis Outline . . . 13

I Acquisition and Reproduction of Real Motion **15**

2 Acquisition of Plants Motion **17**
- 2.1 Introduction . . . 18
- 2.2 Motion Capture . . . 18
  - 2.2.1 Origin of Motion Capture . . . 18
  - 2.2.2 Assisted Motion Capture . . . 20
  - 2.2.3 Markerless Motion Tracking . . . 22
- 2.3 Data Acquisition . . . 23
  - 2.3.1 Experimental Setup . . . 23
  - 2.3.2 Cameras Calibration . . . 24
  - 2.3.3 Recorded Data and First Results . . . 27
- 2.4 Extracting Motion from Videos . . . 29
  - 2.4.1 Comparison of Region . . . 31
  - 2.4.2 Statistical Descriptors . . . 31
  - 2.4.3 Differential Methods . . . 33
  - 2.4.4 Optical Flow and Particles Tracking . . . 36
- 2.5 Algorithm Development . . . 38
  - 2.5.1 Pyramidal Box Matching Tracking . . . 38
  - 2.5.2 Robust Tracking . . . 39
  - 2.5.3 Tracking Particles with Limited Life Span . . . 42

3 User Oriented Application for Video Tracking **45**
- 3.1 Introduction . . . 46
- 3.2 Visualization of Motion Flow . . . 47

3.3 User Interface to Control Time . . . 49
3.4 Manual Input . . . 51
3.4.1 Key-Frame Approach . . . 51
3.4.2 Sketching Motion . . . 53
3.5 User Interface for Correction . . . 55
3.5.1 Correction in Space . . . 56
3.5.2 Time Correction . . . 57
3.6 Conclusion . . . 58

4 Extraction of Plant Structure and Retargetting of Motion Data 59
4.1 Introduction . . . 60
4.2 Related Works . . . 61
4.2.1 Structure from Video Motion . . . 61
4.2.2 Animation from Video Motion . . . 62
4.3 Building a Hierarchy of Branches from a Single Video View . . . 63
4.3.1 Clustering Metric . . . 64
4.3.2 Selection of a Hierarchy of Branches using Clustering . . . 65
4.4 Creating 3D Shape and Motion . . . 69
4.4.1 Creating Motion of the Terminal Groups . . . 69
4.4.2 Propagating Motion to Intermediate Groups . . . 69
4.4.3 Extending Groups to 3D Shape and Motion . . . 70
4.5 Controlling the Animation of Complex 3D Plant Models . . . 72
4.5.1 3D Animation of a Plant by Skinning . . . 72
4.5.2 Interactive 3D Modeling . . . 73
4.6 Discussions on our Approach . . . 74
4.7 Conclusion . . . 75

II Mechanical Simulation 77

5 Real-Time Animation of Trees using Simulation 79
5.1 Introduction . . . 80
5.1.1 General Principles . . . 80
5.1.2 Outline of the Chapter . . . 82
5.2 Wind Models . . . 82
5.2.1 Physical Simulation . . . 83
5.2.2 Procedural and Phenomenological . . . 84
5.2.3 Stochastic Wind . . . 86
5.3 Modeling the Structural Elements . . . 87
5.3.1 Undeformable Segments with Angular Spring . . . 87
5.3.2 Uniform Beam . . . 88
5.3.3 Cosserat Rod . . . 90
5.4 Structural Modeling . . . 92

5.4.1 Dynamics Formalism . . . 92
5.4.2 Independent Elements . . . 94
5.4.3 Articulated Structures . . . 94
5.4.4 Finite Element Method . . . 95
5.5 Time Integration . . . 96
5.5.1 Static Equilibrium . . . 96
5.5.2 Explicit and Implicit Method . . . 97
5.5.3 Spectral Method . . . 99
5.6 Summary . . . 101

6 Modal Animation **103**
6.1 Introduction . . . 104
6.2 Modal Analysis and Modal Animation Framework . . . 105
6.2.1 Finite Element Method . . . 105
6.2.2 Modal Analysis . . . 108
6.2.3 Animation using the Modes of Deformation . . . 109
6.3 Wind Projection Basis . . . 110
6.4 Implementation Issues . . . 112
6.4.1 GPU Implementation . . . 113
6.4.2 Error Correction . . . 113
6.4.3 Configuration . . . 115
6.5 Results . . . 116
6.6 Limitations and Future Work . . . 117
6.7 Conclusion . . . 118

7 Conclusion **119**

List of figures **125**

Bibliography **133**

chapter 1

Introduction

1.1 Rationale

Vegetation is present all around us and we are accustomed to see trees and other plants everyday. Its accurate representation is thus an essential part of the realistic depiction of natural scenes in a virtual environment. Due to the complexity of both vegetation and its reaction to wind load, such representation are still being unresolved issues of research in computer graphics. In a first step to the development of suitable models of animated plants, it is thus essential to understand as much as possible all the phenomena which produce the observable motion of trees.

1.1.1 Trees

Structural Complexity

One of the main characteristics of trees is their diversity. Compare to most other natural or man-made physical objects, trees and shrubs have a rather complex architecture. A first structural representation can be extracted by looking at the biological organization of branches and trunk (see figure 1.1).

At a cellular level, a plant can be described by the organization of its growth. This is done by two distinct types of cells: the primary and the secondary meristem. The first is responsible for the increase in length and provide a natural segmentation of the branches. The second causes them to grow laterally to become more robust.

Plant growth evolves in function of many factors of the environment and can produce a wide diversity of structures. Considering a larger scale, the architectures of plants can be very different depending on the species and the environment. However complex they may be, we can see that patterns emerge. Francis Hallé [Hal05] provided a classification of the possible hierarchical structure of trees (see figure 1.2).

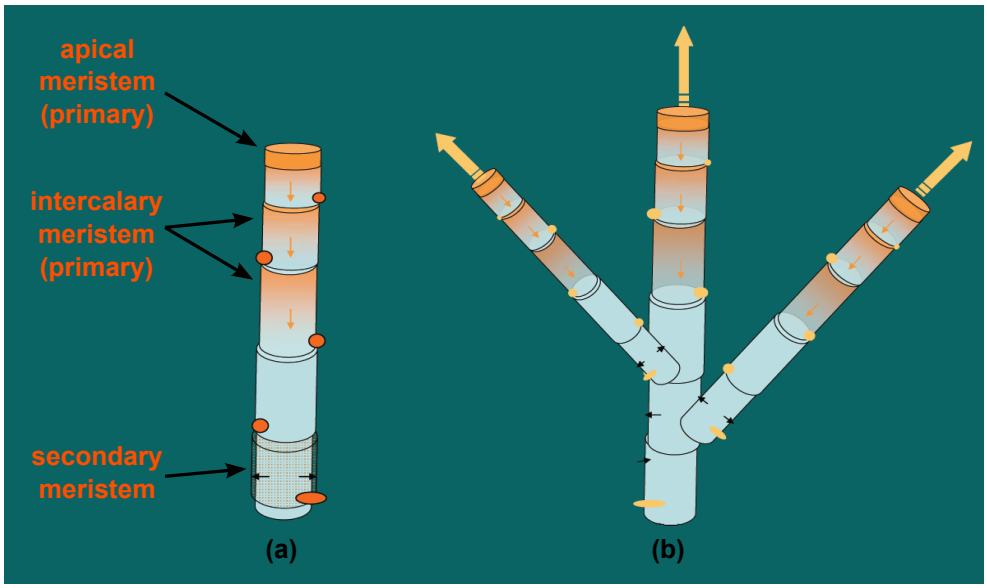


Figure 1.1: Biological structure of trunk and branches.

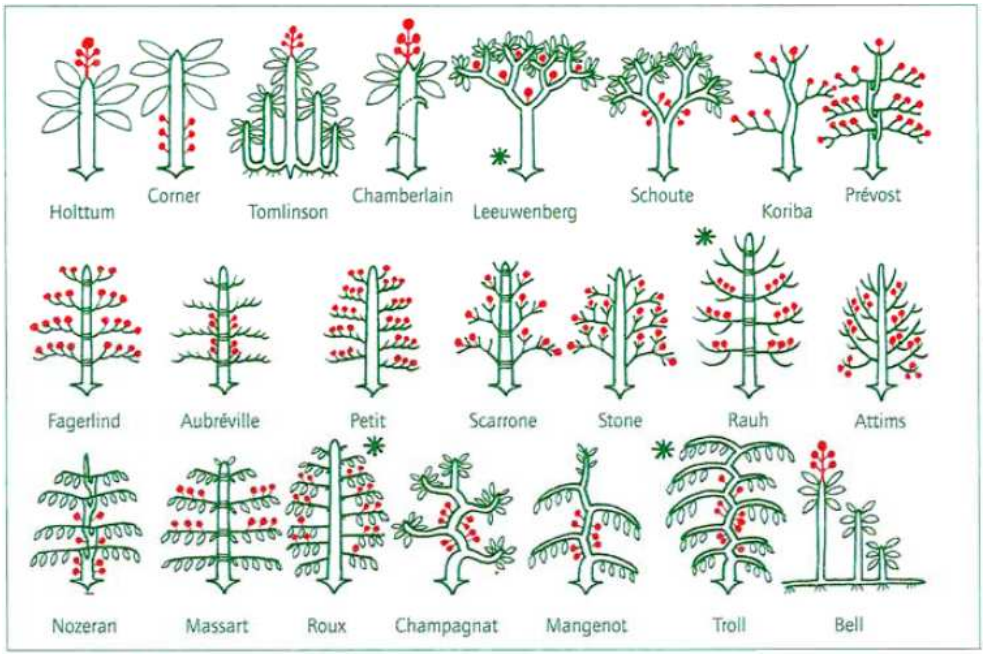
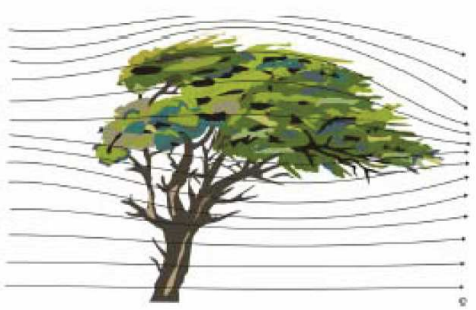


Figure 1.2: Classification of possible hierarchical structure of trees. The four more common structures are mark with an asterisk. (Image taken from [Hal05])

In the end, one of the main difficulty for studies concerning plants is the number of elements (branches, leaves,...) that have to be taken into account and their spatial organization. Even for a little shrub, the structure can be composed of hundreds or more branches.



Interaction with the Wind

An other important factor is that plants evolve in a dynamical environment. In particular its interaction with the surrounding air flow, which interest us most in this thesis, has many complex consequences. For example, results have emphasize the reaction of plant growth to external wind load, a phenomena called *thigmomorphogenesis* (see figure 1.3). A primary explanation is that under mechanical strain plants change from having a principal growth in height (of the primary meristem) to an increase of the branches diameters (secondary meristem) in order to become more robust.

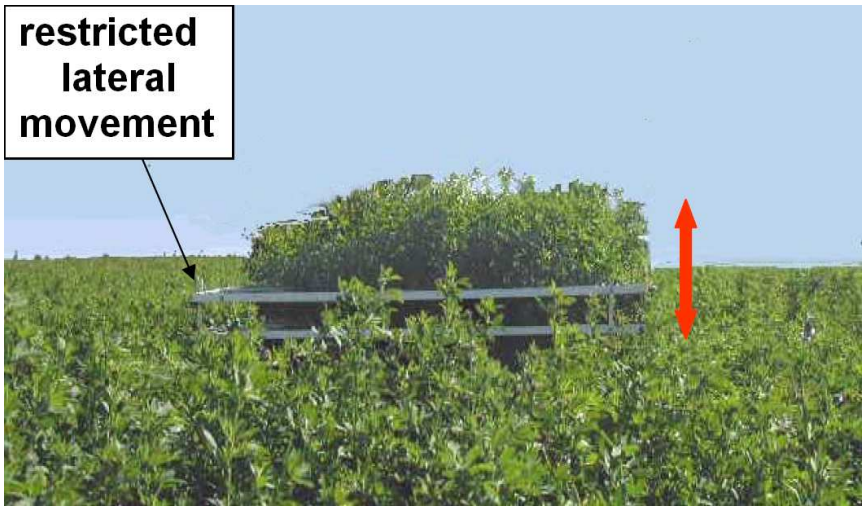


Figure 1.3: Experiment showing how important a factor the wind is to the growth in height of crop. A patch of a crop field has its lateral motion (due to wind) restricted. It results in a growth much higher of the crop. (Image and experiment presented in [MC06])

An other example is the critical consequences that can arise which concern agronomic and ecological risk. It is obvious that under strong enough wind, plant eventually breaks. However contrary to what it might be believed, the wind speed is not the only cause. The strong interaction between the wind fluctuation and the branches, or trunk, oscillation can provide a better explaination for some observed phenomena. Figure 1.4(a) shows a case of *plant lodging* where the crop have bend too much to raise again. In the same manner, in figure 1.4(b) a destructed part of a forest is presented. In both cases, the area is localized in middle of the field and the forest respectively while the wind has been of similar speed over a wider area.

(a) Field lodging

(b) forest after the *Lothar and Martin* windstorms (1999)

Figure 1.4: Breaking of plants at different scales due to wind action. Studies describe the phenomena as a consequence of a *locking mechanism* between the wind frequency and the plants oscillation.

These examples aim to emphasize the complexity of the interaction between plants and air flow. Trees are not growing in order to withstand statically the wind load but rather to respond coordinately. This observation indicates the difficulty of developing a realistic animation methods for computer graphics.

1.1.2 Tree Animation in Computer Graphics

Since the beginning of computer graphics, researches related to the representation of plants have primarily focused on their modeling and rendering. While it has produced impressive results, animation techniques remain of visually poor quality. Obviously the logical process of research had first to develop suitable techniques to create and render natural looking 3D models and in some cases reproduce real (static) trees in a virtual environment. With the development of computer technology, virtual scenes containing complex vegetation has now become ordinary. Realistic animation should thus be provided for these models as static vegetation cannot fulfill desired perception of realism.

The main difficulty of this task is the complexity of the plants structure, their diversity in shape and mechanical property and their complex interaction with the environment. Current computer performance now allows applying general off-line simulation techniques to compute the dynamics of thousands of branches interacting with fluids such as the wind.

However this technological development has also led for real-time applications to render natural scenes with high quality and complex geometry. This increase in model complexity has then generated the need for suitable (*i.e., low cost*) animation techniques. As the existing accurate simulation methods are still too computationally expensive, developers usually resort to ad-hoc techniques. Such methods typically apply pseudo-random oscillation to the branches. If they

can be satisfying for models with simple structures such as grass or palm trees, they fail to provide realistic animation of complex branches hierarchy. There are thus open research areas on real-time animation of plants.

1.2 Contribution

The primary goal of my Ph.D. is to develop innovative animation techniques that provide useful alternatives for real-time application. To this end, two main approaches have been explored. First I focused on reproducing the real motion of plants. Then, in a second part, I worked on the design of a new computer simulation suitable for real-time animation.

Led by the first approach requirements, a large part of my work has been to search of suitable ways of recording and processing the observable movement of plants. One of the main scientific contribution of my Ph.D. has been to record and gather many video sequences on the real motion of shrubs and trees. In most cases, a single camera fixed on a tripod was used as the recording device, often without setting up a calibration process (*i.e.,* the procedure to estimate the position and the projection matrix of the video camera). In these cases, the sequences were mostly gathered as case studies for the extraction of motion from videos.

I have also been involved in a set of experiments to extract more detailed data on real motion of a tree. The main goal was to collect as much information as possible for several sequences of the motion of one tree stressed by the wind and by manual load. For these experiments, the tree was first digitalized in order to know its structure. Then for each sequence we have recorded the wind speed, the tree foliage motion (filmed in stereo using two cameras), and the movement of a few specific branches using a magnetic tracking device. An important issue has been to setup a protocol which allows the calibration of stereo recording suitable for this experimental setting.

For my Ph.D., we made the choice to focus on the use of video camera as main input device and work on methods to extract the 2D motion present in video sequences to obtain data. The main reason of using this approach is that video cameras provide a much simpler experimental setup compared to traditional motion capture systems. Moreover the use of such input has not been studied previously for the case of plants. As a tradeoff however, it requires to extract reliable motion data from the video sequences obtained. This is a difficult task that has been one of the main challenges throughout my research.

Several well known vision algorithm were tested to estimate the foliage motion. However most typical (and necessary) assumptions made by these methods are not adapted to videos of animated foliage which presents many hard impediments. To cope with this difficulties, I have first developed some algorithms to increase the speed and robustness of vision tracking. But all-automatic methods have limitations that cannot always be overcome. I have thus focus on designing a user interface mixing the automatic tracking techniques with user actions in order to quickly extract reliable motion data that can be used for further studies.

The concluding step of my work based on the use of videos, in the point of view of the presented Ph.D., is the reproduction of observed animation on virtual models of trees. This is also quite

a complex issue that was not studied previously. Traditional techniques were developed mostly for articulated subject such as human and animals and cannot be used for plants. On the contrary of models of known (and simpler) structures, the organization of branches is an unknown when extracting the plant motion. An important result of my work is an innovative method to estimate a valid hierarchical structure of plants present in a video only using a statistical study of the estimated video motion. We use a clustering algorithms to extract a hierarchical classification of a set of features tracked in the video. An important contribution is the metric employed. It allows the obtained classification to be topologically similar to the real plant structure. Geometrical constraints are then used to model a hierarchy of virtual branches animated according to the video motion. Finally, using a traditional animation technique we have shown that the obtained structure can be used to retarget the observed branches movement on a wide range of virtual plant models. The results of this work has been published in the Symposium on Computer Animation 2006 [DRF06].

An other part of my work has been to study mechanics and computer simulation in order to develop a new real-time animation technique for virtual plant models. We obtained a method to compute the dynamical response with respect to wind flow that can be controlled interactively. The proposed approach is based on vibration analysis of elastic structures and their aerodynamic interaction with fluids, such as air. Due to the complexity of plant structures, realistic simulation methods are very time consuming in particular because the interaction between the wind flow and each branch has to be considered.

A significant contribution of my work has been to show that, using simple approximation of the wind load model, the run-time computations could be reduced drastically for a simulation method based on the modes of vibration of the structures. To this end, we introduce the precomputed *modal projection basis* which allows to compute the tree modal stress at run-time with a simple projection of the wind speed onto this basis. Moreover, the obtained simulation method can take full advantage of the parallel processing capabilities of computer graphics cards which allows the animation of thousands of trees in real-time. This method has been published at the Eurographics conference on computer graphics 2009 [DRBR09].

During the second half of my Ph.D., I have been involved with the ANR project Chêne-Roseau [CHE]. This research project, founded by the *Agence National pour la Recherche (ANR)*, is a interdisciplinary collaboration focused on the interaction of trees with the wind. This project is composed of four research teams: LadHyX of *Ecole Polytechnique* which study fluid mechanics and fluid-structure interaction; Piaf of *Institut National de la Recherche Agronomique (INRA)* at Clermont-Ferrand, working on the biomechanics of plants; Ephyse of INRA Bordeaux specialized on numerical simulation of larger scale interaction between the atmosphere and vegetation; and Evasion of the computer graphics community mostly focused on virtual representation of natural scenes.

It has been a great opportunity for my Ph.D.. The collaborations have brought me more acute knowledge in the phenomena I studied and led me to be involved with researches of wide scope. My participation has been in particular on data acquisition of real plants motion and, through-

out my collaboration with Mathieu Rodriguez (LadHyX / Piaf), on the development of the new real-time simulation method.

1.3 Thesis Outline

This thesis is organized in two parts. The first is on the acquisition and the reproduction of the motion of real plants and contains three chapters. The second is on mechanical simulation of plants dynamics and is divided in two chapters.

In the first part, chapter 2 describes experimental work I have done and participated to within the Chene-Roseau project and shows the obtained data on plants response to natural wind load and manual excitation. We also discuss how features observable in a video can be automatically tracked along the sequence and presents some methods I have developed.

These algorithms have been put in a software I developed that also focus on user interaction to compensate for the limitation of automatic techniques. An explanation is given in chapter 3 of how manual input can be used to extract reliable motion data from video.

At the end of the first part, chapter 4 presents our results on structure extraction from 2D motion data (previously extracted from a video) and its retargeting. The statistical study of 2D motion data I have developed is discussed. We show how it can be used to extract a valid hierarchical branches structure that holds the plants motion and how it is used to reproduce the observed motion on a virtual model.

In the second part, simulation method of tree response to wind load is discussed. In chapter 5, a state of the art of existing real-time animation technique is given. We introduce several concept of mechanics and simulation of elastic structure dynamics in order to compare all described methods.

Finally chapter 6 presents the methods developed in collaboration with Mathieu Rodriguez on the real-time simulation of thousands of trees in response to interactive wind.

Part I

Acquisition and Reproduction of Real Motion

chapter 2

Acquisition of Plants Motion

Contents

2.1 Introduction **18**
2.2 Motion Capture **18**
2.2.1 Origin of Motion Capture 18
2.2.2 Assisted Motion Capture 20
2.2.3 Markerless Motion Tracking 22
2.3 Data Acquisition **23**
2.3.1 Experimental Setup 23
2.3.2 Cameras Calibration 24
2.3.3 Recorded Data and First Results 27
2.4 Extracting Motion from Videos **29**
2.4.1 Comparison of Region 31
2.4.2 Statistical Descriptors 31
2.4.3 Differential Methods 33
2.4.4 Optical Flow and Particles Tracking 36
2.5 Algorithm Development **38**
2.5.1 Pyramidal Box Matching Tracking 38
2.5.2 Robust Tracking 39
2.5.3 Tracking Particles with Limited Life Span 42

2.1 Introduction

Whether the goal is the reproduction of real motion of trees on virtual models or for mechanical studies of plants dynamics, reliable techniques to record and extract plants motion are necessary. Methods to record real motion data, called motion capture (*mocap*), are frequently used for many types of subjects and applications. However, in comparison to traditional subject such as humans and animals, the capture of the plants motion is a particularly difficult task due to the relative complexity of their structures. As a consequence very few data exist on plant motion. This chapter is divided in two parts that represent the two main issues of motion capture: suitable experimental setups to record data and methods to process them.

The first part discuss one of the main scientific contribution of the works presented in this thesis which has been to record and gather data on real motion of plants. In section 2.2 we first give an overview of existing motion capture techniques. Then a description is given in section 2.3 of the experiments protocol we have developed to record tree motion and the data obtained.

During my Ph.D. we have focused on the use of video cameras as main input device to the extraction of real motion of plants. It represents a very challenging task to provide a suitable mocap method based only on video sequences that has not been done previously. The second part of this chapter discuss this issues. In section 2.4 we first describe the basis of computer vision, *i.e.,* methods to extract motion from videos. Then in section 2.5 we provide some contributions developed to increase the quality of motion estimation in videos of animated foliage.

2.2 Motion Capture

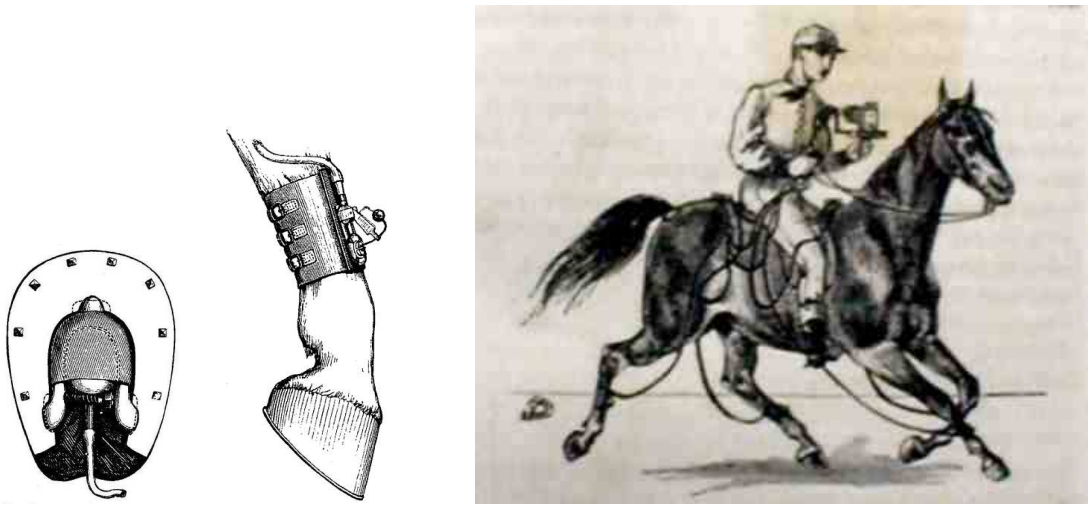
There exist a lot of different acquisition techniques using multiple types of hardware. Their experimental suitability depends on many factors. For example, specific methods are available for each type of motion to be captured such as the displacement of articulated rigid bodies or the motion of fluids. Another important criterion is the ratio between financial cost and desired precision. Finally for many cases such as motion capture of wild animals or outdoor trees, the required hardware should be usable in the specific environment. In this section we review only a short sample of existing methods in order to give a global description of the possible acquisition systems.

2.2.1 Origin of Motion Capture

Early works started in the mid-nineteenth century at the same time as the invention of photography. Two of the first representative examples can be seen in the works of the French scientist Étienne-Jules Marey and American Edward James Muybridge.

Marey participated to the fundamental innovation in physiology called the *graphical method*. He developed many methods using ingenious devices to record precisely the motion of humans and

animals (see figure 2.1). This *graphical method* meant essentially to carry scientific experiments that record, and study, precise measures of motion data (in particular for complex biological system). Many of these methods are now considered the first motion capture techniques.



(a) Device 1 (b) Device 2 (c) Setup on horse

Figure 2.1: (a) and (b) Devices to record contacts of a horse hoof with the ground. (c) Setup to record all the contacts of the hooves of a horse during locomotion using device 1 or 2.

On the other hand, Muybridge was originally a photographer and his work was more artistic than scientific. However, he and Marey have strongly influenced each other's work. It first occurred when Muybridge came across Marey's work on horse locomotion. Marey produced experimental data showing that for a brief moment a galloping horse had all four hooves off the ground. However it did not convince the equestrian community. Muybridge made then an experiment and recorded a horse gallop with a set of cameras launched one after the other when the horse passed. The resulting image sequence, *the horse in motion*, provided a visual proof of Marey's theory (see figure 2.2).

This experiment has then brought Marey to use serial pictures as a method of studying the mechanics of locomotion (see figure 2.3). The complex setting of Muybridge's method (setting multiple independent cameras next to each other) did not allow easy recording of many animals motion such as birds flying. Marey then developed many devices to record more image sequences in different environment, such as the chronophotographic gun (see figure 2.4) that lead to the invention of cinematography.

Since then, techniques and applications of motion capture have quickly expanded as new hardware appeared. Nowadays motion capture is a primarily tool used for physiological study of locomotion and general mechanical study of biological system. It is one of the main animation techniques of virtual models in the film and computer game industry, and it is also used as an input method for human interaction in augmented reality environment.

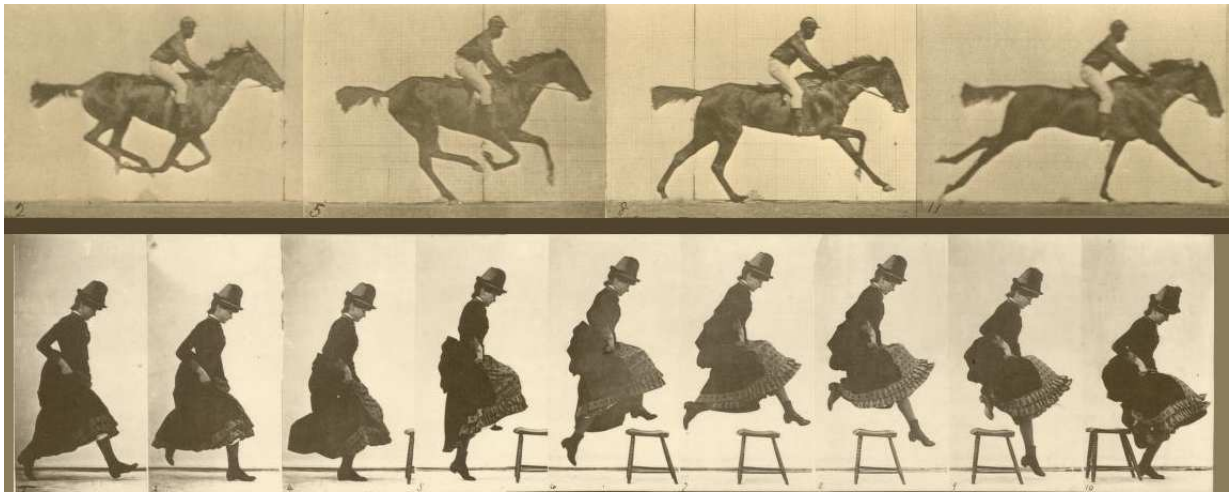
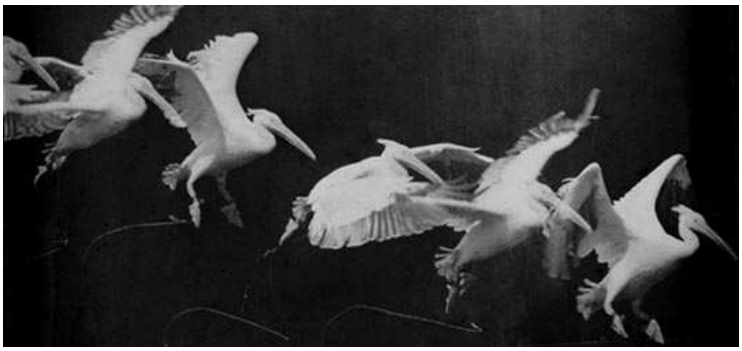


Figure 2.2: Excerpt from two of the Muybridge sequences. Top: excerpt from sequence *Animals and Movements, Horses, Gallop; thoroughbred bay mare, Annie G*. The first image proves Marey's theory that all hooves of horses leave the ground at some point during gallop. Bottom: excerpt from sequence *Movements, Female, Jumping: running straight high jump*.

(a) Landing of a cat



(b) Landing of a pelican

Figure 2.3: Two sequences from Marey's results.

2.2.2 Assisted Motion Capture

Most motion capture techniques uses specialized hardware attached onto the subject such as markers or sensors. These markers can be of many types for which the acquisition system records the appropriate data such as position or orientation.

Figure 2.4: The chronophotographic gun design by Marey. A camera is mounted on a modified traditional gun where a photographic film is stored in the chamber. It allowed capturing up to twelve pictures by second.

To track the displacement or deformation of an object, one possibility is to record the change of orientation of its parts relatively to each others. Inertial sensors such as gyroscopes or exoskeletons can be used to record the orientation of the elements of the subjects. This approach is used by Kenneth James [Jam03, JHA06] to measure to displacement of the trunk of trees.

By using multiple markers one can reconstruct the overall shape of an articulated body. The main advantage is that each marker is independent and can be easily attached on the subject. On the other hand, for objects that exhibit non-rigid deformations or complex structures, the reconstruction may not be precise enough. For example the displacement of the tip of long branches obtained using the change of rotation angles at joints between all the parent branches propagate inaccuracy of the measures. Moreover, depending on the technology used the weight of the sensors may influence the dynamics of the lighter branches.

Another possibility is to track the spatial positions at specific parts of the subject. Vlasic *et al.* [VAV+07] used a combination of ultrasonic and inertial sensors to extract the motion of a human. The ultrasonic system works by estimating the time of flight of sounds from emitters with known positions to the sensors (receivers) and use triangulation to reconstruct 3D position. This is a cost effective solution for tracking but requires an a priori model of the subject (in this case a human skeleton). Magnetic systems provide a more expensive but more precise option. An emitter creates a particular magnetic field such that the sensors can record their position and their orientations. However, typical hardware can track only a few markers in real-time. We describe in section 2.3 how such a system is used in our experimental protocol.

The most common approaches are optical systems. Two main types exist: recording either natural or infrared light using the suitable cameras. Both usually use passive markers that reflect light. They are attached to the subject who is filmed by one or several cameras. Then the markers are located in video sequences and 3D reconstruction algorithms are used to compute their 3D positions. Infrared optical systems have become the most common methods for motion capture of human (see figure 2.5). The advantage over optical systems using traditional cameras is that they can be used in natural lighting conditions. However they need a more complex setup, with suitable infrared projectors and cameras.

Figure 2.5: Tracking a human in motion using an infrared optical system. The subject has retroreflective markers set at specific parts if her body. A projector (red light at the top of the image) emits infrared light reflected by the markers. It is then detected in video sequences that film the subject.

2.2.3 Markerless Motion Tracking

With the increase of computer processing power and the development of computer vision algorithm, more and more motion capture techniques are developed to be used without marker but only with video sequences of the subject in motion.

For this type of mocap methods, multiple cameras can be used together with an algorithm for 3D reconstruction. However the setup necessary for these approaches would make the use of markers comparatively simple. The main advantage of markerless methods comes when only one camera suffices to extract 3D motion. However, theoritical representations provide only underconstrianed problems when the data are recorded from a single view point.

Many models have been developed for human face and body (see [Gav99], [FAI+05] or [Smi07] for state-of-the-art reports). These techniques resort to use prior knowledge of human topological structure, such as segmenting the human body in a set of rigid parts (head, body, arms, legs) or the relative position of the eyes, nose and mouth for human faces. Even then, many ambiguities make these mocap problems very challenging.

Very few works have been done on markerless mocap for other subjects than humans. Methods have been designed to track animals' motion such as the approaches developed by Bregler *et al.* [BMP04] or by Favreau *et al.* [FRDC04] which use the optical flow to animate a virtual model of quadrupeds. But it exists no model for plants for which the topological structure is unknown (only its hierarchical nature can be considered). Moreover computer vision techniques often fail on video sequences of foliage animated by wind because of the highly fluctuating motion of leaves and the branches occlusion. They thus cannot guaranty to have a precise motion extraction.

2.3 Data Acquisition

During this thesis, many video sequences of shrubs and trees motion were recorded. In most cases, a camera fixed on a tripod was used as the recording device often without setting up a calibration process (see section 2.3.2). These sequences were mostly gathered as case studies for the extraction of motion from videos.

As part of the Chene-Roseau project [CHE], we have been involved in a set of experiments to extract more detailed data on real motion of a tree. The experiments took place at the *Institut National de la Recherche Agronomique* (INRA) in the city of Clermont-Ferrand. The main goal was to collect as much information as possible for several sequences of a tree motion stressed by the wind. These data have then been stored and will be made available to the research community through the Chene-Roseau project.

2.3.1 Experimental Setup

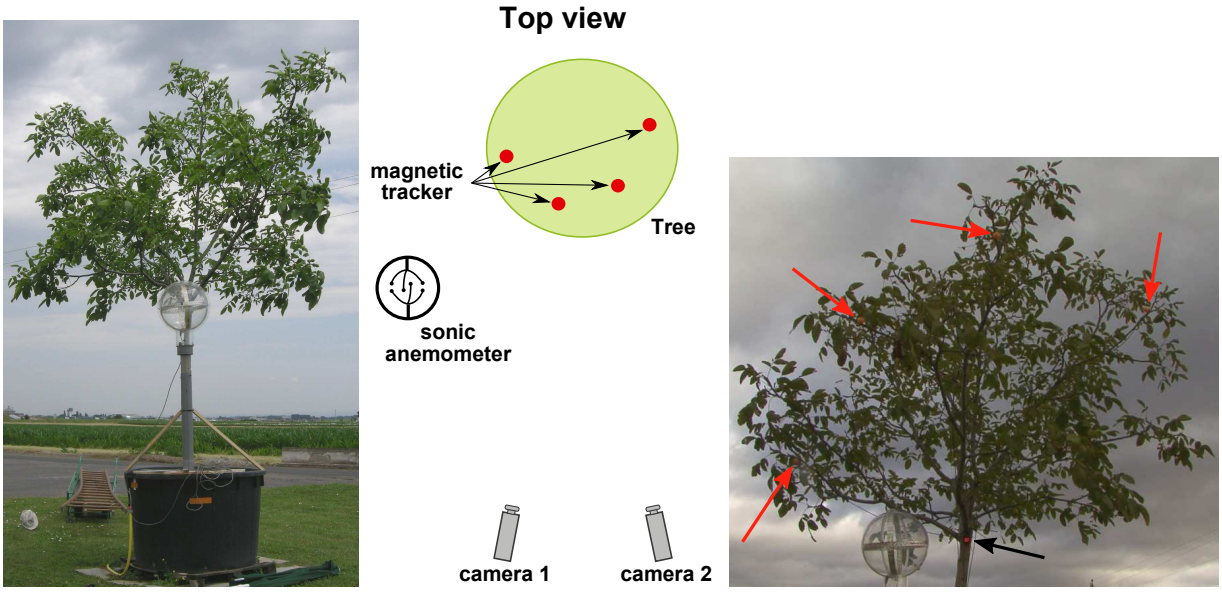
All the experiments were done with one tree that has grown at the INRA. Before the experiments took place, it has been entirely digitalized by the people of the INRA using their expertise in this field (same process as described in [SRG97]). The first piece of data obtained is then the full structure of the tree in its resting position.

(a) Magnetic tracking devices (b) One magnetic marker

Figure 2.6: The magnetic tracking system is composed of a main unit (box in the middle of image (a)) and of a transmitter that generates a magnetic field (sphere on the right of image (a)). Up to four magnetic receivers can be connected to the main unit and tracked in real time. A software developed at the INRA controls all this system.

For all the sequences, three types of data were recorded: videos, wind speed and 3D magnetic tracking of a few selected branches. The wind speed is recorded using a sonic anemometer installed as close as possible to the tree such that it does not disturb the magnetic system. The 3D tracking system (see figure 2.6) is composed of a magnetic transmitter that emits a magnetic field

and of small receivers, *i.e.,* the markers for which the positions (and orientations) are recorded in real-time ($\geq 25Hz$). They were placed on four different branches for which the 3D motion trajectories were then recorded (see figure 2.7). Then, for all sequences two cameras provided video recording of the tree.



(a) (b) (c)

Figure 2.7: (a) Tree used for experimentation. (b) Experimental layout: Two cameras film the tree from one side, four magnetic markers are attached to representative branches and a sonic anemometer records the wind speed next to the tree. (c) An image taken by camera 2. The positions of the magnetic markers are indicated by red arrows, and the LED device used for synchronization is indicated by a black arrow.

All the data extracted during the experiments have to be calibrated both in time and space. First all recording have to be synchronized, *i.e.,* such that all data (images of the two cameras, reading of the four magnetic markers and wind speed from the anemometer) are coordinated temporally. The magnetic markers position and the wind speed are recorded together by the same software, thus already coordinated. However the two cameras need temporal reference with this system. To this end we designed a device made of a grid of LED (visible from both cameras) that are enlightened when recording the magnetic markers position (see figure 2.7). Second, the magnetic tracking system records all markers positions in the same reference frame, defined relatively to the magnetic transmitter. However, for videos the projection matrices of both camera and their relative positions have to be estimated.

2.3.2 Cameras Calibration

There are multiple ways to calibrate cameras. For any method, a real object with known shape, called the *calibration rig*, is filmed such that specific features can be detected in the video image.

Then the projection matrix of the camera is computed in order for the theoretical projection of this object to match the observed projection in the video image (see figure 2.8).

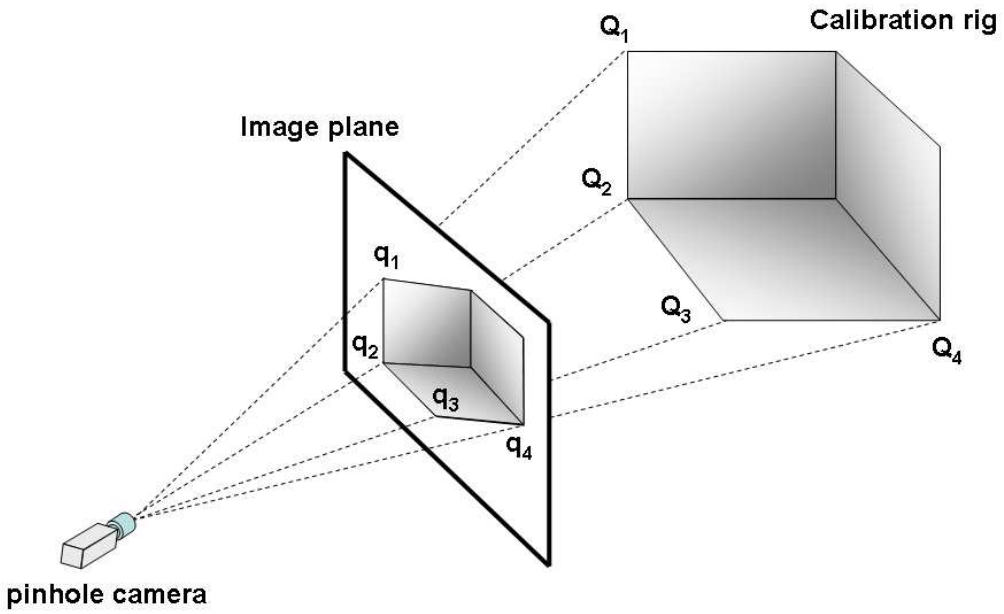


Figure 2.8: Representation of the *pinhole* camera. On the right, the calibration rig for which we know the shape, *i.e.,* the 3D positions Q_i (in the reference frame of the calibration rig). The projection matrix of the camera is calibrated such that it projects them onto their observed projections in the image plane, *i.e.,* the 2D positions q_i.

Mathematical formulation

Let $Q_i = (x_i, y_i, z_i, 1)^T$ be a set of n points of the calibration rig and $\mathbf{q}_i^k = (u_i^k, v_i^k, 1)^T$ their observed projection in the image k, both formulated in homogeneous coordinates. The pinhole camera model has the form:

$$\mathbf{q}_i^k \propto \mathbf{A}\,[\mathbf{R}^k\ \mathbf{t}^k]Q_i \tag{2.1}$$

$$\text{with} \qquad \mathbf{A} = \begin{pmatrix} \alpha & \sigma & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{pmatrix}$$

The extrinsic parameters are contained in the 3×4 matrix $[\mathbf{R}^k \ \mathbf{t}^k]$. It is composed of a 3×3 rotation matrix $\mathbf{R}^k$ and a 3×1 translation vector $\mathbf{t}^k$ which relates the frame of the calibration rig (considered as the world reference frame) to the camera coordinate system for each image k; A is called the camera intrinsic matrix, where (c_x, c_y) are the coordinates of the principal point (usually the center of the image plane), α and β the scale factors of the x and y axes of the image, and σ the parameter describing the skewness between the two axes of the image plane.

A simple calibration algorithm directly solves equation (2.1) by finding the coefficients of the 3×4 projection matrix $\mathbf{P}^k = \mathbf{A}\ [\mathbf{R}^k \ \mathbf{t}^k]$ independently for each image k. However to obtain a good calibration, this approach requires not only to provide enough point correspondences, but also needs a three-dimensional calibration rig to obtain uniqueness of the solution. Moreover for the calibration to be well estimated the calibration rig should cover (*i.e.,* span) as much as possible the volume that is filmed to reduce the influence of the measurement noise. Thus this is not a suitable solution for our purpose. The size of the tree volume would require a very big calibration rig.

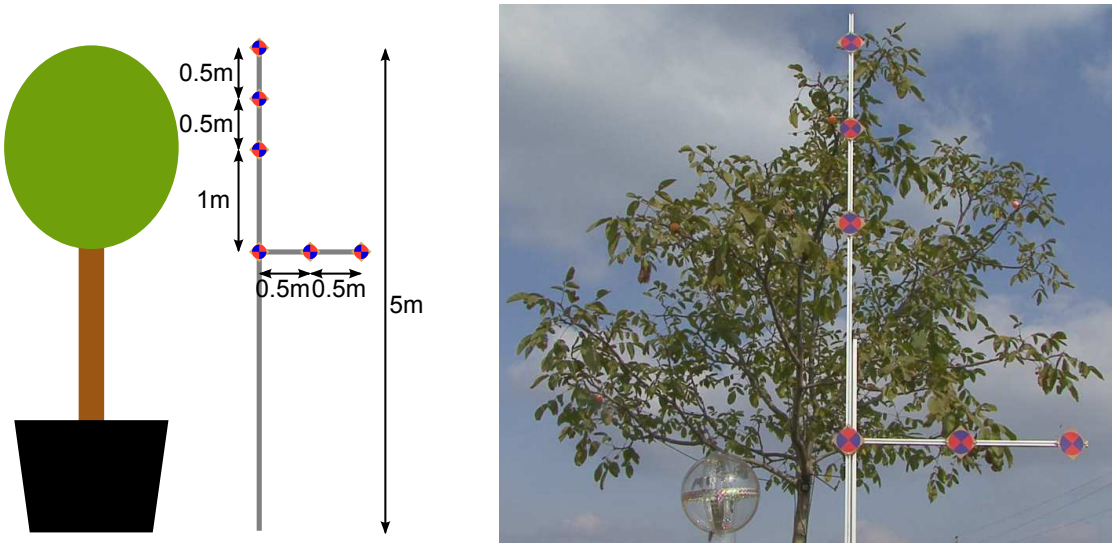
Practical calibration rig

Instead we use the technique introduced by Zhang [Zha99] that only requires several views of a planar calibration rig. The idea is to estimate, first the intrinsic parameters (matrix $\mathbf{A}$) using several views of the calibration rig, and in a second step to estimate the extrinsic parameters (matrix $[\mathbf{R}^k \ \mathbf{t}^k]$) for each views k. Such algorithms have become common and we use the version included in the computer vision library OpenCV.

This approach is often used with a calibration rig made of a flat cardboard on which a checkboard pattern has been drawn. Then during the calibration process, this cardboard is filmed while it is moved all around the filmed volume (to provide the best estimate of the intrinsic parameters). Due to the size of the tree used during experiment (between five and six meters tall), this setup would require a cardboard of more than one meter width and it should be displaced at several meters high.

To simplify the calibration process, we designed a calibration rig made of a main bar of five meters length to which a smaller bar (one meter) is attached perpendicularly (see figure 2.9). Then six targets, easily detectable in an image, are fixed on the bars: three along the first, two on the second and one at their junction.

During the experiments, the first step is to position and configure the cameras (zoom, focus, shutter speed, etc...) such as the tree foliage fills most of the image. Then the internal clocks of the two cameras are synchronized (the camera we employed provided the suitable synchronization tool). In this way, all the video sequences obtained afterwards can be fitted in time to the sequence of the other camera and pairs of images taken at the same time are easily selected. Moreover, a LED device visible by both camera (see figure **??**) that is launched and stoped by the magnetic tracking system (which also records the wind speed) allows temporal synchronization of all recordings.



(a) (b)

Figure 2.9: The calibration rig we designed.

Finally the video sequence is done where the calibration rig is displaced all around the tree. Each time a camera is moved or some of their intrinsic parameters are changed, this calibration sequence has to be made again. From this sequence a set of images is selected such that the calibration rig appears all around the tree and with different exposure angles (see figure 2.10).

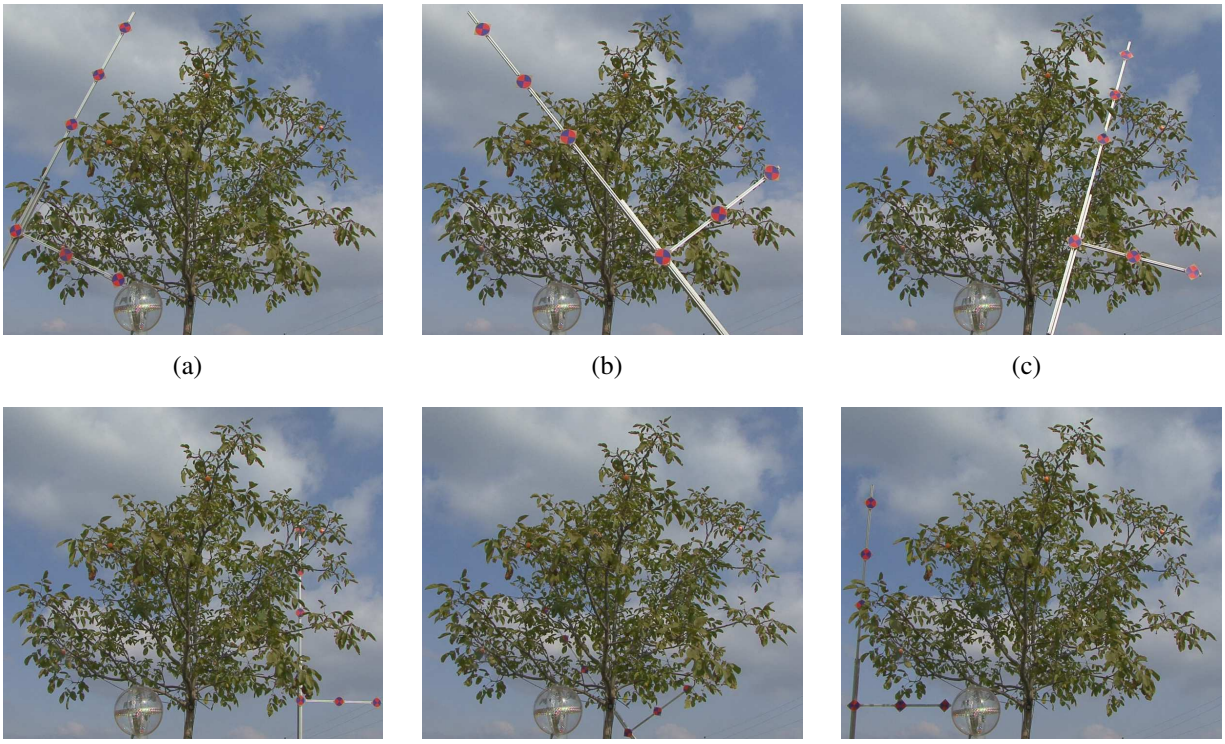
2.3.3 Recorded Data and First Results

Three recording sessions have been done. The first two were at the beginning and end of summer and the third during winter (*i.e.,* without leaves). In each case, the tree response for two types of external forces have been recorded: natural wind and localized impulse obtained by pulling selected branches manually using a rope. For each recorded sequence, the 3D positions of four markers and two videos from two different views were recorded. Moreover the markers were placed inside colored balls to be easily tracked in the videos. This has enabled to test the quality of the video calibration. The four markers were tracked in both videos then their 3D positions were reconstructed to be compared with the 3D magnetic tracking.

3D Reconstruction from Videos

Let Q be one of the 3D markers position in one frame of the sequence. Let $\mathbf{P}_1$ and $\mathbf{P}_2$ be the 3×4 projection matrix of the first and second camera respectively. And let $\mathbf{q}_1$ and $\mathbf{q}_2$ be the positions of the markers in the respective frames of both videos:

$$\mathbf{q}_1 \propto \mathbf{P}_1\, Q$$
$$\mathbf{q}_2 \propto \mathbf{P}_2\, Q$$



(a) (b) (c)

(d) (e) (f)

Figure 2.10: Subset images of a sequence used for calibration.

Now let $\mathbf{p}_i^k$ be the k^{th} row of the projection matrix $\mathbf{P}_i$, such that:

$$\mathbf{P}_i = \begin{bmatrix} \mathbf{p}_i^1 \\ \mathbf{p}_i^2 \\ \mathbf{p}_i^3 \end{bmatrix} \qquad \forall i$$

Then, expressed in standard coordinates, we have:

$$\begin{cases} u_i &= \dfrac{\mathbf{p}_i^1\ Q}{\mathbf{p}_i^3\ Q} \\ v_i &= \dfrac{\mathbf{p}_i^2\ Q}{\mathbf{p}_i^3\ Q} \end{cases} \qquad \text{with} \quad \mathbf{q}_i = \begin{pmatrix} u_i \\ v_i \end{pmatrix} \qquad \forall i$$

$$\Leftrightarrow \begin{cases} \mathbf{p}_i^1\ Q - u_i\ \mathbf{p}_i^3\ Q &= 0 \\ \mathbf{p}_i^2\ Q - v_i\ \mathbf{p}_i^3\ Q &= 0 \end{cases} \qquad \forall i$$

$$\Leftrightarrow \begin{bmatrix} \mathbf{p}_1^1 - u_1\,\mathbf{p}_1^3 \\ \mathbf{p}_1^2 - v_1\,\mathbf{p}_1^3 \\ \mathbf{p}_2^1 - u_2\,\mathbf{p}_2^3 \\ \mathbf{p}_2^2 - v_2\,\mathbf{p}_2^3 \end{bmatrix} Q = \mathbf{0} \tag{2.2}$$

Because of noise, this system usually has no exact solution but the trivial zero vector. Least square minimization algorithm is then used to find the best non-trivial solution for equation (2.2).

Results

The 3D position of the four markers were reconstructed then compared with the motion curve obtained using magnetic tracking. Figure 2.11 shows the 3D trajectories and theirs spectrum. Table 2.1 shows the average errors.

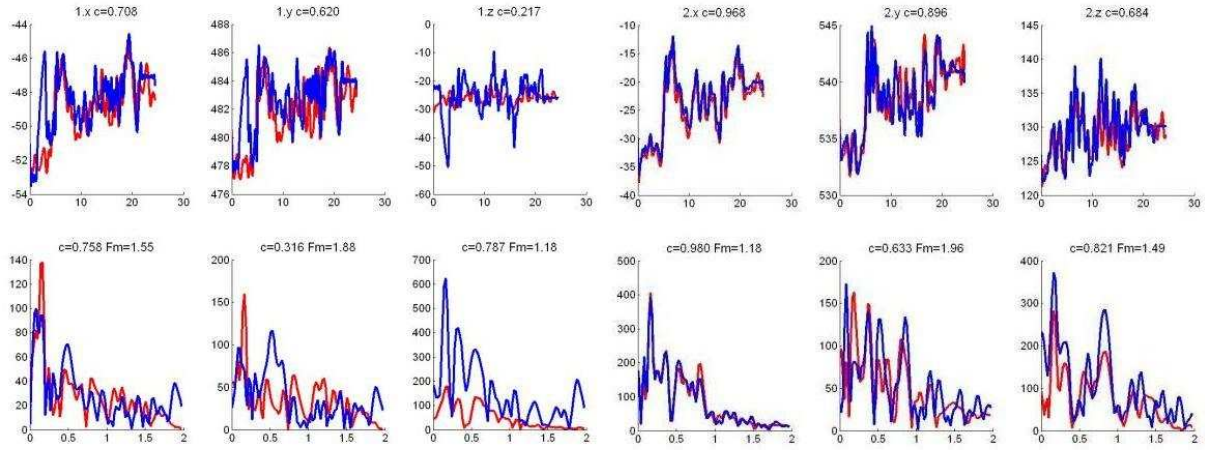
	marker 1	marker 2	marker 3	marker 4
X	1,56 (1.15 ± 1.06)	1,66 (1.37 ± 0.93)	2,05 (1.68 ± 1.18)	2,11 (1.77 ± 1.15)
Y	2,18 (1.70 ± 1.37)	1,44 (1.14 ± 0.88)	2,53 (2.09 ± 1.44)	1,85 (1.47 ± 1.12)
Z	6,43 (4.70 ± 4.39)	2,80 (2.12 ± 1.83)	3,02 (2.57 ± 1.59)	2,21 (1.70 ± 1.41)

Table 2.1: Root Mean Square (RMS) of the distance between the estimated position obtained by the two types of tracking, given for each coordinate and marker. In bracket: the average and standard deviation of the distance. Values are given in centimeters.

2.4 Extracting Motion from Videos

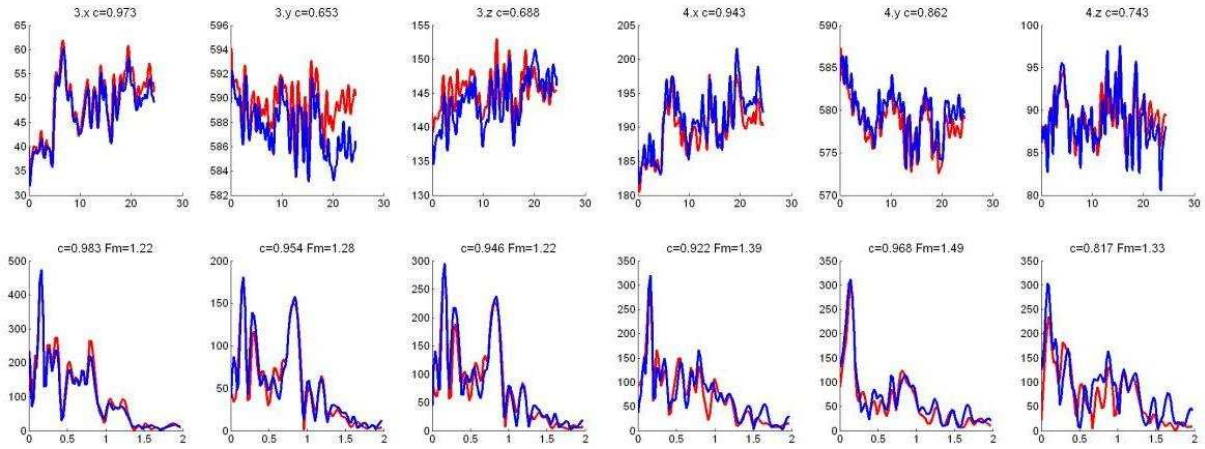
Once video recordings has been obtained, suitable methods are required to extract the observed tree motion. The optical flow of a video sequence is the pattern of motion of the pixels in the images. Simply put, it is a vector field defined over all the pixels (and for each image) that represents the projection of the motion of the scene in the image plane of the camera. Methods to estimate a correct optical flow are a very power full tool to an optical motion capture system.

However, it is far from being a solved issue of computer graphics. A huge amount of optical flow techniques exist. Moreover many methods use such an initial optical flow algorithm and try to increase the quality of the estimated motion using some post-processing methods. Considering



(a) Marker 1

(b) Marker 2



(c) Marker 3

(d) Marker 4

Figure 2.11: 3D reconstruction of the four markers position from videos. For each subfigure: The columns are for the x, y and z coordinates respectively; The top row graphs show the trajectories of the markers extracted from magnetic tracking (red) and from 3D reconstruction from video (blue); The second row shows their spectrum obtained using discrete Fourier transform. Data are given in centimeters.

the initial optical flow techniques, most can be classified in three types: comparison of region, comparison of statistical descriptors, and differential methods. In this section we describe each of these approaches. Then in the next section methods that we developed are presented.

2.4.1 Comparison of Region

This is the algorithmically simplest type of motion extraction of video sequence. An example of these is the box matching algorithm. Let R be a region of interest in an initial image I_t at time t, we want to know the displacement d^* of this region to the next image I_{t+1} at time $t+1$, then the box matching algorithm returns the solution:

$$d^* = \arg\min_d \sum_{p \in R} \|I_t(p) - I_{t+1}(p+d)\|^2 \tag{2.3}$$

So basically, it is the displacement that minimizes the difference of pixels color of the image region R. Note that the process to find the minimum is not optimized it-self but instead the algorithm systematically computes the difference for all possible displacements d. To obtain the motion flow over the whole image, this approach simply divides the original image in a set of square regions, possibly overlapping, and repeated the minimization for all regions independently.

Another example is to maximize instead the cross-correlation of the region of interest:

$$d^* = \arg\max_d \frac{\sum_{p \in R}(I_t(p) \cdot I_{t+1}(p+d))}{\sqrt{\sum_{p \in R} I_t(p)^2} \cdot \sqrt{\sum_{p \in R} I_{t+1}(p+d)^2}}$$

This technique is employed in *Particle image velocimetry*, typically used by the physicist community to track motion of particles in fluids. The initial formulation indicates that the correlation method has similar computational cost than the matching approach. However typical application formulates it in the spectral domain, using Fourier Transform, which greatly reduces the required computations in trade with algorithmic complexity and stability. In some situations, the correlation method obtains better results. However practical experience shows that the size of the region R can have a strong influence on the quality of the results. It thus enforces careful manual configurations to reach an optimal quality which is usually done by trial and error.

2.4.2 Statistical Descriptors

The main idea of these methods is to use some statistical descriptor of the neighborhood of the pixels to be tracked (such as histogram of colors) instead of just the color. They are usually more interesting for region of bigger size and, depending on the algorithm, they address more complex motions than just translation without additional cost.

Mean Shift Tracking

In this technique [CRM00] the motion of a particular region of an original image is estimated by comparing histograms of colors using the original mean shift algorithm. Let an image region

have n pixels and a partition of all possible colors in k bins B_i. A color histogram is described by a k-dimensional vector, say h, that contains the percentage of color following the partition:

$$h_i = \frac{\text{number of pixel of color in} B_i}{n}$$

To track a particular region of an initial image, having a color histogram $\mathbf{p}$, the mean shift tracking looks for a region in the new image having an histogram $\mathbf{q}$ that maximizes the *Bhattacharyya* coefficient ρ:

$$\rho = \sum_i \sqrt{\mathbf{p}_i \cdot \mathbf{q}_i}$$

The algorithm uses mean shift to compute an optimization over this coefficient (see section 2.5.2 for details on mean shift optimization). As mean shift tracking is using color histograms, it enables tracking of object with complex transformations and image noise. However for some types of image textures, such as foliage where any parts have very similar color histograms, this scheme is not efficient.

SIFT

Lowe [Low99] proposed the *Scale Invariant Feature Transform* (SIFT) and developed further this idea of statistical descriptor. The algorithm proceeds first by detecting a set of robust descriptors in two images and maps most similar pairs. This algorithm was initially developed for object recognition: An object is represented by a set of precomputed descriptors, and often only a few are necessary to detect the object in any images. However if the chosen pair of images are those where the motion flow has to be estimated from, then the set of mapped descriptors represents a robust subset of the overall motion flow between these images.

This method has three main parts. First the set of representative features are detected in the images, then a statistical descriptor is computed for each of these features and finally they can be matched together. To detect interesting features in an image I, it first computes a set of images $L(\sigma)$ for various values of σ:

$$L(\sigma) = (G(\sigma) - G(k \cdot \sigma)) \otimes I \qquad k > 1$$

where $G(s)$ is the zero-centered Gaussian with standard deviation s (see figure 2.12) and where $\otimes$ is the convolution operator. The features are then chosen at the local maximum and minimum of these $L(\sigma)$ images. A second step consists in removing detected descriptors with low contrast (strongly subjected to the influence of noise) and those found on edges (for similar reasons as for the KLT tracker described in section 2.4.3).

A descriptor is defined for each detected feature as a vector containing the histogram of the image gradients found in its neighborhood. The main goal of SIFT is to be invariant to scaling,

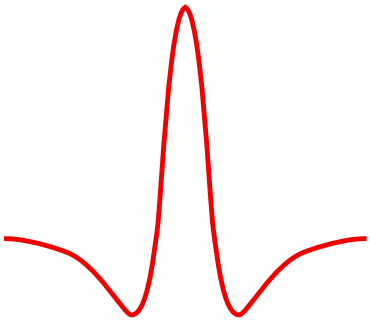


Figure 2.12: Difference of Gaussians ($G((\sigma) - G(k \cdot \sigma))$).

rotation and change of illumination (as well as robust to other minor affine transformations). To this end, all data stored by the descriptors are taken from the scale it has been detected from. If a feature represented by one descriptor in an image, appears in an other image but with a different scaling, a descriptor containing the same data should be detected at the corresponding scale. The matching of both descriptors can thus be done independantly of their scaling. Moreover all gradients are stored with their direction relative to the strongest gradient direction, achieving rotation invariance. Finally these gradients are normalized to achieve invariance to change of illumination.

Once all the descriptors for a pair of images are computed, they are matched together using some scheme to avoid computing comparisons of all possible pairs. The main advantage of this technique is the invariance to many image transformations and lighting. However during my Ph.D., quick experiments on videos of animated foliage have been done and did not produce very satisfying results. As for most video tracking method, the quality of motion estimates in difficult parts of the image sequences was not higher than for other tested methods and took longer computation time. On the other hand, improvements of the initial approach have been produced since the experiments. In particular, the matching process could probably be improved to suit more adequately the goal of video motion estimation. A more careful investigation could exhibit interesting possibilities.

2.4.3 Differential Methods

To compute the optical flow between two images, a common assumption is the conservation of pixel intensity. Some techniques have provided possible algorithms that are not assuming this hypothesis but it usually strongly reduces efficiency. However, by using it one may derive a mathematical formulation of the optical flow problem that has the form of a simple differential equation. Let I be the video volume such that $I(x,y,t)$ is the pixel at coordinate (x,y) in image at time t. The conservation of intensity means that:

$$I(x,y,t) = I(x+dx, y+dy, t+dt) \tag{2.4}$$

for pixel motion (dx, dy) between the image at time t and $t+dt$. Computing the optical flow that reduces to find the suitable displacement $(u, v) = (dx/dt, dy/dt)$ for each pixel. The differential methods are algorithms that propose a solution using the Taylor expansion to the first order of equation (2.4):

$$I(x+dx, y+dy, t+dt) \approx I(x,y,t) + \frac{dI}{dx}dx + \frac{dI}{dy}dy + \frac{dI}{dt}dt \tag{2.5}$$

from equation (2.4), this reduce to:

$$\begin{aligned} \frac{dI}{dx}dx + \frac{dI}{dy}dy + \frac{dI}{dt}dt &= 0 \\ \frac{dI}{dx}\frac{dx}{dt} + \frac{dI}{dy}\frac{dy}{dt} + \frac{dI}{dt}\frac{dt}{dt} &= 0 \\ \frac{dI}{dx}u + \frac{dI}{dy}v + \frac{dI}{dt} &= 0 \\ I_x\, u + I_y\, v + I_t &= 0 \end{aligned} \tag{2.6}$$

We are left with one equation and two unknowns u and v. This underconstrained problem is called the *aperture problem*. Many algorithms have proposed a solution using hypothesis on the neighborhood. The two best known are from Horn and Schunk [HS80] and from Lucas and Kanade [LK81].

Horn and Schunk

This algorithm proposes to solve equation (2.6) using iterative scheme that minimizes:

$$f = \int\int (I_x u + I_y v + I_t)^2 + G(u,v)^2 \quad dxdy \tag{2.7}$$

$$G(u,v) = \alpha\left[\left(\frac{du}{dx}\right)^2 + \left(\frac{du}{dy}\right)^2 + \left(\frac{dv}{dx}\right)^2 + \left(\frac{dv}{dy}\right)^2\right] \tag{2.8}$$

where $G(u,v)$ is a regularization term that basically enforces a smoothness of the motion flow and propagates information to solve the aperture problem. α is a parameter of this method that is basically proportional to the expected smoothness of the optical flow.

The main limitation of this algorithm is that it is necessary to compute the motion flow for the full image to get the solution for one pixel.

KLT tracker

Another solution has been proposed by Lucas and Kanade. Let us first rewrite equation (2.6) as:

$$(I_x, I_y) \cdot (u, v) = -I_t$$

In this method, to solve the aperture problem the motion flow is simply considered constant in a neighborhood. Thus, for each pixel of the image we have:

$$\begin{pmatrix} I_x^1 & I_y^1 \\ I_x^2 & I_y^2 \\ \vdots & \vdots \\ I_x^n & I_y^n \end{pmatrix} \cdot (u, v) = \begin{pmatrix} -I_t^1 \\ -I_t^2 \\ \vdots \\ -I_t^n \end{pmatrix} \tag{2.9}$$

where the indices indicate the n neighbors of the pixels. We have now an overconstrained set of equations that can be solved using least-square. As it is, this technique has the advantage to be very quick and every pixel motion can be computed independently. But it has also the disadvantage that if the gradient information in the neighborhood is redundant then the solution becomes unstable. Let A be the matrix:

$$A = \begin{pmatrix} I_x^1 & I_y^1 \\ I_x^2 & I_y^2 \\ \vdots & \vdots \\ I_x^n & I_y^n \end{pmatrix}$$

Then A needs to be well conditioned to provide a stable solution by least square. Typically if the gradient is null for all the pixels of the neighborhood (*i.e.,* uniform color, $A^T A$ has no strong eigenvalue) or if all of them have the same gradient (*i.e.,* unidirectional blend of colors, $A^T A$ has only one strong eigenvalue) then equation (2.9) cannot provide satisfying results (see figure 2.13).

The consequences is that it is useless to solve equation (2.9) for the pixels that are not corners. Shi and Tomasi [ST94] have defined a simple criterion to select the correct pixels to track. In short they compute the lower eigenvalue of the matrix $A^T A$ for each pixel of the original image, and select the local maximums of these as the features to track. Both techniques [LK81] and [ST94] are often used together. This process is called the KLT features tracker (for Kanade Lucas and Tomasi).

Iterative and pyramidal KLT tracker

Bouguet [Bou00] developed the initial Lucas and Kanade optical flow estimator to provide a more useful tracking algorithm. The first limitation of the KLT tracker is coming from the initial

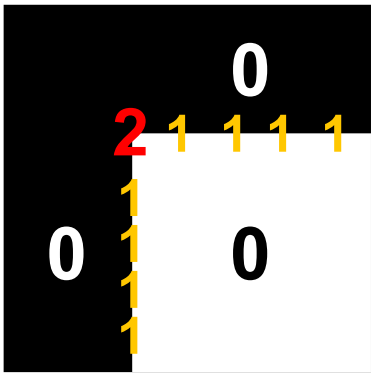


Figure 2.13: A simple image where the numbers of non-zero eigenvalue of A^TA are indicated. Over the black or white area, the gradient is null. In equation (2.9), the matrix A contains only zeros and the eigenvalues of A^TA are both null. For each edge (horizontal or vertical) all the pixels have the same gradient, thus the rows of A are equal and A^TA has one non-zero eigenvalue. Finally at the corner between both edges, some pixels of the neighborhood (the rows of A) have a strong vertical component and some have a strong horizontal component. Then A^TA is well conditioned and has two strong eigenvalues.

linear approximation given in equation (2.5). Compared to an optimization algorithm to minimize a function (equation (2.4) in our case), using linear approximation is quite common. For example the gradient descent method used it to compute the update at each optimization step. Similarly Bouguet proposes to use iteratively the KLT tracker to provide a better estimate of the optical flow.

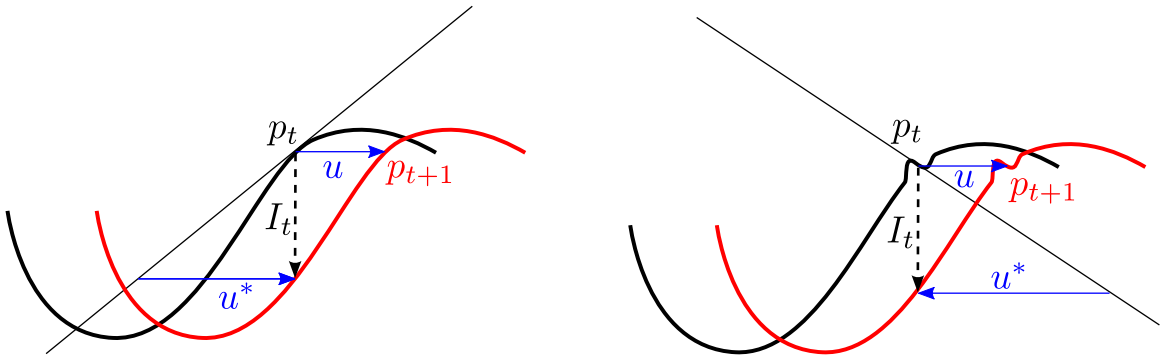
The other important improvement is the use of KLT tracking to several level of resolution of the images. Because of the linear approximation the original algorithm is very sensitive to high frequencies of the images intensity (see figure 2.14). In [LK81], the authors indicated already that smoothing the images can increase the quality of the tracking. In [Bou00] the KLT tracking is first applied on a lower resolution of the images, thus high frequency are not interfering with the tracking and a coarse estimation of the motion is obtained quickly. Then it is refined successively in the higher resolution until reaching the initial images resolution.

2.4.4 Optical Flow and Particles Tracking

Depending on the application, one may prefer to track specific features over several images instead of only extracting the motion flow between pairs of images (see figure 2.15).

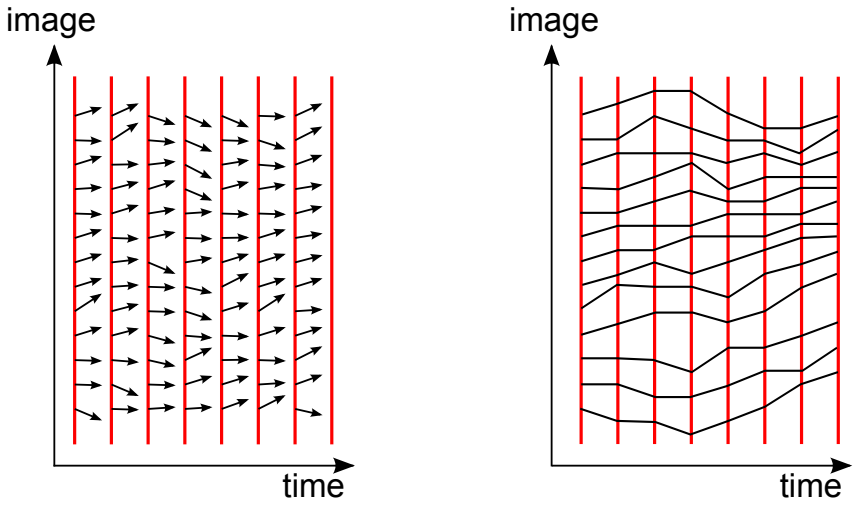
Using any optical flow, one may derive the estimated object motion by simply letting the feature follow the flow. However, it is not necessary to compute the motion of pixels that are not part of the tracked object. Methods such as Horn and Schunk then require more computations than strictly needed as it is necessary to compute the flow over the whole image.

However for particles tracking, two important limitations may produce erroneous data. First because of noise each motion estimate is not exact and the deviations from the correct motion often



(a) smooth curve (b) local high frequencies

Figure 2.14: Graph picturing the Lucas-Kanade method in the one-dimensional case (thus, without the aperture problem). The estimate u^* of the displacement u of pixel p is obtained using equation (2.6), *i.e.,* in 1D $I_x u^* - I_t = 0$. For both graphics (a) and (b), the black curve represents the intensity of the original one-dimensional image at time t and the red curve is the same image at time $t+1$ that moved to the right. (a) The image is smooth, and the estimate u^* of u at p is good (but too long). An iterative process should converge. (b) A deformation of high frequency with low amplitude has been added at p. As a consequence the estimate u^* is in the opposite direction of the real displacement. A divergence of iterative estimation is most probable.



(a) Optical flow (b) Particles tracking

Figure 2.15: Case of a sequence of one-dimensional images. Each vertical line (red) represents one image. In (a) for each pixel of each image, an estimation of the motion is computed. In (b) particles are initiated in the first image then tracked recursively along the image sequence.

accumulate along the video. The motion path then diverges. Second, when a tracked object in the scene is partially or fully occluded then a correct solution cannot be found using direct estimation of the pixels motion. All the particles following the hidden object are then set on another

element of the scene (typically on top of what hides the object). Then they follow this element afterwards for the rest of the video sequence. Moreover, this change of aim also occurs because of the accumulated errors (motion path divergence), especially in complex video such as those of animated foliage.

2.5 Algorithm Development

During my Ph.D., we aimed at using videos as main acquisition device because of the ease of deployment during experimental process. The main difficulty of such an input system is that motion of foliage in a video is particularly hard to track. Differential motion estimators require the pixels displacement to be short enough in relation to the main frequencies of the images luminosity. However foliage are made of leaves that appear tiny on the image while there are the main features that motion can be estimated from. Furthermore, any part of the foliage tends to have very similar appearance. Statistical descriptors are then difficult to define such that they can differentiate patches of leaves from one another. The quality of a direct color comparison of such patches is also restricted by their similarity. Finally many other factors interfere with the tracking algorithm such as change of lighting and occlusion.

From experience, it is possible to find some vision algorithms that give better results depending on the video and its subject. But there is no specific approach that provides a better quality of the estimated optical flow for all videos of any types of animated foliage. Thus, in a practical point of view, several algorithms should be available when extracting tree motion from any video.

An important part of the software development done during my Ph.D. was aiming at providing a sufficiently large set of vision algorithms. In particular algorithms running at interactive frame rate were explored. The implementation framework was based on the QT library [QT] for the development of the user interfaces and the OpenCV library (for Open Computer Vision) [OPE] to implement the vision algorithm.

First the Horn and Schunk and the Lucas and Kanade optical flow were implemented. Due to the poor quality of the motion flow obtained for the videos we intended to use, the ameliorated KLT tracker provided by Bouguet became for a time the main technique employed.

A few additional methods to this first set of algorithms were developed. For practical constraints, object tracking algorithms should be robust yet running at interactive frame rate.

2.5.1 Pyramidal Box Matching Tracking

In a general manner, the box matching approach (see section 2.4.1) provides the best initial estimate as it looks for the image region that fits best a tracked area. This method should then be available in a video motion extraction framework. However as mentioned above, the main limitation is the computation time required because of the necessary comparison of image regions

with all possible displacement. To achieve quick tracking a pyramidal implementation of the basic box matching algorithm was developed.

Let $\mathcal{R}_0$ be the region in the first image we want to track and $\mathcal{R}_X$ the region in the next image at position $X = (x, y)$. These textures can be represented by a $n \times m$ matrix, and for any region $\mathcal{R}$, $\mathcal{R}(i, j)$ is the pixel at position (i, j) in $\mathcal{R}$. The box matching algorithm does the minimization:

$$X^* = \arg\min_X \|\mathcal{R}_0 - \mathcal{R}_X\| \tag{2.10}$$

$$\text{with} \qquad \|\mathcal{R}\| = \sum_{i=0}^{n} \sum_{j=0}^{m} |\mathcal{R}(i, j)|^2$$

This minimization itself is not optimized. The selection of the best solution is done after computing all possibilities. If the search window (*i.e.,* the set of possible X) is $p \times q$ pixels large then the computation cost of the algorithm is $n \times m \times p \times q$. The algorithmic complexity is then $\Theta(n \cdot m \cdot p \cdot q)$. For example, if R_0 is 20 pixels square and the search window is 30 then 360000 pixel comparisons are necessary for each tracked region of each image of the video sequence.

To accelerate this algorithm, we can use pyramidal images (see figure 2.16). It means a set of images computed from the original ones that have a lower resolution (the smaller is the resolution, the higher the image is in the pyramid). Typically, the image of a level of the pyramid I^{k+1} has its dimensions equal to half of those of the image at previous level of the pyramid I^k. Usually one pixel of image I^{k+1} has a value computed as an average of the corresponding pixels in I^k, such as:

$$I^{k+1}(i, j) = \frac{1}{4} \cdot (I^k(2i, 2j) + I^k(2i + 1, 2j) + I^k(2i, 2j + 1) + I^k(2i + 1, 2j + 1))$$

In the image of level k of the pyramid, both the search window and the size of the target region are divided by 2^k. The complexity of the box matching algorithm is thus reduced to $\Theta((2^{-4k} n \cdot m \cdot p \cdot q)$. The value of k is chosen according to the size of the target region. From the solution of equations (2.10) at level k, the solution at the level $k + 1$ can be obtained by repeating the minimization only for the four corresponding positions.

However, because of the loss of information due to reduction of resolution, keeping only the position with minimum color difference at some level may not always lead to the best solution at the lower pyramid level. The best matching region at the lower level should still match well the region R_0 at lower resolution. Thus keeping the set of pixels with least color difference, such as the best 5 percent, is enough.

2.5.2 Robust Tracking

As mentioned above, videos of animated foliage are particularly difficult cases for optical flow algorithm. The main limitation of differential approaches is the noise present in the extracted

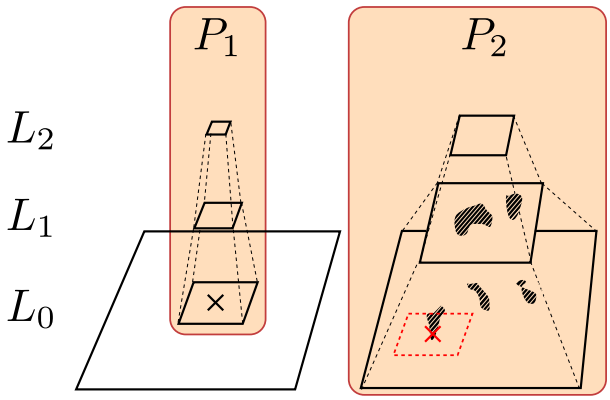


Figure 2.16: Pyramids of images used to accelerate the box matching method. L_0, L_1 and L_2 are levels of the pyramids (of depth three) from 0 for the full resolution to 2 for the lower resolution. P_1 is the pyramid of the region to be tracked in the first image (R_0 is marked with an $\times$ in its initial resolution). P_2 is the pyramid of the search window in the second image. Once the box matching algorithm as been computed for all the positions in P_2 at level L_2, the best displacements (ex: the 5% with minimum image difference) are projected in the level L_1 (represented by the striped region). At the lower level, the box matching is done over these pixels only, and best displacements are further projected. At level L_0 the displacement with minimum image difference is taken as solution (the red region marked with a red $\times$).

speed. Typically this noise has not a homogeneous repartition. It is either small for correct estimations or produces strong outliers. Extracted motion cannot be considered trustworthy at all pixels. However for a particular region to track, the estimated displacements of all the pixels can represent a set of stochastic draws of the probability density function (PDF) of the region motion. A statistical approach can then be used to extract a more valid estimation of the motion. We now describe a robust estimation technique of an image region motion using such a method that rejects outliers and maximizes expectation.

We use the iterative and pyramidal KLT tracker to get a first estimation of the motion of the pixels of the region of interest. This algorithm has the advantage to provide a good results sample containing mostly correct motion estimations. However any method is suitable, as long as a set of n displacement vectors x_i are obtained. We use the *multivariate kernel density estimators* introduced by Parzen [Par62] as an estimation of the continuous PDF of the region motion. Let k be a kernel function. For two-dimensional variables the density estimator with kernel k and radius h is of the form:

$$\tilde{f}_k(x) = \frac{1}{nh^2} \sum_{i=1}^{n} k(\frac{(x_i - x)}{h})$$

As Fukunaga and Hostetler [FH75], we use the mean shift algorithm to perform a gradient ascent and find the displacement x that maximize $\tilde{f}_k(x)$. This is a tracking algorithm similar to the one developed by Comaniciu *et al.* [CRM00], however we are not comparing color histogram so the

Bhattacharyya distance is not appropriate. Instead we use the *Mahalanobis* distance that is often used for outlier detection in noisy data. Let μ be the sample mean and Σ the sample covariance matrix, the Mahalanobis distance from any point x to μ is defined by:

$$\|x-\mu\|_{Mahalanobis} = \left((x-\mu)^T\Sigma^{-1}(x-\mu)\right)^{\frac{1}{2}}$$

For the kernel we take a radius parameter $h = 1$ for simplification but its influence is compensated by the use of the Mahalanobis distance as it is shown further. Let then $k(x)$ denote the kernel function:

$$\begin{aligned} k(x_i - x) &= exp\left(-\frac{1}{2}\|x_i - x\|^2_{Mahalanobis}\right) \\ &= exp\left(-\frac{1}{2}(x_i - x)^T\Sigma^{-1}(x_i - x)\right) \end{aligned}$$

Mean shift iteratively estimates the gradient of $k(x)$ to find x that maximize k using a gradient ascent algorithm. The idea is to consider the estimate of the PDF gradient as the gradient of the estimated density function:

$$\begin{aligned} \tilde{\nabla} f \equiv &\quad \nabla \tilde{f_k} \\ \nabla \tilde{f_k} = &\quad \frac{1}{n}\sum_{i=1}^{n} \nabla k(x_i - x) \\ &= -\frac{1}{2n}\sum_{i=1}^{n} k(x_i - x)\cdot(x_i - x)^T\Sigma^{-1} \\ &= -\frac{1}{2n}\left[\sum_{i=1}^{n}(k(x_i - x)\cdot x_i) - \sum_{i=1}^{n}(k(x_i - x)\cdot x)\right]^T \Sigma^{-1} \\ &= -\frac{1}{2n}\sum_{i=1}^{n} k(x_i - x)\left[\frac{\sum_{i=1}^{n} k(x_i - x)\cdot x_i}{\sum_{i=1}^{n} k(x_i - x)} - x\right]^T \Sigma^{-1} \\ &= \frac{1}{2}\ \tilde{f_k}(x)\ \mathbf{M}_k(x) \end{aligned}$$

$$\text{with} \qquad \mathbf{M}_k(x) = -\left[\frac{\sum_{i=1}^{n} k(x_i - x)\cdot x_i}{\sum_{i=1}^{n} k(x_i - x)} - x\right]^T \Sigma^{-1}$$

Then the mean shift procedure is defined recursively by computing the mean shift vector $\mathbf{M}_k(x)$ and translating x by $\mathbf{M}_k(x)$. At start we initialize x^0 as the mean of all the x_i and compute the correlation matrix Σ^0 such that:

$$
\begin{aligned}
x^0 &= \frac{1}{n}\sum_{i=1}^{n} x_i \\
\Sigma^0 &= \frac{(X^0)^T(X^0)}{n-1} \\
\text{with} \qquad X^0 &= \begin{pmatrix} x_1^T & \dots & x_n^T \end{pmatrix}^T
\end{aligned}
$$

Then, at each iteration of the procedure, we use the unbiased weighted sample covariance to narrow the radius of influence of the kernel:

$$
\begin{aligned}
x^{k+1} &= x^k + \mathbf{M}_k(x^k) \\
\Sigma^{k+1} &= (X^k)^T W^k (X^k) \\
\text{where} \qquad W^k &= \begin{pmatrix} w_1 & 0 & \dots & 0 \\ 0 & w_2 & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \dots & \dots & w_n \end{pmatrix} \\
\text{with} \qquad w_i &= \frac{w_i^*}{(\sum_{i=1}^n w_i^*)^2 - \sum_{i=1}^n (w_i^*)^2} \\
\text{and} \qquad w_i^* &= \frac{k(x_i - x)}{\sum_{i=1}^n k(x_i - x)}
\end{aligned}
$$

This algorithm has the advantage to cancel outliers influence while it does not *smooth out* the extracted motion as a simple average of the x_i would. However it uses results from a first tracking algorithm. If this original motion estimation is of too poor quality then any approach of this type cannot provide satisfying results.

2.5.3 Tracking Particles with Limited Life Span

To track an object moving in a video sequence, a set a initial features defined over this object in the first image can be tracked iteratively from image to image. Then the target object trajectory can be defined by the motion of these features. The main limitation is that, what ever the vision algorithm chosen for the features tracking, errors accumulate along the video and divergence from the correct motion path often occurs.

On the other hand, errors are usually uncorrelated with the correct motion. A feature which has a similar estimated motion over several frames has then a higher probability to be tracked correctly.

From these two observations, a tracking system has been developed to estimate the motion flow of the video with a set of particles having limited life span. Then for any particle at any image

estimated motion can be compared over a few frames checked for sufficient correlation. In comparison to the previously described techniques that discard motions uncorrelated in space, here we want to reduce influence of motions uncorrelated in time. A simple approach is to weight the influence of particles by a function proportional to inverse of the difference in speeds between several frames. If s_1 and s_2 are the speeds of a particle starting at the first and second image respectively, then we can use the function:

$$w_i = exp(-\|s1 - s2\|^2)$$

Our method starts by defining particles covering the first image, for example using KLT features detector. Then these features are tracked until they are considered lost. Then at each frame, new particles are added to replace deleted ones. However it is important to obtain a motion flow that covers the entire video. The density of particles in each image has thus to be maintained. Instead of using a complex dynamic distribution algorithm such as proposed by Vanderhaeghe *et al.* [VBTS07], we simply select the starting position of the new features using the Shi and Tomasi method [ST94] (see *KLT tracker* in section 2.4.3). However this technique is only applied over areas of the image that are distant enough from the features that are still tracked.

chapter 3

User Oriented Application for Video Tracking

Contents

3.1 Introduction **46**
3.2 Visualization of Motion Flow **47**
3.3 User Interface to Control Time **49**
3.4 Manual Input **51**
3.4.1 Key-Frame Approach 51
3.4.2 Sketching Motion 53
3.5 User Interface for Correction **55**
3.5.1 Correction in Space 56
3.5.2 Time Correction 57
3.6 Conclusion **58**

3.1 Introduction

However complex a vision algorithm may be, it will not work everywhere and the estimated motion flow cannot be considered always reliable. For some type of application, such as surveillance system, automatic techniques are required. But for our problem we need instead as much confidence in the extracted motion as possible. To this end, human interaction is required to at least inspect the result and most probably correct it. Moreover, for situations where tracking algorithms fail, manual methods is necessary to easily input the whole motion data.

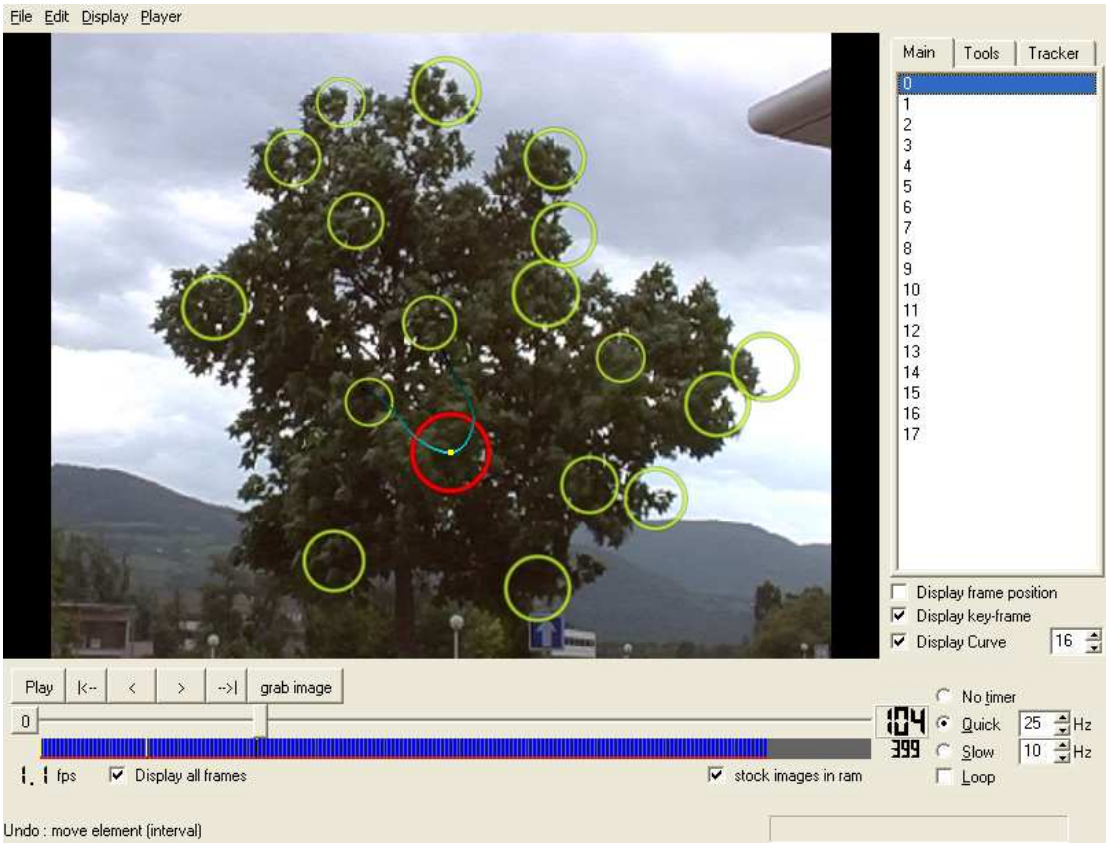


Figure 3.1: Software to extract motion in video using both automatic tracking and user interaction. The circles drawn on top of the video image are the features to track.

The work presented in this thesis do not aim at using optical flow (*i.e.,* the motion vector field for each image of the video as defined in the previous chapter). Instead we are interested in the extraction of the motion path of specific features, such as a piece of the foliage that is moving as a whole. To this end, we developed interaction schemes that use automatic tracking algorithms, but also focus on intuitive user interface to correct extracted motion paths, and allow manual inputs of a target trajectory (see figure 3.1).

In figure (3.2) a chart is given that described the typical user workflow to extract motion of target objects in a video. The element indicated by (*) in the chart is the starting step. It requires user visual evaluation to select a feature to track. This task is done after a brief overview of the video.

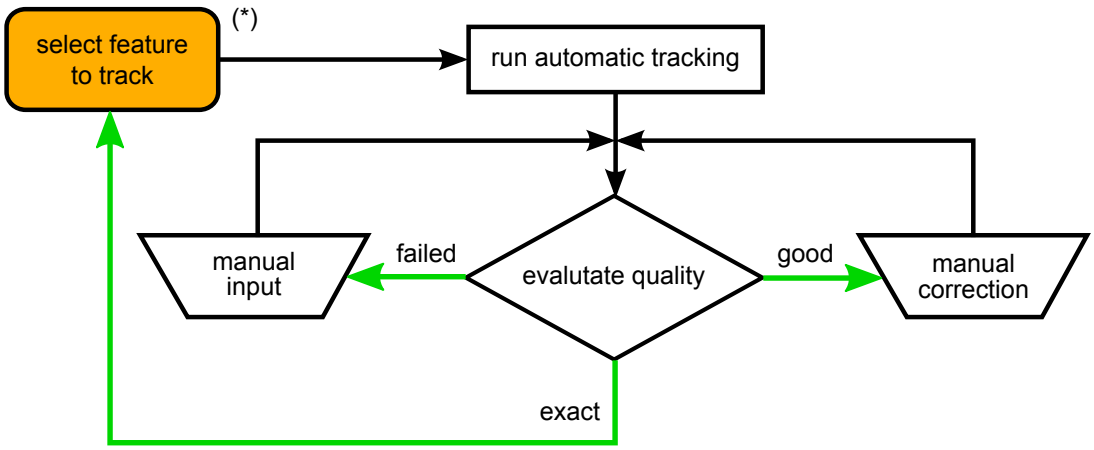


Figure 3.2: Flowchart describing the organization of user's work using our application for the task of motion extraction of selected targets of the video.

In section 3.2 we describe a visualization tool used to help this selection procedure. Then the first action is to run some automatic vision algorithm such as the techniques described in the previous chapter. Once again user evaluation is necessary to assess to quality of the tracking. According to this quality, the user has three choices. If the tracking is good enough then the user can start a new feature tracking, thus come back to the first step of the workflow (*). If the quality is mostly correct but the extracted path contains some errors, then manual corrections have to be applied. A description of the tools provided for this correction step is given in section 3.5. Finally, if automatic tracking totally fails, or if it requires too many corrections, then the initial input should be done manually. Section 3.4 describes the interface for manual input of motion trajectory available in the presented application.

3.2 Visualization of Motion Flow

A specific problem in working with videos of animated foliage is the difficulty of the visual recognition of the motion structure in still images. Most often, it is almost impossible to see where are the underlying branches when the foliage is not moving. A tool showing information that emphasizes motion structure is thus very useful.

Motion flow is usually displayed in one of two ways.

- The first possibility is to draw a little arrow for each motion feature. But when too many of these arrows are drawn visual perception of structure become complex, especially if the motion estimations are noisy.
- The second common method is to render motion with a color where the speed in the x and y axis are mapped to two of the three color channels of an image: red, green or blue.

Once again this mapping is not perceptually suitable for structure recognition as the color domain is too restricted, *i.e.,* not spanning enough the color spectrum.

Similarly to the second approach, we propose a method where the motion is displayed by a color representation. However instead of using only two color channel of an *RGB* color space, the motion is mapped onto the *HSL* color space (which stands for **H**ue, **S**aturation, **L**ightness, see figure 3.3) which has a topology well adapted to the represention of speed information.

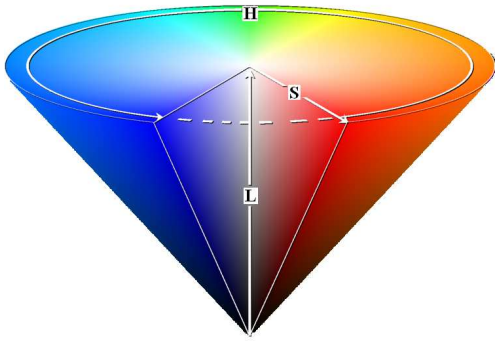


Figure 3.3: The conical representation of the HSL color space. The hue coordinate (H) is the radial position around the cone, the saturation (S) is the radius to the axis of the cone and the lightness is the height of the cone.

The mapping we use is defined from the polar representation of the speed vector to the outer surface of the HSL conical space, *i.e.,* saturation is always one. Slow motion are displayed in black, and high speed are strong, not saturated colors. Then the radial coordinate in polar representation of speed is directly used as the hue coordinate.

An other important factor can be taken into account: the information on motion structure is mostly contained in fast movements. As this interface for visualization is drawn on top of the video images, the lightness (*i.e.,* the speed) is also mapped to the alpha channel of the displayed colors. Thus the features in the video appear as changing color when they are moving, and structures with similar motion reveal themselves (see figure 3.4). Let θ and ρ be the direction and norm of the speed vector, then the displayed color with coordinates H, S, L and α (respectively for hue, saturation, lightness and opacity) are:

$$\begin{aligned} H &= \theta \\ S &= 2 \\ L &= min(c\rho, 1) \\ \alpha &= L \end{aligned}$$

where c is a user controllable parameter.

(a) (b)

Figure 3.4: Superposition of motion information using speed to HSL mapping. (a) Original image from a video sequence. Structure is *invisible*. (b) With the motion information, the red circle indicates an emphasized structure.

When most of the scene in the video is in motion, the described visualization tool then shows a colorful image containing finally too much information to easily perceive the organization of motion. To help extract specific structures, controllable parameters are provided to let the user restrict the display of motion information to a specific direction θ^*, and range $\delta\theta$ from this direction. Then the only speed vector to be display must respect the rule:

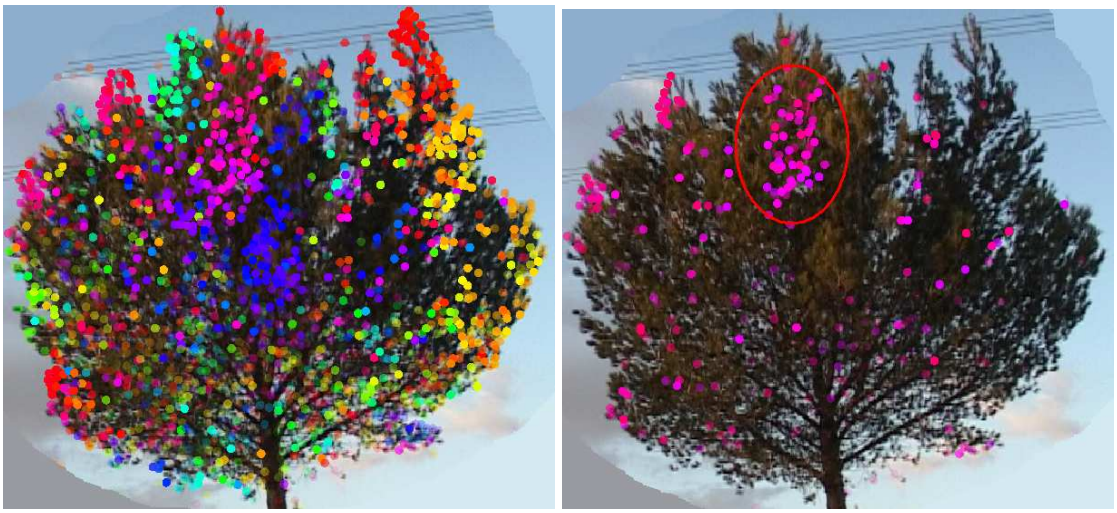
$$\left\|\theta - \theta^*\right\| \leq \delta\theta$$

where the norm $\|\cdot\|$ is the euclidean distance in the circular space of angles. Combined with the parameter c defined above, the interface allows to easily emphasize structures in the video (see figure 3.5).

3.3 User Interface to Control Time

An important part of the user interface is the video player which should be made of a simple, yet precise, set of controls (see figure 3.6). For fast inspection, correction and manual edition of motion information, a user needs first to be able to shift back and forth along the video easily.

For the development of a user interface, controls have to be designed to take into account the possible input hardware. Using a typical computer, these are the mouse and the keyboard. In the case of an application suitable for the extraction of video motion using manual interaction, the mouse should mainly be used for the editing process. The user should then be able to control the time index of the video with the keyboard only.



(a) (b)

Figure 3.5: Using direction clamping on displayed color. (a) With everything displayed. (b) The display is restricted to features with a specific motion direction. One structure is pointed out (red circle).

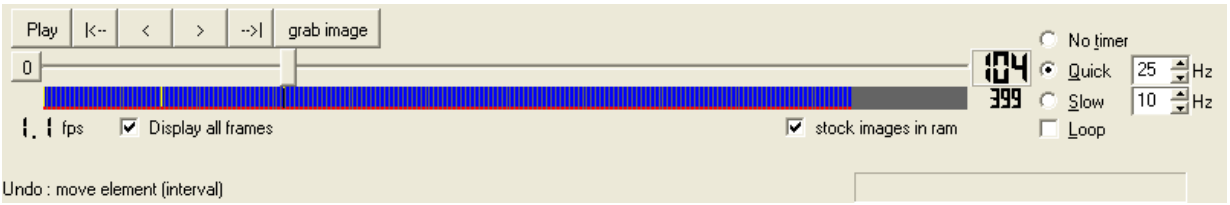


Figure 3.6: Graphical interface of the video player. On the right, the user can configure the two frame-rates (quick and slow) at which the video can be played.

Looking at a traditional video player, only few controls over the displayed frame are provided: the *play/pause* button and the *stop* button (*i.e.,* return to the first frame of the sequence) and often a cursor indicating the current image that can be displaced manually. But only the button can be directly replaced by keyboard interaction. In our application we kept these but also designed more refined controls. The idea is to provide three different speeds for the video to be played both forward and backward in time. The first speed is the set composed of the simple *next* and *previous* image button, then the two others are controled by the *play/pause* button for which the frame rate can be configured (initially real-time and half this speed). All these buttons can be activated using the keyboard, letting the mouse free for data editing (see figure 3.7). These interface become quickly natural to the user, and let his attention entirely focus on its main goal: extracting correct motion trajectories of target objects in the video.

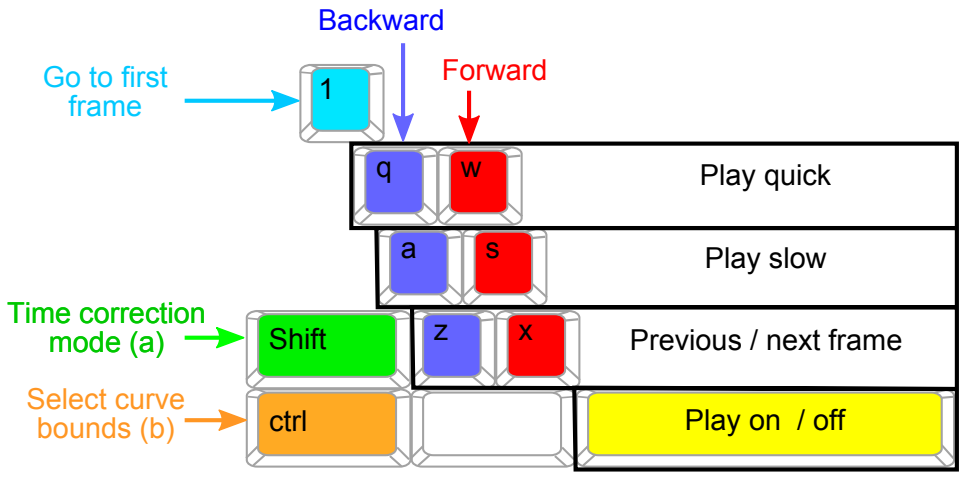


Figure 3.7: Keyboard shortcuts used by our application. The layout of the control buttons allows simple and efficient user interaction. (a) Used by the time correction tool described in 3.5.2. (b) Used by the space correction tool described in 3.5.1.

3.4 Manual Input

When automatic vision algorithms fail, the user needs to manually input the whole motion of the target object. Considering user interface issues, the logical workflow is to first input an initial path that can then be enhanced. To provide simple and quick alternatives to automatic tracking, we propose here two input schemes. The first one follows the traditional key-frame approach used in many commercial software. The second proposes an innovative method to the specific problem of manually tracking animated foliage.

3.4.1 Key-Frame Approach

For this type of interaction, the user input several key-frame positions along the video, then the shape of the motion curve between these positions is defined by interpolation. We use cubic Hermite spline for this task. Let p_0, p_1, s_0 and s_1 be respectively the position and speed of two key-frames at time t_0 and t_1. Then the interpolated position at time $t \in [t_0, t_1]$ is:

$$p(t) = h_{00}(t)p_0 + h_{10}(t)s_0 + h_{01}(t)p_1 + h_{11}(t)s_1$$

using the following Hermite basis functions (see figure 3.8):

$$\begin{cases} h_{00}(t) = (1-2\tau)(1-\tau)^2 \\ h_{10}(t) = \tau(1-\tau)^2 \cdot (t_1 - t_0) \\ h_{01}(t) = \tau^2(3-2\tau) \\ h_{11}(t) = \tau^2(t-\tau) \cdot (t_1 - t_0) \end{cases}$$

$$\text{with} \qquad \tau = \frac{t - t_0}{t_1 - t_0}$$

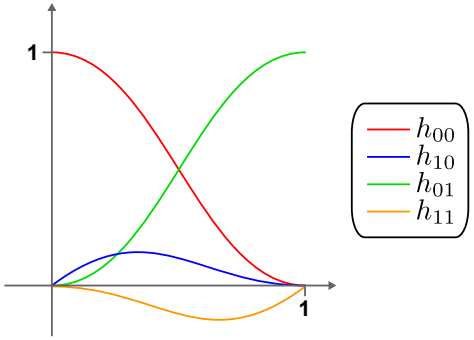


Figure 3.8: The four Hermite basis functions used for cubic interpolation.

The user can easily indicate the positions p_0 and p_1 of the key-frame, but it is more difficult to indicate the speed. To define the speed $\mathbf{s}_k$ at a key-frame at time k, we first look for the previous and next key-frames at time k_0 and k_1 respectively. Let the function $\mathbf{p}(t)$ be the recorded position at time t and let the average speed vectors $\mathbf{s}_0$ $\mathbf{s}_1$ be defined by:

$$\mathbf{s}_0 = \frac{\mathbf{p}(k) - \mathbf{p}(k_0)}{k - k_0}$$
$$\mathbf{s}_1 = \frac{\mathbf{p}(k_1) - \mathbf{p}(k)}{k_1 - k}$$

Then the speed s_k at time k is given by (see figure 3.9):

$$\mathbf{s}_k = \alpha \cdot \mathbf{s}_0 + (1-\alpha) \cdot \mathbf{s}_1$$

$$\text{where} \qquad \alpha = \frac{k_1 - k}{k_1 - k_0}$$

As the motion of foliage results from the underlying branches structure dynamics, it is usually smooth and does not require many key-frame in order to be well estimated. Moreover, when using this approach all user action are done using the mouse. Coupled with the keyboard controls of the video player as described above, it provides a very quick input method for the extraction of target motion path in a video.

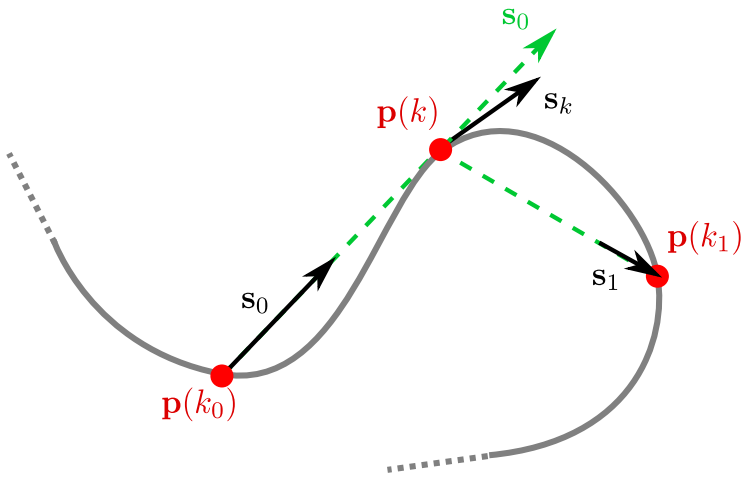


Figure 3.9: An example of the speed estimation for a key-frame at time k. In this example, even if $p(k)$ is almost spatially equidistant to $p(k_0)$ and $p(k_1)$ in the image space, the ratio between time interval is $(k_1 - k) = 2(k - k_0)$, so $\alpha \approx 0.667$: s_0 has twice more influence than s_1.

3.4.2 Sketching Motion

In section 3.2, a perceptive limitation has been described: the difficulty for the user to discern visually the underlying structure of the branches in still image of foliage. This structure is however important as it holds the observed foliage motion. For the key-frame input approach, it often leads to a strange visual phenomena when the video stops playing: In a very short time, the knowledge a user has of the exact location of a target (*i.e.,* part of the foliage) is lost. It then becomes a very complex task to set the key-frame positions exactly.

The perceptive ability of human visual system to extract structure from motion has been extensively studied. A famous experiment, made by Gunnar Johansson [G.06], shows that displaying only a few isolated bright points that were recorded when attached to a walking human, are enough for a viewer to recognized human motion. In our case, the issue is not to recognize the motion but the ability to very quickly infer structure information, and to overcome the perceptive limitation due to the restricted time for such information to be remembered. The interface of the video player (described in section 3.3) can help as it lets the user easily shift the video back and forth in time to maintain perceptive information of motion. However it slows down the overall input process.

This facts leads us to design an interface for manual input while the video is playing. The idea is simple, as the player is easily controlled and does not require much attention, the user can *draw* the target motion while it moves.

In computer graphics, research in sketching interface have proposed interesting manual devices. These works have mostly focus on tools for the modeling of 3D virtual objects [ZHH96, IMT99, SS08]. Some research have also be done on sketch-based interaction as an input device for the control of 3D animation. However these approaches usually aim at fitting motion generated with dynamics simulation to the user's input [Coh92, PSE03, TBvdP07].

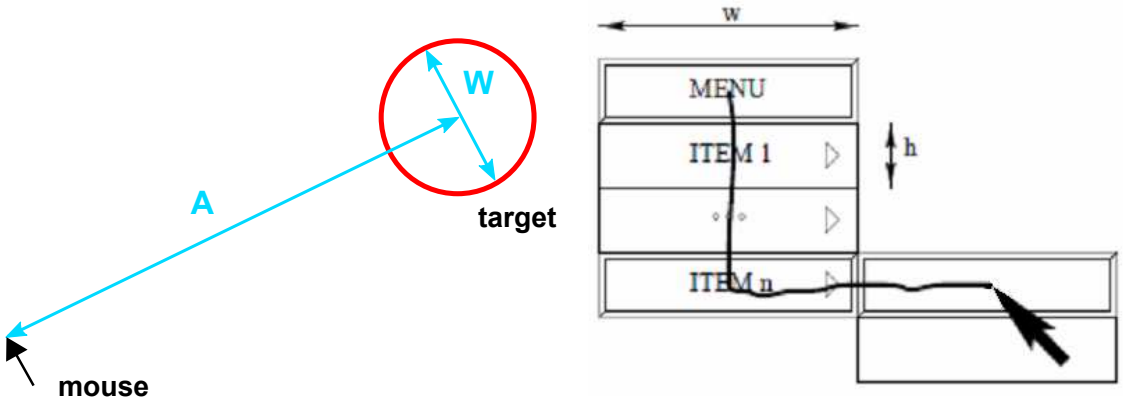
On the other hand, the human-computer interaction community has made researches that provides interesting results that are more related to our problem. Historically, a first model defined by Fitts [Fit54] helps predict the time T required to rapidly move to a target of radius W and at distance D (see figure 3.10(a)):

$$T = a + b \ln(\frac{D}{W}) \tag{3.1}$$

where a and b are two constants depending on the user and the experimental setup. Since this model, several developments have been propose for more complex task. The *steering law* apply to the task of following a specified trajectory of width W [AZ97]:

$$T = a + b \int_{\mathcal{P}} \frac{ds}{W(s)} \tag{3.2}$$

where a and b are two constants similar to those in equation (3.1) and where $\mathcal{P}$ is the path to follow (see figure 3.10(b)). For both Fitts law and steering law, the main point is to describe the complexity of the task as a function of the distance D (for Fitts law) and the width of the target or of the path. For example, in the case of the Fitts law, the task of reaching a target is of logarithmic complexity with respect to distance and inverse of the target width.



(a) Fitts law (b) Steering law (figure taken from [AZ97])

Figure 3.10: Typical tasks to which Fitts law and steering law apply.

In our case, we are interested by the task of pursuit tracking of a moving target. The user plays the video at a selected speed, typically using the slow frame-rate described in 3.3. This is done by one hand, then using the mouse with the other he follows the target.

Some researches has been done in order to model the human tracking as a function of a target trajectory. First, Hoffmann has developed the Fitts model for targets having a constant speed [Hof91]. Then other models have been set for the task of pursuit of moving targets with speed changes unpredictable by the user [AMM90, ES00]. However these researches meant to provide prediction models of the trajectory, represented the user's sketch, from a known trajectory of the

target. Most of the parameters of these models are very dependent of the experiment and the user.

A set of experiments have been done using this manual tracking interface. The main results shows that each user has a different approach. Trying to invert the models by finding good default parameters does not provide good results in all cases. Automatically improving the precision of manual tracking is thus a difficult problem that may not have solution. We still extract and use one of the parameters from [ES00]. The time delay in human response is estimated around $115ms$. So each recorded position of a user stroke is shifted in time by this amount.

Finally, this sketching interface is used as the automatic tracking to record a first motion curve that usually requires some corrections. It is actually a difficult tool in case of fast motion with sudden changes. However branches motion in trees strained by natural wind are almost always smooth and the motion path is fairly predictable. Unlike the automatic tracking the goal is more to get a curve that has a correct trajectory in space than to have the correct timing. Indeed the main difficulty of manual tracking is to respect the exact speed of the tracked feature. In section 3.5.2, we discuss the method we propose that allows to correct the timing afterward.

3.5 User Interface for Correction

The motion path of a target is defined by a set of positions, one for each image of the video. It can be thought of as a 3D trajectory through the video volume (*i.e.,* where the x and y axis are the images coordinate and the z-axis is the time). For editing, instead of displaying only a feature position and its speed at each frame, the motion curve is projected in the image plane to provide more visual information on the features dynamics (see figure 3.11). However only the path over a few neighboring frames are shown as displaying more would be confusing.

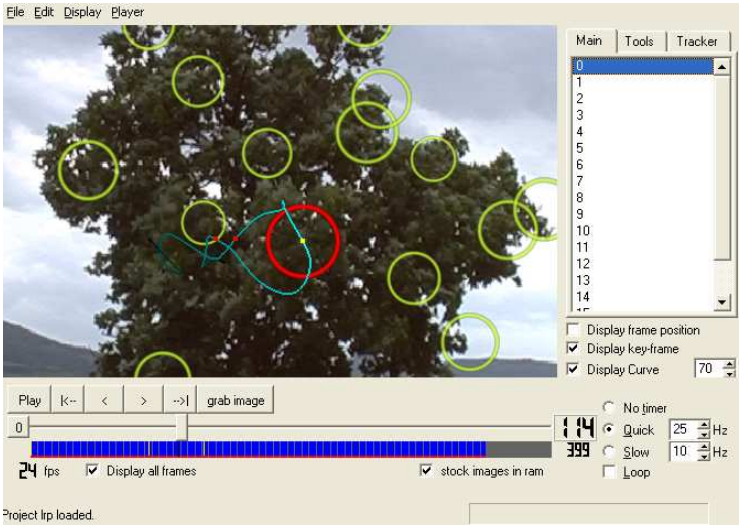


Figure 3.11: Displaying the motion curve of a selected target. The red dots are key-frames.

For the task of tracking target objects, the user naturally segments the work and input the motion curve over one subset of the whole video at a time. For a key-frame initial input, the final curve is a set of key-frame position and interpolated values in-between. For automatic tracking and for manual tracking the user works step-wise. A key-frame separates each input sequence and correction can be required for the in-between frames.

All the corrections are done by the displacement of the position of the motion curve at one frame which then becomes a key-frame. We now describe how this displacement made by the user at one frame is propagated along the motion curve.

3.5.1 Correction in Space

Using the key-frame approach, one can refine the motion curve simply by adding more key-frames where needed, or by displacing one in the case of a previous user error. For both actions, the motion curve only needs to be updated around the new (or displaced) key-frame.

However, to correct curves obtained through automatic or manual tracking, an other approach is necessary. A method was designed to provide the correction that is usually expected. The idea is to use the Hermite spline as defined before to compute the displacement of the motion curve between the next and the previous key-frames if they exist.

The typical error that occurs in automatic tracking algorithms, when it does not fail entirely, is when an estimated feature's motion has diverged slowly from the correct path. On the other hand, for manual tracking, usually the input curve has approximately the correct shape. But it is often shifted locally from the correct trajectory. For both cases, the positional error at two neighboring frames are almost the same. Thus for a correction applied to one frame the neighboring part of the curve should be corrected similarly.

We provide an interpolation process that propagate the correction along the curve. It is thus necessary to know how far this propagation should go. At maximum, the previous and the next key-frame are the bounds, as they should be already correct. If the user want to restrict a curve correction to a shorter segment then he can set as key-frame the last and/or next correct position. To simplify this task, we provide a user tool that shows the part of the curve that is affected by a user correction at the current frame (see button (a) in figure 3.7). Then the user can set any position of the displayed curve as key-frame directly which removes the need of time displacement along the video.

Once the section of the curve to be corrected is selected, a displacement of the feature current position can be interpolated. The most simple propagation scheme is a linear interpolation. However, most often it does not produce satisfying results. In the correction algorithm we provide, the displacement of the motion path is obtained by computing the displacement of a virtual spline.

Let $\mathbf{p}(t)$ be the initial motion curve for $t \in [t_0; t_1]$, *i.e.,* between the the closest key-frame in one of the direction and the position to be corrected. We define a virtual spline $\mathbf{s}(t)$ as defined in section 3.4.1 but instead of using the same speed vectors, the speed of the target at these frames (*i.e.,* $\dot{\mathbf{p}}(t_0)$ and $\dot{\mathbf{p}}(t_1)$) are taken. Then from the displacement made by the user (either at t_0 or t_1), we

compute again the spline $\mathbf{s}^*(t)$ still using the initial speed vectors. Finally, the corrected curve $\mathbf{p}^*(t)$ is obtained by applying the applying the spline discplacement:

$$\mathbf{p}^*(t) = \mathbf{p}(t) + (\mathbf{s}^*(t) - \mathbf{s}(t)) \qquad \forall t \in [t_0, t_1]$$

Figure 3.12: Spatial correction. The user corrects the position at one frame, and it is propagated along the motion curve. In light brown is the curve segment (initial curve) that is affected. The user has previously indicated the bound of this segment on one side. The other bound is the next key-frame (red dot). In light red curve with white points is the corrected motion curve, the points are the frames positions.

3.5.2 Time Correction

For smooth motion, it is easier when using manual tracking to input a motion curve that is spatially correct than temporally. Thus the speed along the curve may not be accurate. The correction tool described just above is not suitable since the shape of motion path is deformed. Moreover, for the task of manual tracking, experiments have shown that the user has more confidence and provide a better initial target's trajectory if it does not try to respect timing. But then a suitable tool that provides quick corrections of such errors is necessary.

First, using a keyboard shortcut, the user can switch to a time correction mode. Then its action are restricted: the possible correction can only be done by displacement along the curve, *i.e.,* a selected target's position at one frame can only be shifted in time. All the points of the motion curve are then moved according to a linear interpolation of the time-shift. After a manual input of an initial motion path, the suitable correction procedure is to go back and locally correct the timing (see figure 3.13).

(a) Backward in time (b) Forward in time

Figure 3.13: Correction of the motion curve only in time. The user can move the features only along the curve. The white dots are the new positions at all frames.

3.6 Conclusion

The application described in this section provides the necessary blend of automatic vision algorithm and user interface to allow motion extraction out of videos with any level of complexity. When computer vision techniques provide good results, then the motion of features in the video can be quickly estimated. The more complex the video is for these techniques the more the user has to contribute. In most cases, only simple corrections are necessary, but even in the worst scenario motion extraction is still available as it can be done entirely by hand. Furthermore the user interface described in this chapter enables efficient inputs. It reduces greatly the user's effort which is a considerable advantage as the task of video motion extraction can become long and complex.

chapter 4

Extraction of Plant Structure and Retargetting of Motion Data

Contents

4.1 Introduction **60**
4.2 Related Works **61**
4.2.1 Structure from Video Motion 61
4.2.2 Animation from Video Motion 62
4.3 Building a Hierarchy of Branches from a Single Video View **63**
4.3.1 Clustering Metric 64
4.3.2 Selection of a Hierarchy of Branches using Clustering 65
4.4 Creating 3D Shape and Motion **69**
4.4.1 Creating Motion of the Terminal Groups 69
4.4.2 Propagating Motion to Intermediate Groups 69
4.4.3 Extending Groups to 3D Shape and Motion 70
4.5 Controlling the Animation of Complex 3D Plant Models **72**
4.5.1 3D Animation of a Plant by Skinning 72
4.5.2 Interactive 3D Modeling 73
4.6 Discussions on our Approach **74**
4.7 Conclusion **75**

4.1 Introduction

The two previous chapters describe methods to extract reliable information on the visible motion of animated trees in video sequences. Here we discuss our work on statistical study of such two-dimensional motion data in order to estimate information on the structure of the animated plants. We show results on four video sequences, two of shrubs and two of trees.

Video motion does not contain all information on the trees dynamics: at best it is the projection of the real movement of the trees onto the video plane. First the depth information in the view frame of the video camera is unknown. But more importantly, only the motion of the visible parts of the tree can be extracted from the video. This restriction is especially important for trees with foliage as the leaves hide most or all the branches.

These facts indicate a need for insightful method to extract information of higher order from the motion observed in video. The inherent 2D projection of data recorded in video sequences is a loss of information on the plants dynamics. But the visible parts of the plants in videos are the most external element of the structures. Their movement is thus influenced by the motion of all the underlying branches. Suitable statistical methods can be developed in order to extract useful data on the hidden parts of the plants. In this chapter we describe such a method that estimate topological information on a plant structure using clustering algorithm applied on the 2D video motion.

First feature tracking, such as described in the previous chapters, is applied on the video footage, allowing the 2D position and velocity of several features to be extracted. These features should cover as much as possible the entire foliage. Then we use statistical clustering to obtain a hierarchical organization of these features. An interesting contribution of our method is the metric employed by the classification algorithm: the *distance of movement* (see section 4.3.1).

A set of branches is then estimated such that each cluster is a groups of leaves (*i.e.,* the tracked features) that all descend from the same branch. However this structure is only topological. Only the position of the terminal leaves are known, not the position of the branches. Using these leaves motion in the video plane, more precise knowledge of the spatial organization of branches structures is then defined from geometrical constraint.

The hierarchy obtained through clustering is used to synthesize a 2D hierarchical geometric structure of branches that terminates according to the cut-off threshold of the classification algorithm. This step extracts both the shape and the motion of a hierarchy of features groups identified as geometrical branches. A simple heuristic allows next to extend the 2D hierarchy to three dimensions by estimating 3D spatial distribution of the features from their 2D positions.

A key contribution of this approach is that the 3D hierarchical structure can be efficiently used as a motion controller to animate any complex 3D model of similar but non-identical plants using a standard skinning algorithm (*i.e.,* mapping of a structure motion to another geometrical model). Thus, a single video source of a moving plant becomes an input device for a large class of virtual plants. To our best knowledge, this is the first markerless motion capture technique that allows the reproduction of real plants motion onto virtual models.

Our method is based on the observation of nature and provides visually convincing results without requiring the simulation of the exact motion of every branch and leaf. Physical phenomena such as wind forces, structural elasticity, or inter-collision of the branches are statistically modeled from video, rather than explicitly simulated physically.

After reviewing related works in section 4.2, section 4.3 presents the algorithm to extract a topological hierarchy of animated branches using statistical clustering. This step provides a set of results embedded in the video image plane. Section 4.4 describes how these structures can be projected into three dimensions and animate a synthetic 3D replicas of the images found on the input 2D video. We then show in section 4.5 how our method can be used to animate similar, but non-identical, complex 3D models of shrubs.

4.2 Related Works

4.2.1 Structure from Video Motion

The goal of extracting structural information from a single monocular view has been extensively studied in the Computer Vision literature. In particular many works have been done on topics such as segmentation of distinct objects in videos [Zha06] and depth estimation.

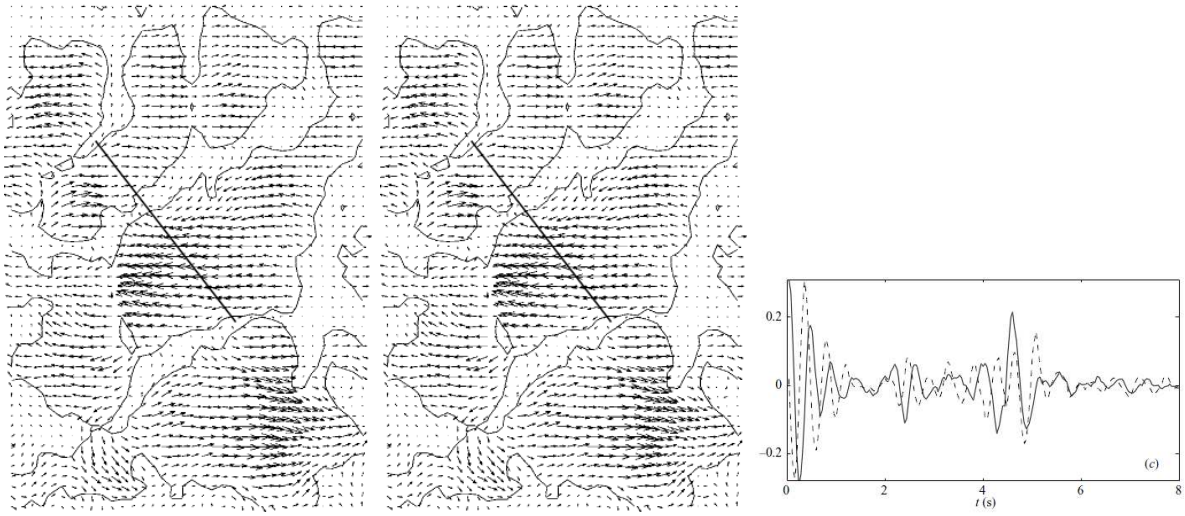
In the domain of *shape from motion*, factorization methods have proven that rank constrains allow to extract shape and motion of a collection of 3D rigid bodies [CK98] or of a 3D deformable body [TYAB01]. These approaches require to estimate in advance the rank constraint, which is related to the number of rigid groups in the case of multiple rigid bodies or the number of linear degrees of freedom in the case of deformable body. In the case of moving trees and other plants, motion of branches is not as rigid as motion of different part of an articulated body such as humans and animals. This makes these rank methods not adapted to our case. Moreover our goal here is not to recover the exact 3D shape and motion of the original plant, but rather to investigate how standard video sequences can be used as motion controller for complex 3D model of plants.

Py *et al.* [PdLMH05, PdLM06] uses bi-orthogonal decomposition (BOD) to extract from a video, waves of coherent motion called *honami* that can be seen on a field canopy. BOD is a statistical method similar to Principal component analysis (PCA) that decomposes a set of vectors evolving with time in two types of components: a set of *chronos* μ_k and a set of *topos* Ψ_k (see figure 4.1). The input is the vector field representing the estimated motion of a canopy in each pixel of each image of the sequence $v(x,y,t)$. The bi-orthogonal decomposition then compute K principal components (the sets $\{\alpha,\mu,\Psi\}_k$) such that:

$$v(x,y,t) = \sum_{k=1}^{K} \alpha_k \, \mu_k(t) \cdot \Psi_k(x,y) \tag{4.1}$$

Moreover as for PCA, BOD sorts the chronos and topos by order of influence (decreasing α_k). This method allows to detect coherent oscillatory evolution of the optical flow. In the case of

canopies animated by directional wind the honami can be found in the first components of the decomposition.



(a) (b) (c)

Figure 4.1: (a) and (b) are the two first topos and (c) their associated chronos obtained using BOD on the optical flow of a video of a canopy animated by the wind. In (a) and (b) iso-contours of the topos magnitude are drawn. (images taken from [PdLM06])

There are a few flaws to the use of BOD for the extraction of structural information on a hierarchical branches system. First for each extracted component, the topos is defined over the entire image. Most of them have a localized influence, but BOD does not guarantee a precise segmentation in space. Moreover spatial correlation of wind waves has strong impact on the decomposition. In the case of [PdLM06] this is a quality as the goal is to detect honami. In contrast the extraction of animation structures should more emphasize on the statistical independence amongst different area of the foliage. An other limitation of BOD to our problem is that the localization in the image of the topos are fixed along the video. But motion of branches (and their descending leaves) can have a bigger amplitude in the image plane than the displacement of the field crop as studied in [PdLM06].

4.2.2 Animation from Video Motion

Video-based animation has been mostly explored for character animation. Forsyth *et al.* provide a thorough study of this issue in [FAI+05]. There are many different types of approaches. However all are based on a priori model of human or animal structure. For example, Bregler *et al.* in [BM98] and [BMP04] propose a tracking model, called *twist* and *exponential maps*, to extract the pose of a kinematic chain of rigid bodies that can be used to track humans or animals. This model is obtained from the hypothesis of conservation of light intensity (as described in section 2.4.3) applied to the projection of an articulated structure onto the projective plane of the video.

The gait of animal locomotion from live video documents has also been put in focus by Favreau *et al.* [FRDC04] to provide a new animation method for virtual animal model. In this approach the cyclic nature of animal locomation is extracted from the two first components obtained from PCA applied on the video where the subject is segmented from the background. These serves then as a basis to map video motion data to the pose of the virtual model. Instead of using a priori knowledge on the specific 3D structure of the subject, this approach requires periodicity of the observed motion.

Concerning plants, Sun *et al.* propose the use of *video-input driven animation* (VIDA), a system to estimate the wind velocity from videos of animated trees or other oscillating systems [SJF03]. They invert the equations of motion of a mechanical model of an oscillating beam (that represent the trunk of a tree) to infer a representative wind field that would account for that motion. It allows the introduction of additional effects onto the original video, such as synthetic snow, leaves or dust, as well as new trees, all of which would be coherently controlled through interactions with the estimated wind field. In the method described in this chapter, a video of a plant is analyzed to directly build a model of its branches structure. The extracted model can then be used to animate a 3D synthesized plant without estimating explicitly the wind force.

4.3 Building a Hierarchy of Branches from a Single Video View

In this section, we describe how the structure and motion of a hierarchy of branches can be automatically estimated from feature positions tracked in a video sequence. Using the software and tracking methods described in the two previous chapters, a set of N features of the foliage and visible branches are tracked along the video. Let T be the number of images in the sequence, we have N discrete curves $\mathbf{p}_i$ such that:

$$\mathbf{p}_i(t) = (x_i(t), y_i(t)) \quad \textit{wheret} \in [0, T]$$

The velocity $\mathbf{v}_i(t)$ of feature i is computed in 2D image co-ordinates as the difference between the positions at two consecutive frames:

$$\begin{aligned} \mathbf{v}_i(t) &= (u_i(t), v_i(t)) \\ &= \mathbf{p}_i(t+1) - \mathbf{p}_i(t) \quad \textit{wheret} \in [0, T-1] \end{aligned}$$

The number of features N needed by our methods depends mainly of the size and number of visible branches of the filmed plants. For example, using automatic tracking the *ficus* model used for illustration along this chapter requires around 200 features, while around 1000 to 2000 are used for the trees shown in the section 4.6.

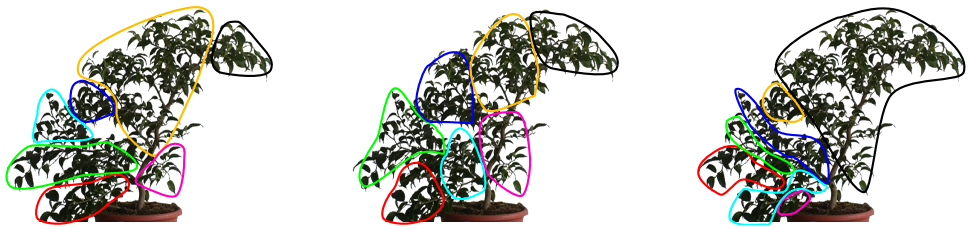
4.3.1 Clustering Metric

The first part of our method consist in grouping features that are descending of a same branch. To this end we use clustering analysis, *i.e.,* statistical techniques that produce a partition of a given set, here containing the tracked features, with respect to the relative distances between its elements.

The results obtained from clustering is thus directly related to the metric (*i.e.,* the distance function) employed. A simplistic approach would be to use euclidean distance between features position. This would obviously lead to incorrect clustering in the sense that the features from different branches can be grouped together. A logical hypothesis is that branches and leaves motion are most often similar when they are close in the hierarchical structure of the plant. We have then tried using the euclidean distance between the speeds $\mathbf{v}_i$ of the features. However better results were obtained when using a distance between the direction of these displacement vectors. Considering the visualization tool described in section 3.2, we typically want to group together features that appears with same color. This led us to use the distance of movement d_{mvt} that is defined for any speed $\mathbf{v}_1(t)$ and $\mathbf{v}_2(t)$ at any time t by:

$$d_{mvt}(\mathbf{v}_1(t), \mathbf{v}_2(t)) = \arccos\left(\frac{\mathbf{v}_1(t) \cdot \mathbf{v}_2(t)}{|\mathbf{v}_1(t)||\mathbf{v}_2(t)|}\right) \tag{4.2}$$

Figure 4.2 compares a ground truth clustering made manually and the results of an automatic clustering based on feature positions and speed direction.



(a) Ground truth obtained from manual clustering

(b) Automatic clustering based on feature positions only

(c) Automatic clustering based on speed direction

Figure 4.2: Comparison between a manual clustering that shows an expected classification of the features in seven groups, results from clustering using euclidean distance in the image and using the distance of movement.

The clustering obtained using distance of movement produce correct results concerning motion. However it is quite different from a human made classification which is based on visual perception. In figure 4.3.1 many little groups are generated on the left side of the shrub while only one on the right containing half of the visible foliage. This happens because the motion present in the original video is created by strong wind impulses coming from the left side. Thus the complexity of the plants movement is mostly localized on this side. From a perceptive point of view however, the overall motion should be decomposed in regions of similar size.

To obtain a result more satisfying perceptively, we consider a composite distance for the clustering which integrates both the position and the velocity of the features. In our method, we use the sum of the euclidean distance between positions and the angular distance between velocities where both are normalized by their respective standard deviation observed over all the features along the whole sequence. However the resulting dissimilarity function is not a correct metric from a mathematical point of view as the distance of movement does not respect identity of indiscernible ($d(x,y) = 0 \Leftrightarrow x = y$) and the triangle inequality ($d(x,y) \leq d(x,z) + d(z,y)$). The advantage is that it avoids the needs to configure the relative influence of both distances and produces good results (see section 4.5.1 and 4.6).

As one might expect, our experiments showed that a per frame approach introduces instabilities, because incoherence between class composition occurs from frame to frame. To reduce this effect, a criterion is computed for each frame to evaluate its relevance to coherent motion. In a statistical point of view, we need to reject outlier frames which reduce the quality of the clustering. Such frames occur for two reasons: one because of unstructured motion such as case (a) of Figure 4.3 due, for example, to low motion of branches that brings out the influence of tracking noise; one because of overall dominant motion, such as case (b) of Figure 4.3, which gives too much weight to the distances of positions as it is not representative of a per branch movement structure.

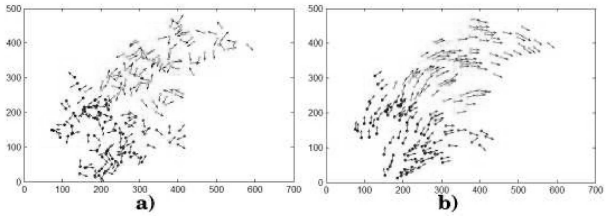
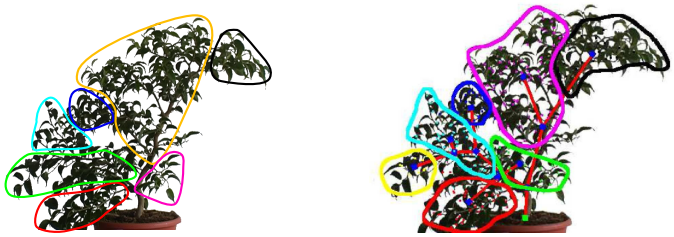


Figure 4.3: (a) Unstructured motion, (b) Strong wind inducing a single overall motion.

To characterize these two cases, we firstly compute the average of distances between all pairs of points for each frame. Secondly, we compute the mean and standard deviation of this per frame value over the whole sequence. Frames with a value above or below two times the standard deviation with respect to the mean value are considered as outliers. Finally, the distances used by the features clustering is the average of composite distances between pairs of features computed over the selected inlier frames. Figure 4.4 shows the result on the *ficus* sequence using this automatic classification method.

4.3.2 Selection of a Hierarchy of Branches using Clustering

We now discuss details of the clustering algorithm and how it automatically generates a hierarchy of branches. Most classification approaches fall into three main categories [Cor71], [Gor87], [HTF01]:



(a) Ground truth obtained from manual clustering

(b) Automatic clustering based on distance combining feature position and velocity

Figure 4.4: Comparison of the ground truth classification with the results obtained with our automatic method.

- *Partition*, where all individuals are clustered around representative data points (typically using k-means methods),
- *Division*, where an explicit ordering criterion iteratively splits individuals into hierarchical classes (leading to a hierarchical description of the data set–typically *kd*-tree methods),
- *Aggregation*, where individuals are iteratively compared and combined by closest pairs (leading also to a hierarchical representation).

Because we want to find a hierarchy, and because we have no inherent, explicit ordering criterion, we adopted the third choice with an aggregative method. Initially, this approach start with a list of all the features and compute the distances between all possible pairs. We use the dissimilarity function described in the previous section.

Choice of a clustering criterion

The aggregative method iteratively merge the closest pair. Each time two elements of the list are combined, they are removed from the list and a new cluster containing both is added. It is thus necessary to choose a suitable criterion that defined the distances between two elements that are already groups of features.

A possible criterion is to use the average distance. Given three elements x, y, z, where x and y are already merged into an aggregate group H, z is compared to the group H by evaluating a distance $d(H,z)$. Several choices are possible for this distance.

$$d(H,z) = \frac{d(x,z) + d(y,z)}{2} \tag{4.3}$$

where $d(x,z)$ is the features dissimilarity function described in the previous section.

By extension, H contains the subgroups Q_i and Q_j and not just individual elements, the equivalent distance of z to the group H is given by

$$d(H,z) = \frac{n_i d(Q_i,z) + n_j d(Q_j,z)}{n_i + n_j} \tag{4.4}$$

where n_i and n_j are the number of elements contained by the subgroups Q_i and Q_j respectively.

Finally, when all individuals are already clustered into a group, further aggregation requires to evaluate the distance between groups. The following formula is thus used:

$$d(Q_i,Q_j) = \frac{1}{n_i \cdot n_j} \sum_{x \in Q_i} \sum_{y \in Q_j} d(x,y) \tag{4.5}$$

where X and Y are groups containing respectively n_X and n_Y individual elements.

This average distance criterion provides a simple approach for aggregative clustering. However, it turned from our experiments that this approach lacked of robustness to changes of initial conditions and provided unbalanced groups. To overcome this limitation, we have used a refined approach which considers the concept of inertia, known as the Ward criterion. The inertia $I(S)$ of a set S of n_S features is:

$$I(S) = \frac{1}{n_S} \sum_{x \in S} d^2(x, g_S) \tag{4.6}$$

$$\text{with} \quad g_S = \frac{1}{n_S} \sum_{x \in S} x \text{ is the center of gravity of } S \tag{4.7}$$

If Ω is the set of all features and of size N, the equation (4.6) of a classification of Ω in k groups Q_i can be simplified to (using the theorem of Huygens):

$$I(\Omega) = I_{inter} + I_{intra} = \frac{1}{N} \left[\sum_{i=1}^{k} n_i d^2(g_i, g_\Omega) + \sum_{i=1}^{k} I(Q_i) \right] \tag{4.8}$$

The value of I, the total inertia of the system, is constant as it is independent of the clustering. At the initial stage all groups Q_i contain only one feature, and thus the *intra*-group inertia, I_{intra}, is null. Then, at each stage of the clustering process, the inertia of the system will be transferred from the *inter*-group inertia, I_{inter}, to the *intra*-group inertia by an increment Δ_I depending on which elements will be aggregated together. The principle of the Ward criterion is to aggregate, at each stage, the pair of elements which will minimize the increment Δ_I. If we develop the increase of *intra*-group inertia for a combination of two cluster Q_i and Q_j, and after some rewriting, this increment can be computed by:

$$\Delta_I(Q_i,Q_j) = \frac{n_i \cdot n_j}{n_i + n_j} d^2(g_i, g_j) \tag{4.9}$$

In our case, it is not possible to define the center of gravity of a group that would account for the features distance function described in the previous section. The solution is to use directly the increment of inertia Δ_I as the new distances and maintain a list of them for all pairs currently in the list of elements. Each time a new cluster is created, the increment of inertia that would arise by combining this new group with any other element of the list can be computed from the previously known Δ_I. Let Q_k the new cluster made by merging two elements Q_i and Q_j, then for any other element Q_m we have:

$$\Delta_I(Q_k, Q_m) = \frac{1}{n_k + n_m}\Big[(n_i + n_m)\Delta_I(Q_i, Q_m) + (n_j + n_m)\Delta_I(Q_j, Q_m) - n_m\Delta_I(Q_i, Q_j)\Big] \tag{4.10}$$

It should be noted that the equations (4.9) and (4.10) is obtained by considering an euclidean metric. We still use this criterion mostly for the improvement of the final results which basically induce questions on the meaning of inertia in our case. In particular, it can be seen by comparing equation (4.9) with equation (4.5) that the difference between the weighting parameters (depending on n_i and n_j) shows that the ward criterion tends to cluster little groups first.

Branches structure

Aggregative methods induce a hierarchical representation known as *dendrogram*. Given a required number of classes, the final clustering is obtained by adjusting a cut-off line on the dendrogram representation as illustrated in Figure 4.5.

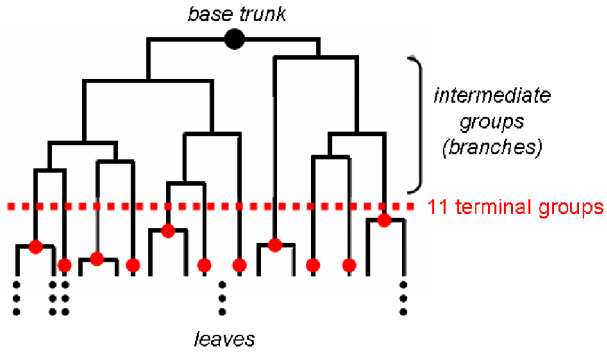


Figure 4.5: Hierarchical clustering and cut-off line for the determination of the number of classes identified as terminal groups.

From Figure 4.5, once a number of classes is set, all the features below the cut-off are grouped as *terminal groups*. Groups above the cut-off line automatically provides a hierarchical structure of the shrub with *intermediate groups*, "up" to the base trunk. A branch is identified as joining two groups. By changing the location of the cut-off line, a different choice of branches is coherently selected. This approach automatically produces a levels of detail selection mechanism based on the statistical distribution of data only.

4.4 Creating 3D Shape and Motion

The algorithm of the previous section yields a topological hierarchy of branches that is valid over the entire video sequence. This section presents a method to automatically deduce 3D shape and motion from this structure. The shape and motion of the branches (terminal and intermediate groups) are first defined in the 2D video image plane and then converted to 3D.

4.4.1 Creating Motion of the Terminal Groups

For each frame of the video sequence, we consider the distribution of the features within a terminal group to deduce the 2D parameters of an ellipse. This ellipse corresponds to the 95% isocurve of a Gaussian distribution defined through the covariance matrix of the data set. A third axis is created perpendicular to the image view plane, with a length equal to the shortest axis of the 2D ellipse. This generates 3D ellipsoids for each frame and each terminal group. The center of each ellipsoid provides the position of the group in the image plane co-ordinate system.

This approach is inspired by methods for inflating 3D shape from 2D structure, typically illustrated by the "Teddy" system [IMT99], where 2D contours drawn by hand are automatically converted into 3D volumes. In our case, we do not create complete continuous polygonal surfaces, but instead we project the features away from the image plane onto the ellipse to give a different depth to each one.

4.4.2 Propagating Motion to Intermediate Groups

The terminal groups are now geometrically positioned. For each frame, the position of an intermediate group is firstly computed as the average of its two child groups (each group has exactly two children, recalling Figure 4.5). This initialization process starts with the terminal groups as their location is already known from the previous section. Spatial locations are propagated down to the root node, providing the position of the intermediate group in the image-plane co-ordinate system. An additional node is then added at the base of the trunk for visualization only.

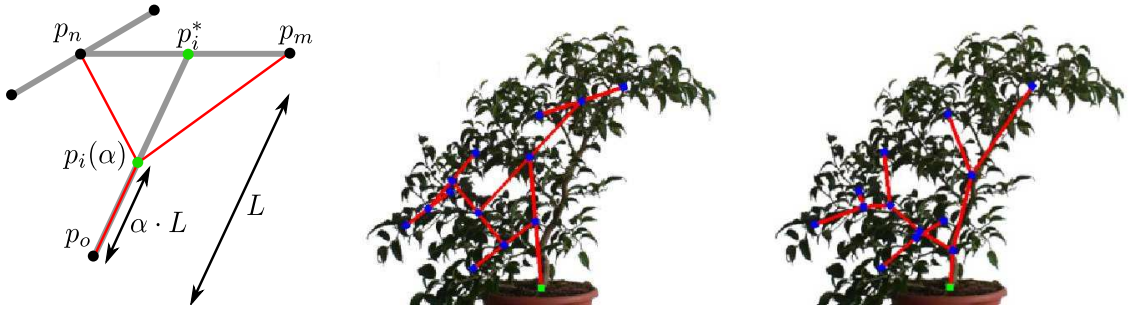
This first step results in unrealistic T-junctions for branches (see figure 4.6(b)). To create more realistic tree shapes, intermediate groups initially at T-junction are shifted toward their parent group. It is realistic to expect that branch lengths do not change over the duration of the animation. This fact provides geometric constraints that we use to improve the shape of the extracted structure.

For each intermediate node, a *shifting coefficient* α is estimated by minimizing the standard deviation of the length of the neighboring branches over the whole video sequence. Let p_i be the node to update, p_i^* its initial position between the two child nodes p_n and p_m and p_o the position of the parent node (see Figure 4.6):

$$\alpha = \arg\min\left[\sigma(\| p_i(\alpha) - p_n \|) + \sigma(\| p_i(\alpha) - p_m \|)\right] \tag{4.11}$$

$$\text{with} \quad p_i(\alpha) = p_o + \alpha(p_i^* - p_o) \tag{4.12}$$

To keep a coherent shape, this optimization process is started from the root node and propagated toward the terminal groups. To obtain best result the position of the first node of the hierarchy is manually placed to its real location. Intuitively, the algorithm propagates this manual displacement to the successive nodes using equation (4.11) and following the hierarchical order of the structure.



(a) Optimization of intermediate node locations

(b) Extracted structure before optimization

(c) Extracted structure after optimization

Figure 4.6: (a) The position of a node p_i is initiated as the average (p_i^*) of its children node p_n and p_m. Then it is optimized to minimize equation (4.11). (b) and (c) Extracted structure before and after the optimization.

The position of every group (intermediate and terminal) is then transformed from the global co-ordinate system of the image plane to a set of local co-ordinate systems along the branch using the topology of the hierarchy. Group positions are converted into local polar co-ordinates. The length of each branch is fixed at its mean to keep fixed-length branches. The motion of each branch is expressed as a rotational animation curve in the 2D image plane. The clustering and 2D geometrical hierarchy is illustrated in Figure 4.7.

4.4.3 Extending Groups to 3D Shape and Motion

So far, the shape and motion of the branch structure is embedded in the 2D image plane (only the projected features considered as leaves are shaped in 3D as described in section 4.4.1). We propose two methods to finally extend them to 3D: one method relies on user interaction, the other one is fully automatic.

The **first method** keeps the geometrical hierarchy obtained from the previous section as is and is presented to the user to manually edit the tree structure. To make this process intuitive in 3D, for each branch node, axes are oriented so that:

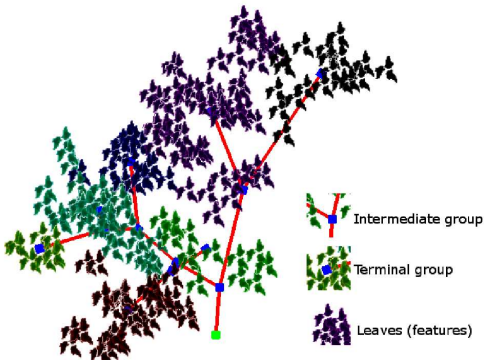


Figure 4.7: Final geometrical hierarchy of groups.

- the branch joining the parent to child group corresponds to the x-axis,
- the z-axis is perpendicular to the image plane,
- the y-axis is obtained by cross-product of the x and z axes.

Motion of the branch is assumed to be mainly contained in the image plane, which is orthogonal to the z axis, resulting in a 2D rotational motion oriented around the z axis for the branch. An animation curve as a rotation around this z axis is kept as the whole motion of the branch. The motion is thus embedded into the image plane. To create a non-planar 3D distribution of branches, while keeping the motion of the branch in a plane, it is proposed to the user to simply rotate this plane by changing the y-axis only. By changing only one degree of freedom, a wide variety of orientations can be intuitively created. This method was used to implement the rendering where each leaf is assigned a small texture (Figure 4.8). This shape editing has to be done for a single frame only, the animation staying in a predefined plane. Note that by grouping all the leaves of the same terminal group into a display list, real-time rendering is easily achieved in this case. Intermediate groups are animated as a standard hierarchy of rigid co-ordinate systems.

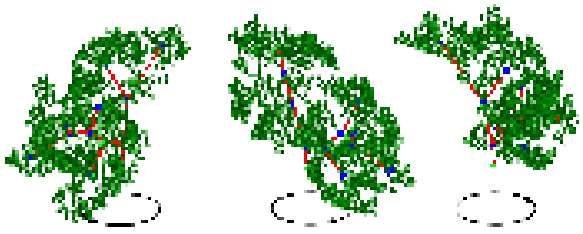


Figure 4.8: 3D geometrical structure with texture sprites for leaves.

The **second method** provides 3D location and motion of groups automatically. The algorithm in section 4.4.1 is slightly modified: for each frame, the distribution of the leaves is now estimated over the entire plant data set, and not on a per-group basis any more. This creates a global 3D ellipsoid for the plant. The location of the center of the terminal groups is therefore estimated in 3D by projecting their 2D positions perpendicularly from the image plane onto the ellipsoid.

It thus provides a depth value to each terminal group. The algorithm described in section 4.4.2 remains valid in 3D and is applied to propagate motion to the intermediate group and compute the *shifting coefficient* α. Now that the terminal groups are initially distributed in 3D, so too are the intermediate groups. 3D positions are also converted into local co-ordinates of the hierarchy. Motion is now converted into spherical co-ordinates, instead of polar co-ordinates as above. Note that depth is inflated into one direction only. To cover the other half part of 3D space, the results are simply mirrored with respect to the image plane. This method provides denser 3D space coverage of the branches and was used in the animation method described in the next section and illustrated by Figures 4.10 and 4.11.

4.5 Controlling the Animation of Complex 3D Plant Models

The previous section gave a method to extract an animated 3D hierarchical structure, and renders in real-time a 3D replica of the input video footage. In this section, we present an interesting result in which a similar but different structure of a more complex 3D plant model can be animated using the same extracted 3D tree structure.

4.5.1 3D Animation of a Plant by Skinning

The leaves attached to a terminal group only contribute to the rendering and can be removed without altering the tree structure. The key idea is to use this tree structure as a control skeleton for animating a 3D model using skinning algorithms, just as it is currently done for character animation. The complex 3D model of a shrub is bound to the control skeleton using the standard algorithm for smooth skinning implemented in the *Maya* software. As the branches of the skeleton move, so do those of the shrub (Figure 4.9).

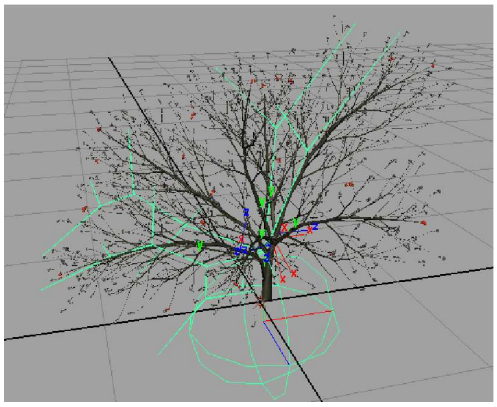


Figure 4.9: 3D geometrical tree structure used as a control skeleton for animation by skinning.

Although the geometrical structure of the control skeleton does not exactly reflect the structure of the target 3D model, the results are surprisingly convincing. We believe that such results are

achieved due to the fact that the 3D model used as a target has been generated by a computer program, and as such, it has a rather isotropic branches and leaves density.

Figure 4.10: Video footage controlling 3D animation of the *ficus* model (motion blur corresponds to a 1/50s shutter speed).

Figure 4.11: Original footage of the *xmas* model and synthesis target for animation of a complex 3D model of shrub

4.5.2 Interactive 3D Modeling

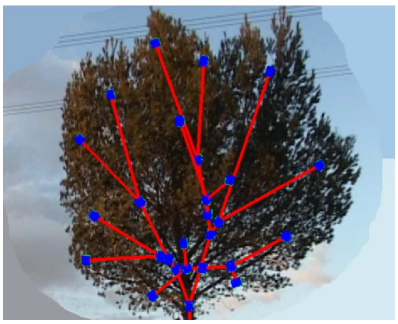
The isotropic assumption may be too strong to be respected in some cases, such as the one on the shrub displayed on Figure 4.10. Indeed, the structure of this shrub is quite unbalanced. As in the previous case, the skeleton structure is first scaled to match the target 3D model. The user then assembles 3D branches with foliage to construct a final tree, guided by the location and orientation of the control skeleton. Based on the 3D hierarchical structure provided by the video analysis, this modeling process could be automated. Figure 4.12 illustrates an example of this interactive modeling. In this case, six branches are copied from an existing 3D model and composed together in a new configuration.

Figure 4.12: Three main steps of the interactive modeling of shrubs that fit the control skeleton.

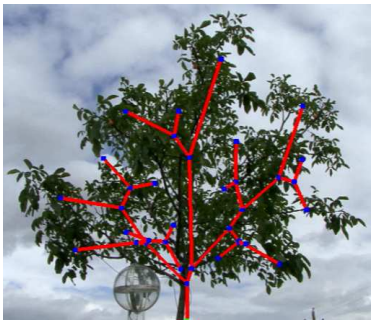
4.6 Discussions on our Approach

For the work presented in this chapter, the main goal has been to provide a new motion capture method to animate virtual plant models using video only. We have thus mostly focus on the use of automatic tracking techniques as an input process for our method to require minimal user work. However as described in the previous chapter such tracking techniques have limitations that induce quality issues of the input data. The motion of the terminal groups begin the mean of the features displacement, our method is robust to noise, but coherences in the tracking error cause some visually unrealistic movement.

Typically we obtain better results when the foliage is sparse and the leaves rigid as for the *ficus* model (see figure 4.10). For the *xmas* and *clermont* sequences (figures 4.11 and 4.13(b)), the reduction of the quality of the input motion mostly due to the leaves fluctuating deformation. Finally, occlusion of branches induced some motion irregularities in the *salon* sequence (figure 4.13(a)). However the structure extracted by our method still maintain consistency.



(a) Extracted structure for the *salon* sequence



(b) Extracted structure for the *clermont* sequence

Figure 4.13: Result obtained on video sequences of two full grown tree.

A specific aspect of our approach is that it is based on single-view video footage. Although our method generates 3D structure, working with a single view naturally implies that all the relevant motion of the plant is extracted from a single viewing plane. To take maximal advantage of this configuration, our methodology has been to film the plants so that the wind flow is parallel to the view plane. It turned out that most visually pertinent motion was observed using this approach. Using multiple cameras to effect stereoscopic reconstruction could allow to extract full 3D plant motion. However, this would raise other problems such as feature correspondence.

Furthermore, using a video sequence as the only input to create motion restricts the animation to motion observed in this sequence. However, motion synthesis methods could be adapted to our case to compose longer sequences. An interesting development of our method would be to estimate the wind (for example by adapting the method presented in [SJF03]) and use it as a control parameter for the generation of 3D animation. The resulting model would provide a method to control realistic plant motion from intuitive wind parameters.

Finally, the flickering appearance of rapid motion of leaves are natural part of a realistic aspect of a tree under the influence of wing. This point is not addressed by our method. The goal is to

provide a final result which can be easily integrated into a standard computer animation pipeline, based on an animated hierarchy of textured rigid objects. In general we have not addressed any geometrical rendering issues. This can obviously be improved. Specifically, due to the difficulty to obtain virtual plant model containing structural information (*i.e.,* which polygones are leaves and to which branches they are attached to) the skinning method has been applied not only on the branches but also on the leaves vertices, resulting in sometimes visible unnatural deformation.

4.7 Conclusion

We have demonstrated that subtle and complex plant motion can be visually modeled and mapped onto an animated 3D model with motion extracted from a single-view video. The coupling of video analysis and hierarchical clustering offers an appealing alternative to physical simulation, which is known to be difficult to control. An unusual application of skinning combined with our method indicated that a wide range of complex 3D models may also be animated.

The main contributions of the work presented in this chapter are twofold. First we show that statistical clustering using a suitable dissimilarity function provides consistent information on the structure of motion of plants extracted from a video and on the structure of the plant itself. Second we provide a first motion capture method for the animation of virtual plant (to our knowledge) which only requires a single video sequence.

Part II

Mechanical Simulation

chapter 5

Real-Time Animation of Trees using Simulation

Contents

5.1 Introduction **80**
5.1.1 General Principles 80
5.1.2 Outline of the Chapter 82
5.2 Wind Models **82**
5.2.1 Physical Simulation 83
5.2.2 Procedural and Phenomenological 84
5.2.3 Stochastic Wind 86
5.3 Modeling the Structural Elements **87**
5.3.1 Undeformable Segments with Angular Spring 87
5.3.2 Uniform Beam 88
5.3.3 Cosserat Rod 90
5.4 Structural Modeling **92**
5.4.1 Dynamics Formalism 92
5.4.2 Independent Elements 94
5.4.3 Articulated Structures 94
5.4.4 Finite Element Method 95
5.5 Time Integration **96**
5.5.1 Static Equilibrium 96
5.5.2 Explicit and Implicit Method 97
5.5.3 Spectral Method 99
5.6 Summary **101**

5.1 Introduction

In the first part of this dissertation, the acquisition and reproduction of motion from real plants has been discussed. However in computer graphics and in most mechanical study of plants dynamics, existing methods generate plants motion using computer simulation. As for any deformable structure, a large range of methods can be used for this task. These can be very different and their suitability should be evaluated with respect to the application there are dedicated to.

In this chapter we give an overview of such simulation methods. We mostly focus on existing real-time animation techniques that are specifically developed for 3D plant models. In particular, special interest is given to simulation methods that model plants response to external wind load. Different approaches can be found in other literatures that could be used to animate plants. But the specificities of real-time application containing animated trees usually restrict their usability.

Moreover for graphics purpose, the quality of a simulation is based on the perception of realism which is difficult evaluate. For example, compared to a more traditional assessment of simulation results, one can consider the relative importance of precision on spatial position and on oscillatory behaviour. The complex geometry of trees actually hides most deformation artifacts while an animation with unnatural spectral description, both in space and time, is quickly perceived as unrealistic.

5.1.1 General Principles

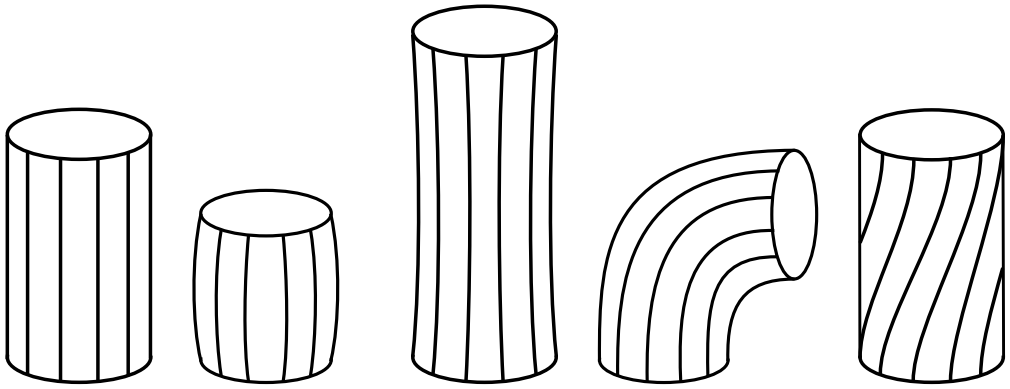
Reducing Dimensionality

In general, the complexity of an animation structure involves a tradeoff between simplicity and accuracy of the model. In the first chapter of this thesis, a short biological description of the diversity and complexity of plants has been given. In its most complex representation, each biological part of a plant can be modeled as a volume with a specific topology made of a specific material. However in the context discussed here, we want to model the dynamical deformation of this structure with respect to external forces in such a way that it can be simulated in real-time. At the physical scale we are interested in, *i.e.,* from a shrub to a whole tree, a cellular representation is not suitable. A few simplifications are thus necessary, and commonly accepted, such that the plants structures are defined to allow significant reduction of the computational simulation cost.

For branches structures, instead of the more common *volumic* model employed for many mechanical simulation of other types of objects, tree are usually model using a *lineic* representation. Lineic basically means that the structures can be expressed as connected curves. Typically a branch can be modeled by a line passing through its center and any deformation of a point of this line represents a displacement of the branch cross section linked to this point. Existing methods do not always refer to this representation explicitly. However, to our knowledge every animation structures effectively used for real-time animation of trees can be expressed this way.

Deformation and Parametrization

Even considering lineic representation, the branches deformation remains volumic. A local deformation can be categorized in a set of possible types shown in figure 5.1. Typically for trees, one expect to keep only bending and possibly torsion. In particular the length should be preserved as much as possible.



(a) At rest (b) Shrink (c) Stretch (d) Bending (e) Torsion

Figure 5.1: The typical deformations of branch segment.

An important criteria to differentiate modeling approaches is the parametrization that can be used to express the structure deformation. We can identify two classes, namely cartesian and generalized. Generalized coordinates methods directly model the degrees of freedom (d.o.f.) as the animation parameters: the bending and torsion of the branches. It has two main advantages: the mechanical modeling is easier and the length implicitly kept constant. On the other side, the use of cartesian coordinates has the advantage of being more suitable for rendering the geometry: Due to the number of elements necessary to model a realistic tree, avoiding the computation of the spatial transformation at run time represents a substantial benefit. However this approach makes it usually difficult to keep branches length unchanging. Simulation methods exist to enforce this as a mathematical constraint. But for real-time animation techniques, approaches using cartesian parametrization usually apply a simple correction process before rendering the geometry.

Discretization

To further simplify the mechanical model, a discretization of the structure is necessary. The main purpose of this step is to reduce the simulation of a tree dynamics involving a theorically infinite number of d.o.f. to a problem dependent on a small number of d.o.f.. A plant is thus represented by a finite set of segments which are attached by punctual nodes from the lineic representation. These nodes can also be called the control points or the joints of the structures. The mechanical properties of the segments depend on their shape (*e.g.,* cylinders of adequate radius). And any deformation of the structure is described by the dynamical state at the nodes. In a complex

representation, the animation structure can be made of hundreds of thousands elements to match as closely as possible the initial shape and mechanical characteristic even for a simple plants. However considering dynamics for real-time animation, such a structure is not necessary. One usually restrict the number of elements to be directly proportional to the d.o.f. expected to have an influence for the animation (see section 5.3 for the description of possible types of d.o.f.).

It should be noted that the discretization does not necessarily lead to a discontinuity of the deformation. First, approaches such as the Finite Elements Method (F.E.M), often model the smooth deformation of the structure as an interpolation between the nodes (see section 5.4.4). Second, the mechanical representation of a plant can be quite different from the geometry used for rendering. Such as described in chapter 4 a much more complex geometric model can be attached to the (discretized and lineic) mechanical model. The animation structure is then referenced as the *skeleton* and the animation of the geometric model according to the motion of the skeleton is done using a *skinning* algorithm.

5.1.2 Outline of the Chapter

Due to the still important theorical diversity of the presented approaches, a full comparative description is a quite difficult task. Moreover animating trees (possibly hundreds or more) in real-time is a chalenging task that requires simplifications. And actually, many existing methods for tree animation are not very rigorous in a mechanical point of view. Thus it would not provide much interest to provide a state of the art based on a classification of these approaches only. We have instead chosen to describe all concepts relevant to a real-time animation framework dedicated to the simulation of plant motion. Then in each section, related works are cited and compared.

Before addressing the main issues of elastic structure simulation, section 5.2 gives a quick overview of possible methods to generate wind. Then section 5.3 presents how the elementary parts obtained from the structure discretization can be modeled. Section 5.4 describes how the mechanical modeling of the whole plants can be done from these initial element representations. Possible numerical integration approaches to compute the evolution in time of the dynamics are then explained in section 5.5. Finally, to summarize the chapter, section 5.6 provides a table that classifies all the presented animation methods with respect to the discussed concepts.

5.2 Wind Models

Before addressing the possible mechanical modeling and simulation approaches in more details, we want in this section to provide an overview on the types of external forces which can be used to produce the animation. In particular we focus here on different ways to describe wind forces. However many animation methods do not depend on a specific formulation of external forces. In many cases, only the information about the wind field at any time (*i.e.,* $wind(x,y,z,t)$) is required.

Concerning more general interaction (*e.g.,* contacts), suitable mechanical simulation techniques can be found that are more appropriate than existing simulation methods developed for trees. For example, Hadap in [Had06] proposes a scheme to simulate hierarchical articulated structures and their collisions. Such approaches aim for accuracy and robustness, but are more costly and more difficult to implement. In this section, and in the chapter more generally, we focus on suitable methods for real-time animation of large vegetable scene with respect to the wind load. For this reason we do not discuss other types of interaction such as collision which are usually omitted to increase computational speed. It should be noted however that it is often possible to still simulate collisions in real-time under simplifying assumptions. Methods such as Weber [Web08] approximate the response to such interactions by carefully choosing the forces applied on the tree with respect to the detected intersections between branches and with other objects of the scene.

Concerning wind, we identify three main types of formulation: simulation of fluid dynamics, procedural methods, and stochastic wind. We now explain these and briefly discuss their advantages and limitations.

5.2.1 Physical Simulation

A direct approach to generate wind is to use fluid simulation. This topic has been widely studied by the computer graphics community during the last few years. Most works aim to provide realistic motion flow of complex phenomena. Modeling an accurate representation of the interaction of wind and trees is actually a very difficult task. But for the need of real-time animation, the computation cost of the wind generation should stay low. On the other hand, part of developed fluid simulation techniques have focussed on increasing computational speed. Moreover it is often sufficient to model a 2D fluid flow when simulating forest trees in an outdoor landscape. As a consequence this can be considered as a usable wind generation method to drive a real-time tree animation framework.

Fluid dynamics is commonly described using the Navier-Stokes equations. Most often the assumption of incompressible flow with constant density ρ is made. If $\mathbf{u}$ is the speed of the flow at any point in space, simulation methods then solves the systems of equations:

$$\frac{d\mathbf{u}}{dt} = -\mathbf{u}\cdot\nabla\mathbf{u} + \nu\nabla^2\mathbf{u} + \mathbf{f} - \frac{1}{\rho}\nabla p \tag{5.1}$$

$$\nabla\cdot\mathbf{u} = 0 \tag{5.2}$$

where ∇ is the del operator (indicating the spatial gradient), p is the pressure (in Pa), ν is the viscosity coefficient of the fluid (in $Pa\cdot s$), $\mathbf{f}$ is the external force (in $N\cdot kg^{-1}$) and ρ is the volumic masse of the fluid (in $kg\cdot m^{-3}$). Variables p, $\mathbf{f}$ and $\mathbf{u}$ are function of space and time.

Stam [Sta99] was the first to provide a technique using these equations to compute fluid flow in real-time. He made it possible by using an implicit formulation for the time integration (see section 5.5). Harris [Har04] has later developed this technique to be computed on G.P.U. In short

to update a flow field from one time step to the next, these methods proceed through a set of steps were the flow **u** is iteratively updated to include each of the three first terms of the right side of equation (5.1). Then using a suitable projection operator (defined from the obtained flow field) the state of the fluid flow at the next time step can be derived such that it respects equation (5.1) with constraint (5.2).

Akagi *et al.* [AK06] are the only ones that have used fluid simulation for real-time animation of trees. Their goal is to compute both the action of wind on the tree and reversely how the air flow changes in function of the presence of plants in the scene. The influence of trees on wind is represented by the right external forces **f** in equation (5.1) (the effect of wind on trees is discussed in the following sections).

Literature on fluid simulation provides a great deal of methods for fluid-solid interaction. However, the main difficulty for this task is to model correctly the interaction at the interface between the fluid and the rigid object. To make such methods usable one needs to increase the resolution of the grid such that the cells are thiner than the smallest parts of the trees. That would be clearly intractable, especially for thin branches and leaves. For the purpose of real-time animation, Akagi *et al.* propose to define the external forces applied on the wind to be proportional to the mass of branches and leaves present in each grid cells. This simple scheme allows then to reduce arbitrarily the resolution of the grid which can then be chosen relatively to the desired simulation speed and precision (see figure 5.2).

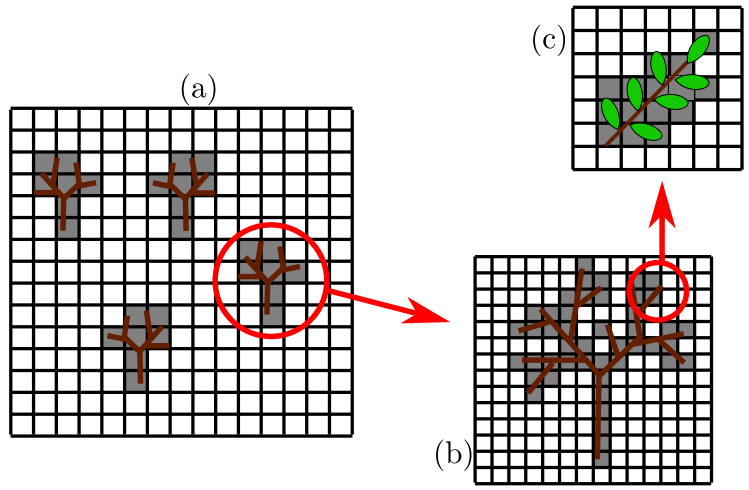


Figure 5.2: Tradeoff between precision and simulation complexity of the method of Akagi *et al.* [AK06]. The grid can be defined with any resolution (from the lower (a) to the higher (c)). Then the forces that the tree apply on the wind is proportional to wind speed multiplied by the quantity of wood and leaves in each grid cell.

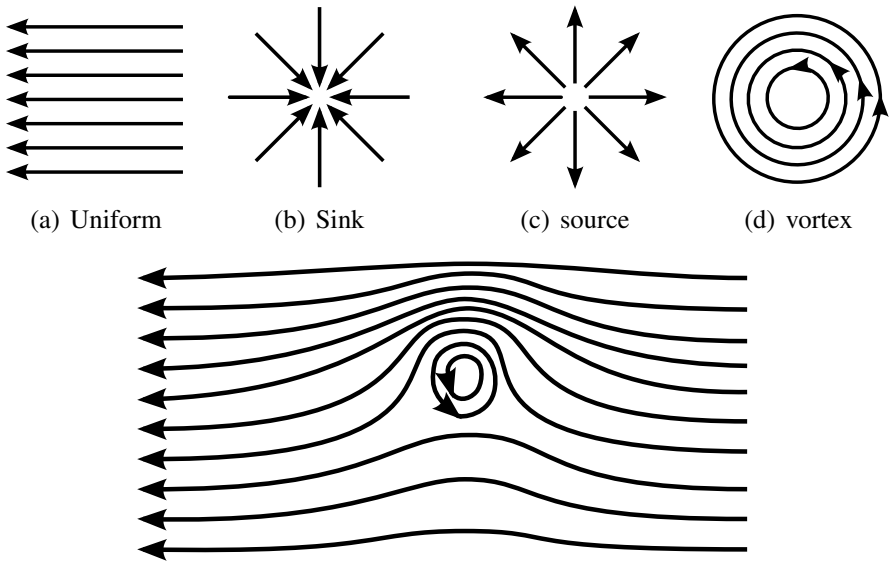
5.2.2 Procedural and Phenomenological

Fluid simulation provides realistic wind but is most often too computationally costly for real-time applications where the computational ressources allocated for the animation of vegetation

is low. The direct opposite approach to physical simulation is procedural representation of the wind flow. However simple, this is still the most commonly used approach in computer game industry [Sou07] where the only goal is usually just to add simple but not necessarily realistic motion to vegetation. Zioma [Zio07] describes such an approach. The wind is simply formulated as a sum of sinusoidal functions dependent on time. The obvious advantage is the small cost but the main limitation is that realism can only be obtained by careful and tedious manual configuration of the parameters.

An other important limitation is that it is not controlable. To allow objects in the scene to react to punctual or interactive events, phenomenoligical representation of wind flow are more appropriate. These are basically localized elements around which a procedurally formulated wind flow with limited scope is generated (see figure 5.3). It was first introduced by Wejchert and Haumann [WH91]. They define *flow primitives* that are individual solutions of the Navier-Stokes equation. A few additional constraints on the fluid flow are used to simplify the system of equations (5.1) and (5.2) to a system of linear differential equations. The sum of any wind primitives that matches the equation is thus also a valid solution. Finally the wind speed at any position in the scene is obtained by summing all speeds produced by the wind primitives.

This approach has later been used directly without the constraint to respect the Navier-Stokes equation. Perbet and Cani [PC01] defined 2D masks of wind to animate prairies. For trees and forests, Giacomo *et al.* [GCF01] use circular wind fields as sinusoidal functions of space and time, and Sousa [Sou07] does the same with omni-directional wind sources (with optional attenuation, similarly to a point light source)



(a) Uniform (b) Sink (c) source (d) vortex

(e) Sum

Figure 5.3: (a)-(d) 2D wind flow primitives. (d) Example of the sum of two primitives, a uniform and a vortex wind flow. Their sum is still solution of the simplified Navier-Stokes equation presented by Wejchert and Haumann [WH91].

5.2.3 Stochastic Wind

The third and last approach consists in on statistical description of wind flow. Its use for trees animation originates from civil engineering, applied mechanics and other related fields. The idea is to express the response of deformable objects in function of the statistical description of the fluctuation of the fluid flow (*e.g.,* wind or river). In particular it has a great importance for risk prevention as it helps to detect frequential coupling of a designed building and its environment.

For computer simulation, it has provided an interesting type of wind model based on the frequential description of natural wind coupled with a random generator from stochastic process. Many tree animation technique [SF92, Ono97, Sta97, OTF+04, ZST+06, HKW09] have used this type of wind model but with different approaches.

A wind field can be seen as a spatio-temporal vector field where the time variations are modeled as stationary stochastic processes. Usually we can extract the mean speed of the wind and focus on the modeling of the turbulence only. The *natural* caracteristics of wind is then defined by a spectral model of its fluctuations. Considering time only, one approach is the use of perlin noise as Ono [Ono97] but most of the methods discussed here [SF92, OTF+04, ZST+06, HKW09] are using models of the wind $w(f)$ which can all be synthetized in the form:

$$w(f) = \frac{a}{(1+bf)^{\beta}} \tag{5.3}$$

where f is the frequency, a and b are values possibly depending on the wind mean speed and β is a parameter (often taken as $5/3$). These models are based on specialized literature in wind engineering. Note that in the case of [OTF+04] and [HKW09] the technique called $1/f^{\beta}$ noise generator is used as described in [PS88]. There are several ways to generate one stochastic draw following a specific spectral model that we do not detail here. However, a simple approach consists in applying an inverse discrete fourier transform (with suitable normalization) on a complex signal whose norm matches the desired spectral model and a random argument (*i.e.,* the *phase* of the signal).

Ota *et al.* omit spatial constraint and compute an independent wind for each element of the tree as Habel *et al.* [HKW09] do it for the branches only. An obvious consequence is that some incoherence in the relative motion of neighboring part of the tree structure sometimes occur. Stam [Sta97] proposes to filter the spectral wind function of each tree element. Let $w_i^*(f)$ be the initial spectral representation for the ith component of a tree, then the final wind force applied on this element is computed as the weighted sum:

$$w_i(f) = \sum_{\forall j} h_{i,j}(f) \cdot w_j^*(f) \tag{5.4}$$

where $h_{i,j}(f)$ is a filter that converges to zero as the distance between element i and j increases and which also depends on the frequency f in such a way that it enforces some spectral correlation of the wind flow (typically, h is high for low frequencies).

When the wind flow is computed on a grid an other, more complex, solution has been used by Shinya and Fournier [SF92] and by Zhang *et al.* [ZST+06]. It consists in defining analytically the spectral representation of the wind field such that it respects specific spatial cross-correlation models (also called *cross power spectrum*). While equation (5.3) gives a description of a natural wind flow at one position in space, this method provides a model of a spatial (3D) spectral representation of the wind that is used to generate the stochastic wind (For example using the method described above).

A last issue to address here is the problem of the size of the data. The wind flow obtained from stochastic methods require in theory to be computed on a 4D array. For some cases, this can be reduced to 3D if the height is omitted. This can be further reduced by considering the Taylor's hypothesis of *frozen flow* as done in [SF92] and [Ono97] as well as for the leaves in [HKW09]. Let a 2D turbulent wind array $w(x,y,t^*)$ be defined for a specific time t^*. The hypothesis is that the turbulences are contant (frozen) relatively to the mean speed μ of the flow, that for simplicity we consider to be on the x-axis only. Then at any time t the wind array can be expressed as:

$$w(x,y,t) = w(x-\mu t, y, t^*) \tag{5.5}$$

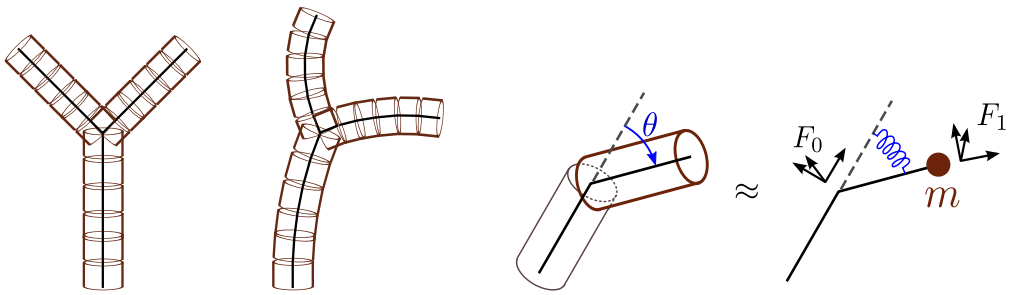
5.3 Modeling the Structural Elements

As described in the first section, a tree structure is discretized in a set of segments. However there exists several models to represent these elements which define the d.o.f. of the structure. The dynamics model of the whole structure is then constructed in function of these. Considering tree animation methods, we can distinguish three types that we describe here. For the sake of comparison we give the expression of potential and kinetic energy for each of these models in function of the d.o.f..

5.3.1 Undeformable Segments with Angular Spring

A simple approach to represent the structure elements is to use undeformable bodies which are attached with each others in a hierarchical articulated structure. For tree animation, in particular for real-time approaches, most methods [Ono97, SO99, GCF01, EMF03, BK04, OTF+04, AK06, WWG06, ZST+06, Zio07, Web08] use this representation and thus do not consider internal dynamics of branches segments. It has the advantage to simplify the dynamical model to a hierarchical articulated structure. On the other side the deformation of the whole structure is done only through rotations at the joints (see figure 5.4). It thus requires a discretization with a higher number of elements to obtain smooth enough deformations.

For each segment we consider the joint at its base (which connects the segment with its parent). Its d.o.f. are rotations usually modeled as angular springs with stiffness proportional to the square of the segment radius. The existing tree animation methods have used diverse approaches to model the dynamics. It is thus difficult to provide an global description. A mechanical correct



(a) Structure at rest

(b) Structure bended

(c) A single segment is modeled by a point mass angular springs

Figure 5.4: (a) and (b) Articulated structure made of a set of rigid segments. (c) Model of one element of the structure. F_0 and F_1 are the reference frame before and after the element respectively. F_1 is obtained after a rotation and translation of F_0.

representation would be a rigid body with constant density (as used by Featherstone to model robotic articulated structures [Fea83]). However tree animation methods usually model each couple made of a joint and the following segment as a point mass angular springs. From this representation, the potential and kinetic energy, v and τ respectively, of a set (joint,segment) can be defined. For a two dimensional element, let θ be the angle from the resting position, m its mass, L the length of the segment and k the stiffness of the spring, then:

$$v = \frac{1}{2} k \cdot \theta^2$$
$$\tau = \frac{1}{2} I \cdot \dot{\theta}^2$$
$$\text{with} \quad I = m \cdot L$$

where I is the moment of inertia and where the overdot indicates the time derivative.

5.3.2 Uniform Beam

Other approaches to represent the elementary parts of the animation structure are based on deformable elements. Each segment of the tree is then modeled such that its mechanics can be defined analytically under some deformation constraints and with respect to its boundary conditions. The whole plant deformation can then be described by attaching these elements together and applying suitable boundary conditions.

The most common segment model with smooth deformation is the elastic beam with uniform density. The lineic deformation of the beam is defined in cartesian coordinates. The volumic deformation of a beam is modeled by the continuous displacement of the cross section following the center line (see figure 5.5).

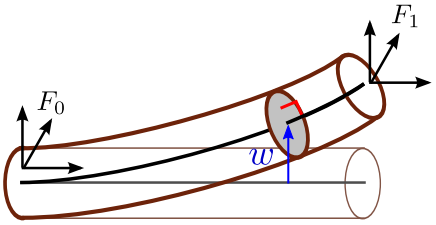


Figure 5.5: Euler-Bernoulli beam model. The bending and torsion are defined with respect to the spatial derivatives of the displacement expressed in cartesian coordinates (here w). F_0 and F_1 are the reference frames before and after the element respectively. The transformation from F_0 to F_1 is only a translation equal to the displacement of the end of the beam.

The dynamics of the beam can then be described using the potential energy v (strain) and kinetic energy τ. For example, take the two dimensional beam with its main direction along the x-axis and its width along the y-axis. The deformation of the beam is defined by the displacements u and w along the x and y-axis respectively. Using the most common Euler-Bernoulli beam model, *i.e.,* where the cross section stays perpendicular to the center line, and considering only lineic displacement along the y-axis, then the strain energy is given by [GR94, HBW99]:

$$v = \frac{1}{2} \int_{A(x)} \int_{0}^{L} Ey^2 \left(\frac{\delta^2 w}{\delta x^2} \right)^2 dx dA \tag{5.6}$$

$$= \frac{1}{2} \int_{0}^{L} EI \left(\frac{\delta^2 w}{\delta x^2} \right)^2 dx \tag{5.7}$$

using the *geometric moment of inertia* I defined by:

$$I = \int_{A(x)} y^2 dA \tag{5.8}$$

where L is the length and $A(x)$ the cross section of the beam at x, E is the Young modulus. The geometric moment of inertia I represents the influence of the shape of the cross section (usually considered as a circle). Following the same model, the kinetic energy is:

$$\tau = \frac{1}{2} \int_{0}^{L} \int_{A(x)} \rho \dot{w}(x)^2 dA dx \tag{5.9}$$

$$= \frac{1}{2} \int_{0}^{L} m(x) \ \dot{w}(x)^2 dx \tag{5.10}$$

where we define the *lineic mass m* such that:

$$m(x) = \int_{A(x)} \rho dA \tag{5.11}$$

This deformable beam model has been widely studied by the physician community and is often employed in mechanics engineering. Developments have been proposed such as the Rayleigh beam model that also considers the kinetic energy of the cross section $\dot{u}$ (see figure 5.5) or the Timoshenko model that allows the cross section not to stay perpendicular to the center line (see [HBW99] for a complete comparison).

However, regarding plant animation, only the Euler-Bernoulli model has been used. Shinya and Fournier [SF92] were first to represent independent branches with this model. As for Habel *et al.* [HKW09] the dynamics is described through an analytical formulation of the whole beam deformation with respect to external load. In the case of Shinya and Fournier, the modal decomposition in vibration modes (*i.e.,* modal analysis, see next chapter) is used as a basis to solve the branches dynamics. And in the case of Habel a quadratic function approximates the *static* response of the beam to a uniform wind load. The Euler-Bernoulli beam model is also employed by Stam [Sta97] with the F.E.M to represent the whole tree structure. More details on modal decomposition, F.E.M and static response are given in section 5.4, section 5.5.1 and chapter 6.

The main benefit of using deformable beams as opposed to articulated bodies is that this way complex dynamics can be represented using less elements. Typically to allow real-time animation to take advantage of the high parallelism of Graphics Processing Unit (G.P.U) it is necessary that the position of the control points can be computed independently from each other at runtime. Because of the hierarchical topology of trees the updates of these positions are usually dependent of the lower elements of the structure. Habel *et al.* and Zioma [Zio07] by-pass this limitation by simply computing for each node the displacement of all the parent branches.

5.3.3 Cosserat Rod

Deformable beam has the advantage to be easy to use. However its cartesian parametrization comes with a few limitations. Some types of large deformations are difficult to model and the constancy of segments length is difficult to enforce. A generalized parametrization can then be preferred. Such a model has been provided by the Cosserat brothers [CC09]. For comparison with other models we provide here a brief introduction. For a clear yet thorough description of this model and its application in computer graphics, we recommend the Ph.D dissertation of Bertails [Ber06].

The shape of a Cosserat rod is given by two elements: a parametrized curve $\mathbf{r}(s), s \in [0, L]$ which corresponds to the center line of the rod, and an orthonormal basis $(\mathbf{u}_1(s), \mathbf{u}_2(s), \mathbf{u}_3(s))$ attached to this curve (see figure 5.6). The first two vector of the basis $\mathbf{u}_1$ and $\mathbf{u}_2$ define the plane of the cross section of the curve at s. For our cases, *i.e.,* mechanical representation of trees, the *Kirchhoff hypothesis* is followed: the rod is inextensible and without shear. Thus as for the Euler-Bernoulli

beam, the cross section is always perpendicular to the center line, so the third component of the basis $\mathbf{u}_3$ is tangent to $\mathbf{r}(s)$.

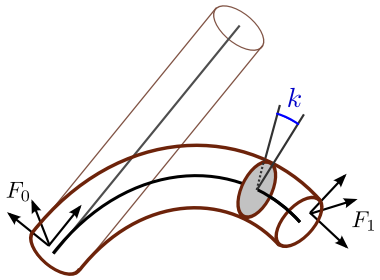


Figure 5.6: Cosserat rod model. The lineic displacement is defined with respect to the deformation of the rod, *i.e.,* bending and torsion (only bending k is represented here). F_0 and F_1 are the reference frame before and after the element respectively. The transformation from F_0 to F_1 is the integral of the deformation along the rod.

The deformation of such a rod is defined using the *Darboux vector* $\mathbf{\Omega}(s) = (k_1(s) \cdot \mathbf{u}_1(s), k_2(s) \cdot \mathbf{u}_2(s), k_3(s) \cdot \mathbf{u}_3(s))$. The first two components are the bending of the rod and the third its torsion. Finally we can express the shape of the rod at any position s of the control curve by:

$$\frac{\partial \mathbf{u}_i}{\partial s}(s) = \mathbf{\Omega}(s) \times \mathbf{u}_i(s) \tag{5.12}$$

For a two dimensional Cosserat rod, *i.e.,* the deformation is defined by only one bending $k(s)$, the potential and kinetic energy (under the Kirchhoff hypothesis) is then given by:

$$v = \frac{1}{2} \int_0^L EI\, k^2 ds$$

$$\tau = \frac{1}{2} \int_0^L m\, \dot{\mathbf{r}}(s)^2 ds$$

Overall we can see that the potential energy v has a simpler formulation than for the beam model as it is directly function of the d.o.f.. However the kinetic energy τ can be quite difficult to compute in the most general cases. This model was first used by Wu *et al.* [WCYF99] to model branches. They directly followed the results of Raboud *et al.* [RFL96] on the static simulation of a Cosserat rod with uniform curvature and torsion (at any time step, $\Omega(s)$ is constant over s). Bertails *et al.* developed a more general model first for hair animation [BAC+06] and later for hierachical structure such as trees [Ber09]. In their approach they use the *Super-Helices* model: F.E.M based on Cosserat rods with uniform bendings and torsion (which in fact are helices). The two approaches are still very different in how the whole structure is modeled and animated. These issues are discussed in section 5.4 and 5.5.

5.4 Structural Modeling

Using the elementary parts of the structure, the dynamics of the whole tree can then be modeled. The goal is to design an algorithm that simulate this dynamics. To this end, one first need to mathematically model the mechanical dependencies between all the elements.

5.4.1 Dynamics Formalism

Before describing the three main approaches used by tree animation methods for mechanical modeling, we first introduce the possible formalism that can be used for the task of modeling the equations of motion that drive a mechanical system.

Newtonian

The most well known and most commonly used is the Newtonian formalism. For each element of the structure the dynamics is modeled by the Newton's second law of motion:

$$\mathbf{f} = m\,\ddot{x} \tag{5.13}$$

where $\mathbf{f}$ is the sum of all forces applied on the element. These are the external forces (wind) and the internal forces, *i.e.,* the forces acting between the neighboring elements of the tree. The set of these differential equations defined for all elements of the structure, can be viewed as a system of equations that has to be solved at each time step of the simulation.

The Newtonian formalisme is intuitive, and seems easy to use. In many situations however, it leads to errors in the modeling of the dynamical equations. More precisely, it initially requires the expression of Newton's second law in cartesian space. It has often been used to model the mechanics of trees where the deformations are represented by rotations and torsions. But these modeling approaches are usually failing to model important parts of the dynamics.

Lagrangian

As an alternative the Lagrangian formalism can be used. One of its main advantages is that it is based only on the expression of the mechanical energy of the system with respect to any types of d.o.f. (so-called *generalized coordinates*).

For a system with only conservatives forces, the principle of stationary action (Hamilton's principle) states that the action $\mathcal{S}$ (defined by equation 5.14) of a system is extremum. If the generalized coordinates are represented by the vector $\mathbf{q}$, then we have:

$$\mathcal{S} = \int \mathcal{L}(\mathbf{q}, \dot{\mathbf{q}}, t)\, dt \tag{5.14}$$
$$\mathcal{L} = \mathcal{T} - \mathcal{U}$$

where $\mathcal{L}$ is a functional called the Lagrangian of the system and where $\mathcal{T}$ and $\mathcal{U}$ are respectively the kinetic and potential energies of this system. Hamilton's principle leads to a set of differential equations (one for each d.o.f.), called Euler-Lagrange equations, of the form:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = 0 \tag{5.15}$$

that needs to be solved during simulation (note that the signe ∂ indicates the derivative of a functional). In the case of non conservatives forces, equation (5.15) becomes:

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = f_i \tag{5.16}$$

where f_i are the *generalised forces* (named after the generalised coordinates) which define how the external forces $\mathbf{f}_{ext}$ influence each d.o.f.. Let Ω be the tree structure, $\mathbf{r}(p)$ the cartesian position of a point $p \in \Omega$, then:

$$f_i = \int_{\Omega} \mathbf{f}_{ext} \cdot \frac{\partial \mathbf{r}}{\partial q_i} d\Omega \tag{5.17}$$

This concept of generalized forces is very important when the d.o.f. are not only the cartesian position of the nodes of the trees. The generalized forces are basically the sum of the external forces projected on the displacement along the structure induced by each d.o.f..

Finally, as the potential energy is independent of the speed of the system $\dot{\mathbf{q}}$, the simulation of a tree under wind load (and other non-conservatives external forces) is expressed with respect to $\mathcal{U}$ and $\mathcal{T}$ by:

$$\frac{d}{dt}\frac{\partial \mathcal{T}}{\partial \dot{q}_i} - \frac{\partial \mathcal{T}}{\partial q_i} + \frac{\partial \mathcal{U}}{\partial q_i} = \int_{\Omega} \mathbf{f}_{ext} \cdot \frac{\partial \mathbf{r}}{\partial q_i} d\Omega \tag{5.18}$$

Example for a Single 2D Point Mass Angular Spring

Newton's second law for an angular oscillator of stiffness k is given by:

$$\|\mathbf{f}_{ext} \times \mathbf{r}\| - k \cdot \theta = I \cdot \ddot{\theta} \tag{5.19}$$

where $\mathbf{r}$ is the axe of the branch segment (of length L), and $\mathbf{f}_{ext}$ is the external force applied on the segment. In this 2D case, the rotation is done around a single axis. The torque, *i.e.,* first term

of left hand side of equation 5.19, can then be directly expressed as a scalar. The second term represents the restoring force of the oscillator.

Using the lagrangian formalism, we obtain the same equations based on the derivatives of the potential energy v and kinetic energy τ (see section 5.3.1). Equation (5.18) becomes:

$$\frac{d}{dt}\frac{\partial \tau}{\partial \dot{\theta}} - \frac{\partial \tau}{\partial \theta} - \frac{\partial v}{\partial \theta} = \mathbf{f}_{ext} \cdot \frac{\partial \mathbf{r}}{\partial \theta}$$

$$I \cdot \ddot{\theta} - 0 + k \cdot \theta = \|\mathbf{f}_{ext} \times \mathbf{r}\|$$

5.4.2 Independent Elements

The goal of section 5.4 is to give an overview of the methods to model the equation of the dynamics of the whole tree structure with respect to the d.o.f.. However most existing methods of tree animation [Ono97, WCYF99, OTF+04, BK04, AK06, ZST+06, Zio07, HKW09] actually do not consider global dynamics. Instead the d.o.f. of each element are computed independently. The advantages are reduced computation time and more accessible implementation. On the other hand it usually implies a tradeoff between realism and configuration complexity to choose suitable mechanical parameters.

For all these methods, the deformation of the tree follows its hierarchical structure: when one element is displaced all the succeding branches in the hierarchy moves accordingly. Regardless of the parametrization (*e.g.,* cartesian or rotational) the consequence of omitting this in the dynamics model may have critical consequences. In particular, the kinetic energy related to each d.o.f. (thus the inertia of the system which depends on it) suffers most of such a simplification. An other important loss is the physical influence of the external forces as it can be seen from the definition of the generalized forces described above.

To reduce the effect on the inertia, manual setting of the elements mass can be used. Instead Ono [Ono97] models each element as point mass angular spring using the approximation he calls *effective mass*, *i.e.,* the sum of the masses of all the succeding branches. Habel *et al.* [HKW09] also uses such a summed mass but only to compute the spectral response of a branch to a stochastic wind (see section on spectral method 5.5.3). In a similar way Ono [Ono97], as well as Akagi and Kitajima [AK06] consider the external forces applied on each element as being the sum of the external forces on all the child branches. This can be viewed as an approximation of the generalized forces (defined above) where the projection on the displacements induced by the d.o.f. is omitted.

5.4.3 Articulated Structures

As described in section 5.3.1, trees can be represented as a set of rigid (undeformable) segments attached by joint acting as angular springs. To simulate articulated bodies, general methods that are not specific to elastic structure were developed. Such techniques as the Featerstone's

algorithm [Fea83] or the Lagrange multiplier approach [Bar96] could be adapted to articulated structure with angular spring joints.

However concerning tree animation, existing methods that use articulated structures propose their own specific models. Di Giacomo *et al.* [GCF01] and Weber [Web08] follow the newtonian formalism to model the equation of the dynamics for each segment with respect to the relative forces exchanged between neighboring elements (internal forces). The obtained set of equations can then be represented by one matrix equation. In the case of Weber [Web08], it is solved at each time step. However such an approach requires suitable numerical solver of large systems differential equation. To reduce computational cost, at each time step, Di Giacomo *et al.* update each equation individually with respect to the dynamical state of the adjacent segments at the previous time step. The limitation of such an approach can be seen in the number of steps necessary for an external impulse applied on one segment to propagate through the structure.

To obtain a linear complexity in term of the number of segment, Sakagushi and Ohya [SO99], Beaudoin and Keyser [BK04] as well as Wu *et al.* [WWG06] propose simulation methods based only on a top-down propagation of the forces but process it entirely at each time step. The idea is that most of the tree-wind interaction happens at the leaves. Then the algorithm updates the dynamical state of the tree parts for the branches tips to the tree root, and transfer some of the forces from the child to the parent segments. The advantage is that it is simple to implement and can provide animation of good quality. However all these methods lack in a mechanical justification of their simplified models.

5.4.4 Finite Element Method

Articulated models restrict the whole deformation of the tree to a set of rotations around a finite number of points. To represent continuous deformation of the structure, Finite Elements Method (F.E.M) should be used. The principle is to reduce the possible deformations to a finite number of d.o.f. while keeping smooth deformation (*i.e.,* of C^1 continuity or more). There are many different types of representation. Regarding lineic model and more specifically those used for tree animation, we can identify two types: one based on Euler-Bernoulli beams and one on Cosserat rods. These have been used by Stam [Sta97] and by Bertails [Ber09] respectively.

In the first case, the lineic structure is discretized as a set of beams separated by nodes. The d.o.f. of the system are the cartesian coordinates of these nodes and the possible deformations are the continuous interpolation between them. The interpolation is defined using the *shape function*, *i.e.,* an interpolation matrix. The potential energy and kinetic energy of each beam can then be modeled as a function of the displacement of the their neighboring nodes using this shape function. A thorough description of this mechanical modeling approach is given in the next chapter.

For Cosserat rods, the F.E.M employed in [Ber09] was first proposed by Bertails *et al.* in [BAC+06] and called *Super-Helices* to simulate hair strands. It has then be updated initially to reduce computational complexity and incidentally became suitable to model elastic hierarchical bodies such as trees. The segments of the structures are modeled by Cosserat rods (under Kirchhoff hypothesis), each with constant bending and torsion. The d.o.f. are then the set of

these parameters for all the elements of the structure. The bending and torsion are defined as derivatives with respect to the *material frame* of the rod, *i.e.,* an orthogonal basis attached to the center line such that two axis are in the plane of the cross section. By enforcing continuity of this material frame between neighboring rods, the whole tree deformation has C^1 continuity.

For both F.E.M approaches the total potential energy and kinetic energy and their derivatives with respect of the d.o.f., can be defined analytically. And the system of equations that models the dynamics of the structure follows directly from the Lagrangian formalism.

5.5 Time Integration

In the previous section we have given an overview of the possible approaches to model the equations that represent the dynamics of the tree structure. These are differential equations that describe the evolution of the d.o.f. with respect to time and external forces. In the present section we identify three general approaches to handle the temporal integration of tree simulation.

5.5.1 Static Equilibrium

Many times along this chapter simplification methods have been described. They are typically used to cope with plants complexity for the purpose of real-time animation. These were done for mechanical modeling issues. Similarly several methods [WCYF99, OTF+04, AK06, ZST+06, HKW09] reduce the dynamics equation to the case of the static equilibrium.

By extension, we define here the *static state* (or *equilibrium state*) of the mechanical structure as the rest state of both the structure and the external forces. In the context of wind interaction, it represents the constant deformation of the structure, expressed through its d.o.f., with respect to a constant wind. Mathematically, for an elastic body interacting with fluids of constant speed, the equation of equilibrium can be modeled by omitting the kinetic energy of the system (*i.e.,* its inertia). One can consider the equilibrium as the deformation to which a tree converges if a constant wind is applied for a long enough time. The speed of the tree is then null and so is its kinetic energy. The whole dynamics can then be expressed in the form of the following matrix equation:

$$K \cdot \mathbf{q} = \mathbf{f} \tag{5.20}$$

where $\mathbf{q}$ is the vector containing all the structure d.o.f., K is a matrix representing the stiffness of the system and $\mathbf{f}$ is a vector containing the generalized forces (with respect to the d.o.f.).

It is used as a simplification by considering the system as always being in the equilibrium state even when submitted to a variable wind. It leads to significant reduction of both the dynamics and the algorithmic complexity. Regarding the latter, we can see how simple this approach is by

comparison with more realistic methods described in the rest of this section. Basically the derivatives with respect to time are not taken into account and the dynamical state can be computed from the external forces independently at any time.

However concerning dynamics approximation, it may be difficult to perceive the exact meaning and consequences of such an approach. In practical terms, for elements that converge very quickly to their equilibrium (with respect to the time between two simulation steps) this simplification is acceptable as it does not induce visual artifacts. Leaves and tiny branches can be considered as such. However for bigger branches (more specifically for d.o.f. affecting a high amount of mass) it then induces lack of realism. The wind also has to be considered. If its fluctuations does not contain strong high frequencies, *i.e.,* if its variations are sufficiently smooth, this simplification is less perceptible.

5.5.2 Explicit and Implicit Method

In the most general case, the mechanical modeling discussed previously in this chapter leads to a system of equations that can be formulated by:

$$M\ddot{\mathbf{q}}(t) + K\mathbf{q}(t) = \mathbf{f}(t) \tag{5.21}$$

where $\mathbf{q}$ and $\ddot{\mathbf{q}}$ are the vector of the d.o.f. and its second derivative with respect to time, M and K are matrices that represent the inertia and stiffness of the system, and $\mathbf{f}$ contains the generalized forces. The matrices M and K are function of the dynamical state ($\mathbf{q}$ and its time derivatives) and thus depend on time. For simplicity, we do not take this into account for the rest of the section. It should however be understood that these matrices should be reevaluated at each time step. Note also that we do not model damping in equation (5.21) in order to clarify the rest of this section. It is can be included using an additional term depending on the first time derivative of $\mathbf{q}$ (*i.e.,* $D\dot{\mathbf{q}}(t)$). However the following discussions could be developed to cope with the inclusion of this damping terms.

Its has been described how the spatial representation of the tree is discretized in order to model the equation (5.21). Similarly to solve this equation which is continuous in time, temporal discretization has to be done. For this issue however the problem is to estimate the state of the system at some time knowing only the dynamical state at a previous time step.

Solutions to this problem, called *numerical integration*, can be categorized in two main approaches, namely explicit and implicit methods. We give in this section a quick overview of these and illustrate them by the *forward* and *backward* Euler methods respectively. First the principle is exposed for the example of an ordinary differential equation of the first order and a generalisation to our case (second order) is given later. Suppose we have an equation of the form:

$$\dot{\mathbf{q}}(t) = F(t, \mathbf{q}(t)) \tag{5.22}$$

where the function F is defined analytically. Using Taylor expension of $\mathbf{q}(t)$ at the first order we obtain the following linear approximation:

$$\dot{\mathbf{q}}(t) \approx \frac{\mathbf{q}(t+\Delta t) - \mathbf{q}(t)}{\Delta t} \tag{5.23}$$

where Δt is typically the length of one time step. For discrete time, the forward and backward methods are then based on the respective formulas:

$$\text{Forward (explicit):} \quad \dot{\mathbf{q}}_t = \frac{\mathbf{q}_{t+\Delta t} - \mathbf{q}_t}{\Delta t} \tag{5.24}$$

$$\text{Backward (implicit):} \quad \dot{\mathbf{q}}_{t+\Delta t} = \frac{\mathbf{q}_{t+\Delta t} - \mathbf{q}_t}{\Delta t} \tag{5.25}$$

where time dependency is now written with subscript to indicate discretization. We want to find $\mathbf{q}_{t+\Delta t}$, or equivalently $\Delta\mathbf{q} = \mathbf{q}_{t+\Delta t} - \mathbf{q}_t$. From equation (5.22), equation 5.24 and 5.25 then become:

$$\text{Forward:} \quad \Delta\mathbf{q} = \Delta t \cdot \dot{\mathbf{q}}_t = \Delta t \cdot F(t, \mathbf{q}_t) \tag{5.26}$$

$$\text{Backward:} \quad \Delta\mathbf{q} = \Delta t \cdot \dot{\mathbf{q}}_{t+\Delta t} = \Delta t \cdot F(t+\Delta t, \mathbf{q}_{t+\Delta t}) \tag{5.27}$$

This two equations are very similar. However in equation (5.27) we can see that to compute the evolution of the system d.o.f. ($\Delta\mathbf{q}$) it is necessary to evaluate the function F at time $t+\Delta t$, which is unknown at time t. This is the main difficulty of implicit methods which comes as a tradeoff for increase precision and stability.

Concerning the equation of motion (5.21), it is necessary to cope with the second order derivative of $\mathbf{q}$. To this end, the variable $\mathbf{v}$ is defined such that:

$$\mathbf{v}_t = \dot{\mathbf{q}}_t \tag{5.28}$$

Numerical integration computes $\Delta\mathbf{v}$ which is then used to update $\mathbf{v}_{t+\Delta t}$ and $\Delta\mathbf{q}$ using:

$$\mathbf{v}_{t+\Delta t} = \mathbf{v}_t + \Delta\mathbf{v} \tag{5.29}$$

$$\Delta\mathbf{q} = \Delta t \cdot \mathbf{v}_{t+\Delta t} \tag{5.30}$$

The forward method is then fairly simple:

$$\Delta\mathbf{v} = \Delta t F(\mathbf{q}_t, \mathbf{v}_t) \tag{5.31}$$

So for equation (5.21), it is enough to compute:

$$\Delta\mathbf{v} = \Delta t M^{-1}(\mathbf{f}_t - K\mathbf{q}_t) \tag{5.32}$$

Existing tree animation methods have often used this approach [SF92, Ono97, SO99, GCF01, BK04, WWG06] as it is usually sufficient. However, depending on the mechanical modeling

employed, instabilities in the numerical integration may occur. In such a case the dynamics equation is said to be *stiff* and requires to reduce the length of the simulation time steps. On the other hand the use of implicit methods can considered. Applying equation (5.26) to the case of equation of motion (5.21), the following needs to be solved:

$$\Delta\mathbf{v} = \Delta t\ F(t+\Delta t, \mathbf{q}_{t+\Delta t}, \mathbf{v}_{t+\Delta t}) \tag{5.33}$$
$$= \Delta t\ M^{-1}(\mathbf{f}_{t+\Delta t} - K\mathbf{q}_{t+\Delta t}) \tag{5.34}$$

In equation (5.34), we can see many parameters depending on the next time steps. Each of them can be seen as an unkown variable that needs to be solved. In particular the external forces $\mathbf{f}_{t+\Delta t}$ might be very difficult to take into account as it can depend on many other parameters. Moreover as stated above, the mass and stiffness matrix (M and K) may also depend on time. In function of the specific (analytic) equation, this can be solved. However most often one resort to *semi-implicit* methods (such as [Ber09]) that basically still keep the evaluation of some of these parameters at time t.

In [BW97] Baraff proposes to approximate the function F by its first order Taylor series expansion and rewrite equation (5.33) by:

$$\begin{aligned}\Delta\mathbf{v} &= \Delta t\ F(t+\Delta t, \mathbf{q}_{t+\Delta t}, \mathbf{v}_{t+\Delta t}) \\ &= \Delta t\ F(t+\Delta t, \mathbf{q}_t + \Delta\mathbf{q}, \mathbf{v}_t + \Delta\mathbf{v}) \\ &\approx \Delta t\left(F_0 + \frac{\partial F_0}{\partial t}\Delta t\right) + \frac{\partial F_0}{\partial \mathbf{q}}\Delta\mathbf{q} + \frac{\partial F_0}{\partial \mathbf{v}}\Delta\mathbf{v} \\ \text{where}\quad F_0 &= F(t, \mathbf{q}_t, \mathbf{v}_t)\end{aligned}$$

Weber in [Web08] uses this approach but omit the time derivative of F and add instead a external parameter specific for collision.

5.5.3 Spectral Method

Concerning computer graphics, spectral approaches can be used to solve the dynamics of tree under influence of wind. They provide an interesting type of temporal integration methods whose use can be very suitable for tree animation but rather specific to wind interaction. In particular the main advantage of spectral methods comes from its use with stochastic methods for wind generation (described in section 5.2.3).

For the general case of equation (5.21), the spectral method propose to solve this differential equation in the spectral domain. Let $\mathcal{F}$ denote the Fourier transform, we define:

$$\widetilde{\mathbf{f}}(\xi) = \mathcal{F}(\mathbf{f}(t)) \tag{5.35}$$
$$\widetilde{\mathbf{q}}(\xi) = \mathcal{F}(\mathbf{q}(t)) \tag{5.36}$$

From the definition of Fourier transform, we have:

$$\begin{aligned}\mathcal{F}(\ddot{\mathbf{q}}(t)) &= (2\pi i\xi)^2\mathcal{F}(\mathbf{q}(t)) \\ &= -(2\pi\xi)^2\,\widetilde{\mathbf{q}}(\xi)\end{aligned} \tag{5.37}$$

In the spectral domain, equation (5.21) then becomes:

$$-(2\pi\xi)^2 M\widetilde{\mathbf{q}}(\xi) + K\widetilde{\mathbf{q}}(\xi) = \widetilde{\mathbf{f}}(\xi) \tag{5.38}$$

We can then define the *spectral response* of the structure as the function $G(\xi)$ and obtain the solution for $\widetilde{\mathbf{q}}$:

$$G(\xi) = \left(K - (2\pi\xi)^2 M\right)^{-1} \tag{5.39}$$
$$\widetilde{\mathbf{q}}(\xi) = G(\xi)\cdot\widetilde{\mathbf{f}}(\xi) \tag{5.40}$$

This method takes full advantage when the wind forces are initially defined in the spectral domain. For the general case, it reduces the dynamics problem to an equation of the form $A \cdot x = B$ (to be solved for vector x) for each frequency ξ. Then as for spectral wind an inverse Fourier transform is applied, only here it is directly on the d.o.f..

A further simplification is possible. As in [Sta97], this approach can be combined with modal analysis. It is a typical method in mechanical vibration analysis to compute the *modes of vibrations* of the structures. The set of these reduced d.o.f. represents an alternative basis of the possible deformations of the structure. The advantages of this method is that all the modes are mechanically independant, *i.e.,* the mass and stiffness matrix become diagonal in this basis. Thus the dynamics of each d.o.f. can be seen as an independent one-dimensional oscillator (see next chapter for more details on modal analysis). For each d.o.f. q_i (*i.e.,* any element of $\mathbf{q}$) we can define the natural frequency ξ_i of the modes such that $k_i = (2\pi)^2 m_i \xi_i^2$ where m_i and k_i are the i-th diagonal elements of matrices M and K respectively. Equation 5.39 and 5.40 then simplify to:

$$\begin{aligned}g_i(\xi) &= \left(m_i(2\pi)^2\cdot(\xi^2 - \xi_i^2)\right)^{-1} \\ \widetilde{q_i}(\xi) &= g_i(\xi)\cdot\widetilde{f_i}(\xi)\end{aligned}$$

Habel *et al.* [HKW09] also use spectral methods. As stated above, they consider each branch as mechanically independent and simplify the equation of dynamics by modeling their deformation as always being in the equilibrium state even for dynamical winds. On the other hand in

their method, the equilibrium equation is rather considered as a generalised d.o.f. that relates the wind force with the amplitude of deformation. Then each of these is modeled as an oscillator, with a natural frequency obtained from empirical studies, that can be animated using the spectral method. Moreover Habel *et al.* also propose the use of *2d motion texture* that allows to produce infinite sequences of animation. Basically it consists in pre-computing a 2D texture of noise that has a suitable spectrum (*i.e.,* $g_i(\xi) \cdot \widetilde{f_i}(\xi)$) for both dimensions. Then for any direction a line in this texture can be seen as a stochastic draw usable for any d.o.f..

Finally, the spectral animation approach has the advantage to simplify the computation of the dynamics. In particular it is well suited to independent d.o.f. as the final cost is reduced to computation of of a suitable noise function. The main limitation is that the spectrum representation must be fixed. Changing the type of fluctuation and, further more, the mean speed of the wind is not mechanically correct.

5.6 Summary

In this chapter, all the existing approaches (to our knowledge) proposed for tree animation are compared. Table 5.6 summarizes the classification of the paper following the issues addressed by each section of this chapter.

Authors	Wind	branch representation	structural modeling	time integration
Shinya and Fournier [SF92]	stochastic	beam (modal analysis)	none	explicit (*)
Ono [Ono97]	stochastic	articulated	independent (effective mass)	explicit
Stam [Sta97]	stochastic	beam	Lagrangian (F.E.M, modal analysis)	spectral
Sakaguchi and Ohya [SO99]	any	articulated	Newtonian (top-down)	explicit (*)
Enhua Wu *et al.* [WCYF99]	procedural	Cosserat rod	independent	static
Giacomo *et al.* [GCF01]	procedural	articulated	Newtonian (iterative)	explicit
Ota *et al.* [OTF+04]	stochastic	articulated	independent	static
Beaudoin and Keyser [BK04]	any	articulated	Newtonian (top-down)	explicit
Akagi and Kitajima [AK06]	simulation	articulated	independent (effective mass)	static
Guo Wu *et al.* [WWG06]	any	articulated	Newtonian (top-down)	explicit
Zhang *et al.* [ZST+06]	stochastic	articulated	independent (*)	static
Sousa [Sou07]	procedural	heuristic	none	static
Zioma [Zio07]	procedural	articulated	independent	static
Weber [Web08]	any	articulated	Newtonian	semi-implicite
Bertails [Ber09]	any	Cosserat rod	Lagrangian (F.E.M)	semi-implicite
Habel *et al.* [HKW09]	stochastic	beam	independent	spectral

Table 5.1: List of all methods presented in this chapter. The (*) indicates that not enough information are given, even implicitely, to know which approach has been used. In each case, we have choosen the only method that can be directly develop from the content found in the respective paper.

chapter 6

Modal Animation

Contents

6.1 Introduction **104**
6.2 Modal Analysis and Modal Animation Framework **105**
6.2.1 Finite Element Method 105
6.2.2 Modal Analysis 108
6.2.3 Animation using the Modes of Deformation 109
6.3 Wind Projection Basis **110**
6.4 Implementation Issues **112**
6.4.1 GPU Implementation 113
6.4.2 Error Correction 113
6.4.3 Configuration 115
6.5 Results **116**
6.6 Limitations and Future Work **117**
6.7 Conclusion **118**

6.1 Introduction

In this chapter we describe our work on mechanical simulation of trees dynamics under wind load which have led to a publication at Eurographics 2009 [DRBR09]. Most of this research was initiated and developed through a interdisciplinary collaboration within the research project *Chene-Roseau* [CHE]. In particular we worked with Ph.D. student Mathieu Rodriguez specialized in the mechanical studies of biological structure and Ph.D. student Lionel Baboud whose work focus on suitable representations of virtual model for real-time application. The goal was twofold: first to provide a software to visualize results of physical simulations and second to develop a new animation technique for real-time applications. To this end we put together computer graphics expertise in real-time animation and rendering of virtual models and knowledge in the mechanics of oscillating structures.

The result we obtained is a real-time method to animate complex scenes of thousands of trees under a user-controllable wind load. We use modal analysis to extract the main modes of deformation which act as a set of reduced degrees of freedom (d.o.f.) of the mechanical model of a 3D tree. The animation of the model at run time is entirely described using this deformation basis. The main limitation of such an approach for real-time applications is that the modal forces, *i.e.,* the forces that are used to compute the dynamics of each of these modes, require an integration of the wind load over the whole tree at each time step of the simulation which can be very computationally expensive.

The main contribution of the work presented in this chapter is a new precomputed basis of the modal stress of the tree under wind load. At runtime, this basis allows to replace the modal projection of the external forces induced by any directional wind by a simple mapping. Moreover, this approach is suitable for implementation on graphics hardware which increases by far the efficiency of the simulation. Finally, the dynamics of the modes is simulated using low computation cost independent of the plant complexity. Displaying a model only requires the minimal computation depending on the level of detail of the rendered geometry. Our method can thus provide real-time animation even for large scenes containing thousands of trees.

Figure 6.1: A single tree and a forest animated with the presented method.

In section 6.2 an overview is given on the mechanical modeling employed and how modal analysis can be used in a animation framework of virtual plants. We then show in section 6.3 how it is possible to pre-compute a projection basis to reduce the cost of the modal integration of the wind load. Using this basis, we can define a direct mapping from the wind parameters (*i.e.,* its direction and speed) to the modal forces.

6.2 Modal Analysis and Modal Animation Framework

In this section, we describe the mechanical model used to compute the dynamics of a tree. We introduce modal analysis and explain how it can be used in a real-time animation framework.

Mechanical modeling and modal analysis have been thoroughly studied. The main goal of the section is to provide a specialized description of the specific model we use for trees. We advise the reader interested in more general mechanical models and more details on modal analysis to refer to specialized literature such as [GR94].

6.2.1 Finite Element Method

Numerical simulation consists in solving the dynamics of a mechanical system over time. In our method, we compute the simulation on a discretized model obtained using the Finite Elements Method (F.E.M). The principle is to decompose the mechanical system in a set of *elements* separated by *nodes*. Typically, nodes are first defined at branches junction then, depending on their length, more may be added along each obtained segment. The continuous system's deformation is then defined as a specific interpolation between the nodes displacement. The software Cast3m 2000 software [CAS] was used to compute finite elements modeling.

In our case as in [Sta97], the system domain Ω is the *lineic* representation of the tree. Then the FEM discretizes Ω as a set of segments Ω^e (see fig. 6.2).

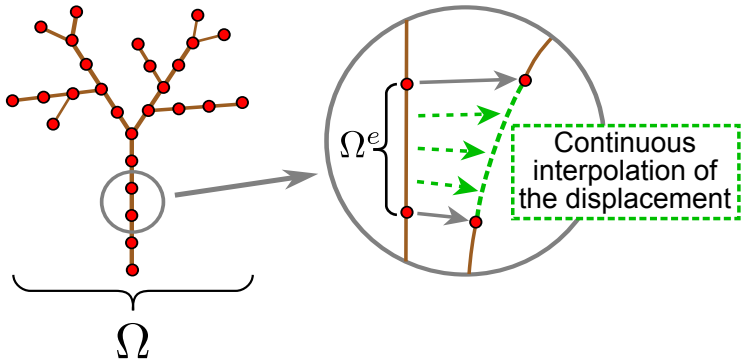


Figure 6.2: Finite element discretization.

In the F.E.M representation of the tree, each element is initially defined in its local frame where the x-axis is in the direction of the segment at rest. This local frame maps to the global frame

according to a rotation matrix R^e. The deformation of an element is then assessed through interpolation of its border nodes displacements: linear for the main axis and Hermitian for the y and z axis.

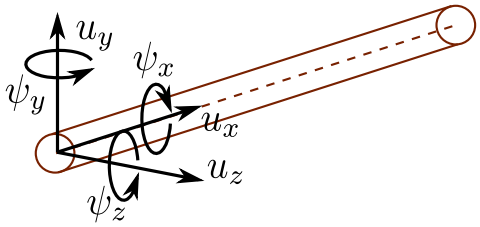


Figure 6.3: The six degrees of freedom of a node.

A node has thus six degrees of freedom (see fig. 6.3): the three spatial displacements (u_x, u_y, u_z), the torsion around the main axis (ψ_x) and the derivatives ψ_y and ψ_z as defined by [GR94]:

$$\psi_y = -\frac{du_z}{dx} \tag{6.1}$$

$$\psi_z = \frac{du_y}{dx} \tag{6.2}$$

Mathematically, the interpolation of an element deformation is modeled such that, at time t, the displacement $\mathbf{u}(p,t)$ of any point $p \in \Omega^e$ is:

$$\mathbf{u}(p,t) = N^e(p)\mathbf{u}^e(t) \tag{6.3}$$

where $\mathbf{u}^e(t)$ is the 12-dimensional vector containing the displacements of the two element border nodes, and $N^e(p)$ is the *shape function matrix* that defines the interpolation between these nodes:

$$N^e(p) = \begin{pmatrix} l_1 & . & . & . & . & . & l_2 & . & . & . & . & . \\ . & h_1 & . & . & . & h_2 & . & h_3 & . & . & . & h_4 \\ . & . & h_1 & . & h_2 & . & . & . & h_3 & . & h_4 & . \\ . & . & . & l_1 & . & . & . & . & . & l_2 & . & . \end{pmatrix} \tag{6.4}$$

where the dots represent zeros and the l_i and h_i are function of p. If L is the length of the element, the l_i represent a linear interpolation and the h_i are the cubic Hermite basis functions:

$$\begin{aligned} \text{let} \quad \pi &= \frac{p}{L} \\ \text{then,} \quad l_1 &= \pi \\ l_2 &= (1-\pi) \\ h_1 &= 1-\pi^2+\pi^3 \\ h_2 &= L\pi(1-\pi)^2 \\ h_3 &= \pi^2(3-\pi) \\ h_4 &= \pi^2(\pi-1) \end{aligned}$$

In accordance with most models of trees, branches are modeled as cylindrical beams [SFL06]. The local stiffness K^e and mass M^e matrices of an element e of length L are defined according to the potential energy v^e and kinetic energy τ^e, with respect to the displacements of the border nodes $\mathbf{u}^e$, by the following relations [GR94]:

$$v^e = \frac{1}{2}\int_0^L EA(\delta u_x)^2 + EI((\delta^2 u_y)^2 + (\delta^2 u_z)^2) + GJ(\delta\psi_x)^2 dx \tag{6.5}$$

$$= \frac{1}{2}\mathbf{u}^{eT} K^e \mathbf{u}^e \tag{6.6}$$

$$\tag{6.7}$$

$$\tau^e = \frac{1}{2}\int_0^L \rho A\left(\dot{u}_x^2 + \dot{u}_y^2 + \dot{u}_z^2\right) + \rho J\,\dot{\psi}_x^2 dx \tag{6.8}$$

$$= \frac{1}{2}\dot{\mathbf{u}}^{eT} M^e \dot{\mathbf{u}}^e \tag{6.9}$$

Here the dot notation refers to time derivatives and $\delta = \frac{d}{dx}$. In our implementation, we use the following values:

$$\begin{aligned} E &= 10^{10} Pa, \\ G &= 2.6E, \\ \rho &= 10^3 kg.m^{-3}, \\ I &= \pi r^4/16, \\ J &= \sqrt{2}I, \\ A &= \pi r^2 \end{aligned}$$

where r is the average radius of the element.

The next step consists in constructing the global mass and stiffness matrices, M and K respectively, of the whole structure from these local matrices. First, each of the local matrices are transformed such that the energies are defined with respect to the nodes displacement expressed in the global frame of the tree. These changes of orientation are done using the rotation matrices R^e (see [Sta97] or [GR94] for a full description of this procedure).

Let $\mathbf{u}(t)$ be the $6 \times n$ vector containing the displacements of all the n nodes of the structure, at time t. The local matrices (of size 12×12) are all added at the suitable locations inside the global matrices M and K (of size $6n \times 6n$) to correspond to the positions insides $\mathbf{u}(t)$ of the d.o.f. of their respective border nodes.

Using the Lagrangian formulation (see section 5.4.1), we obtain the equation of motion describing the dynamics of the whole tree:

$$M\ddot{\mathbf{u}}(t) + C\dot{\mathbf{u}}(t) + K\mathbf{u}(t) = \mathbf{f}(t) \tag{6.10}$$

where $\mathbf{f}(t)$ is the external force (see section 6.3). The middle term of the left hand side of equation 6.10 represents dissipative forces of the system. Based on the *Rayleigh hypothesis*, the damping matrix C is taken as a linear combination of M and K using the user defined coefficients α and β:

$$C = \alpha M + \beta K \tag{6.11}$$

6.2.2 Modal Analysis

Modal analysis is the decomposition of the deformations of a mechanical system into a set of special deformations called *vibration modes*. These modes are the solutions $\mathbf{u}(t)$ of the free vibration equations:

$$M\ddot{\mathbf{u}}(t) + K\mathbf{u}(t) = 0 \tag{6.12}$$

$$\text{such that} \quad \mathbf{u}(t) = \lambda_i(t) \cdot \varphi_i \tag{6.13}$$

where the φ_i are constant vectors called the *modal deformations*, and the $\lambda_i(t)$ are scalar functions of time. Substituting (6.13) in (6.12) gives:

$$M^{-1}K\varphi_\mathbf{i} = \mu_i\varphi_\mathbf{i} \quad \text{with} \quad \mu_i = -\frac{\ddot{\lambda}_i(t)}{\lambda_i(t)} \tag{6.14}$$

As K, M and φ_i are constants, μ_i are constant scalars.

Finally, the eigenvectors and eigenvalues of $M^{-1}K$ are the solutions of equation (6.14). Let Φ be the matrix containing all the modal deformations φ_i, and $\mathbf{q}(t)$ the expression of vector $\mathbf{u}(t)$ in eigenspace:

$$\Phi\mathbf{q}(t) = \mathbf{u}(t) \tag{6.15}$$

Substituting (6.15) in equation (6.10) and multiplying on the left by Φ^T, we get:

$$\underline{M}\,\ddot{\mathbf{q}}(t) + \underline{C}\,\dot{\mathbf{q}}(t) + \underline{K}\,\mathbf{q}(t) = \Phi^T\,\mathbf{f}(t) \tag{6.16}$$

where $\underline{M} = \Phi^T M \Phi$, $\underline{C} = \Phi^T C \Phi$ and $\underline{K} = \Phi^T K \Phi$. A classical result of modal analysis is that these matrices are diagonal [GR94]. We can then rewrite equation (6.16) as $6n$ independent equations such that, for each element q_i of vector $\mathbf{q}$:

$$\ddot{q}_i(t) + \gamma_i \dot{q}_i(t) + \omega_i^2 q_i(t) = \frac{1}{m_i} f_i(t) \tag{6.17}$$

$$\text{with} \quad \begin{cases} \omega_i^2 = \dfrac{k_i}{m_i} \\ \gamma_i \ = \alpha + \beta * \omega_i^2 \end{cases} \tag{6.18}$$

where $f_i(t)$ is the *modal force* induced by the wind (see section 6.3), ω_i the natural frequency, γ_i the damping coefficient of mode i, m_i and k_i the modal mass and stiffness respectively (*i.e.,* the diagonal elements of $\underline{M}$ and $\underline{K}$) and α and β come from equation (6.11).

6.2.3 Animation using the Modes of Deformation

Modal analysis is well suited to our purpose because it extracts the most representative components of the structure deformations. For the case of plants, the vibration modes associated with low frequencies (ω_i) have the most significant (and largely distributed) influence on the final motion [RdLM08]. Typically the first modes (of lower frequencies) are large deformations of the whole tree. Then as the modal frequencies increase, the deformation becomes more and more confined on the last tiny branches (see figure 6.5). Hence only a small subset of the mode matrix Φ needs to be kept in order to model most of the motion complexity.

Equation (6.17) means that each mode behaves as a one-dimensional harmonic oscillator (see figure 6.4). Each vibration mode can be seen as a deformation of the whole structure oscillating independently from the others.

At each time step, the simulation of modal animation consists in the following procedure:

1. Compute $f_i(t)$, the modal projection of the wind.
2. Update the dynamics of the modes using the explicit Euler method:

$$\ddot{q}_i^{t+\delta t} \leftarrow f_i(t)/m_i - \gamma_i \dot{q}_i^t - \omega_i^2 q_i^t \tag{6.19}$$

$$\dot{q}_i^{t+\delta t} \leftarrow \dot{q}_i^t + \delta t\ \ddot{q}_i^{t+\delta t} \tag{6.20}$$

$$q_i^{t+\delta t} \leftarrow q_i^t + \delta t\ \dot{q}_i^{t+\delta t} \tag{6.21}$$

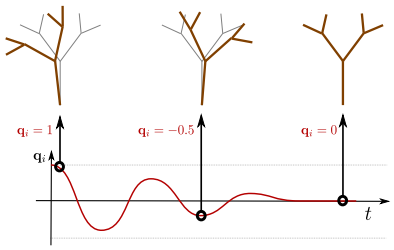


Figure 6.4: A vibration mode is a deformation of the whole tree behaving as a harmonic oscillator.

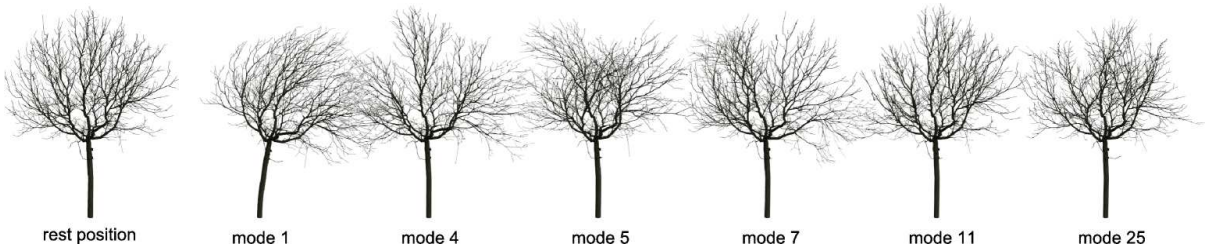


Figure 6.5: The modes of deformation of the walnut model. During animation the final displacement of a tree is a combination of its modal deformation. The modes are sorted in ascending order of their natural frequencies.

3. Compute the updated displacement:

$$\mathbf{u}^{t+\delta t} = \sum_{\forall i} q_i^{t+\delta t} \varphi_i \tag{6.22}$$

Finally, only the evaluation of the modal forces $f_i(t)$ requires costly computations. This is the main difficulty that our technique proposes to resolve. To this end we now describe a basis that can be computed off-line and avoids costly computations at runtime.

6.3 Wind Projection Basis

Integrating a general wind load over the whole tree requires a sum over all the elements (branches) at each time step of the animation (as done by [Sta97]). Doing so prevents real-time animation of sufficiently complex trees. By making the assumption that the wind load is uniform over the whole tree, a projection basis can be precomputed that allows to animate the tree in real-time under the load of any directional wind.

Note that the wind is considered uniform only over each tree but can vary between distinct instances. The speed and direction of the wind are left unconstrained and can be freely controlled at runtime.

We start by modeling the drag load induced by the contact of wind on a rigid structure. As in [SFL06, dL08], we model the wind load on any point $p \in \Omega$ at time t by:

$$\mathbf{f}(p,t) = \frac{1}{2}\rho C_D D(p) \|\mathbf{v}(p,t) - \dot{p}\| (\mathbf{v}(p,t) - \dot{p}) \tag{6.23}$$

Where $\mathbf{v}(p,t)$ is the wind speed vector and $D(p)$ is the diameter of the branch at point p. The air density ρ and the drag coefficient C_D, depending on the shape of the branches, are constant. We merge them as one scalar factor C.

From this local definition, one can derive the modal wind load for each mode i by modal projection [GR94]:

$$f_i = \int_{\Omega} \mathbf{f}(p) \cdot \varphi_i(p) dp \tag{6.24}$$

where we omit the time parameter for clarity, and where $\varphi_i(p)$ is the vector containing the i^{th} modal deformation at p.

To simplify computations, we do not take into account the direction of branches in equation (6.23) for the drag coefficient. This simplification is reasonable because the modes with low frequency are mostly perpendicular to the branches. The influence of the branches parallel to the modal forces is then essentially negligible.

In the case of a freely defined finite element discretization, equation (6.24) is computed as the sum of the elements contribution:

$$f_i = \sum_e \int_{\Omega_e} \mathbf{f}(p) \cdot R^e N^e(p) \varphi_i^e dp \tag{6.25}$$

where R^e is the rotation matrix mapping the local frame of the element e to the global frame. φ_i^e is the vector containing the i^{th} modal deformation of both element's border nodes (in the local frame) and $N^e(p)$ is the *shape function matrix* defining the interpolation between these nodes (see appendix).

To compute the modal forces, equation (6.24) can be discretized over the nodes of the FEM (in the frequency domain) as in [Sta97]:

$$\widehat{f_i} = \sum_{n \in \Omega} \widehat{f}(n) \cdot \varphi_i(n) \tag{6.26}$$

For simple structure it is a suitable approach but for complex trees such a modal projection would be prohibitive. Using a simplified structure is not a solution as it would result in a poor approximation of the effective modal stress.

In our method, to reduce the cost of the modal projection of the wind forces at run time, we focus on extracting the time dependent parameters out of the integral. The remaining part is then precomputed.

The first simplification is to consider R^e as a constant: the wind load is computed on the tree in its rest position. The orientation changes of the branches is not significant as they remain small for natural trees motion. Compensatory models were tested but did not add noticeable improvement. With this approximation, the modal deformation term is constant and only the wind force depends on time. To extract runtime parameters, we assume a wind model such that:

1. From the wind force, we only keep the *mean aerodynamic stress*:

$$\mathbf{f}(p) = C\ D(p)\|\mathbf{v}(p,t)\|\mathbf{v}(p,t)$$

   The remaining part of the external force, called the *aerodynamic damping*, can be compensated by an increase of the damping coefficients α and β from equation (6.11).

2. We define the wind with a variable direction and mean speed, but we assume it to have an uniform value $\mathbf{v}(t)$ over the tree.

Using these simplifications, equation (6.24) becomes:

$$f_i = \int_\Omega C\ D(p)\|\mathbf{v}(t)\|\mathbf{v}(t).\phi_i(p)dp \tag{6.27}$$

$$= C\|\mathbf{v}(t)\|\mathbf{v}(t).\int_\Omega D(p)\phi_i(p)dp \tag{6.28}$$

It results that if we pre-compute the wind projection vectors p_i defined as:

$$p_i = C\int_\Omega D(p)\phi_i(p)dp \tag{6.29}$$

then the modal wind load can be easily computed at runtime using:

$$f_i = \|\mathbf{v}(t)\|\mathbf{v}(t).p_i \tag{6.30}$$

6.4 Implementation Issues

The mechanical modeling, modal analysis and the pre-computation of the projection basis require the use of complex algorithms that must be performed using specialized tools. We used the software Cast3m 2000 [CAS] specialized in the study of mechanical structures using the finite element method.

6.4.1 GPU Implementation

A powerful aspect of the modal representation is that most computations are completely independent. It thus allows to take full advantage of the massive parallel computing power offered by present graphics hardware (GPU). This enables us to animate large forest scenes in real time (see section 6.5).

The two main parameters of our tree model are the number of nodes of the skeleton and the number of modes kept for animation.

1. As can be seen in equation (6.22), the updated position of each node can be computed independently from the others using only the dynamic modal state (q_i). Thus the nodes positions can be stored in a texture in GPU memory (matrix $\mathcal{U}$ of figure 6.6) and updated using a fragment program and an offscreen buffer. Animating the tree mesh can then be done by reading this texture in a dedicated vertex program. Any level-of-detail rendering model using a (possibly time-varying) subset of the animation nodes can be used, without the need to update the remaining unused nodes.

2. Adjusting the number of modes used for the computation of updated nodes positions gives a handful tradeoff between animation quality and frame-rate. For distant views of forests only the first few modes needs to be kept, while adding more modes at close range adds detail to the animation. As for the nodes, each modal state $(q_i, \dot{q}_i)$ can be updated independently from the others, they are stored in a texture in GPU memory (matrix $\mathcal{Q}$ of figure 6.6), and updated by a dedicated fragment shader.

The overall animation process can be summarized in three main steps (see figure 6.6):

1. Update of the modal state for each mode of each tree (item 1 and 2 of section 6.2.3). It requires to compute the local wind speed for each tree instance.

2. Deformation of the skeletons from the rest position (item 3 of section 6.2.3)

3. Rendering each tree with deformed geometry using skinning on the animated skeleton.

6.4.2 Error Correction

Modal deformation being a linear approximation of the displacement of the branches nodes, it is mostly adapted for low amplitude motion. Notably, branches lengths are not kept constant when submitted to strong wind. We cannot rely on using rotating branches (like Habel *et al.* [HKW09] and Zioma [Zio07]) as it would break parallelism between nodes computations and does not extend to complex structures. This classical issue of modal analysis could be addressed by *modal warping* [CK05]. In our specific case of tree animation we observed that when a single mode i is selected, the corrected displacement of each node n lies on a smooth trajectory (see figure 6.7)

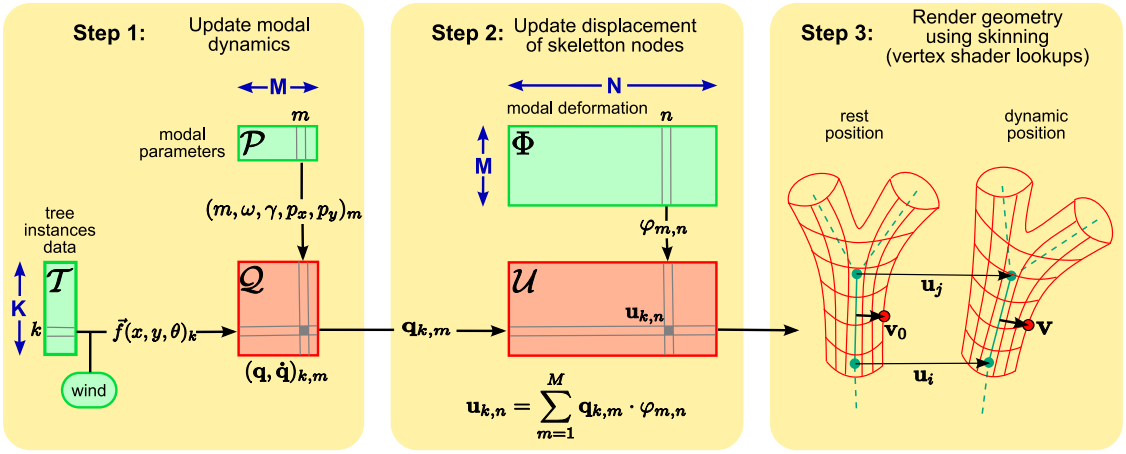


Figure 6.6: Computation steps of the animation framework. K is the number of tree instances, M the number of modes used for the animation and N the number of control points of the skeleton. The colored rectangles represent textures stored in GPU memory. The input matrices $\mathcal{T}$, $\mathcal{P}$, Φ are constant while the matrices $\mathcal{Q}$, $\mathcal{U}$ store the dynamic variables. $\mathcal{T}$ contains the tree instance data (position, orientation); $\mathcal{P}$ contains the modal parameters used to compute the dynamics; Φ contains the modal deformations of the skeleton control points; $\mathcal{Q}$ contains the modal states $(q_i, \dot{q}_i)$; $\mathcal{U}$ contains the skeleton's node displacements. Several representations can be used for the wind such as a procedural equation or a flow array.

which can be faithfully approximated by a quadratic curve $\mathbf{u}$ after a non linear reparametrization s:

$$\mathbf{u}(q) = s \cdot \mathbf{v} + s^2 \cdot \mathbf{w} \tag{6.31}$$

$$\text{with} \quad s = a \arctan(b \cdot q) \tag{6.32}$$

$$\tag{6.33}$$

Our error correction consists then in replacing equation (6.22) by:

$$\mathbf{u}_n = \sum_i s_{i,n} \cdot \mathbf{v}_{i,n} + (s_{i,n})^2 \cdot \mathbf{w}_{i,n} \tag{6.34}$$

$$\text{with} \quad s_{i,n} = a_{i,n} \arctan(b_{i,n} \cdot q_i) \tag{6.35}$$

where vectors $\mathbf{v}_{i,n}$, $\mathbf{w}_{i,n}$ and scalars $a_{i,n}$, $b_{i,n}$ are stored in matrix Φ in figure 6.6, instead of the displacement vectors (roughly doubling the memory size). They are optimized using sampled corrected deformations, corrected by pulling each node towards its parent to match initial branches lengths, in a depth-first fashion. The results show that this approach yields a sufficiently good correction for branches lengths to allow the application of a strong wind load.

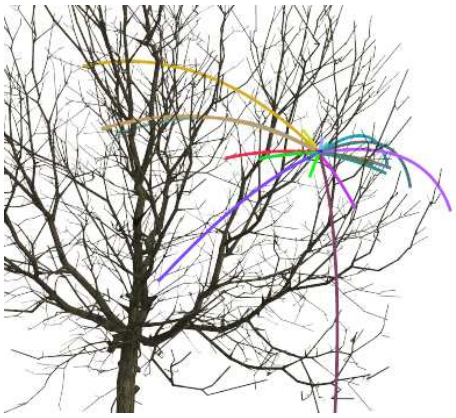


Figure 6.7: Correct trajectories obtained by selecting each mode successively (each color corresponds to one mode). These trajectories can be faithfully approximated by second degree polynomial curves.

6.4.3 Configuration

Instead of looking for an all-automatic framework, we believe that meaningful controls are important. We now present few parameters that can be derived from the modal decomposition and provide useful configuration possibilities.

The modes are computed only from the branch structure. The insertion of tiny elements such as leafs in the modal analysis procedure is not well defined and may bring numerical instabilities. For now, we consider the leafs solely as a geometric outfit, but their omission has still an influence on the overall dynamics. Two main characteristics are missing: their mass (*i.e.,* their inertia), and their aerodynamic damping.

The first two controls are the damping coefficients α and β from equation (6.11). As we can see from equation (6.18), they can be selected outside modal decomposition. In theory, they represent the internal damping of the branches. Here however they can be used to compensate for the aerodynamic damping.

The last control enables to change the modal inertia and has thus a great impact on simulated dynamics. Let us imagine we want to change the mass matrix of equation (6.10), by:

$$M' = m * M \tag{6.36}$$

With some scalar m. Then the modal mass m_i in equation (6.16) will also be multiplied by m. From equation (6.18), modal frequencies and damping are also affected:

$$\omega'_i = m^{-\frac{1}{2}} * \omega_i \quad (6.37)$$

$$\gamma'_i = m * \alpha + \beta * \omega'_i \quad (6.38)$$

Finally, setting m with a high value will act as the tree is heavier, *i.e.,* with more inertia, which can compensate for the leaves weight, or simulate a tree of bigger scale. Inversely, a value less than one enables the use of the same structure to represent a lighter plant.

The important point is that all of these parameters can be changed at run time without additional computations.

6.5 Results

We have tested the presented method on various models on an Intel Xeon 3.2 GHz with a GeForce 8800GTS. The following results were obtained on a Walnut tree digitized from precise measurements (from [SRG97], see figure 6.9) and a virtual model generated using the method of Runions *et al.* [RLP] (see figure 6.8).

Figure 6.8: The oak tree model.

Using our implementation we use a *level of details* system (LOD). When activated, we are able to animate and render over 4000 trees while maintaining real-time frame-rates (30 Hz minimum). For the walnut tree with 3437 nodes, the undecimated version of the mesh is made of approximately 120000 vertices while the most decimated level of details only uses 300 vertices. However our LOD implementation is unsophisticated and could be much more optimized. The only significant data stored in GPU memory is the nodes displacements texture (array $\mathcal{U}$ in figure 5) whose size is $K \times N$ (*e.g.* 10 Mb for 1000 trees with 3000 control points each).

Tables 6.1 and 6.2 show that it is always possible to reach high frame-rates by making a tradeoff between the number of trees and the animation level of detail (*i.e.,* the number of nodes and the number of modes).

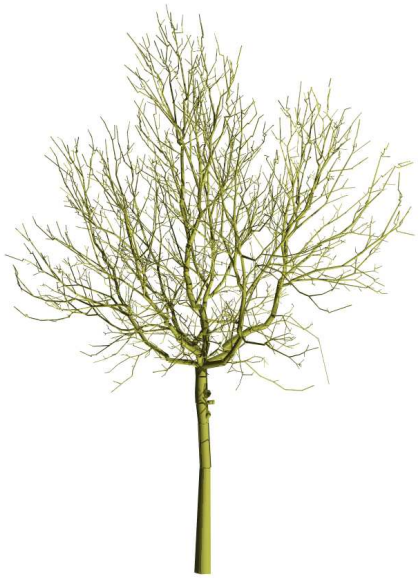


Figure 6.9: Digitalized walnut tree.

K \ N	20	100	500	1000	2000	8000
100				712	385	103
1000		690	169	87	43	11
8000	426	102	21	11		

Table 6.1: Influence of the number of trees (K) and the number of nodes (N) on animation frame-rates (in Hz), with 25 modes and rendering deactivated.

M \ K	500	1000	2000	4000
5	73	39	19	10
15	57	29	14	7
25	43	22	11	6

Table 6.2: Influence of the number of modes (M) and the number of trees (K) on the frame-rate (in Hz). The model has 3437 nodes and approximately twice as much vertices, and the LOD system is deactivated to ease interpretation (*i.e.,* every node is always updated).

6.6 Limitations and Future Work

As explained in section 6.2.3, the omission of the vibration modes with highest frequencies strongly reduces the computational cost of simulation but also removes the highly fluctuating motion of tiny branches and leaves. However this can be replaced by using techniques such as [OTF+04] for close view and animated textures further away. In our implementation, the leaves

rigidly follow the nodes they are attached to. But the complexity of branches animation already provides very convincing motion.

Mathematically, the main limitation of our method is the assumption of spatial uniformity of the wind over each tree. In particular the attenuation of the wind by the tree is not taken into account (*i.e.,* the branches in the 'back' of the tree should receive less wind). It does not lower visual realism as the modal dynamics keeps natural oscillatory behaviors. However pre-computing a more expressive basis would allow to increase animation precision.

Finally the parallelism of the method coupled with efficient GPU programming enables the animation of thousands of trees. It would require a more aggressive level-of-detail scheme to allow real-time rendering of even larger forest scenes.

6.7 Conclusion

We showed a new approach to compute the modal projection of the wind load allowing a drastic reduction of computations at runtime. To this end, we introduce a precomputed basis for the projections of interactive directional wind in a modal animation framework.

Our implementation of the method shows that it is possible to efficiently animate and render thousands of trees. We think the presented technique is perfectly adapted to real-time applications such as computer games or simulators.

chapter 7

Conclusion

My work during this Ph.D. has addressed many theoretical issues related to the acquisition and the creation of plants motion.

First an experimental protocol has been developed for the video recording of real animated trees which has allowed the acquisition of quality data on tree motion.

We have also proposed two new computer vision methods for the extraction of 2D motion out of such videos. One aims at increasing the speed of the exhaustive comparison of images areas in a region matching approach. The other increases the robustness of differential tracking methods.

As all automatic tracking techniques do not always produce reliable motion data, particularly for the case of video of animated foliage, we have then studied how a suitable user interface can help obtaining better results. First, efficient approaches for the quick correction of such data has been developed and second, we have designed an alternative method to manually input the whole trajectory of moving features in a video.

My main contributions has been to develop two new animation methods for virtual plants:

- One is the first markerless motion capture techniques that reproduces the plant motion observed in a video onto virtual models. We have shown that careful statistical studies of the extracted foliage motion allows the estimation of a branches structure that hold this motion.
- The second animation method is on computer simulation of tree dynamics in response to interactive wind. We introduced the pre-computed *wind projection basis* that allows the real-time animation of forest containing thousands trees.

For both method, I obtained a publication in well-known international conferences: the first has been published in the Symposium on Computer Animation 2006 [DRF06] and the second [DRBR09] has been accepted at Eurographics 2009.

Finally, the interdisciplinary collaboration within the research project Chene-Roseau has allowed me to participate to longer term researches with influences outside of the computer graphics field.

In particular, my participation has concerned the development and use of the video recording protocol and the acquisition of complex sets of real data on trees dynamics. In addition, my collaboration with Ph.D. student Mathieu Rodriguez has led to the development of the real-time computer simulation method for the animation of trees.

My work concerning the use of videos of animated foliage and the expertize I develop on this particular issue is still brought to the Chene-Roseau project. It will continue in the future through a post-doctorate position at the LadHyX laboratory.

List of Figures

1.1 Biological structure of trunk and branches. 8
1.2 Classification of possible hierarchical structure of trees. The four more common structures are mark with an asterisk. (Image taken from [Hal05]) 8
1.3 Experiment showing how important a factor the wind is to the growth in height of crop. A patch of a crop field has its lateral motion (due to wind) restricted. It results in a growth much higher of the crop. (Image and experiment presented in [MC06]) 9
1.4 Breaking of plants at different scales due to wind action. Studies describe the phenomena as a consequence of a *locking mechanism* between the wind frequency and the plants oscillation. 10

2.1 (a) and (b) Devices to record contacts of a horse hoof with the ground. (c) Setup to record all the contacts of the hooves of a horse during locomotion using device 1 or 2. 19
2.2 Excerpt from two of the Muybridge sequences. Top: excerpt from sequence *Animals and Movements, Horses, Gallop; thoroughbred bay mare, Annie G.* The first image proves Marey's theory that all hooves of horses leave the ground at some point during gallop. Bottom: excerpt from sequence *Movements, Female, Jumping: running straight high jump.* 20
2.3 Two sequences from Marey's results. 20
2.4 The chronophotographic gun design by Marey. A camera is mounted on a modified traditional gun where a photographic film is stored in the chamber. It allowed capturing up to twelve pictures by second. 21
2.5 Tracking a human in motion using an infrared optical system. The subject has retroreflective markers set at specific parts if her body. A projector (red light at the top of the image) emits infrared light reflected by the markers. It is then detected in video sequences that film the subject. 22
2.6 The magnetic tracking system is composed of a main unit (box in the middle of image (a)) and of a transmitter that generates a magnetic field (sphere on the right of image (a)). Up to four magnetic receivers can be connected to the main unit and tracked in real time. A software developed at the INRA controls all this system. 23

2.7 (a) Tree used for experimentation. (b) Experimental layout: Two cameras film the tree from one side, four magnetic markers are attached to representative branches and a sonic anemometer records the wind speed next to the tree. (c) An image taken by camera 2. The positions of the magnetic markers are indicated by red arrows, and the LED device used for synchronization is indicated by a black arrow. 24
2.8 Representation of the *pinhole* camera. On the right, the calibration rig for which we know the shape, *i.e.,* the 3D positions Q_i (in the reference frame of the calibration rig). The projection matrix of the camera is calibrated such that it projects them onto their observed projections in the image plane, *i.e.,* the 2D positions q_i. 25
2.9 The calibration rig we designed. 27
2.10 Subset images of a sequence used for calibration. 28
2.11 3D reconstruction of the four markers position from videos. For each subfigure: The columns are for the x, y and z coordinates respectively; The top row graphs show the trajectories of the markers extracted from magnetic tracking (red) and from 3D reconstruction from video (blue); The second row shows their spectrum obtained using discrete Fourier transform. Data are given in centimeters. 30
2.12 Difference of Gaussians $(G((\sigma) - G(k \cdot \sigma))$. 33
2.13 A simple image where the numbers of non-zero eigenvalue of $A^T A$ are indicated. Over the black or white area, the gradient is null. In equation (2.9), the matrix A contains only zeros and the eigenvalues of $A^T A$ are both null. For each edge (horizontal or vertical) all the pixels have the same gradient, thus the rows of A are equal and $A^T A$ has one non-zero eigenvalue. Finally at the corner between both edges, some pixels of the neighborhood (the rows of A) have a strong vertical component and some have a strong horizontal component. Then $A^T A$ is well conditioned and has two strong eigenvalues. 36
2.14 Graph picturing the Lucas-Kanade method in the one-dimensional case (thus, without the aperture problem). The estimate u^* of the displacement u of pixel p is obtained using equation (2.6), *i.e.,* in 1D $I_x u^* - I_t = 0$. For both graphics (a) and (b), the black curve represents the intensity of the original one-dimensional image at time t and the red curve is the same image at time $t+1$ that moved to the right. (a) The image is smooth, and the estimate u^* of u at p is good (but too long). An iterative process should converge. (b) A deformation of high frequency with low amplitude has been added at p. As a consequence the estimate u^* is in the opposite direction of the real displacement. A divergence of iterative estimation is most probable. 37
2.15 Case of a sequence of one-dimensional images. Each vertical line (red) represents one image. In (a) for each pixel of each image, an estimation of the motion is computed. In (b) particles are initiated in the first image then tracked recursively along the image sequence. 37

2.16 Pyramids of images used to accelerate the box matching method. L_0, L_1 and L_2 are levels of the pyramids (of depth three) from 0 for the full resolution to 2 for the lower resolution. P_1 is the pyramid of the region to be tracked in the first image (R_0 is marked with an $\times$ in its initial resolution). P_2 is the pyramid of the search window in the second image. Once the box matching algorithm as been computed for all the positions in P_2 at level L_2, the best displacements (ex: the 5% with minimum image difference) are projected in the level L_1 (represented by the striped region). At the lower level, the box matching is done over these pixels only, and best displacements are further projected. At level L_0 the displacement with minimum image difference is taken as solution (the red region marked with a red $\times$). 40

3.1 Software to extract motion in video using both automatic tracking and user interaction. The circles drawn on top of the video image are the features to track. 46
3.2 Flowchart describing the organization of user's work using our application for the task of motion extraction of selected targets of the video. 47
3.3 The conical representation of the HSL color space. The hue coordinate (H) is the radial position around the cone, the saturation (S) is the radius to the axis of the cone and the lightness is the height of the cone. 48
3.4 Superposition of motion information using speed to HSL mapping. (a) Original image from a video sequence. Structure is *invisible*. (b) With the motion information, the red circle indicates an emphasized structure. 49
3.5 Using direction clamping on displayed color. (a) With everything displayed. (b) The display is restricted to features with a specific motion direction. One structure is pointed out (red circle). 50
3.6 Graphical interface of the video player. On the right, the user can configure the two frame-rates (quick and slow) at which the video can be played. 50
3.7 Keyboard shortcuts used by our application. The layout of the control buttons allows simple and efficient user interaction. (a) Used by the time correction tool described in 3.5.2. (b) Used by the space correction tool described in 3.5.1. . . . 51
3.8 The four Hermite basis functions used for cubic interpolation. 52
3.9 An example of the speed estimation for a key-frame at time k. In this example, even if $p(k)$ is almost spatially equidistant to $p(k_0)$ and $p(k_1)$ in the image space, the ratio between time interval is $(k_1 - k) = 2(k - k_0)$, so $\alpha \approx 0.667$: s_0 has twice more influence than s_1. 53
3.10 Typical tasks to which Fitts law and steering law apply. 54
3.11 Displaying the motion curve of a selected target. The red dots are key-frames. . . 55
3.12 Spatial correction. The user corrects the position at one frame, and it is propagated along the motion curve. In light brown is the curve segment (initial curve) that is affected. The user has previously indicated the bound of this segment on one side. The other bound is the next key-frame (red dot). In light red curve with white points is the corrected motion curve, the points are the frames positions. . . 57

3.13 Correction of the motion curve only in time. The user can move the features only along the curve. The white dots are the new positions at all frames. 58

4.1 (a) and (b) are the two first topos and (c) their associated chronos obtained using BOD on the optical flow of a video of a canopy animated by the wind. In (a) and (b) iso-contours of the topos magnitude are drawn. (images taken from [PdLM06]) 62
4.2 Comparison between a manual clustering that shows an expected classification of the features in seven groups, results from clustering using euclidean distance in the image and using the distance of movement. 64
4.3 (a) Unstructured motion, (b) Strong wind inducing a single overall motion. 65
4.4 Comparison of the ground truth classification with the results obtained with our automatic method. 66
4.5 Hierarchical clustering and cut-off line for the determination of the number of classes identified as terminal groups. 68
4.6 (a) The position of a node p_i is initiated as the average (p_i^*) of its children node p_n and p_m. Then it is optimized to minimize equation (4.11). (b) and (c) Extracted structure before and after the optimization. 70
4.7 Final geometrical hierarchy of groups. 71
4.8 3D geometrical structure with texture sprites for leaves. 71
4.9 3D geometrical tree structure used as a control skeleton for animation by skinning. 72
4.10 Video footage controlling 3D animation of the *ficus* model (motion blur corresponds to a 1/50s shutter speed). 73
4.11 Original footage of the *xmas* model and synthesis target for animation of a complex 3D model of shrub 73
4.12 Three main steps of the interactive modeling of shrubs that fit the control skeleton. 73
4.13 Result obtained on video sequences of two full grown tree. 74

5.1 The typical deformations of branch segment. 81
5.2 Tradeoff between precision and simulation complexity of the method of Akagi *et al.* [AK06]. The grid can be defined with any resolution (from the lower (a) to the higher (c)). Then the forces that the tree apply on the wind is proportional to wind speed multiplied by the quantity of wood and leaves in each grid cell. 84
5.3 (a)-(d) 2D wind flow primitives. (d) Example of the sum of two primitives, a uniform and a vortex wind flow. Their sum is still solution of the simplified Navier-Stokes equation presented by Wejchert and Haumann [WH91]. 85
5.4 (a) and (b) Articulated structure made of a set of rigid segments. (c) Model of one element of the structure. F_0 and F_1 are the reference frame before and after the element respectively. F_1 is obtained after a rotation and translation of F_0. . . 88

5.5 Euler-Bernoulli beam model. The bending and torsion are defined with respect to the spatial derivatives of the displacement expressed in cartesian coordinates (here w). F_0 and F_1 are the reference frames before and after the element respectively. The transformation from F_0 to F_1 is only a translation equal to the displacement of the end of the beam. 89
5.6 Cosserat rod model. The lineic displacement is defined with respect to the deformation of the rod, *i.e.,* bending and torsion (only bending k is represented here). F_0 and F_1 are the reference frame before and after the element respectively. The transformation from F_0 to F_1 is the integral of the deformation along the rod. . . 91

6.1 A single tree and a forest animated with the presented method. 104
6.2 Finite element discretization. 105
6.3 The six degrees of freedom of a node. 106
6.4 A vibration mode is a deformation of the whole tree behaving as a harmonic oscillator. 110
6.5 The modes of deformation of the walnut model. During animation the final displacement of a tree is a combination of its modal deformation. The modes are sorted in ascending order of their natural frequencies. 110
6.6 Computation steps of the animation framework. K is the number of tree instances, M the number of modes used for the animation and N the number of control points of the skeleton. The colored rectangles represent textures stored in GPU memory. The input matrices $\mathcal{T}$, $\mathcal{P}$, Φ are constant while the matrices $\mathcal{Q}$, $\mathcal{U}$ store the dynamic variables. $\mathcal{T}$ contains the tree instance data (position, orientation); $\mathcal{P}$ contains the modal parameters used to compute the dynamics; Φ contains the modal deformations of the skeleton control points; $\mathcal{Q}$ contains the modal states $(q_i, \dot{q}_i)$; $\mathcal{U}$ contains the skeleton's node displacements. Several representations can be used for the wind such as a procedural equation or a flow array. 114
6.7 Correct trajectories obtained by selecting each mode successively (each color corresponds to one mode). These trajectories can be faithfully approximated by second degree polynomial curves. 115
6.8 The oak tree model. 116
6.9 Digitalized walnut tree. 117

Bibliography

[AK06] Yasuhiro Akagi and Katsuhiro Kitajima. Computer animation of swaying trees based on physical simulation. *Computers & Graphics*, 30(4):529–539, 2006. 5.2.1, 5.2, 5.3.1, 5.4.2, 5.5.1, 5.6, 7

[AMM90] A. Abdel-Malek and V. Marmarelis. A model of human operator behavior during pursuit manual tracking-what does it reveal? *Systems, Man and Cybernetics, 1990. Conference Proceedings., IEEE International Conference on*, pages 674–676, Nov 1990. 3.4.2

[AZ97] Johnny Accot and Shumin Zhai. Beyond fitts' law: models for trajectory-based hci tasks. In *CHI '97: CHI '97 extended abstracts on Human factors in computing systems*, pages 250–250, 1997. 3.4.2, 3.10(b)

[BAC+06] Florence Bertails, Basile Audoly, Marie-Paule Cani, Bernard Querleux, Frédéric Leroy, and Jean-Luc Lévêque. Super-helices for predicting the dynamics of natural hair. In *ACM Transactions on Graphics (Proceedings of the SIGGRAPH conference)*, August 2006. accepted to Siggraph'06. 5.3.3, 5.4.4

[Bar96] David Baraff. Linear-time dynamics using lagrange multipliers. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 137–146, 1996. 5.4.3

[Ber06] Florence Bertails. *Simulation de Chevelures Virtuelles*. PhD thesis, Institut National Polytechnique de Grenoble, 2006. 5.3.3

[Ber09] F. Bertails. Linear time super helices. *Computer Graphics Forum (proceedings of Eurographics'08)*, 2009. 5.3.3, 5.4.4, 5.5.2, 5.6

[BK04] Jacob Beaudoin and John Keyser. Simulation levels of detail for plant motion. In *SCA '04: Proceedings of the 2004 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 297–304, 2004. 5.3.1, 5.4.2, 5.4.3, 5.5.2, 5.6

[BM98] C. Bregler and J. Malik. Tracking people with twists and exponential maps. In *Conference on Computer Vision and Pattern Recognition (CVPR '98)*, pages 8–15, 1998. 4.2.2

[BMP04] Christoph Bregler, Jitendra Malik, and Katherine Pullen. Twist based acquisition and tracking of animal and human kinematics. *International Journal of Computer Vision*, 56(3):179–194, 2004. 2.2.3, 4.2.2

[Bou00] Jean-Yves Bouguet. Pyramidal implementation of the Lucas-Kanade feature tracker. Intel Corporation, Microprocessor Research Labs report, 2000. 2.4.3

[BW97] David Baraff and Andrew Witkin. Implicit methods for differential equations (in physically based modeling: Principles and practice). In *SIGGRAPH '97: ACM SIGGRAPH 1997 courses*, 1997. 5.5.2

[CAS] Cast3m 2000. http://www-cast3m.cea.fr/. 6.2.1, 6.4

[CC09] Eugène Cosserat and François Cosserat. *Théorie des corps dédormables*. Hermann, 1909. 5.3.3

[CHE] Chene-roseau. http://www.ladhyx.polytechnique.fr/public_cr/index.html. Mechanism of dynamic interactions between wind and flexible plants, project granted by the National Research Agency (ANR), France. 1.2, 2.3, 6.1

[CK98] J.P. Costeira and T. Kanade. A multibody factorization method for independently moving objects. *IJCV*, 29(3):159–179, 1998. 4.2.1

[CK05] Min Gyu Choi and Hyeong-Seok Ko. Modal warping: Real-time simulation of large rotational deformation and manipulation. *IEEE Transactions on Visualization and Computer Graphics*, 11(1):91–101, 2005. 6.4.2

[Coh92] Michael F. Cohen. Interactive spacetime control for animation. *SIGGRAPH Computer Graphics*, 26(2):293–302, 1992. 3.4.2

[Cor71] R.M. Cormack. A review of classification. *J. of the Royal Statistical Society, Series A*, 134(3):321–367, 1971. 4.3.2

[CRM00] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, 2:142–149, 2000. 2.4.2, 2.5.2

[dL08] Emmanuel de Langre. Effects of wind on plants. *Annual Review of Fluid Mechanics*, 40:141–168, 2008. 6.3

[DRBR09] Julien Diener, Mathieu Rodriguez, Lionel Baboud, and Lionel Reveret. Wind projection basis for real-time animation of trees. *Computer Graphics Forum (Proceedings of Eurographics 2009)*, 28(2), mar 2009. 1.2, 6.1, 7

[DRF06] Julien Diener, Lionel Reveret, and Eugene Fiume. Hierarchical retargetting of 2d motion fields to the animation of 3d plant models. In *ACM-SIGGRAPH/EG Symposium on Computer Animation (SCA)*. ACM-Siggraph/Eurographics, 2006. 1.2, 7

[EMF03] Luis Carlos Yano Endo, Carlos Hitoshi Morimoto, and Antonio Elias Fabris. Real-time animation of underbrushes. In *WSCG '03: Proceedings of the 11th international conference in central europe on computer graphics, visualization and computer vision*, 2003. 5.3.1

[ES00] Kevin C. Engel and John F. Soechting. Manual tracking in two dimensions. *J Neurophysiol*, 83(6):3483–3496, 2000. 3.4.2

[FAI+05] David A. Forsyth, Okan Arikan, Leslie Ikemoto, James O'Brien, and Deva Ramanan. Computational studies of human motion: part 1, tracking and motion synthesis. *Foundations and Trends in Computer Graphics and Vision*, 1(2-3):77–254, 2005. 2.2.3, 4.2.2

[Fea83] R. Featherstone. The calculation of robot dynamics using articulated-body inertias. *International Journal of Robotics Research*, 2:13–30, 1983. 5.3.1, 5.4.3

[FH75] K. Fukunaga and L. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *Information Theory, IEEE Transactions on*, 21(1):32–40, 1975. 2.5.2

[Fit54] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, 1954. 3.4.2

[FRDC04] Laurent Favreau, Lionel Reveret, Christine Depraz, and Marie-Paule Cani. Animal gaits from video. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA) 2004*, 2004. 2.2.3, 4.2.2

[G.06] Johansson G. Visual perception of biological motion and a model for its analysis. *Am. J. Bot.*, 93(10):1522–1530, 2006. 3.4.2

[Gav99] D. M. Gavrila. The visual analysis of human movement: a survey. *Computer Vision and Image Understanding*, 73(1):82–98, 1999. 2.2.3

[GCF01] Thomas Di Giacomo, Stéphane Capo, and François Faure. An interactive forest. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*, pages 65–74. Springer, 2001. 5.2.2, 5.3.1, 5.4.3, 5.5.2, 5.6

[Gor87] A.D. Gordon. A review of hierarchical classification. *J. of the Royal Statistical Society, Series A*, 150:119–137, 1987. 4.3.2

[GR94] M. Géradin and D. Rixen. *Mechanical Vibrations: Theory and Application to Structural Dynamics*. 1994. 5.3.2, 6.2, 6.2.1, 6.2.1, 6.2.1, 6.2.2, 6.3

[Had06] S. Hadap. Oriented strands: dynamics of stiff multi-body system. In *SCA '06: Proceedings of the 2006 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 91–100. Eurographics Association, 2006. 5.2

[Hal05] Francis Hallé. *Plaidoyer pour l'arbre*. Actes Sud, 2005. 1.1.1, 1.2, 7

[Har04] M.J. Harris. Fast fluid dynamics simulation on the gpu. In *GPU Gems: Programming Techniques, Tips, and Tricks for Real-Time Graphics*, pages 637–665. Addison-Wesley, 2004. 5.2.1

[HBW99] Seon M. Han, Haym Benaroya, and Timothy Wei. Dynamics of transversely vibrating beams using four engineering theories. *Journal of Sound and Vibration*, 225(5):935 – 988, 1999. 5.3.2, 5.3.2

[HKW09] Ralf Habel, Alexander Kusternig, and Michael Wimmer. Physically guided animation of trees. *Computer Graphics Forum (proceedings of Eurographics'08)*, 2009. 5.2.3, 5.2.3, 5.2.3, 5.3.2, 5.4.2, 5.5.1, 5.5.3, 5.6, 6.4.2

[Hof91] E. R. Hoffmann. Capture of moving targets: A modification of fitts' law. *Ergonomics*, 34(2):211–220, 1991. 3.4.2

[HS80] Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. Technical report, 1980. 2.4.3

[HTF01] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer-Verlag, 2001. 4.3.2

[IMT99] Takeo Igarashi, Satoshi Matsuoka, and Hidehiko Tanaka. Teddy: A sketching interface for 3d freeform design. In *Proc. of ACM SIGGRAPH'99*, pages 409–416, 1999. 3.4.2, 4.4.1

[Jam03] Kenneth R. James. Dynamic loading of trees. *Journal of arboriculture*, 29:165–171, November 2003. 2.2.2

[JHA06] Kenneth R. James, Nicholas Haritos, and Peter K. Ades. Mechanical stability of trees under dynamic loads. *Am. J. Bot.*, 93(10):1522–1530, 2006. 2.2.2

[LK81] B. Lucas and T. Kanade. Iterative image registration technique with an application to stereo vision. In *Proc. of 7th International Joint Conference on Artificial Intelligence (IJCAI)*, pages 674–679, 1981. 2.4.3, 2.4.3, 2.4.3

[Low99] D.G. Lowe. Object recognition from local scale-invariant features. *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, 2:1150–1157, 1999. 2.4.2

[MC06] Bruno Moulia and Catherine Coutand. Thigmomorphogenetic acclimation of plants to moderate winds as a major factor controlling height growth and biomass distribution in crops, as demonstrated in alfalfa (medicago sativa l.). In *Proceeding of the 5th Plant Biomechenics Conference*, 2006. 1.3, 7

[Ono97] Hiromi Ono. Practical experience in the physical animation and destruction of trees. In *Eurographics Workshop on Computer Animation and Simulation, EGCAS 1997*, pages 53–64. Springer, 1997. 5.2.3, 5.2.3, 5.3.1, 5.4.2, 5.5.2, 5.6

[OPE] Opencv. http://sourceforge.net/projects/opencv/. Intel Corporation. 2.5

[OTF+04] Shin Ota, Machiko Tamura, Tadahiro Fujimoto, Kazunobu Muraoka, and Norishige Chiba. A hybrid method for real-time animation of trees swaying in wind fields. *The Visual Computer*, 20:613–623(11), 2004. 5.2.3, 5.2.3, 5.3.1, 5.4.2, 5.5.1, 5.6, 6.6

[Par62] E. Parzen. On estimation of a probability density function and mode. *Annals of mathematical statistics*, 33:1065–1076, 1962. 2.5.2

[PC01] Frank Perbet and Maric-Paule Cani. Animating prairies in real-time. In *I3D '01: Proceedings of the 2001 symposium on interactive 3D graphics*, pages 103–110, 2001. 5.2.2

[PdLM06] Charlotte Py, Emmanuel de Langre, and Bruno Moulia. A frequency lock-in mechanism in the interaction between wind and crop canopies. *Journal of Fluid Mechanics*, 568:425–449, 2006. 4.2.1, 4.1, 4.2.1, 7

[PdLMH05] Charlotte Py, Emmanuel de Langre, Bruno Moulia, and Pascal Hemon. Measurement of wind-induced motion of crop canopies from digital video images. *Agricultural and Forest Meteorology*, 130:223–236, 2005. 4.2.1

[PS88] Heinz-Otto Peitgen and Dietmar Saupe, editors. *The Science of Fractal Images*. Springer-Verlag New York, Inc., New York, NY, USA, 1988. 5.2.3

[PSE03] Jovan Popovic, Steven Seitz, and Michael Erdmann. Motion sketching for control of rigid-body simulations. *ACM Transactions on Graphics*, 22(4):1034–1054, October 2003. 3.4.2

[QT] Qt cross-platform application framework. http://trolltech.com/products/qt/. Developed by Trolltech. 2.5

[RdLM08] Mathieu Rodriguez, Emmanuel de Langre, and Bruno Moulia. Effects of architecture and allometry on tree vibration modes. *Submitted to American Journal of Botany*, 2008. 6.2.3

[RFL96] D. Raboud, M. G. Faulkner, and A. W. Lipsett. A segmental approach for large three-dimensional rod deformations. *International Journal of Solids and Structures*, 33(8):1137 – 1156, 1996. 5.3.3

[RLP] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. Modeling trees with a space colonization algorithm. 6.5

[SF92] Mikio Shinya and Alain Fournier. Stochastic motion-motion under the influence of wind. *Computer Graphics Forum*, 11(3):119–128, 1992. 5.2.3, 5.2.3, 5.3.2, 5.5.2, 5.6

[SFL06] Damien Sellier, Thierry Fourcaud, and Patrick Lac. A finite element model for investigating effects of aerial architecture on tree oscillations. *Tree Physiology*, 26:799–806, 2006. 6.2.1, 6.3

[SJF03] M. Sun, A. Jepson, and E. Fiume. Video input driven animation (vida). In *International Conference on Computer Vision (ICCV) 2003*, 2003. 4.2.2, 4.6

[Smi07] C. Sminchisescu. Learning and Inference Algorithms for Monocular Perception. Applications to Visual Object Detection, Localization and Time Series Models for 3D Human Motion Understanding, 2007. University of Bonn, Faculty of Mathematics and Natural Sciences. Habilitation Thesis. 2.2.3

[SO99] Tatsumi Sakaguchi and Jun Ohya. Modeling and animation of botanical trees for interactive virtual environments. In *Proceedings of the ACM symposium on Virtual reality software and technology (VRST '99)*, pages 139–146, 1999. 5.3.1, 5.4.3, 5.5.2, 5.6

[Sou07] Tiago Sousa. *Gpu gems 3*, chapter Vegetation Procedural Animation and Shading in Crysis (chapter 16). Addison-Wesley Professional, 2007. (Crytek). 5.2.2, 5.6

[SRG97] H. Sinoquet, P. Rivet, and C. Godin. Assessment of the three-dimensional architecture of walnut trees using digitising. *Silva Fennica*, 31:265–273, 1997. 2.3.1, 6.5

[SS08] Ryan Schmidt and Karan Singh. Sketch-based procedural surface modeling and compositing using Surface Trees. volume 27, pages 321–330, 2008. Proceedings of Eurographics 2008. 3.4.2

[ST94] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proc. IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 593–600, 1994. 2.4.3, 2.5.3

[Sta97] Jos Stam. Stochastic dynamics: Simulating the effects of turbulence on flexible structures. *Compututer Graphics Forum*, 16(3):159–164, 1997. 5.2.3, 5.2.3, 5.3.2, 5.4.4, 5.5.3, 5.6, 6.2.1, 6.2.1, 6.3, 6.3

[Sta99] Jos Stam. Stable fluids. In *SIGGRAPH '99: Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 121–128, 1999. 5.2.1

[TBvdP07] Matthew Thorne, David Burke, and Michiel van de Panne. Motion doodles: an interface for sketching character motion. In *SIGGRAPH '07: ACM SIGGRAPH 2007 courses*, page 24, 2007. 3.4.2

[TYAB01] Lorenzo Torresani, Danny Yang, Gene Alexander, and Christoph Bregler. Tracking and modelling non-rigid objects with rank constraints. In *Proc. IEEE CVPR 2001*, 2001. 4.2.1

[VAV+07] Daniel Vlasic, Rolf Adelsberger, Giovanni Vannucci, John Barnwell, Markus H. Gross, Wojciech Matusik, and Jovan Popovic. Practical motion capture in everyday surroundings. *ACM Transaction on Graphics, Proceedings of the 2007 SIGGRAPH conference*, 26(3):35, 2007. 2.2.2

[VBTS07] David Vanderhaeghe, Pascal Barla, Joëlle Thollot, and François Sillion. Dynamic point distribution for stroke-based rendering. In *Rendering Techniques 2007 (Proceedings of the Eurographics Symposium on Rendering)*, pages 139–146, 2007. 2.5.3

[WCYF99] Enhua Wu, Yanyun Chen, Tao Yan, and Jinhui Feng. Reconstruction and physically-based animation of trees from static images. In *Eurographics Workshop on Computer Animation and Simulation (EGCAS)*. Springer, 1999. 5.3.3, 5.4.2, 5.5.1, 5.6

[Web08] Jason P. Weber. Fast simulation of realistic trees. *IEEE Computer Graphics and Applications*, 28(3):67–75, 2008. 5.2, 5.3.1, 5.4.3, 5.5.2, 5.6

[WH91] Jakub Wejchert and David Haumann. Animation aerodynamics. In *SIGGRAPH '91: Proceedings of the 18th annual conference on Computer graphics and interactive tehniques*, 1991. 5.2.2, 5.3, 7

[WWG06] Guo Wu, Li WenHui, and Feng Guanghui. Physically based animation of broadleaf plant. *International Journal of Computer Science and Network Security (IJCSNS 2006)*, 06:198–204, 2006. 5.3.1, 5.4.3, 5.5.2, 5.6

[Zha99] Zhengyou Zhang. Flexible camera calibration by viewing a plane from unknown orientations. *Proceedings of the Seventh International Conference on Computer Vision (ICCV'99)*, 01:666–673, 1999. 2.3.2

[Zha06] Yu-Jin Zhang. *Advances in Image and Video Segmentation*. Idea Group Inc (IGI), 2006. 4.2.1

[ZHH96] Robert C. Zeleznik, Kenneth P. Herndon, and John F. Hughes. Sketch: An interface for sketching 3d scenes. pages 163–170, 1996. 3.4.2

[Zio07] Renaldas Zioma. *Gpu gems 3*, chapter Gpu-generated procedural wind animations for trees (chapter 6). Addison-Wesley Professional, 2007. 5.2.2, 5.3.1, 5.3.2, 5.4.2, 5.6, 6.4.2

[ZST+06] Long Zhang, Chengfang Song, Qifeng Tan, Wei Chen, and Qunsheng Peng. Quasi-physical simulation of large-scale dynamic forest scenes. In *Computer Graphics International*, pages 735–742, 2006. 5.2.3, 5.2.3, 5.3.1, 5.4.2, 5.5.1, 5.6