Animating Non-rigid Bodies using Motion Capture

Jie Long

A dissertation submitted to the faculty of
Brigham Young University
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

Michael Jones, Chair
Parris Egbert
Bryan Morse
Eric Mercer
Sean Warnick

Department of Computer Science

Brigham Young University

January 2013

ABSTRACT

Animating Non-rigid Bodies using Motion Capture

Jie Long
Department of Computer Science, BYU
Doctor of Philosophy

Simulating the motion of a non-rigid body under external forces is a difficult problem because of the complexity and flexibility of the non-rigid geometry and its associated dynamics. Physically based animation of objects moving in the wind is computationally expensive, so simulation-based approaches oversimplify the model by ignoring important effects, such as tree's sheltering. Motion capture records actual responses of a non-rigid body to external forces and helps solve these problems. Mainly focusing on natural trees and ropes as instances of non-rigid bodies, we present a new approach to building motion for objects in wind using incomplete motion capture data from non-rigid bodies. The incomplete motion capture data are automatically labeled by a cluster-based algorithm while noises are removed. For places with no motion capture data, we estimate forces and motion by interpolating the motion capture data according to the object's characteristics. We discuss a physically or statistically based approach to animate the whole non-rigid object. Basing our work on the collected motion capture data and the estimated motions, we can produce visually plausible and scalable animations of non-rigid objects under external forces at interactive frame rates.

Keywords: Motion capture, non-rigid bodies, plant modeling, wind dynamics, particle flow

ACKNOWLEDGMENTS

Growing up I dreamed of being a scientist, but I did not even know what a real scientist was . Five years ago, BYU opened its arms to me to help me pursue a career as a scientist. My life has been changed dramatically in ways I could not even imagine years ago . I have so much gratitude to BYU and the Computer Science Department for having me here, as well as gratitude for my family, friends, and colleagues for all their support. I am truly and deeply indebted to so many people that there is no way to acknowledge them all, or even any of them, properly, with the limit of words here.

I would like to express my deepest gratitude to my advisor, Dr. Michael Jones. It was my great pleasure to work with him in the Computer Generated Natural Phenomena Lab . I am extremely grateful for his help, support, and encouragement over the last five years. I was impressed by his honest and humble personality as well as his sense of humor. I really appreciate his patience in guiding me in my research area. He has all of my respect as a lifelong best friend and mentor.

Besides my advisor, I am grateful to my thesis committee: Dr. Parris Egbert, Dr. Bryan Morse, Dr. Eric Mercer, and Dr. Sean Warnick. Thank you for your time and efforts in reviewing all my documents and materials. I appreciate all the encouragement, insightful comments, and hard questions. With all these, I am growing up and becoming a real researcher.

Thanks to my colleagues for stimulating discussions and a fun environment. I am especially grateful to Anthony Selino, Seth Holladay, Lanny Lin, Cory Reimschussel, Anthony Hall, Bryce Porter, Tonglaga Bao, and Ontario Britton at BYU. We are good friends learning and growing together.

I would like to thank my parents, Zeyu and Minglu, for their unconditional support. If not for them, there would be no me . Thank you for teaching me to be an honest person in all the good and bad times. Thank you for always supporting me and standing on my side. I

would also like to especially thank my grandmas. Both of them have left our world but I know they and their love will be with me no matter whom I am and where I am, forever.

Most of all I would like to thank my husband, Steve, who always gave me love and encouragement. Thank you for your protection and supporting a lovely place for me to stay and be able to concentrate on my study and research.

BYU accepted me and taught me how to be a good and useful person in our society. I have learned to do scientific research, which is one of the most important parts for why I am here. The world is our campus. I will remember the school's instructions and use them as my guidance in my future career and life. My learning will not stop upon receiving my degree, but will move forward to the bigger world campus. I believe that, all the things I've learned from here, in terms of being a good person in both my professional career and personal life, will benefit me for now and the future. I am so proud that I could have the chance to be stamped by BYU.

Table of Contents

List of Figures

Chapter 1

Background, Motivation, and Overview

Simulating non-rigid bodies with data from motion capture is a new application of motion capture. We present a motion reconstruction approach for non-rigid bodies in the context of motion capture. Passive optical motion capture records movements of non-rigid bodies by tracking locations of retroreflective sensors placed on subjects. By processing the recorded motion data and combining the data with physical and statistical models, we create complete animation of non-rigid bodies. We focus on animating natural trees and ropes as instances of non-rigid bodies. Motion is captured in the presence of external forces, such as wind and human interaction.

Rope is one of the simplest non-rigid bodies. Animating ropes is a simple platform from which to evaluate the motion capture process, including data collection and processing. The capture process then can be generalized and applied to other non-rigid subjects, such as natural trees. We present methods that can be used to create natural tree motion in wind using physical and statistical models. The resulting motion is visually plausible and scalable.

Animating non-rigid bodies has important applications in computer-generated movies, video games, and forestry engineering. In movie and video games where computational resources are a concern, directors need a simple process to use the motion of non-rigid bodies as part of telling a story. Motion capture hardware has been popular in movie and game production for many years. Motion capture equipment is easy to set up and provides motion data with good accuracy in both time and space. The hardware integrates camera calibration algorithms and greatly simplifies the process of recording position information in real-world

3D space. Many movies use motion capture to record motion from real characters and then drive virtual characters with the collected motion data [12]. Motion capture produces realistic motion data while reducing the cost for creating motion in different environments or in fictional environments.

The equipment has flexibility for capturing both rigid bodies and non-rigid bodies. However, most research and applications focus on rigid bodies. As a non-rigid body, a tree has motion that is important in film and games because trees swaying in the wind can set the mood or feel for a scene, so directors can use tree motion as a tool for telling stories. In the field of forestry, our research helps forestry engineers understand the influence of wind on trees. This can inform decisions for pruning trees or protecting structures. Rope, the other application explored in this work, has simple structure and dynamics. But when the motion of a human character interacting with the rope is captured in the same scene, the replay of both rope and human motion becomes difficult. Movie makers may be able to use our motion capture process and create animation with interactions between rigid and non-rigid bodies.

Human motion capture using a rigid body model, such as [25, 44, 62, 68], is a well-studied area in motion capture but is less applicable to our research, as removing the rigid body assumption changes the problem. There are two common approaches to extracting human motion from motion capture data. One is to extract a skeleton model or kinematic model from motion data [25, 44]. The other is to predefine the skeleton and to apply motion capture-data to animate the predefined skeleton [62, 68]. Most of the methods have an important assumption that the distance between any pair of markers is non-changeable. However, this assumption does not stand for non-rigid bodies, which is the essential difference for motion capture of rigid bodies and non-rigid bodies. Instead of a skeleton of rigid parts, non-rigid motion capture focuses on reconstructing a mesh that deforms to match a moving surface such as a face or cloth. We investigate non-rigid motion capture but we focus on curved splines rather than curved surfaces.

Motion capture of non-rigid bodies is difficult. Prior work on non-rigid motion capture is mainly focused on facial motion [24, 54] and cloth motion [4, 27, 40, 63]. Both facial motion and cloth motion aim to represent the motion of a surface. Motion capture of thin, rod-shaped non-rigid bodies such as rope or tree branches is a fundamentally different problem. For instance, rope, as a non-rigid body, is more naturally represented as a curved spline rather than as a curved plane (though a curved spline can be used to drive the motion of a plane). Worring's [64] prior work in reconstruction of a line-shaped object in 3D from several computer images builds non-rigid rope motion but would require a foreground– background separation step and a sharp image of the rope in each frame.

Part of animating trees in wind is modeling the 3D shape of the tree to be animated. Tree shape modeling has long been studied on computer graphics. A tree with natural appearance greatly improves visual quality of tree animation. These methods include particle systems [29, 34, 46, 50], L-systems [20, 21, 43], parametric models [60], photographs [29, 45, 58] or videos [8, 19]. Most of these methods result in satisfying tree shapes but do not leverage the 3D positions of motion capture markers, which are recorded as part of a motion capture session but require a different set of inputs. Image- or video-based approaches convert a set of 2D input images into 3D tree models by filling in the missing dimension. Motion capture systems can record tree shape in 3D with high precision (using similar techniques for converting a set of 2D images to a 3D model). Prior work [23] in reconstructing tree shape from motion capture data using either exact measurements or markers placed within the crown does not scale and does not apply when leaves occlude the branching structure.

Animation of trees in wind has been discussed for many years [1, 52, 56]. Prior work in animating 3D tree models focuses on recreating branch motion due to wind turbulence. Wind turbulence has been simulated and has been synthesized from the frequency spectrum of turbulence created by tree crowns. Simulation-based models create tree motion based on the tree's biomechanical characteristics and wind dynamics [1, 11]. Spectral approximation describes tree swaying and wind velocity field using some computer-generated noise. These

systems include techniques based on photographs [65] or videos [8], and some parameter-based spectral models [11, 52]. In most of the previous research, the wind field is created using noise and fluid simulation. While simulation models capture visually important wind–tree effects, such as crown sheltering, they require expensive computations that are not currently feasible in interactive applications such as games. Spectral approximation ignores sheltering effects and requires significant user intervention, but is computationally efficient. Rather than being based on actual captured tree motion, simulation models and spectral approximations are both theoretically based. Motion capture avoids the directability and computation problems of simulated wind fields but may yield data that validates simulation-based models.

Our work differs from the previous work by introducing motion capture in simulating non-rigid bodies. It is also the first work to motion capture bendy thin rods and the first to replay motion capture data from a tree. The resulted animations are mostly driven by data instead of pure physical simulation or stochastic process. We describe a complete approach to animating natural trees using motion capture, including data collection, data processing, generating tree shape, and combining statistics and wind dynamics for the animation. The thesis discusses two approaches for tree animation with different motion capture setups, which require different processes and create tree motion with varying scalability. Rope motion is created using a data-driven approach rather than the pure physical model that is common in previous research. The rope project helps us to better understand the motion-recording capability of motion capture for non-rigid bodies. Using motion capture simplifies the process to parameterize a physical model of a rope under specific motion.

1.1 Thesis Statement

A small portion of known movements collected from motion capture can produce complete movements of non-rigid bodies by combining the motion capture data with physical and statistical models. We show that the resulting motion is visually plausible for trees and rope.

1.2 Publications

This dissertation consists of seven papers, two of which are under review. The work can be divided into three parts: rope motion reconstruction, tree shape reconstruction, and tree motion reconstruction. The remaining chapters contain these papers. Chapter 2 is a pilot project in which a simplified motion capture process is presented with statistical analysis for replaying natural tree motion where the motion is not scalable. There is no attempt to retarget or extend the captured data. While we were able to replay the tree motion, it requires a series of careful measurements to recreate the exact tree shape, and the motion cannot be transferred to similar shapes or situations. Chapter 3 describes replaying rope motion. The project focuses on processing motion capture data and uses rope's simple structure to validate estimated data against the collected motion data. The process for motion capture data includes identifying and removing noises and labeling markers. This process was used for both tree and rope motion capture. Chapters 4 and 5 begin a different approach to the problem. Rather than directly replay the original motion on the original structure, in Chapter 4 and 5 we reconstruct the approximate tree shape from the captured motion in preparation for replaying similar motion on a similar structure. Chapter 6 extracts a force field from captured motion that can be used to animate any tree shape.

In Chapter 2, the pilot project aims to replay natural tree swaying in wind. Retroreflective markers are placed on a small cherry tree. We infer a skeleton from tree motion data and repair the motion data using a rigid body model. The motion data contain gaps and errors for branches that bend. Motion-data repair is critical because trees are not real rigid bodies. These ideas allow the reconstruction of tree motion, including global effects, but without a complex physical model. Instead, it employs a statistical model. This project builds a complete pipeline for motion capture, including equipment set up, data collecting interface, 3D tree model, and motion integration. However, the scalability of this approach is limited and it requires intensive labor to reproduce the exact tree structure. The work requires markers being placed about 10 cm apart on each branch segment. Leaves have to

be removed to enlarge the visibility of markers to motion capture cameras. Branches are assumed to be rigid (to some degree) with small bending capability.

Chapter 3 begins a study of motion capture for rope and takes a closer look at animating non-rigid bodies using a non-rigid model for motion capture. Rope has a simple structure with no branches and simple geometry. The model is a good test bed for evaluating different algorithms for both data processing and motion replaying. In this project, we provide a more general approach for processing motion capture data from passive optical motion capture for non-rigid bodies rather than adapting a rigid body model as in Chapter 2. The data collection and processing process includes labeling markers and identifying and removing noise. We provide clustering, gap repair, and marker swap detection algorithms based on linear interpolation and forward differencing under the assumption that the rope does not stretch. Indexed marker positions are connected with a spline in each frame to approximate the original rope. The model produces visually plausible animations of rope motion from data collected for a person interacting with rope. However, the method fails when the rope experiences large accelerations that result in motion that is not modeled by forward differencing.

Reconstruction of tree shape, as in Chapters 4 and 5, is an important step in replaying motion capture of trees under external forces, which is the goal of the work presented in Chapter 6. A realistic 3D tree model helps researchers estimate and compare computer-generated animation against the original motion. Existing algorithms for generating branching structures for image synthesis in computer graphics are not adapted to the unique data set provided by motion capture. In Chapter 4, we discuss a method for tree shape reconstruction using particle flow on input data obtained from a passive optical motion capture system. Initial branch tip positions are estimated from averaged and smoothed motion capture data. Branch tips, as particles, are also generated within the bounding space defined by a stack of bounding boxes or a convex hull. The particle flow, starting at branch tips within the bounding volume under forces, creates tree branches. The resulting shapes are realistic and

similar to the original tree crown shape. Several tunable parameters provide control flexibility over branching shape and arrangement.

In Chapter 5, we combine a procedural method with particle flow to generate a 3D tree shape. Using L-systems for describing tree branches as particles, our method introduces a hemisphere to generate particles, uses a growth level to simulate different ages of branches, and applies a dynamic bounding box to detect local growth area in a tree. This new method enhances the management of tree shapes by easing the control over distributions of branches and leaves. We also demonstrate that the method has potential to simulate phototropism and growth around physical barriers. We evaluate this model by particle flow and the complexity of this method, showing performance competitive with existing methods.

Chapter 6 discusses non-rigid motion capture by extracting external forces from motion capture data and then replaying those forces to create animation. We explore this idea in the context of motion capture of natural trees in wind. Motion of a tree in wind is decomposed into three forces: wind-induced drag, branch elasticity, and damping by the leaves. Given a model of elasticity and damping, the drag force can be isolated and used to estimate wind velocity. The extracted velocity field is extended to a larger volume and enriched with a turbulence model. That wind field can be replayed on a tree model that includes elastic and damping properties to create similar motion. The work contained in this chapter is the culminating work of this dissertation.

We list all the citations for each chapter in the order in which they appear.

1. Jie Long, Cory Reimschussel, Ontario Britton, Anthony Hall, and Michael Jones. Performance Capture for Natural Tree Motions in the Wind. Motion in Games , MIG, 2010.

2. Jie Long, Bryce Porter, Michael Jones. Motion Capture of Rope, not yet published.

3. Jie Long and Michael Jones. 3D Tree Modeling using Motion Capture. IEEE The Fourth International Symposium on Plant Growth Modeling, Simulation, Visualization and Application (PMA '12).

An extended version of this paper is requested for submission to the journal Frontiers in Computer Science .

4. Jie Long and Michael Jones. A Realistic 3D Tree Model based on L-Systems. Report for UNEP Eco-Peace Leadership Center (EPLC), 2008.

5. Jie Long and Michael Jones. Estimating wind flow from tree motion using motion capture data, not yet published.

In addition, the research on motion capture of trees was presented as a short talk in SIGGRAPH 2009, where a short abstract was published.

Jie Long, Cory Reimschussel, Ontario Britton, and Michael Jones. Motion capture for natural tree animation. International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2009: Talks. New Orleans, Louisiana, Article No. 77, 2009.

Data captured as part of this work was used in a study of tortuosity as a metric for evaluating branch motion paths. The citation for that paper is given below but the paper is not included in the dissertation.

Michael Jones and Jie Long. Tortuosity as a Metric for Evaluating Branch Motion Paths. IEEE The Fourth International Symposium on Plant Growth Modeling, Simulation, Visualization and Application (PMA '12).

Chapter 2

Motion Capture for a Natural Tree in the Wind

Jie Long, Cory Reimschussel, Ontario Britton, and Michael Jones. Motion capture for natural tree animation. *International Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 2009: Talks.* New Orleans, Louisiana, Article No. 77, 2009.

Abstract

Simulating the motion of a tree in the wind is a difficult problem because of the complexity of the tree's geometry and its associated wind dynamics. Physically based animation of trees in the wind is computationally expensive, while noise-based approaches ignore important global effects, such as sheltering. Motion capture may help solve these problems. In this paper, we present new approaches to inferring a skeleton from tree motion data and repairing motion data using a rigid body model. While the rigid body model can be used to extract data, the data contains many gaps and errors for branches that bend. Motion data repair is critical because trees are not rigid bodies. These ideas allow the reconstruction of tree motion, including global effects, but without a complex physical model.

2.1 Introduction

We address the problem of animating natural trees in games with greater accuracy but without additional computational overhead compared to techniques based on velocity or force textures—such as [11]. We believe that motion capture is one way to accomplish this goal. Motion capture of tree motion in the wind is difficult because the tree branching structure is both important and difficult to model and because branches are non-rigid bodies at large deflections.

Accurate animation of trees is important to both CG animators and forestry ecologists. CG animators can use plausible and directable models of tree motion in digital storytelling. In a game, trees moving in the wind can be used to emphasize weather or create a sense of foreboding. Forestry ecologists can use models of tree motion to design pruning methodologies that maximize yield while minimizing windthrow potential.

Many approaches have been taken to modeling tree structure and geometry. Recent photo-based approaches to tree modeling [29, 45, 58] are particularly relevant to this work. Photo-based methods plausibly recreate 3D natural tree models by approximating the branching structure of photographed trees. However, these models are created without considering tree motion. This means that the branching structure may not match the motion of the tree.

Prior work in animating 3D tree models focuses on recreating branch motion due to wind turbulence. Wind turbulence has been simulated and has been synthesized from the frequency spectrum of turbulence created by tree crowns. Simulation-based models create tree motion based on the tree's biomechanical characteristics and wind dynamics [1, 11]. Spectral approximation describes tree swaying and wind velocity field using some computer-generated noise. These systems include techniques based on photographs [65] or videos [8] and some parameter-based spectral models [11, 52].

Each approach is insufficient. While simulation models capture visually important wind–tree effects, such as crown sheltering, they require expensive computations that are not

currently feasible in interactive applications such as games. Spectral approximation ignores sheltering effects and requires significant user intervention but is computationally efficient. Rather than being based on actual captured tree motion, simulation models and spectral approximations are both theoretically based.

In this paper, we present novel approaches to tree structure estimation from motion capture data and tree motion repair using interpolation. We approximate the tree structure using a minimal spanning tree over position and movement data collected during motion capture. We detect and repair the collected data using interpolation techniques based on curve fitting and machine learning. The resulting tree model and animations are realistic recreations of a tree moving in the wind and include sheltering effects while supporting fast playback. We avoid modeling wind fields explicitly because their end effect is measured directly in the motion of the leaves and branches. Since this represents only the initial stages of applying motion capture to the problem of tree animation, we focus simply on the motion of one specific tree subject. We leave for future work questions such as how the results might scale to other trees or subject models. The animations resulting from this work can be seen in the video that accompanies this paper.

2.2 Related Work

In this section we discuss closely related work in tree modeling, tree animation, and motion capture.

2.2.1 Static Tree Modeling

Static tree models describe tree shapes including topology, texture, and geometry for the trunk, branches, and leaves. Tree models for motion capture data need to capture the branching structure of a specific tree such that the captured motion looks plausible when animating the model. This is a unique challenge in tree modeling that has not been addressed by prior work.

Position-aware L-systems [43] have been used with some success to create models of specific plants, but these results are difficult to reproduce. The processes of controlling the branching structure using the silhouette and setting the rule parameters is difficult.

Photo-based approaches [29, 45, 58] can produce plausible tree shapes that match a given tree but estimate the internal branching structure using methods such as particle flow [29]. Estimates of the internal branching structure are not sensitive to the motion of the original tree. We use similar methods based on photographs to create a bounding volume for the tree shape. In addition to images, we also use motion capture data to recreate a plausible internal branching structure in which points contained in one branch have similar movement.

Diener et al. approximate shrub structure based on single-camera video data of a shrub in the wind [8]. Diener uses a clustering method to identify clumps of the shrub with similar motion and then builds a skeleton that corresponds to the clustering. Our approach is similar, but we skip the clustering step and build a skeleton directly from the marker positions and motion data in 3D rather than 2D video data.

2.2.2 Animation of Trees

Prior work in tree animations relies primarily on simulation-based methods and spectral approximations. Both approaches produce plausible tree movements in the wind while ignoring some effects to remain tractable. Most of these methods simplify the complex dynamics of leaf–wind interaction, which is the primary cause of branch motion. One study [49] found that much of the motion of a branch could be accounted for by the presence or absence of leaves. Motion capture obviates the dynamic model but introduces several additional problems.

Simulation-based methods use computational fluid dynamics to simulate the effect of wind on trees. Akagi and Kitajima [1] do allow trees to influence the wind using a two-way coupled model based on the Navier-Stokes equations, with an additional term for external forces. The simulation is based on a stable approach [56] to the marker and cell method.

Akagi and Kitajima use virtual resistive bodies to account for tree structures smaller than the grid resolution and add adaptive resolution and a boundary conditions map to improve performance by allocating grid resolution only where needed. Simulation-based methods are currently too computationally expensive for use in games.

Spectral approximations of trees in wind use approximations to the recorded spectra of wind passing through trees to generate motion. This method was first used by Shinya and Fournier [52] and later by Chuang [6], Habel [11], and Zhang [67]. Other work also relies on approximations in the frequency domain but uses different techniques to approximate turbulence [30, 32, 56]. Spectral methods have also been combined with physical simulation [33, 67]. Spectral approximations result in plausible motion and are efficient enough for games but ignore the bidirectional wind–tree interactions, such as sheltering effects. These effects are important for visual realism and are captured using motion capture. Our objective is to create animation data which can be used as efficiently as textures but which are more accurate.

More recent work [9, 11] animates tree motion in a computationally economical way. Diener [9] simplifies the wind model using a pre-computed wind projection basis taken from vibration modes rather than a harmonic oscillator model. As with Habel [11], the wind is assumed to be spatially uniform for a single tree. At run time, the wind load is estimated for all nodes on a tree relative to the wind projection basis, and this can be combined with a level-of-detail model to render a forest of thousands of trees in real time. Each of these methods ignores the effect in return of trees on the wind and therefore omits all forms of sheltering. Another less significant problem is that the turbulence used in these models matches actual turbulence only in the frequency domain and not necessarily in the time domain. While many turbulence patterns share frequency spectrums with those created by tree crowns, only one pattern matches the spatial properties of the actual turbulence created by a specific tree in a specific wind. Our work captures the motion of a tree as it moves in the turbulence created by that tree.

Our work is similar to video-based approaches in that we capture and analyze tree motion. Unlike video- [8] or image-based [65] approaches, we obtain a motion path for a cloud of points in 3D rather than applying 2D motion to 3D skeletons. Our methods may also yield new insights into how to use video data in the animation of trees.

2.2.3 Motion Capture

Motion capture for trees is more difficult than performance capture of human subjects because trees are both rigid and non-rigid (depending on the applied force, among other factors) and have more complex and less predictable topologies.

Motion capture systems have been widely used for human or animal performance capture [48, 68]. We use a method similar to Kirk [16] to automatically generate rigid skeletons from optical motion capture data. Since tree branches are both rigid and non-rigid, the data do not contain a constant distance between markers. We use a rigid body algorithm to solve the marker indexing problem. Because some of the data is collected from non-rigid motion, this introduces additional noise and gaps in the data. A central contribution of this paper is a way to repair this data for tree motion. Another approach to this problem would be to investigate marker indexing algorithms for non-rigid bodies. Doing so may reduce the amount of noise and gaps in the motion data.

2.3 Motion Capture

We use an optical motion capture system to collect position and motion data from which we reconstruct tree structure and movement. For this paper, the system consists of 12 OptiTrack FLEX:V100 cameras arranged in a circle in a 4 m X 4 m room indoors. A cherry tree sapling with a height of 2 m was used as the test subject. The tree was placed in the center of these cameras and a fan was used to create wind at different speeds near the tree. The system has not been deployed for trees larger than 3 m and we believe it would be impractical for large

Figure 2.1: Motion capture setup for a natural tree using an optical motion capture system.

trees. We believe it would be more practical to explore methods for extrapolating small tree motion to create large tree motion than it would be to capture large tree motion.

Reflective markers are placed on each branch and some leaves. The arrangement of markers on a single branch segment depends on the flexibility of the branch. If the branch is thin and flexible, the distance between markers is about 8 cm; for a rigid branch, such as the trunk, the distance between markers is about 15 cm. Placing markers more closely together allows us to approximate a flexible branch as a series of rigid linkages. This results in cleaner motion data with fewer gaps because the motion capture system depends upon a user-defined set of fixed-length rigid links in order to track and label the markers as they move. The benefits of this approach are especially evident under higher wind speeds, when branches begin to flex and bow. This placement strategy assumes that the tree crown is sparse. Trees with dense crowns will require a different strategy.

We placed approximately 100 markers on the tree to collect branch motion. The 3D positions of all reflective markers are recorded at 100 frames/sec. Leaf motion is recorded separately using three markers on each leaf in a smaller representative sample. Figure 2.1 shows the arrangement of markers for both branches and leaves motion capture.

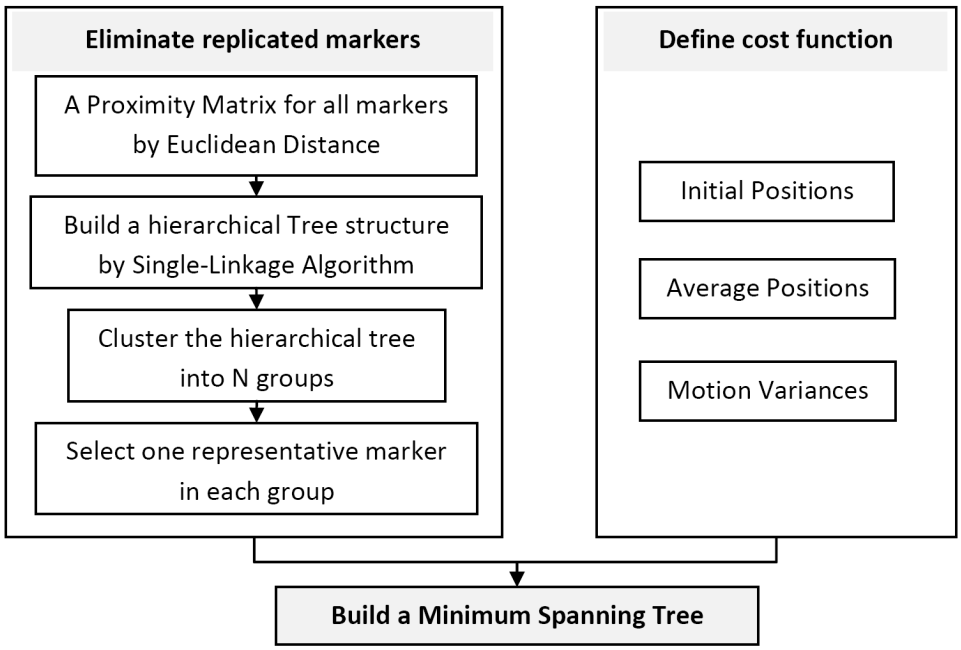


Figure 2.2: Steps in building a tree skeleton.

2.4 Build Static Tree Geometries

One significant problem with motion capture of trees, compared to human performance capture, is that the branching structure, or topology, of a tree is less predictable and more complex than that of a human. A minimal spanning tree algorithm is used with a cost function derived from motion data to create a plausible branching structure. The branching structure is plausible when animating it with the captured motion looks plausible. The cost function is one of the contributions of this paper.

2.4.1 Skeleton and Topology Estimation

Figure 2.2 summarizes the process of estimating the skeleton and topology. This process has three steps. First, hierarchical clustering eliminates replicated recorded markers in each frame. Next, we use position and motion data for each marker to define a cost function. The cost function is used to estimate the plausibility of merging two different markers. A minimum spanning tree algorithm uses a different cost function to connect the markers into a

tree-like skeleton that will have plausible motion when animated using the captured motion data.

The first step, shown on the left side of Figure 2.2, is to eliminate duplicate, yet slightly different, recorded positions of a single marker. We use Euclidian distance as the clustering metric. The single linkage algorithm groups markers into a hierarchy of n clusters, where n is the number of markers originally placed on the tree. Within each frame, a cluster is reduced to a single representative marker. When all frames have the correct number of markers, we further refine the representative position of each marker, either by choosing its position in the tree's rest pose or by averaging its position over time. The tree skeleton is built from the n representative marker positions. This skeleton is used for the entire capture sequence.

The second step, shown on the right side of Figure 2.2, computes costs for creating connections in the control skeleton between different pairs of markers based on the recorded position and motion data. Connection costs are computed for pairs of representative markers with one marker from each cluster. The cost function consists of three elements: initial position, average position over time, and variance of position over time. We assume that the branch motion is periodic. The average position is similar to the position while the variance reflects the amplitude of the movement. The initial positions are recorded when there is no wind and the tree is stationary. The average positions are calculated as shown in the next equation in which m is the number of recorded frames and p_i is the 3D position for a marker at the ith frame:

$$\bar{d} = \frac{1}{m}\Sigma_{i=1}^{m} p_i.$$

The variance in position is similarly defined as

$$\sigma^2 = \frac{1}{m}\Sigma_{i=1}^{m} (p_i - \bar{d})^2.$$

Let α, β, and γ be constant weighting parameters; then the cost to connect markers M_a and M_b is given by

$$\omega = \alpha||p_{M_a} - p_{M_b}|| + \beta||\bar{d}_{M_a} - \bar{d}_{M_b}|| + \gamma||\sigma_{M_a} - \sigma_{M_b}||.$$

The cost to connect markers M_a and M_b is low when M_a and M_b are close in both position and movement.

In the third step, we use Prim's MST algorithm with the node at the bottom of the trunk as a starting point to build the tree skeleton. Pairs of markers with similar position information and movements have low connection cost and are connected in the skeleton. This skeleton is directly taken as the input tree structure for rendering in the next step.

Figure 2.3 shows the importance of each part of the cost function. The right side of Figure 2.3 shows a tree created using just the change in variance as the cost function. This cost function results in branch tips connected to branch tips because variance increases as one moves along a tree from trunk to branch tips. The middle tree was created using only positional information. While the structure is accurate for much of the tree, several points are connected incorrectly across the middle of the tree. This will result in implausible motion when animated using the motion capture data. Using both metrics, along with average position, results in a more accurate model shown on the right.

By combining these parameters, we connect markers with similar position and movement. For a tree with 98 clusters of markers, 66.26% of the resulting connections are correct when compared with the actual tree. Most errors are from connecting markers in the correct branch segments but at the wrong junction points within the branch segment. This cost function occasionally connects markers from different branches but which share close positions and movements. In these cases, the motion of physically adjacent branch tips is similar and the resulting animation is still plausible.

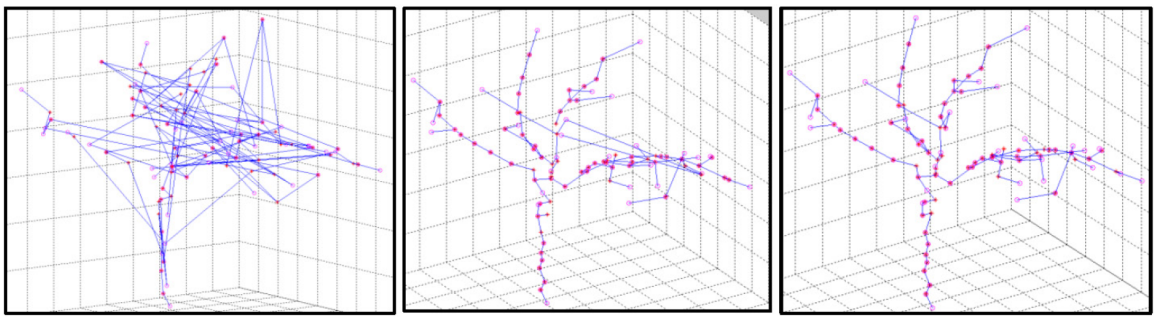


Figure 2.3: Reconstructed tree topologies using variance, initial position, and a combination of variance, initial position, and average position.

2.4.2 Geometry of Branches and Leaves

After the tree skeleton is created, the next step is to generate the geometric mesh. The marker points in a single branch are used as control points to create a curve. A second curve is placed at the first marker point in the branch and oriented to the first two points in the branch. A closed circular shape is swept along the profile curve to create a NURBS surface. The profile and shape curves are discarded, leaving just the branch geometry.

Then we bind the mesh to the skeleton. This step is separated from the previous steps so that the artist has more flexibility to modify the automatic mesh before it is bound to the skeleton. After any needed updates, the mesh is bound to the geometry. Once the geometry is bound, the artist again has the flexibility to manually tweak the binding.

Finally, a 3D point cloud inferred from photographs guides the manual placement of leaves. The leaves are placed to fill the volume occupied by the original tree. The tree volume is created using inverse volumetric rendering [45] applied to 37 photographs taken from a known camera position. The resulting 3D point cloud is exported to a 3D modeling package and, after manually matching the tree skeleton with this point cloud, we manually place leaves on branches while remaining in the recorded crown volume.

2.5 Build Tree Motion

In this section, we describe branch motion repair and leaf motion synthesis. Branch motion repair is the process of identifying and eliminating errors, gaps, and noise from the motion capture data. The resulting motion is used to animate the 3D tree model created in the prior section.

We used the rigid body algorithm that was shipped as part of the NaturalPoint Arena software to convert unindexed point clouds into an animated skeleton. Because tree branches are non-rigid at large deflections, the resulting motion contains more gaps and errors than one might expect to find for rigid body motion capture. We use linear interpolation, a filter, and a machine learning algorithm to repair the resulting motion. The NaturalPoint Arena software provides some interpolation processes to fix motion gaps, but requires the user to manually identify gap regions and select a correction method. We automate gap detection and correction with different methods, depending on the gap size. A machine learning based method for addressing large gaps is one of the contributions of this paper. We use a standard curve fitting technique for small gaps.

2.5.1 Filter-based Noise Detection and Removal

For some non-rigid motion, the rigid body motion capture system introduces anomalous artifacts to the motion signal, resulting in sporadic popping motions of certain leaves and twigs. These artifacts are detected using convolution-based filtering techniques and are replaced by fitting Bezier curves over the corresponding sections of the motion signal.

2.5.2 Small Gaps in Data

Small gaps in data are short sequences of 100 frames or less in which no position data is recorded for a marker. Small gaps occur when a marker becomes occluded or is otherwise lost. Linear interpolation is used to repair small gaps because linear interpolation can be done quickly and is good enough for these gaps.

Linear interpolation predicts missing marker positions based on the positions of neighbors. For a marker with missing position data, we find the two nearest neighbors with available position data. Then we compute Euclidean distances among the positions of these three markers and a velocity for each marker. Different distance metrics can be used. By doing linear interpolation according to the positions and velocities, we estimate the position for the missing marker.

Linear interpolation works well if all three markers have similar movement. However, if the motions of two different, but adjacent, missing markers have their positions interpolated from the same set of nearby neighbors, the resulting interpolated motion may not preserve each marker's unique periodic motion. This may happen even though we aim to make the interpolated motion fit smoothly with the existing motion for each marker. However, losing periodic movements for a short period of time when repairing a small gap still results in visually plausible motion.

2.5.3 Large Gaps in Data

Repairing large gaps in data is done using a more sophisticated interpolation scheme so that the synthesized motion has good periodic properties. Large gaps in data refer to gaps which comprise more than 40% of the entire motion trace collected for a single marker. A machine learning algorithm builds a function that is used to infer motion that is used to fill large gaps.

Given the connection between two adjacent markers, motion data for both markers at low wind speed, and motion data for one marker at high speed, the machine learning algorithm trains a support vector machine (SVM) and defines a correlation function. This approach is based on the observation that good data is captured for all markers at low wind speed, but large gaps appear in the data for some markers at high wind speed. The SVM learns a correlation between data from two markers collected at low wind speeds. This relationship is used to estimate missing motion at high wind speeds under the assumption that the relationship is not sensitive to wind velocity.

The tree skeleton structure is used to find the nearest topological neighbor with motion data for both high and low speeds. In most cases, markers at branch tips have missing data while markers at the branch base have the required data. This is because markers at the branch base move more rigidly than markers on tips. In these cases, the marker at a tip has large gaps in motion data and its nearest neighbor in the direction of the branch base often lies on the same branch.

Sequential minimal optimization (SMO) [14, 38] trains an SVM, which defines the correlation function between the two markers' positions at low speed. In order to improve the precision of the correlation function and to avoid phase differences, the motion data from each series is sorted in ascending order of displacement. Let M_a be the nearest neighbor to M_b, which is a marker with missing motion at high speed. M_a and M_b both have motion data at low speed. A learned function F estimates the position of M_b given M_a. Position data from M_a recorded at high wind speeds is given to F, which then estimates M_b's position at high wind speeds.

Figure 2.4 shows the estimated and actual position for one marker at low and high wind speeds. The vertical axis is the displacement and the horizontal axis is the frame number. In this figure, the motion of marker M_a at low wind speed, which is the topmost trace on the left, is used with the recorded motion of marker M_b, which is the lower trace on the left, to learn a function that predicts the position of M_b given the position of M_a. For comparison, we placed the predicted position of M_b on the graph as well. The predicted position of M_b closely matches the actual position at low wind speeds. At high wind speeds, shown on the right, we held back the recorded position of M_b and predicted the position of M_b in each frame given only the position of M_a. The predicted position of M_b at high speeds closely matches the actual position of M_b but tends to overestimate the amount of displacement in M_b.

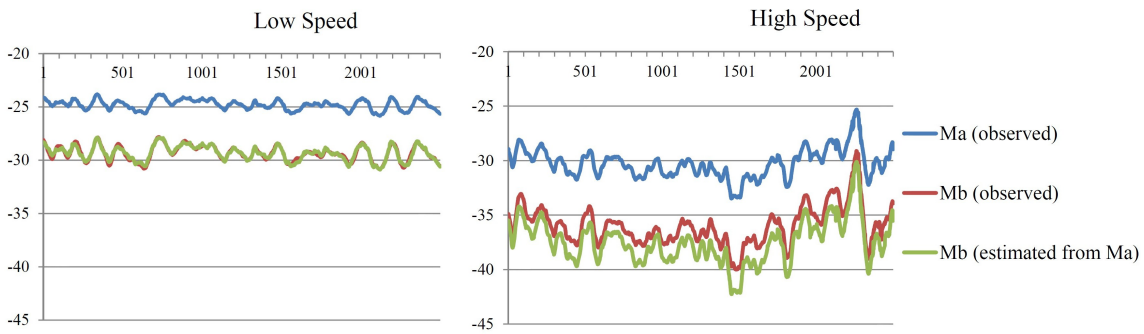


Figure 2.4: Predicted versus actual displacement for a marker at low and high wind speeds.

2.5.4 Leaf Motions

Motion data applied to leaves is based on motion captured from only a few leaves. This motion is scaled and offset to simulate a greater variety and randomness of leaf motion. The leaf geometry deforms along motion curves applied at the end, at the middle, and near the stem.

The complexity of leaves moving in the wind precludes any attempt to correlate leaf movement with the movement of the branch it is on. Leaves can be quite turbulent or almost still on a branch that is either very still or sweeping any position through its arcs of movement. However, motion of the leaves is scaled with the branch motion to suggest that they are driven by the same wind. These two motion sets can also be decoupled for an artist to achieve a particular effect.

2.6 Results

The final animation is shown in the video that accompanies this paper. In that video, most motion capture artifacts have been removed and the motion looks reasonable. Results in skeleton estimation and motion repair, which are the main contributions of this paper, were given in the preceding sections.

2.7 Conclusion and Future Work

A plausible tree skeleton can be reconstructed using a minimal spanning tree algorithm over a cost function defined using position and motion data. The skeleton is plausible in the sense that replaying the capture motion on the skeleton looks realistic. Gaps and errors in motion capture data for trees can be replaced with data interpolated from neighboring branch motion. These are important steps toward realizing motion capture of trees for tree animation in games. Motion capture of tree motion is a good match for motion in games because the resulting motion is realistic but requires only replaying, rather than simulating, actual motion.

We had hoped to get better results with the repaired data and the rigid body algorithm we used. Based on these results, we believe that investigating other approaches to processing the point cloud are more promising than repairing the errors caused by using the rigid body algorithm we used.

Future work could take several interesting directions. One of these is to avoid defining rigid bodies for each branch while capturing motion by defining the tree as a non-rigid body, which is a truer representation of its natural form. More work needs to be done to be sure the algorithm scales well for capturing the motion of other tree subjects as well as for transferring captured motion from one tree to another. By capturing data from multiple trees at once, the interactions among them could be studied and applied to simulate groups of trees or even forests.

Chapter 3

Motion Capture of Rope

Jie Long, Bryce Porter, Michael Jones. Motion Capture of Rope.

Abstract

We reconstruct rope motion from passive optical motion capture data using a statistical model without a dynamic model of rope behavior. Progress in motion capture for faces and cloth has limited applicability to the motion capture of rope because rope is a curved spline rather than a curved surface. We present clustering, gap repair, and marker swap detection algorithms based on linear interpolation and forward differencing under the assumption that the rope does not stretch. Indexed marker positions are connected with a spline in each frame to approximate the original rope. The model produces visually plausible animations of rope motion from data collected for a person interacting with rope. The method fails when the rope experiences large accelerations that result in motion that is not modeled by forward differencing.

3.1 Introduction

Motion capture for rope is a challenging problem because rope is a non-rigid body. The problem is more difficult when the motion of a human character interacting with the rope is also captured in the same scene because, before markers are indexed from frame to frame, it is difficult to separate markers on the actor from markers on the rope. We present a statistical motion estimation scheme with a clustering model to achieve motion capture of rope from passive optical motion capture data. Our solution consists of novel approaches to clustering and the repair of gaps and swaps, as well as segmenting rope motion from human character motion.

Motion capture of human characters interacting with rope may have applications in the production of computer-generated movies and video games. Using motion capture data simplifies recording the interaction between characters and ropes while preserving the subtleties of that interaction. Capturing interactions between rope and characters may be particularly compelling when motion capture is already being used to record character motion. In this paper, we focus on separating rope motion from actor motion and then reconstructing the motion of the rope. A variety of existing methods can be used to reconstruct actor motion from motion capture data. The approach allows reconstructing the interactions between characters and thin, non-rigid bodies.

Prior work on non-rigid motion capture is mainly focused on facial motion ([24, 54]) and cloth motion ([4, 27, 40, 63]). Both facial motion and cloth motion aim to build a mesh system for representing the motion of a surface. Motion capture of rope is a fundamentally different problem because rope is more naturally represented as a curved spline rather than as a curved plane (though a curved spline can be used to drive the motion of a plane).

Human motion capture using a rigid body model (such as [25, 44, 62, 68]) is a well-studied area in motion capture but is less applicable to our research as removing the rigid body assumption changes the problem. Worring's [64] prior work in reconstruction of a line-shaped object in 3D from several computer images solves essentially the same problem

but would require a foreground–background separation step and a sharp image of the rope in each frame.

Our solution reconstructs free motion of rope and ribbon from recorded motion capture data. The captured data consists of an unindexed point cloud of purported marker positions. A marker segmentation algorithm is used to first separate the rope markers from the character so that the two motion streams can be reconstructed individually. A specialized clustering method processes the captured rope motion data and creates motion traces for markers in 3D space. Each cluster represents the motion from a single reflective marker. The clustering process also automatically detects and eliminates most noise. After creating these motion traces, marker swaps are detected and repaired by assuming that the rope does not stretch. Motion capture data marked as noise are revisited and used to fill gaps in marker traces if the data fits existing motion traces without extending the rope length. Small gaps are filled using linear interpolation and big gaps are filled using a structure-based midpoint displacement algorithm. Human character motion can be reconstructed using any of a variety of algorithms for reconstructing human motion from optical motion capture data. The resulting motion streams containing rope and human motion, respectively, are recombined to recreate the original motion in 3D.

Our contributions are the following:

(1) A marker segmentation algorithm that separates rope markers from the character markers.

(2) A clustering algorithm to aggregate a point cloud into motion traces for non-rigid ropes and ribbons.

(3) Forward differencing to estimate marker locations using previously labeled locations.

(4) A recursive midpoint displacement algorithm for fixing gaps.

These contributions allow us to recreate the interactions of a character with a rope or ribbon from passive optical motion capture data. Side-by-side playback of the resulting 3D

motion and video of the original motion show that the captured motion closely matches the original motion.

3.2 Related Work

The process of reconstructing motion from motion capture data includes data collection, data processing, and model mapping. Commercially available motion capture packages provide tools for creating the motion of a few objects, including human bodies and faces. Current research mainly aims to optimize the usage of collected motion data and to improve the quality of the resulting motion. However, fundamental issues in the motion capture of non-rigid bodies require additional investigation.

There are two common approaches to extracting human motion from motion capture data. One is to extract a skeleton model or kinematic model from motion data [25, 44]. The other is to predefine the skeleton and to apply motion capture data to animate the predefined skeleton [62, 68]. Wen [62] uses a least-square optimizing method to match motion capture data to a human skeleton. Zordan [68] maps motion capture data to a predefined skeleton using a force-based physical model. In our research, the structure of the rope model is a simple non-branching curve, so we predefine the rope shape and apply our algorithm to automatically map motion data onto the predefined structure.

Liu and colleagues [22] take a data-driven approach to completing human motion from a limited set of markers. A similar approach to rope motion reconstruction would require recording and analyzing many different rope motions, which could then be fit to partial data.

Instead of a skeleton of rigid parts, non-rigid motion capture focuses on reconstructing a mesh that deforms to match a moving surface such as a face or cloth. We also investigate non-rigid motion capture, but we focus on curved splines rather than curved surfaces.

Motion capture for the human face is a well-studied problem. Approaches that use 3D scans of face geometry [5, 35], a model of the underlying tissue [54], and displacement maps [24, 26] have been investigated. These methods are not adequate for capture of rope motion

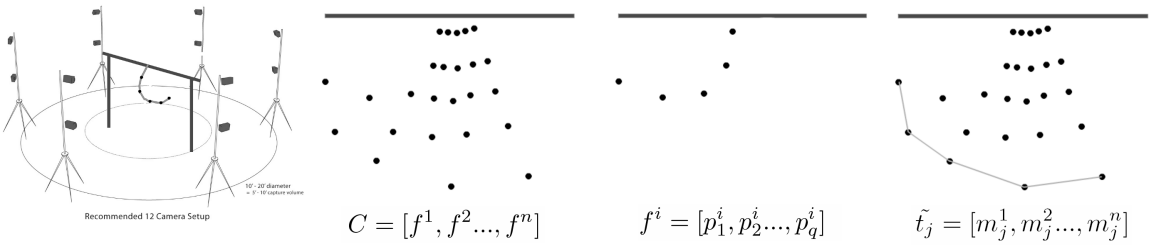


Figure 3.1: Motion capture setup and notation for describing captured data.

because modeling deformed surface motion tracks a 2D surface rather than a 1D spline. In practice a rope can undergo a wider range of motion, (such as coils, loops, and spins) than a face, but markers on rope are visible from all angles, unlike markers on faces, which are visible from only one side of the face.

Motion capture for cloth is also a well-studied problem. Prior work has been based on analysis of video of cloth motion, including customized texture patterns [63], optical flow [27], a stereo pair of images [40], and finding simulation parameters based on video [4]. As with motion capture of the human face, we have posed the motion capture problem for rope differently because rope has a different geometry and range of motions.

A finite element model can simulate rope motion [13], including effects such as tension, shear, bending torsion, contact, and friction. Physical models for involve a parameter system to describe mechanics and structure and can be computationally intensive.

3.3 Motion Capture Data

We use an optical motion capture system produced by NaturalPoint for recording data.[1] This section presents the notation used to describe motion capture data.

After the motion capture arena is set up, an actor stands in the center of the arena. There must be enough space on all sides of the actor so that the rope can move freely and still be in the view of the surrounding cameras. Before recording the motion capture session, the actor assumes a T pose while the rope hangs vertically and separate from the actor. This

[1]The system consists of 12 OptiTrack Flex:V100R2 camera mounted on light poles arranged in a circle with a radius of approximately 2.5 meters.

allows the marker segmentation and rope reconstruction algorithms to be initialized properly. After initialization, the actor can then interact with the rope.

A motion capture session is n frames of motion capture data:

$$C = [f^1, f^2..., f^n],$$

where each frame $f^i (1 < i < n)$ consists of an unordered set of q provisional marker locations:

$$f^i = [p_1^i, p_2^i..., p_q^i].$$

Each $p_j^i (1 < j < q)$ denotes the jth marker position at frame i. Figure 3.1 illustrates these concepts. The left image shows a motion capture setup with a rope dangling from a pole in the center of an arena. The next image shows a complete capture session, C, consisting of n frames of data. Next, the marker positions recorded for a single frame f^i are shown. Finally, the light gray line connects positions in a trace, t_j, of the position of a single marker across frames.

In the following expressions, the superscript i of p_j^i always denotes the frame number. A provisional marker location p_j^i is a triple denoting a 3D position in space. A marker location m_j^i is the location of a specific marker with index j in frame i. An initial trace, $\tilde{t_j}$ is an incomplete estimate of the position of a single marker over time and is a list of marker locations.

$$\tilde{t_j} = [m_j^1, m_j^2..., m_j^n]$$

A final trace, t_j, is an initial trace after processing to repair gaps and undo marker swap.

In a single trace, a gap is a sequence of frames for which no provisional marker locations were assigned. Gaps are caused by either occluded markers or failure to assign a marker

location to a trace. The position of marker j for which a position is not unknown in frame p is denoted by $m_j^p = \perp$ as a gap.

Noise occurs when marker locations are created based on transient reflections in the capture arena. Noise that creates motion outside our assumptions about rope motion is automatically detected and eliminated during data processing.

3.4 Reconstructing Rope Motion

Rope motion can be reconstructed from unindexed marker positions using forward differencing and a novel midpoint displacement algorithm without a physical model assuming that the rope does not stretch. Our aim is to reconstruct rope motion from unindexed motion capture data in the presence of noise and a human actor.

The main process is shown in Figure 3.2. Markers on the rope are first separated from markers on the human actor so that markers tracking rigid bodies (such as the human actor's bones) can be processed separately from markers tracking non-rigid rope. The separation process is described in Section 3.4.5; that section also shares ideas from the reconstructing process. The unindexed points remaining after separating rope and actor motion are likely to represent markers attached to rope and are labeled into separate traces using a nearest-neighbor based clustering algorithm. All traces are assigned to a rope structure with a predefined length. Marker swap and gaps in motion traces are detected and repaired using statistical analysis along with the assumption that the rope does not stretch. Provisional marker locations not included in a trace in the first pass are classified as noise but may not be noise. After the first motion reconstruction pass, markers classified as noise are added to existing traces if they match the rope structure and motion. A post-processing step checks rope length and guides another round of processing if necessary. The process iterates until gaps, noise and rope stretching fall below acceptable standards or the algorithm fails to converge. In our research, some data, especially those with swaps and gaps, required two or three rounds of processing, but most of the collected motion capture data required only one

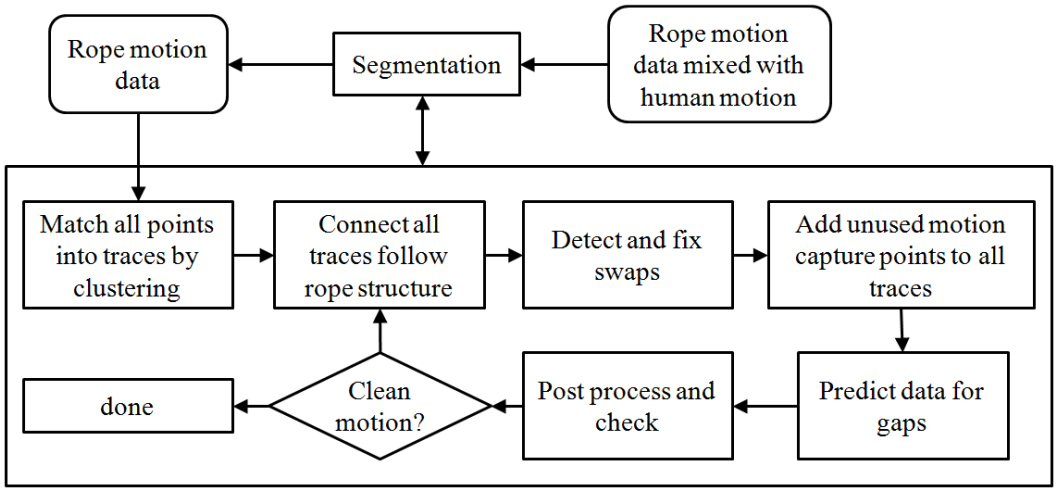


Figure 3.2: Process for converting unindexed marker positions into motion paths.

round of processing and the algorithm did not converge for data with noise and many large accelerations of the rope.

After all motion capture data are matched into traces while noises and gaps are fixed, a Catmull-Rom spline connects all marker positions and creates a smooth curve among these positions on a rope.

3.4.1 Clustering Positions into Traces

For frame f^i and initial trace $\tilde{t_j}$ we attempt to find a provisional position p_x^i that is a good match for $\tilde{t_j}$ in that frame. If no match is found, we set $m_j^i = \perp$.

$$m_j^i = \begin{cases} p_x^i & \text{if } p_x^i \text{ is a good match for trace } \tilde{t_j} \text{ in frame } f^i \\ \perp & \text{otherwise} \end{cases}$$

First, a small portion of the capture session with little or no movement is processed in order to find the number and locations of markers. Next, a second round of clustering assigns provisional marker positions across all frames to zero or one of the identified traces. Positions not assigned to a trace are marked as noise and may be added to a trace later. Each round of clustering is discussed in detail below.

First Round of Clustering

The first round of clustering is performed on a stationary prefix session in which the markers remain stationary for approximately one second. This is similar to capturing a T pose during a motion capture session with a human actor.

Two parameters, γ and minPoints, are predefined. Parameter γ is a 3D vector of the maximum allowable movement range for a marker during the stationary prefix. Parameter minPoints is a threshold for the minimum number of provisional marker locations that must be assigned to a trace for that trace to be retained in the next round. A trace set T is initialized by separate initial traces $\tilde{t_j}$ as

$$T = \tilde{t_1}, \tilde{t_2}, \tilde{t_3}....$$

In frame i, a position p_x^i is appended as a marker location to trace $\tilde{t_j^i}$ when it is the closest provisional marker location to the average position of all the previous marker locations in a trace and it is within the threshold γ. If no such provisional location exists for a trace $\tilde{t_j^i}$, a gap $\tilde{t_j^i} = \perp$ is marked for trace j in frame i. When a provisional marker location p_x^i is not appended to any trace, a new trace $\tilde{t_j^i}$ is initialized with $\tilde{t_j^i} = p_x^i$. After all provisional marker locations are processed, traces which have fewer recorded positions than the threshold minPoints are discarded and the positions are marked as unused data for future analyses.

Second Round of Clustering

The second round of clustering matches markers to traces using forward differencing (FD) to predict future marker locations. FD with order s predicts an expected position D_j^i for trace $\tilde{t_j^i}$ in frame i. Differencing Δ_j^s is computed with order $s (s > 0)$ for trace $\tilde{t_j^i}$ from the previous frames $(i - s)$ to $(i - 1)$:

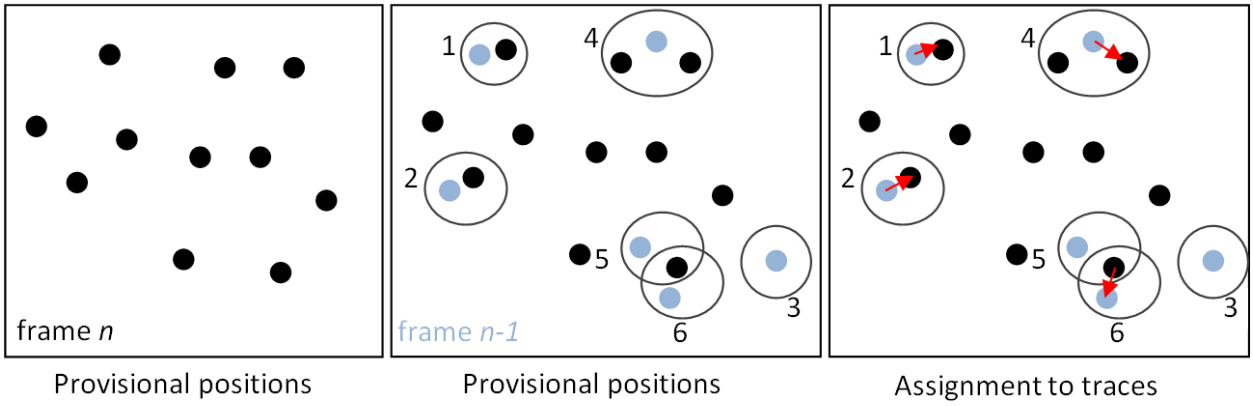


Figure 3.3: Clustering process for the second round.

$$\Delta_j^s = \Delta^s m_j = \sum_{w=1}^{s} (-1)^s \begin{pmatrix} s \\ w \end{pmatrix} m_{j+s-w}. \tag{3.1}$$

After getting the differencing values Δ_j^s, the predicted new position D_j^i for trace j in frame i is estimated as

$$D_j^i = \sum_{w=0}^{s} \begin{pmatrix} s \\ w \end{pmatrix} \Delta_j^w m_j^{i-w}. \tag{3.2}$$

The predicted position D_j^i preserves motion continuity from the previous s frames in trace t_j. Finding a new provisional marker position in the neighborhood of the predicted position D_j^i is more precise than searching in the locus of the average position because the expected position varies with velocity and acceleration.

However, large accelerations can result in a predicted marker position that is at a different location than the actual marker. If the actual marker lies outside the neighborhood of the predicted position then the marker will be missed, a gap will be created in the trace, and the marker position will be marked as noise. Later, after repairing gaps and swaps, it may be possible to insert the missing marker position into the trace.

Figure 3.3 illustrates the clustering process. The image on the left shows the provisional marker positions in f^i. FD on markers 1 through 6 from f^{i-s} to f^{i-1}, where positions of markers 1 through 6 from f^{i-1} are shown in light gray, is used to identify likely positions of

markers 1 through 6 in f^i. These likely positions are called neighborhoods and are contained in the gray circles in the middle image. Next, for all marker positions m_j^i in the neighborhood of D_k^i compute:

$$\tilde{t_k^i} = m_j^i = min_{\forall p_x^i \forall D_y^i} \left\| p_x^i - D_y^i \right\|. \tag{3.3}$$

A provisional marker location p_x^i is appended to trace $\tilde{t_k^i}$ if p_x^i is the nearest provisional marker location in the neighborhood D_k^i predicted from trace $\tilde{t_k^i}$. This is shown on the right side of Figure 3.3 as one of the provisional markers is assigned to traces 1, 2, and 4. In each case, the provisional marker location is assigned to a trace that has the nearest neighborhood D_k^i to the location. If no such p_x^i lie within the forward differencing estimation then $\tilde{t_k^i} = \perp$. In Figure 3.3, trace 3 is assigned to $\perp$ because no provisional locations lie close enough to the position of trace 3 from f^i. A provisional marker location p_x^i might be mapped to more than one trace—as is the case for traces 5 and 6 in the figure. In this case, the closest trace keeps the provisional marker location p_x^i while the others are marked as having a gap. In this case, trace 6 is closer to the shared position and trace 5 is marked with a gap. If a provisional marker location p_x^i is not clustered with any trace then that position is marked as unused data. The remaining black dots in the right side of Figure 3.3 are marked as unused data and may be assigned to traces later, which is discussed in Section 3.4.4.

Figure 3.4 demonstrates a trace graph for 5 dangling ropes instrumented with 5 markers each. The left image is before clustering and the right is after. All traces are painted in different colors. Black circles indicate the stationary positions of each marker. Noise and unused data are detected but not shown in the image of the reconstructed traces on the right.

3.4.2 Swaps

Marker swap occurs when marker positions are assigned to the wrong trace. In our data, marker swaps occur at a rate of about one swap per thousand frames of data. While swaps are uncommon, swaps lead to visually implausible animations, as the rope appears to twist

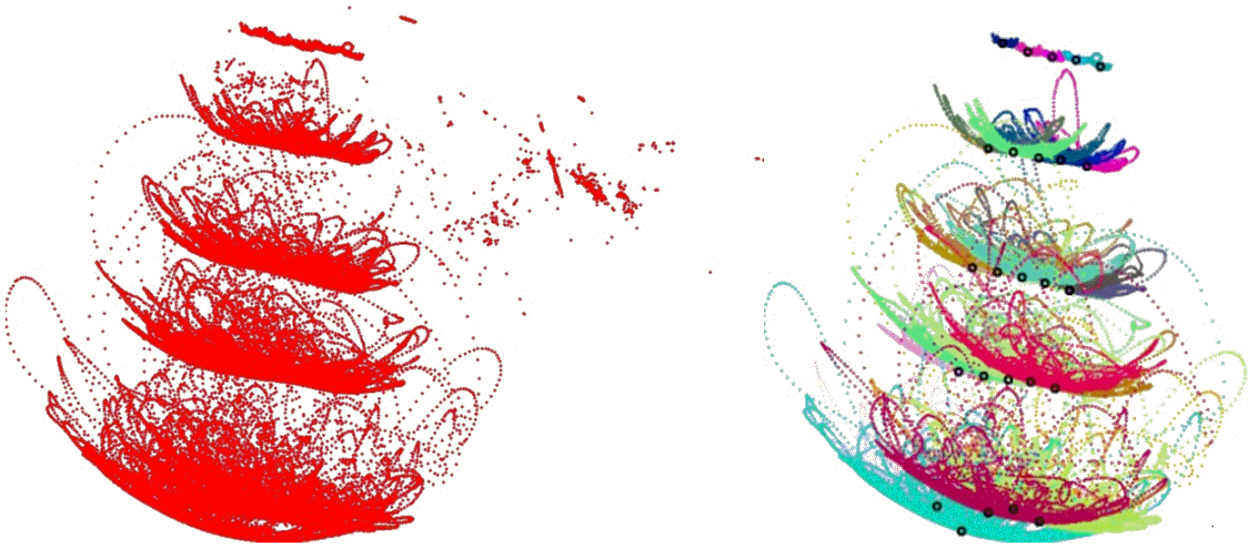


Figure 3.4: An input point cloud and clustering result.

and bend unnaturally. In many cases, swaps lead to the appearance of the rope stretching. This is the key to detecting such swaps. A rope with length greater than the initial stationary length is called a stretched rope and likely indicates a marker swap.

A segment is the span of rope between any two markers on a single rope. Stationary marker positions from the first round of clustering are denoted $\bar{m}_x$ and $\bar{m}_y$. With a distortion factor α for length error tolerance, a rope segment between markers m_x and m_y is considered stretched if

$$\left\| m_x^i - m_y^i \right\| > \alpha * \left\| \bar{m}_x - \bar{m}_y \right\| . \tag{3.4}$$

The distortion factor α is small enough to detect swaps before they are visually significant. We set the value of α in [0.1,0.15].

We implemented two approaches for finding rope stretching: a case-based approach and brute force. The case-based method only checks rope length when there is a reasonable chance of swapping while the brute force approach checks rope length in every segment in every frame.

In the case-based approach, we collect candidate swapping points based on two assumptions: (1) when the distance between any two provisional marker locations is smaller

than a threshold *minDist* and (2) after a gap whose length is bigger than a threshold *maxGapLen*. After collecting all candidate swapping points, the segment length at a candidate swapping frame is checked using equation 3.4. After two markers approach in space, the following N frames have higher probability of containing a swap because the clustering-based labeling process might swap positions. Therefore, segment length is checked the N frames after markers pass in close proximity. A related segment is a rope segment that contains candidate swapping points. A swap of two candidate points is detected and repaired if the potential swapping satisfies two requirements: (1) before the swap, both lengths of related rope segments are stretched, and (2) after the swap, at least one of these lengths is corrected.

In the brute force method, the rope length is checked in each frame using equation 3.4 with a distortion factor α. When rope length is stretched in a frame, we try all permutations for all marker positions for all ropes in a scene. Swap repair stops when all rope lengths are not stretched or when all permutations have been checked. However, in the worst case, the number of permutations is in the order of $n!$, where n is the number of marker locations. That makes this approach infeasible in most cases.

3.4.3 Gaps

Given a trace, a gap is a sequence of frames in which recorded marker positions were not assigned to that trace. Gaps are repaired by filling gaps with estimated locations. A small gap has less than five continuous missing frames. Small gaps are fixed by linear interpolation. Fixing a large gap uses assumptions about rope shape, statistical characteristics of the marker and the movement of other markers on the same rope. This section focuses on repairing large gaps.

A gap session is a span of time in which multiple traces have gaps. During a gap session, multiple traces might have gaps with different durations. Smaller gaps and gaps in traces with small ranges of motion (if any) are repaired first.

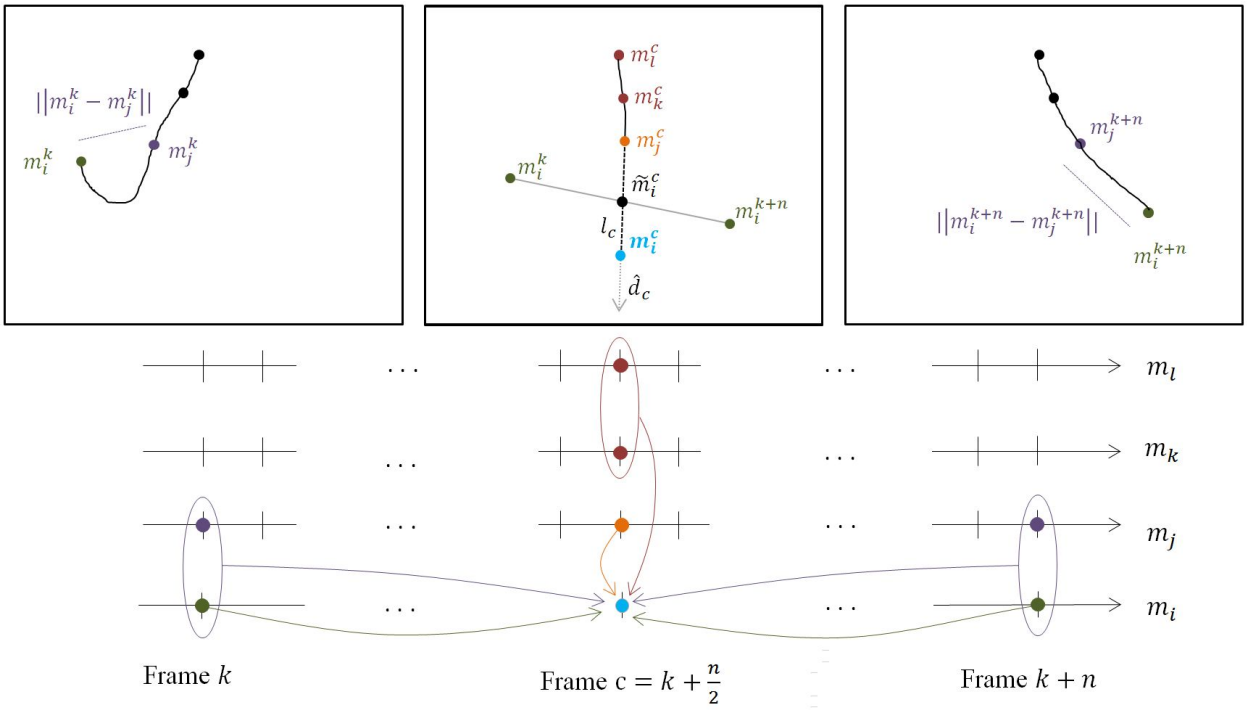


Figure 3.5: Estimating the position of marker m_i in the center frame of a long gap using the positions of markers m_j, m_j1, and m_j2.

A recursive midpoint displacement method estimates marker positions through large gaps. This process is shown in Figure 3.5. Suppose the trace of marker m_i includes a gap from frames k to $k + n$ and that the trace of nearby marker m_j is complete over the same span. Marker m_j is adjacent in the sense that it lies next to, or near, marker m_i on the rope segment. Figure 3.5 contains rope positions in the top series of frames, and timelines at the bottom of the figure show which marker positions in which frames are used to estimate the position of marker m_i at the frame $k + n/2$ in the middle of the gap. Marker locations in the top series of frames and the timelines are color-coded. The process involves estimating the distance between m_i and m_j, estimating the position of m_i, and then estimating direction from m_j to m_i. These estimates are generated from known marker positions in frames k, $k + 2$ and the midpoint frame.

The estimated distance between known markers for the segment at the middle frame, l^c (where $c = k + n/2$), is computed by averaging the segment length between m_j and m_i

(shown in purple in the figure) starting in frame f_k and ending in frame f_{k+n}.

$$l^c = \frac{\left(\left\|m_i^k - m_j^k\right\| + \left\|m_i^{k+n} - m_j^{k+n}\right\|\right)}{2} \tag{3.5}$$

The preliminary location of m_i in frame c, which is denoted $\tilde{m_i^c}$, is the average of the locations of m_i at the start and end frames.

$$\tilde{m_i^c} = \frac{m_i^k - m_i^{k+n}}{2} \tag{3.6}$$

The position of m_i at the start and end frames is shown in green in Figure 3.5.

Next, the direction from m_j^c to m_i^c is estimated using the weighted sum of the direction from m_j^c to $\tilde{m_i^c}$ and the direction between two other nearby markers m_{j1}^c and m_{j2}^c (shown in red in Figure 3.5). The direction $\hat{d^c}$ is given by

$$\hat{d^c} = Norm\left[\alpha * \left(m_j^c - \tilde{m_i^c}\right) + \beta * \left(m_{j1}^c - m_{j2}^c\right)\right], \tag{3.7}$$

where α and β are weighting parameters.

Finally, the location of marker m_i^c in frame c is determined by moving a total of l^c units along the normalized vector $\hat{d^c}$ starting at known marker location m_j^c:

$$m_i^c = m_j^c + l^c \hat{d^c}. \tag{3.8}$$

The new position is shown in blue in Figure 3.5.

The midpoint displacement method proceeds recursively and fills data in all the gaps in the gap session. When length of a sub-session gap is fewer than 5 frames, it will be fixed using linear interpolation.

3.4.4 Add Unused Data

The unused data are revisited and added to existing traces if applicable. The nearest unused provisional location p_y^i to a marker location $m_x^{i\pm1}$ in the neighboring previous or next frame fills a gap location $m_x^i = \perp$ if

$$m_x^i = p_y^i = min_{\forall m_x^i \forall p_y^i} \left\| p_y^i - m_x^{i\pm1} \right\|, \tag{3.9}$$

where $y \in y_1, y_2, ...y_z$ and ranges over unused locations.

The segment length resulting from using the closest provisional location p_u^i is evaluated using equation 3.4. If the segment is not stretched and equation 3.9 is satisfied, the unused provisional location will be added in a trace to replace a gap m_x^i.

The process is repeated over the entire session. When there is a gap, the left and right ends of the gap are processed first because there are data available in the neighboring previous or next frames. This process is repeated for all gaps until no more unused data is added to a gap.

3.4.5 Separate Rope Markers from Actor Markers

Rope motion captured with a human actor is separated from the actor's motion. The segmentation is an important step in our method for recreating animation of an actor interacting with rope. Rope markers are separated from markers on the actor after grouping markers into traces across frames. The clustering method used to group markers is identical to the clustering method described above. Alternatively, it may be possible and more desirable to separate rope and actor markers by tracking the actor skeleton as a collection of rigid bodies and classifying the remaining marker positions as rope markers.

After the segmentation, rope motion is reconstructed from purported rope marker positions by repairing gaps and swaps and then adding unused markers as described previously.

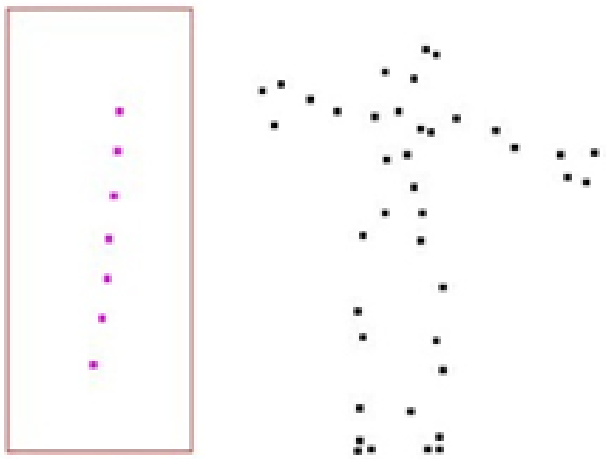


Figure 3.6: Rope marker selection.

To initialize rope marker locations, a user makes a simple selection of all and only the markers that belong to the rope, as shown in Figure 3.6.

As in Section 3.4.1, at each frame we attempt to find a provisional marker location that is in close proximity to the next estimated position for a marker trace. When no provisional marker location can be found for a trace, instead of labeling a gap, we create a temporary marker for the marker trace. The temporary marker allows the algorithm to continue tracking a marker despite the presence of noise and gaps in the motion capture data.

In order to track a marker, we match provisional markers to traces using a linear combination of two predicted positions, V_j^i and N_j^i. The first predicted position, V_j^i, is based on the last known velocity of the marker using equations (3.1), (3.2), and (3.3). The other position, N_j^i, is predicted using a midpoint displacement method, as described in Section 3.4.3, using equation (3.5)–(3.8).

Then V_j^i and N_j^i are combined as follows:

$$F_j^i = \alpha V_j^i + (1 - \alpha) N_j^i, \tag{3.10}$$

where α is one over the number of consecutive temporary markers. This means that the more consecutive temporary markers the trace has the less we rely on V_j^i, because it could have strayed too far from the actual rope markers. After fully segmenting the rope markers from the human markers it is possible to reconstruct the motion streams for each independently.

3.5 Results

We present results that indicate that, under the assumption that the rope does not stretch, rope motion can be plausibly reconstructed from passive optical motion capture data without a physical model of rope dynamics. This can be done in the presence of a human actor whose motion is tracked in the same capture session.

Figure 3.7 shows the reconstruction of a motion capture session in which the character is jumping rope. Comparison with still frames taken during the capture session shows that both the marker segmentation and reconstruction algorithms accurately reproduce the original motion.

Gap filling results in motion that is similar to missing motion in both magnitude and direction and has good continuity with surrounding motion. Figure 3.8 shows the average error for reconstructed marker positions for gaps created by holding back data from a motion capture session. The horizontal axis shows the gap size in frames and the vertical axis shows the mean error for all frames in gaps of that size. The data in Figure 3.8 were generated from 5,000 randomly generated gaps. The curve is not smooth because of the random factors we included in the analysis. A gap created at different phases of a motion path might produce different values of error.

Figure 3.9 compares the original shape of the model with our results. In the top row of Figure 3.9(a), we compare the ground truth motion data (left) with data generated by the gap-filling algorithm (right). The data shown in dash-dots on the left and right sides of each image is held back during gap filling. The synthesized motion, which is the bottom marker

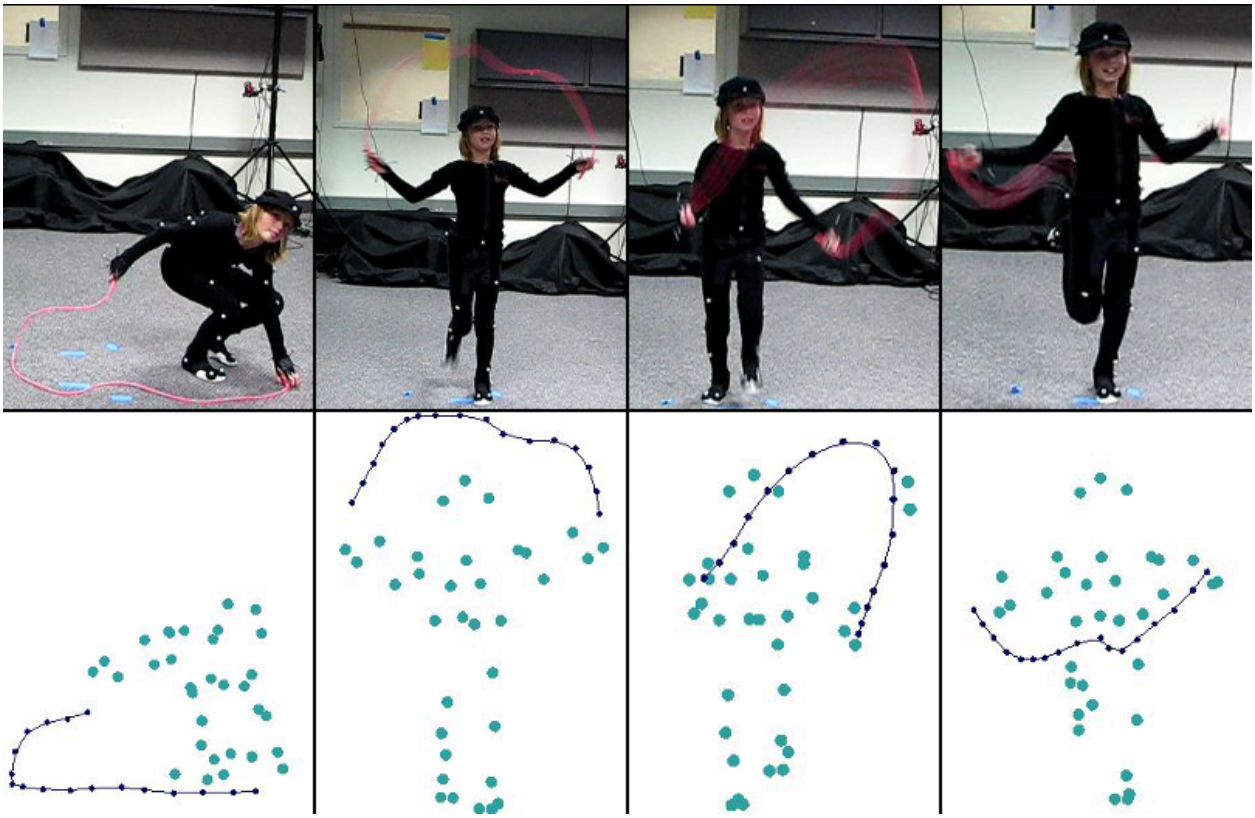


Figure 3.7: Screenshots of the original motion, segmented rope and character markers, and the reconstructed rope with the human markers.

in solid lines on the right image of (a), preserves good continuity with existing motion and rope shape. Figure 3.9(b) shows traces (right) created from raw point cloud (left).

The process accurately extracts motion in a variety of settings. Still images from both video taken during a capture session and reconstructed motion from that session are shown in Figure 3.9(c)–(f). Each scenario is described subsequently, and complete video clips are included as reference material with this paper.

Five dangling ropes, as shown in Figure 3.9(c), have one end fixed to a horizontal support bar and rope motion is driven by hitting the ropes with a stick. This scenario is easy to process because one end of the rope is anchored to a fixed position in space and rope motions are relatively small. The resulting motion is smooth with few gaps and little noise.

We recorded a rope dropping from the horizontal support bar (shown in Figure 3.9(d)). The rope has one end fixed to the bar. The rope is nudged over the edge of the bar using a

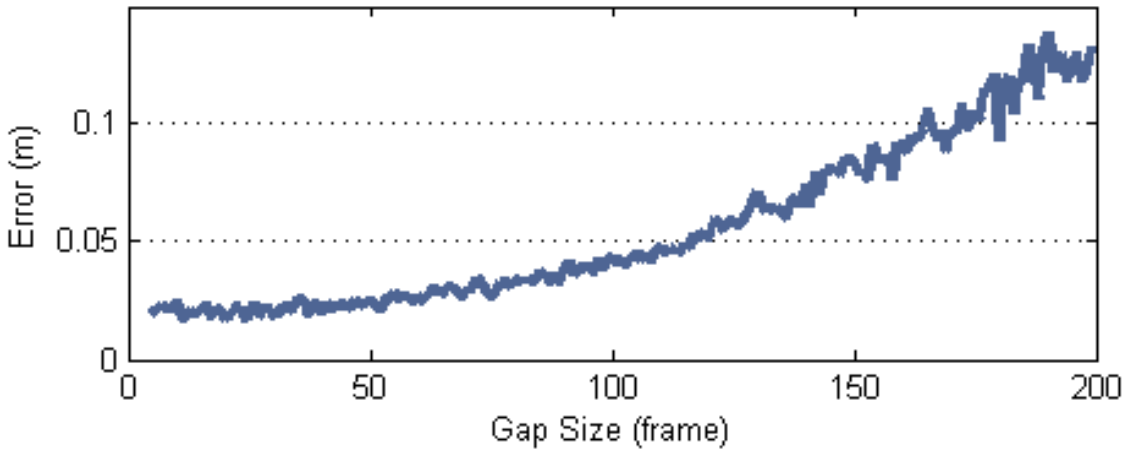


Figure 3.8: Analysis of gap-filling algorithm. The horizontal axis is gap size and the vertical axis is error, which is the difference between the motion captured data after clustering and the data estimated by our method.

stick. This scenario is more complex than the five dangling ropes because the rope moves more quickly and with more bending. We processed this data in inverse time order, so the structure of dangling rope is clearer at the end than at the beginning of the session .

The motion of unanchored ribbons is more difficult to process because ribbons have less mass and are not anchored in this example. Figure 3.9(e) and (f) show two frames from two sessions involving ribbons attached to a wand that is waved freely in the capture arena. Ribbon is also more difficult because it is a lighter material that accelerates with less force than rope. By mapping different textures to the single ribbon motion, we create some interesting effects, such as animating lace cloth and a Japanese fish flag based on captured data.

3.6 Conclusion and Discussion

Our work produces visually plausible rope motion from passive optical motion capture data using a statistical model under the assumption that the rope does not stretch. The algorithm uses a novel clustering scheme, forward differencing, and a recursive midpoint scheme to automatically detect and remove most noise, gaps, and marker swaps. The algorithm preserves

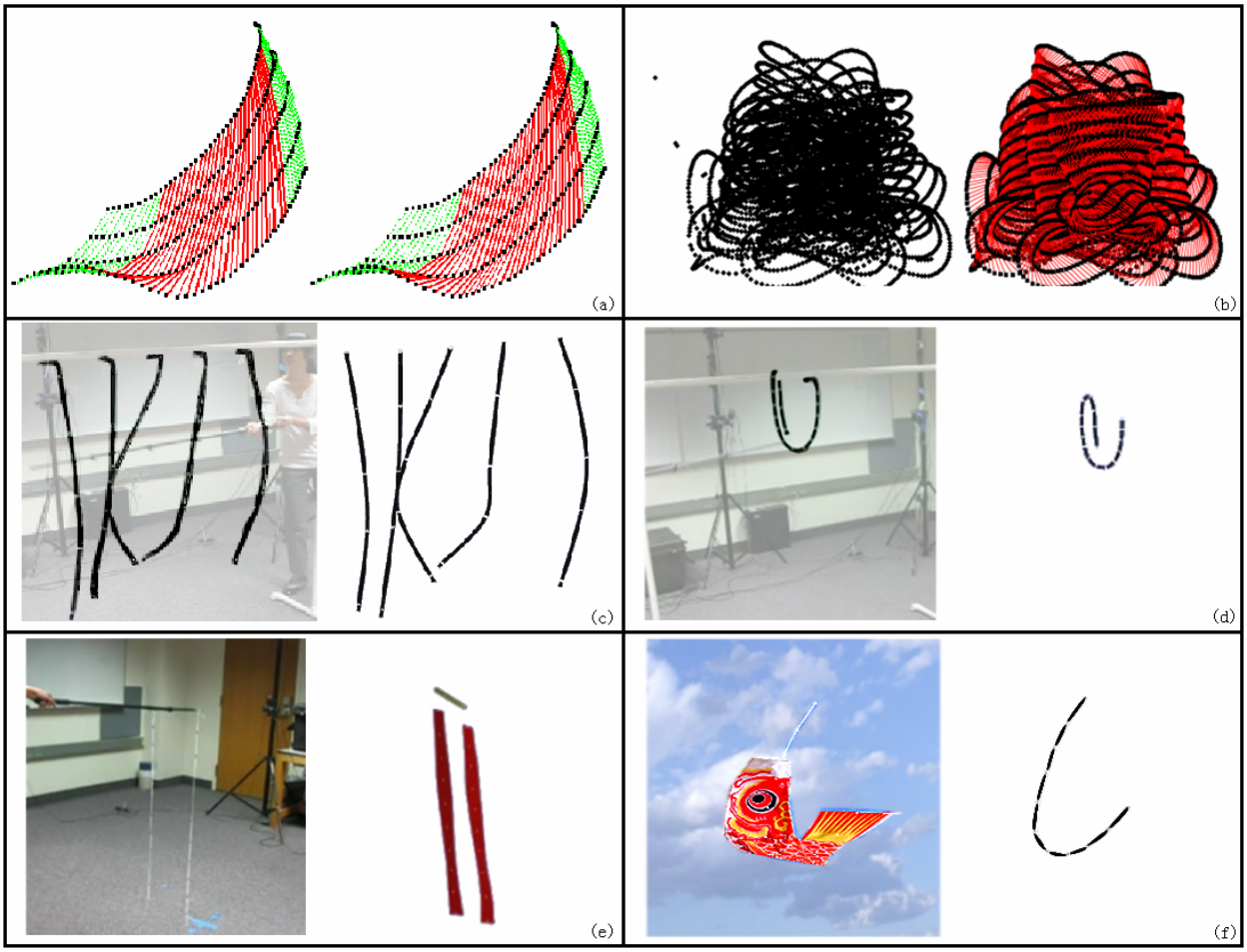


Figure 3.9: Screenshots of reconstructed motion of ropes and ribbon.

continuity of motion in traces and fits the shape of rope. This work lays a foundation for further investigation of motion capture for non-rigid bodies using statistical rather than physical models. The approach to the problem may advance motion capture results for non-rigid bodies driven by complex or poorly understood physical systems.

Complicated motions (such as spirals, collisions, sudden changes in movement, or extremely fast movement) are not well handled in this model. Our assumptions for detecting swaps may be oversimplified relative to natural movement. Consideration of other factors, such as velocity or acceleration, might improve gap-filling results. We have used a simple method for interpolating rope position between markers. More complex methods may result

in more plausible results particularly when the distance between markers on the rope is large.

Chapter 4

3D Tree Modeling Using Motion Capture

Jie Long and Michael Jones. 3D Tree Modeling using Motion Capture. *IEEE The Fourth International Symposium on Plant Growth Modeling, Simulation, Visualization and Application (PMA '12)*, to appear.

Abstract

Recovering tree shape from motion capture data is a first step toward efficient and accurate animation of trees in wind using motion capture data. Existing algorithms for generating models of tree branching structures for image synthesis in computer graphics are not adapted to the unique data set provided by motion capture. We present a method for tree shape reconstruction using particle flow on input data obtained from a passive optical motion capture system. Initial branch tip positions are estimated from averaged and smoothed motion capture data. Branch tips, as particles, are also generated within bounding space defined by a stack of bounding boxes or a convex hull. The particle flow, starting at branch tips within the bounding volume under forces, creates tree branches. The resulting shapes are realistic and similar to the original tree crown shape. Several tunable parameters provide control over branch shape and arrangement.

Figure 4.1: Maple model.

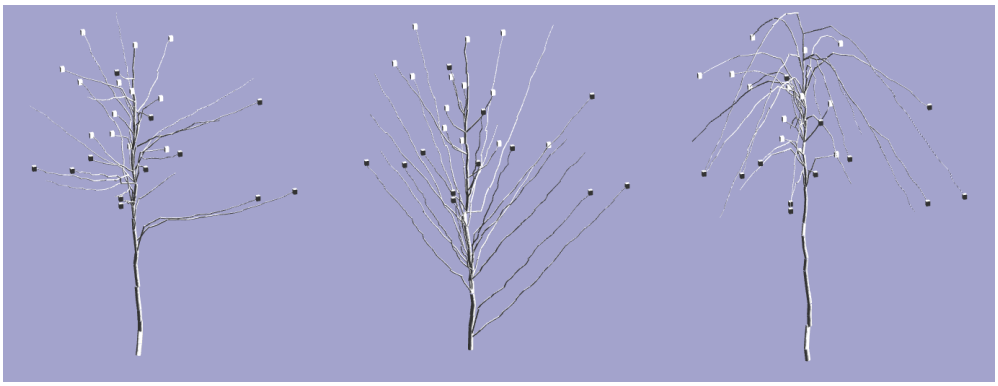


Figure 4.2: Various tree shapes generated from the same set of motion capture data.

4.1 Introduction

Reconstruction of tree shape from motion capture data is an important step in replaying motion capture of trees under external forces, such as natural wind. Motion capture provides a fast and easy way to collect the locations over time of retroreflective marker locations placed on an object. In this paper, we address the problem of creating 3D tree shape from motion capture data. We also discuss best practices for collecting motion data from a tree. This research focuses on reconstructing static 3D tree shape with branching skeletons from data collected by a motion capture system.

Solutions to the motion capture problem for trees can be applied to problems in visual effects and the study of tree motion. Motion capture is a potential solution because motion capture data includes effects that are difficult to model in simulation, such as variable branch stiffness, non-uniform variation in size, and emergent effects due to leaf deformation.

Tree shape modeling has long been studied in computer graphics. Past methods include particle systems [29, 34, 46, 50], L-systems [20, 21, 43], parametric models [60], photographs [29, 45, 58], and videos [8, 19]. Most of these methods result in satisfying tree shapes but do not leverage the 3D positions of motion capture markers, which are recorded as part of a motion capture session but require a different set of inputs. Image- or video-based approaches convert a set of 2D input images into 3D tree models by filling in the missing dimension. Motion capture systems can record tree shape in 3D with high precision (using similar techniques for converting a set of 2D images to a 3D model). Prior work in reconstructing tree shape from motion capture data using either exact measurements or markers placed within the crown does not scale and does not apply when leaves occlude the branching structure.

We reconstruct 3D tree shape using particle flow with motion capture data as input. Passive optical motion capture[1] records locations of reflective markers in the capture arena. We place markers only at branch tips on the edge of the tree crown. Approximately 30 markers cover the crown shape of a medium-sized tree. We do not put markers on each branch tip because passive optical motion capture systems cannot reliably track more than 70 markers. A particle flow algorithm generates branching structures starting at the recorded tip positions. Additional starting points are defined within the estimated volume of the tree crown using a vertical stack of bounding boxes or a convex hull. The bounding space approximates the tree crown and bounds particle flow and creation. The step length of a particle's flow varies with the distance to the nearest trunk point. The direction of particle motion is a combination of three forces: gravity, shape-format, and wind. The shape-format guides the particles to preserve the original tree shape. The dominant wind direction is recorded during the motion capture process. We also introduce two vectors with one pointing to the nearest trunk point and the other pointing to a constant predefined attractor point. These vectors are factors for the direction and magnitude of the forces.

[1] http://www.naturalpoint.com/optitrack/

In this paper, we propose a new particle flow method for reconstructing tree shape from only motion capture data. Our primary contributions are

- a simplified particle flow algorithm for constructing tree shapes from a sparse set of branch tip positions as collected as part of motion capture and
- a method for deploying passive optical motion capture to reconstruct the shape and motion of trees in the laboratory.

The combination of motion capture with a particle flow method provides a fast and easy approach for creating complex 3D tree shapes. The resulting tree shapes are similar to the original trees, as shown in Figure 4.1, and can be used to replay motion similar to the captured motion. With the flexibilities of tuning the forces, we can produce several tree shapes besides the original shape. Figure 4.2 shows various tree shapes generated from the same set of motion capture data and demonstrates the flexibility of our forces for guiding particle flow.

4.2 Related Work

Our work is most closely related to prior work in tree modeling and applied motion capture. Our purpose is to investigate methods for creating tree shapes on which motion capture data can be replayed. Compared to prior work in modeling trees, we use motion capture as our equipment to collect partial information of tree shape and run particle flow to complete the modeling. Compared to prior work in motion capture, we design a new data collection process for non-rigid bodies and reconstruct realistic 3D models out of the data.

4.2.1 Tree Modeling

L-systems [20] generate a tree's branching structure using axioms and rules in a concurrent context-free rewriting system. L-systems have been extended in many ways. The extension most relevant to our work is [43], in which L-systems are enriched with partial differential

equations and can be parameterized to reconstruct the shape of a specific tree or plant. We chose not to investigate L-systems, or Weber and Penn's parametric tree model [60], for this application because particle flow from recorded branch tip positions closely matches the data collected in motion capture.

More recent work in tree shape modeling involves particle systems. With the exception of Palubicki's work [34], particle systems approximate tree shapes obtained from photographs [29, 45, 58]. Palubicki et al . devised a particle flow which approximates bud fate models. These methods are not directly applicable to motion capture data because these methods use photographs or environmental conditions that are not collected during motion capture. Photographs of the tree could be taken during motion capture (and indeed are taken during optical motion capture), but we only take marker positions as input because this simplifies data collection and processing by reusing the background removal and image alignment performed as part of marker position calculation.

4.2.2 Applied Motion Capture

Motion capture has been mainly used in rigid bodies, such as human motion. It produces positions for points on an object over time with very small measurement errors.

Motion capture systems have been widely used for human or animal motion capture [25, 44, 62]. Kirk [16] automatically generates rigid skeletons from optical motion capture systems by preserving a constant distance for each rigid part. These algorithms assume that the distance between markers on the same bone is invariant and cannot be directly applied to non-rigid bodies, such as natural trees, because the distance between markers is not invariant as the object deforms.

Prior work in motion capture for non-rigid bodies includes several approaches to facial motion (including [26, 54]). These methods are based on domain-specific features of the facial structure or patterns. Obviously these domain-specific features do not apply to tree shape reconstruction or motion capture.

The uniform branching structure of human bodies has led to well-understood processes for deploying markers on a human. The number and placement of markers is a critical part of successful motion capture using a passive optical system. Trees have a more complex and less predictable topology and require a different approach to marker placement. Previously [23], researchers attempted to directly replay the motion of a tree in wind following the exact motion paths collected for all branches by putting markers on every branch of the tree. Leaves on the tree create marker occlusion, which results in poor data. In addition, manually defining branch topology to exactly match the subject tree is labor intensive. In this paper, we design a data collection and tree modeling process to overcome these difficulties by building a similar, but not exact, copy of the branching structure from a partial collection of branch tip positions. Ongoing work focuses on replaying collected motion such that the motion looks natural on an approximate copy of the branching structure.

4.3 Motion Capture of Trees

In this section, we describe how to collect data from trees using a motion capture system such that the data supports reconstruction of tree shape. The data is collected indoors on trees with heights less than 2.5 meters. The data collection process results in an unindexed set of marker locations over time for a small set of instrumented tree branch tips.

We use a passive optical motion capture system (Optitrack V100 by NaturalPoint)[1] to collect data. The passive optical capture system strikes a balance between conflicting features. Passive optical systems can reliably track up to 70 markers, and some markers weigh only a few grams. This is ideal for working with tree crowns. Active optical and active magnetic systems use heavier markers and cannot track more than 20 markers at once. The magnetic systems have the advantage of not having visibility occlusions and being able to track position and rotation, but they are more expensive than passive optical systems and

[1] http://www.naturalpoint.com/optitrack/

can track fewer branch tips. Magnetic system markers are heavier and may alter branch tip motion.

Collecting data from natural trees is challenging for passive optical motion capture systems because trees are non-rigid bodies and are partially self-occluding. The following method for deploying passive optical motion capture systems collects data from which tree shape and motion can be inferred.

Twelve or more cameras are arranged in a circle around the tree with six cameras located approximately 0.8 meters above the ground and another six cameras are located 3.3 meters above the ground. For each camera, the field of view is adjusted to include the entire tree. About 30 markers are placed on branch tips throughout the crown. Markers are square retroreflective markers with a surface area of about 1 cm and adhesive backing. Markers are placed to cover branch tips on each major branch from the stem and to provide nearly uniform coverage of the crown. On these branch tips, the square-shaped markers are wrapped to cover the whole tip so these markers are visible to most of the cameras from different view angles. Uniform coverage improves both shape reconstruction and motion capturing. Placing markers such that their motion paths overlap complicates algorithms for extracting motion paths from unindexed marker positions. For a medium-sized tree, the number of branch tips exceeds the number of markers so that not every branch tip is covered by a marker. Leaves around the marker are removed to improve marker visibility so the resulting 3D tree shape preserves the shape of the original tree crown. While recording tree motion we use an electric fan to create wind around the tree because data is collected indoors. The wind direction is inferred from where the fan sits relative to the tree. Other statistic methods, such as PCA (principle component analysis) can also compute the dominant wind direction after motion data of branch tips are processed.

Photographs in Figure 4.3 show the arrangement of the motion capture cameras and the reflective markers as deployed on an indoor pine tree. Markers are placed at branch tips

Figure 4.3: Marker placement on the crown periphery avoids occlusion while generating data that can be used to generate crown structure.

shown as white dots in the image. The right side image shows marker locations in the red box on the left side image.

Our design requires less user intervention, produces cleaner motion capture data , and may support animation of tree motion.

4.4 Data Processing

Simply inferring positions from one frame of captured data is not adequate due to noise. These branch tips with markers are called "recorded" or "captured" tips. However, the initial locations of recorded branch tips may contain errors due to noise in either the system or the capture environment.

A clustering algorithm approximates a single initial position from a collection of captured initial positions for recorded branch tips while minimizing error from the motion capture system. The clustering algorithm analyzes positions over many frames of motion capture data and eliminates gaps and noise. Gaps occur when a marker is not present in a frame. This algorithm uses forward differencing to predict a position for a marker at frame M based on positions in previous frames. The closest marker in the next frame is added to the motion trace for the marker if there is a marker located close enough to the predicted position. If there is no marker position recorded close enough to the predicted position, that marker is marked with no data, i.e. a gap, for that frame. Gaps are repaired using interpolation over

the motion path for a marker. If the number of marker positions recorded for a marker over time is less than 1/4 of the total frames, that partial marker trace is marked as noise and eliminated.

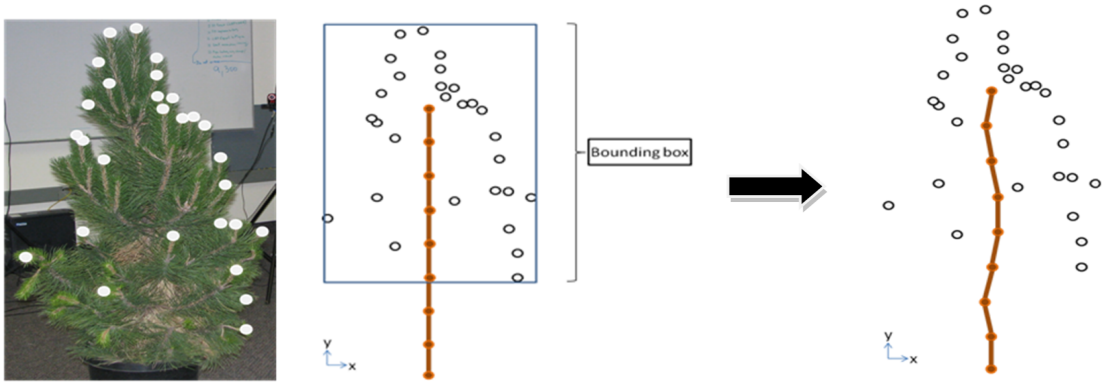
If the capture process includes several hundred frames of data in which the tree is not moving, averaging branch tip positions over these frames results in precise estimates of initial positions. In most cases, the number of markers inferred from the clustering process matches the number of markers placed on the tree.

4.5 Generating 3D Tree Shape

Particle flow is a well-studied approach to generating 3D branching structure for trees [29, 34, 46, 50]. We adapt the method to motion capture data. The 3D tree crown boundary inferred from motion capture data constrains the particle range and preserves the original 3D tree silhouette. We use a stack of bounding boxes or convex hulls to represent the crown boundary. Particles, either generated randomly in the boundary or locations of branch tips recorded by motion capture, are moving towards trunk nodes. Three forces—gravity, internal force, and external force—drive the particle flow process. The paths of the particle flow produce branching structure. By attaching leaves to the branching structure, we generate a 3D tree model that has similar appearance as the natural tree shape.

We synthesize a trunk in the center of the crown shape, as shown in Figure 4.4. Figure 4.4a shows a photograph of a pine tree with markers placed on its crown. Figure 4.4b contains a bounding box of these marker locations and a straight vertical line representing trunk shape. The length of the line is scaled by the crown height of the bounding box. On this line segment, we generate about 10 trunk nodes. Random offsets to these nodes in the x and z direction are added as shown in Figure 4.4c.

A particle represents a branch tip. One group of particles is 3D positions of branch tips recorded from motion capture, which is described in Section 4.4. Figure 4.4a shows a photograph of a pine tree with markers placed on the crown. All the branch tip locations



(a) A pine tree with markers.

(b) A straight line represents the trunk.

(c) Adding random offsets looks more natural.

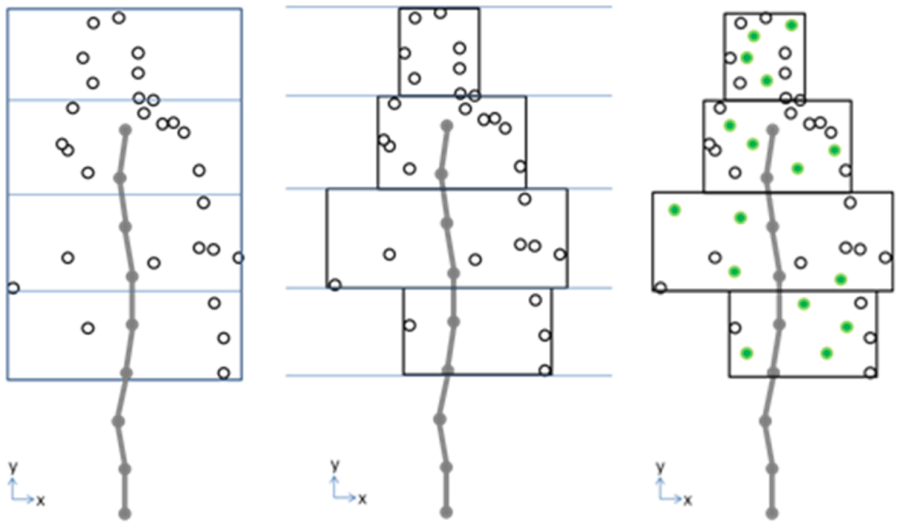
Figure 4.4: A simple trunk model is added to the collection of branch tip positions.

recorded by motion capture are labeled with white dots. These particles are shown in black circles in Figure 4.5c and Figure 4.6.

Because of motion capture's limited capability, we cannot record locations for every branch tip on the tree. Another group of particles is randomly generated within the bounded space of all the recorded branch tip positions. In Figure 4.5c and Figure 4.6, these particles are green.

Instead of using one single bounding box for the whole tree crown, we create a vertical stack of bounding boxes as shown in Figure 4.5. The new bounding boxes more closely match the tree shape. In Figure 4.5c, the crown height is evenly divided into four parts. Particles are randomly generated inside the smaller bounding boxes, as shown using green dots in Figure 4.5c. The total number of branch tips, including motion capture branch tips and randomly generated particles, is set to be similar to the original tree. The number of green dots in each box is proportional to the number of black dots. Therefore, we maintain a similar total amount and distribution of natural tree branch tips.

Alternatively, we use a convex hull for all the particles, as shown in Figure 4.6. A convex hull provides more precise bounding space than the stack of bounding boxes. However, it requires a (slightly) more complex boundary detection scheme and more implementation



(a) (b) (c)

Figure 4.5: New particles are added within a vertical stack of bounding boxes.

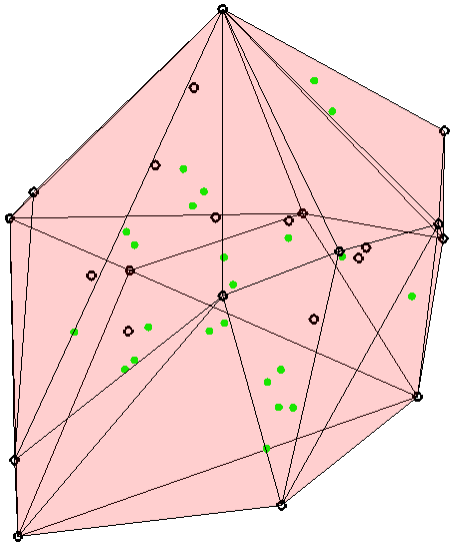
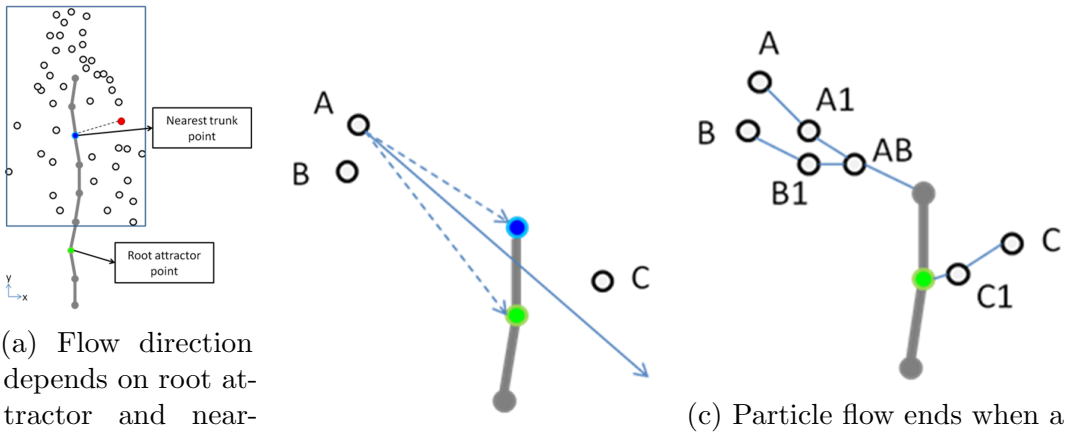


Figure 4.6: A convex hull for all the particles. Particles in black color are from motion capture and particles in green color are randomly generated inside the convex hull.

details. Because of the precision that a convex hull brings, we recommend this approach for building a bounding volume.

For the pine tree's trunk, we generate nine nodes in a straight vertical line and add random offsets in the x and z directions to these trunk nodes. The length of the line is scaled about 1.2 times the pine tree's crown height.



(a) Flow direction depends on root attractor and nearest trunk point positions.

(b) Root attractor point S_r and nearest trunk point S_t.

(c) Particle flow ends when a particle merges with a neighbor or with the trunk.

Figure 4.7: A simple particle flow results in crown branching structure.

Trunk nodes contain a root attractor point and a nearest trunk point, as shown in Figure 4.7. The root attractor point, shown in green, is the trunk node closest to the lower bound of the bounding box for the entire crown. The nearest trunk point, which is shown in blue for the red particle in Figure 4.7a, is the closest trunk node to a single particle.

Particles move under directions of three forces: gravity, shape-format, and dominant wind. In Figure 4.8, we describe the direction and magnitude of each force. Gravity points vertically down to the ground. We assume that a particle has higher mass when it is closer to the trunk. This assumption follows the observation that when closer to the trunk, a branch has a larger radius. For a particle with higher mass, it has a larger magnitude of gravity and moves faster in the vertical downwards direction. Under this assumption, we set the magnitude of gravity proportional to the distance between a particle position S_p and nearest trunk point S_t.

We call the second force shape-format. Arborists distinguish styles of growth habit of trees using different classifications, such as excurrent and decurrent. The force tries to guide particle flow to follow the growth pattern of the original tree. Simulating different growth patterns requires different definitions of the shape-format force. In this research, we provide a simple example of shape-format definition. The shape-format force, shown in Figure 4.8,

Force	Direction	Magnitude
Gravity	(0,-1,0)	$\lvert S_p - S_t \rvert$
Shape-format	$\overline{S_p - S_r}$	$\alpha\ ((S_{py}^1 + \cdots + S_{py}^n)/n)$
Wind	F_w	$\lvert F_w \rvert$

Figure 4.8: Forces: gravity, shape-format, and dominate wind. S_p: position of a particle. S_{py} is position of a particle in y direction. S_t: position of the nearest trunk point. S_r: position of the root attractor point. F_w: wind measured from motion capture setup. α: scaling weight in range of [0, 1]. n: number of particles.

guides the particle flow process by height and depth of a tree crown. The direction of the force is pointing to the root attractor point S_r from particle location S_p. The magnitude of the force is the average height of all the particles with a weight parameter α. The higher the center of all the particles, the stronger the shape-format force points to the root attractor point S_r. This force is a pre-computed global force, which is a constant for all the particles. The direction of the force ensures that a particle finally merges to trunk nodes, and therefore all the branches grow towards the trunk.

The dominant wind direction is recorded from the motion capture setup, as described in Section 4.3. Wind force is a special case of external force acting on the tree's branching shapes and structures. While doing motion capture, we record the location of the electrical fan. Because we only use one fan to create wind, that is the only source of explicit external force. Alternatively, the dominant wind direction can be inferred from tree movements recorded in motion capture data. Statistic methods, such as PCA , can estimate the dominate wind direction.

The flow of particles starts at branch tips. Some particles merge in the flow process while others eventually reach and merge to the trunk. At each time step, we compute for step size and direction of a particle. The step size L is:

$$L = \beta * ||S_p, S_t||, \tag{4.1}$$

where β is a tunable parameter in range of [0.1, 0.5] for most of our tree models. The step direction combines all the three forces. Figure 4.7b shows the computation of the direction for particle i, which is shown as particle A in Figure 4.7b. Each force has a weighting parameter ω that tunes the relative importance of each vector and provides flexibility in creating branching shapes. The particle step direction $\vec{D_i}$ is given by

$$\vec{D_i} = (\omega_1 * \vec{G_i} + \omega_2 * \vec{S_i} + \omega_3 * \vec{W_i})/3, \tag{4.2}$$

where ω is the weight of the direction, $\vec{G_i}$ is direction of gravity, $\vec{S_i}$ is direction of shape-format, and $\vec{W_i}$ is the wind direction for particle i where $i \in [1, ..., n]$.

Using step size L and direction $\vec{D}$, the new particle position $V(m)$ is given by

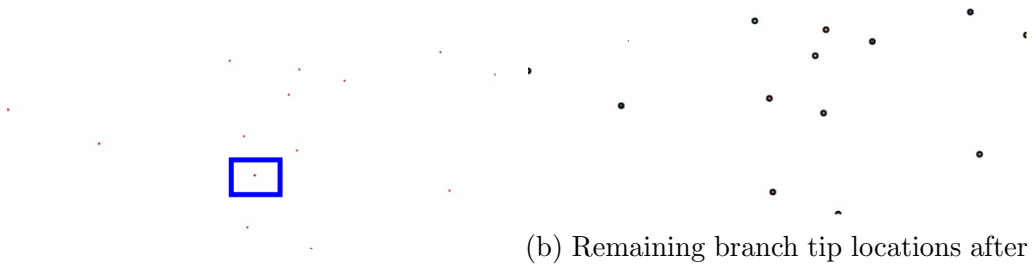
$$V(m) = V(m-1) + L * \vec{D_i}, \tag{4.3}$$

where m is current time step and $V(m-1)$ is the particle position at the last time step.

At each time step, after updating all the particle positions, particles might be merged. When the distance between a pair of particles is less than a predefined merging threshold, those particles are combined. When the distance between a particle and a trunk point falls below a predefined merging threshold, that particle is merged with the trunk. In Figure 4.7c, we demonstrate the paths of particle flow. Particle A, B, and C are moved using the step and direction. Particle A and B are merged at point AB and finally merged to the trunk. Particle C merged to the trunk point after two time steps of movement.

4.6 Adding Leaves

Tree leaves are visually important to 3D tree models. After the branching structure is generated, leaves are attached. We use predefined leaf shapes, growth patterns, densities, and sizes. Also, because leaves do not always start growing at the beginning of a branch, we



(a) Original collected data.

(b) Remaining branch tip locations after clustering.

Figure 4.9: Clustering that removes noise that leads to spurious marker positions.

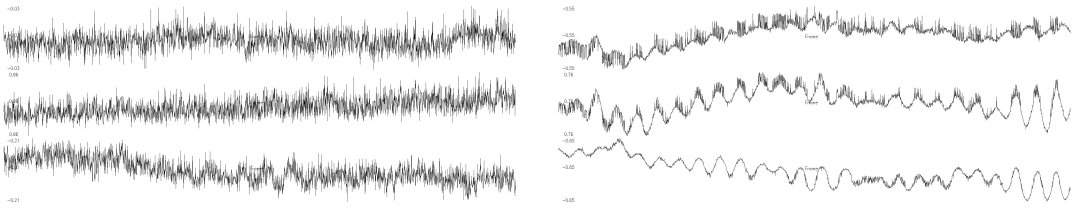
set a parameter called the leaf starting point. This parameter is proportional to a branch length. For example, when the parameter is 0.3, it starts leaves after the point that is 0.3 times the branch length away from the branch starting point.

4.7 Results

Results are given for multiple trees, including maple and pine trees. The maple tree is instrumented with 24 markers placed on the periphery of the crown at branch tips and the pine tree is instrumented with 35 markers also located on branch tips. We collect the stationary locations of these markers for about 20 seconds at the capture rate of 100 frames/sec.

Figure 4.9a shows all the recorded locations as red dots for a single frame, and Figure 4.9b shows averaged locations from each cluster of marker positions for clusters in which the number of frames with a position in that cluster is 1/4 of the total frame count. The data is recorded when the tree is stationary. Notice that a point in the blue box in Figure 4.9a is identified as noise and removed by the clustering algorithm. For the maple tree, after clustering and averaging only 24 markers remain and this matches the actual number of markers placed on the tree.

Although initial marker positions are collected before wind is applied when the tree is stationary, the data contain a small amount of noise, which can be removed to create a single initial marker position. In Figure 4.10, we show marker positions over time for two



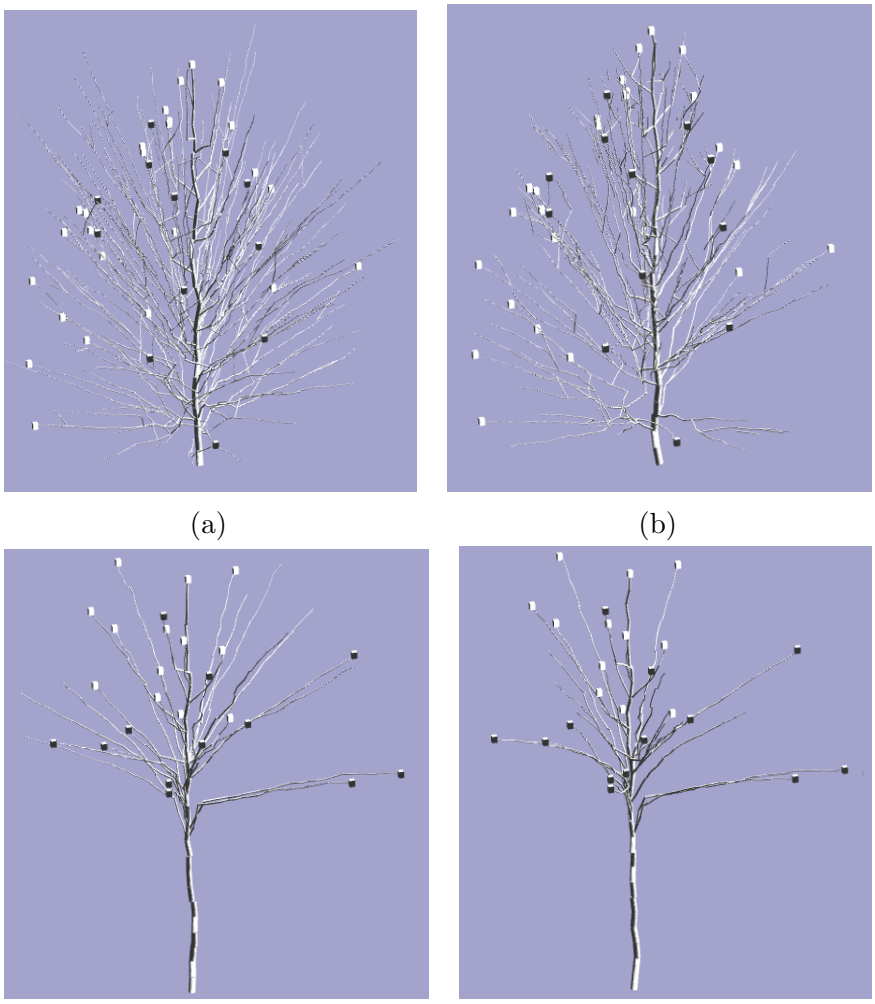
(a) Close to white noise. (b) Noise with apparent periods.

Figure 4.10: Marker positions over time contain a small amount of spurious motion that is removed by averaging.

markers during the stationary phase. The horizontal axis shows time while the vertical axis shows a marker location in 3D space. In the first image, movement in each direction spans 10^{-3} meters. In addition, in the second image there is apparently a small periodic motion for the stationary branch tip. The range of motion is also within 10^{-3} meters. In both cases, averaging removes this small motion and estimates initial marker positions based on the average rather than a single position in a single frame.

A vertical stack of bounding boxes is a better approximation for the crown volume than a single bounding box and results in better crown shapes. Each bounding box contains branch tips and additional branch tips are added to each box. Figure 4.11a and Figure 4.11b illustrate the difference between tree crowns created with and without a stack of bounding boxes for a pine tree. Random particles placed uniformly within a single bounding box result in a cube shaped tree. Placing particles in a vertical stack of bounding boxes better approximates the original crown shape. Future work might include investigating non-uniform distributions of randomly inserted points instead of using bounding boxes.

In Figure 4.11c and Figure 4.11d, we demonstrate the difference between tree shapes using stacked box bounding volumes and those using convex hull bounding volumes. The bounding box approach provides a looser bounding condition and allows more random factors in the final tree shape. The convex hull approach more closely approximates the original tree crown shape.



(a) (b)

(c) (d)

Figure 4.11: Bounding volumes affect tree shape.

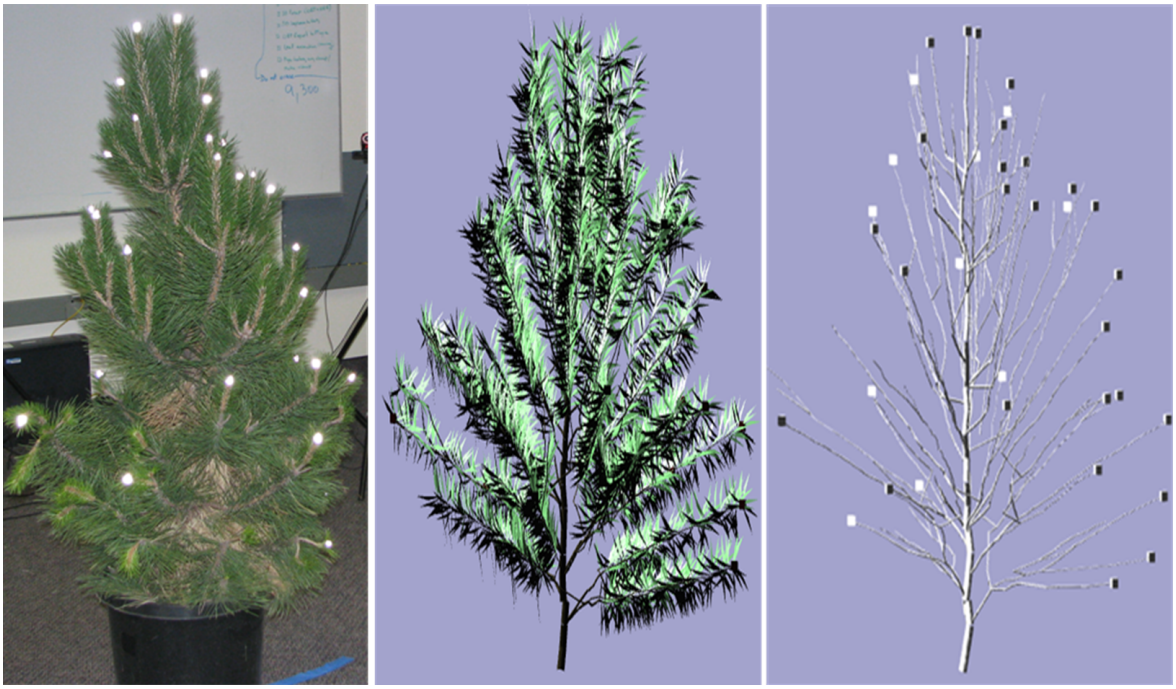


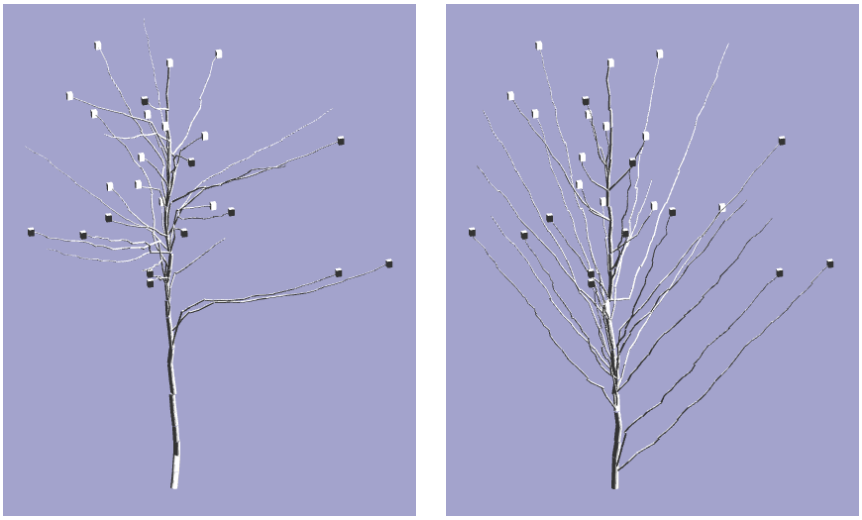
Figure 4.12: Pine tree model. 3D tree models created from branch tip positions are similar to the shapes of the trees from which branch tip positions were collected.

Particle flow starting from branch tips and using our simplified algorithm results in tree crown shapes that mimic the shape of the tree from which data were collected. In the Figure 4.12, we show the results from reconstructing a pine tree.

Besides replaying the original tree shape, our approach has enough flexibility to produce different tree shapes based on the same set of motion capture data. Three forces with their scales create particles' moving paths, which represents branching structures. Figure 4.13 demonstrates that shape-format force sets the global attracting trunk node, which controls the converging direction of particle flow. The resulting tree models display trees' growth styles in terms of excurrent and decurrent.

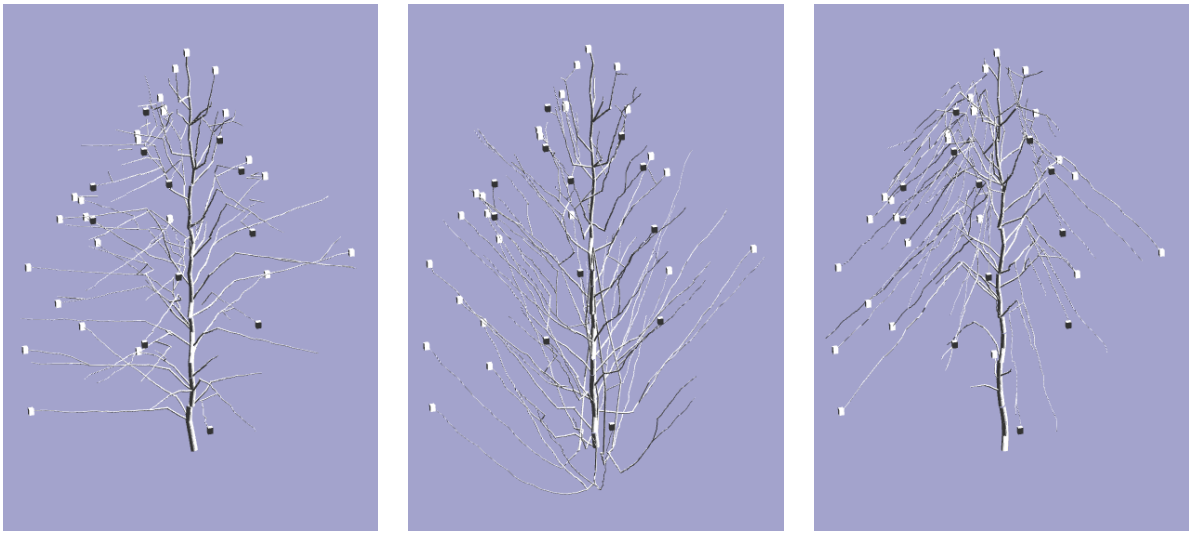
Figure 4.14 shows the impact of gravity on trees' shapes with various values for the weighting factor. When the factor is set to be negative, we create a special willow-like tree shape.

Figure 4.15 shows tree models with different weighting factors of wind force. Bigger values of the factor produces branches bending more toward the wind direction. Notice that



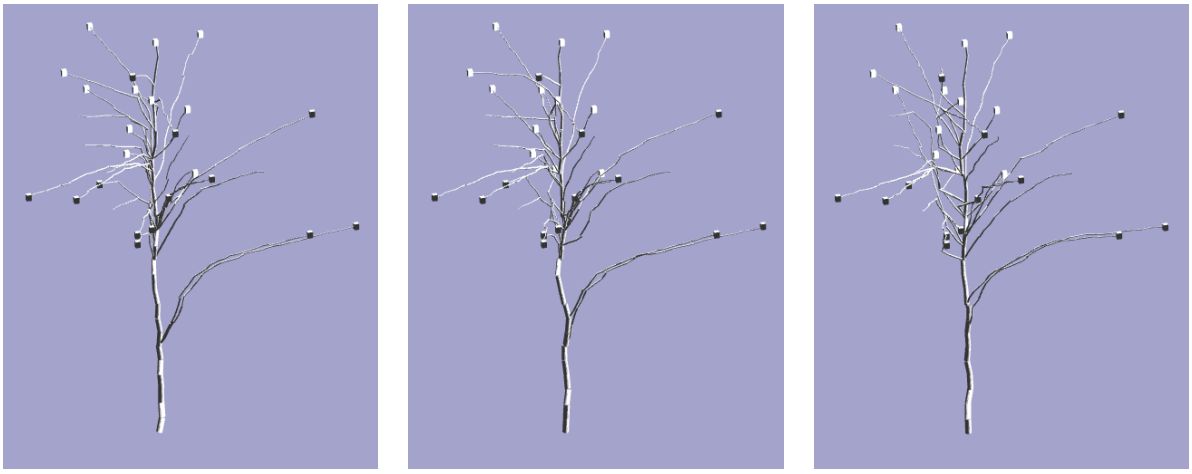
(a) (b)

Figure 4.13: Shape-format force.



(a) weighting factor: 0.01 (b) weighting factor: 0.20 (c) weighting factor: -0.20

Figure 4.14: Tree shapes with different weight factors for gravity.



(a) Weighting factor: 0.01. (b) Weighting factor: 0.07. (c) Weighting factor: 0.15.

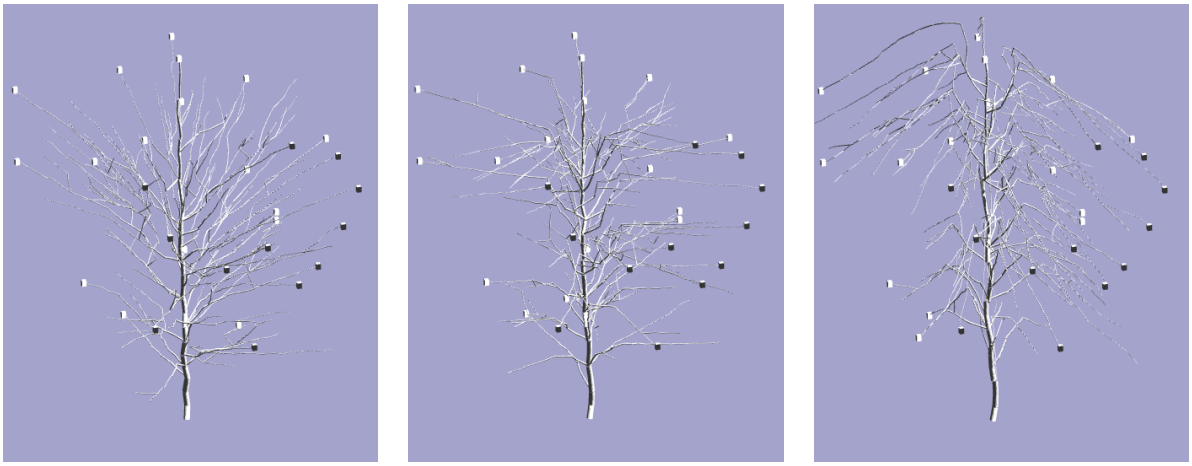
Figure 4.15: Tree shapes with different weighting factors for wind force.

the differences among these tree models are small, especially when compared to the influence from the other two forces. This is because the value of force for a global wind direction is set to be much smaller than that of the other two forces. Otherwise, when wind force dominates the direction of particle movement, the particles might not be able to merge to a tree's trunk and might violate natural tree's shape.

In Figure 4.16, we generate 3D tree models with different parameters for particle flow with the same set of motion capture data. These results demonstrate that our approach produces visually plausible tree shapes, which becomes scalable for more extended shapes through tweaking parameters of the three forces.

4.8 Discussion and Future Work

Placing retroreflective markers on branch tips evenly spaced throughout the crown on trees located in a passive optical motion capture arena results in data that can be used to reconstruct tree shape and which may be usable for replaying branch motion. This can be done using a simplified particle flow system starting from recorded branch tip positions supplemented with additional random branch tip positions within a horizontal stack of bounding boxes and by



(a) (b) (c)

Figure 4.16: Various tree shapes.

setting two control parameters. A new data collection process designed for trees may extend the use of motion capture to include trees and eventually other networks of non-rigid bodies.

Future work includes extending the process to large trees outdoors as well as improving methods for animating the resulting tree models using the motion capture data. We have reconstructed an approximate tree crown branching structure. Replaying the captured motion data will require care to ensure that the motion of the approximate branching structure does not include uncorrelated motion for branch tips who share a common parent.

Chapter 5

A Realistic 3D Tree Model Based on L-Systems

Jie Long and Michael Jones. *A Realistic 3D Tree Model based on L-Systems. Report for UNEP Eco-Peace Leadership Center (EPLC)*, 2008.

Abstract

Constructing a 3D tree model manually is time consuming due to the natural complexity of tree shapes. We introduce a new morphology-based method using L-systems for realistic 3D tree modeling. Using L-systems for describing tree branches as particles, this method (1) introduces a hemisphere to generate particles, (2) uses a growth level to simulate different ages of branches, and (3) applies a dynamic bounding box to detect local growth area in a tree. This new method enhances the management of tree shapes by easing the control over distributions of branches and leaves. To further validate the method, we demonstrate that the method can simulate photorealism and growth around physical barriers. We evaluate this model by particle flow and by complexity, showing performance competitive with existing methods.

5.1 Introduction

The natural complexity of trees has been challenging computer graphics for decades. Many applications—film, 3D video games, city planning, and forestry—require clear, detailed 3D tree models. Although methods exist for creating photorealistic tree models, these methods are still cumbersome. Efficient methods for constructing 3D tree models are needed to deal with the intractable computation of the detailed geometry. Since trees have many properties due to the growth environment and kinds, models with global controls over shapes are also important.

In recent years, techniques on image-based reconstruction and L-systems have been widely used in 3D tree modeling research. Both methods have advantages and shortcomings. Image-based reconstruction generates natural-looking 3D trees through image processing, but models are limited to trees in the image and most need time-consuming manual modifications. Although L-systems are efficient and easily implement in 2D or 3D tree models, using L-systems representation alone makes it difficult to control small components in a tree (e.g., a certain twig).

In this paper, we introduce a new method for 3D tree modeling based on L-systems. This new method presents a 3D tree model with three innovations: a hemisphere, level controls, and a dynamic bounding box. First we build a branch library using L-systems. A branch unit, which is also a unit of L-systems and works as a particle, has several twigs. A hemisphere on the top of a tree model controls the distribution of branches and leaves. The growth levels simulate different ages of branches and leaves. A dynamic bounding box constrains the local growth area in a tree by avoiding outer forces. In the implementation, particles are generated on the hemisphere surface and begin to move in the hemisphere with different starting angles. A ray defined by the position and angle of a particle is attracted to the nearest branch in the existing tree. After a new branch attaches to the existing tree, we enlarge the bounding box to the new tree shape. This bounding box grows with the tree volume and can detect collisions with other objects or growth obstacles. In each growth

interval, a certain number of branches are generated and distributed. In addition, leaves have initial parameters for shapes, sizes and colors. We also distribute leaves from the hemisphere surface. In each growth stage, leaves have different sizes and colors, but the same shape.

There are four steps in this 3D tree modeling. First, L-systems generate different branch patterns. Second, a hemisphere designates probability distributions for generating particles. Third, a branch from the branch library is randomly selected, a particle is generated on the designated area, the ray to the tree model is computed, and the nearest growth point is found. Finally, after constructing branches for the whole tree, leaves with different sizes and colors are added to this model.

Using this new method, a 3D tree model is more efficient and controllable. We take advantage of L-systems to describe branches for efficiency while overcoming the control scale problem. L-systems and image-based methods control the whole tree at one time. They generate tree models as a whole. However, our method manages small components of a branch, but not the smallest components of twigs. The second advantage of this new method is the ease of control over tree shape. The hemisphere controls distribution probabilities to shape a tree. The bounding box flexibly constrains a proper growth area to detect outer barriers like rocks and buildings. The parameters of a desired shape are far less than existing methods. In addition, our method can distribute leaves with the same shapes but with flexible sizes and colors due to the growth level of branches. Current methods including L-systems and image-based approaches can't manage the age-based distributions of leaves. Also, we carried out a set of experiments to validate this new method: photokinesis simulation and growth around physical barriers.

5.2 Related Work

Two main research directions in tree modeling are biology-based and morphology-based. Models that mimic biological data are based primarily on patterns of tree growth. Some tree modeling software, such as AMAP, COSSYM, and SVS, simulate tree growth based

directly on biological data. Morphology-based methods focus on reconstructing tree shapes from photographs or remote sensing (RS) images. Both biology-based and morphology-based methods have applications in different areas. Forestry research uses biology-based models, and 3D games and movies use morphology-based models. Our system is mainly morphology-based while also considering some biology characteristics of natural trees. In this section, we discuss two primary morphology-based tree modeling methods: L-systems and image-based models.

5.2.1 L-systems

An L-system is a formal grammar that describes the recursive growth of a tree. The rules of the grammar must be written by the user. Since the rules are applied locally, small changes in the rules may cause large changes in the overall tree shape. Such behavior makes modeling quite difficult. Various extensions of L-systems have been proposed, including parametric [41], open [28], and differential L-systems [42]. These extensions are able to create a variety of effects, but also require additional parameters from the user. Prusinkiewicz et al. [43] present a modeling interface for L-systems to enhance the modeling ease, but a large set of parameters still has to be defined by the user.

L-systems have both advantages and shortcomings. L-systems generate 2D and 3D tree models efficiently. This method is easy to implement. However, L-systems generate tree models at one time after defining an algorithm. Further modification of models is difficult. L-systems go too far in simplifying tree models, so the results are not realistic.

5.2.2 Image Reconstructions

Reconstruction of tree shapes from photographs is an active area of research in 3D tree modeling. 3D tree models from this method look natural because they are based on the morphology of actual trees. People can use 2D source images, which are easily collected using consumer digital cameras, to generate 3D tree models for almost any interesting trees.

Shlyakhter et al. [53] direct the growth of L-systems using photographs. The registered input images reconstruct a visual hull. The medial axis diagram of the hull constructs the tree skeleton. L-systems describe smaller branches and leaves.

Reche Martinez et al. [45] describe a very precise, though complex, image-based approach. In this case a set of carefully registered photographs determines the volumetric shape of a given tree. The volume is divided into cells; for each cell a set of textures compute a valid visual representation. The complete set of textures represents the tree quite faithfully.

Neubert et al. [29] present a method to produce 3D tree models from 2D photographs using particle flows. Using image information, the author estimates an approximate voxel-based tree volume. Performing a 3D flow simulation, particles form the twigs and branches. The botanical rules for branch thicknesses and branching angles produce the geometry of the tree skeleton.

Tan et al. [58] propose an approach for generating 3D tree models from images. This method requires little user intervention. This research populates the tree with leaf replicas from segmented source images to reconstruct the overall tree shape. In addition, shape patterns of visible branches can predict those of obscured branches.

5.2.3 Other Methods

There are some other famous approaches for tree modeling. Aono et al. [2] presented the A-system. Oppenheimer [31] proposed the animation based on a fractal method. Reeves [47] introduced a modeling method based on particle flow. Reffye et al. [7, 66] presented a model based on botanical structures. Weber et al. [61] presented a method on generating trees in several steps. Kurth's team [17] developed LIGNUM [36] for 3D tree modeling.

5.3 Tree Modeling Using L-systems

Our method has four main parts: a branch library on L-systems, a hemisphere, growth level controls, and a dynamical bounding box. L-systems are easy for describing, controlling, and

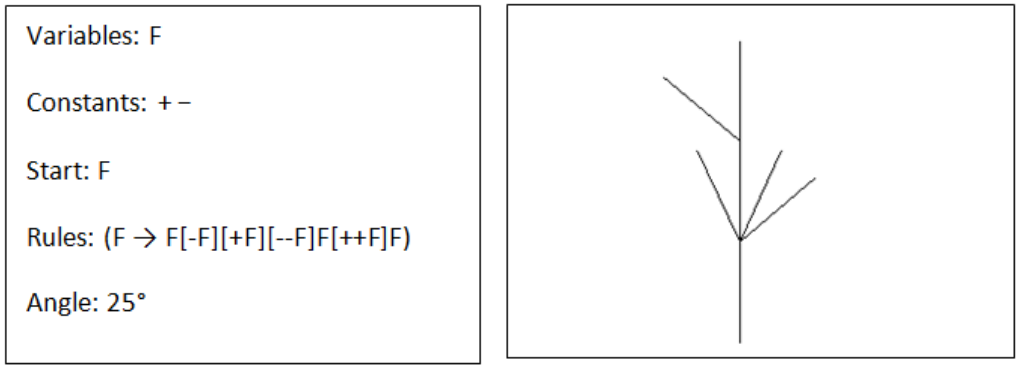
implementing branch patterns. We can define iterations for each pattern. Users who know L-systems can also define their own branch types easily. The hemisphere controls distribution probabilities for branches and leaves, so we can control tree shapes with a proper randomness. Different growth levels of the tree provide corresponding parameters for both branches and leaves. The dynamical bounding box constrains the local growth area and detects outer growth barriers. In a tree model, both local and global characteristics of a tree are considered. Randomly selected L-systems branches control the local shape for every branch. However, we use the probability hemisphere to control the global shape of a tree.

These four parts work together to produce many tree shapes with efficiency and a natural look. Distribution probabilities for branches and leaves are designated by the hemisphere. This process enables simulations referred to as probability distributions, such as the photokinesis simulation. The bounding box controls local tree shape by detecting outer factors and enables blocked growth. The growth level manages different growth stages for the tree model to decide growth parameters for branches and leaves.

5.3.1 Branch Library on L-systems

L-systems define different branch shapes in our research. Constructing branch shapes is a key problem in simulating 3D trees. Different L-systems algorithms result in different branch patterns. For each branch pattern, we define its rules, angle, and number of iterations. In Figure 5.1, we define a branch pattern using a one-iteration L-system (5.1a) and show the corresponding branch shape (5.1b):

After giving an L-system for each branch pattern, we use an OpenGL library to draw the corresponding branch. Each fragment (see lines in Figure 5.1b) of a branch is described by a 3D cylinder. We then apply an angle at the joints to rotate these cylinders. Then we get a 3D branch based on L-systems algorithms. All 3D branches in the branch library have the same length and radius for every twig. When applying a branch to a tree model, we compute the length and radius by parameters from the growth point.



(a) L-systems algorithms. (b) The corresponding branch.

Figure 5.1: Tree structure under L-systems.

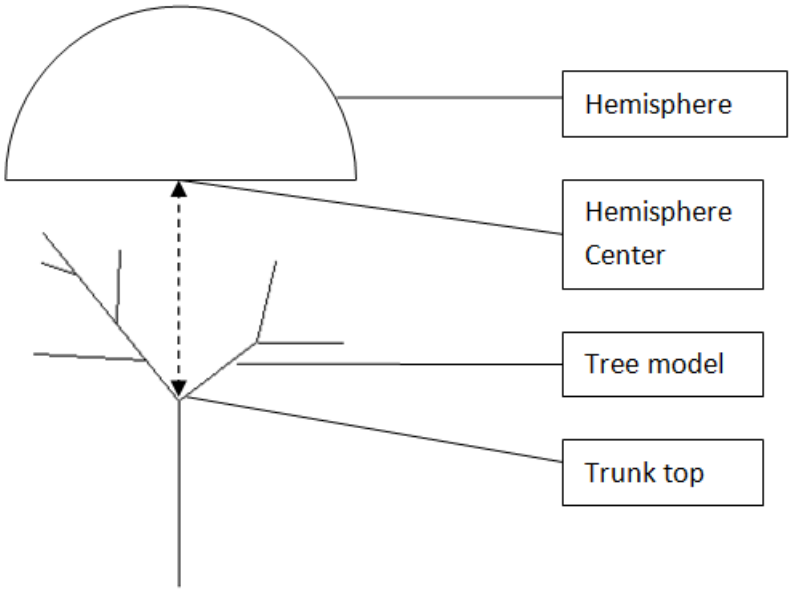


Figure 5.2: Side view of a 3D tree model and hemisphere.

5.3.2 Hemisphere for Probabilities Distribution

A hemisphere controls the distribution of branches and leaves. We define it by a position, a radius, and a probability distribution. The position of this hemisphere is on the top area of a tree model. Its diameter is set by users for different purposes. For example, simulating photokinesis requires the hemisphere to be big enough to include the track of sun movement. The probability distribution divides the hemisphere into several parts and assigns probabilities of particle generation to each part.

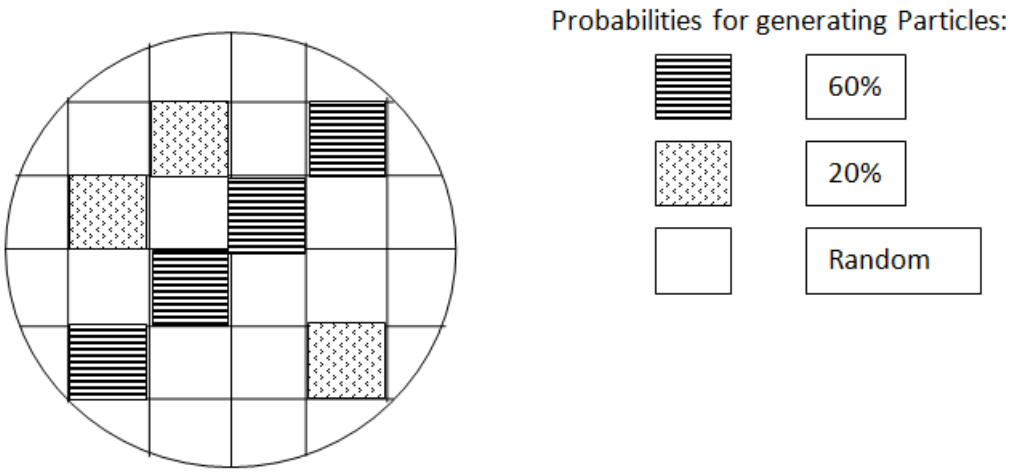


Figure 5.3: An example of the probability distribution of a hemisphere (up view).

In Figure 5.2, we show the definition of a hemisphere. The size of the hemisphere is bigger than the tree crown. The position is right next to the tree model and the center of the hemisphere is perpendicular to the top of the trunk.

In Figure 5.3, we show an example of hemisphere division and a probability distribution. We cut the hemisphere into several parts. Each part has a specific probability of generating particles. These probabilities control the shape of the 3D tree model.

The functions of the hemisphere have three parts: arranging probabilities for generating particles, defining directions for particles, and setting initial positions for particles. As in Figure 5.3, these probabilities decide how particles generate in different parts. In this example, particles are mainly generated in the shaded areas. If the shaded area is the track of the sun's movement, branches in a tree mainly grow towards these areas. When generating a particle on the hemisphere, we first decide the initial position of this particle according to the probability of its region. We then give an initial direction for this particle to move in the 3D scene.

After initializing a particle and its movement in the scene, we define a landing constraint for the particle. The constraint is defined by the user. For example, we can use the nearest Cartesian distance as the constraint. The distance is from the particle's position on the hemisphere to branch top points.

After using the landing constraint to define the landing point on existing tree branches, we select a branch pattern from the L-systems branch library. Using the growth parameters of the landing point, we are able to attach the new branch to the existing tree.

5.3.3 Bounding Box for Local Growth Control

A bounding box defines the minimal 3D rectangular volume around the existing tree model. The bounding box computes the current growth space. When the tree model grows larger, the size of the bounding box grows simultaneously. Therefore, the bounding box can detect whether there are intersections between the bounding box of the tree and other objects. For example, after adding a new branch, if the tree's bounding box intersects with a rock's bounding box, we can delete this branch and generate the next branch.

The bounding box is an approximate method for defining the tree's growth area. In fact, the most exact way is to calculate all points on the tree body. However, this approach is very expensive and not necessary. In contrast, using a bounding box we only need to calculate for two points and detect the growth area approximately. Since a tree has many branches, such approximation highly reduces computations while properly detecting outer barriers.

5.3.4 Growth Level Controls

We introduce the level of growth to control the growth of branches and leaves. In Figure 4, the tree model has three growth levels. Figure 5 shows the corresponding tree model for the growth levels in Figure 4. Because natural trees grow new branches every year, the leaf sizes and colors change according to branch ages and different parts of trees. For example, the color of younger branches and leaves is light green while older ones are dark green or brown. The growth level parameter tracks the age of branches and is used to define leaf and bark appearance. When we add new branches or leaves, the age parameter indicates the right colors, sizes, and other parameters for them.

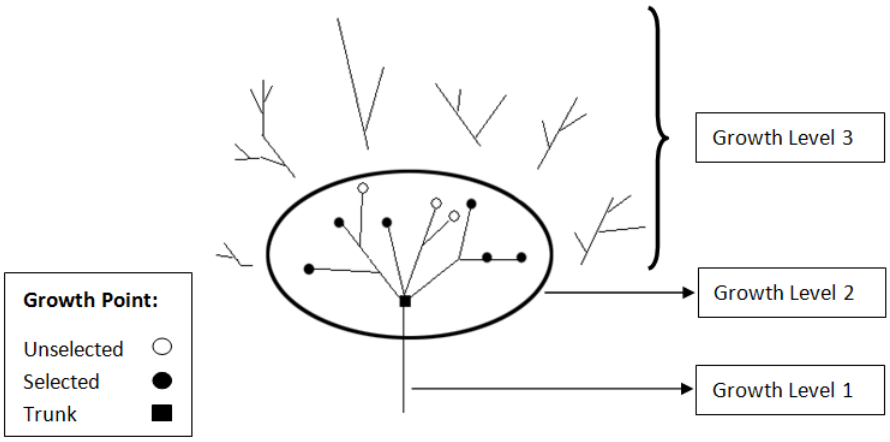


Figure 5.4: Growth level 1, growth level 2, and growth level 3 in a tree model.

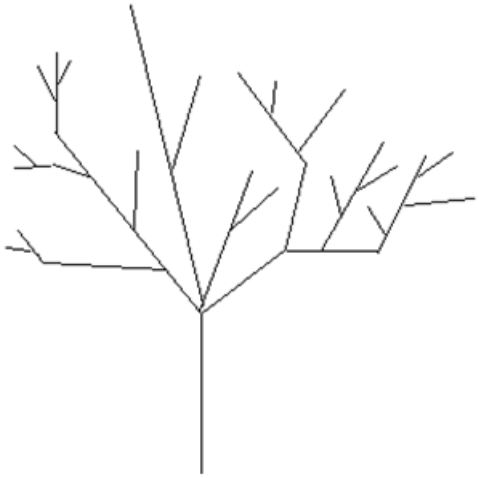


Figure 5.5: A tree model with three growth levels.

Growth levels set different particle generations. In Figure 5.4, the first generation of particles is the trunk, the oldest part in a tree, having a growth number 1. Then the second generation, branches growing on trunks, has a growth number 2. After adding growth level 3, we get a tree model as shown in Figure 5.5.

Leaves grown in these parts should have darker colors and bigger sizes than those on top, which has a higher growth number. Therefore, different growth levels can simulate different ages of branches and work as a reference for leaves.

5.3.5 Tree Generation Steps

Based on the discussed four parts—L-systems, a hemisphere, a dynamical bounding box, and growth level controls—we can generate a 3D tree model. There are three main steps in this process: (1) define the hemisphere and the branch library, (2) generate particles and control their movements, and (3) distribute the particles in a tree model. We discuss these steps below.

(1) Define a hemisphere and a branch library.

We define the hemisphere's initial position, size, and probability distribution. The position and size is decided by the scene and the tree size while the center of the hemisphere is identical to the top of the trunk. The probability distribution is defined by users for different purposes. This distribution also reflects the projection of 3D branches on the hemisphere. Therefore, the probability distribution can control the shape of the tree.

As for the branch library based on L-systems, we define different L-systems algorithms and get different patterns of branch shapes.

(2) Generate particles and control their movements.

We generate a certain number of particles for a growth level at the same time. For example, 10 particles for growth level 2. Then we select proper positions for these particles on the tree.

A particle denotes a branch selected from the branch library. The particle is generated on the surface of the hemisphere according to the probability distribution. With an initial position on the hemisphere and an initial orientation, the particle moves in a Brownian way in the scene.

(3) Distribute the particles in a tree model.

During the Brownian movement, the particle compares and finds a nearest distance among all growth points. Then the branch denoted by this particle is attached to this nearest growth point. After the particle settles, we calculate a new bounding box to replace the old one for this tree. However, there is one exception. After attaching the new branch to the

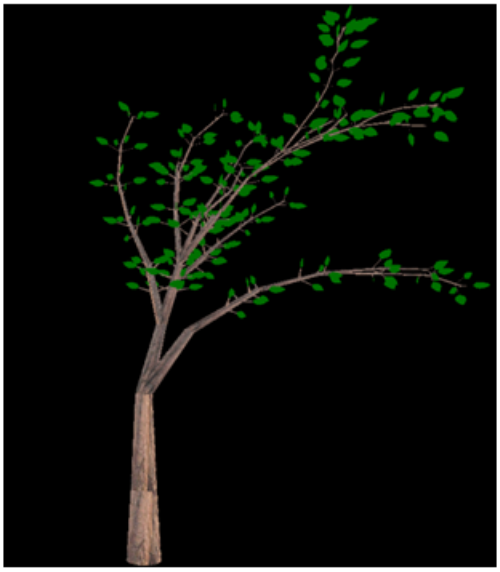


Figure 5.6: A result for photokinesis simulation.

existing tree model, if the new bounding box overlaps some outer blocks, the particle with this branch is cancelled and we then go to the next particle generation.

After one particle is generated and distributed in a tree model, we repeat step (2) and step (3) until we reach a user-defined numbers of particles.

5.3.6 Examples of 3D Tree Modeling

Based on this new method with L-systems, we can add more constraints to this model to simulate some properties of natural trees. Here we introduce implementations of trees' photokinesis simulations and of growth control over blocks.

Trees' Photokinesis Simulations

The photokinesis simulation relies on the probability distribution on the hemisphere. This simulation is used to make the tree branches grow towards the sun. In order to give more weight to the lighting area, we add more probability of generating tree branches in the track of sun movement. Then the hemisphere works as the sky and the probability distribution is assigned by the sun's track. Figure 5.6 shows one result of photokinesis simulation.

Growth Control Over Blocks

We control the growth over blocks by using the dynamic bounding box. When simulating 3D tree growth over blocks, the tree should avoid growing in the blocked area. We use bounding boxes to control blocks, such as outer buildings or rocks. First, we calculate bounding boxes of the blocks. During steps to distribute branches, when a tree's bounding box goes into the bounding boxes of blocks, the current particle is deleted. Thus we can eliminate tree branches grown in the blocked areas.

Trees' Fruits Simulation

We can add fruits on a tree using this method. The process of fruits simulation is similar to the leaf attachment. We define the size, shape, and color for fruits. Then we generate particles on the probability hemisphere and distribute the fruits to different growth levels according to the tree model.

5.4 Results

This new method works well and has some advantages. This model controls the local shape of branches using L-systems and the global shape of trees using the probability hemisphere. Different growth levels approximately simulate the growth process. The bounding box detects outer obstacles. We analyze these characters in detail below.

5.4.1 Growth Probability Control by Hemisphere

This method introduces a hemisphere control over tree growth by distributing particle probabilities. The tree shape is decided by the predefined probabilities on the surface of this hemisphere. As we use a particle system to control the distributions of branches, the particles' initial sizes and orientations are defined on this hemisphere. Thus, the main shape of a tree is decided and we can use different types of branch patterns to generate the details. Figure 5.7

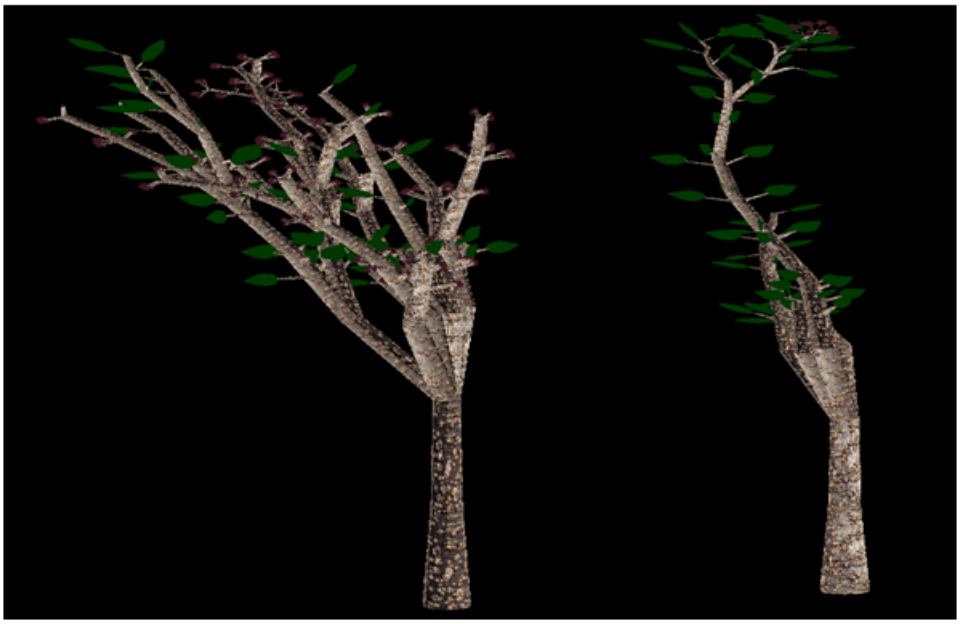


Figure 5.7: (a) Branches towards the same direction; (b) branches clustered together for one direction.

shows two types of hemisphere control on tree shape. In Figure 5.7(a), all branches are grown towards the same direction. In Figure 5.7(b), there is only one priority growth direction.

5.4.2 Growth Level Controls

We introduce a new conception of growth level controls. In this control, we give different growth levels for different branches, leaves, and even fruits with different ages. Branches, leaves, and fruits with the same growth level can share some parameters, such as sizes, colors, and shapes. In Figure 5.8, we show two results for the growth level work. In Figure 5.8(a), this tree model has needle-shaped leaves and heart-shaped leaves. It also has small red flowers on the top level. In Figure 5.8(b), we add some fruits to this tree model.

5.4.3 Growth Control over Small Components

This new method has advantages in controlling small components in a tree. Compared to the DLA (Diffusion-Limited Aggregation) method [15, 59] for tree modeling, our method reduces computing work by reducing the particles. Using a branch type as a particle rather than

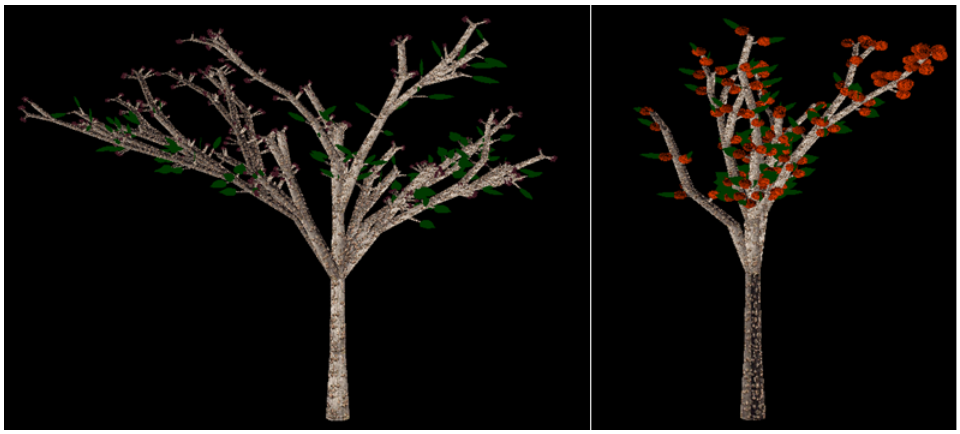


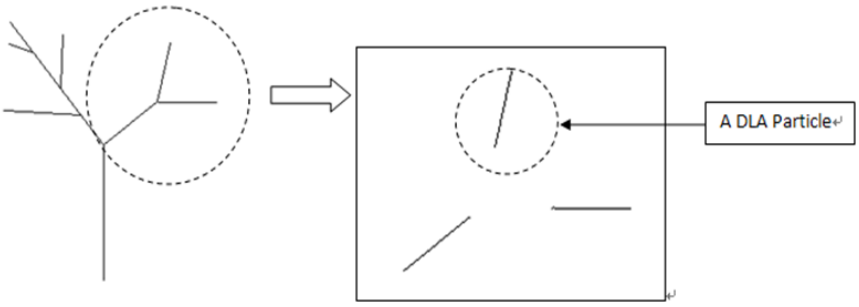
Figure 5.8: (a) A 3D tree model with different leaf shapes and flowers; (b) A 3D tree model with fruits.

assigning a DLA particle for every twig, fewer particles are needed for the same number of twigs in a tree. Since computation for each DLA particle is the same, fewer DLA particles requires less computation. In Figure 5.9, we have the same tree shape. If we use traditional DLA in Figure 5.9a, the circled branch needs three particles to calculate. But in Figure 5.9b, we calculate one DLA particle for three twigs in one branch unit.

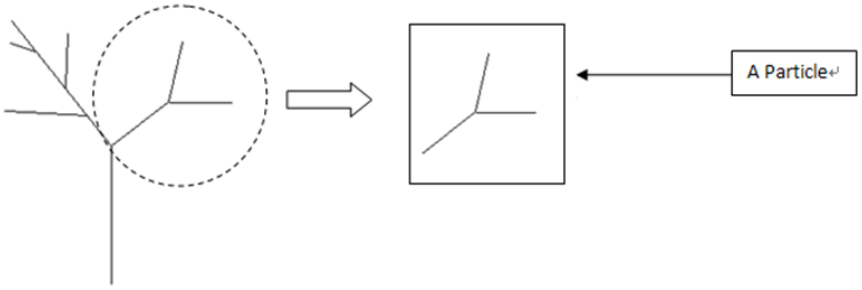
5.4.4 Growth Control for Leaves

Our method provides a new method of leaf simulation, which is difficult for most current methods. Leaf simulation is difficult to achieve by L-systems because L-systems can't control the positions of leaves randomly on branches. Even the popular image-based approaches have difficulties handling leaf simulation. These approaches usually generate branches very well. However, after branch construction, leaves are added randomly and often can't attach to the branches. Our method uses the hemisphere to handle the overall distribution of leaves in a tree, and we apply growth level controls to define leaves with different shapes, sizes, and colors according to the branch properties to which they attach.

This approach is also effective. We only need to change leaf parameters according to the numbers of total growth levels. At a certain growth level, we define the shapes, sizes, and colors for leaves in that level. As for the whole tree, branches have leaves of the same



(a) DLA particles.



(b) Particles in our method.

Figure 5.9: Particles with tree shape.

age. In the natural world, that means the old branches have old and big leaves while young branches have young and small leaves in the same shapes.

5.4.5 Growth Control by Bounding Box

Using dynamic bounding boxes in detecting proper positions for DLA particles gives flexible controls over 3D tree models. We generate the DLA particle in the bounding box and find a position in this bounding box. By controlling this bounding box, we can control the growth of the tree. For example, if the current bounding box overlaps a building's bounding box, the current DLA particle should be canceled.

Figure 5.10 shows an example of tree growth. In this example, the 3D tree model tries to avoid the white box to grow. This process is similar to the process of a tree growing to avoid outer barriers such as bridges or buildings.

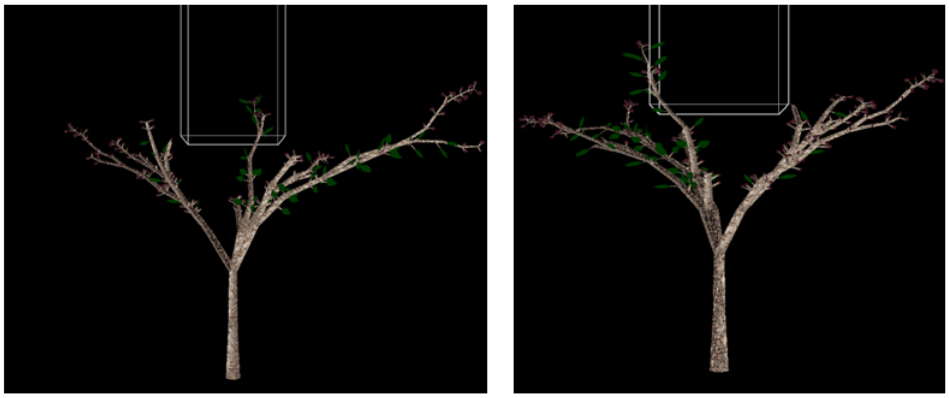


Figure 5.10: Trees grow to avoid the white outer box viewed from two different directions.

Figure 5.11: (a) Combination method tree; (b) traditional L-systems tree.

5.4.6 Randomness

Compared to traditional L-systems, our method generates tree models with more random shapes. Every branch unit carries an L-system algorithm. Because particles control every branch unit, the whole tree doesn't follow any L-systems algorithms. This approach solves artificial iterations from single pure L-system algorithms. In Figure 5.11, (a) is a tree model from the combination method and (b) is from a traditional L-systems algorithm. Figure 5.11(a) has a random look while (b) has a self-similar character that makes the tree model artificial.

Figure 5.12: (a) Combination method tree; (b) traditional DLA tree.

Randomness from the DLA method goes too far for tree modeling. As shown in Figure 5.12, the shape from (b) might be more curved than is common in trees. Using the combination method, we can reduce the curvature by reducing the number of particles, which stand for L-systems branch units rather than twigs.

5.5 Conclusions and Discussions

We introduce a new method based on L-systems for 3D tree models. In this method, a hemisphere controls probability distributions for branches, leaves, and fruits; growth level controls the distributions for different ages of branches; and a bounding box detects the outer collisions. This new method employs a moving particle for a branch unit, which reduces computation costs of traditional methods like DLA. The hemisphere controls the probability distributions to simulate some natural properties of trees or some special effects. The growth level can manage the internal growth in a tree through age controls. In the tree construction process, the bounding provides an easy and efficient way to control tree growth. Therefore, we can manage the main shapes using the hemisphere while controlling the internal growth

through growth level controls. Under these controls, randomness is added by randomly selected tree branches from the L-systems branch library.

The future work in this research may focus on adding more controls over particle movements. We can try to find some more efficient approaches for distributing L-systems branch units. We can also simulate tree animations using this model. Furthermore, we can apply forestry equations to make tree models follow real growth rules.

5.6 Practical Application Plan

In this paper, we present an innovative method on 3D tree modeling. Tree modeling is a hot research topic in both forestry and computer graphics. This new method with L-systems can produce a user-defined or random 3D tree model. With this tree modeling method, we can control every part in a 3D tree model, including trunks, branches, leaves, flowers, and fruits. For these parts, we can change colors, textures, and shapes very flexibly. The second advantage is about the level control. Based on this control, parts in a tree with different ages look different. The third advantage is the possibility of hemi-sphere control. This is a new method for controlling the shape of a tree by controlling the probable distribution of particles on this hemisphere. Because of the advantages of this new method, it might have three effects: social, political, and economic, which are discussed below.

However, since this new method is currently a basic idea, further work focused on different application areas is need to achieve these goals.

5.6.1 Social Effects

There are two main aspects of this new 3D tree modeling method that have social effects. One is the innovation of this method. Another is the use of this 3D tree model in our society.

The innovation of this method provides a new method for 3D tree generating. It can help researchers to understand tree modeling and even find better solutions. This new method takes advantage of traditional L-systems and solves the problems with L-systems. We also

introduce a new approach using hemisphere and level control. What's more, the bounding box technique can also be applied to this 3D tree model for detecting outer collisions of growth.

This new method can also be used in less technical communities. Based on our idea, we can produce more types of trees and control all parts in a tree. If this method can be applied to work as a demo in some community or school, it helps more people to know how trees grow in a computer. Furthermore, if more biological characters are added, forestry researchers can use this model to predicate the growth of trees and then evaluate the wood productions. However, in addition to our tree model, more work is needed to do to achieve these social effects.

5.6.2 Political Effects

This model has little effect on politics. However, we suggest this tool for governing trees in a forestry department. At present, most forestry departments use a database or even paper to record and manage trees. With our 3D tree model, we can display these data in visual 3D and thus get more direct information from these data. Therefore, our new method, if there are any political effects, can help forestry departments and researchers to get more information from tree data and help to make some decisions.

5.6.3 Economic Effects

One reason for the interest in 3D trees is the wide applicability and economic value. As a product of this simulating tool, the automatic generated tree model can reduce manual work and achieve a good simulating result. Our 3D tree model can be used in commercial 3D games, commercial software of tree models, city planning, and so on.

In commercial 3D games, our tree model can set up a good scene or background. If further work can be done, we will try to improve the efficiency of the current model to make it more applicable.

In commercial software, the method for generating a 3D tree shape is important. Current 3D tree modeling software, such as XFrog, has a great market. Our method suggests a new method of 3D tree modeling and is easy to control. Since this model can also be used in 3D tree modeling software, the method has potential economic value in software design.

In city planning, we can use different shapes of 3D tree models to view corresponding designs. For example, when we choose a type of tree for use on a street, we can generate different 3D tree shapes to make comparisons.

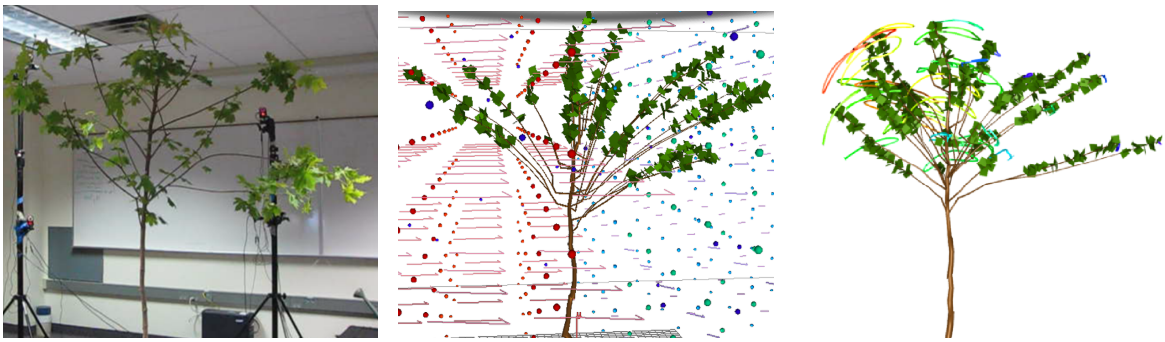
Chapter 6

Animating Trees Using Wind Fields Estimated from Motion Capture Data

Jie Long and Michael Jones. Estimating wind flow from tree motion using motion capture data.

Abstract

We present non-rigid motion capture by extracting external forces from motion capture data and then replaying those forces to create animation. We explore this idea in the context of motion capture of natural trees in wind. Motion of a tree in wind is decomposed into three forces: wind-induced drag, branch elasticity, and damping by the leaves. Given a model of elasticity and damping, the drag force can be isolated and used to estimate wind velocity. The extracted velocity field is extended to a larger volume and enriched with a turbulence model. That wind field can be replayed on a tree model that includes elastic and damping properties to create similar motion.



(a) Motion capture setup. (b) Extracted wind field. (c) Branch motion paths.

Figure 6.1: Wind flow can be estimated from motion capture data and used to recreate similar tree motion.

6.1 Introduction

We address the problem of extracting a spatially varying wind velocity field from partial motion captured data and simulating networks of flexible tree branches embedded in a turbulent flow described by this wind field. This problem is a specific instance in which capturing motion and directly replaying it is difficult. Directly replaying motion capture data is difficult in other settings, such as swimming and rope motion capture, as well. The problem of replaying tree motion in wind is important for animators and game developers. Tree motion can be an important background element in outdoor settings.

Animation of trees in wind has been discussed for many years [1, 52, 56]. In most of the previous research, the wind field is created using noise and fluid simulation. Motion capture avoids the directability and computation problems of simulated wind fields but may yield data that validates simulation-based models.

Motion capture of non-rigid bodies is difficult. Prior work has focused mostly on cloth and facial motion capture [18, 24, 26, 54]. In these methods, the focus is on overcoming difficulties associated with reconstructing the motion of a deforming plane. We focus on the motion of a deforming network of rods. Rather than directly reconstructing the motion of the deforming object, we reconstruct the forces that create the motion. The forces can then be extended and enhanced to recreate similar motion.

We solve the problem of extracting a wind field from motion capture data by separating the forces acting on a tree and isolating the force due to wind. There are three primary forces that create tree branch motion: elasticity, damping, and drag. Elasticity and damping can be estimated directly from position and velocity information computed from marker positions. Elastic and damping forces are subtracted from total force to obtain drag. Given drag, we can solve for wind velocity using the aerodynamic drag equation. All of these steps depend on estimates for elasticity, damping, mass, and drag coefficients. These coefficients are estimated from the forestry and graphics literature and can be adjusted to create different motion effects. The extracted wind field has low spatial resolution due to the distances between markers. A sub-grid scale turbulence model with higher resolution restores motion due to omitted high-frequency small-scale turbulence. We use a classical turbulence model composed of a mean flow velocity and turbulence velocity. The mean flow velocity is computed from motion capture data while the turbulence velocity is created using a *tke* model [37, 39, 51]. The *tke* turbulence model trains the mean velocity field and modulates tuned noises. By integrating these tuned noises into the wind field, we can restore branches' high-frequency motion while preserving the coherence of branch movements on a tree from large-scale turbulence and small-scale turbulence.

The process is illustrated in Figure 6.1. A tree is instrumented with small retroreflective markers, placed in a passive optical motion capture arena, and subjected to wind. A wind field is extracted, as shown in the middle image. The wind field is applied to a tree model and the resulting motion paths of branch tips are shown in the right-most image.

Our primary contribution is creating complete tree motion from partial motion data collected from motion capture. We discuss methods for sampling and extracting the external force on tree branches as well as energy transformation between tree and wind.

6.2 Related Work

Our work is most closely related to prior work in motion capture of flexible objects and in animating trees. In contrast with prior work in motion capture, we focus on flexible rods rather than flexible planes and on extracting external forces rather than directly extracting motion. In contrast with prior work in animating trees, we estimate a wind field from motion capture data rather than simulating the wind field using noise or approximating it using fluid simulation.

Motion capture has been widely used in simulating human motion [18, 25, 44, 62]. Prior work in motion capture of flexible objects focuses on reconstructing a mesh, which deforms to match a moving surface. Ma et al. [26] train a polynomial displacement map and apply this map to create high-resolution facial expressions, including muscle deformation, wrinkles, and skin pores. Lorenzo et al. [24] build a surface-oriented deformation paradigm to animate facial expressions with user intervention. Sifakis et al. [54] combine motion capture data with an anatomical model to produce a model of facial musculature, passive tissue, and the underlying skeletal structure. A key difference between natural trees and facial motion capture is that the drag forces that create natural tree motion in wind may be simpler to extract from motion than the muscular forces involved in facial motion. In this paper, instead of continuously reconstructing a surface, as one does with faces or cloth, we animate trees that have open structures and more degrees of freedom. Also, not only do we create tree motion, but we also simulate wind dynamics, which are scalable.

Recent work extracts forces rather than motion. Kwatra et al. [18] simulate human swimming and interaction with water by combining motion capture data and fluid dynamics. Motion capture records swimming motion with an articulated rigid skeleton model. The forces at joints are computed. By combining the forces with fluid dynamics, this method creates human swimming motion as well as water movement. Our research computes wind forces from motion capture data. Compared to Kwatra et al. [18], this would be like capturing the

motion of a swimmer in water in order to capture the fluid dynamics around the swimmer. A key difference in our model is that we have assumed that the tree does not initiate motion.

Sun et al. [57] propose a method to extract motion patterns from video sequences and reapply these patterns to simulate computer-generated objects. The research focuses on the schema of video input driven animation (VIDA). Motion information is analyzed from 2D video and then incorporated to a conceptual model, such as wind or water dynamics. Our research follows a similar process to the schema. Instead of capturing 2D video, our motion capture system provides more accurate motion information in 3D space. Our research also calculates the interactive energy between trees and wind using particle flow in both space and time domains. By introducing a turbulence model, our tree motion provides more flexibility of simulation control and creates plausible natural tree sway in wind.

Approaches for simulating tree motion in wind with either one- or two-way coupling in a fluid simulation are based on the Navier-Stokes equations. Akagi [1] takes this approach but uses a coarse simulation grid, which omits significant high-frequency fluid turbulence.

The more common approach is to approximate wind dynamics with a frequency-tuned noise model. Ota et al. [32] apply an experimental noise model of $1/f^{\beta}$ to simulate the motion of branches and leaves. Shinya and Fournier [52] present a motion model based on a stochastic process and physical dynamics. They use a power spectrum and autocorrelation of wind to generate a spatiotemporal wind velocity field similar to that created when wind flows through trees. Habel [11] builds a 2D-motion, rather than velocity, texture by combining a Gaussian field with a frequency-tuned 2D velocity field based on a wind dynamics equation with a harmonic oscillator model. The motion texture synthesizes branch motion directly without an integration step. This runs in real time for three moderately complex trees. Stam [56] creates filters for white noise and generates wind fields from samplings. He defines and applies the filtering rules in the frequency domain. The noise model with physical dynamics provides control flexibility and works to create motion for complicated large-scale scenes [67]. In our research, wind field calculation is driven by branch movements recorded from motion

capture. A turbulence model preserves local wind dynamics using particle flow. Using a smooth window, our approach also avoids the complicated time integration of the dynamic system.

Models for simulating large deformation in networks of flexible rods can be applied to animation of trees in wind. Barbic and Zhao [3] present a scalable method for simulating both internal and external forces that can be applied to trees. If combined with a model of external forces due to wind, this may result in convincing methods for animating trees in wind.

Tree motion can also be simulated using data from video or motion capture. Diener [8] records 2D video and extracts features for a single plant. By analyzing the video with these features using hierarchical retargeting algorithms, the 2D motion is projected into 3D space and animates a large class of virtual shrubs. Long et al. [23] reconstruct 3D tree motion in wind using motion capture. Reflective sensor markers are placed along branches. Leaves have to be sparse to ensure the visibility of all markers exposed to capture motion. This approach produces visually realistic movements of a cherry tree in wind. However, this method can only create motion of the original tree and is not scalable to other models. In our research, wind field information is trained from motion capture data. Using particle flows for local wind energy transfer, we are able to simulate the motion of multiple objects in a scene. In addition, reflective markers are placed on the tree crown instead of along a single branch. This design maximizes the visibility of markers to motion capture and records more accurate movement data. It also facilitates the computation with the dynamic model of wind and tree.

6.3 Methods

We estimate a wind field from tree motion using motion capture data. Natural tree motion is captured using passive optical motion capture. The data are analyzed to estimate both a 3D model of the tree geometry and a wind field. The estimated wind field approximates the wind that created the motion recorded in the motion capture data. A fluid simulation

enriched with a synthetic turbulence model transfers energy between wind and tree. The enriched fluid simulation drives animation of a tree model.

Our work can be divided into three parts: motion capture, tree modeling, and wind–tree interaction. We describe each part in the following sections. Of those three parts, wind–tree interaction presents the most difficult and interesting problems.

6.3.1 Motion Capture

As in [23], twelve optical motion capture cameras are placed in a circle around a tree indoors. A fan with varying speed and direction creates tree movement. The cameras record the motion of markers placed on exterior branch tips. Placing markers on branch tips avoids problems with occlusion. Typically we use about 30–70 markers, depending on the size and shape of the tree. Markers are distributed evenly to cover the crown shape.

Optical motion capture records unindexed locations of all the reflective markers in a scene. The recorded data can be processed to label unindexed locations and to eliminate swaps and repair gaps. The algorithm uses forward differences to predict the future position of a point and then minimize the distance between the predicted and the recorded points to add a new position to an existing trace. Details can be found in [23]. At the end of the process, collected marker positions are clustered into paths for each marker.

6.3.2 Tree Modeling

Particle flow can generate 3D branching structures [29, 50, 58]. In most cases, this approach depends on inverse volumetric rendering to identify the position of tree mass. The tree mass is then filled with a branching structure using either particle flow or pre-built libraries of small branching structures. Motion capture data simplify the process because explicit image segmentation and camera calibration are not needed, as they are part of the motion capture process.

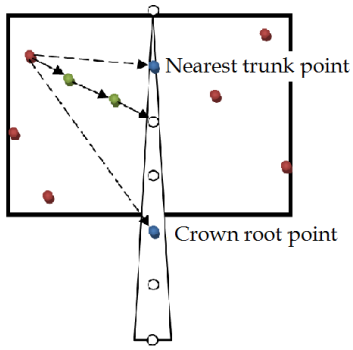


Figure 6.2: A particle with its nearest trunk point and its crown root point.

Particle systems generate the branching structure. In our process, particles are generated at the 3D positions recorded for the branch tips on which reflective markers were placed. We generate additional particles distributed evenly on the periphery of the region bounded by the tree crown. Each of these particles also represents a branch tip.

The particles flow toward a predefined trunk placed vertically in the center of the crown shape. Branch shapes and hierarchies are defined when particle paths connect. Particle flow starts at branch tips and ends at the trunk. Each particle has a direction, a step length, and a threshold for merging. As in [29], the particle direction combines the direction to the crown root point and to the nearest trunk point, shown in Figure 6.2. The red dots in the image indicate locations of branch tips. The crown root point is the lowest point on the trunk near the bounding box of tree crown. The nearest trunk point is on the trunk that has the shortest distance toward a particle. Particles flow from branch tip toward the trunk, which is unlike the particle system in [34], where particles flow from trunk to branch tip. The direction of the first step in the flow is the combination of direction to the nearest trunk point and the crown root point. In the following steps, the flow direction is also a combination of these two directions, but any two particles are merged if their distance is under the threshold of merging distance.

6.3.3 Wind–tree Interaction

Wind–tree interaction is based on a mixed Lagrangian/Eulerian model of the wind field extracted from motion capture data enriched with a subgrid turbulence model. The Eulerian grid stores wind velocities as estimated from motion capture data. Lagrangian particles carry velocity and turbulent kinetic energy *tke* from frame to frame. The *velocity grid* is replaced every frame while the *fluid particles* retain state from frame to frame. Fluid particles carry energy back and forth between the global wind field, the local turbulence model, and the tree. In this way we simulate tree movement as well as the distribution of energy in a dynamic wind velocity field.

Proxy geometry is used to detect and approximate the effects of collisions between fluid particles and tree geometry. Both the effect of wind on trees and the effect of the tree on wind are calculated.

Because the global velocity field is generated from motion capture data, which may contain noise, this field may not be continuous in the time domain. Averaging velocity values for a single grid across several adjacent frames smoothes these variations but also smoothes small-scale turbulence effects. In order to compensate for lost turbulence effects as well as to better preserve wind continuity over time, particles store *tke* across frames and manipulate the scale of turbulence due to characteristics of tree and wind.

Wind Velocity Field

The global grid-based velocity field is the source of wind energy in the scene. Wind force is estimated directly from recorded displacements of branch tips. We solve for wind velocity at branch tips using drag equations and the estimated wind force. This gives a sparse collection of velocity estimates. Interpolation and extrapolation build a grid of velocity values from the sparse collection. Figure 6.3 shows an orthographic projection of such a field. The green dots indicate marker positions, the red arrows represent extracted wind velocities, and the blue arrows represent velocities interpolated or extrapolated from the estimated velocities.

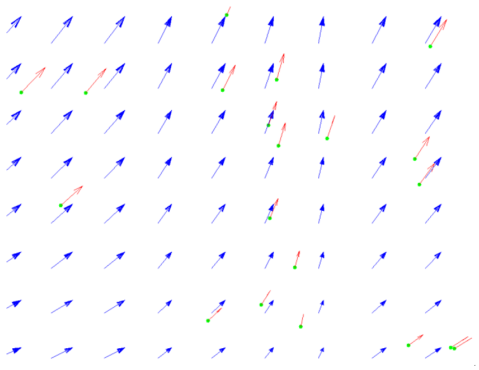


Figure 6.3: Vector field of wind velocities with estimated and interpolated values. Wind velocity estimates based on recorded tree motion are shown in red with interpolated velocities in blue.

The wind field in Figure 6.3 is quite smooth. This is because small-scale turbulent eddies are below the spatial sampling resolution of the motion capture data. A subgrid turbulence model based on *tke* recreates small eddies in the velocity field.

We isolate the drag force in order to estimate the wind velocity. Assuming no external forces other than wind influence the tree motion, the motion of a branch is caused by aerodynamic drag and internal elasticity and damping forces. Elastic forces tend to restore the branch to a resting position and damping forces reduce velocity. In this model, the forces acting on a tree are given by

$$F = F_{wind} + F_{elastic} + F_{damping} = F_{wind} + c\dot{s} + ks. \tag{6.1}$$

Differences between marker positions in successive frames give estimates for position s, velocity $\dot{s}$ and acceleration $\ddot{s}$. All the derivatives are approximated with the marker positions using backward differencing. The displacement s, velocity $\dot{s}$ and acceleration $\ddot{s}$ are calculated as

$$s^t = q^t - q^{t-1}, \dot{s}^t = (s^t - s^{t-1})/2.0, \ddot{s}^t = (\dot{s}^t - \dot{s}^{t-1})/2.0, \tag{6.2}$$

where q^t is a marker position at time t. The elastic force $F_{elastic}$ is the product of branch elasticity and displacement from the rest position. Similarly, the damping force is the product

of the damping coefficient and the velocity. We estimate damping c and spring coefficients k from biomechanical parameters. Since $F = m\ddot{s}$, we substitute $m\ddot{s}$ for F, make similar substitutions for the elastic and damping forces, and then solve for F_{wind} to obtain

$$F_{wind} = m\ddot{s} - c\dot{s} - ks. \tag{6.3}$$

The drag equation gives the force created by wind as a function of the wind velocity V (and other constant parameters explained below):

$$F_{wind} = 0.5\rho(V^2)AC_D, \tag{6.4}$$

where ρ is air density, V is wind velocity relative to branch movement, A is the aerodynamic cross section, and C_D is the drag coefficient.

Substituting the value of F_{wind} calculated using equation (6.3) into equation (6.4) and solving for V^2 gives

$$V = \frac{F_{wind}}{\|F_{wind}\|} * \sqrt{\left|\frac{m\ddot{s} - c\dot{s} - ks}{0.5\rho AC_D}\right|}. \tag{6.5}$$

The solution is based on the assumption that the acceleration is constant and therefore the direction of velocity V and force F_{wind} is preserved as the same in a short period of time. In our case, the time interval is 0.01 sec from motion capture setup. The direction of the velocity V is computed using the unit normal vector of F_{wind} as shown in equation (6.5). The displacement s, velocity $\dot{s}$ and acceleration $\ddot{s}$ are computed using equation (6.2) from the marker locations.

The equation (6.5) calculates a wind velocity estimate in each frame at the location of each branch tip. Motion capture markers are distributed evenly through the crown in order to sample the global wind field over a large area.

Interpolating and extrapolating velocity values over the entire motion capture volume results in a grid-based velocity field. The size of a grid cell is set to be close to the average

distance traversed by a single marked branch. By doing this, we can optimize the usage of motion capture data during the interpolating and extrapolating process. In each frame, the ideal case is that each grid cell contains exactly one motion-captured point. In our case, the grid resolution is $10 * 10 * 10$. We use linear interpolation to fill the grid from sampled values because linear interpolation is simple. More sophisticated methods, such as Kriging (as in [10]) or distance-weighted kernel-based methods, could also be used.

The grid-based wind velocity field is computed from each individual frame of branch tip data. However, the wind velocity field does not vary smoothly from frame to frame because there are gaps and jumps in the data. To improve temporal continuity, velocity values over the neighboring 5 frames are averaged.

Turbulent motion below the sampling scale of the motion capture data cannot be captured by this model. In space, we are only able to extract turbulent effects that are large enough to influence the motion of two adjacent markers. Because leaves have small mass and large surface area, small-scale turbulence results in visually significant motion on tree crowns. That turbulence is well below the sampling limit of the data. The mean flow inferred from motion capture data will be enriched with a turbulence model at a finer resolution.

Rather than apply the velocity field directly to the tree, we introduce particles into the field and collide particles with proxy spheres attached to the tree. These particles have zero mass and do not affect the mean flow. This scheme simplifies calculation of wind–tree interaction by replacing cell–tree collisions with point–sphere collisions. Cell–tree collisions can be expensive for determining what fraction of a cell contains a fraction of a generalized cylinder representing the tree branch. With particles and proxy geometry, estimating wind velocity can be reduced to a distance-weighted average of the particles contained in a sphere.

Wind Effects on a Tree

Simulating wind effects on a tree creates branch motion. The simulation is the key to creating natural tree motion. We have discussed creating a grid-based wind velocity field using

motion capture data. Because of motion capture's capability to record a limited number of markers, the resolution of the gird is low. The low resolution simulates big-scale turbulence but lacks small-scale turbulence information due to a high-frequency wind field. Selino and Jones [51] solve this problem by introducing the small-scale turbulence, which is computed from large-scale turbulence. This solution fits into our problem very well. In our case, the low resolution of the wind velocity field produces large-scale turbulence. The large-scale turbulence will be tuned by a *tke* model to create small-scale turbulence. The combination produces wind effects, including both large- and small-scale turbulence, on a tree.

The simulation of wind effects on a tree preserves coherence of velocity from several perspectives. The mean flow velocity computed from interpolating motion capture data contains information of tree movements as a whole. The *tke* model keeps track of turbulence changes in the flow and thus tunes the noise value due to these changes. The Gaussian noise field is filtered in the frequency domain to match the frequency of turbulence observed in tree crowns. The model is derived from a classical turbulence model and generates turbulence effects, including both large scale and small scale.

Our approach is different from Selino's work because of the source of large-scale turbulence. Instead of a real-time fluid simulation, we generate the large-scale turbulence offline from motion capture. In Figure 6.4, we demonstrate our turbulence model to generate tree motion. The turbulence is computed in two parts as in large scale and small scale. The large-scale turbulence is created from the grid-based wind velocity field. The small scale is generated from the *tke* model. After solving the turbulence velocity, we apply a drag equation as shown in equation (6.4) to solve for the tree motion represented by branch displacements.

The turbulence velocity $V(t)$ applied to a branch segment is computed as in equation (6.6):

$$V(t) = v(t) + \alpha\sqrt{k(t)}N(t), \tag{6.6}$$

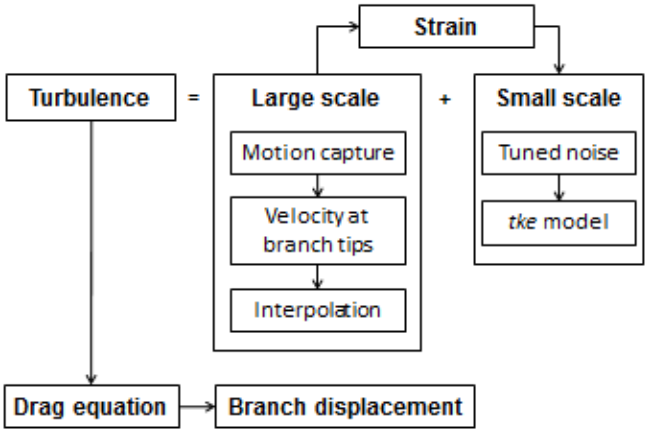


Figure 6.4: Turbulence simulation to create branch motion.

where $v(t)$ is the mean flow velocity for the large-scale turbulence at time t, α is a tunable weight parameter, $k(t)$ is from *tke* model, and $N(t)$ is a sampled value from a frequency-tuned time-varying noise field.

In the simulation, each fluid particle carries a mean flow velocity from the global wind velocity field and a *tke* estimate from the turbulence model. The value $k(t)$ of the *tke* is estimated using a two-equation *tke* budget based on strain in the velocity field as shown in equation (6.7) and (6.8). The *tke* model is adapted from Selino [51] and Pfaff et al. [37], and is based on Pope [39]. The value $k(t)$ represents turbulent kinetic energy and is generated when the strain in the mean flow velocity is not zero. The decay term ϵ models dissipation rate of that turbulence energy. When the turbulence production P exists due to strain, the k and ϵ are computed as follows:

$$\frac{\partial k}{\partial t} = P - \epsilon, \frac{\partial \epsilon}{\partial t} = C_{\epsilon 1}\frac{P\epsilon}{k} - C_{\epsilon 2}\frac{\epsilon^2}{k}, \tag{6.7}$$

where P is the production of turbulence, t is current time step, and model constants are $\epsilon 1$ = 1.44 and $\epsilon 2$ = 1.92. When the production P of turbulence in the fluid becomes zero, the values of k and ϵ dissipate and compute as in the equation (6.8):

$$k(t) = k_0\left(\frac{t}{t_0}\right)^{-n}, \epsilon(t) = \epsilon_0\left(\frac{t}{t_0}\right)^{-(n+1)}, \tag{6.8}$$

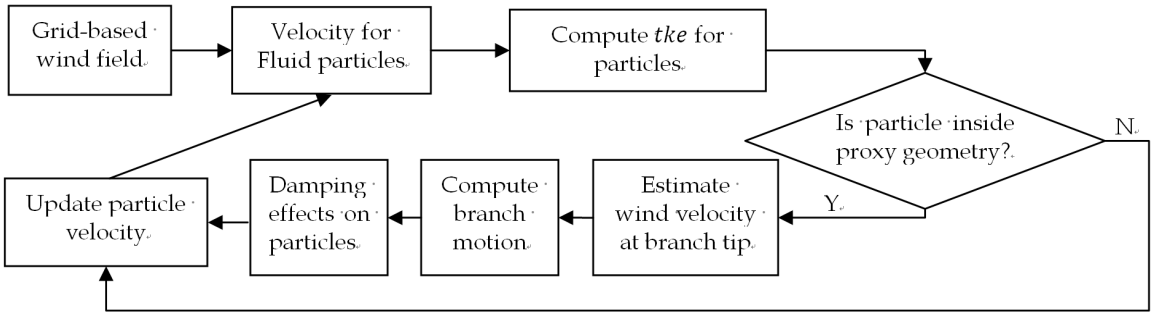


Figure 6.5: Energy flow in a wind velocity field.

where t is current time step, n is a constant decay component computed by $\frac{1}{C_{\epsilon 2-1}}$, k_0 and ϵ_0 are the most current values, respectively, in the past when there exists turbulence production P, and the reference time t_0 is computed as $n\frac{k_0}{\epsilon_0}$.

Based on equation (6.7) and (6.8), we compute for the value of $k(t)$ in equation (6.6).

The noise $N(t)$ is sampled from a continuous noise field. That noise field is tuned to match the frequency distribution of turbulent flow through trees given in Simiu [55].

The mean flow velocity $v(t)$ is sampled using a distance-weighted average of particles' velocity contained within the proxy sphere of a branch tip. Proxy geometry is defined for each branch segment using 3D spheres with radii proportional to branch segment length and diameter as suggested from Selino and Jones [51].

Figure 6.5 shows the energy flow during the simulation. Wind energy is transferred from the grid-based velocity field to particles. The turbulence model calculates *tke* from the transferred wind energy. When particles pass through proxy geometry, velocity is converted to a drag force using equation (6.4) and branch motion is created.

Tree Effects on Wind

Tree effects on wind are split into local and global effects. Local effects happen at the scale of a few leaves and branch segments. Global effects happen at the scale of the entire tree. Local effects are simulated using a simple damping model on wind particles, and global effects are recorded during motion capture. Simulating local effects allows for variations in wind flow

due to differences between the shape and dynamics of the animated tree compared to the shape and dynamics of the actual tree used in motion capture.

Tree effects on wind on the local scale are calculated as a damping force for particles within the proxy geometry representation of tree mass. Consider the velocity $v_i(t)$ of particle i in frame t sampled before calculating branch movement. Suppose that particle i lies in proxy geometry for some branch in frame t. After displacement of the branch tip has been calculated, if the particle lies within the proxy geometry then we compute the influence of the branch on the particle velocity. The new velocity of particles in the proxy geometry is computed as follows in equation (6.9):

$$v_i^{'}(t) = v_i(t)(1 - \beta), \tag{6.9}$$

where β is a constant decay rate that changes particle velocity based on damping by the tree. The rate is constant for each branch, but varies due to the size, material, leaf shapes, and other properties of the tree. We set the reduction rate between 0.02 and 0.05 in most simulations. This reduction smoothes out the mean velocity over time while the high-frequency details are compensated from the *tke* turbulence model.

In the next frame $t + 1$, we first load the estimated global grid-based wind field generated from motion capture data. The particle velocity $v_i^{'}(t)$ from the previous frame is used as the mean flow velocity to compute the current particle's turbulent kinetic energy $k(t + 1)$. Then we use equation (6.5) to calculate wind velocity $V(t + 1)$ experienced by the tree geometry.

The global effect of the tree on the wind is inferred from the motion capture data. Branches on the leeward side of the tree have less motion. When the particle exits the proxy geometry, the damping is no longer computed, so the particle velocity is reset to match the estimated global velocity v.

Tree Animation

The movement of the whole tree is computed from the drag force at each branch tip. We use a damped mass-spring model to define branch dynamics. When the density, stiffness, and damping coefficients match the estimates used to calculate wind velocity, the resulting motion is similar to the original motion. The drag forces are calculated for each branch segment.

6.4 Results

We present results that show animation of a tree in wind using a wind velocity field inferred from motion capture data for a similarly sized tree moving in wind.

We first create a 3D tree shape using particle flow. In Figure 6.6, the small white cubes on some of the branch tips indicate the location and total number of reflective markers we capture for the tree. The locations of branch tips that do not correspond to a marker location are created within the stacked bounding boxes. We generate tree leaves based on the branching structure and instance leaf size and shape randomly. By the refined bounding box method, the generated 3D tree shape also is visually similar to the crown shape of the original tree. The similarity between the generated tree model and the actual tree shape can be seen in parts (a) and (b) of Figure 6.1. It is significant that the tree shape does not perfectly match the original tree shape. Applying the motion capture data indirectly as an inferred wind velocity field, rather than directly as a set of motions, allows for variations in tree shape and branching structure.

The *tke* model is a significant part of producing realistic motion. In Figure 6.7, we compare the results generated with and without turbulent *tke*. Each figure shows motion paths from an animation of a tree in wind with and without sub-grid turbulence based on the *tke* model. The path on the left includes motion due to sub-grid *tke* and the path on the right does not. In both cases, moving branches trace out arcs of similar size as they sway left and right. Adding *tke* has the effect of adding variation to each sway motion so that the arcs are each in a slightly different location. As a result, the motion path on the left, which

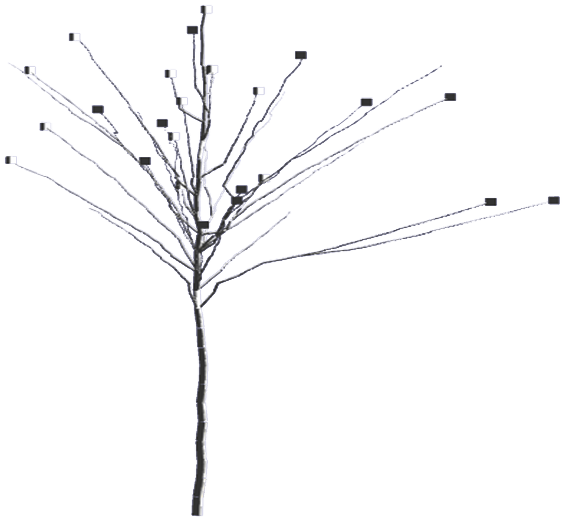


Figure 6.6: 3D branching structure with marker locations.

Figure 6.7: Branch motion paths with (left) and without (right) sub-grid turbulence. Adding turbulence adds small variations to each branch sway, as seen in the less compact motion path on the left.

includes *tke*, is less compact while the motion path on the right, which does not include *tke*, contains many overlapping arcs.

The images in Figure 6.8a and 6.8b compare recorded motion of a tree with motion generated for a similar tree using the wind field inferred from motion capture data. The actual tree motion is shown using green traces on the left and the generated motion is shown on the right. Note that motion is not captured for every branch tip on the left, but motion is generated for every branch tip on the right. The images have been aligned so that the wind direction is the same in both cases. Branches in similar positions on the crown have similar motion.

In Figure 6.9, we demonstrate that our method produces plausible tree sway motion and results in tree motion effects observed in nature, such as sheltering. Branches bend with

(a) Original tree motion. (b) Animated tree motion.

Figure 6.8: Motion paths for the animated tree are similar to motion paths in the original tree for branch tips at the same relative crown location.

the wind and then return to their rest position. Branch sheltering within a single tree crown occurs when branches on the leeward side of the crown experience lower wind velocities than branches on the windward side because drag exerted by the windward branches reduces the wind velocity. In the figure, the dominant wind direction is from left to right and from the lower left corner. We can observe that branches, which are on the left side (the windward side), have bigger amplitude of motion than others. Besides completing tree motion of a single similar tree with partial motion capture data, our method can also be extended to create tree motion for multiple trees using the wind field. The trees in Figure 6.10 move in the same wind field. Although this creates the appearance of a group of trees moving in a shared environment, wind is not damped as it passes from one tree to another. In the video clip attached to this paper, we provide a side-by-side comparison of the video originally recorded during the motion capturing process and the synthesized tree animation. From the video, we can observe that sheltering effects among branches are successfully simulated.

6.5 Discussion and Future Work

We have presented animation of non-rigid bodies using partial motion capture data by extracting a wind velocity field rather than replaying motion data directly. This simplifies the problem as we no longer need to match motion capture data to a precise reconstruction of

Figure 6.9: Several frames from the animation of a maple tree in a wind field extracted from motion capture data.

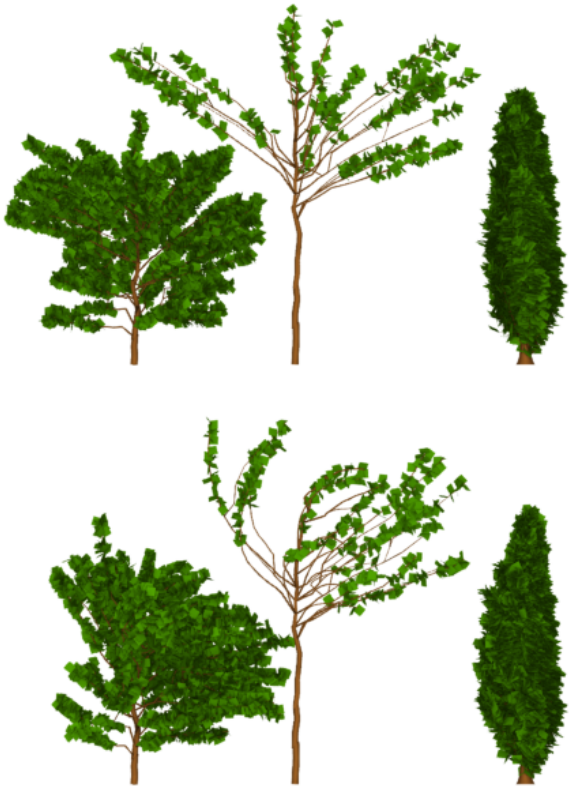


Figure 6.10: Multiple trees swaying in wind field estimated from the motion of a single tree.

the original capture subject. It also results in complete and coherent motion from incomplete data that contains discontinuous motion. We have also presented a method for enriching the extracted force field to include fine-resolution turbulence. This is possible because the extracted velocity field can be analyzed and enriched, much like position graphs can be analyzed and enriched in other applications.

This work opens a new direction in the motion capture of non-rigid bodies in spatially smooth force fields. We have investigated this idea in the context of trees and wind. Future work might focus on other objects, such as cloth, in other flows.

Chapter 7

Discussion and Conclusion

The research builds a new model of motion capture for non-rigid bodies under external forces. From partially recorded movements on non-rigid subjects, we create complete animations of these or similar subjects using physical and statistical models. We present a generic pipeline for capturing movements of non-rigid bodies using passive optical motion capture. We have demonstrated animation of non-rigid bodies using partial motion capture data by extracting external forces rather than replaying motion data directly. The approach simplifies the problem as we no longer need to match motion capture data to a precise reconstruction of the original capture subject. It also results in complete and coherent motion from incomplete data that contain discontinuous motion.

Chapter 2 reconstructs tree motion under natural wind. It builds a plausible tree skeleton using a minimal spanning tree algorithm over a cost function defined using position and motion data. Gaps and errors in motion capture data for trees are replaced with data interpolated from neighboring branch motion. These are important steps toward realizing motion capture of trees for tree animation in games. We had hoped to get better results with the repaired data and the rigid body algorithm we used. Based on the results of this pilot project, we believe that investigating other approaches to processing the point cloud are more promising than repairing the errors caused by using the rigid body algorithm we used.

In Chapter 3, our work produces visually plausible rope motion from passive optical motion capture data using a statistical model under the assumption that the rope does not stretch. The algorithm preserves continuity of motion in traces and fits the shape of rope.

This work lays a foundation for further investigation of motion capture for non-rigid bodies using statistical rather than physical models. The approach to the problem may advance motion capture results for non-rigid bodies driven by complex or poorly understood physical systems.

Chapter 4 generates a 3D tree model using particle flow along with motion capture data. The particle flow system starts from recorded branch tip positions supplemented with additional random branch tip positions within a horizontal stack of bounding boxes and by setting two control parameters. A new data collection process designed for trees may extend the use of motion capture to include trees and, eventually, other networks of non-rigid bodies.

Chapter 5 introduces a new method based on L-systems for 3D tree modeling. This new method employs a moving particle for a branch unit, which reduces computation time compared to diffusion-limited aggregation. We control the main crown shapes with the hemisphere while controlling the internal growth structure using growth levels. Under these controls, randomness is added by randomly selecting tree branches from the L-systems branch library. By introducing the random factors, it is difficult to produce the exact shape of the original tree. Also, small changes in parameters might produce big changes in the output tree shape, which is a common problem when employing L-systems.

Chapter 6 is based on all of the previous work in our research. In this project, we present animation of non-rigid bodies using partial motion capture data by extracting a wind velocity field rather than replaying motion data directly. This simplifies the problem as we no longer need to match motion capture data to a precise reconstruction of the original capture subject. It also results in complete and coherent motion from incomplete data that contains discontinuous motion. We have also presented a method for enriching the extracted force field to include fine-resolution turbulence. This is possible because the extracted velocity field can be analyzed and enriched, much like position graphs can be analyzed and enriched in other applications. This work opens a new direction in the motion capture of non-rigid bodies in spatially smooth force fields. We have investigated this idea in the context of trees

and wind. Future work might focus on other objects, such as cloth, in other flows. We can also extend the process to larger trees outdoors.

As inferred from the rope reconstruction project, complicated motions (such as spirals, collisions, sudden changes in movement, or extremely fast movement) are not well handled in our model. Our assumptions for detecting swaps may be oversimplified relative to natural movement. In the future, consideration of other factors, such as velocity or acceleration, might improve gap-filling results. We have used a simple method for interpolating rope position between markers. More complex methods may result in more plausible results, particularly when the distance between markers on the rope is large.

Our research discusses animating non-rigid bodies using motion capture, but mostly focuses on replaying motion for trees and ropes. We present a general application of motion capture for non-rigid bodies, including a data collection process and data cleaning algorithms. The work can be extended to non-rigid bodies other than trees and ropes. By emphasizing different non-rigid subjects under various force fields, followed by our general application of motion capture, the data cleaning algorithms can be improved with the context, and different physical and/or statistical models might be required to build an animation. Besides building animations that replay the original movements, resulting estimated motion can also validate the real collected data against theoretic physical and/or statistical models. Therefore, the presented application of motion capture for non-rigid bodies can go further in different research areas and become a useful tool in these areas.

References

[1] Yasuhiro Akagi and Katsuhiro Kitajima. Computer animation of swaying trees based on physical simulation. *Computers and Graphics*, 30(4):529–539, 2006.

[2] Masaki Aono and Tosiyasu Kunii. Botanical tree image generation. *IEEE Computer Graphics Applications*, 4(5):10–34, 1984.

[3] Jernej Barbic and Yili Zhao. Real-time large-deformation substructuring. *ACM Transactions on Graphics*, 30(4):91:1–91:7, 2011.

[4] Kiran S. Bhat, Christopher D. Twigg, Jessica K. Hodgins, Pradeep K. Khosla, Zoran Popovi, and Steven M. Seitz. Estimating cloth simulation parameters from video. In *Proceedings of the Eurographics Symposium on Computer Animation*, pages 37–51, 2003.

[5] Bernd Bickel, Mario Botsch, Roland Angst, Wojciech Matusik, Miguel Otaduy, Hanspeter Pfister, and Markus Gross. Multi-scale capture of facial geometry and motion. *ACM Transactions on Graphics*, 26:33–41, 2007.

[6] Yung-Yu Chuang, Dan B. Goldman, Ke C. Zheng, Brian Curless, David H. Salesin, and Richard Szeliski. Animating pictures with stochastic motion textures. *ACM Transactions on Graphics*, 24(3):853–860, July 2005.

[7] Phillippe de Reffye, Claude Edelin, Jean Francon, Marc Jaeger, and Claude Puech. Plant models faithful to botanical structure and development. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, pages 151–158, 1988.

[8] Julien Diener, Lionel Reveret, and Eugene Fiume. Hierarchical retargetting of 2d motion fields to the animation of 3d plant models. In *Proceedings of the Eurographics Symposium on Computer Animation*, pages 187–195, 2006.

[9] Julien Diener, Mathieu Rodriguez, Lionel Baboud, and Lionel Reveret. Wind projection basis for real-time animation of trees. In *Proceedings of the Eurographics Symposium on Computer Animation*, pages 533–540, March 2009.

[10] Melanie Ganz, Marco Loog, Sami Brandt, and Mads Nielsen. Dense iterative contextual pixel classification using kriging. In *Proceedings of the Computer Vision and Pattern Recognition Workshops*, volume LNCS 5519, pages 87–93, June 2009.

[11] Ralf Habel, Alexander Kusternig, and Michael Wimmer. Physically guided animation of trees. In *Proceedings of the Eurographics Symposium on Computer Animation*, pages 523–532, March 2009.

[12] Jody D. Jesser and Lisa Fitzpatrick. *The Making of Avatar*. Abrams, 2010.

[13] Wenguang Jiang, Minsi Yao, and James M. Walton. A concise finite element model for simple straight wire rope strand. *International Journal of Mechanical Sciences*, 41(2): 143–161, 1999.

[14] Sathiya S. Keerthi, Shirish Shevade, Chiranjib Bhattacharyya, and Krishna K.R. Murthy. Improvements to Platt's SMO algorithm for SVM classifier design. *Neural Computation*, 13(3):637–649, March 2001.

[15] Theodore Kim, Jason Sewall, Avneesh Sud, and Ming Lin. Fast simulation of Laplacian growth. *IEEE Computer Graphics and Applications*, 27(2):68–76, 2007.

[16] Adam G. Kirk, James F. O.Brien, and David A. Forsyth. Skeletal parameter estimation from optical motion capture data. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 782–788, June 2005.

[17] Winfried Kurth and Branislav Sloboda. Growth grammars simulating trees-an extension of L-systems incorporating local variables and sensitivity. *Silva Fennica*, 31(3):285–295, 1997.

[18] Nipun Kwatra, Chris Wojtan, Mark Carlson, Irfan Essa, Peter J. Mucha, and Greg Turk. Fluid simulation with articulated bodies. *IEEE Transactions on Visualization and Computer Graphics*, 16:70–80, 2010.

[19] Chuan Li, Oliver Deussen, Yi-Zhe Song, Phil Willis, and Peter Hall. Modeling and generating moving trees from video. *ACM Transactions on Graphics*, 30:127:1–127:12, 2011.

[20] Aristid Lindenmayer. Mathematical models for cellular interaction in development: Parts I and II. *Journal of Theoretical Biology*, 18(3):280–315, March 1968.

[21] Bernd Lintermann and Oliver Deussen. Interactive modeling of plants. *IEEE Computer Graphics Applications*, 19(1):56–65, January 1999.

[22] Guodong Liu and Leonard McMillan. Estimation of missing markers in human motion capture. *The Visual Computer*, 22(9):721–728, September 2006.

[23] Jie Long, Cory Reimschussel, Ontario Britton, Anthony Hall, and Michael Jones. Motion capture for a natural tree in the wind. In *Proceedings of the Third International Conference on Motion in Games*, pages 158–169, 2010.

[24] Manuel S. Lorenzo, Manuel Sonchez, Lorenzo James, James D. Edge, Scott A. King, and Steve Maddock. Use and re-use of facial motion capture data. In *Proceedings of the Vision, Video and Graphics Conference*, pages 135–142, July 2003.

[25] Hui Lou and Jinxiang Chai. Example-based human motion denoising. *IEEE Transactions on Visualization and Computer Graphics*, 16(5):870–879, September 2010.

[26] WanChun Ma, Andrew Jones, JenYuan Chiang, Tim Hawkins, Sune Frederiksen, Pieter Peers, Marko Vukovic, Ming Ouhyoung, and Paul Debevec. Facial performance synthesis using deformation-driven polynomial displacement maps. *ACM Transactions on Graphics*, 27:121:1–121:10, December 2008.

[27] Volker S. Marcus and Marcus A. Magnor. Cloth motion from optical flow. In *Proceedings of the 9th International Fall Workshop on Vision, Modeling and Visualization*, pages 117–124, 2004.

[28] Radomir Mech and Przemyslaw Prusinkiewicz. Visual models of plants interacting with their environment. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*, pages 397–410, New York, NY, USA, 1996.

[29] Boris Neubert, Thomas Franken, and Oliver Deussen. Approximate image-based tree-modeling using particle flows. *ACM Transactions on Graphics*, 26(3):88:1–88:11, July 2007.

[30] Hiromi Ono. Practical experience in the physical animation and destruction of trees. In *Proceedings of the 8th Eurographics Workshop on Computer Animation and Simulation*, pages 149–159. Springer, Budapest, Hungary, 1997.

[31] Peter E. Oppenheimer. Real time design and animation of fractal plants and trees. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*, pages 55–64, New York, NY, USA, 1986.

[32] Shin Ota, Tadahiro Fujimotoa, Machiko Tamura, Kazunobu Muraoka, Kunihiko Fujita, and Norishige Chiba. $1/f^{\beta}$ noise-based real-time animation of trees swaying in wind

fields. In *Proceedings of the Computer Graphics International Conference*, pages 52–59, July 2003.

[33] 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(10):613–623, 2004.

[34] Wojciech Palubicki, Kipp Horel, Steven Longay, Adam Runions, Brendan Lane, Radomr Mech, and Przemyslaw Prusinkiewicz. Self-organizing tree models for image synthesis. *ACM Transactions on Graphics*, 28(3):58:1–58:10, July 2009.

[35] Sang II. Park and Jessica K. Hodgins. Capturing and animating skin deformation in human motion. *ACM Transactions on Graphics*, 25(3):881–889, July 2006.

[36] Jari Perttunen, Risto Sievanen, and Eero Nikinmaa. Lignum: a model combining the structure and the functioning of trees. *Ecological Modelling*, 108(1-3):189–198, 1998.

[37] Tobias Pfaff, Nils Thuerey, Jonathan Cohen, Sarah Tariq, and Markus Gross. Scalable fluid simulation using anisotropic turbulence particles. *ACM Transactions on Graphics*, pages 174:1–174:8, 2010.

[38] John C. Platt. Fast training of support vector machines using sequential minimal optimization. *Advances in Kernel Methods: Support Vector Learning*, pages 185–208, 1999.

[39] Stephen B. Pope. *Turbulent Flows*, chapter 10, pages 373–382. Cambridge University Press, 2000.

[40] David Pritchard and Wolfgang Heidrich. Cloth motion capture. In *Proceedings of the ACM SIGGRAPH Sketches and Applications*, volume 22, pages 263–271, New York, NY, USA, 2003.

[41] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer-Verlag New York, Inc., New York, NY, USA, 1990.

[42] Przemyslaw Prusinkiewicz, Mark S. Hammel, and Eric Mjolsness. Animation of plant development. In *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*, pages 351–360, New York, NY, USA, 1993.

[43] Przemyslaw Prusinkiewicz, Lars Mundermann, Radoslaw Karwowski, and Brendan Lane. The use of positional information in the modeling of plants. In *Proceedings of the 28th*

Annual Conference on Computer Graphics and Interactive Techniques, pages 289–300, New York, NY, USA, 2001.

[44] Stjepan Rajko and Gang Qian. Real-time automatic kinematic model building for optical motion capture using a Markov random field. In *Proceedings of the IEEE International Conference on Human-Computer Interaction*, pages 69–78, Berlin, Heidelberg, 2007.

[45] Alex Reche-Martinez, Ignacio Martin, and George Drettakis. Volumetric reconstruction and interactive rendering of trees from photographs. *ACM Transactions on Graphics*, pages 720–727, 2004.

[46] William T. Reeves. Particle systems A technique for modeling a class of fuzzy objects. In *Proceedings of the 10th Annual Conference on Computer Graphics and Interactive Techniques*, pages 91–108, New York, NY, USA, April 1983.

[47] William T. Reeves and Ricki Blau. Approximate and probabilistic algorithms for shading and rendering structured particle systems. In *Proceedings of the 12th Annual Conference on Computer Graphics and Interactive Techniques*, pages 313–322, New York, NY, USA, 1985.

[48] Bodo Rosenhahn, Uwe G. Kersting, Andrew W. Smith, Jason. K. Gurney, Thomas Brox, and Reinhard Klette. A system for markerless human motion estimation. In *Proceedings of the 27th Conference on Pattern Recognition*, pages 230–237, Berlin, Heidelberg, 2005.

[49] Mark Rudnicki and D. Burns. Branch sway period of 4 tree species using 3-d motion tracking. In *Proceedings of the Fifth Plant Biomechanics Conference*, pages 25–30, Stockholm, Sweden, 2006.

[50] Adam Runions, Brendan Lane, and Przemyslaw Prusinkiewicz. Modeling trees with a space colonization algorithm. In *Proceedings of the Eurographics Workshop on Natural Phenomena*, pages 63–70, 2007.

[51] Anthony Selino and Michael Jones. Large and small eddies matter: Animating trees in wind using coarse fluid simulation and synthetic turbulence. *Computer Graphics Forum*, 24:417–425, July 2012.

[52] Mikio Shinya and Alain Fournier. Stochastic motion motion under the influence of wind. In *Proceedings of the Eurographics Symposium on Computer Animation*, volume 11, pages 119–128, 1992.

[53] Ilya Shlyakhter, Max Rozenoer, Julie Dorsey, and Seth Teller. Reconstructing 3d tree models from instrumented photographs. *IEEE Computer Graphics and Application*, 21 (3):53–61, May 2001.

[54] Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Transactions on Graphics*, 24(3):417–425, 2005.

[55] Emil Simiu and Robert H. Scanlan. *Wind Effects on Structures: Fundamentals and Applications to Design*. Number V. 1 in Wiley-Interscience Publication. John Wiley, 1996.

[56] Jos Stam. Stochastic dynamics: Simulating the effects of turbulence on flexible structures. *Computer Graphics Forum*, 16(3):159–164, 1997.

[57] Meng Sun, Allan D. Jepson, and Fiu Eugene. Video input driven animation. In *Proceedings of the 9th IEEE International Conference on Computer Vision*, volume 2, pages 96–106, Washington, DC, USA, 2003.

[58] Ping Tan, Gang Zeng, Jingdong Wang, Sing Bing Kang, and Long Quan. Image-based tree modeling. *ACM Transactions on Graphics*, 26(3):87, 2007.

[59] Tamas Vicsek. *Fractal Growth Phenomena*. World Scientific Publishing Co Pte Ltd; 2nd edition, 1991.

[60] Jason Weber and Joseph Penn. Creation and rendering of realistic trees. In *Proceedings of the 22nd Annual Conference on Computer Graphics and Interactive Techniques*, pages 119–128, New York, NY, USA, 1995.

[61] Roger Weber, Hans-Jorg Schek, and Stephen Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of the 24th International Conference on Very Large Data Bases*, pages 194–205, San Francisco, CA, USA, 1998.

[62] Gaojin Wen, Zhaoqi Wang, Shihong Xia, and Dengming Zhu. From motion capture data to character animation. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 165–168, New York, NY, USA, 2006.

[63] Ryan White, Keenan Crane, and D. A. Forsyth. Capturing and animating occluded cloth. *ACM Transactions on Graphics*, 26:34:1–34:9, 2007.

[64] Marcel Worring, Pia Pfluger, Arnold Houtsmuller, and Adriaan Houtsmuller. Measurement of 3d line shaped objects. *Pattern Recognition Letters*, 15:497–506, 1994.

[65] Enhua Wu, Yanyun Chen, Tao Yan, and Xiaopeng Zhang. Reconstruction and physically-based animation of trees from static images. In *Proceedings of the Eurographics Symposium on Computer Animation and Simulation*, pages 157–166, Milano, Italy, September 1999.

[66] Hongping Yan, Philippe De Reffye, Jonathan Leroux, and Baogang Hu. Study on plant growth behaviors simulated by the functional-structural plant growth model — greenlab. In *Proceedings of the Plant Modeling, Visualization, and its Applications*, pages 118–125, October 2003.

[67] Long Zhang, Chengfang Song, Qifeng Tan, Wei Chen, and Qunsheng Peng. Quasi-physical simulation of large-scale dynamic forest scenes. In *Proceedings of the 24th Computer Graphics International Conference*, pages 735–742, Hangzhou, China, June 2006.

[68] Victor B. Zordan and Nicholas C. Van Der Horst. Mapping optical motion capture data to skeletal motion using a physical model. In *Proceedings of the Eurographics Symposium on Computer Animation*, pages 245–250, Aire-la-Ville, Switzerland, Switzerland, 2003.