# Three Applications of Optimization in Computer Graphics

Jeffrey Smith

April 2003

CMU-RI-03-1XX

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Submitted in partial fulfillment of the requirements*
*for the degree of Doctor of Philosophy.*

**Thesis Committee:**
Irving Oppenheim, Chair
Jessica Hodgins
Omar Ghattas
Paul Heckbert



This research was supported in part by the National Science Foundation under Grant No. CCR-xyzzy.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of Carnegie Mellon or the U.S. Government.

# Contents


**1 Introduction** **1**

1.1 Problem Statement . . . . . 2

1.2 Scope . . . . . 2

**2 An Introduction to Optimization** **5**

2.1 Unconstrained vs. Constrained . . . . . 5

2.1.1 Unconstrained Optimization . . . . . 6

2.1.2 Constrained Optimization . . . . . 6

2.2 Global vs. Local . . . . . 8

2.2.1 Simulated Annealing . . . . . 9

2.2.2 Genetic Programming . . . . . 9

**3 Gait Synthesis** **11**

3.1 Using Computation to aid Animation . . . . . 11

3.1.1 Controllers . . . . . 12

3.1.2 Motion Capture and Retargeting . . . . . 14

3.1.3 Keyframing . . . . . 15

3.1.4 Forward Simulation . . . . . 15

3.1.5 Spacetime Constraints . . . . . 16

3.2 The Biology of Legged Locomotion . . . . . 17

3.3 A Closer Look at Spacetime Optimization . . . . . 17

3.3.1 Constraints . . . . . 18

3.3.2 Objective Function . . . . . 19
3.4 A simple example . . . . . 20
3.4.1 Pose constraints . . . . . 21
3.4.2 Mechanical constraints . . . . . 21
3.4.3 Newtonian constraints . . . . . 22
3.4.4 The parameterized equations . . . . . 24
3.4.5 Solution techniques . . . . . 26
3.4.6 Discussion of Sample Problem Results . . . . . 26
3.5 A Reformulation of Spacetime Optimization . . . . . 28
3.5.1 Application of the Reduced Gradient Method . . . . . 29
3.5.2 The Physical Simulation . . . . . 31
3.6 Gradient Estimation . . . . . 33
3.6.1 Finite Differences . . . . . 33
3.7 Reduced Gradient Spacetime Optimization with Finite Differences . . . . . 36
3.7.1 Pogo stick results . . . . . 37
3.7.2 Luxo results . . . . . 41
3.7.3 Discussion of Finite Difference Results . . . . . 42
3.8 Reduced Gradient Spacetime Optimization with Automatic Differentiation . . . . . 47
3.8.1 Results with Automatic Differentiation . . . . . 48
3.9 Simulated Annealing . . . . . 52
3.9.1 Our Simulated Annealing Experiments . . . . . 53
3.10 Summary and Discussion . . . . . 56
3.10.1 Speeding up the Simulation . . . . . 57
3.10.2 Different Physics Formulations . . . . . 57
3.10.3 Initial Guess Quality . . . . . 58
3.10.4 High Dimensionality . . . . . 58

**4 Truss Structures** **61**

4.1 Introduction . . . . . 61

4.1.1 Background . . . 61
4.2 Representing Truss Structures . . . 63
4.2.1 Constructing the Model . . . 63
4.3 Optimizing Truss Structures . . . 64
4.3.1 Mass Functions . . . 65
4.3.2 Cross-Sectional and Geometry Optimization . . . 66
4.3.3 Objective and Constraint Functions . . . 67
4.4 Results . . . 68
4.4.1 Bridges . . . 69
4.4.2 Eiffel Tower . . . 69
4.4.3 Roof Trusses . . . 70
4.4.4 Michell Truss . . . 70
4.4.5 Timing Information . . . 70
4.5 Summary and Discussion . . . 71

**5 Constant Mean Curvature Structures 81**
5.1 Introduction . . . 81
5.1.1 Background . . . 81
5.2 Modeling CMC Objects . . . 84
5.3 Optimizing CMC Objects . . . 86
5.3.1 Geometric Constraints . . . 87
5.3.2 Constructing the Initial Structure . . . 88
5.3.3 Solving the Optimization Problem . . . 89
5.4 Results . . . 89
5.4.1 Soap Films and Bubbles . . . 89
5.4.2 Tents and Membrane Structures . . . 91
5.4.3 Pneumatic and Cushion Structures . . . 92
5.4.4 Timing Information . . . 92
5.5 Summary and Discussion . . . 93

**6 Conclusions and Future Work** **95**

6.1 Better Initial Guesses . . . . . 96

6.2 Global Optimization . . . . . 97

6.3 Final Lessons . . . . . 98

# List of Figures


3.1 The environment of our sample spacetime problem . . . . . 21

3.2 Plots of the X and Y locations of the ball versus time . . . . . 27

3.3 Plot of the ball's trajectory in space. Tick marks on the trajectory occur every 1/30th of a second . . . . . 27

3.4 A graph of the countours of the objective function defined in Equation 3.20. The linear constraint on this function is shown as a thick black line. . . . . 29

3.5 A graph of the reduced gradient function defined in Equation 3.21. The value of this function is plotted on the vertical axis against the value of the remaining variable, $x_1$. 30

3.6 A two-dimensional pogo stick . . . . . 37

3.7 The two poses for the pogo stick's crouch task. The desired starting pose is shown on the left, and the desired final pose is shown on the right. . . . . 38

3.8 The Luxo lamp in its rest pose . . . . . 41

3.9 Initial motions for trials #1 through #4 (from top to bottom) of the Luxo Lamp forward leap task. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally. . . . . 43

3.10 Initial motions for trials #5 through #8 (from top to bottom) of the Luxo Lamp forward leap task. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally. . . . . 44

3.11 Final motions for trials #1 through #4 (from top to bottom) of the Luxo Lamp forward leap task, computed using finite-difference approximated gradients. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally. . . . . 45

3.12 Final motions for trials #5 through #8 (from top to bottom) of the Luxo Lamp forward leap task, computed using finite-difference approximated gradients. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally. . . . . 46

3.13 Final motions for trials #1 through #4 (from top to bottom) of the Luxo Lamp forward leap task, computed using automatic differentiation. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally. 49

3.14 Final motions for trials #5 through #8 (from top to bottom) of the Luxo Lamp forward leap task, computed using automatic differentiation. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally. 50

3.15 The biped in its rest pose . . . . . 51

4.1 A cantilever bridge generated by our software, compared with the Homestead bridge in Pittsburgh, Pennsylvania. . . . . 74

4.2 A cooling tower at a steel mill created by our software compared with an existing tower. From left to right: a real cooling tower, our synthesized tower, and the same model with the obstacle constraints shown. . . . . 75

4.3 From top to bottom: the data specified by the user (loads are depicted as green spheres and anchors as white cones); the free joints added by the software above the loads; the automatically generated initial connections (beams). This structure was the initial guess used to create the bridge shown in Figure 4.4. . . . . 76

4.4 A typical railroad bridge and similar truss bridge designed by our software. . . . . 77

4.5 A depiction of Euler buckling under a compressive load. . . . . 77

4.6 From left to right: initial structure, tripod solution, derrick solution. . . . . 77

4.7 A bridge with all trusswork underneath the deck. . . . . 78

4.8 A perspective and side view of a through-deck cantilever bridge. . . . . 78

4.9 Our trusswork tower, compared with a detail of the Eiffel Tower. Because they are ornamental and not structural, the observation decks are not included in our tower. . 79

4.10 Three types of roof trusses: from left to right, cambered Fink, composite Warren, and Scissors. Illustrations of real trusses are shown at the top of each column. The lower two trusses in each column were generated with our software. . . . . 79

4.11 From left to right: The initial structure from which we began the optimization, our final design, and an optimal Michell truss after an illustration in Michell [83]. The red sphere on the left of each image is an obstacle (on the surface of which are the anchors), and the green sphere on the right is the load which must be supported. Gravity points down. . . . . 79

5.1 A group of double and triple bubbles. . . . . 82

5.2 Our model of the membrane structure that forms the roof of the central exhibition space of the San Diego Convention Center, compared with a photo of the real structure, used courtesy of the San Diego Convention Center Corporation. . . . . 83

5.3 A soap film formed between two U-shaped wire boundaries. The initial geometry, an intermediate step in the optimization, and the final geometry are shown on the right. . . . . 84

5.4 The 1-ring neighborhood of $x_i$, consisting of the vertices $x_{i0}$ through $x_{i5}$. . . . . 86

5.5 The external angles of the edge connecting $x_i$ and $x_j$. . . . . 87

5.6 The voronoi region of of vertex $x_i$, for a the face formed with 1-ring neighbor vertices $x_{i0}$ and $x_{i1}$. . . . . 88

5.7 From left to right, a double and triple bubble. . . . . 89

5.8 The soap film formed by a cubical framework. . . . . 90

5.9 The five distinct, stable soap films bounded by an octahedral wire frame. Shown in order, from left to right, of least to greatest total surface area. . . . . 90

5.10 The top of a circus tent generated by our software, compared with a real tent. The lower, draped part of the tent could be modeled with existing cloth simulation techniques. Photograph © 2003 FreeFoto.com. Used with permission. . . . . 91

5.11 Our model of a pneumatic structure, compared with a real structure enclosing heated tennis courts. . . . . 92

5.12 A pneumatic exhibition space. Our model is shown on top, and a photograph of the architect's model of the proposed structure is shown below. Photograph © 2003 MIT Press. Reprint permissions applied for. . . . . 93

# List of Tables


3.1 Initial guesses at the crouching motion for the pogo stick . . . . . 38

3.2 Final results for the crouching motion for the pogo stick . . . . . 39

3.3 Initial guesses at the jumping motion for the pogo stick . . . . . 40

3.4 Final results for the jumping motion for the pogo stick . . . . . 40

3.5 Initial guesses at the leaping motion for the Luxo lamp . . . . . 42

3.6 Final results for the leaping motion for the Luxo lamp, computed using finite-difference approximated gradients . . . . . 43

3.7 Final results for the leaping motion for the Luxo lamp, using automatic differentiation 48

3.8 Initial guesses at the hopping motion for the biped . . . . . 52

3.9 Final results for the hopping motion for the biped . . . . . 53

3.10 Initial guesses at the leaping motion for the Luxo lamp . . . . . 54

3.11 Parameter values for initial guesses at leaping motion for the Luxo lamp . . . . . 55

3.12 Final results for the leaping motion of the Luxo lamp . . . . . 55

## Abstract

This thesis addresses the application of nonlinear optimization to three different problems in computer graphics: the generation of gait cycles for legged creatures, the generation of models of truss structures, and the generation of models of constant mean curvature structures.

We first present work on the automatic motion generation for legged creatures. Our technique is a reformulation of spacetime optimization, which has been used in the past to semi-automatically generate realistic, physically-based motion of simple articulated characters. Our approach poses the task of motion sythesis as the process of solving a large, constrained nonlinear optimization problem. The objective function of these problems is a measure of consumed energy, and the constraints are a combination of the laws of physics and a high-level description of the motion we wish to see. Our technique replaces the Newtonian constraints that previous techniques have used to enforce physical realism with a dynamic simulation, which makes the spacetime constraints framework more flexible and potentially more powerful.

We then present a method for designing truss structures, a common and complex category of buildings, using non-linear optimization. Truss structures are ubiquitous in the industrialized world, appearing as bridges, towers, roof supports and building exoskeletons, yet are complex enough that modeling them by hand is time consuming and tedious. We represent trusses as a set of rigid bars connected by pin joints, which may change location during optimization. By including the location of the joints as well as the strength of individual beams in our design variables, we can simultaneously optimize the geometry and the mass of structures.

In the third section of this thesis, we present a method for creating models of surface-area minimizing, constant mean curvature objects. Constant mean curvature objects, which include such diverse natural and man-made structures as thin-film membranes, sails, pneumatic structures, and soap bubbles and films, are both common and often difficult to create by hand. Using techniques of constrained non-linear optimization, we can automatically generate models of these structures, typically minimizing surface area while maintaining a constant mean curvature over the whole surface in addition to volume or other geometric constraints.

We conclude this thesis with a discussion of the advantages and shortcomings of optimization as a technique for solving modeling and animation problems in computer graphics.

# Chapter 1

# Introduction

It is universally acknowledged that generating realistic models and motion for computer animation is difficult. Typically, these tasks are done by hand, requiring a great deal of work by skilled artists. Achieving realism can be particularly difficult not only because of the exacting nature of artistic pursuit, but also because humans are very adept at spotting flaws in otherwise visually appealing scenes or motions. Increasingly, computer animators have been turning to automatic and semi-automatic techniques in order to aid them in modeling and animation tasks. Automatic techniques are attractive not only because they can reduce the amount of work an animator is required to do, but also because they can help artists avoid subtle mistakes or inaccuracies that may cause the viewer to lose their willing suspension of disbelief.

The best example of these kind of automatic modeling and animation techniques are simulation tools. Physical simulation has been successfully used to model many complex physical phenomena, such as brittle fracture [89], smoke and opaque gases [32], fire [87], and water [134, 34]. Because the output of a physical simulation is related to the input in an extremely complex, nonlinear way, simulation techniques such as these work best in situations where realism is more important than control. At the other end of the spectrum are tools such as inverse kinematics and function interpolation, which allow animators to quickly express themselves, but do not assist the animator in achieving realistic results.

Ideally, computer animators and modelers would like tools somewhere in between these two extremes; tools that add realism (or aid in achieving it) but also give the user control over the system being modeled. Some very successful research has been done in this middle ground between realism and control. For example, Prusinkiewicz's work in modeling trees and herbaceous plants [107, 106], uses high-level abstractions of natural growth that allows artists to change intuitive parameters that affect the final shape and form of the plants. Parish and Müller [95] use similar techniques (L-systems) in order to generate realistic models of cities, using geographic and demographic maps as

input. In this thesis, we examine another general technique — nonlinear optimization — that offers the potential of allowing a user to easily create realistic objects and motion, while retaining a great deal of control.

## 1.1 Problem Statement

*There exists a need for automatic tools which can aid animators and modelers in designing realistic and complex character motions and models of objects. Compared to current techniques, such tools would allow these artists to work more efficiently and to create objects and motion that are physically realistic.*

## 1.2 Scope

Constrained nonlinear optimization is a numerical method for finding the best solution among many that satisfies some set of user demands. We apply this technique to three problems in computer graphics. The first of these three applications is the generation of physically realistic motion for articulated characters. We use the "spacetime constraints" approach to motion generation, in which the motion of a character is optimized to fit certain constraints ("be in this position at this time") while minimizing the amount of energy the character spends.

We then explore two different modeling problems. First, we examine the task of using nonlinear optimization to generate physically realistic models of truss structures, a common type of building. To generate these structures we formulate the design problem as one of minimizing the total mass of the truss, while maintaining certain user-specified structural design elements, such as the location of connections to the ground and the ability to support a certain load. For the second modeling task, we use optimization techniques to generate models of constant mean curvature objects, such as soap bubbles and tensile membranes. These objects are represented as triangulated meshes which seek to minimize their surface areas, while maintaining a constant mean curvature across the surface, as well as other user-specified geometric constraints.

We conclude this thesis with a discussion of how optimization worked, or failed to work, on our three computer graphics problems and derive some lessons for future researchers.

The specific contributions of this thesis to the field of computer graphics are

- A demonstration that optimization is a useful technique for modeling complex, physical structures.
- A demonstration that optimization can capture physical detail and at the same time give the user a great deal of control over the objects being modeled.

- A new, reduced gradient formulation of spacetime optimization that is more flexible than previous formulations.
- A discussion of when optimization is an appropriate tool for modeling and animation in the field of computer graphics.

# Chapter 2

# An Introduction to Optimization

Optimization is the field of study concerned with finding the best solution to problems that have many possible answers. The mathematical investigation of optimization problems dates back to at least the mid-17th century, when both Fermat and Newton proposed a method for finding the optimum of a single variable function. Their heuristic was to find the point $x^*$, at which the value of the function is the same as the value of a closely neighboring point, $x^* + dx$, where $dx$ was a tiny value (then called a "virtual displacement"). Although greatly improved by the development of calculus, their observation that the (locally) best solution to an equation of one variable will lie in the region where its derivative is zero is still fundamentally true.

However, not all optimization problems are simple equations of one variable. Similarly, sometimes we are interested in finding the best *possible* answer and not merely the best answer in a particular region. We also often have particular demands about the form of the solution, called "constraints," that must be satisfied for a solution to be acceptable. Finding answers to these kind of problems is where the science, and frequently the art, of optimization lies.

In this chapter, we do not intend to give an exhaustive survey of optimization and optimization algorithms. Excellent summaries of the field can be found in Gill, Murray and Wright's book [135] as well as many others. Instead, we wish to give a quick overview of optimization along with some observations on how it has been used in the field of computer graphics.

## 2.1 Unconstrained vs. Constrained

One of the most fundamental divisions that can be made among optimization problems is the presence of *constraints* on the problem. Constraints are conditions that must be satisfied by an answer in order for it to be acceptable. These constraints can be simple bounds on a variable, such as "$x > 5$", or can be complex, nonlinear functions of many different variables.

### 2.1.1 Unconstrained Optimization

Unconstrained optimization problems have no constraints on acceptable answers, but instead consist solely of a function that is to be minimized (or maximized). Unconstrained optimization problems are written as:

$$\min \qquad G(\vec{q}) \tag{2.1}$$

where $G$ is our *objective function*, the function that we are seeking to minimize. $\vec{q}$ is the vector of the *optimization variables*, the variables which the optimization algorithm is allowed to change in order to decrease the objective function.

A common type of unconstrained optimization problem is "least squares fitting." Least squares techniques attempt to minimize the sum of the squares of the difference between a set of data, $\vec{y}$, and a set of curves, $f$, parameterized by $\vec{q}$:

$$\min \qquad G(\vec{q}) = \sum [y_i - f(q_0, ..., q_{N-1})]^2 \tag{2.2}$$

The sum of the squares of the differences are used instead of the absolute values of the differences because this allows the residuals to be treated as a continuous differentiable quantity. If the curves, $f$, are linear we can calculate the parameters with some simple determinants. Nonlinear least squares fitting, such as that used in Gleicher's motion warping work [41] and Welch and Witkin's paper on variational modeling [130], is usually accomplished by iteratively applying a linearized form of the function, $f$, until convergence is achieved.

Other frequently used methods for solving unconstrained optimization problems are gradient descent, and the more sophisticated conjugate gradient descent [115]. Simple gradient descent works by computing the gradient of the objective function, descending that slope a small amount, and then recomputing. Conjugate gradient descent is similar in nature, but descends the conjugate directions of the function instead of its gradient, which results in far fewer steps to convergence for poorly conditioned problems.

### 2.1.2 Constrained Optimization

Unsurprisingly, constrained optimization problems are generally more difficult to solve than unconstrained ones. Reflecting this difficulty, there are many specialized methods for solving constrained optimization problems, the choice of which depends on the nature of the specific problem's objective function and constraints. For example, problems with linear objective functions and linear constraints are often solved with simplex methods, which search the edges of the constraint "solid" to find the best solution. The problems that we investigated for this thesis all consisted of a non-linear objective function with mixed linear and non-linear constraints. Thus, in this section we will

focus our attention on a selection of methods for solving this particular type of constrained nonlinear optimization problem.

### Penalty Methods

One of the most popular approaches to solving constrained optimization problems is to transform them into easier-to-solve unconstrained problems, using *penalty methods*. Penalty methods operate by adding scaled values of the constraint functions, $C(\vec{q})$, to the objective function. For example, a constrained problem with $M$ constraints (which, with no loss of generality, we assume to be equality constraints):

$$\begin{array}{ll} \min & G(\vec{q}) \\ \text{s.t.} & C_0(\vec{q}) \quad\quad = 0 \\ & \cdots \\ & C_{M-1}(\vec{q}) \;\; = 0 \end{array} \tag{2.3}$$

is transformed with a penalty method into an unconstrained problem:

$$\min G^*(\vec{q}) = G(\vec{q}) + \sum_{i=0}^{M-1} \gamma_i C_i^2(\vec{q}) \tag{2.4}$$

The scaling factors $\gamma_i$ are called *penalty terms* and may be static or change in value as the optimization algorithm proceeds.

An obvious disadvantage of penalty methods is that adding scaled constraint functions to the objective function makes it and its gradients far more complex. Furthermore, penalty methods are guaranteed to be ill-conditioned as the penalty term increases, resulting in long, narrow "valleys" in the optimization terrain which can slow convergence. However, because unconstrained problems are much easier to solve than constrained ones, penalty methods continues to be widely used, for example in Lee and Shin's motion editing work [73].

### Sequential Linear Programming

A second class of techniques for solving nonlinearly constrained optimization problems is sequential linear programming (SLP). Sequential linear programming techniques are iterative; that is, they operate by solving a series of simpler subproblems (in this case, linear subproblems) which approximate the main problem.

One popular implementation of this technique is the *projected Lagrangian method*, implemented in the MINOS software package [85]. In this method, the subproblems' constraints are the true linear constraints (if any) of the main problem, combined with linearizations of the nonlinear constraints.

The subproblem's objective function is an augmented Lagrangian function, $G_L$. Using the notation established for equation 2.3, this function can be written as:

$$G_L(\vec{q}, \vec{\lambda}) = G(\vec{q}) - \vec{\lambda}^T \vec{C}(\vec{q}) + \vec{\lambda}^T A\vec{q} \tag{2.5}$$

where A is the Jacobian matrix of *active constraints* with respect to the variables. Active constraints are those constraints on the problem that are currently being violated. $\vec{\lambda}$ is the vector of the Lagrange multipliers.

Sequential linear programming techniques can yield good convergence characteristics if the original objective function, $G$, is not very nonlinear. A difficulty with these methods, however, is the need to estimate the Lagrange multipliers at each iteration. Moreover, the set of active constraints may change between iterations, especially in the case of inequality constraints.

#### Sequential Quadratic Programming

A third class of techniques for solving nonlinear optimization problems is sequential quadratic programming (SQP). Similar to sequential linear programming, SQP operates by iteratively solving a series of simpler approximations to the true problem. Although both techniques use linear approximations of the active constraints, SQP uses quadratic approximations of the objective function. Within the family of SQP techniques there is a further distinction between those algorithms which use true Hessians (Newton methods) and those which use approximations of the Hessian (quasi-Newton methods). The former is more accurate, but the latter techniques often have better initial convergence properties [135]. Sequential quadratic programming seems to be the most popular technique for solving constrained optimization problems in computer graphics. It has been used in the vast majority of spacetime optimization work, including Witkin and Kass's original paper [132], Cohen's work on spacetime [26], Rose and his colleagues' work on motion transitions [112] as well as Popović's thesis [103]. SQP has also been used by Tak and his colleagues in order to correcting unbalanced motion while preserving motion characteristics [122], as well as in Gortler and Cohen's work on variational modeling with wavelets [44].

## 2.2 Global vs. Local

A common property of all the nonlinear optimization methods we have so far discussed is that they are all "local" methods. Local optimization methods are only capable of finding the best answer that is "near" some initial starting point. In general, nonlinear problems will have many solutions, called *local minima* that are locally optimal but not globally optimal. Consequently, local optimization techniques such as penalty methods and sequential quadratic programming can be quite sensitive to the initial values of the optimization variables.

However, certain optimization techniques, called *global methods*, are intended to find the absolute best possible solution to an optimization problem. Global optimization has been heavily researched (see Horst and Pardalos[57] for a fairly up-to-date reference to the field), but very few general techniques exist, and the state of the art is much less advanced than for local methods. Most global optimization techniques, such as simulated annealing, genetic programming, and clustering techniques, have been developed for unconstrained problems. However, most of these methods for unconstrained problems have been adapted to handle constraints, usually with penalty or barrier methods.

### 2.2.1 Simulated Annealing

Simulated annealing is a relatively recently developed technique for global optimization[69], which has become popular for solving problems with discrete and, more recently, continuous variables. Simulated annealing, as one would guess from its name, is based on an analogy to the thermodynamic process of annealing. It has been known for centuries that if a liquid (for example, molten iron) is rapidly cooled, the molecules do not have time to seek out the minimum energy configuration (a crystal). However, if the metal is gradually cooled, the molecules will lose their thermal mobility more slowly and have time to line up into crystalline arrays with a ordered structure billions of atoms across. Simulated annealing applies this analogy to the problem of global minimization: if we descend the objective function too quickly (using a "greedy" method) we risk finding ourselves trapped in a local minimum. However, if we stochastically choose at every iteration whether to take a step that decreases the objective, or whether to take a step that increases it briefly, there is a better chance that we will find the global minimum over the long term. In the field of graphics, simulated annealing has been used by Grzeszczuk and Terzopoulos [48] to learn low-level controllers and higher-level behaviors for the locomotion of snakes and fish. Hodgins and Pollard [56] used this optimization technique in order to tune a small number of parameters in order to adapt motion to new characters. Kaplan and Salesin [67] used simulated annealing in order to solve "Escherization" problems of creating regular tilings from irregular images.

### 2.2.2 Genetic Programming

Sometimes we are interested in finding a global solution for a problem that has no easy, closed-form objective or constraint functions, or for systems where the relationship between these functions and the optimization variables is unclear or poorly understood. In cases like these, so-called "evolutionary algorithms" are often applied. Evolutionary algorithms, sometimes called genetic algorithms or genetic programming, are stochastic search methods that mimic the metaphor of natural biological evolution [42] and are often used to search large, multidimensional spaces.

Evolutionary algorithms operate by first creating a large population of potential solutions to the optimization problem. At each iteration, or "generation," the algorithm creates a new set of potential solutions by selecting individuals according to their level of fitness in the problem domain and "breeding" them together using operators inspired by natural genetics. This process gradually results in the evolution of populations of solutions that are better than the individuals from which they were created.

Genetic programming has been used successfully in several computer graphics applications. Ngo and Marks [86] used evolutionary techniques to find parameters for a stimulus-response controller for simple, two-dimensional articulated creatures. Similarly, Van de Panne and Fiume [78] used genetic programming in order to generate the nonlinear weights of a sensor-actuator network in order to control two-dimensional creatures. Shortly thereafter, Sims applied genetic algorithms for the more difficult task of generating both the physical structure as well as the motion controllers for creatures in virtual worlds [117].

# Chapter 3

# Gait Synthesis

Despite their striking visual differences, the techniques of character animation in both traditional "pen and ink" animation and computer animation are fundamentally quite similar. In both styles of animation, characters — by which we mean any expressive, dynamic object such as humans, animals, or singing top-hats — are usually animated by hand, while automatic techniques such as motion capture and rotoscoping remain an uncommon, but growing, alternative.

The standard technique of animating characters by hand is called *key framing*. Key-framing requires the lead animator to draw only a few frames per second called "key frames". These frames define the most important poses of the characters: key moments of action or emotional expression that define its personality. The remaining frames of the animation (for a total of 24 per second) are then interpolated between these key frames, and drawn by artists known, for obvious reasons, as *in-betweeners*.

Similarly, the standard technique of computer animation requires that an animator specify only the key frames of the motion of a character. However, instead of human in-betweeners, software interpolates between these important poses and creates the in-between frames automatically. Obviously, the pose of a character is specified much differently in computer animation than in traditional animation. Instead of drawing a sketch of the character, a computer animator specifies the values of the character's *degrees of freedom* (DOF's): the position of the character in space and the angles of its various joints. Once the DOF's of a character have been key-framed, in-betweening becomes a problem of function interpolation.

## 3.1 Using Computation to aid Animation

However, the direct control of the degrees of freedom of a character can be tedious and non-intuitive, especially for complex, articulated characters such as humans and animals. In order to reduce

the amount of effort required to produce complex animations, computer graphics researchers have explored a large number of automatic and semi-automatic motion synthesis techniques. In this section, we offer a brief review of this literature and describe where among this body of work our technique fits.

In the following sections, we organize the motion synthesis literature by the techniques that the authors used; controller based techniques are grouped together, as are methods that use motion capture, and so forth. We then sort these groups according to how much "knowledge" they require in order to generate plausible legged locomotion. We have loosely defined "knowledge" as information about the way a specific character moves or the mechanics of legged locomotion for a particular class of creatures (for example, bipeds). This knowledge can be used as part of the input to the technique, such as the human motion capture data used by motion warping techniques, or it can be incorporated into the algorithm itself, such as the knowledge of gait cycles that is programmed into motion controllers. "High knowledge" systems tend to be created for very specific characters and tasks and are difficult to apply to other types of problems. In contrast, "low knowledge" systems, because they contain less domain-specific information, are more general and can be more easily applied to generating legged locomotion for different characters.

### 3.1.1 Controllers

Until very recently, the most common "high knowledge" approach to generating legged locomotion has been the engineering technique of *control algorithms*. Control algorithms (or "controllers") can be thought of as sets of equations that use the observed state of a system to generate *control signals* which then directly or indirectly control the evolution of that system. For example, the cruise control in automobiles is a simple controller that applies gas or the brakes in proportion to the difference between the current and the desired speed. Legged locomotion is much more complicated than speed control, however, and the control algorithms are proportionally more complex. Often, the controllers for legged creatures are represented as a finite state machine, where each state (defined by properties of the creature and the world) has its own goals and rules for applying forces and torques in order to satisfy them. Control algorithms have a long history in engineering but were slow to enter the field of computer graphics. largely because the dynamic systems interesting to animators are hard to construct and were expensive to simulate.

In the late 1980's, Brudelin and Calvert [22] used a dynamic model and a controller in order to generate the leg motions of a walking human. After the leg motions were computed, kinematic (not dynamic) motions of the feet and upper body were computed, using oscillatory patterns observed in real human motion. Raibert and his colleagues at CMU and the MIT leg lab did work on creating legged robots that could balance, walk and dynamically run [109]. Adapting their earlier work with laboratory robots to the field of animation, Raibert and Hodgins [108] developed controllers for a

biped, a quadruped and a planar monopod that could walk, run and hop. However, their control algorithms had to be written specifically for each creature, and further tuned for new motions.

Another early example of using controllers to generate legged locomotion for animation is the 1990 paper by Pai and his colleagues [94], wherein they used a dynamic model of the human body driven by hand-built controllers to generate human walking. An interesting aspect of this approach was that their model was programmed with high-level constraints and commands such as "keep the torso vertical" and "lift and place a foot." However, these constraints were available only on certain parameters and could interfere destructively with one another.

Hodgins [64] presented a technique for animating men and women performing various sports which also combined control algorithms and the dynamic simulation of the athletes' bodies. Van de Panne and his colleagues [79] introduced a two-level architecture for gait control, consisting of a high-level state machine called the *pose control graph* and a set of low-level controllers. Transitions between states was realized by the simple controllers on each joint, which computed the required forces and torques. He later presented a method which used control mechanisms without sensors, but with a limited parameterization that allowed the generation of several common periodic gaits [125].

Laszlo and his colleagues [65] also used control algorithms — specifically, finite state machines combined with simple controllers — coupled with sensory feedback to produce walking motion in bipedal characters. Similar combinations of controllers with simple sensor-actuator feedback systems have been described by van de Panne and his co-authors in 1990 [80] and 1993 [78].

A common drawback among all of these controller-based methods is that, while they can produce realistic-looking motion, the controller programs must be written and adjusted by hand for each distinct motion of each character. In order to make these techniques more automatic, Ngo and Marks [86] used genetic programming in order to perform a global search to solve simple, two-dimensional legged motion problems. Although mislabeled as spacetime optimization, this technique is controller-based: the variables that the optimizer could change were the parameters of a set of stimulus-response behavior rules that governed the behavior of the characters. Their technique generated novel and interesting gaits for simple characters, but did not generate "realistic" motion, and would not scale well to more complex, three-dimensional characters.

In a 1997 paper, van de Panne [126] suggested a novel approach to the synthesis of gaits. Given the position and timing of footprints, he argues, one can generate the motion of a bipedal character. This method is far from automatic, though, and requires a great deal of information about the gait to be specified. Recently, Lo and Metaxas [76] have developed a method of generating dynamically correct human motion using optimal control, coupled with a very fast method of recursive dynamics. Results, however, are preliminary and appear to lack the realism of those produced with other methods.

### 3.1.2 Motion Capture and Retargeting

Another family of high-knowledge techniques is based upon editing a high-quality recording of real human or animal motion. This input, generally called *motion capture data*, is gathered by either optically or magnetically tracking special tags attached to key points on the actor's body [28]. As opposed to controller-based motion generation, where the knowledge is algorithmic, the knowledge in motion capture based techniques lies in the input to the algorithms.

In 1995, Witkin and Popović [133] proposed a technique for editing pre-existing motion, such as motion capture data or keyframed motion, by geometrically warping the curves of the motion representation parameters. Using their method, the user defines a set of keyframes which act like constraints on the desired motion, in order to generate a smooth deformation of the original parameter curves. Although flexible and fast, their technique did not contain any deep knowledge of legged locomotion and thus the warped motion looked unnatural, especially when the warping was extreme,

That same year, Bruderlin and Williams [23] applied techniques from image and signal processing to the same problem of transforming and editing motion capture data. Using techniques such as multiresolution filtering, motion warping and displacement mapping, they were able to modify and combine pre-captured motion data with only slight computational expense. However, similar to Witkin and Popović's work, only small transformations to the original motion could be made before significant realism was lost.

Gleicher [41] used optimization to adapt high quality motion from one articulated character to another with the same topology, but different limb lengths. His optimization-based method minimized the change required to retarget motion signals, while maintaining the user-specified constraints which capture the motion's basic features. Because the cost function his technique sought to minimize was based on similarity to the input, the initial motion had to be of very high quality, and fairly close to the desired output.

Lee and Shin [73] used a hierarchical representation for motion curves combined with an optimization-based inverse kinematics algorithm, in order to alter high-quality input to meet constraints on the motion. They were able to warp walking and roped climbing motions to take into account turning, changing terrain, changing character shapes, as well as generating smooth transitions between different motion clips. Their technique differed from Gleicher's work [41] in that Lee and Shin made a distinction between relationships of configurations *within* a frame and relationships of configurations *between* frames. This change in formulation created smaller, although more numerous, optimization problems, which are easier and faster to solve. Tak and his colleagues [122] also applied optimization techniques to warping motion capture data. Their work, however, was focused on the narrower problem of correcting balance errors in dynamic motion, such as walking.

Recently, work has been done by Zordan and Hodgins [138] on modifying motion capture data to interact with dynamic simulations. Their work used a control system that followed the trajectories of motion capture data, while dynamically maintaining balance. Additionally, a small library of motion sequences was modified in order to accomplish dynamic tasks, such as throwing a punch or swinging a tennis racket. The control system reacted to any forces generated by these actions (and the collisions they entailed) by adjusting parameters within the motion controller.

### 3.1.3 Keyframing

The prototypical "low knowledge" approach to generating character animation is simple keyframing and function interpolation. Keyframing can be used to animate any dynamic characteristic of any object, and thus is extremely flexible. However, the keyframing paradigm itself contains no knowledge of what is appropriate or realistic motion; it depends entirely on the skills of the animator. In order to aid animators, keyframing techniques are frequently combined with *inverse kinematics*. Inverse kinematics, researched for decades in the field of robotics [27], allows the animator to specify the position of a character's hand, for example, and from this input calculate the values of the degrees of freedom (the angles of the wrist, elbow, shoulder and so on) that are required to achieve this position. This technique is far more intuitive than setting the DOF's by hand, but still leaves to the animators the problem of exactly specifying the motion at the key frames

### 3.1.4 Forward Simulation

Creating motion for characters in order to tell a story is difficult enough on its own. If animators are also interested in generating physically realistic motion, the task of creating this motion by hand becomes much more difficult. However, because macroscopic physical laws are well understood and lend themselves to mathematical solution, we can use a physical simulation to aid the animator in her quest for realism. These *forward simulation techniques* [8] have been proposed to automatically generate the motions of characters or complex object, such as water and cloth. These techniques are extremely realistic but are difficult to control. Initial conditions must be set very precisely to get the desired final behavior, and interaction with the animated objects is typically done via applied forces and torques, a very non-intuitive control method. Popović and his co-authors [102] presented work on using multiple-shooting methods in order to allow the user to control the motion of simple rigid bodies by manipulating the output and intermediate frames of a physical simulation. However, this work has not been extended to actuated objects such as articulated characters.

Because of the potential power of forward simulation, a great deal of work has been done on the problem of giving the animator higher-level or more intuitive control over the "inputs" to a simulation. These gait generation techniques are typically based upon a dynamic simulation of a

character interacting with an environment. As inputs, they require a mechanical specification of the character, including its linkages and muscles, and a high-level set of goals that the generated motion must satisfy. No prior knowledge of what the goal motion should look like is required. Sims [117, 116] used genetic algorithms to simultaneously evolve both the structure of virtual creatures and their gaits, which were codified as a control algorithm. This technique produced novel creatures that could complete simple goals, such as swimming towards a point or leaping into the air, but was extremely slow and was not suitable for generating realistic gaits of animator-specified characters.

### 3.1.5 Spacetime Constraints

Spacetime optimization, introduced in 1988 by Witkin and Kass [132], was an attempt to combine the intuitive control offered by inverse kinematics and key-framing with the automatic generation of motion provided by dynamic simulation methods. In the spacetime framework, the user specifies high-level *constraints* on the character that describe what the character should be doing (such as "walk across the room"), an *objective function* that describes how the character should do it (such as "minimize the amount of energy spent"), and the *physics* of the character: the geometry, mass and structure of the character and the physical laws of the environment in which it exists. Combined, the constraints, objective function and physics define a non-linear constrained optimization problem. The solution to one of these optimization problems is a set of functions that describe how the degrees of freedom of the character change through time. Spacetime optimization is a very general technique, applicable to any type of dynamic character, although traditionally it has been applied to only a small selection of legged and non-legged characters.

Witkin and Kass used spacetime optimization to generate jumping motion for a simple Luxo lamp. In order to make constructing the optimization problems easier, they discretized over time the variables of the system (the location of the Luxo's foot, the force its leg exerted, etc.) as well as the equations of motion. By using such a simple representation, however, they lost valuable gradient information and had to use the technique of finite differences to calculate gradients of the objective and constraint functions. In 1992, Cohen [26] developed a three-dimensional animation system based on the spacetime paradigm. In addition to using a continuous representation for the variables that enabled calculation of true gradients and Hessians of the spacetime equations, this system allowed the user to interactively guide the simulation, speeding convergence.

The next major advance in spacetime constraints came from Liu, Gortler and Cohen [75]. In their paper, they proposed a multi-grid approach, using B-spline wavelets as their functional basis, and applied the compiler design technique of common sub-expression elimination to reduce the complexity of the mathematical expression to be evaluated. Popović [104, 103], applied spacetime constraints to the problem of motion transformation. Although he demonstrated that simplification of the character's topology could result in dramatic speed-ups, his method for solving spacetime

optimization problems was not significantly different than that used in Cohen's earlier paper.

Liu and Popović [74] used a novel, stripped-down version of spacetime optimization in order to synthesize highly dynamic motions from a keyframed initial guess. Their implementation of spacetime optimization did not use a full formulation of physics, but instead worked with the patterns of angular and linear momentum, using a model of how momentum is transferred during contact with the ground.

Most recently, Fang and Pollard [31] presented a version of spacetime optimization wherein the equations of motion were reformulated in order to facilitate inexpensive gradient computation. Using their technique, computing first derivatives of the equations of motion has a cost that scales linearly with the number of degrees of freedom of the character. This technique is a significant improvement over finite difference and even analytic techniques, where the computation cost scales as the square of the number of degrees of freedom.

## 3.2 The Biology of Legged Locomotion

A great deal of attention has been paid to the subject of legged locomotion in the zoological and biomechanical literature. It is beyond the scope of this thesis to give an exhaustive account of this work, but a few papers and concepts that have become important in animation and computer graphics may be noted.

One biomechanical concept that has become important within animation is the notion that mammalian gaits and motion, including those of humans, minimize forces and energy expenditure [3, 5, 131, 137]. This minimization principle has found its way, in a vastly simplified form, into animation practice, such as in the objective function formulations in spacetime problems.

The zoological and biomechanical literature also contains comparisons between various modes of locomotion (walking, galloping, pronking, etc.) and discusses under what physical circumstances a gait is altered. These comparisons of gaits have been made across many animal species [4, 38, 120], as well as for specific animals [35, 63], and have influenced how controllers are constructed for various gait cycles. In general, however, the biomechanical literature is more concerned with building detailed empirical models of the structure of animal legs and their gaits than with the problem of synthesizing gaits for new creatures.

## 3.3 A Closer Look at Spacetime Optimization

In the preceding sections we described the previous work and the current state of the art in the computer graphics literature for both gait synthesis and spacetime optimization. In this section, we

describe the spacetime constraint paradigm in more detail, our unique insights into the problem, and discuss the numerical techniques we used to solve these optimization problems.

Characters, as they have been defined in this thesis, can be considered to be dynamic collections of rigid, massy elements, called *links*, connected with rotational and prismatic joints. Characters have a finite number of kinematic degrees of freedom and a number of muscles, or *actuators*, connecting the links that make up their skeleton. The kinematic degrees of freedom in a character represent the angles of the rotational joints, or the extensions of prismatic joints, between connected links Actuators act as muscles, applying forces and torques between connected links. Typically, there is one actuator for each degree of freedom. These links, joints and actuators together define the character's *topology*.

As a character moves, the values of its degrees of freedom must correspondingly change; thus, we refer to the function of a DOF over time as $r(t)$, and the collection of all such DOF's of a single character as $\vec{r}(t)$. Similarly, the set of forces and torques exerted by the character's actuators are denoted as $\vec{F}(t)$. In the standard formulation of spacetime optimization, both the actuator forces and the values of the character's degrees of freedom are *optimization variables*; that is, they are the quantities that are varied in order to optimize our character motions for a particular task.

### 3.3.1 Constraints

When we set out to generate motion of a dynamic character, we generally have in mind some set of goals or tasks that we wish it to perform. These goals can be simple ("stand still without falling down"), complex ("jump into the air, catch a branch and climb the tree as fast as you can"), or somewhere in between ("walk across the room without hitting anything or falling down"). Setting aside the artificial intelligence issues of task planning, most character goals we are interested in achieving deal with motion: being at specific locations (sometimes with specific velocities) at specific times. In the context of generating motion with constrained optimization, these goals are called *pose constraints*.

Pose constraints, denoted as $C_p$, specify the position or orientation of the character or its parts at some specific time. For example, we might want our character to begin its motion seated in a chair and then rise to walk across the room. The state of being "seated in a chair" would be expressed as a (perhaps large) set of pose constraints on the system, specifying the location of the center of mass, the angles of the hips and knees, and so forth. The set of all pose constraints for a particular motion is written as a vector: $\vec{C}_p$.

In addition to user-defined pose constraints, the environment in which the character moves will impose its own physical limitations. In the traditional formulation of spacetime optimization, these limitations are called *mechanical constraints* ($\vec{C}_m$). Mechanical constraints are different from pose

constraints in that they encode the forces necessary to satisfy the constraint. For example, a common mechanical constraint is that the character's feet cannot pass through the floor. For this constraint, the floor would exert a workless force on the feet of the character to insure that they did not interpenetrate. Mechanical constraints also include any external forces exerted by the environment, such as wind or contact with other characters.

Conventional spacetime constraint problems also have a third and final class of constraints: the *Newtonian constraints*, $\vec{C}_n$. Roughly speaking, Newtonian constraints insure that $F = ma$ holds true for all degrees of freedom over time. This relationship must be enforced with constraints because, as mentioned earlier, spacetime optimization problems have traditionally been formulated with the degrees of freedom and the forces as independent optimization variables. Many researchers have noted that Newtonian constraints are by far the most complex type of constraint in spacetime optimization problems: "a great deal of difficulty with finding the spacetime solution stems directly from the fact that it is very hard to find a feasible starting point which satisfies all Newtonian constraints." [103, pp. 41] This observation becomes important for our reformulation of spacetime optimization problems in Section 3.5.

### 3.3.2 Objective Function

Even when given clear and specific goals, such as "walk down the hallway in ten seconds, without falling over," there will generally be an infinite number of ways for the character to satisfy them. In order to pick the "best" of these possible sets of motions, we must have some metric, called the *objective function* that the optimization system uses to rate them. The biomechanical literature gives us many possible metrics to choose from, depending on the type of animal and the task it is performing. Most animation researchers have chosen to base their objective function on a simple measure of total energy consumed by the character's actuators, approximated as the integral over time of the sum of the actuator forces and torques squared:

$$G(\vec{r}, \vec{F}) = \int_0^T \sum_{i=0}^{N_F - 1} F_i^2(t) dt \tag{3.1}$$

where $T$ is the length of the animation and $N_F$ is the number of forces and torques. To this energy consumption term, researchers have variously added weighted measures of balance, smoothness of motion, and static stability, as well as other factors.

Given an objective function and a set of constraints, we can write our non-linear constrained

optimization problem:

$$\begin{array}{lll} \min & G(\vec{r}, \vec{F}) & \\ \text{s.t.} & \vec{C}_p(\vec{r}, \vec{F}) & = 0 \\ & \vec{C}_m(\vec{r}, \vec{F}) & = 0 \\ & \vec{C}_n(\vec{r}, \vec{F}) & = 0 \end{array} \tag{3.2}$$

Because we are solving for a set of functions over time, namely $\vec{r}$ and $\vec{F}$, this optimization is a calculus of variations problem. To avoid the difficulty of solving for arbitrary functions, these functions are typically represented with some set of basis functions (e.g. B-splines or wavelets) transforming the optimization problem into one of solving for coefficients of the basis functions:

$$\begin{array}{lll} \min & G(\vec{q}) & \\ \text{s.t.} & \vec{C}_p(\vec{q}) & = 0 \\ & \vec{C}_m(\vec{q}) & = 0 \\ & \vec{C}_n(\vec{q}) & = 0 \end{array} \tag{3.3}$$

where $\vec{q}$ is the coefficient vector.

## 3.4 A simple example

In order to illustrate the structure of spacetime optimization problems, we will construct a simple example, work through the various constraint equations, examine the parameterization of $\vec{r}$ and $\vec{F}$, and finally solve it.

With no loss of generality, we will consider a two-dimensional character navigating around a largely empty environment. The character in this case is an "actuated rubber ball": a point mass that can apply forces only when in contact with the ground. The "world" the character navigates in is empty except for the floor and a 2 meter wall (see Figure 3.1).

The degrees of freedom of this character are its position in space over time:

$$\vec{r}(t) = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \tag{3.4}$$

When in contact with the ground (or, conceivably, the wall), the character can exert an arbitrary force against the surface:

$$\vec{F}(t) = \begin{bmatrix} F_x(t) \\ F_y(t) \end{bmatrix} \tag{3.5}$$

For this example, we use the simple objective function:

$$G(\vec{F}) = \int_0^T [F_x^2(t) + F_y^2(t)]dt \tag{3.6}$$

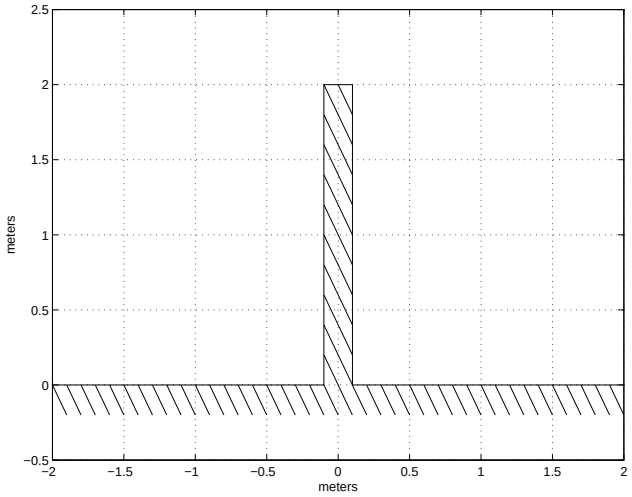


Figure 3.1: The environment of our sample spacetime problem

### 3.4.1 Pose constraints

After defining the character and objective function, we can define the task we wish our character to perform and set up the corresponding pose constraint functions. For this example, the character's task will be to start with zero velocity at the location $(-2, 1)$ at $t = 0.0$, bounce once on the left side of the wall, clear the wall by being above $(0, 2.0)$ at $t = 1.0$, and land on the far side of the wall at $(2, 0)$ at $t = 2.0$. The requirements are implemented as the following pose constraints:

$$\vec{C}_p(\vec{r}, t) = \begin{bmatrix} x(t = 0.0) & = & -2 \\ y(t = 0.0) & = & 1 \\ \dot{x}(t = 0.0) & = & 0 \\ \dot{y}(t = 0.0) & = & 0 \\ x(t = 1.0) & = & 0 \\ y(t = 1.0) & > & 2.0 \\ x(t = 2.0) & = & 2 \\ y(t = 2.0) & = & 0 \end{bmatrix} \tag{3.7}$$

### 3.4.2 Mechanical constraints

In order to construct the mechanical constraints and the Newtonian constraints, we must first determine when the character will be in contact with the ground. The times of contact must be known before we construct the problem because the equations that govern these constraints depend on whether the character is touching the ground or in free-fall. For example, consider a simple mass falling under the influence of gravity with no external forces. When this mass is in free fall, the

equation of motion is $\ddot{y} = G$, where $G$ is our gravitational constant. However, when resting on the ground, the forces acting on it will be in balance and thus its acceleration will be zero.

We define one mechanical constraint for each instant when the character is in contact with a surface. The first moment of contact happens when the character, dropped from its initial position at $(-2, 1)$ and hits the ground at $(-2, 0)$ at $t = 0.4518$. The second contact occurs when the character lands at $(2, 0)$ at $t = 2.0$. Assuming for simplicity's sake that our character collides with the floor inelastically and with infinite friction, the corresponding mechanical constraints are

$$\vec{C}_m(\vec{r}, t) = \begin{bmatrix} y(t = 0.4518) & = & 0 \\ y(t = 2.000) & = & 0 \end{bmatrix} \tag{3.8}$$

These contact constraints represent a mechanical interaction that involves the transmission of force between the character and the floor. Thus, in order to satisfy physics (the Newtonian constraints), this contact force must be calculated. We calculate these mechanical forces in a manner similar to Lagrange multipliers, by adding one new optimization variable, $\lambda_m^{(i)}$, for each force $i$ "created" by a mechanical constraint:

$$F_m^{(i)}(\vec{r}) = \lambda_m^{(i)} \frac{\partial C_m^{(i)}}{\partial \vec{r}} \tag{3.9}$$

The values of these new variables, and the forces they create, are solved for as part of the optimization process.

### 3.4.3 Newtonian constraints

The final set of constraints that must be defined for our example are the Newtonian constraints. Recall that these constraints express the physics of the system as a relationship between the independently represented forces and accelerations. The formulation of these equations change, however, depending on the whether the character is in contact with the ground or not. When the character is in free-fall, the Newtonian constraints are trivially

$$\vec{C}_n(\vec{r}, t) = \begin{bmatrix} \ddot{x}(t) & = & 0 \\ \ddot{y}(t) & = & G \end{bmatrix} \tag{3.10}$$

where $G$ is our gravitational constant, -9.8 meters/second-squared. When the character is in contact with the ground, however, the Newtonian constraints become more complex:

$$\vec{C}_n(\vec{r}, \vec{\lambda}, t) = \begin{bmatrix} F_x(t) = m\ddot{x}(t) \\ F_y(t) = m\ddot{y}(t) + mG + F_m^{(i)} \end{bmatrix} \tag{3.11}$$

where $F_m^{(i)}$ is the mechanical force from the appropriate mechanical constraint.

The Newtonian constraints raise a significant issue that we have not had to confront with the pose or mechanical constraints. Namely, how do we enforce constraints over a range of times? The standard solution is to chop the time interval of the problem into many "frames" per second and enforce the constraints at those times. If we choose to have 30 frames per second, our continuous Newtonian constraints become 120 discrete Newtonian constraints (30 frames per second, two seconds of animation, and two dimensions for each constraint). Expanding the constraints in this way, as well as using the appropriate formulation of the Newtonian constraints for frames of contact and free fall, we have:

$$\vec{C}_n(\vec{r}) = \begin{bmatrix} \ddot{x}(t=0) & = 0 \\ \ddot{y}(t=0) & = G \\ \ddot{x}(t=0.0333) & = 0 \\ \ddot{y}(t=0.0333) & = G \\ \quad \dots & \\ \ddot{x}(t=0.4333) & = 0 \\ \ddot{y}(t=0.4333) & = G \\ F_x(t=0.4518) & = m\ddot{x}(t=0.4518) \\ F_y(t=0.4518) & = m\ddot{y}(t=0.4518) + mG + F_m^{(0)} \\ \ddot{x}(t=0.4667) & = 0 \\ \ddot{y}(t=0.4667) & = G \\ \ddot{x}(t=0.5) & = 0 \\ \ddot{y}(t=0.5) & = G \\ \quad \dots & \\ F_x(t=1.9667) & = m\ddot{x}(t=1.9667) \\ F_y(t=1.9667) & = m\ddot{y}(t=1.9667) \\ F_x(t=2.0) & = m\ddot{x}(t=2.0) \\ F_y(t=2.0) & = m\ddot{y}(t=2.0) + mG + F_m^{(1)} \end{bmatrix} \tag{3.12}$$

Our objective function (equation 3.1) is an integral over the entire timespan of the animation, and must also be discretized. However, because our character can only exert forces when in contact with the ground we only need to concern ourselves with times $t = 0.4518$ and $t = 2.0$:

$$G(\vec{F}) = F_x^2(t=0.4518) + F_y^2(t=0.4518) + F_x^2(t=2.0) + F_y^2(t=2.0) \tag{3.13}$$

We assume that all contact events take the same amount of time, and thus have normalized this factor out of our objective function.

Thus, our full optimization problem is

$$
\begin{aligned}
\min \quad & G(\vec{F}) = F_x^2(t=0.4518) + F_y^2(t=0.4518) + F_x^2(t=2.0) + F_y^2(t=2.0) \\
\text{s.t.} \quad & x(t=0.0) = -2 \\
& y(t=0.0) = 1 \\
& \dot{x}(t=0.0) = 0 \\
& \dot{y}(t=0.0) = 0 \\
& x(t=1.0) = 0 \\
& y(t=1.0) > 2.0 \\
& x(t=2.0) = 2 \\
& y(t=2.0) = 0 \\
& y(t=0.4518) = 0 \\
& \ddot{x}(t=0) = 0 \\
& \ddot{y}(t=0) = G \\
& \ddot{x}(t=0.0333) = 0 \\
& \ddot{y}(t=0.0333) = G \\
& \quad \dots \\
& \ddot{x}(t=0.4333) = 0 \\
& \ddot{y}(t=0.4333) = G \\
& F_x(t=0.4518) = m\ddot{x}(t=0.4518) \\
& F_y(t=0.4518) = m\ddot{y}(t=0.4518) + mG + F_m^{(0)} \\
& \ddot{x}(t=0.4667) = 0 \\
& \ddot{y}(t=0.4667) = G \\
& \ddot{x}(t=0.5) = 0 \\
& \ddot{y}(t=0.5) = G \\
& \quad \dots \\
& F_x(t=1.9667) - m\ddot{x}(t=1.9667) = 0 \\
& F_y(t=1.9667) - m\ddot{y}(t=1.9667) = 0 \\
& F_x(t=2.0) = m\ddot{x}(t=2.0) \\
& F_y(t=2.0) = m\ddot{y}(t=2.0) + mG + F_m^{(1)} \\
& F_m^{(0)} = \lambda_m^{(0)} \frac{\partial C_m^{(0)}}{\partial \vec{r}} \\
& F_m^{(1)} = \lambda_m^{(1)} \frac{\partial C_m^{(1)}}{\partial \vec{r}}
\end{aligned}
\tag{3.14}
$$

### 3.4.4 The parameterized equations

The penultimate step in solving this optimization problem is to choose a parameterization for $\vec{r}(t)$ and $\vec{F}(t)$. For this example, we choose to use Hermite splines for their simplicity of implementation and their intuitive parameters. Hermite splines, which are described in more depth by Foley and his

coauthors[33], are piece-wise cubic functions with four parameters: the values of the end-points ($p_1$ and $p_2$) and their slopes ($\dot{p_1}$ and $\dot{p_2}$). Using this notation, the equation for a single Hermite spline is

$$\begin{bmatrix} x(t) \\ y(t) \end{bmatrix} = (2t^3 - 3t^2 + 1)\vec{p_1} + (-2t^3 + 3t^2)\vec{p_2} + (t^3 - 2t^2 + t)\dot{\vec{p_1}} + (t^3 - t^2)\dot{\vec{p_2}} \tag{3.15}$$

The choice of the number of Hermite splines used to represent the trajectories $\vec{r}$ and $\vec{F}$ can have important effects on the ability of the optimizer to find the best solution. Too few splines, and the representation may not be flexible enough to represent the correct answer; too many and the optimizer will stand a greater chance of becoming stuck in a local minima or failing to converge at all. For the current example, the constraints and the equations of state are both simple, so we choose to use only three Hermite splines to represent each trajectory. Because we want the splines to be $C^1$ continuous, the end-point of segment one will be the start-point of segment two, and so on. However, because our character bounces off the ground at $t = 0.4518$, the slope at that point will not be continuous. Thus, for a particular degree of freedom $r(t)$ the nine spline parameters will be $(\vec{p_1}, \vec{p_2}, \dot{\vec{p_1}}, \dot{\vec{p_2}}^-, \dot{\vec{p_2}}^+, \vec{p_3}, \vec{p_4}, \dot{\vec{p_3}}, \dot{\vec{p_4}})$. Our problem contains two trajectories that must be parameterized, the $x$ and $y$ locations of the particle, and thus we have 18 spline parameters. These spline coefficients are grouped together with the mechanical constraints' Lagrange forces into a single vector of optimization variables, $\vec{q}$.

Rather than evenly spacing the knots in time (i.e. at $t = 0.0$, $t = 0.6667$, $t = 1.3333$ and $t = 2.0$), we instead chose to place them at $t = 0.0$, $t = 0.4518$, $t = 1.0$ and $t = 2.0$ in order to better reflect the form of the problem. This choice, combined with our parameterization, allows some of the constraints to become drastically simplified. The pose constraints that occur at the beginning and end of the spacetime segment:

$$\begin{aligned} x(t = 0.0) &= -2 \\ y(t = 0.0) &= 1 \\ \dot{x}(t = 0.0) &= 0 \\ \dot{y}(t = 0.0) &= 0 \\ x(t = 2.0) &= 2 \\ y(t = 2.0) &= 0 \end{aligned} \tag{3.16}$$

can now be represented as direct constraints on our spline parameters:

$$\begin{aligned} p_1^{(x)} &= -2 \\ p_1^{(y)} &= 1 \\ \dot{p_1}^{(x)} &= 0 \\ \dot{p_1}^{(y)} &= 0 \\ p_4^{(x)} &= 2 \\ p_4^{(y)} &= 0 \end{aligned} \tag{3.17}$$

Similarly, the pose constraints at $t = 1.0$, midway through the animation:

$$\begin{aligned} x(t=1) &= 0 \\ y(t=1) &> 2 \end{aligned} \tag{3.18}$$

are also simple constraints on parameters:

$$\begin{aligned} p_3^{(x)} &= 0 \\ p_3^{(y)} &> 2 \end{aligned} \tag{3.19}$$

In general, however, constraints will not fall neatly on the knots of splines, and will have to be evaluated as a more complex function of the neighboring knots. The 120 Newtonian constraints must also be written as functions of the appropriate Hermite parameters. Because these constraints occur every 1/30th of second and our Hermite knots are not evenly placed in time, care must be taken when writing out the parameterized Newtonian equations.

### 3.4.5 Solution techniques

With a complete problem specification in hand, we can apply standard constrained optimization techniques to solve for a solution to our problem. Starting with Witkin and Kass, researchers have preferred Newtonian sequential quadratic programming (SQP) techniques for solving the spacetime optimization problems. The chief advantage of Newtonian SQP is that it converges quadratically when close to the solution. Unfortunately, this algorithm can have poor convergence properties when far from a local minimum[136]. Hence, some researchers prefer to use a quasi-Newton method, such as the Broyden-Fletcher-Goldfarb-Shanno (BFGS), which can have slightly better initial convergence properties. Liu, Gortler and Cohen[75] used this latter method while Popović[103] proposed to use a hybrid of the two. To solve the problem outlined above, we use the limited-storage quasi-Newton technique implemented in SNOPT[37].

### 3.4.6 Discussion of Sample Problem Results

We solved the sample spacetime problem described by Equation 3.14, using SNOPT. Our problem, with 24 variables and 129 equality and inequality constraints, took between 20 and 60 seconds to solve, depending on the initial guess and the accuracy requested from the optimizer. Nearly all of the runs converged to the correct answer, with only a few very poor initial guesses resulting in failure. The optimal solution is shown in Figures 3.2 and 3.3. The plot on the left in Figure 3.2 shows the value of the X coordinate over the course of the two second animation, and the plot on the right shows the Y coordinate. Figure 3.3 shows the trajectory of the ball through space.

As we have seen, even trivial spacetime optimization problems can create hundreds of nonlinear Newtonian constraints, and more complex characters in three dimensions can easily require thou-

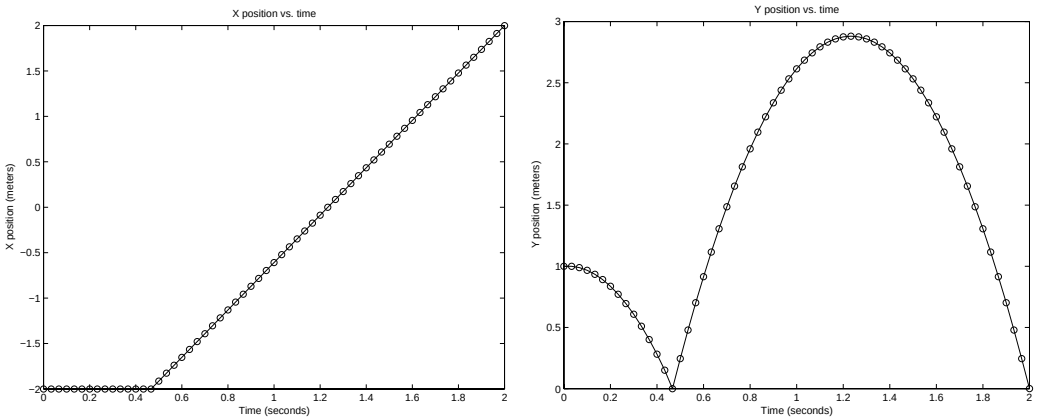


Figure 3.2: Plots of the X and Y locations of the ball versus time

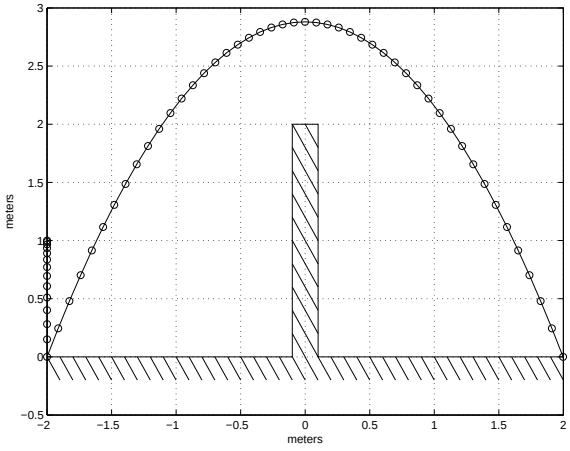


Figure 3.3: Plot of the ball's trajectory in space. Tick marks on the trajectory occur every 1/30th of a second

sands, even for brief animations. Problems such as these, with a large number of highly nonlinear constraints, pose difficulties for continuous optimization techniques.

An equally significant drawback of the standard technique for framing spacetime optimization problems, however, is that it requires the user to specify in advance the times when any part of the character will be in contact with the ground. In the example described above, we had to calculate ahead of time when the actuated ball would be touching the ground, and therefore able to push against it, and when it would be in the air, and therefore in free-fall.

For simple characters with a small number of degrees of freedom, this specification is usually not difficult. However, for complex articulated characters, such as a biped or quadruped, the specification of ground contact events can be very time-consuming. Furthermore, requiring the animator to decide when and where the feet of a character strike the ground can so highly constrain the answer that the "automatic" advantages of spacetime optimization are lost. This drawback becomes even more significant if we are interested in generating gaits for fantastic or extinct creatures. If no reference motion exists or the animator has no intuition about how a creature moves, the generated motion has no guarantee of being anything like how the creature would "really" walk or run. Ideally we would like to solve both of these problems — the number of Newtonian constraints and the required user-specification of footfall — at once, and therefore simplify the problem as well as making it more flexible.

## 3.5 A Reformulation of Spacetime Optimization

In the fields of optimization and optimal design, the *reduced gradient method* has become a common technique for dealing with equality constraints, such as our Newtonian ones, that arise from non-linear differential equations[2, 1]. The reduced gradient method aims to eliminate these complicated equality constraints, often called *state equations*, and the *state variables* associated with them, thereby reducing the dimensionality of the optimization problem.

This constraint elimination is accomplished by solving the state equations for the values of the state variables, given the current values of the degrees of freedom. The state variables are then used to evaluate the objective function and remaining constraints. In addition to reducing the dimensionality of the optimization problem, the use of a reduced gradient method relieves the optimizer from its unnatural role as an differential equation solver. This technique has found great success in the optimal design literature, having been used in designs constrained by Navier-Stokes flows [11, 36], compressible flow airfoil design[90, 92, 114] and structural optimization[91, 111].

To illustrate the reduced gradient method applied to constrained optimization problems, consider

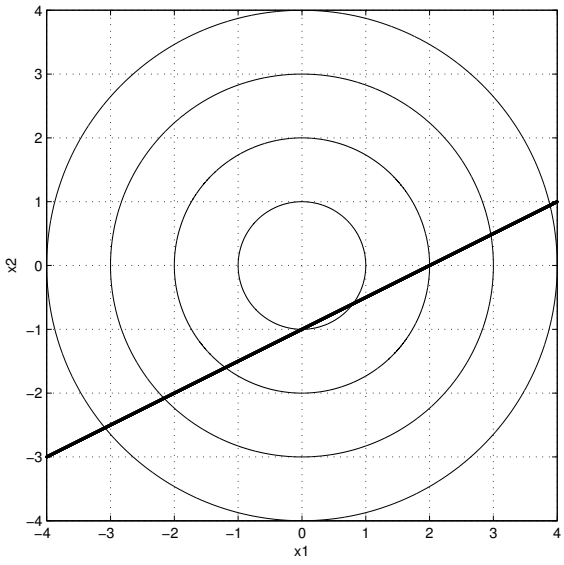


Figure 3.4: A graph of the countours of the objective function defined in Equation 3.20. The linear constraint on this function is shown as a thick black line.

the simple problem

$$\begin{aligned} \min \quad & x_1^2 + x_2^2 \\ \text{s.t.} \quad & x_2 = \tfrac{1}{2}x_1 - 1 \end{aligned} \tag{3.20}$$

The contours of the objective function, and the linear constraint imposed on it, are shown in Figure 3.4. Here, our "state equation" is the constraint $x_2 = \frac{1}{2}x_1 - 1$. We use this equality constraint to replace $x_2$ with $\frac{1}{2}x_1 - 1$ in the objective function, eliminating one of the variables in the system as well as transforming the problem into a more easily solved unconstrained problem:

$$\min x_1^2 + \left(\frac{1}{2}x_1 - 1\right)^2 \tag{3.21}$$

This function of one variable is shown im Figure 3.5.

This reduction in dimensionality and simplification of the problem comes at a price, of course. The objective function, and hence its gradient, has become more complex, and the state equation (the constraint) must be evaluated before the objective can be computed. In this case, the state equation is trivial to compute but in more complex systems these equations could be difficult to calculate, or the result of a time-consuming iterative process, such as a physical simulation.

### 3.5.1 Application of the Reduced Gradient Method

As mentioned in Section 3.3.1, one of the most significant obstacles to solving spacetime optimization problems is the number and complexity of the Newtonian constraints. These constraints, which

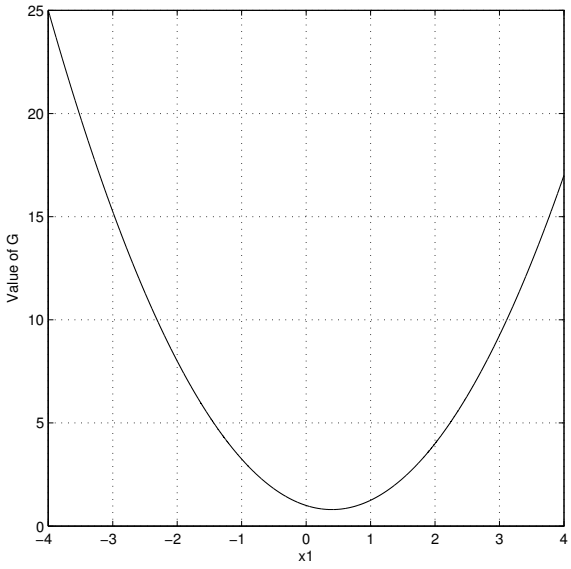


Figure 3.5: A graph of the reduced gradient function defined in Equation 3.21. The value of this function is plotted on the vertical axis against the value of the remaining variable, $x_1$.

attempt to enforce physical realism by constraining $F = m\ddot{x}$ at every frame in the animation, are ordinary differential equations that have been discretized in time. Because of their number, complexity, and the fact that they all result from the same computational process, Newtonian constraints are excellent candidates for elimination with the reduced gradient technique.

Applying the reduced gradient method to spacetime optimization problems is straightforward, but results in a significantly reformulated problem. Instead of attempting to maintain physical plausibility with Newtonian constraints (our state equations), we instead run a standard physical simulation. The inputs that drive this simulation are the forces and torques exerted by the character's actuators and the outputs (the state variables) are the positions and velocities of the character's component parts. The use of a physical simulation instead of Newtonian constraints simplifies the problem significantly, eliminating the position and velocity variables, $\vec{r}$, as well as both the Newtonian and the mechanical constraints:

$$\begin{array}{ll} \min & G(\vec{F}) \\ \text{s.t.} & \vec{C}_p(\vec{F}) \quad = 0 \end{array} \tag{3.22}$$

The objective function, $G(\vec{F})$ remains a simple function of the forces and torques, and does not require any modification. Pose constraints, which are constraints on positions and velocities, are now evaluated by simply checking the appropriate physical quantity at the indicated times. Each time we change the optimization variables of the system (the forces and torques of the actuators), we re-run the simulation in order to re-evaluate the constraints.

In addition to eliminating Newtonian constraints, a simulation-based reduced gradient formulation has the additional benefit of freeing the animator from having to completely specify footfall and other ground contact events. Recall that traditional spacetime optimization problems require explicit equations of motion to be declared at every frame of the animation. Because the equations of motion depend on what parts of the creature, are in contact with the ground, declaring these equations requires us to specify ground contact events. However, if we use a simulation to generate our physically correct motion, and thereby eliminate the Newtonian constraints, the animator will no longer be required to specify footfall. This change in the formulation of spacetime optimization allows us to investigate and generate motion with less foreknowledge of the results.

### 3.5.2 The Physical Simulation

Our characters are collections of rigid elements, and interact with a simple world composed of similarly rigid objects. Thus, the type of simulation we are interested in performing is called *rigid body simulation*. Rigid body simulation has a long history (see, for example, Goldstein [43] or Baraff [9] for a survey of the relevant literature) and is, for all practical purposes, a "solved" problem. Rather than implementing a constrained rigid body simulator ourselves, we used the commercial simulation software package SD/FAST.

SD/FAST takes a high-level description of a mechanism composed of linked rigid bodies, like our characters, and generates the C code which will compute the dynamics. Although it creates correct and efficient code for the dynamics, SD/FAST explicitly does not handle self-collisions or contact with the ground. Instead, these aspects of the simulation must be handled by code written by the user. We chose to ignore self-collision for our characters, not only because of the added computational expense but also because, for our simple characters and motions, self-collision was not considered to be a significant "threat" to the realism of the results. In order to simulate contact with the floor or other similarly immobile objects, we applied two techniques with varying success.

Our first implementation of a collision-resolution mechanism was to impose non-linear constraints on the physical simulation (*not* on the spacetime optimization problem) which forbid certain points on the links in the creature from passing through the floor. Simulation constraints such as these are resolved by SD/FAST as Lagrange multipliers: by calculating and applying the reaction forces necessary to satisfy the constraints. Numerical instability in the simulation is avoided by the use of Baumgarte stabilization [10]. Because the reaction forces will exactly counter any constraint violation, this method of simulating collision results in perfectly inelastic collisions.

Although simple in theory, in practice this technique for simulating collisions did not work well. In addition to the technical challenges of this method (deriving the constraint functions and their derivatives, and tuning the parameters), resolving collisions with constraint forces is not a method well-suited for use within an optimization framework. One reason for its failure is that

as an optimizer changes the optimization variables, it may explore regions where the simulation constraints are violated; for example, the character's feet starting out below the ground. If this violation is large enough, the resulting constraint forces may be enormous and cause instability in the simulation. Even if these violating configurations are detected before the simulation becomes unstable, it is unclear how such information should be communicated to the optimizer so it can avoid such configurations in the future. Merely passing a large (i.e. bad) objective function value to the optimizer is unwise, because the optimizer will use this information to make possibly incorrect assumptions about the cost space near that point.

A second and more serious reason for the failure of this approach to collision handling is that the constraint resolution method used by SD/FAST interferes with the gradient approximation required for sequential quadratic programming (see Section 3.6.1). We believe that this interference is caused by the "stiff", on/off nature of the collision constraints. If small variations in the input cause large changes in behavior in the simulation (the difference between a foot striking the ground or missing it, for example), the optimizer may have a hard time determining the best direction to travel in the parameter space.

The failure to generate good results with a constraint-based collision resolution method led us to try the simpler technique of non-linear springs. Standard springs follow Hooke's law and increase their exerted force linearly as their extension increases:

$$F_{gc} = -k_s(x - \bar{x}) \tag{3.23}$$

where $k_s$ is the stiffness parameter, $x$ is the current length of the spring, and $\bar{x}$ is the rest length. In contrast, the behavior of nonlinear springs is a nonlinear function of the extension. We used the ground contact model derived by Anderson and Pandy[6]:

$$F_{gc} = 0.5336e^{-1150(p_y - y_0)} - 1000\dot{p}_y g(p_y) \tag{3.24}$$

$$g(p_y) = \frac{1}{1 + 10e^{500(p_y - g_0)}} \tag{3.25}$$

where $F_{gc}$ is the ground-contact force exerted by the spring, $p_y$ is the vertical distance between the point in contact and the ground, $\dot{p}_y$ is their relative velocity, and $y_0$ (set by Anderson and Pandy to $0.0065905$) is a parameter that determines when the spring force becomes significant (i.e. more than 0.5 Newtons). All distances are given in meters, velocities in meters per second, and forces in Newtons. The function $g(p_y)$ acts as a damping function, where $g_0$ is a parameter that determines when the damping force is applied. The constants in these equations were tuned by Anderson and Pandy for simulation of human jumping. In our system the characters are significantly less massy and their behavior less energetic and thus we changed the constants to be

$$F_{gc} = 0.5336e^{-700(p_y - y_0)} - 1000\dot{p}_y g(p_y) \tag{3.26}$$

$$g(p_y) = \frac{1}{1 + 10e^{200(p_y - g_0)}} \tag{3.27}$$

where $y_0$ now equals $0.0035423$, and $g_0$ is set to $0.005$.

Examining equations 3.26 and 3.27, we see that our nonlinear springs will exert negligible forces on objects that are more than 5 millimeters away from the ground. These springs can therefore remain attached at all times, further making the simulation of ground contact appear "smooth" to the optimizer.

## 3.6 Gradient Estimation

In the preceding section we have briefly mentioned some of the issues involved with tuning the physical model to work within an optimization framework. Specifically, gradient-based optimization techniques, such as sequential quadratic programming, require that we can calculate the gradients of both the objective function as well as any constraints. However, the reduced gradient formulation makes the evaluation of the gradients of the pose constraints significantly more difficult. Because the values of these constraints now depend on the results of a physical simulation, we are left with the problem: how do we differentiate a physical simulation?

### 3.6.1 Finite Differences

A physical simulation is, in essence, just a function, albeit one without a closed-form expression. The standard method of approximating the derivative of a function is finite differencing. Referring back to the definition of derivative

$$\lim_{h \to 0} f'(x) \approx \frac{f(x+h) - f(x)}{h} \tag{3.28}$$

we see that we can, in theory, calculate the derivative of a function of one variable to any precision with only two function calls. For the case of functions of more than one variable, the partial derivative is similarly defined. For a function of $N$ variables, the partial derivative of $f(\vec{x}_0)$, with respect to the $i^{\text{th}}$ parameter can be estimated by calculating the function at two points: $\vec{x_0} = \{x_{0,0}, x_{0,1}, \ldots, x_{0,N-1}\}$ and $\vec{x_1} = \{x_{0,0}, x_{0,1}, \ldots, x_{0,i} + h, \ldots, x_{0,N-1}\}$:

$$\lim_{h \to 0} \frac{\partial f(\vec{x_0})}{\partial x_{0,i}} \approx \frac{f(\vec{x}_1) - f(\vec{x}_0)}{h} \tag{3.29}$$

For a function of $N$ variables, this *forward difference method* requires $N + 1$ function calls to evaluate all the partials at a given point. Although this method of approximating partial derivatives is efficient, it is also inaccurate. According to *Numerical Recipes in C* [105], forward difference gradient approximation is only "first order accurate"; that is, the accuracy of the gradient estimation

increases linearly as $h$ decreases. Furthermore, limited machine precision will cause equations 3.28 and 3.29 to generate inaccurate results when extremely small values are chosen for $h$.

A more accurate technique for estimating partial derivative is the *central difference method*, which requires two function evaluations for each partial:

$$\frac{\partial f(\vec{x_0})}{\partial x_{0,i}} \approx \frac{f(\vec{x}_2) - f(\vec{x}_1)}{2h} \tag{3.30}$$

where $x_1$ is as above, and $\vec{x_2} = \{x_{0,0}, x_{0,1}, \ldots, x_{0,i} - h, \ldots, x_{0,N-1}\}$. This method, although more expensive than forward differencing, is "second order accurate," meaning the accuracy of the estimate increases as the square of the decrease in $h$.

**Disadvantages of the Finite Difference Approach**

The finite difference approach to approximating function derivatives is common and easy to implement, yet it also has several disadvantages. The primary disadvantage of these methods is the computational cost of evaluating the function many times. A typical spacetime optimization problem might have 70 scalar variables and 100 pose constraints. Using the inexpensive forward difference method of approximating the gradients of the constraints, we would have to evaluate our function (i.e. run a dynamic simulation) 7001 times (one evaluation at $x_0$ plus 70 further evaluations for each constraint).

In practice, the required number of simulations will probably be less than this. Frequently, constraints are imposed at only a few different times — keyframes, in effect. Because pose constraint functions, such as "the position of the head" or "the angle of the knee joint," are not actually separate functions but rather merely read-outs of data generated during simulation, if more than one constraint is imposed at a given time in the animation, we will only need to run the simulation to that time once in order to evaluate them all. A typical gait synthesis problem might have five of these "key times," and thus only require 351 ($5 \times 70 + 1$) function evaluations to evaluate all the constraints. However, even this smaller number of simulations will take a significant amount of computation time to evaluate.

A second disadvantage of finite difference methods is the selection of an appropriate value of the difference interval, $h$. A small $h$ results in roundoff errors, caused by subtracting two almost equal numbers. Similarly, a large value for $h$ will result in truncation errors, as the omitted higher order terms in our integrator's Taylor expansion become significant. According to *Numerical Recipes in C*, the optimal choice of $h$ for approximating gradients with the central difference method is

$$h \approx \sqrt{\epsilon_f}\tilde{f} \tag{3.31}$$

where $\epsilon_f$ is the *fractional accuracy*, and $\tilde{f}$ is the *curvature scale* of the function $f$.

Curvature scale, also called the "characteristic scale" of a function, is an estimate of the scale over which the function changes:

$$\tilde{f} \equiv \sqrt{\frac{\|f\|}{\|f'''\|}} \tag{3.32}$$

where $\|f\|$ is the "average" magnitude of the function and $\|f'''\|$ is the "average" magnitude of the function's third derivative. We can roughly estimate $\tilde{f}$ given some knowledge of the function and the likely range in the values of $\vec{x}$, or we can measure it experimentally.

For spacetime optimization problems, the functions that we are interested in evaluating are the pose constraints, and thus $f$ will generally be the position or orientation of a point on a creature. With our units in meters and radians, typical values of $f$ will have a magnitude of 1. The third derivative (with respect to time) of a pose constraint, $f'''$, is proportional to the first derivative of the actuating forces

$$f''' \approx F' \tag{3.33}$$

which we empirically determined to have a magnitude of 100. Using these estimates, we calculated that $\tilde{f} = \sqrt{0.01} = 0.1$.

The fractional accuracy, $\epsilon_f$, of a function is the accuracy to which it is computed. For "normal" analytic functions, the fractional accuracy of the function is the machine precision, $\epsilon_m$ (about 8 decimal places of accuracy for single-precision floating point numbers, and 16 places for double-precision numbers). However, in the case of reduced gradient spacetime optimization, the functions whose gradients we are approximating are the result of a dynamic simulation, which is a process with limited accuracy. Thus, in order to calculate $\epsilon_f$, and ultimately the optimal choice for $h$, we must determine how accurate the simulation is.

**Simulation Accuracy**

At its core, physical simulation is the process of integrating a set of ordinary differential equations. Inaccuracy in this process can arise from two causes: truncation error (also known as "discretization error") and roundoff error. Roundoff error is caused by the finite precision of numerical representation on computers, and will depend on the number and type of arithmetical operations used in an integration step. The total roundoff error in a simulation increases in proportion to the number of integration steps used, and thus prevents us from using a very small integration step size ($\Delta t$).

Truncation error in numerical integration is caused by truncation of the infinite Taylor series required to form the integration algorithm. This kind of error, which would be present even if we had access to infinite precision arithmetic, depends on the step size used during the integration, the order of the integration method, and the "shape" of the specific problem being solved.

The physical simulation package which we used, SD/FAST, uses a fourth-order Runge-Kutta integrator. Ideally, the fractional error, $\epsilon_f$, associated with integrating an ordinary differential equation over a unit-interval using this method is approximately

$$\epsilon_f \sim \frac{\epsilon_m}{\Delta t} + \Delta t^4 \tag{3.34}$$

where $\epsilon_m$ is the machine precision and $\Delta t$ is the integration step-size. Using double precision math, where $\epsilon_m = 2.22 \times 10^{-16}$, the minimum possible integration error of $3.002 \times 10^{-13}$ is achieved when $\Delta t = \sqrt[5]{\epsilon_m} = 7.400 \times 10^{-4}$.

In practice, however, this error can be much larger. Very stiff systems of differential equations (systems in which small changes in the input can cause large changes in the output) can increase this error or require the use of a smaller, "less optimal" $\Delta t$. Our physical simulation includes a stiff, nonlinear spring model of ground contact (see Section 3.5.2). This stiff contact model necessitates the uses of a smaller time step ($\Delta t = 10^{-5}$) in order to achieve the modest fractional accuracy of $\epsilon_f = 10^{-6}$.

We can use this value of the fractional accuracy, as well as the previously calculated curvature scale, in order to compute our our best choice for $h$. According to equation 3.31, the optimal $h$ is proportional to $\sqrt{\epsilon_f}\tilde{f}$. Using our previously derived values for $\epsilon_f$ and $\tilde{f}$ gives us $h = \sqrt{10^{-6}}\,0.1 = 10^{-4}$. This finite difference interval will give us a forward difference accuracy on the order of $10^{-4}$ and a central difference accuracy on the order of $10^{-6}$.

## 3.7 Reduced Gradient Spacetime Optimization with Finite Differences

Our first attempt at solving our reformulated, reduced gradient spacetime optimization problems used finite difference approximated gradients. The sequential quadratic programming software that we used to solve our spacetime problems is called SNOPT[37]. A feature of this software is that it can efficiently approximate unknown elements in the gradients of its objective or constraint functions. It estimates these gradients using two methods, depending on the progress of the optimization. For the beginning and middle of the optimization problem, when we are trying to rapidly explore the space around the initial point, SNOPT uses the inexpensive forward differencing method. As it approaches an optimal solution, SNOPT switches to the more accurate (but more expensive) central difference method. This hybrid approach to gradient estimation is significantly faster than using central differences alone, and does not affect convergence. We tested our reduced gradient reformulation of spacetime optimization with finite difference approximated gradients on two creatures: a two-dimensional hopping "pogo stick" (Figure 3.6) and a 3D monopod "Luxo" lamp (Figure 3.8).

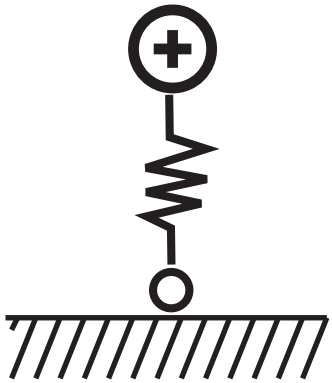

Figure 3.6: A two-dimensional pogo stick

### 3.7.1 Pogo stick results

Our two-dimensional actuated pogo stick has one actuator, a prismatic joint that can exert an arbitrary (although bounded) force on the spring connecting its "head" to its "foot." The optimization variables, $\vec{q}$, for this character are the initial configuration of the pogo stick and the parameters of the representation of the forces exerted by its actuator over the course of the animation. The configuration variables are the position and velocity of the head, and the position and velocity of the foot. The trajectory of the force exerted by the leg actuator was represented with a six knot (i.e. five segment) Hermite spline. We chose this number after experimentation indicated that fewer knots would provide insufficient detail to represent the actuator forces for fast motion, and more knots would merely add extra variables and complexity to the problem with no improvement in the quality of the results. We performed roughly 50 trials with the pogo stick, attempting to generate motion for two different tasks: crouching and jumping vertically.

#### Pogo stick crouch results

The following tables describe eight representative trials of the simpler of these tasks: the "crouch" motion. The pose constraints for this task were for the character to begin in the rest position (shown at the left in Figure 3.7) with zero velocity and end, one second later, crouched lower (shown on the right in Figure 3.7). Table 3.1 provides a summary of the motions used as initial guesses for this task. In this table, each trial is characterized by three numbers: the "cost" of the motion, i.e. the value of the objective function, the total number of equality and inequality constraint violations, and the sum of these violations. All initial motions were either generated by hand, or were slightly randomized variations of other guesses.

Table 3.2 similarly describes the results of these trials. In each case, we indicate how long the

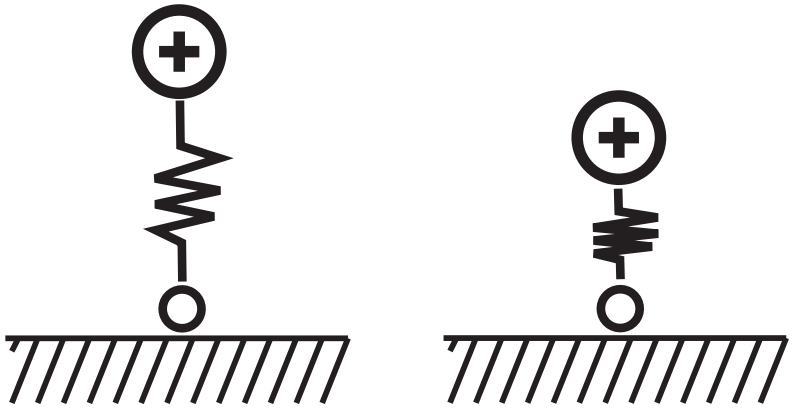

Figure 3.7: The two poses for the pogo stick's crouch task. The desired starting pose is shown on the left, and the desired final pose is shown on the right.

| Trial No. | Cost | Number of constraint violations | Sum of constraint violations |
|---|---|---|---|
| 1 | 0.000 | 4 | 2.535 |
| 2 | 0.110 | 1 | 0.189 |
| 3 | 0.168 | 3 | 2.491 |
| 4 | 0.000 | 1 | 0.440 |
| 5 | 965.029 | 1 | 0.458 |
| 6 | 19559.029 | 2 | 0.108 |
| 7 | 4000.000 | 2 | 0.822 |
| 8 | 6231.821 | 3 | 1.450 |

Table 3.1: Initial guesses at the crouching motion for the pogo stick

| Trial No. | Cost | Number of constraint violations | Sum of constraint violations | Time (seconds) | Status |
|---|---|---|---|---|---|
| 1 | 0.110 | 1 | 0.189 | 862 | Marginal |
| 2 | 0.110 | 1 | 0.189 | 313 | Unimproved |
| 3 | 0.110 | 1 | 0.290 | 196 | Failure |
| 4 | 0.025 | 2 | 0.181 | 454 | Marginal |
| 5 | 182.309 | 3 | 14.581 | 175 | Failure |
| 6 | 10551.778 | 2 | 2.310 | 129 | Failure |
| 7 | 1340.202 | 1 | 0.022 | 411 | Success |
| 8 | 2051.118 | 2 | 0.010 | 312 | Success |

Table 3.2: Final results for the crouching motion for the pogo stick

optimization took (in seconds) and provide measures of the quality of the results. In addition to the quantitative measures of cost and constraint violation, we report a qualitative "status" of the motion. A status of "success" indicates that the task was successfully done, even though there may remain small pose constraint violations. "Marginal" indicates that the motion mostly satisfied our pose constraints, but failed in some important way, such as not crouching deeply enough, or having a larger than acceptable final velocity. "Failure" indicates a complete failure of the motion to satisfy the pose constraints that define the task; i.e. the creature does not crouch at all. "Unimproved" means that the optimizer failed to make any progress in improving the initial motion

**Pogo stick vertical jump results**

Tables 3.3 and 3.4 show the initial status and results from ten representative trials of the vertical jumping task. The pose constraints for a vertical jump are similar to the crouch; begin in the rest position and end, one second later, with both the head and the foot at least 0.5 meters in the air.

These first results from our reduced gradient reformulation were somewhat encouraging. Even given our poor initial guesses at the desired motions, slightly more than half of our trials resulted in acceptable results. We also noticed that initial guesses that do not have the "right kind" of motion, i.e. an initial guess at a jumping motion where the character does not leave the floor, rarely converge to a good answer. Although having only 16 variables and 6 pose constraints, our two-dimensional pogo stick problems captured some of the interesting behavior of more complex three-dimensional character animations. For example, the vertical leap task required the pogo stick to anticipate the motion, storing energy in its spring sufficient to leap into the air, and change the timing of the contact between the foot and the ground appropriately. With its short run times and simple geometry, these problems also allowed us to tune aspects of our system, such as the nonlinear spring parameters,

| Trial No. | Cost | Number of constraint violations | Sum of constraint violations |
|---|---|---|---|
| 1 | 0.000 | 1 | 0.300 |
| 2 | 182.816 | 1 | 0.731 |
| 3 | 153.866 | 2 | 0.695 |
| 4 | 150.423 | 1 | 0.668 |
| 5 | 2140.785 | 1 | 0.300 |
| 6 | 16828.855 | 1 | 0.300 |
| 7 | 3715.553 | 1 | 0.098 |
| 8 | 4963.818 | 3 | 1.174 |
| 9 | 698.310 | 2 | 0.704 |
| 10 | 800.000 | 3 | 0.650 |

Table 3.3: Initial guesses at the jumping motion for the pogo stick

| Trial No. | Cost | Number of constraint violations | Sum of constraint violations | Time (seconds) | Status |
|---|---|---|---|---|---|
| 1 | 0.000 | 1 | 0.330 | 187 | Unimproved |
| 2 | 153.866 | 2 | 0.695 | 263 | Marginal |
| 3 | 150.423 | 1 | 0.668 | 168 | Marginal |
| 4 | 147.113 | 1 | 0.669 | 247 | Unimproved |
| 5 | 311.808 | 1 | 0.300 | 315 | Failure |
| 6 | 16828.855 | 1 | 0.093 | 301 | Marginal |
| 7 | 508.026 | 2 | 0.457 | 261 | Failure |
| 8 | 880.916 | 1 | 0.925 | 577 | Marginal |
| 9 | 441.633 | 1 | 0.025 | 422 | Success |
| 10 | 450.071 | 2 | 0.038 | 394 | Success |

Table 3.4: Final results for the jumping motion for the pogo stick

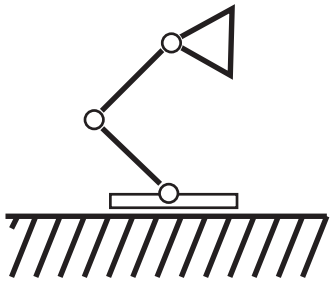

Figure 3.8: The Luxo lamp in its rest pose

simulation step size and optimization parameters.

### 3.7.2 Luxo results

After testing our system with the simple two-dimensional pogo stick, we moved on to a more complex, three-dimensional character. The first 3D character for which we attempted to generate motion was a Luxo lamp (see Figure 3.8). Although modeled and simulated in 3D, this simple, one-footed character is "planar" in that it is only capable of exerting forces in the vertical XY plane. While this limitation does not allow us to generate certain motions (such as jumping in the air and landing facing to the right), the planar Luxo lamp allows us to examine the simulation complexities of full 3D motion.

The Luxo lamp has three rotational actuators: one at the joint between the foot and the lower part of the stand (the "ankle"), one between the lower and upper links in the stand (the "knee") and one between the upper link and the lampshade (the "neck"). Each of these joints has angular limits, and each actuator has bounds on the amount of force it can exert. The lengths and masses of the various elements (the foot, the two links of the stand, and the shade) were based on measurements of a real desk lamp.

The optimization variables for these problems, $\vec{q}$, are composed of the initial configuration of the lamp, $\vec{r}_0$, (the positions and velocities of its component parts) and the Hermite spline parameters of the forces exerted by the actuators over the course of the animation. Initial tests showed that between 6 and 8 knots (describing, respectively 5 to 7 cubic segments) were sufficient to capture the complexity of the actuator forces over time. Fewer knots provided insufficient detail to represent the sometimes quickly changing forces in the Luxo lamp, while more knots added extra variables and complexity to the problem with no improvement in the quality of the results.

Tables 3.5 and 3.6 show the initial status and results of eight representative trials of a simple "jump into the air" task. Specifically, the task was to begin in the rest position (shown in Figure 3.8), be at least 0.3 meters in the air after 0.3 seconds, and to be on the ground in the rest position

| Trial No. | Cost | Number of constraint violations | Sum of constraint violations |
|---|---|---|---|
| 1 | 11.642 | 15 | 11.138 |
| 2 | 11.642 | 12 | 9.618 |
| 3 | 6.061 | 20 | 15.661 |
| 4 | 3.688 | 19 | 6.222 |
| 5 | 20.068 | 26 | 3.084 |
| 6 | 6.751 | 17 | 1.119 |
| 7 | 6.751 | 11 | 0.510 |
| 8 | 1.403 | 7 | 0.564 |

Table 3.5: Initial guesses at the leaping motion for the Luxo lamp

again at $t = 1.0$ seconds. In these tables, the values of the objective function have been scaled by a factor of $10^{-4}$. Filmstrips of these eight initial motions are shown in Figures 3.9 and 3.10.

The results of our Luxo lamp problems were less encouraging than our first tests with the pogo stick. The time the optimizations took was substantially longer, running between 10 and 20 hours per trial, and the answers were of poorer quality. Roughly one half of our experiments generated marginally acceptable results, such as trials 3, 5, and 7 (shown in Figures 3.11 and 3.12) Although these trials appear to have poor initial guesses, the initial motions were in fact qualitatively quite similar to the desired motion. As with the pogo stick tests, initial motions that contained leaps of any height were more easily optimized to produce desired leaping task than initial motions which never left the ground.

### 3.7.3 Discussion of Finite Difference Results

The reduced gradient formulation of spacetime optimization problems gives us a simplicity of form and a generality that traditional spacetime optimization lacks. However, we can see from these simple tests that our reformulation does not solve the problems that plague spacetime optimization. Specifically, the quality of our solutions is still dependent on the quality of the initial guesses; more so in the case of the Luxo lamp than in for the pogo stick. Also, solution times are still long, although our method of approximating gradients are partly to blame for this problem. Finite difference approximations require multiple evaluations of our simulation, which is expensive and has limited accuracy. In order to address this problem, as well as to perhaps increase the effectiveness of our optimization algorithm, we investigated automatic differentiation, a more sophisticated method of generating gradients of complex functions.

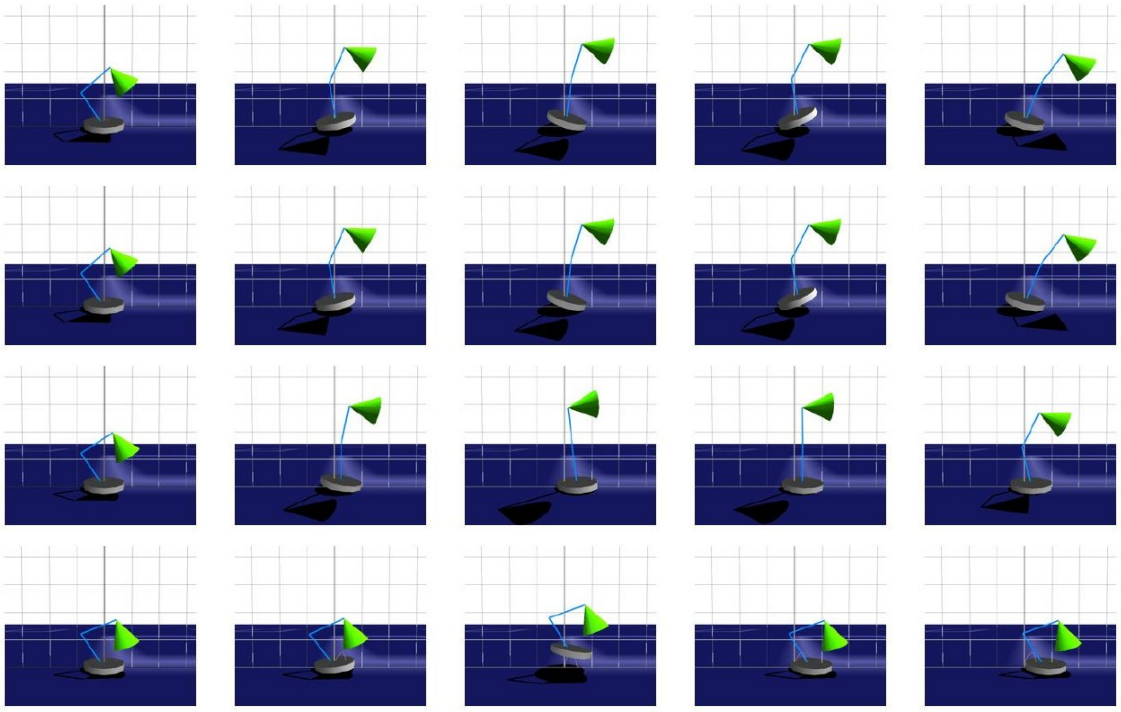

Figure 3.9: Initial motions for trials #1 through #4 (from top to bottom) of the Luxo Lamp forward leap task. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally.

| Trial No. | Cost | Number of constraint violations | Sum of constraint violations | Time (seconds) | Status |
|---|---|---|---|---|---|
| 1 | 1062.130 | 11 | 1.985 | 34933 | Failure |
| 2 | 885.119 | 8 | 1.294 | 45546 | Failure |
| 3 | 102.773 | 10 | 0.587 | 53881 | Marginal |
| 4 | 152.908 | 16 | 2.878 | 78167 | Failure |
| 5 | 54.568 | 12 | 1.174 | 18588 | Marginal |
| 6 | 9.828 | 7 | 0.293 | 29308 | Marginal |
| 7 | 27.258 | 11 | 1.239 | 42350 | Marginal |
| 8 | 1041.615 | 5 | 1.666 | 10485 | Failure |

Table 3.6: Final results for the leaping motion for the Luxo lamp, computed using finite-difference approximated gradients

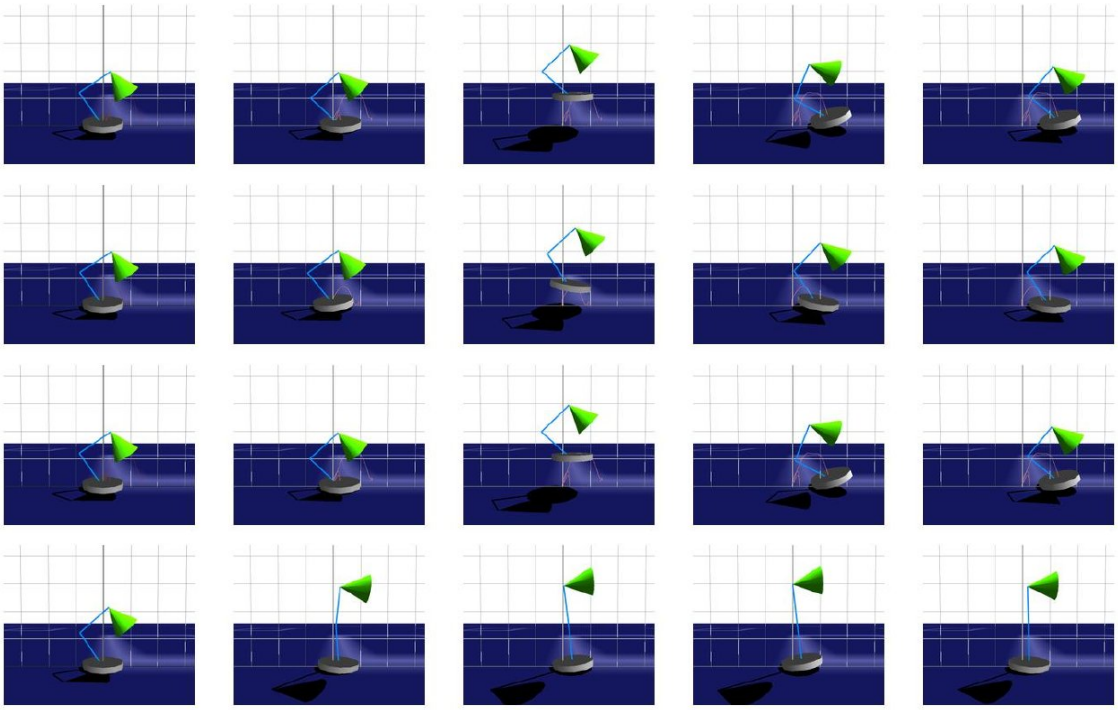

Figure 3.10: Initial motions for trials #5 through #8 (from top to bottom) of the Luxo Lamp forward leap task. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally.

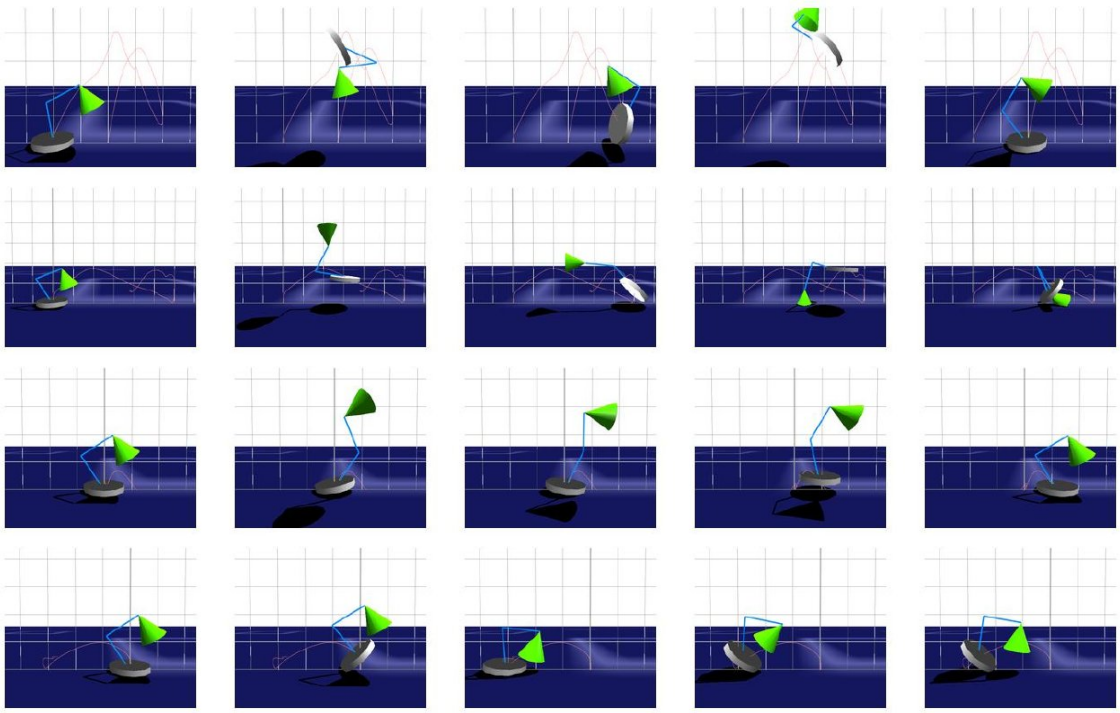

Figure 3.11: Final motions for trials #1 through #4 (from top to bottom) of the Luxo Lamp forward leap task, computed using finite-difference approximated gradients. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally.

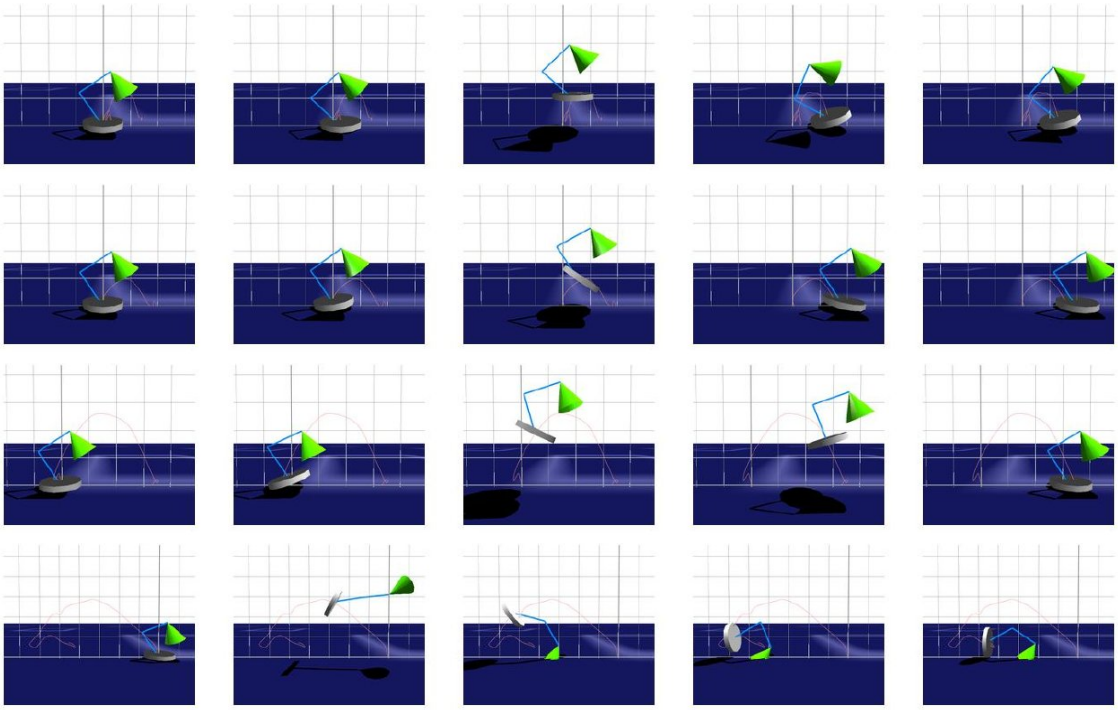

Figure 3.12: Final motions for trials #5 through #8 (from top to bottom) of the Luxo Lamp forward leap task, computed using finite-difference approximated gradients. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally.

## 3.8 Reduced Gradient Spacetime Optimization with Automatic Differentiation

Unlike the finite difference method, automatic differentiation (AD) does not compute approximations of the derivatives, but rather generates an analytical *differentiated function* that exactly computes the gradients. AD techniques rely on the fact that functions are executed on a computer as a sequence of elementary operations such as additions and multiplications, and elementary functions such as $\sin$ and $\log$. By applying the chain rule

$$(f(g(x)))' = f'(g(x))g'(x) \tag{3.35}$$

repeatedly to these operations and functions, derivatives of arbitrary order can be computed automatically, and will be accurate to the working precision. For a more detailed description of automatic differentiation, see Griewank's article on this topic [45].

Automatic differentiation is a very powerful technique for computing derivatives of complex functions. Its accuracy and run-time efficiency is superior to finite difference approximations, and AD's flexibility, reliability and development time efficiency surpass those of hand-coding approaches. Furthermore, AD allows the exact computation of derivatives in cases, such as ours, where no hand-coded differentiation is possible. The automatic differentiation tool ADIFOR (Automatic Differentiation of Fortran) [13, 14, 18] has been used to generate derivative code for applications in areas such as large-scale optimization[7, 17], computational fluid dynamics[19, 24, 58], weather modeling[16], and aerodynamic analysis and modeling[15]. Because ADIFOR is widely used and well-tested, we used this package for our work, although in its C form (ADIC) instead of the FORTRAN version.

The automatic differentiation literature shows us that, as with the finite difference method, it is important that the function to be differentiated is smooth. Although automatic differentiation techniques can easily handle non-smooth functions such as "if" statements, these discontinuities can cause difficulties for the optimizer and should be avoided. The alterations we made to ground contact, described in Section 3.5.2, are sufficient to guarantee smoothness for the purposes of automatic differentiation.

The computational cost of evaluating a differentiated function is proportional to the product of the number of variables and the cost of evaluating the original function[46, 47]. In order to determine the constant of proportionality, we ran some simple experiments with the Luxo lamp. Using a 275MHz R10000 SGI Octane, we timed how long it took to compute a one second animation of the Luxo lamp leaping into the air and landing on the ground. In order to avoid the possibility of skewed results from a single data point, we used five different motions for the same task, and averaged the run times. We found that the original (undifferentiated) SD/FAST simulation took an average of $0.70$ seconds to generate this one second animation. The differentiated function took $73$ seconds,

| Trial No. | Cost | Number of constraint violations | Sum of constraint violations | Time (seconds) | Status |
|---|---|---|---|---|---|
| 1 | 78.147 | 6 | 0.337 | 10196 | Success |
| 2 | 33.039 | 7 | 1.539 | 2239 | Marginal |
| 3 | 603.681 | 19 | 1.588 | 7798 | Marginal |
| 4 | 7.196 | 6 | 0.372 | 8529 | Success |
| 5 | 680.707 | 22 | 2.305 | 13196 | Failure |
| 6 | 4.625 | 15 | 0.979 | 722 | Marginal |
| 7 | 91.316 | 17 | 3.157 | 4173 | Marginal |
| 8 | 50.763 | 7 | 0.776 | 647 | Marginal |

Table 3.7: Final results for the leaping motion for the Luxo lamp, using automatic differentiation

giving us a time constant of 100.

Note that unlike finite difference approximations, the cost of evaluating gradients with automatic differentiation does not depend on how many constraint gradients are evaluated. Rather, the cost depends on the latest time at which one of the constraints is imposed. In order to evaluate the gradients at time $t = t_1$, we must run the differentiated simulation from $t = 0.0$ to $t = t_1$, which will incidentally compute any gradients we need for all the times in between. For example, the cost to compute the constraint gradients for a problem with a pose constraint at $t = 1.0$ will be unchanged if more pose constraints are added that have evaluation times prior to $t = 1.0$. This unusual cost dependency is the result of the way in which the constraint gradients are evaluated: as the output of the differentiated simulation code.

### 3.8.1 Results with Automatic Differentiation

We tested our reduced gradient reformulation of spacetime optimization with automatically differentiated gradients on two creatures: the Luxo lamp and a three-dimensional planar biped. Our first goal was to investigate how using automatic differentiation instead of finite difference approximated gradients changed the convergence behavior of the Luxo lamp optimization problems outlined in Section 3.7.2. To do this testing, we used the same eight initial motions described in Table 3.5. The topology of the Luxo lamp and the tuning parameters of the optimizer were identical; the only difference was in how the gradients were computed and the step-size used in the simulation.

Table 3.7 shows the results from the use of automatic differentiation are significantly better than those we obtain with a finite-difference approach. The times required to solve these eight test problems were generally an order of magnitude less than that required by the finite difference

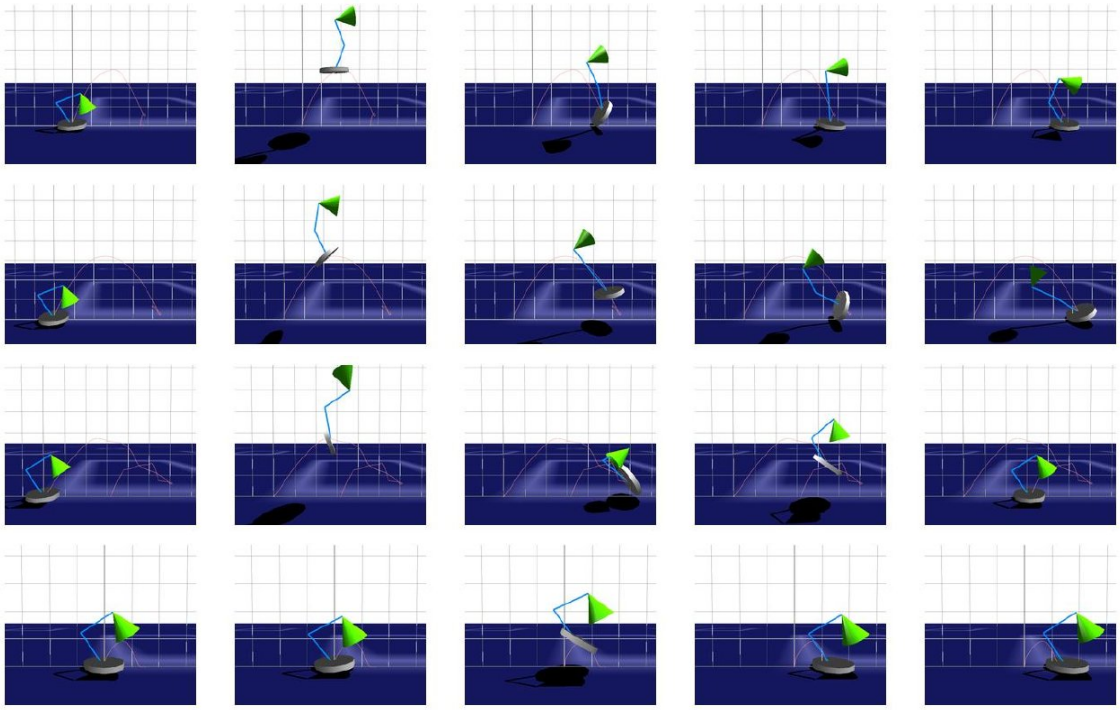

Figure 3.13: Final motions for trials #1 through #4 (from top to bottom) of the Luxo Lamp forward leap task, computed using automatic differentiation. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally.

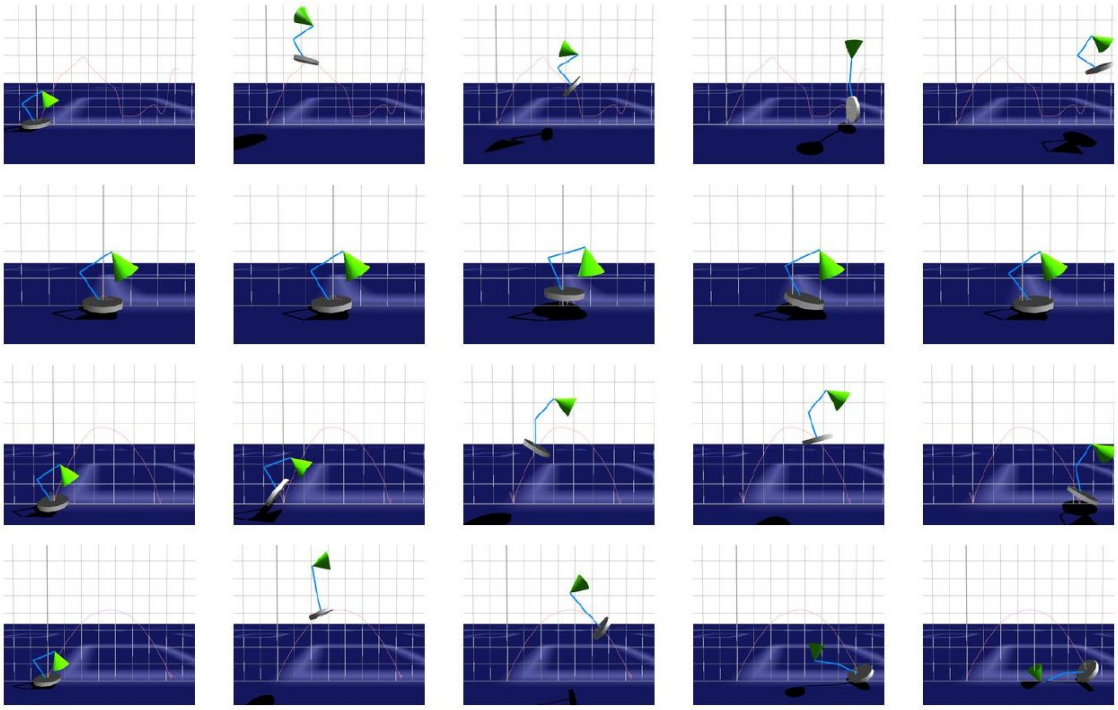

Figure 3.14: Final motions for trials #5 through #8 (from top to bottom) of the Luxo Lamp forward leap task, computed using automatic differentiation. The shown frames occur every 1/4th of a second. Grid lines occur every 10 cm, both vertically and horizontally.

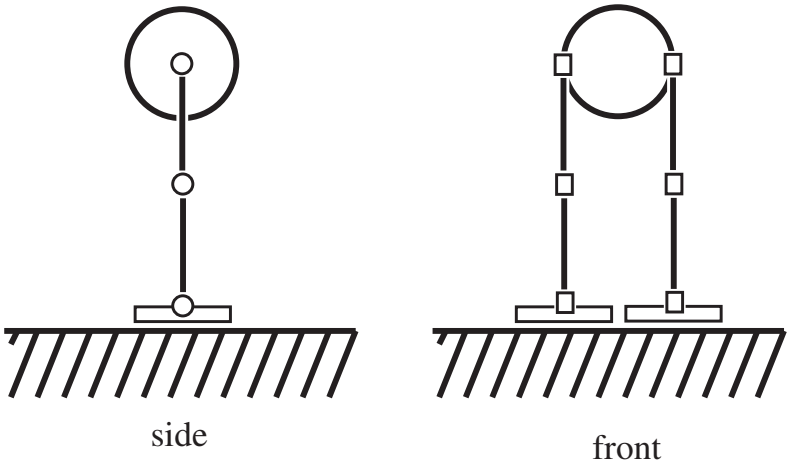


Figure 3.15: The biped in its rest pose

method. Part of this speed-up is the result of the larger $\Delta t$ that automatic differentiation allows us to use. If we run trial #6 with the smaller $\Delta t$ used for finite differences ($\Delta t = 10^{-5}$), we obtain essentially the same result as trial #6 in Table 3.7, but this optimization now takes $6937$ seconds. Our automatic differentiation based method yields an additional speed-up because the cost of constraint gradient evaluations are dependent on the time of the last constraint, not on the number of the constraints. Thus, fewer calls to the expensive gradient computation functions are required ($2864$ using finite differences versus $481$ using automatic differentiation in trial # 6). Further, the higher accuracy of the gradients allow the optimizer to operate more efficiently.

**Biped**

We also tested our technique on a more complex character: a simple biped (see Figure 3.15). As with the Luxo lamp, our biped is fully three-dimensional and operates in a 3D world. Our biped has six actuators: one at the ankle, knee and hip of each leg. The optimization variables, $\vec{q}$, for this character are the initial configuration of the body (its initial Cartesian and angular positions and velocities) and the parameters of the forces exerted by its actuators over the course of the animation. As with the Luxo lamp, the trajectories of the forces exerted by the actuators were represented with six knot (i.e. five segment) Hermite splines.

We performed roughly 50 trials with the biped, attempting to generate motion for a forward hop (a broad jump). Optimization and simulation parameters remained the same, with the exception of the $\Delta t$, which had to be reduced to $10^{-5}$ in order to maintain simulation and gradient accuracy. Tables 3.8 and 3.9 show the initial and final status, respectively, of eight attempts at generating a small vertical hop. In these tables, the the objective function (the "cost") has been scaled by a factor

| Trial No. | Cost | Number of constraint violations | Sum of constraint violations |
|---|---|---|---|
| 1 | 0.799 | 15 | 1.405 |
| 2 | 0.804 | 13 | 1.168 |
| 3 | 0.573 | 28 | 2.968 |
| 4 | 1.198 | 30 | 3.416 |
| 5 | 0.981 | 25 | 1.710 |
| 6 | 0.950 | 27 | 1.582 |
| 7 | 0.963 | 23 | 2.633 |
| 8 | 0.798 | 19 | 1.486 |

Table 3.8: Initial guesses at the hopping motion for the biped

of $10^{-4}$.

Despite the positive results from the Luxo lamp experiments, our method failed to generate similarly acceptable motion for larger biped problems. We attribute this failure to two factors. First is the fact that the biped is much less physically stable than the Luxo lamp. Specifically, the biped is only stable when both its legs are touching the ground and the feet are oriented appropriately. Although maintaining balance is a problem that all motion synthesis methods must deal with, the added difficulty of this task undoubtedly contributed to the failure of our spacetime optimization technique. A more significant factor, however, is that the number of variables in the biped is twice that of the Luxo lamp problem. A larger number of variables makes the optimization space exponentially larger, and therefore much harder to explore. We believe that this so-called "curse of dimensionality" [12] is largely responsible not only for the drastically increased run-times of our algorithm, but also for its failure to reliably converge.

## 3.9 Simulated Annealing

Although a gradient-based method such as sequential quadratic programming is the most natural choice for solving spacetime optimization problems, it is also clear that these techniques have certain drawbacks. Specifically, the gradients required by SQP and related techniques are expensive to compute. With our reduced gradient approach, function evaluations (i.e. running a physical simulation) can be quite expensive and gradient evaluations can cost hundreds or thousands as much as a single function evaluation, depending on their method of computation.

Gradient-based approaches have a further disadvantage in that they are all local methods. That is, these optimization methods are only able to search "near" their starting point for the best answer,

| Trial No. | Cost | Number of constraint violations | Sum of constraint violations | Time (seconds) | Status |
|---|---|---|---|---|---|
| 1 | 0.799 | 18 | 1.464 | 61123 | Failure |
| 2 | 0.796 | 28 | 2.806 | 58392 | Failure |
| 3 | 0.026 | 5 | 0.210 | 64919 | Marginal |
| 4 | 0.028 | 26 | 3.365 | 37120 | Failure |
| 5 | 0.903 | 26 | 2.260 | 45671 | Marginal |
| 6 | 0.796 | 24 | 1.581 | 188928 | Failure |
| 7 | 1.361 | 34 | 5.445 | 378715 | Failure |
| 8 | 0.205 | 24 | 3.329 | 371207 | Failure |

Table 3.9: Final results for the hopping motion for the biped

and consequently they can be quite sensitive to the quality of the initial guess. In contrast, global optimization techniques seek to find the absolute best possible set of variables that satisfy the constraints. Although heavily researched (see Horst and Pardalos[57] for a fairly up-to-date reference to the field), global minimization is still considered one of the most difficult types of optimization problem to solve. Certain techniques, however, are well-developed and have been used successfully by researchers to solve other complex problems.

As mentioned in Section 2.2.1, simulated annealing is a popular technique for the global optimization of problems with discrete and, more recently, continuous variables. The standard implementations of simulated annealing do not use gradients or gradient information to explore the optimization space. For problems where gradients do not exist, this property is an advantage but for smooth problems such as our own, it can be a significant drawback. The gradients and Hessians of our objective and constraint functions contain extremely useful information about the optimization space. These functions tell the optimizer which directions it should explore, or which directions to ignore, in order to find better objective function values or to reduce constraint violations. Ignoring this information, if it is available, means abandoning one of our most important sources of knowledge about the problem and it is for this reason that simulated annealing is rarely used to solve continuous optimization problems. However, in cases where evaluating the gradient or the Hessian is considerably more expensive than evaluating the value of the objective function, this "disadvantage" of simulated annealing could be a benefit.

### 3.9.1 Our Simulated Annealing Experiments

We ran a short series of experiments in order to investigate the application of simulated annealing to the solution of reduced-gradient spacetime optimization problems. We chose to generate motions

| Motion | Cost | Number of constraint violations | Sum of constraint violations |
|---|---|---|---|
| A | 51.347 | 6 | 1.397 |
| B | 5.536 | 10 | 1.889 |
| C | 2.598 | 6 | 0.679 |

Table 3.10: Initial guesses at the leaping motion for the Luxo lamp

for the Luxo lamp making a forward leap. This task was chosen because we had some success with sequential quadratic programming, so we knew that certain problems could be solved, but we also found that not all initial guesses succeeded and thus there is room for possible improvement.

For our tests, we used the simulation parameters described in Sections 3.5.2 and 3.8. Rather than implementing our own simulated annealing algorithm, we used Ingber's ASA software [60, 61], choosing this package both for its power and for the available literature on its use and limitations. Although there has been recent work on constrained simulated annealing [129, 128], it remains most widely used and understood as a unconstrained technique. Thus, we used a penalty method in order to convert our constrained optimization problem into an unconstrained one.

We ran a series of 25 experiments in order to explore the behavior of simulated annealing on spacetime problems. In Tables 3.10, 3.11 and 3.12 we show the status of nine representative experiments. Table 3.10 shows the status of the initial guesses at the desired motions: their cost (the value of the objective function, scaled by $10^{-4}$), the total number of constraint violations (equality and inequality), and the sum of these violations.

Unlike in our earlier experiments, we used only three different initial motions to generate the nine results reported here. The choice to use a smaller number of initial motions was made because simulated annealing, being a global method, is less sensitive to initial conditions than sequential quadratic programming [61]. Instead, variation in the optimization problems is achieved through a combination of changing the random number seeds used by the algorithm and varying the annealing schedule parameters. Table 3.11 shows the values of these parameters used for each trial. The second column indicates which of the three motions ("A", "B" or "C"), described in Table 3.10, were used in each trial. The third and forth columns report the values of the *initial annealing temperature* and the *temperature ratio scale* respectively. The initial annealing temperature is a measure of the "randomness" of the search at the beginning of the run. The temperature ratio scale sets the rate of annealing. Larger values of this parameter result in slower cooling and thus longer run times as well as (perhaps) a better chance at converging to a global minimum. Further information about these parameters can be found in the ASA manual, which is distributed with the software.

The results of nine representative trials are summarized in Table 3.12. We can see that, despite

| Trial No. | Motion | Initial Temperature | Ratio Scale |
|---|---|---|---|
| 1 | A | 1.0 | 0.00001 |
| 2 | A | 10.0 | 0.00001 |
| 3 | B | 1.0 | 0.00001 |
| 4 | B | 10.0 | 0.00001 |
| 5 | B | 100.0 | 0.00001 |
| 6 | C | 1000.0 | 0.00001 |
| 7 | B | 1.0 | 0.0001 |
| 8 | B | 1.0 | 0.001 |
| 9 | B | 1.0 | 0.000001 |

Table 3.11: Parameter values for initial guesses at leaping motion for the Luxo lamp

| Trial No. | Cost | Number of constraint violations | Sum of constraint violations | Time (seconds) | Status |
|---|---|---|---|---|---|
| 1 | 591.130 | 15 | 3.967 | 51123 | Failure |
| 2 | 619.920 | 16 | 5.649 | 54238 | Failure |
| 3 | 542.765 | 28 | 19.238 | 65061 | Failure |
| 4 | 831.254 | 30 | 25.955 | 66414 | Failure |
| 5 | 1489.686 | 32 | 46.290 | 43617 | Failure |
| 6 | 6.223 | 2 | 0.210 | 38402 | Marginal |
| 7 | 648.927 | 32 | 32.195 | 51498 | Marginal |
| 8 | 1229.692 | 36 | 51.129 | 49100 | Failure |
| 9 | 633.639 | 31 | 17.728 | 62729 | Failure |

Table 3.12: Final results for the leaping motion of the Luxo lamp

rather long run times, none of our trials converged to anything like an acceptable answer. The best final motions, trials 6 and 7, partly accomplished the major goal (i.e. they jumped into the air) but both of these motions failed to land properly, and had very large total constraint violations.

Although by no means exhaustive, our experiments with simulated annealing appear to confirm our initial suspicion that this technique is unsuitable for solving reduced gradient spacetime optimization problems. We believe that the primary reason for this failure is the large number of variables in the problem. Simulated annealing is known to perform poorly on problems with many variables [61]. Unlike gradient-based methods where it is relatively easy (although not necessarily cheap) for the optimizer to find the direction in which it should travel, simulated annealing, like any stochastic method, is forced to sample the state space. Because sampling high-dimensional spaces takes many samples this "curse of dimensionality" appears to affect simulated annealing on problems with fewer variables than for sequential quadratic programming. The large cost of sampling a high-dimensional space more than offset the reduced cost of not needing to compute expensive gradients, resulting in long run times and poor convergence properties

## 3.10 Summary and Discussion

By reformulating spacetime optimization problems, we sought to make spacetime optimization more flexible and more powerful. These advantages, we believed, would arise from allowing the algorithm to search for the optimal times for footfall and other contact events. We believed that by relieving the optimizer from its unnatural role as an ODE solver, we could furthermore reduce the number of iterations the optimizer required, and thus increase performance. An evaluation of our work indicates that we only partially succeeded at these goals.

A reduced gradient approach to spacetime optimization is certainly more flexible than the standard technique. By using forward simulation instead of Newtonian constraints, the optimizer can control footfall and other contact events in order to find the best possible motion that fits the pose constraints. This improvement makes spacetime optimization more automatic, and could conceivably allow a user to generate previously unseen gaits for real creatures or realistic gaits for imaginary animals.

However, this flexibility came at more of a price than we had anticipated. The run times of problems solved using our reduced gradient method were generally longer than those of comparable problems solved the traditional way. The few tests we did with the traditional formulation indicates that the optimizer required fewer iterations to converge using our technique, but individually these iterations took much more time.

### 3.10.1 Speeding up the Simulation

Why are iterations of our reduced gradient sequential quadratic programming algorithm so costly? The obvious answer is because the constraint functions and their gradients must be evaluated by running a dynamic simulation, which is computationally expensive. Profiling our code revealed that nearly 90% of the computation time during an optimization run was spent running the differentiated simulation code.

Our dynamics simulation software, SD/FAST, uses fairly advanced $O(n)$ dynamics algorithms (Kane's method for computing dynamics [66], coupled with a proprietary $O(n)$ formulation of the equations of motion) and it is unlikely that we can gain a significant speed-up with other techniques. Larger simulation step sizes ($\Delta t$) could be used, this would require us to use "softer" differential equations in order to maintain numerical stability. By far the stiffest part of our system is ground contact and thus in order to have a larger integration step size, we would have to make contact "softer" and thus less physically realistic. Because we wish to generate physically realistic motion, this sacrifice would not be acceptable.

### 3.10.2 Different Physics Formulations

Rather than attempting to speed up our dynamic simulation, we could perhaps use a different technique to solve for character motion. Liu and Popović [74] presented work that, rather than solving for muscle forces and torques, used simple, high-level rules about the physics of motion in order to make keyframed input motion look more realistic. Their software altered input motion so that angular momentum is conserved during flight and, using empirically-derived heuristics, momentum is transferred realistically between the ground and the character during contact. Unlike true physical simulation, however, Liu's technique is only applicable to highly dynamic motion, such as jumping and running. Further, the input motion must have certain keyframes, the "transition poses," fully specified before optimization can begin. Both of these drawbacks makes their work, as it stands, unsuitable for the task of automatic gait generation but does indicate an interesting area for future research.

Another alternative to a standard dynamic simulation is proposed in recent work by Fang and Pollard [31]. In this paper, they present a formulation of the equations of motion that allows computation of first derivatives with a cost that scales linearly with the number of degrees of freedom of the character. This technique is a significant improvement over finite difference and automatic differentiation techniques, in which the computation cost scales as the square of the number of degrees of freedom.

Fang and Pollard's formulation of the equations of motion allows them to achieve very fast run times, generating realistic human motion in only a few minutes. However, their technique does not

explicitly compute joint torques and thus enforcing torque limits would multiply the computation cost. Also, this technique necessitates a different objective function: one which minimizes the integral of sum-squared joint accelerations, rather than the integral of sum-squared joint torques. This altered objective function occasionally results in unnatural motion, for example the knees of a human figure exerting unrealistically large torques. Fang and Pollard's work uses the traditional spacetime technique of Newtonian constraints and thus requires the animators to specify the location and timing of footfall and other contact events. As with Liu and Popović's work, it is unclear how we could adapt Fang and Pollard's fast formulation of the equations of motion to a system as flexible as our reduced gradient method.

### 3.10.3 Initial Guess Quality

One advantage that traditional, Newtonian-constraint-based formulations of spacetime optimization problems have over our reduced gradient technique is the ease of crafting initial guesses at the optimal motion. Our approach to spacetime optimization requires the user to craft initial guesses by specifying the torque trajectories of the character's actuators over time. In contrast, techniques with Newtonian constraints allow users to craft initial motions by manipulating the spatial trajectories of the character instead, a far more intuitive approach.

Other than usability issues, this difference is important because the quality of the initial guesses has a large effect on the quality of the output. As mentioned earlier, sequential quadratic programming is a local optimization techniques. That is, it is only capable of finding the best solution that lies in some neighborhood near our starting point. Thus, if our initial guess at a character's motion is "distant," i.e. very different, from the optimal motion, we may be unable to find it. We believe that one of the major reasons for the failure of our spacetime algorithm on larger problems, such as the biped, is that our initial guesses were poor. Poor initial guesses, combined with our very difficult optimization landscape, prevented our local optimization methods from finding reasonable motion. The problems of local minima, and possible ways around them, are discussed in the conclusion of this thesis (Chapter 6).

### 3.10.4 High Dimensionality

Although the quality of our initial guesses certainly had an effect on the quality of the final motions generated by our software, it was not the only factor that caused difficulties. Note, for example, that the initial guesses used to generate the Luxo lamp motions were as poor as those used in the biped examples. However, the Luxo lamp problems almost all gave acceptable or good results, but the biped largely failed.

The most likely reason for this difference in behavior, as mentioned in Section 3.8.1, is that

optimization problems posed with the biped contain twice as many variables as the problems posed with the Luxo lamp. In general the more variables that an optimization problem has, the more difficult it will be to reliably solve. This difficulty is dramatically increased when, as in our problem, these variables are nonlinear.

One approach to solving this problem was presented by Popović in his thesis [103]. Here, he presented a motion-warping technique that transformed complex characters with many degrees of freedom into smaller, lower-dimensional mechanisms that spacetime methods could more easily solve. After warping the input motion using the simpler character model, his software then remapped the final motion on to the full input model. Popović's character transformation technique used knowledge about human motion in general as well as specific information about the motion being warped. For example, a broad jumping human does not use all of the many dozens of degrees of freedom a full human model contains. Because this motion is planar, the motion may be calculated in two dimensions. Further, because the forces exerted by the left and right sides of the character's body are symmetric, half of of the variables may be removed as redundant. Finally, the degrees of freedom for much of the upper body may be abstracted into a single 3 DOF point mass. In this way, Popović's work could reduce the number of degrees of freedom of a broad jumping human figure from 59 to 10.

Unfortunately, this approach to reducing the complexity of character models has significant drawbacks. Specifically, Popović's technique works best for high-energy, dynamic motions and does not perform well with low-energy motion such as walking. Also, simplifying a character is not an automatic process, but requires the user to analyze and guide the simplification process (although this only needs to be done once for each character/motion pair). Finally, because spacetime optimization is performed on simplified models, the results are not guaranteed to be physically realistic when reapplied to the full-dimensional character model. For example, if, as in the broad-jumping example described above, the upper body is reduced to a point mass, angular momentum will not be properly conserved. However, despite these disadvantages, character simplification holds a great deal of potential for reducing complexity in difficult spacetime optimization problems, and thereby allowing faster and better solutions.

# Chapter 4

# Truss Structures

## 4.1 Introduction

A recurring challenge in the field of computer graphics is the creation of realistic models of complex man-made structures. The standard solution to this problem is to build these models by hand, but this approach is time consuming and, where reference images are not available, can be difficult to reconcile with a demand for visual realism. This chapter presents a method, based on practices in the field of structural engineering, to quickly create novel and physically realistic truss structures such as bridges and towers, using simple optimization techniques and a minimum of user effort.

"Truss structures" is a broad category of man-made structures, including bridges (Figure 4.1), water towers, cranes, roof support trusses (Figure 4.10), building exoskeletons (Figure 4.2), and temporary construction frameworks. Trusses derive their utility and distinctive look from their simple construction: rod elements (beams) which exert only axial forces, connected concentrically with welded or bolted joints.

These utilitarian structures are ubiquitous in the industrialized world and can be extremely complex and thus difficult to model. For example, the Eiffel Tower, perhaps the most famous truss structure in the world, contains over 15,000 girders connected at over 30,000 points [50] and even simpler structures, such as railroad bridges, routinely contain hundreds of members of varying lengths. Consequently, modeling of these structures by hand can be difficult and tedious, and an automated method of generating them is desirable.

### 4.1.1 Background

Very little has been published in the graphics literature on the problem of the automatic generation of man-made structures. While significant and successful work has been done in recreating natu-

ral structures such as plants, the trunks and roots of trees, and corals and sponges (summarized in the review paper by Prusinkiewicz [106]), these studies emphasize visual plausibility and morphogenetic realism over structural optimality. Parish and Müller recently described a system to generate cityscapes using L-systems [95], but this research did not address the issue of generating individual buildings for particular purposes or optimality conditons. Computer-aided analysis of simple truss structures, coupled with graphic displays of deflection or changing stresses, has been used for educational purposes within the structural engineering [81] and architecture [98] communities, but these systems are not intended for the design of optimal structures.

In the field of structural engineering, the use of numerical optimization techniques to aid design dates back to at least 1956 when linear programming was used to optimize frame structures based on plastic design theory [54]. Since then, extensive research has been done in the field of "structural synthesis," as it is sometimes called, although its penetration into industry has been limited [124, 49]. Techniques in the structural engineering literature generally fall into three broad categories: geometry optimization, topology optimization, and cross-sectional optimization (also known as "size optimization") [70].

Cross-sectional optimization, the most heavily researched of these three techniques, assumes a fixed topology and geometry (the number of beams and joints, their connectivity, and locations) and finds the shape of the beams that will best, either in terms of mass or stiffness, support a given set of loads. The parameters of the structure that are changed during optimization, called the design variables, are properties that affect the cross-sectional area of a beam such as, for the common case of tubular elements, the radius and thickness of each tube. An example of this technique in practice is the design of the beams that are used to build utility transmission towers [127], where savings of only a few hundred dollars in material costs, when multiplied by the thousands of towers needed for a new transmission route, can be a substantial gain.

Topology optimization addresses the issues that size optimization ignores; it is concerned with the number and connectivity of the beams and joints, rather than their individual shape. Because structure topology is most easily represented by discrete variables, numerical techniques used for topology optimization are quite different from those used for continuous size and geometry optimization problems. Prior approaches to this problem have included genetic programming [25], simulated annealing [110], and "ground structure methods" wherein a highly connected grid of pin-joints is optimized by removing members based on stress limits [53, 96]. A review of these discrete parameter optimization problems in structural engineering can be found in Kirsch [70].

The third category of structural optimization, geometry optimization, lies between the extremes of size and topology optimization. The goal of geometry optimization is to refine the position, strength and, to some extent, the topology of a truss structure. Because these problems are highly non-linear, geometry optimization does not have as lengthy a history as size and topology optimiza-

tion [124]. A common approach, called "multi-level design," frames the problem as an iterative process wherein the continuous design variables are optimized in one pass, and then the topology is changed on a second pass [119]. It is from this literature that we draw the inspiration for our work.

For civil and mechanical engineers, the ultimate goal of structural optimization is a highly accurate modeling of reality. Thus, common to all these techniques, no matter how different their implementations, is a desire for strict physical accuracy. In the field of computer graphics, however, we are often just as concerned with the speed of a solution and its visual impact as with its accuracy. For example, although important to structural engineers, optimizing the cross-sections of the members used to build a bridge would be considered wasted effort in the typical computer graphics application, as the subtle differences between different shapes of beams are hardly noticeable from cinematic distances. Thus, rather than being concerned only with our model's approximation to reality, we are interested in optimizing the geometry and topology of truss structures with the goals of speed, user control and physical realism.

## 4.2 Representing Truss Structures

Truss structures consist of rigid beams, pin-connected at joints, exerting axial forces only. This simple form allows us to represent trusses as a connected set of three-dimensional particles where every beam has exactly two end-points, and joints can accommodate any number of beams. In our model, the pin-joints are classified into three types: free joints, loads, and anchors. *Anchors* are points where beams are joined to the earth, and thus are always in force balance. *Loads* are points at which external loads are being applied, e.g. the weight of vehicles on a bridge. Lastly, *free joints* are pin joints where beams connect but which are not in contact with the earth and have no external loads.

### 4.2.1 Constructing the Model

Before solving for an optimal truss structure, we must have a clear idea of what purpose we want the structure to serve. For example, a bridge must support some minimum weight along its span, the Eiffel Tower must support observation decks, and roof trusses need to support the roofing material. We model these support requirements as loads, which are placed by the user. Although most structural loads are continuous (e.g. a planar roadbed), appoximating loading as a set of discrete load-points is standard practice within the civil and structural engineering disciplines [55].

In addition to having external loads, every truss structure must also be supported at one or more points by the ground. For real structures, the location of these anchors is influenced by topography, geology, and the economics of a particular site, but for our modeling purposes their positions are

specified by the user.

After placement of the anchors and loads, a rich set of free joints is automatically added and highly connected to all three sets of joints (see Figure 4.3). Specifically, our software generates free joints on a regular three-dimensional grid defined by the locations and spacings of the load and anchor points. Currently, our user interface asks the user to provide the number of vertical "layers" of free joints and whether these joints are initially placed above or below the loads (for example, they are placed above the road in Figure 4.3). In addition to this rectilinear placement, we also experimented with random placement of the free joints, distributing them in a spherical or cubic volume surrounding the loads and anchors. We found that random placement did not affect the quality of the final results, but could greatly increase the time needed for convergence.

After generating the free joints, the software automatically makes connections between all three sets of joints, usually connecting each joint to its nearest neighbors using a simple $O(N^2)$ algorithm. Note that during optimization, beams may change strength and position, but new beams cannot be added. In this sense, we are using a ground structure technique, as described in Hemp [53]. The initial structure does not need to be practical or even stable; it is merely used as a starting point for the optimization problem.

## 4.3 Optimizing Truss Structures

The most important property of any structure, truss or not, is that it be stable; i.e. not fall down. For a truss structure to be considered stable, none of the joints can be out of force balance. Because our model consists of rigid beams exerting axial forces only, we can describe the forces acting on any joint $i$ as:

$$\vec{F}_i(\vec{\lambda}) = \vec{g}m_i + \sum_{j=1}^{B_i} \frac{\vec{l}_j}{\|\vec{l}_j\|}\lambda_j \tag{4.1}$$

where $\lambda_j$ is the workless force being exerted by beam $j$, $\vec{\lambda}$ is the vector of these forces for all beams, $\vec{g}$ is the gravity vector, $m_i$ is the mass of joint $i$, $B_i$ is the number of beams attached to joint $i$, and $\vec{l}_j$ is the vector pointing from one end of beam $j$ to the other (the direction of this vector is not important as long as it is consistent).

Given an objective function $G$ (usually the total mass, but perhaps containing other terms), we can optimize a truss structure subject to stability constraints by solving the following problem:

$$\begin{array}{ll} \min & G(\vec{q}) \\ \text{s.t.} & \vec{F}_i(\vec{q}) = 0 \quad i = 1...N_J \end{array} \tag{4.2}$$

where $N_J$ is the number of joints and $\vec{q}$ is the vector of design variables. If we wish to do simple cross-sectional optimization, this design vector is merely $\vec{\lambda}$. If we wish to solve the more interesting

geometry optimization problem, we also include the positions of all the free joints in $\vec{q}$ (see Section 4.3.2 for more details).

In order to avoid physically meaningless solutions, we should also constrain the maximum force that any member can exert:

$$\|\lambda_j\| \leq \lambda_{max} \quad j = 1...N_B \tag{4.3}$$

where $N_B$ is the total number of beams. We constrain the absolute value of $\lambda_j$ because the sign of the workless force will be positive when the beam is under compression and negative when the beam is under tension.

In equation 4.1, we approximate the mass of a joint $m_i$ as half the masses of the beams that connect to it plus, in the case of load joints, whatever external loads may be applied at that joint. Although in reality the mass of a truss structure (exclusive of the externally applied loads) is in its beams rather than its pin joints, this "lumping approximation" is standard practice in structural engineering and is considered valid as long as the overall structure is significantly larger than any component member [55]. This assumption reduces the number of force balance constraints by a factor of two or more, depending on the connectivity of the structure, as well as allowing us to model members as ideal rigid beams.

### 4.3.1 Mass Functions

For a given structural material, the mass of a beam is a function of its shape (length and cross-section). Under tension, the required cross-sectional area of a truss member scales linearly with the force the member exerts [101]. Therefore, the volume of a beam under tension will be a linear function of length and force and, assuming a constant material density, so will the mass $m_T$:

$$m_T = -k_T \lambda_j \|\vec{l_j}\| \tag{4.4}$$

where $k_T$ is a scaling factor determined by the density and tensile strength of the material being modeled, $\|\vec{l_j}\|$ is the length of beam $j$, and $\lambda_j$ is the workless force it exerts (note that $\lambda_j$ here will be negative because the beam is in tension).

Under compression, long slender beams are subject to a mode of failure known as Euler buckling, wherein compressive forces can cause a beam to bend out of true and ultimately fail (see Figure 4.5). The maximum axial compressive force ($F_E$) that can be supported by a beam before it undergoes Euler buckling is governed by the following equation:

$$F_E = \frac{\pi^2 EI}{\|\vec{l_j}\|^2} = \frac{\pi^2 E r^2 A}{\|\vec{l_j}\|^2} \tag{4.5}$$

where $\|\vec{l_j}\|$ is the length of beam $j$, $I$ is its area moment of inertia, and $A$ is its cross-sectional area. $E$ is the Young's Modulus of the material being modeled and $r$ is the radius of gyration, which describes the way in which the area of a cross-section is distributed around its centroidal axis.

Because the cross-sectional area of a member is proportional to the square of $r$, we can rewrite equation 4.5 in terms of $A$ and $r$ as

$$A^2 \propto \frac{F_E \|\vec{l_j}\|^2}{\pi^2 E} \tag{4.6}$$

Because we wish to use beams with minimum mass, $\lambda_j$ (for a given beam $j$) will be equal to $F_E$ and our approximation of the mass function under compression is

$$m_C = \rho A_j \|\vec{l_j}\| = k_C \sqrt{\lambda_j} \|\vec{l_j}\|^2 \tag{4.7}$$

where $\rho$ is the density of the material, $\|\vec{l_j}\|$ is the length of beam $j$, and $\lambda_j$ is the workless force being exerted by the beam. $k_C$ is a scaling factor determined by $\rho$ and the constants in equation 4.6.

Assuming the structures are made of steel I-beams, and using units of meters and kilograms, we use the values of $5 \times 10^{-6}$ for $k_T$ and $5 \times 10^{-3}$ for $k_C$ in equations 4.4 and 4.7 respectively. Because we plan to do continuous optimization, we wish to avoid any discontinuities in the mass function and thus we use a nonlinear blending function between $m_T$ and $m_C$ centered around $\lambda_j = 0$ to smooth the transition.

### 4.3.2 Cross-Sectional and Geometry Optimization

Assuming that the objective function $G(\vec{q})$ in equation 4.2 is merely a sum of the masses of the joints, and that the vector of design variables $\vec{q}$ consists only of $\vec{\lambda}$, we can use the equations developed in the last section to perform a simple version of size optimization. Solving this non-linear, constrained optimization problem will give us, for a fixed geometry, the minimum mass structure that is strong enough to support its own weight in addition to the user-specified external loads.

We are interested, however, in the more useful geometry optimization problem, where both the strengths of the beams and the geometry of the overall structure can be changed. To allow the simultaneous optimization of the sizing and geometry variables, we add the positions of the free joints to the vector of design variables, $\vec{q}$. We do not add the anchor or load positions to the design vector because the locations of these two types of joints are set by the user.

Adding these variables to the optimization problem does not change the form of the equations we have derived, but for numerical stability we now also constrain the lengths of all the beams to be

above some small value:

$$\begin{array}{ll} \min & G(\vec{q}) \\ \text{s.t.} & \vec{F_i}(\vec{q}) = 0 \qquad i = 1...N_J \\ & \|\lambda_j\| \leq \lambda_{max} \quad j = 1...N_B \\ & \|\vec{l_j}\| \geq l_{min} \qquad j = 1...N_B \end{array} \tag{4.8}$$

where $l_{min}$ was set to 0.1 meters in the examples reported here. Because the optimization algorithm we use, sequential quadratic programming, handles inequality constraints efficiently, these length constraints add minimal cost to the solution of the problem.

Similar to the methods discussed in Pederson [96], we use a multilevel design algorithm consisting of two steps. First, we solve the optimization problem described in equation 4.8. Having found a feasible (if not yet globally optimal) structure, the system then merges any pairs of joints that are connected to one another by a beam that is at the minimum allowable length, because these two joints are now essentially operating as one. The system could also, if we wanted, eliminate beams that are exerting little force (i.e. those with small $\|\lambda_j\|$), as they are not actively helping to support the loads. However, we prefer to leave such "useless" beams in the model so as to leave open more topology options for future iterations.

After this topology-cleaning step, the results are examined by the user and, if they are not satisfactory for either mass or aesthetic reasons, the optimization is run again using this new structure as the starting point. In practice, we have found that a single iteration almost always gave us the structures we desired, and never did it take more than three or four iterations of the complete cycle to yield an appealing final result.

### 4.3.3 Objective and Constraint Functions

Although the procedure outlined above generates good results, there are many situations in which we want a more sophisticated modeling of the physics or more control over the final results. Constrained optimization techniques allow us to add intuitive "control knobs" to the system very easily.

For example, in addition to constraints on the minimum length of beams, we can also impose constraints on the maximum length. These constraints imitate the real-world difficulty of manufacturing and shipping long beams. (Due to state regulations on truck flat-beds, girders over 48 feet are not easily shipped in North America.) Other changes or additions we have made to the objective and constraint functions include: minimizing the total length of beams (rather than mass), preferentially using tensile members (cables) over compressive members, and symmetry constraints, which couple the position of certain joints to each other in order to derive symmetric forms.

Another particularly useful class of constraint functions are "obstacle avoidance" constraints, which forbid the placement of joints or beams within certain volumes. In this paper, we have used

two different types of obstacle constraints: one-sided planar and spherical constraints. One-sided planar constraints are used to keep joints and beams in some particular half-volume of space; for example to keep the truss-work below the deck of the bridge shown in Figure 4.7. Implementation of this constraint is simple: given a point on the plane $\vec{r}$ and a normal $\vec{n}$ pointing to the volume that joints are allowed to be in, we constrain the distance from the free joints $\vec{p_i}$ to this plane with $N_J$ new constraints:

$$(\vec{p_i} - \vec{r}) \cdot \vec{n} \geq 0 \qquad i = 1...N_J \tag{4.9}$$

Similarly, to keep the beams and joints outside of a spherical volume, we add a constraint on each beam that the distance between the center of this sphere and the beam (a line segment) must be be greater than or equal to some radius $R$. This distance formula may be found in geometry textbooks, such as Spanier [118]. For optimization purposes, we approximated the gradients of this function with a finite-difference method.

The primary use of obstacle constraints is to allow the user to "sculpt" the final structure intuitively while preserving realism, but they can also serve to produce novel structures by creating local optima. Figure 4.6 shows, from left to right, an initial structure, infeasible because it violates the obstacle avoidance constraint, and two designs produced as solutions from slightly different (random) initial guesses for $\vec{\lambda}$. In each image, the red sphere is the volume to be avoided, the green sphere at the top is the load that must be supported, and the cylinders are the beams, colored cyan or tan depending on whether they are in compression or tension. The anchors are located at the three points where the structure touches the ground. The middle, tripod solution hangs the mass on a tensile member (a cable) from the apex of a pyramid, and the derrick solution on the right supports the mass in a much more complex way (this solution has a mass about 3 times that of the tripod). Both of these solutions are valid and both exist as real designs for simple winches and cranes. Although it is a general concern that nonlinear optimizations can become trapped in sub-optimal local solutions, in our experience this has not been a problem. When, as in the above example, the system produces a locally optimal design, we have found that a few additional iterations of our algorithm are sufficient to find a much better optimum.

## 4.4 Results

We have described a simple, physically motivated model for the rapid design and optimization of models of truss structures. The following examples illustrate the output of this work and demonstrate the realistic and novel results that can be generated.

### 4.4.1 Bridges

Some of the most frequently seen truss structures are bridges. Strong and easy to build, truss bridges appear in a variety of shapes and sizes, depending on their use. A common type of truss bridge, called a Warren truss, is shown in Figure 4.4 with a photo of a real railroad bridge. The volume above the deck (the surface along which vehicles pass) was kept clear in this example by using constraints to limit the movement of the free joints to vertical planes.

The initial guess to generate this bridge was created automatically from thirty user-specified points (the loads and anchors), shown in Figure 4.3. From this description of the problem, the system automatically added 22 free joints (one above each load point) and connected each of these to their eight nearest neighbors, resulting in a problem with 228 variables (22 free points and 163 beams). The final bridge design consists of 48 joints and 144 beams, some of the particles and members having merged or been eliminated during the topology-cleaning step. Similar procedures were used to generate the initial guesses for all of our results.

Using this same initial structure, but with constraints that no material may be placed above the deck, we generated a second bridge, shown in Figure 4.7. Note that the trusswork under the deck has converged to a single, thick spine. This spine is more conservative of materials than the rectilinear trusswork in Figure 4.4, but in the earlier case the constraints to keep the joints in vertical planes prevented it from arriving at this solution.

Another type of bridge, a cantilever truss, is shown in Figure 4.1. As with the bridge shown in Figure 4.7, the cantilever bridge was constrained to have no material above the deck, and the joints were further constrained to move in vertical planes only. However, the addition of a third set of anchor joints in the middle of the span has significantly influenced the final design of this problem. This bridge is shown with a real bridge of the same design: the Homestead High Level Bridge in Pittsburgh, Pennsylvania.

The bridge in Figure 4.8 was generated with the same starting point and the same objective function as that in Figure 4.1, but without the “clear deck” and vertical-plane constraints. Removal of these constraints has allowed the structure to converge to a significantly different solution, called a through-deck geometry.

### 4.4.2 Eiffel Tower

A tall tower, similar to the upper two-thirds of the Eiffel Tower is shown in Figure 4.9. This tower was optimized from an initial rectilinear set of joints and beams, automatically generated from eight user-specified points (the four anchor sites and a four loads at the top). We concentrated on the top two-thirds of the Tower because the design of the bottom third is dominated by aesthetic demands. Similarly, the observation decks, also ornamental, were not sythesized.

### 4.4.3 Roof Trusses

The frameworks used to support the roofs of buildings are perhaps the most common truss constructions. We have generated three different types of roof trusses for two different roof pitches. In Figure 4.10 we show each category of truss (cambered Fink, composite Warren, and Scissors) in its own column, at the top of which is an illustration of a real example. For a given pitch, all three types of trusses were generated from the the same initial geometry. The variation in the results is due to different objective functions: total mass (cambered Fink and Scissors) and total length of beams (composite Warren), and different roof mass (the Scissors trusses have a roof that weighs twice as much as the other two types of trusses).

### 4.4.4 Michell Truss

The Michell Truss is a well-known minimum-weight planar truss designed to support a single load with anchors placed on a circle in the same plane Although impractical because of the varying lengths and curved beams needed for an optimal solution, the Michell truss has been a topic of study and a standard problem for structural optimization work for nearly a century. We have reproduced the Michell truss with our system, starting from a grid-like initial guess and arriving at a solution very close to the analytical optimum (Figure 4.11).

### 4.4.5 Timing Information

Sequential quadratic programming, relying as it does on the iterative solution of quadratic subproblems, is a robust and fast method for optimizing non-linear equations. Even with complicated problems containing thousands of variables and non-linear constraints the total time to optimize any of the above examples from auto-generated initial guesses varied between tens of seconds and less than fifteen minutes on a 275MHz R10000 SGI Octane. Specifying the anchor and load points and the locations of obstacles (if any) rarely took more than a few minutes and with better user-interface design this time could be significantly reduced. For comparison, we timed an expert user of Maya as he constructed duplicates of the cooling tower (Figure 4.2), the Warren truss bridge (Figure 4.4) and the tower shown in Figure 4.9 from source photographs of real structures. We found that modeling these structures at a comparable level of detail by hand took an hour for the cooling tower, an hour and a half for the bridge and almost three hours for the Eiffel Tower. Although informal, this experiment showed that our method has the potential to speed up the construction of models of truss structures enormously, while simultaneously guaranteeing physical realism.

## 4.5 Summary and Discussion

We have described a system for representing and optimizing trusses, a common and visually complex category of man-made structures. By representing the joints of the truss as movable points, and the links between them as scalable beams, we have framed the design as a non-linear optimization problem, which allows the use of powerful numerical techniques. Furthermore, by allowing the user to change the mass functions of the beams, the objective function, and the constraints, we can alter the design process and easily generate a variety of interesting structures.

Other than the location of anchors and loads, the factor that we found to have the largest effect on the final results was the use of obstacle avoidance constraints. Placement of these constraints is a powerful way of encouraging the production of certain shapes, such as the volume inside a cooling tower or the clear traffic deck on a railroad bridge.

Not surprisingly, we also found that the number of free joints added during the initial model construction could affect the look of the final structure. However, this effect was largely one of increasing the detail of the truss-work, rather than fundamentally changing the final shape. The initial positions of the free joints, however, made little difference to the final designs, although they could affect the amount of time required for the optimization. In cases where the initial positions did make a difference, it was generally the result of constraints (such as obstacle avoidance) creating a "barrier" to the movement of free joints during optimization and thus creating local minima. In our experience, however, a few additional iterations of the algorithm were sufficient to get out of these local minima and find a global solution.

We also found that beyond some minimum number of beams per joint (three or four), additional connections to more distant joints had a negligible effect on the final designs. We attribute this lack of impact to two factors. First, because we are minimizing the total mass of the structure (or occasionally the total length of all members), shorter beams connected to closer joints will be used more readily than longer ones, especially if they are under compression. Secondly, by allowing unused beams to "fade away" as the force they exert drops to zero, initial structures with many beams per joint can become equivalent to structures with fewer beams per joint, allowing both more and less complex initial structures to converge to the same answer.

Although successful at capturing the geometric and topological complexity of truss structures, our work does not account for all the details of true truss design. For example, a better objective function would calculate the actual cost of construction, including variables such as connection costs (the cost to attach beams to a joint) and the cost of anchors, which varies depending on terrain and the force they must transmit to the ground. Nor does our model explicitly include stress limits in the materials being modeled (although the constants $k_T$ and $k_C$ in equations 4.4 and 4.7 are an implicit approximation). Similarly, a more complex column formula than the simplified Euler buckling

formula, such as those described in Popov[101], might capture more nuances of real design.

True structural engineering must also take into account an envelope of possible load forces acting on a structure, not a single set as we have implemented. This envelope of forces results from varying loads, such as a different number of cars on a bridge, or loads applied in different directions than the dominant one, such as the effect of wind on a utility tower. The geometry of truss structures originates in a catenary shape, called the *funicular polygon*, which corresponds to the curve a rope would possess when anchored at fixed locations and loaded by a fixed set of forces. The funicular polygon is most recognizable as the shape of suspension bridges and arches, and is the ideal geometry for a fixed loading, such as we have assumed in our work. When seeking to make a three dimensional truss structure stable under varying loads, structural designers usually add additional, triangulating elements to the basic funicular geometry in order to brace it. While effective, this technique is unlikely to produce truly optimal truss structures, and will does not create interesting or novel solutions.

Although the technique described in this chapter does not generate truss structures which are stable under variable loadings, we believe it could be extended to do so. One extension that might accomplish this goal is an iterative application of our current technique, using the same fundamental structure but different loadings at each of the six iterations. Specifically, the first iteration would solve for an optimal structure for a single loading in the standard way, with the exception that we would not "clean up" any beams that are exerting tiny forces (i.e. those with small $\|\lambda_j\|$). The resulting structure would then have a new set of loads applied to it, opposite to the first, and a second solution would be found this time using cross-sectional optimization only. The use of cross-sectional optimization allows us to strengthen the structure against variable loadings, but would not permit the structure to become merely a new funicular polygon. The third iteration would similarly be of cross-sectional optimization, but with loads orthogonal to the forces used in the first iteration. A fourth iteration, with applied loads opposite the third, would be similarly solved, and so forth. Eventually, we would have six closely related solutions with the same topology which could be "averaged" or added together in some way to find the aggregate structure that could support an envelope of possible loads.

A second possible technique for solving for more stable, and thus more realistic, truss structures would require a more fundamental change in our method. Instead of using a standard optimization algorithm to solve for the optimal structure, we could apply a multiobjective optimization technique to find the Pareto-optimal truss structure for multiple loads. Multiobjective optimization is a class of techniques, such as suitably altered sequential quadratic programming or linear programming algorithms, that balance the demands of multiple objective functions [84]. Our objective functions would perhaps be measures of stability in the six coordinate directions, rather than a simple function of mass. The force balance constraints would also have to be changed, perhaps becoming inequality constraints that would allow the structure to be slightly over-engineered in exchange for greater

stability.

Eventually, we would like to be able to include more abstract aesthetic criteria in the objective function. Our current system incorporates the concepts of minimal mass and symmetry (via constraints), but many elements of compelling design are based on less easily quantifiable concepts such as “harmony,” the visual weight of a structure, and use of familiar geometric forms. Because our technique is fast enough for user guidance, implementing even a crude approximation to these qualitative architectural ideals would allow users more flexibility in design and the ability to create more imaginative structures while still guaranteeing their physical realism.

Figure 4.1: A cantilever bridge generated by our software, compared with the Homestead bridge in Pittsburgh, Pennsylvania.

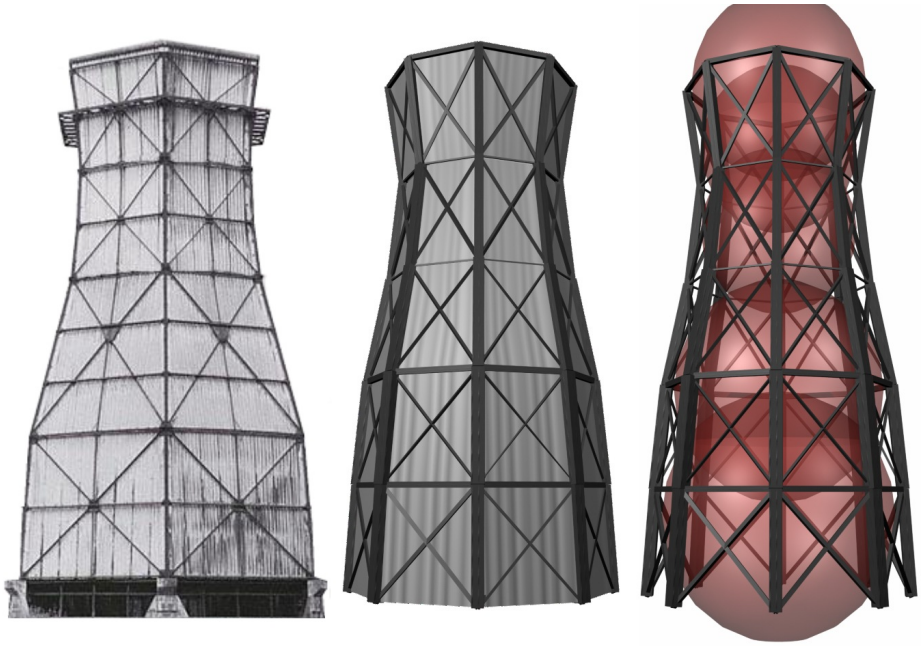

Figure 4.2: A cooling tower at a steel mill created by our software compared with an existing tower. From left to right: a real cooling tower, our synthesized tower, and the same model with the obstacle constraints shown.

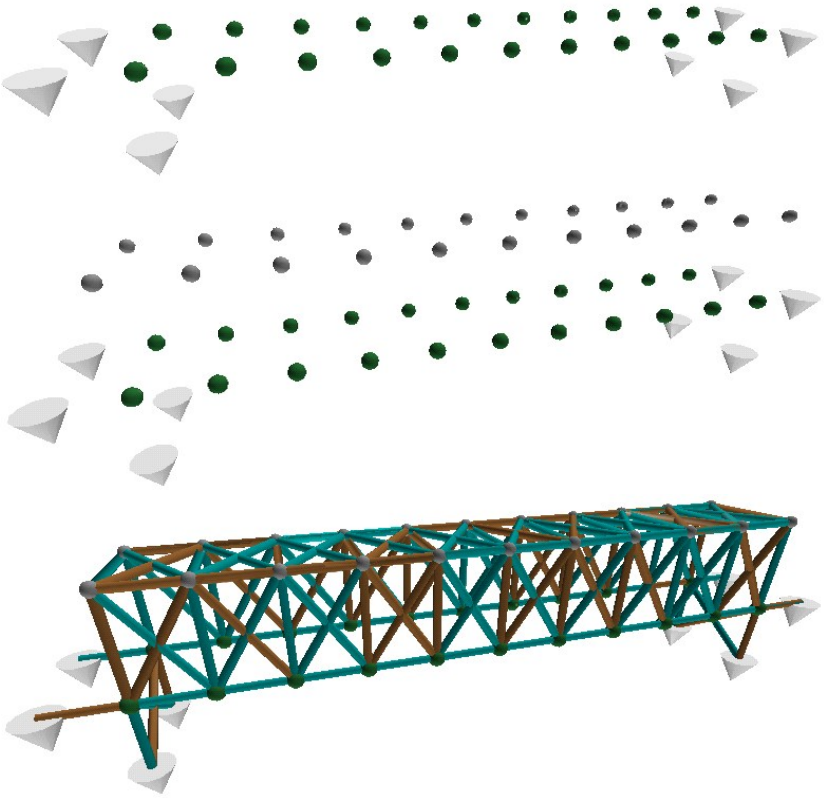

Figure 4.3: From top to bottom: the data specified by the user (loads are depicted as green spheres and anchors as white cones); the free joints added by the software above the loads; the automatically generated initial connections (beams). This structure was the initial guess used to create the bridge shown in Figure 4.4.

Figure 4.4: A typical railroad bridge and similar truss bridge designed by our software.

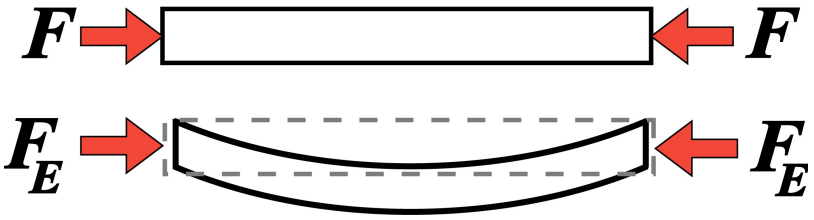


Figure 4.5: A depiction of Euler buckling under a compressive load.

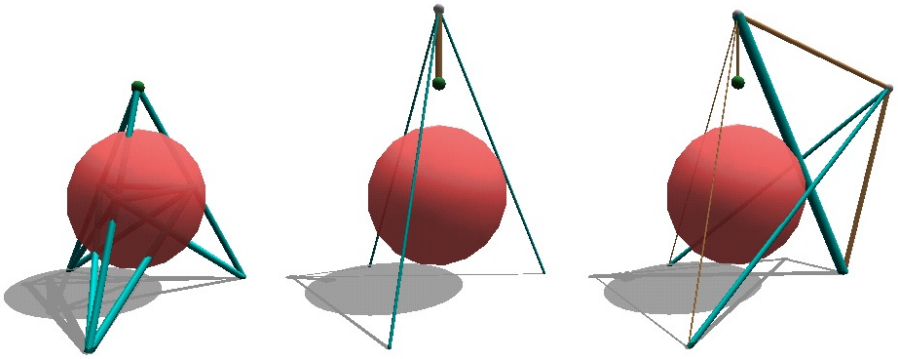

Figure 4.6: From left to right: initial structure, tripod solution, derrick solution.

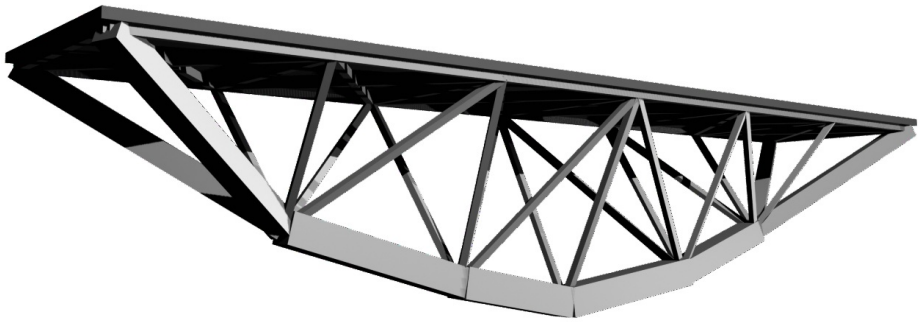

Figure 4.7: A bridge with all trusswork underneath the deck.

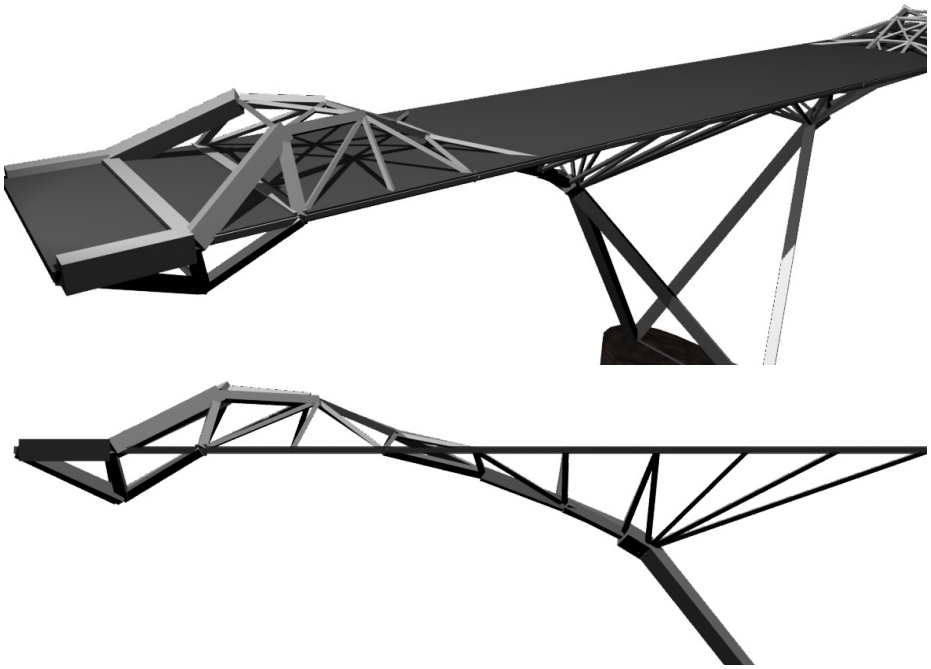

Figure 4.8: A perspective and side view of a through-deck cantilever bridge.

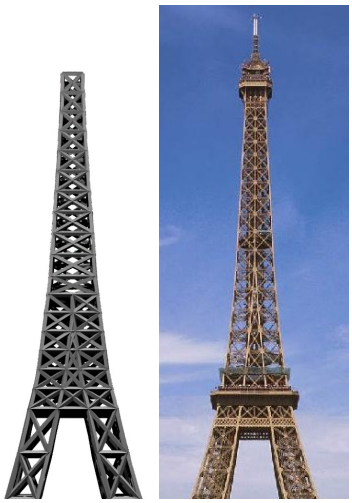

Figure 4.9: Our trusswork tower, compared with a detail of the Eiffel Tower. Because they are ornamental and not structural, the observation decks are not included in our tower.

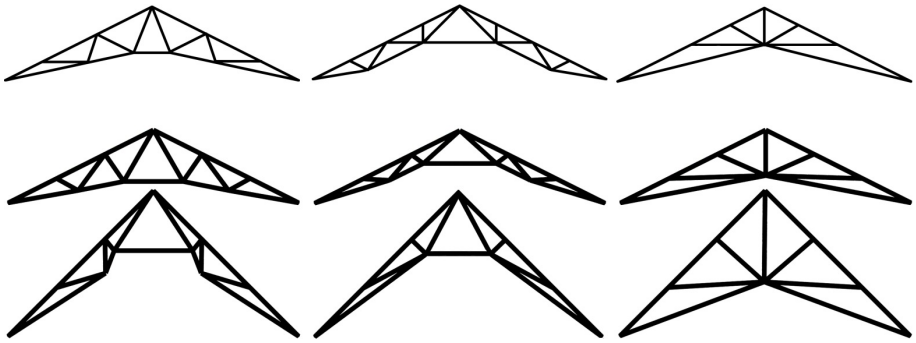

Figure 4.10: Three types of roof trusses: from left to right, cambered Fink, composite Warren, and Scissors. Illustrations of real trusses are shown at the top of each column. The lower two trusses in each column were generated with our software.

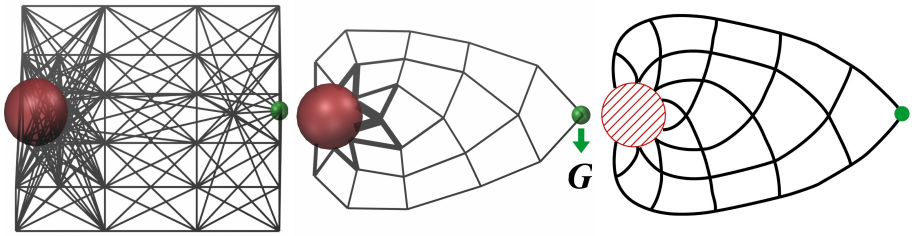


Figure 4.11: From left to right: The initial structure from which we began the optimization, our final design, and an optimal Michell truss after an illustration in Michell [83]. The red sphere on the left of each image is an obstacle (on the surface of which are the anchors), and the green sphere on the right is the load which must be supported. Gravity points down.

# Chapter 5

# Constant Mean Curvature Structures

## 5.1 Introduction

The third computer animation and modeling problem that we examine in this thesis is the task of generating *constant mean curvature structures*. Constant mean curvature (CMC) structures are a broad class of natural and man-made objects whose shape is primarily determined by some surface-area minimizing process working in concert with constraints on the curvature of the surface. Examples of such objects are soap bubbles, as in Figure 5.1, and simple films, such as that shown in Figure 5.3. In this latter image, the soap film "seeks" to minimize its surface area while maintaining its boundary conditions: contact with the wire loops that bound it. The shape of this membrane is further constrained by the physical properties of elastic films, which require that the surface's *mean curvature*, a scalar measure of how a surface locally bends, remain constant across the entire surface. To create this example, the user specified a simple initial guess, shown at the top-right of Figure 5.3 and labeled the vertices that are affixed to the wire loops. The optimizer then minimized the surface area by varying the locations of the non-fixed vertices. Because of the complex interaction of the minimizing process, geometric constraints, and curvature constraints, capturing the structural nuances of constant mean curvature objects is difficult to do with standard modeling tools in all but the most simple cases, and an automated procedure for generating them is desirable.

### 5.1.1 Background

Within the civil engineering and architectural community, a great deal attention has been paid to the analysis of surface-area minimizing structures in general, and constant mean curvature structures in particular (see, for example, Schueller[113] for a bibliography of work in these areas). Civil engineers have been primarily interested in the design and analysis of minimum-area surfaces, such as tents [52], although work has also been done on pneumatic structures [29], such as inflatable domes

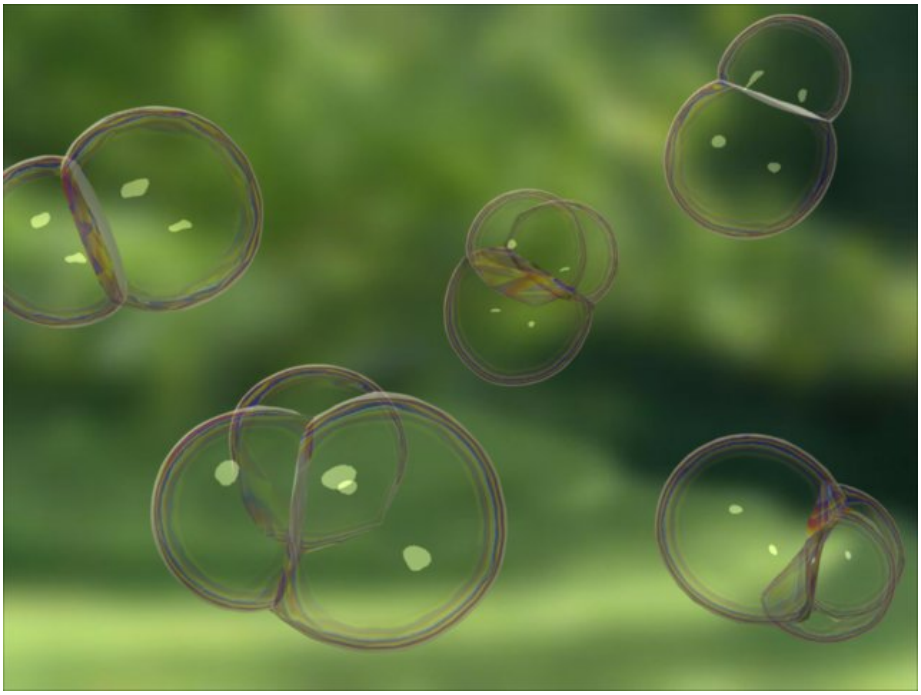

Figure 5.1: A group of double and triple bubbles.

and enclosures. In practice, engineers and architects are generally interested in designing smooth tensile-membrane structures, such as those described by Otto[93], rather than truly mathematically optimal structures. However, formulating this requirement of "smoothness" as a more easily mathematically expressed demand for optimality allows us to apply powerful optimization techniques and quickly generate designs meeting the required constraints.

Mathematicians have also paid a great deal of attention to generating and analyzing constant mean curvature objects. Much of this work, for example Hass and his colleague's proof of the double-bubble conjecture [51], has been concerned with proving the existence of a globally optimal energy minimizing surface for particular conditions. One of the best known examples of this type of problem is Lord Kelvin's volume-filling polyhedron [123], which he claimed was the minimum area, unit volume structure that, when repeated, would gaplessly fill an unbounded volume. His proposed structure has only recently been shown by Phelan and his colleagues[97] to be slightly less than globally optimal. Work has also been done proving that globally optimal surfaces under certain conditions must have particular geometric properties [71, 68, 77]. In a more applied vein, Brakke's surface evolver program [20] was written in order to generate many kinds of minimum-energy structures, although it uses a simpler optimization technique than our method and does not handle arbitrary geometric constraints on surfaces.

Polthier and his colleagues have also written about the mathematics of constant mean curvature

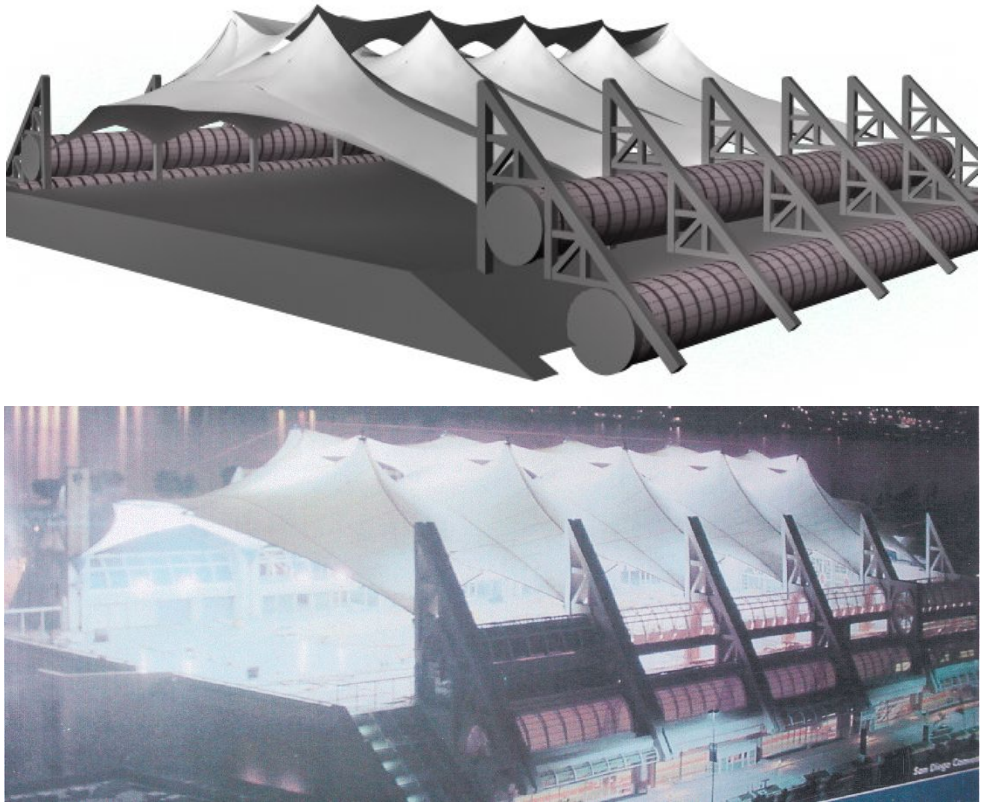

Figure 5.2: Our model of the membrane structure that forms the roof of the central exhibition space of the San Diego Convention Center, compared with a photo of the real structure, used courtesy of the San Diego Convention Center Corporation.

surfaces. His collaborations with Pinkall [99] and Oberknapp [88] described a method for generating simple constant mean curvature surfaces. Unlike our technique, Oberknapp and Polthier do not address the problem in Cartesian space, but rather transform it from $\mathbb{R}^3$ into $\mathbb{S}^3$. This change in the representation allows them to avoid certain mesh issues that techniques that operate in $\mathbb{R}^3$ must deal with, but also restricts their work to surfaces and structures with free boundaries; that is, surfaces without general geometric constraints.

In the field of computer graphics, less work has been done on the topic of generating general constant-mean curvature structures, although techniques for solving specific cases, such as two-dimensional soap films and merged soap bubbles [39], have been developed. Ďurikovič[30] described a dynamic method, based upon discretized equations of elastic deformation, which could model the motion and contact of single, double and triple bubbles. This work focused on issues of animation instead of modeling, and thus used simple geometric construction methods to produce the bubble geometries.

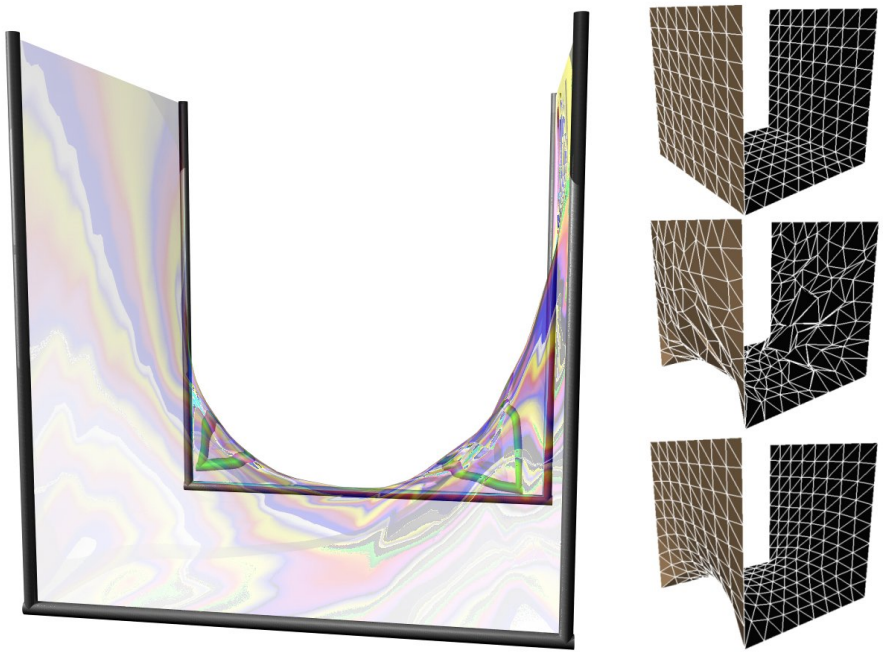

Figure 5.3: A soap film formed between two U-shaped wire boundaries. The initial geometry, an intermediate step in the optimization, and the final geometry are shown on the right.

While creating realistic models of minimum-area, constant-mean curvature objects can be challenging, rendering these objects is generally no more difficult than rendering the materials from which they are made (for example, the canvas of a circus tent). However, the specific task of rendering soap films and bubbles has been studied in more detail, not only because of their aesthetic qualities but also because of their interesting physical properties. Icart and Arquès[59] and Kück and his co-authors[72] have developed methods for realistically animating and rendering so-called *wet foams*: collections of soap bubbles mixed liberally with liquid. Glassner[40] has also worked through the physics of thin, transparent films in order to derive both a physically based and a heuristic technique to render soap bubbles and films.

## 5.2 Modeling CMC Objects

Although constant mean curvature structures are diverse, including objects as different as soap bubbles and tent structures, they can be generally represented as a collection of surfaces, possibly bounding one or more volumes. These surfaces can be manifold, intersect, be attached to immobile boundaries, or have any number of other geometric properties in addition to their defining quality: a constant mean curvature.

Intuitively, mean curvature can be thought of as a measure of how a surface bends in space. Mathematically, if we consider a 2-manifold embedded in three dimensions, we can locally approximate every point on the surface with a tangent plane, orthogonal to the normal $\vec{n}$ at that point. For every unit direction $\vec{e}_\theta$ in that tangent plane, the *normal curvature* $\kappa_\theta$ is the curvature of the curve that belongs to both the surface and the plane formed by $\vec{e}_\theta$ and $\vec{n}$. The mean curvature at a point, then, is simply the average of all the normal curvatures at that location:

$$H = \frac{1}{2\pi} \int_0^{2\pi} \kappa(\theta) d\theta \tag{5.1}$$

Thus, when we describe a surface as having a *constant mean curvature*, we mean that $H$ has the same value at every point on the surface.

Because our representation of constant-mean curvature structures is based on surface properties, it is important that we in turn use an appropriate mathematical model for these surfaces. We have chosen to represent the surfaces of our constant-mean curvature objects as triangulated meshes (a simplicial complex). Not only are polygonal meshes a common representation, making their creation and rendering easier, but they offer the flexibility to represent intersecting, non-manifold surfaces. In order to compute differential geometric quantities such as mean curvature, we use the discrete operators described by Meyer and his co-authors [82].

Specifically, we used their discrete reformulation of the equation for mean curvature of a continuous surface given in Equation 5.1. To reformulate this operator, Meyers and his colleagues assume that a triangulated surface is a linear approximation of some smooth, "true" surface. This assumption allows them to then define pointwise geometric quantitites of the surface, such as curvature, as spatial averages around vertices.

In order to calculate the mean curvature of a particluar vertex, $H(x_i)$, we use the following equation from their paper:

$$H(x_i) = \frac{1}{4A^*} \sum_{j \in N_1(i)} (\cot \alpha_{ij} + cot\beta_{ij})(\|x_j - x_i\|) \tag{5.2}$$

where $N_1(i)$ is the *1-ring neighborhood*, the set of vertices connected to $x_i$ by an edge, illustrated in Figure 5.4. $\alpha_{ij}$ and $\beta_{ij}$ are the opposing angles of the edge connecting $x_i$ and some 1-ring neighbor $x_j$, as shown in Figure 5.5. $A^*$ is the *mixed area* around a particular vertex. Each triangular face connected to vertex $x_i$ contributes a fraction of its total area to the mixed area of the central vertex. The exact area contributed by a particular face depends on whether that face is an obtuse triangle, and whether the angle at $x_i$ is itself obtuse. The rules for calculating $A^*$ are given in the original paper.

For a particular vertex of a triangular face, Meyer defines the *voronoi region* as the area defined by that vertex, the midpoints the adjacent sides, and the circumcenter of the face . The vornoi region

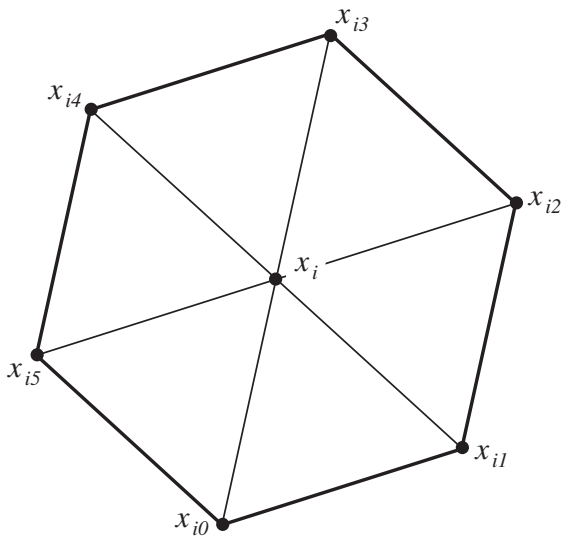


Figure 5.4: The 1-ring neighborhood of $x_i$, consisting of the vertices $x_{i0}$ through $x_{i5}$.

for the vertex $x_i$ is shown as the shaded region in Figure 5.6. The formula for calculating this area for a vertex $x_i$ and a neighboring vertex $x_j$ is

$$A = \frac{1}{8}(\|x_{i0} - x_i\|^2 cot\angle x_{i1} + \|x_{i1} - x_i\|^2 cot\angle x_{i0}) \tag{5.3}$$

Meyers and his co-authors show that these equations will converge to the pointwise (continuous) definition as the mesh elements decrease in size.

## 5.3 Optimizing CMC Objects

Constant mean curvature objects are created by the interaction of a minimizing process and a set of geometric constraints. Thus the construction of CMC objects lends itself to formulation as an optimization problem. In order to generate CMC structures, we wish to minimize total surface area while maintaining a constant mean curvature on each surface. Thus, our vector of design variables, $\vec{q}$, contains the locations of the vertices that make up the surfaces. In addition to these vertex coordinates, we also include variables to represent the mean curvatures, $H_i$, for each surface, $S_i$. Note that a single structure, such as the double bubble shown on the left of Figure 5.7, can be made of multiple surfaces. In this particular case, there are three surfaces: one for each part-spherical bubble and one for the "shared wall" between them. Each of these surfaces has an independent mean curvature; that is, the final curvature of the nearly flat shared wall is not necessarily the same as the curvature of either of the highly curved part-spherical surfaces.

Our objective function, $G(\vec{q})$ is the total area of all the surfaces that comprise the object, and our

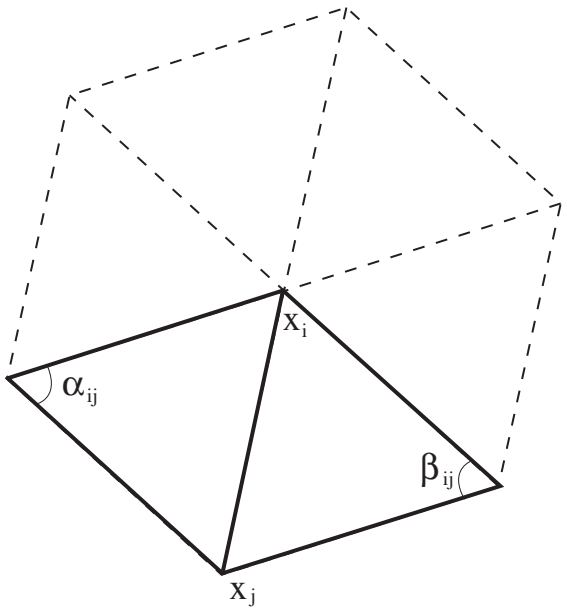


Figure 5.5: The external angles of the edge connecting $x_i$ and $x_j$.

constraint functions, $C_j$, are functions that compute the mean curvature at a specific point $j$. Using this nomenclature, our optimization problem can be written as:

$$\begin{aligned} \min \quad & G(\vec{q}) = \sum_{i=1}^{N_S} \text{Area}(S_i) \\ \text{s.t.} \quad & C_j(Q_i) = H_i \qquad\qquad i = 1...N_S \\ & \qquad\qquad\qquad\qquad\quad j \in Q_i \end{aligned} \tag{5.4}$$

where $N_S$ is the number of distinct surfaces that make up the object. $Q_i$ is the set of vertices where curvature may be computed that are part of surface $S_i$. A particular vertex may be excluded from this set for two reasons. First, the user can label the vertex as one which should be ignored; for example, the point where a tent is supported by a pole will have a curvature different from the fabric which surrounds it. Second, the vertex can be located at an intersection of surfaces and thus will have no valid curvature (curvature can only be computed at points that are locally manifold).

### 5.3.1 Geometric Constraints

Most real-world constant mean curvature objects have other constraints besides area and curvature that must be satisfied. For example, a single soap bubble is constrained not only in curvature, but also to enclose a specific volume. Similarly, a circus tent is "constrained" with stakes to touch the ground at certain points and lifted above the ground by supporting poles at others. Because of the flexible nature of constrained optimization, including such constraints in our formulation is simple.

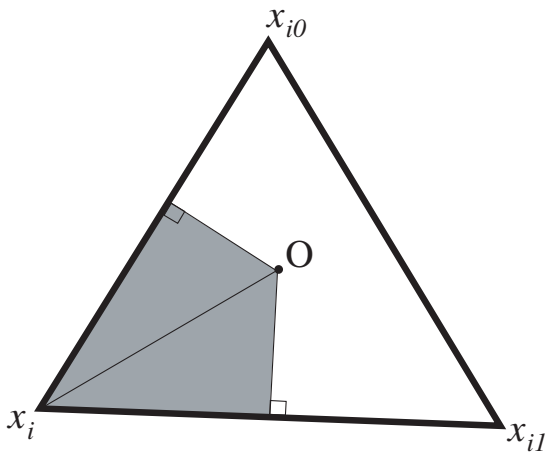


Figure 5.6: The voronoi region of of vertex $x_i$, for a the face formed with 1-ring neighbor vertices $x_{i0}$ and $x_{i1}$.

For the simplest type of boundary constraints, such as the ground stakes securing a circus tent or the wire loop enclosing a soap film, we can "constrain" the positions of vertices by simply not including them in our design vector. More complex boundary constraints, such as constraining the length of a particular chord on the surface or the volume enclosed by a set of surfaces, can be added to our system with appropriate constraint functions.

### 5.3.2 Constructing the Initial Structure

As we have discussed before, optimization techniques require some starting point or initial guess at the solution, which is then gradually improved with respect to the objective function. For the problem of optimizing constant mean curvature objects, this initial guess is a set of surfaces with the same topology, although not necessarily the same geometry, as our final solution. We construct this initial object with standard polygonal modeling tools, including Maya as well as our own software. Starting points for the optimization can be as accurate or as crude as the user wants, although cruder guesses (structures that are dissimilar to the eventual solution) may take longer to converge. For most of the examples given in this chapter, the initial structures were created in only a few minutes using the polygonal modeling tools in Maya. The most complex initial structure was that created for our model of the San Diego Convention Center (Figure 5.2). Because of the complexity of this structure and the difficulty in interpreting the architectural drawings we used as a reference, the initial structure took a little over two hours to construct, although the optimization to the final structure took less than five minutes.

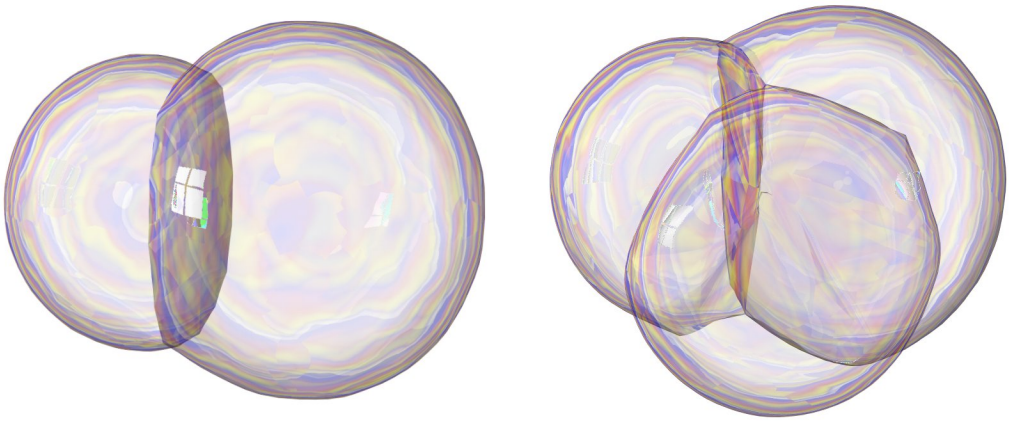

Figure 5.7: From left to right, a double and triple bubble.

### 5.3.3 Solving the Optimization Problem

The optimization problem formulated in Equation 5.4 is another example of nonlinearly constrained, nonlinear optimization. As in previous chapters, we use a quasi-Newton sequential quadratic programming technique to solve this problem. Many of our constraints, such as the crucial constant mean curvature constraints, have no easy closed-form representation. Rather than attempt to derive one, we instead used finite difference methods to approximate the gradients of these constraints. Unlike in the case of gait synthesis, however, the finite difference approximations were not dependent on a particularly expensive process and so the gradient approximations were not costly. Furthermore, because these constraint calculations are not dependent on a limited accuracy function (such as physical simulation) our finite difference approximated gradients were significantly more accurate than those calculated for gait synthesis.

## 5.4 Results

We have described a physically motivated technique for constructing constant mean curvature, minimum-area structures. The following examples illustrate the results of this work and demonstrate the realistic and novel results that can be generated.

### 5.4.1 Soap Films and Bubbles

One of the most common, and perhaps the prettiest, types of constant mean curvature structures are soap bubbles and films. Glassner [40] found that simple rendering techniques produced results as visually appealing as more complex, physically based methods. For the following examples, we have chosen to use a heuristic rendering technique for soap films and bubbles based upon his work

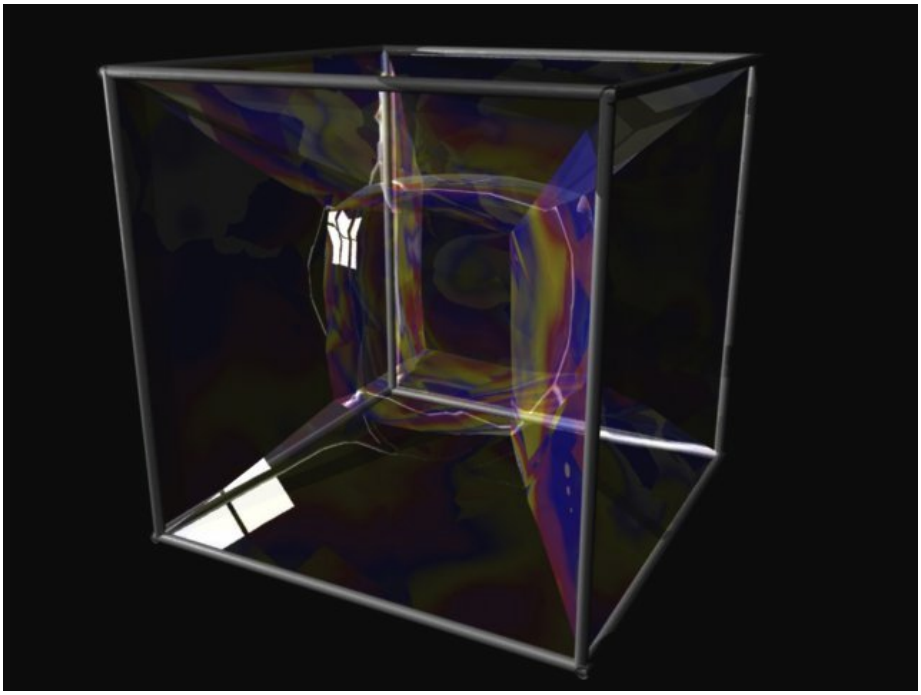

Figure 5.8: The soap film formed by a cubical framework.

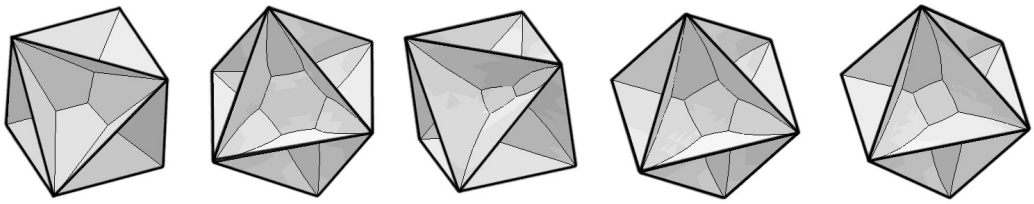

Figure 5.9: The five distinct, stable soap films bounded by an octahedral wire frame. Shown in order, from left to right, of least to greatest total surface area.

as well as that of Sullivan[121].

If a simple wire frame, such as the two U-shaped wires shown in Figure 5.3 are dipped in soap film, a saddle-shaped bubble will result. This simple film has curvature and boundary constraints (the film must remain attached to the wire frame) but no other geometric constraints. Although a simple wire boundary such as this has only one possible minimum area surface, more complex boundaries may have many. For example, a regular octahedral boundary has five topologically different, stable, constant-mean curvature surfaces that touch all twelve edges [62]. These five surfaces, shown in Figure 5.9, are local minima of the surface area, and demonstrate the variety of CMC surfaces that exist for even simple boundary constraints.

An example of a soap bubble with volume constraints is the “double bubble” shown on the left in Figure 5.7. Here, two soap bubbles share a common surface while maintaining independent

Figure 5.10: The top of a circus tent generated by our software, compared with a real tent. The lower, draped part of the tent could be modeled with existing cloth simulation techniques. Photograph © 2003 FreeFoto.com. Used with permission.

volumes (of one and two cubic centimeters, respectively). A similar collection of three soap bubbles is shown on the right of this figure.

Figure 5.8 shows a structure that has boundary constraints as well as a volume constraint: a complex bubble within a cubical frame. Inside the wire cube, and connected to its edges with films, is a small cubical bubble with piecewise-spherical walls. In the real world, the size of this central bubble would be determined by physical conditions at the time the wire frame is dipped in soap solution, such as the viscosity of the soap fluid and the angle at which the frame enters and exits the liquid. For our simulation, we specified the desired volume of this bubble as an optimization constraint.

### 5.4.2 Tents and Membrane Structures

The construction of tents can be also mathematically formulated as area minimizing objects with multiple geometry constraints but no bounded volumes. The top of the circus tent, shown in Figure 5.10, is lifted at each end by a flagpole, supported by another set of poles roughly halfway between the center and the rim of the fabric, and secured around the rim by a third set of supporting poles. Each of these supports was modeled as a geometric constraint.

Figure 5.11: Our model of a pneumatic structure, compared with a real structure enclosing heated tennis courts.

A more complicated set of tent structures composes the exhibition space of the San Diego Convention Center (see Figure 5.2). This complex membrane structure is composed of five repeating peaked structures, two overhanging membranes at either end, and an upper tent structure that runs down the center. Working from a set of architectural drawings, we reconstructed the geometry constraints on each of these surfaces, including boundaries for each piece of "fabric" and the supporting poles for the peaked tent structures. Our optimization algorithm worked within these geometric constraints to produce a structure strikingly similar to the original.

### 5.4.3 Pneumatic and Cushion Structures

Air-filled buildings, often called pneumatic or cushion structures, are another type of man-made, constant mean curvature structure. Less common than tents, pneumatic structures are still frequently used as temporary buildings or enclosures, such as the tennis-court enclosure shown in Figure 5.11. Pneumatic structures can also be quite large, as with the model of a proposed exhibition space, designed by Frei Otto[93], shown in Figure 5.12. Our model of this exhibition hall was built from eight separate pneumatic cells: three large domes surrounded by five smaller inflated enclosures.

### 5.4.4 Timing Information

Although flexible and powerful, optimization can be slow, especially when both the objective and constraint functions are non-linear. However, the technique we have used to solve our optimization problems, sequential quadratic programming, is generally considered to be both fast and robust. We have found that, even with complicated problems containing tens of thousands of variables and

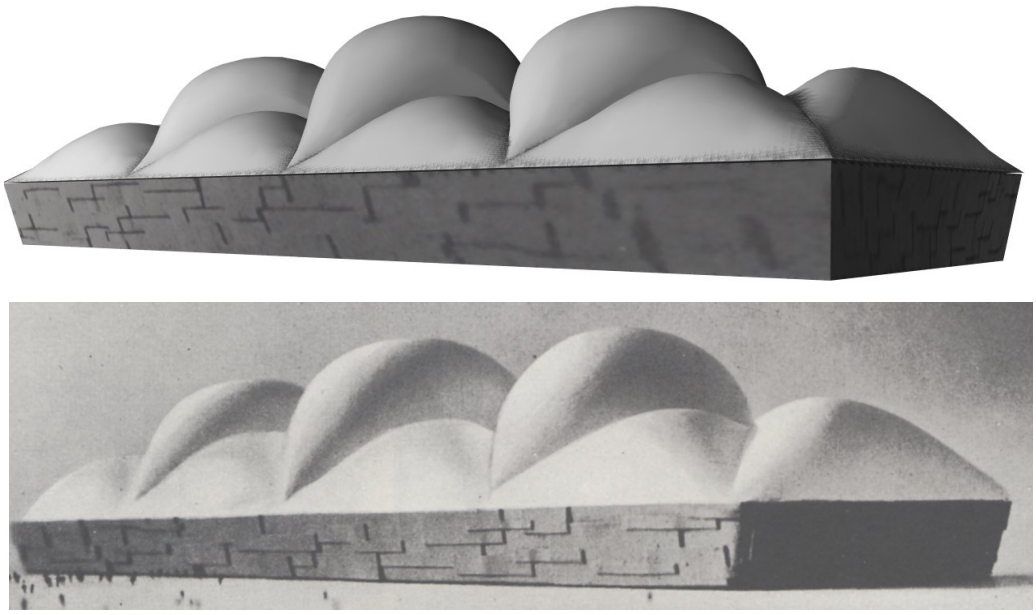

Figure 5.12: A pneumatic exhibition space. Our model is shown on top, and a photograph of the architect's model of the proposed structure is shown below. Photograph © 2003 MIT Press. Reprint permissions applied for.

non-linear constraints, the total time to optimize any of our examples from initial guesses varied between tens of seconds and less than ten minutes on a 1.7 GHz Pentium 4, running Linux.

## 5.5 Summary and Discussion

We have described a system for representing and optimizing constant mean curvature structures, a common category of natural and man-made objects. By using the locations of vertices as variables, and the mean curvature requirement as a set of constraints, we framed the task of generating models of CMC structures as a non-linear optimization problem. In addition to allowing us to use powerful numerical methods to generate these complex structures, optimization techniques also let us easily incorporate intuitive constraints, such as the boundary geometry and required volumes, to the design process. Our results were all generated using curvature constraints and some combination of boundary, volume, and chord-length constraints. However, other geometric constraints may be similarly added to the optimization problem.

A drawback common to many optimization techniques is their long run-times. Nonlinear optimization can often be slow, and will become slower as the number of variables increases. Although our constant mean curvature structure optimization problems ran fairly quickly, one way in which

we could further improve performance would be to use analytic formulations of the curvature constraints, or the technique of automatic differentiation, in order to calculate gradients more efficiently. This change would increase development time, but it is likely that we would gain a significant speed-up, especially for large problems.

Although difficulties with local minima are common to many nonlinear optimization tasks, we did not run into any significant issues of this type while generating CMC structures. We believe that our lack of difficulties with local minima was partly due to the smooth nature of the problems' optimization landscapes and partly due to the ease of creating well-behaved initial structures. The most important factor, however, was the formulation of our optimization problems. Specifically, the topology of the structures could not change during optimization, which meant that local minima that are topologically different from the initial structure were impossible to achieve. A disadvantage of this formulation is that we cannot use a single initial structure to fully explore the space of possible CMC structures for a particular set of constraints. Instead, topologically different local minima, such as those shown in Figure 5.9, must be generated from initial structures with matching topologies. However, the problem of finding all local minima CMC structures for particular geometric conditions is an extremely difficult one with no general, or even widely applicable, solution reported in the literature.

Despite its advantages of flexibility and intuitive control, it might seem inefficient to use nonlinear optimization techniques to generate CMC structures, which are sometimes geometrically quite simple. At first glance it could appear that the same design tasks could be accomplished by the use of a simpler technique, such as dynamic simulation with a spring and point-mass system. While attractive because of the simplicity of implementation, there are several reasons why a spring and mass system could not easily duplicate the results we have shown. One problem is that the behavior of spring/mass meshes is quite sensitive to the connectivity of the springs, and irregular or anisotropic meshes will result in irregular results. This problem can be avoided if the initial mesh is carefully balanced, or if the spring parameters are appropriately tweaked, but both of these work-arounds require some foreknowledge of the final structure's shape and may require impractical amounts of work. Another, more important, flaw in a spring/mass approach is that it is quite difficult to impose arbitrary geometric constraints, such as constrained areas, chord lengths and volumes, on dynamic systems.

Although not well-suited for modeling complex constant mean curvature structures, dynamic simulation can perform a complementary role when paired with an optimization-based modeling approach such as our own. For example, the discretized elastic method described by Ďurikovič[30] could be applied to the models generated by our technique, allowing us to simulate a wide array of soap bubbles and films. Similarly, cloth simulation techniques such as that described by Bridson and his co-authors[21], could be used in conjunction with our models in order to simplify the simulation of complex cloth structures, such as tents and pneumatic buildings.

# Chapter 6

# Conclusions and Future Work

At the end of each of the preceding three chapters, we discussed the shortcomings and successes of using optimization to solve particular problems. Some of these observations are specific to one problem, such the difficulty of smoothly simulating ground contact for gait generation. However, there are also broader lessons, applicable outside these narrow problems, that we can draw from our experiences. In this chapter, we will elaborate on what we learned during our research and attempt to make larger conclusions about the utility of optimization in computer graphics.

One observation we have made, common to each of the three problems we address in this thesis, is that "initial guesses count." In each of these research projects, we observed that the success or failure of optimization was highly dependent on the quality of the initial guess fed to the optimizer. On the face of it, this conclusion seems obvious: high quality initial guesses are, by definition, already close to a correct answer and thus easy to solve. Similarly, very poor initial guesses may be so distant from a region of feasibility or optimality that no algorithm could find an acceptable answer. However, this simple tautology is not the primary reason that good initial guesses were so important. Rather, the quality of the initial guesses has such a large effect on the quality of the output because of local minima in the problem space.

Common to all nonlinear optimization problems, local minima are solutions that are the best within some region, but not the best possible in the whole parameter space. Local minima occur because of the nonlinearities in the objective function, or because of the complex interactions between the objective function and the constraints. Sequential quadratic programming, conjugate gradient descent, and most other optimization techniques mentioned in this thesis are "local techniques." That is, they are only capable of finding the best solution that lies in some local neighborhood near the starting point. This limitation means that if our initial guess is quite different from the best solution, we may be unable to find it. For example, we believe that one of the major reasons for the failure of our spacetime algorithm to solve the biped problems is that our initial guesses were poor.

These poor initial motions, combined with a difficult optimization space with many local minima, resulted our local methods failing to generate reasonable motion.

There are two potential solutions to the problem of local minima. The first and most obvious of these solutions is to craft better initial guesses. Because we are interested in automatic, or semi-automatic, techniques for modeling and animation, this requirement can more accurately be phrased as "to automatically generate better initial guesses." The second solution is to use a different approach to optimization that avoids, or at least helps to fix, the problem of local minima.

## 6.1 Better Initial Guesses

In order to successfully use nonlinear optimization for modeling and animation tasks, we require a reliable way to generate "good" initial guesses. In our investigations, we observed that a good initial guess is not necessarily visually similar the final answer. Instead, we found that each problem possessed some trait, not always obvious, of the initial guesses that was necessary for reliable convergence.

For some problems, this "key trait" was fairly obvious. In the case of generating a leaping motion of the Luxo lamp we found that if the initial motion had the character leaving the ground by even the smallest amount, the problem would eventually successfully converge. Similarly, initial guesses where the Luxo's foot never left the ground almost always failed to yield a good answer. However, for other problems the key trait for constructing the initial guesses was only identified after experimentation. In the case of optimizing truss structures, for example, we found that a regular initial guess (i.e. one with rectilinearly arranged joints) proved to be important to insure convergence to a good solution, even if the initial shape was very different from the eventual optimal answer.

Unfortunately, generalizing these observations for use in new problems is difficult. For a new modeling or animation task, it may be hard to identity the crucial aspect of an initial guess needed for fast convergence without extensive experimentation. If we are unable to identify the salient features of an initial guess which contribute to its success or failure, automatically or semi-automatically generating good initial guesses becomes much more difficult.

Instead of resorting to exploring the problem structure through trial and error, which can be extremely time consuming even for small problems, a possible alternative is to reformulate the optimization problem so that the variables are easier for the user to manipulate. Such a formulation, if possible, not only allows users to craft initial guesses by hand, but also makes it easier to "tune" unsuccessful or partially successful answers.

As a negative example, our reformulation of spacetime optimization has the considerable disadvantage that the optimization variables are the parameters of force and torque curves over time,

which were very non-intuitive to manipulate. When an optimization run ended with, for example, an infeasible but roughly correct answer, it was often unclear how we could tweak the variables in order to fix the motion or even merely move it closer to the desired answer so further iterations of the optimizer could polish it. In contrast, the standard formulation of spacetime optimization (using Newtonian constraints) allows the user to directly manipulate the spatial trajectories of the character. This difference in formulation allows approaches such as Liu and Popović's [74] and Fang and Pollard's [31] to begin the process of optimization with much better initial motions, and consequently converge more quickly and more reliably.

## 6.2 Global Optimization

However, there are cases when these techniques for generating better initial guesses or avoiding local minima are not sufficient. Not every optimization problem can be easily reformulated so the variables are easier for the user to understand. Nor can we always develop the intuition about a problem needed to write code that could automatically generate good initial guesses for our optimization problems. Furthermore, in some cases we may not wish to bias the final answer with our guess at its form. In the case of our gait synthesis and truss structure generation projects, one of our goals was to be able to generate good, physically realistic motion and structures from "low-knowledge" initial guesses, so that we could find novel gaits or models. In these cases, when the techniques discussed above are not sufficient or suitable, our best strategy is to examine optimization algorithms which are less sensitive to poor initial guesses, or are more adept at avoiding local minima

As discussed previously, local methods such as sequential quadratic programming are only capable of finding local minima. For certain problems, the use of purely local optimization techniques does not cause any difficulties. An example of such a problem is our truss optimization work. In this case, our semi-automatically generated initial structures almost always converged quickly to what appeared to be global minima. On the occasions when the optimizer did become "stuck" in an undesirable local minima, merely applying successive iterations of SQP (sometimes after tweaking the solution by hand) was sufficient to leave the region and find better answers. Although local minima are sometimes "good enough," more frequently they are unsatisfactory and a source of concern. If a problem, such as our gait synthesis task, has many unacceptable local minima then the selection of an initial guess becomes more critical and the successful exploration of the parameter space that much more difficult.

Global techniques, such as those discussed in Sections 2.2 and 3.9, would appear to be the solution to this problem. However, methods for global optimization are not as well developed as local optimization methods. As we saw with our experiments with using simulated annealing to synthesize motion for the Luxo lamp in Section 3.9.1, global techniques can be slow, and generally

perform poorly on problems with many variables [100].

Although these trials did not yield promising results, it is possible that another, more sophisticated global technique could fare better. One possibility for future investigation is a hybrid of global and local techniques. A global technique, such as simulated annealing, genetic programming or multiple-restart optimization, could be used to quickly search the optimization space and identify one or more promising initial guesses. The global optimizer would then pass these initial guesses to the local optimizer, which would polish or refine the solution. This hybrid technique could potentially combine the advantages of both global and local optimization.

## 6.3 Final Lessons

Our exploration of the three research problems described in this thesis was motivated by a desire to apply a powerful tool, nonlinear optimization, to the field of computer graphics. In the process of investigating these problems, we discovered with no great surprise that optimization is not appropriate for every modeling and animation task.

The most obvious limitation of optimization as a tool is that not every problem can be expressed as a set of constraints and a function to be minimized. Many natural objects and phenomena — the motions of fire and water, and the shape of trees and animals — are not dominated by a single minimizing or maximizing principle. Furthermore, even for objects that are strongly influenced an underlying optimization process, its effect may be overshadowed by other, less mathematically tractable factors. For example, many buildings are structurally supported by trusses. However, the aesthetic demands of architecture literally and figuratively mask this fundamental dependency, making the goal of "generating realistic buildings with optimization" next to impossible.

However, there are other, less fundamental, limitations to the application of optimization in computer graphics. As we have discussed, problems with many variables are generally unsuitable for optimization-based solution. Nonlinear problems of high dimension will have many local minima, which will cause local optimization methods to perform poorly and cause trouble for most global techniques. Similarly, problems with extremely nonlinear objective functions, or many constraint functions can cause difficulties for optimization algorithms.

Despite these limitations, however, optimization is inherently a very flexible technique. Given a powerful, general algorithm such as sequential quadratic programming or simulated annealing, we can potentially solve a large number of diverse problems with very little change to our code-base. Furthermore, intuitions developed while solving one set of problems can sometimes be transferred to others, speeding the time to solution significantly. Nonlinear optimization, despite the caveats we have discovered and reported here, remains a valuable tool that the field of computer graphics should be aware of and seek to use whenever appropriate.

# Bibliography


[1] J. Abadie. *Nonlinear and Integer Programming*, chapter Application of the GRG algorithm to optimal control problems, pages 191–211. North-Holland Pub. Co., 1972.

[2] J. Abadie and J. Carpentier. *Optimization*, chapter Generalization of the Wolfe reducded gradient method to the case of nonlinear constraints, pages 37–47. Academic Press, 1969.

[3] R. McN. Alexander. Optimum walking techniques for quadrupeds and bipeds. *The Journal of Zoology*, 192:97–117, 1980.

[4] R. McN. Alexander. The gaits of bipedal and quadrupedal animals. *The International Journal of Robotics Research*, 3(2):49–59, 1984.

[5] R. McN. Alexander. A minimum energy cost hypothesis for human arm trajectories. *Biological Cybernetics*, 76:97–105, 1997.

[6] Frank Anderson and Marcus Pandy. A dynamic optimization solution for vertical jumping in three dimensions. *Computer Methods in Biomechanics and Biomedical Engineering*, 2:201–231, 1999.

[7] Brett Averick, Jorge Moré, Christian Bischof, Alan Carle, and Andreas Griewank. Computing large sparse jacobian matrices using automatic differentiation. *SIAM Journal on Scientific Computing*, 15(2):285–294, 1994.

[8] David Baraff. Coping with friction for non-penetrating rigid body simulation. In *SIGGRAPH '91 Proceedings*, volume 25, pages 31–40, July 1991.

[9] David Baraff. Non-penetrating rigid body simulation. In *State of the Art Reports of EUROGRAPHICS 1993, Eurographics Technical Report Series*, 1993.

[10] J. Baumgarte. Stabilization of constraints and integrals of motion in dynamical systems. *Computational Methods in Applied Mechnical Engineering*, 1:1–16, 1972.

[11] Omar Ghattas Beichang He and James F. Antaki. Computational strategies for shape optimization of time-dependant navier-stokes flows. Technical Report CMU-CML-97-102, Carnegie Mellon University, June 1997.

[12] R. Bellman. *Adaptive Control Processes: A Guided Tour*. Princeton University Press, 1961.

[13] Christian Bischof, Alan Carle, Andreas Griewank, and Paul Hovland. Adifor: Generating derivative codes from fortran programs. *Scientific Programming*, 1(1):11–29, 1992.

[14] Christian Bischof, Alan Carle, Peyvand Khademi, and Andrew Mauer. The adifor 2.0 system for the automatic differentiation of fortran 77 program. Technical Report ANL/MCS-P481-1194, Argonne National Laboratories, 1994.

[15] Christian Bischof, Larry Green, Kitty Haigler, and Tim Knauff. Parallel calculation of sensitivity derivatives for aircraft design using automatic differentiation. In *Proceedings of the 5th AIAA/NASA/USAF/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, pages 73–84. American Institute of Aeronautics and Astronautics,, 1994.

[16] Christian Bischof, Gordon Pusch, and Ralf Knoesel. Sensitivity analysis of the mm5 weather model using automatic differentiation. *Computers in Physics*, pages 605–612, 1996.

[17] Christian H. Bischof, Ali Bouaricha, Peyvand M. Khademi, and Jorge J. Moré. Computing gradients in large-scale optimization using automatic differentiation. *INFORMS Journal on Computing*, 9(2):185–194, 1997.

[18] Christian H. Bischof and Alan Carle. Users' experience with ADIFOR 2.0. In Martin Berz, Christian Bischof, George Corliss, and Andreas Griewank, editors, *Computational Differentiation: Techniques, Applications, and Tools*, pages 385–392. SIAM, Philadelphia, Penn., 1996.

[19] Christian H. Bischof, George F. Corliss, Larry Green, Andreas Griewank, Ken Haigler, and Perry Newman. Automatic differentiation of advanced CFD codes for multidisciplinary design. *Journal on Computing Systems in Engineering*, 3:625–638, 1992.

[20] K. Brakke. The surface evolver. *Experimental Math*, 1:141–165, 1992.

[21] Robert Bridson, Ronald P. Fedkiw, and John Anderson. Robust treatment of collisions, contact, and friction for cloth animation. *ACM Transactions on Graphics*, 21(3):594–603, July 2002.

[22] Armin Bruderlin and Thomas W. Calvert. Goal-directed, dynamic animation of human walking. In *Computer Graphics (Proceedings of SIGGRAPH 89)*, volume 23, pages 233–242, July 1989.

[23] Armin Bruderlin and Lance Williams. Motion signal processing. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 97–104, August 1995.

[24] Alan Carle, Lawrence Green, Christian Bischof, and Perry Newman. Applications of automatic differentiation in cfd. In *Proceedings of the 25th AIAA Fluid Dynamics Conference*. AIAA, 1994.

[25] C. Chapman, K. Saitou, and M.J. Jakiela. Genetic algorithms as an approach to configuration and topology design. In *Advances in Design Automation*, volume 65, pages 485–498. ASME, September 1993.

[26] Michael Cohen. Interactive spacetime control for animation. In *SIGGRAPH '92 Proceedings*, volume 26, pages 293–302, July 1992.

[27] John J. Craig. *Introduction to Robotics: Mechanics and Control*, chapter Inverse Manipulator Kinematics, pages 113–145. Addison-Wesley Publishing Company, Inc., 1989.

[28] B. Delaney. On the trail of the shadow woman: the mystery of motion capture. *IEEE Computer Graphics and Applications*, 18(5):14–19, Sep/Oct 1998.

[29] Roger N. Dent. *Principles of Pneumatic Architecture*. Halsted Press, 1972.

[30] Roman Ďurikovič. Animation of soap bubble dynamics, cluster formation and collision. *Computer Graphics Forum*, 20(3):67–75, 2001.

[31] Anthony Fang and Nancy Pollard. Efficient synthesis of physically valid human motion. *ACM Transactions on Graphics*, 22, 2003.

[32] Ronald Fedkiw, Jos Stam, and Henrik Wann Jensen. Visual simulation of smoke. In *Proceedings of ACM SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 15–22, August 2001.

[33] J. Foley, A. van Dam, S. Feiner, and J. Hughes. *Computer Graphics: Principles and Practice*. Addison-Wesley Publishing Co., 2nd edition, 1990.

[34] Nick Foster and Demitri Metaxas. Realistic animation of liquids. In *Graphics Interface '96*, pages 204–212, May 1996.

[35] P.P. Gambarian. *How mammals run: anatomical adaptations*. Wiley, 1974.

[36] O. Ghattas and J. Bark. Optimal control of two- and three-dimensional navier-stokes flows. *Journal of Computational Physicsd*, 1997.

[37] P.E. Gill, W. Murray, and M.A. Saunders. Snopt: An sqp algorithm for large-scale constrained optimization. Technical Report Numerical Analysis Report 97-2, Department of Mathematics, University of California, San Diego, 199786.

[38] Norman Heglund Giovanni Cavagna and C. Richard Taylor. Walking, running and galloping: Mechanical similarities between different animals. In T.J. Pedley, editor, *Scale Effects in Animal Locomotion*, pages 111–127. Academic Press, 1977.

[39] Andrew Glassner. Andrew Glassner's notebook: Soap bubbles: Part 1. *IEEE Computer Graphics and Applications*, 20(5):76–84, September - October 2000.

[40] Andrew Glassner. Andrew Glassner's notebook: Soap bubbles: part 2. *IEEE Computer Graphics and Applications*, 20(6):99–109, November - December 2000.

[41] Michael Gleicher. Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 33–42, 1998.

[42] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[43] H. Goldstein. *Classical Mechanics*. Addison-Wesley, 1983.

[44] Steven J. Gortler and Michael F. Cohen. Hierarchical and variational geometric modeling with wavelets. In *1995 Symposium on Interactive 3D Graphics*, pages 35–42, April 1995.

[45] A. Griewank. The chain rule revisited in scientific computing. *SIAM News*, 24(4):8–24, 1991.

[46] Andreas Griewank. *Mathematical Programming: Recent Developments and Applications*, chapter On Automatic Differentiation, pages 83–108. Kluwer Academic Publishers, 1989.

[47] Andreas Griewank. Some bounds on the complexity of gradients, jacobians, and hessians. Technical Report MCS-P355-0393, Argonne National Laboratory, 9700 S. Cass Ave., Argonne, IL 60439, 1993.

[48] Radek Grzeszczuk and Demetri Terzopoulos. Automated learning of muscle-actuated locomotion through control abstraction. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 63–70, 1995.

[49] R. T. Haftka and R. V. Grandhi. Structural shape optimization—a survey. *Computer Methods in Applied Mechanics and Engineering*, 57:91–106, 1986.

[50] J. Harriss. *The Tallest Tower – Eiffel and the Belle Epoque*. Houghton Mifflin, 1975.

[51] Joel Hass, Michael Hutchings, and Roger Schlafly. The double bubble conjecture. *Electronic Results Announcements of the American Mathematical Society*, 1(3):98–102, 1995.

[52] E.M. Hatton. *The Tent Book*. Mifflin, 1979.

[53] W.S Hemp. *Optimum Structures*. Clarendon, 1973.

[54] J Heyman. Design of beams and frames for minimum material consumption. *Quarterly of Applied Mathematics*, 8:373–381, 1956.

[55] R.C. Hibbeler. *Structural Analysis*. Prentice Hall, fourth edition, 1998.

[56] Jessica K. Hodgins and Nancy S. Pollard. Adapting simulated behaviors for new characters. In *Proceedings of SIGGRAPH 97*, Computer Graphics Proceedings, Annual Conference Series, pages 153–162, August 1997.

[57] R. Horst and P.M. Pardalos, editors. *Handbook of Global Optimization*. Kluwer, 1995.

[58] Paul Hovland, Bijan Mohammadi, and Christian Bischof. Automatic differentiation of navier-stokes computations. Technical Report ANL/MCS-P687-0997, Argonne National Laboratory, 1997.

[59] Isabelle Icart and Didier Arquès. An approach to geometrical and optical simulation of soap froth. *Computers and Graphics*, 23(3):405–418, June 1999.

[60] Lester Ingber. Adaptive simulated annealing. ftp.alumni.caltech.edu/pub/ingber/ASA.tar.zip, 1993–1998.

[61] Lester Ingber. Adaptive simulated annealing (asa): Lessons learned. *Control and Cybernetics*, 25(1):33–54, 1996.

[62] Cyril Isenberg. *The Science of soap films and soap bubbles*. Dover, 1992.

[63] Farish Jenkins. *Primate Locomotion*. Academic Press, 1974.

[64] David Brogan Jessica Hodgins, Wayne Wooten and James O'Brien. Animating human athletics. In *SIGGRAPH '95 Proceedings*, volume 29, 1995.

[65] Michiel van de Panne Joseph Laszlo and Eugene Fiume. Limit cycles control and its application to the animation of balancing and walking. In *SIGGRAPH '96 Proceedings*, volume 30, pages 155–163, 1996.

[66] T. R. Kane and D. A. Levinson. *Dynamics: Theory and Application*. Mcgraw-Hill, 1985.

[67] Craig S. Kaplan and David H. Salesin. Escherization. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 499–510, 2000.

[68] N. Kapouleas. Compact constant mean curvature surfaces in euclidean three-space. *Journal of Differential Geometry*, 33:683–715, 1991.

[69] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimisation by simulated annealing. *Science*, 220:671–680, 1983.

[70] U. Kirsch. Optimal topologies of structures. *Applied Mechanics Reviews*, 42(8):223–238, 1989.

[71] M. Koiso. Symmetry of hypersurfaces of constant mean curvature with symmetric boundary. *Mathematische Zeitschrift*, 191:567–574, 1986.

[72] Hendrik Kück, Christian Vogelgsang, and Günter Greiner. Simulation and Rendering of Liquid Foams. In *Proceedings of Graphics Interface*, pages 81–88, May 2002.

[73] Jehee Lee and Sung Yong Shin. A hierarchical approach to interactive motion editing for human-like figures. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 39–48, August 1999.

[74] C. Karen Liu and Zoran Popović. Synthesis of complex dynamic character motion from simple animations. *ACM Transactions on Graphics*, 21(3):408–416, July 2002.

[75] Zicheng Liu, Steven Gortler, and Michael Cohen. Hierarchical spacetime control. In *SIGGRAPH '94 Proceedings*, volume 28, pages 35–42, July 1994.

[76] Janzen Lo and Dimitris Metaxas. Recursive dynamics and optimal control techniques for human motion planning. Pre-publication copy, 1999.

[77] R. López and S. Montiel. Constant mean curvature discs with bounded area. In *Proceedings of the American Mathematical Society*, volume 123, pages 1555–1558, 1995.

[78] E. Fiume M. van de Panne. Sensor-actuator networks. In *SIGGRAPH '93 Proceedings*, volume 27, pages 335–342, 1993.

[79] E. Fiume M. van de Panne, R. Kim. Virtual wind-up toys for animation. In *Proceedings of Graphics Interface*, pages 208–215, 1994.

[80] Z. Vranesic M. van de Panne, E. Fiume. Reusable motion synthesis using state-space controllers. In *SIGGRAPH '90 Proceedings*, volume 24, pages 225–234, 1990.

[81] C. MacCallum and R. Hanna. Deflect: A computer aided learning package for teaching structural design. In *Proceedings of Education in Computer Aided Architectural Design in Europe*, 1997.

[82] Mark Meyer, Mathieu Desbrun, Peter Schröder, and Alan Barr. Discrete differential geometry operators for triangulated 2-manifolds. In *Proceedings of the International Workshop on Visualization and Mathematics 2002 (Vismath 2002)*, volume 4, May 2002.

[83] A.G.M. Michell. The limits of economy of material in frame structures. *Philosophical Magazine*, 8:589–597, 1904.

[84] Kaisa Miettinen. *Nonlinear Multiobjective Optimization*. Kluwer Academic Publishers, 1999.

[85] B. A. Murtagh and M. A. Saunders. Minos 5.4 user's guide. Technical Report OL 83-20R, Systems Optimization Laboratory, Stanford University, 1983. revised 1995.

[86] J.T. Ngo and J. Marks. Spacetime constraints revisited. In *SIGGRAPH '93 Proceedings*, volume 27, pages 343–350, 1993.

[87] Duc Quang Nguyen, Ronald P. Fedkiw, and Henrik Wann Jensen. Physically based modeling and animation of fire. *ACM Transactions on Graphics*, 21(3):721–728, July 2002.

[88] Bernd Oberknapp and Konrad Polthier. *Visualization and Mathematics*, chapter An algorithm for Disrete Constant Mean Curvature Surfaces. Springer Verlag, 1997.

[89] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 137–146, August 1999.

[90] C.E. Orozco and O. Ghattas. Massively parallel aerodynamic shape optimization. *Computing Systems in Engineering*, 1(4):311–320, 1992.

[91] C.E. Orozco and O. Ghattas. A reduced sand method for optimzal design of nonlinear structures. *International Journal for Numerical Methods in Engineering*, 1(4):311–320, 1992.

[92] C.E. Orozco and O. Ghattas. Infeasible path optimal design methods, with application to aerodynamic shape optimization. *AIAA Journal*, 34:217–224, 1996.

[93] Frei Otto, editor. *Tensile Structures, Volume I: Pneumatic Structures*. MIT Press, 1967.

[94] D. Pai. Programming anthropoid walking: Control and simulation. Technical Report TR 90-1178, Cornell University, 1990.

[95] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *Proceedings of SIGGRAPH 2001*, Computer Graphics Proceedings, Annual Conference Series, pages 301–308, August 2001.

[96] P. Pederson. Topology optimization of three dimensional trusses. In *Topology Designs of Structures, NATO ASI Series – NATO Advanced Research Workshop*, pages 19–30. Kluwer Academic Publishers, June 1992.

[97] Robert Phelan, Denis Weaire, and Kenneth Brakke. Computation of equilibrium foam structures using the surface evolver. *Experimental Mathematics*, 4:181–192, 1995.

[98] M. Piccolotto and O. Rio. Design education with computers. In *Proceedings of ACADIA 95: Computing in Design*, pages 285–299, 1995.

[99] Ulrich Pinkall and Konrad Polthier. Computing discrete minimal surfaces and their conjugates. *Experimental Mathematics*, 2(1):15–36, 1993.

[100] Janos D. Pinter. *Global Optimization in Action*. Kluwer, 1995.

[101] E.P. Popov. *Engineering Mechanics of Solids*. Prentice Hall, 1998.

[102] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew P. Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of ACM SIGGRAPH 2000*, Computer Graphics Proceedings, Annual Conference Series, pages 209–218, July 2000.

[103] Zoran Popović. *Motion Transformation by Physically Based Spactime Optimization*. PhD thesis, Carnegie Mellon University, 1999.

[104] Zoran Popović and Andrew Witkin. Physically based motion transformation. In *SIGGRAPH '99 Proceedings*, volume 33, pages 11–20, August 1999.

[105] W. Press, S. Teukolsky, W. Vetterling, and B. Flannery. *Numerical Recipes in C*. Cambridge University Press, 2nd edition, 1992.

[106] P. Prusinkiewicz. Modeling and visualization of biological structures. In *Proceedings of Graphics Interface*, pages 128–137, 1993.

[107] Przemyslaw Prusinkiewicz and Lars Mndermann. 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. ACM Press, 2001.

[108] Marc H. Raibert and Jessica K. Hodgins. Animation of dynamic legged locomotion. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 349–358. ACM Press, 1991.

[109] M.H. Raibert. *Legged Robots That Balance*. MIT Press, 1986.

[110] G. Reddy and J. Cagan. An improved shape annealing algorithm for truss topology. *ASME Journal of Mechanical Design*, 117(2A):315–321, 1995.

[111] U. Ringertz. Optimal design of nonlinear shell structures. Technical Report FFA-TN-91-18, The Aeronautical Research Institute of Sweden, 1991.

[112] Charles Rose, Brian Guenter, Bobby Bodenheimer, and Michael Cohen. Efficient generation of motion transitions using spacetime constraints. In *SIGGRAPH '96 Proceedings*, volume 30, pages 147–154, 1996.

[113] Wolfgang Scheueller. *The Design of Building Structures*, chapter 8—9, pages 584—812. Prentice-Hall, 1996.

[114] V.H. Schulz and H.G. Bock. Partially reduced sqp methods for large-scale nonlinear optimization problems. In *Proceedings of the Second World Congress of Nonlinear Analysis*, pages 1–12, 1997.

[115] Jonathan Richard Shewchuk. An introduction to the conjugate gradient method without the agonizing pain. http://www.cs.berkeley.edu/ jrs/, August 1994.

[116] Karl Sims. Evolving 3d morphology and behavior by competition. In *Artificial Life IV*, pages 28–39, 1994.

[117] Karl Sims. Evolving virtual creatures. In *SIGGRAPH '94 Proceedings*, volume 28, pages 15–22, 1994.

[118] J. Spanier and K.B. Oldham. *An Atlas of Functions*. Hemisphere, 1987.

[119] W.R. Spillers. *Iterative Structural Design*. North Holland Publishing Co., 1975.

[120] Vladimir Borisovich Sukhanov. *General system of symmetrical locomotion of terrestrial vertebrates and some features of movement of lower tetrapods*. Amerind Publishing Co., 1968.

[121] John M. Sullivan and Fred Almgren. Visualization of soap bubbles geometries. *Leonardo*, 23(3–4):267–271, 1992.

[122] Seyoon Tak, Oh young Song, and Hyeong-Seok Ko. Motion balance filtering. *Computer Graphics Forum*, 19(3):437–446, August 2000.

[123] W. Thompson and Lord Kelvin. On the division of space with minimum partitional area. *Phil. Mag. Lett.*, 69:503, 1887.

[124] B.H.V Topping. Shape optimization of skeletal structures: A review. *Journal of Structural Engineering*, 109:1933–1951, 1983.

[125] Michiel van de Panne. Paramterized gait synthesis. *IEEE Computer Graphics and Applications*, pages 40–49, March 1996.

[126] Michiel van de Panne. From footprints to animation. *Computer Graphics Forum*, 16(4):211–223, 1997.

[127] G.N. Vanderplaats and F. Moses. Automated optimal geometry design of structures. *Journal of the Structural Division of the American Society of Civil Engineers*, 98(ST3), March 1977.

[128] B. W. Wah and Y. X. Chen. Optimal anytime constrained simulated annealing for constrained global optimization. In *Proc. Principles and Practice of Constraint Programming*, pages 425–439. Springer- Verlag, Sept. 2000.

[129] T. Wang. *Global Optimization of Constrained Nonlinear Programming*. PhD thesis, Dept. of Computer Science, Univ. of Illinois, December 2000.

[130] William Welch and Andrew Witkin. Variational surface modeling. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, volume 26, pages 157–166, July 1992.

[131] D.A. Winter. *Biomechaincs and Motor Control of Human Movement*. John Wiley and Sons, 1990.

[132] Andrew Witkin and Michael Kass. Spacetime constraints. In *SIGGRAPH '88 Proceedings*, volume 22, pages 159–168, August 1988.

[133] Andrew P. Witkin and Zoran Popović. Motion warping. In *Proceedings of SIGGRAPH 95*, Computer Graphics Proceedings, Annual Conference Series, pages 105–108, August 1995.

[134] Patrick Witting. Computational fluid dynamics in a traditional animation environment. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 129–136, August 1999.

[135] M Wright and P. Gill. *Practical Optimization*. Academic Press, 1981.

[136] M Wright and P. Gill. *Practical Optimization*. Academic Press, 1981.

[137] G.T. Yamaguchi. *Biomechanics and Movement Organization*, chapter Multiple Muscle Systems. Springer-Verlag, 1990.

[138] Victor B. Zordan and Jessica K. Hodgins. Motion capture-driven simulations that hit and react. In *ACM SIGGRAPH Symposium on Computer Animation*, pages 89–96, July 2002.