Constraint Methods for Neural Networks and Computer Graphics

Thesis by

John Platt

In Partial Fulfillment of the Requirements

for the Degree of

Doctor of Philosophy

California Institute of Technology

Pasadena, California

1989

(Submitted April 26, 1989)

Acknowledgements

I would like to thank the advisors that I had throughout my graduate career: Al Barr, Carver Mead, and John Hopfield. They taught me a lot about what science is and how to do it.

Much of this thesis has appeared as published papers. I would like to thank my co-authors for the stimulating fun they provided me: Al Barr, John Hopfield, Demetri Terzopoulos, and Kurt Fleischer. I would especially like to thank Kurt Fleischer for providing the pictures that illustrated Chapter 6.

Many people around Caltech participated in useful discussions and wrote wonderful software: Ronen Barzel, Dave Gillespie, John Lazzaro, and John Snyder. Thanks for the help.

This thesis was supported by hardware donations from Hewlett-Packard and Symbolics, Inc. The HP 9000 and the Symbolics 3650 are both good machines for software development.

This thesis was further supported by a fellowship from AT&T.

I would finally like to acknowledge the tremendous work of my parents, Felicia and Victor Platt. They made it all possible.

Abstract

Both computer graphics and neural networks are related, in that they model natural phenomena. Physically-based models are used by computer graphics researchers to create realistic, natural animation, and neural models are used by neural network researchers to create new algorithms or new circuits. To exploit successfully these graphical and neural models, engineers want models that fulfill designer-specified goals. These goals are converted into mathematical constraints.

This thesis presents constraint methods for computer graphics and neural networks. The mathematical constraint methods modify the differential equations that govern the neural or physically-based models. The constraint methods gradually enforce the constraints exactly. This thesis also describes applications of constrained models to real problems.

The first half of this thesis discusses constrained neural networks. The desired models and goals are often converted into constrained optimization problems. These optimization problems are solved using first-order differential equations. There are a series of constraint methods which are applicable to optimization using differential equations: the *Penalty Method* adds extra terms to the optimization function which penalize violations of constraints, the *Differential Multiplier Method* adds subsidiary differential equations which estimate Lagrange multipliers to fulfill the constraints gradually and exactly, *Rate-Controlled Constraints* compute extra terms for the differential equation that force the system to fulfill the constraints exponentially. The applications of constrained neural networks include the creation of constrained circuits, error-correcting codes, symmetric edge detection for computer vision, and heuristics for the traveling salesman problem.

The second half of this thesis discusses constrained computer graphics models. In computer graphics, the desired models and goals become constrained mechanical systems, which are typically simulated with second-order differential equations. The *Penalty Method* adds springs to the mechanical system to penalize violations of the constraints. *Rate-Controlled Constraints* add forces and impulses to the mechanical system to fulfill the constraints with critically damped motion. Constrained computer graphics models can be used to make deformable physically-based models follow the directives of a animator.

Table of Contents

Acknowledgements iii

Abstract v

Table of Contents vii

List of Figures xiii

Glossary xvii

Introduction

Chapter 1: Introduction I–1

1.1 Overview of Thesis I–1

1.2 Contributions of Thesis I–2

1.3 Previous Work in Neural Networks Modeling I–2

1.4 For the Neural Network Researcher I–3

1.5 Previous Work in Computer Graphics Modeling I–4

1.6 For the Computer Graphics Implementor I–4

1.7 For the Computer Scientist and Electrical Engineer I–5

1.8 For the Mechanical Engineer I–6

1.9 For the Computer Vision Researcher I–7

Constrained Neural Networks: Theory

Chapter 2: Constrained Optimization Methods II–1

2.1 Introduction: Optimization for Circuits II–1

2.2 Gradient Descent II–2

2.3 Quadratic Forms II–4

2.4 Constrained Optimization II–5

2.5 The Penalty Method II–6

2.6 Exact Penalty Method II–8

2.7 Lagrange Multipliers II–9

2.8 Differential Multiplier Method II–11

2.9 Rate-Controlled Constraints II–13

2.10 Conclusions II–15

Constrained Neural Networks: Applications

Chapter 3: Constrained Circuits III–1

3.1 Introduction: Analog Circuits for Constrained Optimization III–1

3.2 Quadratic Programming Circuit III–1

3.3 Constrained Flip-flop III–3

3.4 Conclusions III–5

Chapter 4: Analog Decoding Using Neural Networks IV–1

4.1 Introduction: Error Correcting Codes IV–1

4.2 Error-Correcting Codes for Neural Networks IV–4

4.2.1 1 in N Code IV–4

4.2.2 Permutation Matrix Code IV–5

4.3 Two Neural Networks that Perform Decoding IV–6

4.3.1 Penalty Method IV–6

4.3.2 Differential Multiplier Method IV–7

4.4 Results IV–7

4.5 Conclusions IV–9

Chapter 5: Constrained Optimizing Splines V–1

5.1 Introduction: What are Constrained Optimizing Splines? V–1

5.2 Constrained Snakes for Computer Vision V–3

5.3 The Traveling Salesman Problem V–4

5.4 Conclusions V–6

Constrained Computer Graphics: Theory

Chapter 6: Deformable Physically-Based Models VI–1

6.1 Introduction: Dynamic vs. Kinematic Models VI–1

6.1.1 Outline VI–2

6.1.2 Notation VI–2

6.1.3 Coordinate Systems VI–2

6.2 Dynamics of Deformable Models VI–3

6.2.1 The Dynamics of a Spring-Mass System VI–4

6.2.2 The Dynamics of an Elastic Model VI–4

6.3 Elastic Materials VI–5

6.3.1 Analysis of Deformation VI–5

6.3.2 Elastic Force for an Isotropic Solid VI–8

6.4 Applied Forces on Deformable Models VI–9

6.5 Approximations of Deformable Models VI–10

6.6 Simulation Tests of Deformable Models VI–15

6.7 Conclusion VI–15

Chapter 7: Constraint Methods for Physical Systems VII–1

7.1 Introduction: Teleological Modeling VII–1

7.1.1 Previous Work VII–3

7.2 The Penalty Method VII–3

7.3 Lagrangian Constraints VII–6

7.3.1 Rate-Controlled Constraints VII–10

7.3.2 Simplifications of Rate-Controlled Constraints VII–11

7.3.3 Inequality Constraints and Lagrange Multipliers VII–12

7.4 Simulation Examples VII–14

7.5 Conclusions VII–15

Constrained Computer Graphics: Applications

Chapter 8: Constraints for the Animation of Flexible Models VIII–1

8.1 Introduction: Constraints are Useful for Creating Animation VIII–1

8.2 Path-following Constraints VIII–2

8.3 Planar Attraction/Repulsion Constraints VIII–3

8.4 Repulsion Constraints from a Implicit Algebraic Model VIII–4

8.5 Constraints that Release VIII–4

8.6 Moldable Materials VIII–5

8.7 Incompressibility Constraints VIII–6

8.8 Results VIII–7

8.9 Conclusions VIII–8

Conclusions and Appendices

Chapter 9: Conclusions IX–1

9.1 Summary IX–1

9.2 Future Work IX–2

9.3 Conclusions IX–3

Appendix A: Numerical Techniques A–1

A.1 Finite Elements for Elasticity A–1

A.2 How to Numerically Solve Differential Equations A–3

A.3 How to Solve Non-Linear Algebraic Equations A–7

A.4 How to Solve Sparse Linear Algebraic Equations A–8

Appendix B: Convergence of the Differential Multiplier Method B–1

Appendix C: Calculus of Variations C–1

Appendix D: An Introduction to Circuit Theory D–1

Appendix E: References E–1

List of Figures

Figure 2.1. How to use optimization theory to create circuits II–2
Figure 2.2. Gradient Descent II–3
Figure 2.3. Quadratic landscape II–4
Figure 2.4. The penalty method makes a trough in state space II–6
Figure 2.5. The exact penalty method II–8
Figure 2.6. At the constrained minimum, $\nabla f = -\lambda \nabla g$ II–9
Figure 2.7. Gradient ascent makes the differential multiplier method stable II–11
Figure 2.8. The state is attracted to the constraint manifold II–12
Figure 2.9. Rate-controlled constraints add an extra synaptic matrix II–14
Figure 3.1. A circuit that implements quadratic programming III–2
Figure 3.2. A flip-flop. U_1 and U_2 are voltages III–4
Figure 3.3. A circuit for constraining a flip-flop III–4
Figure 3.4. Constraint fulfillment for quadratic programming circuit III–6
Figure 3.5. Plot of output of quadratic programming circuit III–6
Figure 3.6. Constraint fulfillment with ringing III–7
Figure 3.7. Constraint fulfillment for a non-linear optimization function III–7
Figure 3.8. Plot of variables for time-dependent non-linear constrained optimization III–8
Figure 4.1. Why error-correcting codes work IV–1
Figure 4.2. Immediate digitization adds errors to the signal IV–2
Figure 4.3. Decoding is like perception IV–3
Figure 4.4. 1 in N code IV–4
Figure 4.5. Permutation matrix code IV–5
Figure 4.6. The decoder finds the nearest permutation matrix IV–8
Figure 4.7. Performance of various neural decoders on the permutation matrix code IV–10
Figure 5.1. A snake is a one-dimensional manifold V–1
Figure 5.2. A symmetry-seeking snake found a finger V–4
Figure 5.3. The snake is attaching to the cities V–5
Figure 6.1. Coordinate systems used in this chapter VI–3
Figure 6.2. Energy of deformation depends on amount of deformation VI–7
Figure 6.3. Diagonal components of the metric tensor regulate lengths between points VI–11
Figure 6.4. Off-diagonal components of the metric tensor regulate angles between points ... VI–12

Figure 6.5. Springs in three faces of a three-dimensional hexahedral elastic solid element ... VI–12
Figure 6.6. A pattern of springs connecting mass points in an elastic surface element VI–13
Figure 6.7. Different discretizations of a spring-mass system VI–13
Figure 6.8. Position of endpoints of two differently discretized models VI–14
Figure 6.9. A simulation of a flag flapping in the wind VI–16
Figure 6.10. The flag later in time VI–16
Figure 6.11. The flag even later in time VI–17
Figure 6.12. A simulation of a carpet falling on a sphere and a cylinder VI–17
Figure 6.13. The carpet changes shape due to external forces VI–17
Figure 6.14. The carpet eventually slips between the other objects VI–18
Figure 6.15. A simulation of jello vibrating on the table VI–18
Figure 6.16. The jello eventually returns to its rest shape: a cube VI–18
Figure 7.1. Levels of representation of an object VII–1
Figure 7.2. A hierarchy of force-based constraint methods VII–2
Figure 7.3. A mass point is attracted by a penalty force to a desired location on a path VII–4
Figure 7.4. A mass point is expelled by a penalty force from the inside of an object VII–5
Figure 7.5. Lagrangian constraints change the forces normal to the constraint manifolds VII–8
Figure 7.6. Lagrangian constraints also create forces that fulfill constraints gradually VII–8
Figure 7.7. Rate-controlled constraints solve a linear system to find Lagrange multipliers .. VII–10
Figure 7.8. $\underline{y}$ and $G(\underline{y})$ for a mass point hitting a plane and sticking briefly VII–13
Figure 7.9. A sphere falling onto a trampoline using the penalty method VII–15
Figure 7.10. The penalty method fails to keep the sphere above the trampoline VII–15
Figure 7.11. Using the penalty method, the sphere eventually falls through the trampoline VII–16
Figure 7.12. A sphere falling onto a trampoline using a rate-controlled constraint VII–16
Figure 7.13. Rate-controlled constraints keep the sphere above the trampoline VII–16
Figure 7.14. The sphere bounces because of the rate-controlled constraint VII–17
Figure 7.15. The sphere eventually comes to rest above the trampoline VII–17
Figure 8.1. The actual path of the mass point gradually matches the desired path. VIII–2
Figure 8.2. Rate-controlled constraints keep mass points on one side of a plane VIII–3
Figure 8.3. A snap constraint VIII–5
Figure 8.4. The rest shape of plastic materials changes after strong deformation. VIII–6
Figure 8.5. Incompressibility preserves the volume of an element VIII–6

Figure 8.6. A compressible cube of jello is picked up by a path-following constraint........ VIII–9
Figure 8.7. The corner of the cube continues to follow a predetermined path............... VIII–9
Figure 8.8. The compressible jello hits the table... VIII–9
Figure 8.9. The jello wriggles on the table.. VIII–10
Figure 8.10. A sphere squashes a seat cushion... VIII–10
Figure 8.11. A cube of incompressible jello above a table................................. VIII–10
Figure 8.12. The jello hits the table.. VIII–11
Figure 8.13. The jello quickly bounces off of the table.................................... VIII–11
Figure 8.14. A lump of moldable incompressible clay sticks to the table................... VIII–11
Figure 8.15. The initial rest shape of the models in figures 8.16 through 8.19............ VIII–12
Figure 8.16. An elastic model is squashed... VIII–12
Figure 8.17. An elastic model returns to its rest shape.................................... VIII–12
Figure 8.18. A moldable model is squashed... VIII–13
Figure 8.19. A moldable model assumes a new rest shape after strong deformation........ VIII–13
Figure 8.20. An elastic solid is falling onto a funnel...................................... VIII–13
Figure 8.21. The solid is held up on the left side of the funnel by static friction........ VIII–14
Figure 8.22. The solid is held up on both sides by static friction.......................... VIII–14
Figure A.1. Euler and Runge-Kutta.. A–3
Figure A.2. The modified midpoint method A–4
Figure A.3. The Bulirsch-Stoer method A–5
Figure A.4. Explicit methods are sometimes unstable.. A–6
Figure A.5. Implicit methods are stable A–6
Figure D.1. Kirchoff's current law... D–1
Figure D.2. A resistor... D–1
Figure D.3. A capacitor.. D–2
Figure D.4. A transconductance amplifier D–2
Figure D.5. An operational amplifier... D–3
Figure D.6. A voltage follower created from an operational amplifier........................ D–3
Figure D.7. An inverter D–4

Glossary

- **Active Flexible Model** — A model whose rest shape changes because of internal forces.
- **AI** — Artificial Intelligence. The simulation of high-level cognitive functions using clever computer programming.
- **Axon** — The part of a neuron which typically sends information to other neurons. See *neuron*.
- **Back-propagation** — A method to cause a feed-forward neural network to learn a function from examples.
- **BDMM** — Basic Differential Multiplier Method. A constrained optimization technique described in chapter 2, which uses a subsidiary differential equation to estimate Lagrange multipliers.
- **Bicubic Patch** — A surface whose position is modeled by a polynomial that is cubic in the two material coordinates.
- **Bistable** — A system is bistable if it has two stable states.
- **Boltzmann Distribution** — An exponential probability distribution. Often arises in physical systems because of statistical mechanics.
- **Bulirsch-Stoer Method** — A method to solve ordinary differential equations, described in Appendix A.
- **Catenary** — The shape a chain attains when it comes to rest if held by the ends and placed in a gravitational field.
- **CCD** — Charged-Coupled Device, used to move analog charges across a chip.
- **Co-Content** — The energy dissipated in resistive elements in a circuit.
- **Coefficient Of Restitution** — The ratio of the normal velocity of a mass point after a collision to that before a collision.
- **Cofactor Matrix** — If M_{ij} is a matrix, then an element of the cofactor matrix C_{ij} is obtained by deleting the ith row and jth column of M, then taking the determinant of the resulting $N-1 \times N-1$ matrix.
- **Collinear** — Two vectors are collinear if they point in the same direction.
- **Compressibility** — A material is compressible if its volume can change because of external forces.
- **Concave** — A concave shape has one or more dents in it.

- **Constraint** — A goal that a given system should fulfill, expressed as an mathematical equation.
- **Convolution** — The convolution of two functions f and g is

$$f * g(s) = \int_{-\infty}^{\infty} f(t)g(t-s)dt.$$

- **Critically Damped Motion** — A motion described by the differential equation

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

where $c^2 = 4km$.

- **DAE** — Differential Algebraic Equation. A system of equations, some of which are differential and others algebraic (without differentials).
- **Damped** — A material is damped if it loses energy because of friction.
- **Dashpot** — A shock absorber.
- **dB** — Decibel. A logarithmic measure. A value x is converted to decibels by

$$x(\text{dB}) = 10\log_{10} x$$

- **Deformable** — A model is deformable if it can change its shape.
- **Dendrite** — A part of a neuron that typically collects information from other neurons. See *neuron*.
- **Digitization** — The process of converting an analog signal to a digital signal.
- **Dirichlet Boundary Condition** — A boundary condition on a partial differential equation which specifies the value of a variable on the boundary of a region as a function of time.
- **DMM** — Differential Multiplier Method. A constrained optimization method described in chapter 2.
- **Dynamic Model** — A model whose motion is described by physics, as opposed to geometry.
- **Dynamical System** — A system of first-order differential equations.
- **Dynamics** — The study of the motion of objects using physics.
- **ECC** — Error-correcting Code. A way of encrypting data so that it can be sent over a noisy channel while minimizing the amount of error.

- **Euclidean Distance** — If there are two vectors $\underline{x}$ and $\underline{y}$, then the Euclidean distance between them is

$$\sum_i (x_i - y_i)^2$$

- **Euler's Method** — A bad method to numerically solve differential equations, described in Appendix A.
- **Excitation** — An input which tends to turn on a neuron.
- **Explicit Method** — An explicit method provides a numerical solution for a differential equation, without converting it to a algebraic equation.
- **Extremization** — The finding of either a maximum, a minimum, or an inflection point of a function.
- **Finite Difference** — The approximation of a derivative with a algebraic expression.
- **Force-based Constraint Method** — A constraint method that adds forces or impulses to a physical system.
- **Flip-Flop** — A bistable circuit consisting of two inverters inhibiting each other.
- **Free-Form Deformation** — The deformation of models using a Jacobian function specified by a designer.
- **FSK** — Frequency-shift keying. The transmission of an alphabet of symbols by transmitting different symbols on different frequencies.
- **Gain** — The slope of a voltage transfer function.
- **Gaussian Distribution** — A probability distribution defined by

$$P(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}.$$

- **Generalized Force** — An entity that contributes to the second time derivative of physical variables. (see generalized mass matrix). Forces and torques are generalized forces.
- **Generalized Mass Matrix** — The matrix $m_i j$ that appears in Newton's law

$$\sum_j m_{ij} x_j = f_i$$

where x_j is a physical variable (e.g., position, rotation), and f_i is a generalized force.
- **Hamming Distance** — The distance between two bit vectors measured by the number of bits different in the two vectors.

- **Hookean** — A material is Hookean if the deformation of the material is a linear function of the forces on the material.
- **Hypercube** — The generalization of a cube to N dimensions.
- **Hyperplanes** — The generalization of a plane to N dimensions.
- **In-Betweening** — A technique where an animator specifies certain frames of an animation and someone or something else fills in the rest of the frames.
- **Incompressible** — A material is incompressible if it preserves its volume under deformation.
- **Inhibition** — An input which tends to shut off a neuron.
- **Implicit Method** — An implicit method provides a numerical solution for a differential equation by converting it to an algebraic equation.
- **Impulsive Force** — An idealized force that is infinite in strength, but takes zero time. The change in the momentum caused by an impulsive force is finite.
- **Isotropic** — A material is isotropic if its response to a external force is invariant in the direction of the external force.
- **Jacobian** — A matrix J_{ij} of derivatives of a vector function $\underline{f}(\underline{x})$:

$$J_{ij} = \frac{\partial f_i}{\partial x_j}$$

- **Kinematics** — The study of positions of objects.
- **Kinetic Energy** — Energy of macroscopic motion.
- **Lagrange Multiplier** — An extra variable, usually called λ, added to solve constrained optimization techniques (see chapter 2). Also, the magnitude of a force required to fulfill a constraint (see chapter 7).
- **Lagrangian** — A functional which a physical system extremizes: the sum of the potential and kinetic energy.
- **Lipschitz Condition** — A limit on the derivative of a function.
- **LU Decomposition** — Lower-Upper Decomposition of a matrix. Used to speed up the solution of a linear system.
- **Lyapunov Function** — A function which never increases as a differential equation evolves. Used to prove the stability of a differential equation.
- **MDMM** — Modified Differential Multiplier Method. A constrained optimization method described in chapter 2, which is a combination of the penalty method and the basic differential

multiplier method.

- **Modeling** — The abstraction of a mathematical model from a physical system.
- **Moldable** — A flexible model is moldable if its rest shape changes under the influence of external forces.
- **Moving Variables** — The variables that move under the influence of differential equations.
- **Multigrid** — A technique of solving partial differential equations by manipulating variables at different spatial scales.
- **Nernst Potential** — A voltage caused by an electrochemical reaction.
- **Neuron** — A cell in the brain that allows the brain to compute. A neuron typically consists of three parts: a set of dendrites, a cell body, and a set of axons.
- **N-flop** — A generalization of a flip-flop to N elements, only one of which is on at a time.
- **Non-Causal** — A system whose output depends on the inputs in the future.
- **NP-complete** — A problem that can be shown to be equivalent in difficulty to the exact solution of the Traveling Salesman Problem: a very hard problem.
- **Operational Amplifier** — A high-gain amplifier that produces a voltage, described in Appendix D.
- **Optimization** — A process of finding a minimum of a function or functional.
- **Overdamped Motion** — A motion described by the differential equation

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

  where $c^2 > 4km$
- **Penalty Method** — A constrained optimization method described in chapter 2, which adds an extra term to the optimization function to penalize violations of constraints with a spring-like force.
- **Phase-Shift-Keying (PSK)** — The transmission of an alphabet of symbols by transmitting different symbols on different phases of a carrier.
- **Physically-Based Model** — A type of computer graphics model that relies on physics to automatically move the model.
- **Poisson Process** — A stochastic process whose events happen randomly, at some average rate.
- **Polyhedron** — A solid whose faces are polygons.
- **Postsynaptic Neuron** — The neuron that receives input from the synapse.

- **Prescribed-Metric Model** — A model whose rest shape is described by a user-specified metric tensor function.
- **Presynaptic Neuron** — The neuron that drives the synapse.
- **Rayleigh Dissipation Function** — A functional whose functional derivative is a frictional force.
- **RCC** — Rate-Controlled Constraint. A constraint method that causes a physical system to satisfy constraints at some rate.
- **Renderer** — A computer program which creates images from models.
- **Regularization** — The conversion of an ill-posed problem into a well-posed problem by using optimization techniques.
- **Shunting Inhibition** — Inhibition of a neuron that works by decreasing the effectiveness of the excitation of that neuron.
- **Simulated Annealing** — A global optimization method that simulates a slowly cooling physical system until a deep minimum is found.
- **Source-Channel Theorem** — Proved by Shannon, it says that given sufficient redundancy, data can be sent down a noisy channel and be recovered noise-free.
- **Statics** — The study of the distribution of forces in an object at equilibrium.
- **Synapse** — A junction between two neurons where information is transferred.
- **tanh** — Hyperbolic tangent.
- **Tensor** — A mathematical quantity which transforms under the tensor transformation rule when expressed in a different coordinate system.
- **Telelogical model** — A teleological model contains user-specified goals that the model tries to fulfill.
- **Torsion** — The twisting of a one-dimensional flexible model.
- **Tridiagonal** — A matrix whose only non-zero elements lie on the main diagonal or on the diagonals one above or one below the main diagonal.
- **Underdamped Motion** — A motion described by the differential equation

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = 0$$

  where $c^2 < 4km$.
- **Variational Derivative** — A derivative of a functional which yields a function.

- **Viscoelastic** — A model whose internal forces depend both on the shape of the model and on the time derivative of the shape of the model.
- **VLSI** — Very Large Scale Integration. A silicon chip that has more than 100,000 transistors.
- **WBGM** — Weighted Bipartite Graph Matching. A problem where N workers must be assigned to N jobs in the most efficient manner.

Introduction

Chapter 1 : Introduction

1.1 Overview of Thesis

Many people use models of natural phenomena to design systems that perform useful tasks. For example, some engineers create circuits that can solve cognitive or perceptual problems by using neural networks, which are approximate models of neural functions. Other designers create computer graphics to communicate or illustrate ideas by creating physically-based models, which describe the shape and motion of physical objects.

Designers who use either computer graphics models or neural network models are frequently frustrated because the models typically do not do what the designer wants. This thesis describes constraint methods that are applicable to models of natural phenomena. These constraint methods cause models to fulfill designer-specified goals.

The first half of this thesis describes how to constrain neural network models. Neural networks typically use first-order differential equations to solve optimization problems. These differential equations describe analog circuits that operate in parallel and can be built in VLSI. A constrained neural network therefore should solve a constrained optimization problem using a system of differential equations. Chapter 2 describes three constraint methods that use differential equations to solve constrained optimization problems. First, the *Penalty Method* adds extra terms to the optimization function in order to penalize violation of the constraints. Second, the *Differential Multiplier Method* adds subsidiary differential equations to the system to estimate Lagrange multipliers. Third, *Rate-Controlled Constraints* add extra terms to the differential equations in order to fulfill the constraints exponentially.

This thesis also discusses applications of constrained neural networks. The algorithms to constrain neural networks are tested by constraining real circuits. A neural network that decodes an error-correcting code is always constrained to produce a code word. A neural network which acts as an active spline is constrained to form a tour for the traveling salesman problem.

The second half of this thesis describes how to constrain physically-based computer graphics models. Physically-based models can either use first-order or second-order differential equations to simulate mechanical systems. If the models use first-order differential equations, then the methods described in the first half of the thesis are applicable. Otherwise, the constraint methods add forces and impulses to the mechanical system in order to make the constraints seem to be fulfilled by invisible hands. This thesis describes three force-based constraint methods that are applicable to mechanical systems. The *Penalty Method* adds springs to the mechanical system in order to penalize violation of the constraints. *Rate-Controlled Constraints* uses inverse dynamics to compute the forces needed to fulfill the constraints with critically-damped motion. Constrained flexible physically-based models can be used to easily create complex, realistic animation.

1.2 Contributions of Thesis

This thesis contributes fundamentally to the study of neural networks and the study of computer graphics. Using the methods in this thesis, neural network and computer graphics researchers can now create better circuits and animations by controlling physically-based and neural models.

For neural networks, I noticed the difficulty in setting the parameters of neural networks that perform optimization. I constructed a neural network that automatically chooses these parameters by using the differential multiplier method [Platt & Barr 87][Arrow, et al.]. I built circuits to test whether the differential multiplier method is appropriate for circuits. I also tested the performance of the differential multiplier method on problems where neural network parameter tuning was difficult: the travelling salesman problem and analog decoding [Platt & Barr 87].

For computer graphics, I noticed that the animation of deformable models was very difficult. In order to look fully realistic, I decided that deformable models should be physically-based [Terzopoulos, et al.]. However, goals needed to be added to physically-based models, so that animators can use the models [Barr88]. Therefore, I extended deformable models so that they would fulfill goals, by adapting constraint stabilization [Platt & Barr 88][Baumgarte]. I also created constraint functions that are appropriate for computer animation.

1.3 Previous Work in Neural Networks Modeling

In the literature of neural network modeling, researchers have approximated the behavior of a set of neurons by a set of differential equations, with one variable per neuron. The differential equations are derived by assuming that the dendritic arbor has no computational structure and that the output of the neuron can be represented by the average frequency of the presynaptic action potentials [Wilson & Cowan] [Grossberg].

Recently, the stability of these differential equations has been shown by the use of a Lyapunov function: a function which always decreases as the differential equation evolves [Cohen & Grossberg][Hopfield]. Lyapunov functions are predictive: given a system of differential equations with a Lyapunov function, the system always approaches a limit set [LaSalle].

Frequently, researchers want to construct circuits, instead of predict what a set of neurons will do. Fortunately, functions can be used to construct circuits as well as analyze them. A mathematical tool to convert functions into circuits is *optimization theory.* There are many good references on optimization theory [Luenberger][Gill, et al.].

Optimization is ubiquitous in the field of neural networks. Many learning algorithms, such as back-propagation [Rumelhart, et al.], learn by minimizing the difference between desired results and observed results. Other neural algorithms use differential equations which minimize a function to solve a specified computational problem, such as associative memory [Hopfield], the traveling salesman problem [Durbin & Willshaw][Hopfield & Tank], analog decoding [Platt & Hopfield], and linear programming [Tank & Hopfield].

Constrained optimization techniques have been used to cause neural networks to fulfill constraints exactly [Ullman][Platt & Barr 87].

1.4 For the Neural Network Researcher

Many optimization models of neural networks need constraints to restrict the space of outputs to a manifold which satisfies external criteria. Optimizations using energy methods yield "forces" which act upon the state of the neural network. The penalty method, in which quadratic energy constraints are added to an existing optimization energy, has become popular recently [Hopfield & Tank][Koch, et al.][Poggio & Torre], but it is not guaranteed to satisfy the constraint conditions when there are other forces on the neural model or when there are multiple constraints.

In chapter 2, we present the *differential multiplier method* (DMM), which satisfies constraints exactly. Forces gradually apply the constraints over time, using "neurons" that estimate Lagrange multipliers. The differential multiplier method is a system of differential equations first proposed by [Arrow, et al.] as an economic model. These differential equations locally converge to a constrained minimum.

Examples of applications of the differential multiplier method include enforcing permutation codewords in the analog decoding problem (see chapter 4) and enforcing valid tours in the traveling salesman problem (see chapter 5).

1.5 Previous Work in Computer Graphics Modeling

There has been a growing interest in physical models in the field of computer graphics. [Weil] used catenaries and splines to approximate the effects of gravity on a cloth hanging from a few points. [Feynman] proposes a more sophisticated model for cloth which uses an energy that measures the bending and strain of a cloth. [Lundin] also has a model for cloth based on internal forces. In these three papers, a static shape of a cloth is computed. Chapter 6 illustrates how to compute the dynamics of a general flexible object [Terzopoulos, et al.].

The dynamics of elastic models are based on an analysis of deformation. Thus, chapter 6 is a extension of [Barr84], who statically deforms solid primitives using Jacobian matrices. [Sederberg & Parry] imposed similar deformations to solids modeled as free-form surfaces. Chapter 6 extends these approaches by adding equations governing the evolution of deformations.

The physically-based elastic models are based on classical elasticity theory. A recommended explanation of elasticity may be found in [Truesdell]; [Fung] is another useful reference for both elasticity and plasticity.

In order to make controllable modeling and animation, researchers in computer graphics have previously studied constraint methods [Badler] [Girard & Maciejewski] [Wilhelms & Barsky] [Armstrong & Green] [Seltzer] [Borning]. [Witkin, et al.] applied the penalty method to parametrized constraints. [Barzel & Barr] and [Isaacs & Cohen] developed dynamic constraints. Chapter 7 extends their work to flexible models [Platt & Barr 88].

1.6 For the Computer Graphics Implementor

Chapter 6 uses elasticity theory to construct differential equations that model the behavior of deformable curves, surfaces, and solids as a function of time. The theory of elasticity describes deformable materials such as rubber, cloth, paper, and flexible metals. Elastically deformable models are dynamic: they respond in a natural way to applied forces, constraints, ambient media, and impenetrable obstacles. The models are fundamentally dynamic and realistic animation is created by numerically solving their underlying differential equations. Thus, the description of shape and the description of motion are unified.

A primary goal of simulating flexible models is to animate physically realistic motions. Examples include simulating the musculature of a human body to create realistic walking, simulating the flow of viscous liquids, such as lava over volcanic rocks, or simulating a sculptor molding clay.

Chapter 7 takes a step towards these goals, by adding constraint properties to flexible models, and, by adding other properties, such as moldability and incompressibility. Using these properties, we can now simulate materials, such as clay, taffy, or putty, that have been very difficult to simulate using previous computer graphics models.

In order to create the pleasing and supple motions discussed above, we incorporate many of the following properties for our flexible models:

- *Physical Realism* — Flexible models should be able to move in natural, intuitive ways. Using the theory of elasticity to animate flexible models is very helpful in creating natural motion.
- *Controllability* — Flexible models should be able to follow an animation script. Models should be able to follow pre-defined paths exactly, while still wriggling in an interesting manner and interacting with other models.
- *Non-interpenetration* — Flexible models should be able to bounce off other models while using a small amount of computer time.
- *Limited Compressibility* — Flexible models should be able to have constant volume, even while being squashed. Models that squash without retaining their volume look as if they are made of sponge; they do not bulge out enough at the sides.
- *Moldability* — Flexible models should be moldable: external forces should mold the rest shape of the model. Models should follow the theory of *plasticity*, which describes materials that do not return to their rest shape after large deformation. Moldable models are a natural way to design shapes.

Chapter 7 applies constraints that apply forces to the physically-based models. The forces are computed via Lagrange multipliers. Chapter 7 also discusses an algorithm from mechanical engineering for computing Lagrange multipliers exactly, by solving a linear system.

1.7 For the Computer Scientist and Electrical Engineer

In the 1950s, computer science was stimulated by the existence proof of a biological computing engine: the brain. As serial digital machines became more powerful, the study of computer science flowered, although the ideal of simulating a brain became more elusive.

In the last decade, researchers have realized the limitations of serial machines. Much effort has gone into developing parallel machines. Parallelism is possible due to the advent of VLSI [Mead & Conway], with hundreds of thousands of transistors on one chip.

However, with the advent of VLSI comes the possibility of emulating neural hardware directly with the analog behavior of transistors. If a few transistors can emulate a neuron, then chips emulating thousands of neurons can be built. Carver Mead has been developing a field of study, called *synthetic neurobiology* which emulates real biological neural systems in silicon [Mead]. In order to do computation, the silicon neural structures must fulfill goals. Thus, constraints and teleological modeling can be applied to these silicon neural structures (see chapter 3).

Also with the increasing power of digital computers came the idea of creating interfaces between humans and computers. Since much of the human brain is devoted to processing visual data, it is useful to have computers to generate pictures for consumption by users.

A visual interface between computers and users can be created by *computer graphics.* Computer graphics is composed primarily of two branches, *modeling* and *rendering.* Modeling is the study of how to create an abstraction of a set of objects. (Notice that neural networks are also abstractions of neurons.) The methods in this thesis contribute to the study of modeling. Rendering is the study of how to create images, such as a colored picture on a screen, given a mathematical model of a set of objects. Rendering is based on the way light interacts with matter [Kajiya].

Computer graphics methods have increased in power and realism as computers have become more powerful. When computer graphics started, computers were very slow and computer memory was very expensive. Thus, it was a challenge to render even the simplest model. Originally, computer graphics used polyhedra to describe the model worlds to be rendered. Eventually, as computers sped up, models with more variables were used, such as bicubic patches. Eventually, algorithmically generated models were used. As the number of variables that described the models increased, the realism of the shape of the models also increased [Fournier, et al.][Fournier & Reeves] [Reeves & Blau].

Even though complex models were starting to be used, the motion of these models were still quite simple. The models' motions must be specified by hand, using splines. There is a study of how objects move: physics. Again, as computers got more powerful, computer graphics modelers started to use Newtonian physics to move the objects [Armstrong & Green] [Terzopoulos, et al.]

Chapters 6 and 7 present a unification of physics and constraints. An animator can specify as much or as little of the motion as desired. The rest of the motion is determined by a physical simulation.

1.8 For the Mechanical Engineer

Many of the tricks and tools in chapter 7 are similar to those used by designers to model real mechanical systems [Nikravesh]. The applications of these tools is new, however. Computer graphics researchers want to create artificial environments, then render them. Ideally, an artificial environment would respond interactively with a computer user. Also, models in computer graphics tend to undergo violent deformation and large amounts of rotation. Thus, the traditional study of small displacement linear elasticity is not applicable. Also, mechanical engineers are frequently very interested in internal stresses in models, whereas computer graphics researchers care about the motions of the surfaces of the models.

1.9 For the Computer Vision Researcher

For a number of years, computer vision researchers have used regularization to solve ill-posed problems in computer vision [Poggio & Torre]. Regularization has been used for edge detection [Kass, et al.], surface reconstruction [Koch, et al.][Terzopoulos 83], and shape from shading [Horn]. Regularization can be used to decrease the noise in the image. But, regularization is optimization with the penalty method, it only fulfills constraints approximately. Ullman used a method similar to the DMM to enforce a constraint exactly [Ullman]. The DMM described in chapter 2 can be used to enforce general constraints on regularization.

Finally, since computer vision can be viewed as inverse problem of computer graphics, the models presented in this paper are of value for reconstructing mathematical representations of non-rigid objects from their images [Terzopoulos 87] (see chapter 5).

Constrained Neural Networks: Theory

Chapter 2 : Constrained Optimization Methods

2.1 Introduction: Optimization for Circuits

An outstanding problem in computer science is to create computers that can perceive and think. Much effort has gone into creating computers with artificial intelligence. Often, researchers design artificial intelligence systems using LISP software that runs on serial synchronous digital hardware.

Serial synchronous digital hardware has problems with perceptual and cognitive tasks with large amounts of noisy input data. If the large amount of data must be handled sequentially, then the hardware is very slow. If the hardware operates in parallel, then as the size of the problem increases, synchronization and heat dissipation become serious problems.

The brain quickly solves perceptual and cognitive tasks with parallel analog hardware, without synchronization or extreme heat dissipation problems. It does so by using large systems of differential equations to compute. Systems of differential equations are naturally analog and parallel. This thesis explores the use of differential equations to solve certain perceptual and cognitive tasks.

Large systems of non-linear differential equations can be difficult to design. The difficulty can be reduced using a design method; a good design method starts with a description of a task and mechanically produces a circuit which performs the task.

Chapter 2 describes a design method based on constrained optimization. A perceptual or cognitive task may be described by properties of its solution. These properties can be divided into two sets: desirable properties and necessary properties.

A circuit to generate a solution with desirable properties can be designed with optimization theory (see figure 2.1). The task is expressed as a minimization of some function. The function is small when the output of the system contains desirable properties of a solution. Now, construct a differential equation that performs the optimization. This differential equation is the neural network, and can be simulated with numerical analysis, or implemented directly as a circuit [Poggio & Torre] [Koch, et al.] [Hopfield & Tank].

This chapter discusses how to create a system of differential equations that generates both desirable and necessary properties of a solution. The necessary properties of a solution are expressed as constraints. The system of differential equations from optimization theory is modified to perform constrained optimization. These differential equations can again be simulated with numerical analysis, or implemented directly as a circuit [Platt & Hopfield] [Platt & Barr].

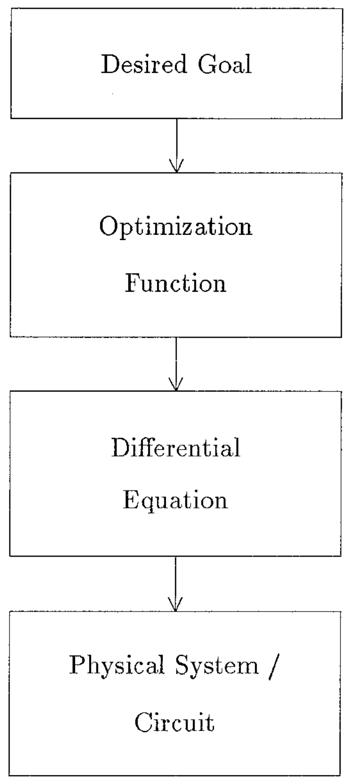


Figure 2.1. How to use optimization theory to create circuits

2.2 Gradient Descent

Optimization algorithms commonly take discrete steps, which gradually try to minimize a function. Many of these algorithms, such as Newton's method [Press, et al.], have excellent local convergence properties. However, analog circuits operate continuously, not by discrete steps. This section describes less advanced optimization algorithms which produce differential equations and hence circuits.

The simplest differential optimization algorithm is *gradient descent*, where the state x_i slides downhill, in the opposite direction of the gradient [Foulds] (see figure 2.2). If the function to be minimized is $\mathcal{E}(\underline{x})$, then at any point, the vector that points towards the the direction of maximum increase of $\mathcal{E}$ is the gradient of $\mathcal{E}$, namely $d\mathcal{E}/dx_i$.

$$\frac{dx_i}{dt} = -\frac{\partial \mathcal{E}}{\partial x_i}. \tag{2.1}$$

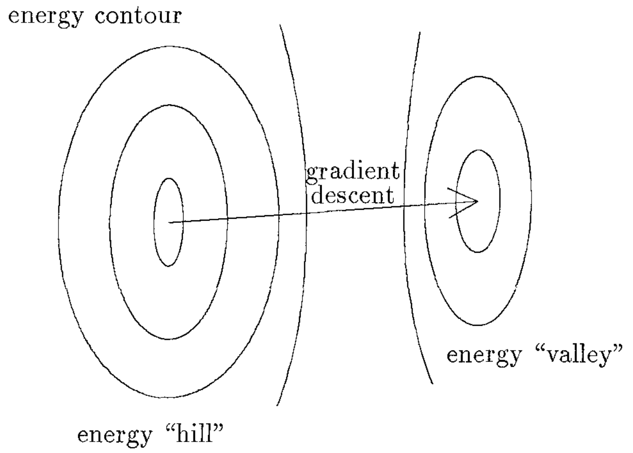


Figure 2.2. Gradient Descent

In circuit design, the x_i represent voltages on a capacitor. In neurobiology, the x_i represent the mean firing rate of a neuron. In general, the x_i are called "moving variables."

Because the state x_i slides downhill, the function $\mathcal{E}$ always decreases, until a local minimum is reached.

$$\frac{d\mathcal{E}}{dt} = \sum_i \frac{dx_i}{dt}\frac{\partial \mathcal{E}}{\partial x_i} = -\sum_i \left(\frac{\partial \mathcal{E}}{\partial x_i}\right)^2 \leq 0. \tag{2.2}$$

Gradient descent is equivalent to immersing the optimization landscape in a viscous fluid like honey and allowing a ball to slowly fall down the optimization function.

The programming of the circuit consists of designing the function to be minimized. The function consists of terms that are minimized when the state x_i fulfills some desirable properties of an answer.

The circuit starts in some initial condition x_i, slides downhill and comes to rest in some local minimum. The circuit can then be reset with an external input to a new initial condition. Or, more elegantly, the optimizing function itself can depend on external inputs, with the state responding continually to any changes in the optimizing function.

The state does not need to slide downhill directly opposite to the gradient. As long as the function $\mathcal{E}$ is decreasing, the system will find a local minimum (if one exists). For example,

$$\frac{dx_i}{dt} = -\sum_j Q_{ij}(\underline{x}, t)\frac{\partial \mathcal{E}}{\partial x_j} \tag{2.3}$$

converges if Q_{ij} is positive definite, since

$$\frac{d\mathcal{E}}{dt} = \sum_i \frac{dx_i}{dt}\frac{\partial \mathcal{E}}{\partial x_i} = \sum_{i,j} \frac{dx_i}{dt} Q_{ij} \frac{dx_j}{dt}. \tag{2.4}$$

The minimization in equation (2.4) arises from circuits. Circuits that perform gradient descent often have different time constants for every variable x_i. These time constants can be represented as a diagonal Q matrix, which is positive definite. Also, equations of the form (2.4) describe a general set of circuits which minimize co-content.

2.3 Quadratic Forms

One of the simplest optimization functions that can be used is the quadratic form.

$$\mathcal{E} = \sum_{i,j} x_i T_{ij} x_j, \tag{2.5}$$

where T_{ij} is a symmetric matrix.

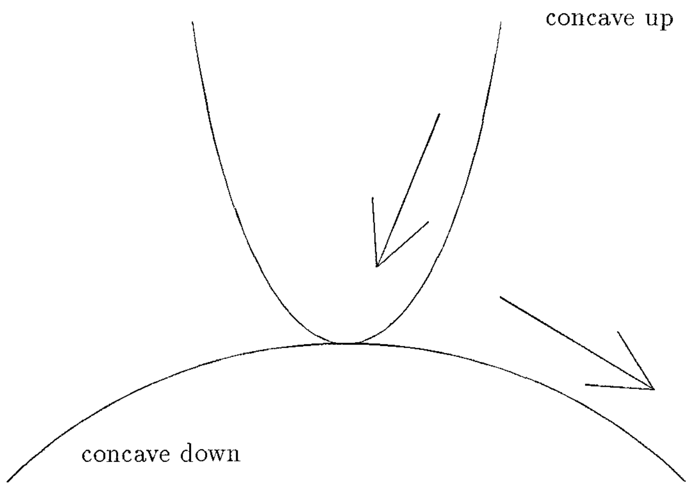


Figure 2.3. Quadratic landscape

The landscape described by the quadratic form looks like a high-dimensional saddle: any slice through the landscape is always a parabola, either concave upwards or concave downwards (see figure

2.3). If the matrix T_{ij} is positive definite, then the landscape looks like a bowl, with the lowest point at $\underline{x} = 0$. Similarly, if T_{ij} is negative definite, then the landscape looks like an upside-down bowl, with the highest point at $\underline{x} = 0$ [Foulds].

The quadratic form corresponds to N neurons symmetrically connected to each other with linear synapses. If the matrix T_{ij} has no zero elements, then every neuron is connected to every other one. Full connectivity is not biologically realistic, nor easy to build in VLSI. If the matrix T_{ij} is sparse (has many zero elements), then fewer wires are necessary, and the circuit becomes easier to build.

Linear terms can be added to the quadratic form:

$$\mathcal{E} = \sum_{i,j} x_i T_{ij} x_j + \sum_i I_i x_i. \tag{2.6}$$

The linear terms correspond to external inputs I_i to each neuron i.

When the quadratic form is not positive definite, then the function is concave up in some directions and concave down in others and gradient descent sends one or more of the state variables x_i to infinity. In real circuits, the power supply voltages limit the possible voltages of the circuits. In biology, the Nernst potentials of the ion pumps also limit the voltages to a fixed region. Thus, quadratic and linear terms often are used with extra terms that limit the state variables to a region of state space.

2.4 Constrained Optimization

A *constrained optimization procedure* finds a minimum of a function on one or more specified manifolds. The prototypical constrained optimization problem can be stated as

$$\text{locally minimize } f(\underline{x}), \text{ subject to } g(\underline{x}) = 0, \tag{2.7}$$

where $g(\underline{x}) = 0$ is a scalar equation describing a manifold of the state space [Gill, et al.]. During constrained optimization, the state vector $\underline{x}$ should be attracted to the manifold $g(\underline{x}) = 0$ and slide along the manifold until it reaches the locally smallest value of $f(\underline{x})$ on $g(\underline{x}) = 0$. Solutions to a constrained optimization problem are restricted to a subset of the solutions of the corresponding unconstrained optimization problem.

Constrained optimization is very useful for creating differential equations for neural networks. There are only a limited set of possible answers to an constrained optimization problem. For example, in the solution to the traveling salesman problem, the salesman may visit each city only once [Lawler, et al.]. Constrained optimization can be used to cause the output of a neural network to be in a limited set.

Conversely, differential equations solve constrained problems better than non-linear algebraic equation. The differential equations are never under-constrained or over-constrained. Also, if the constraints are satisfied gradually, then the system has an opportunity to settle into a good local minimum before being constrained.

2.5 The Penalty Method

The physical interpretation of the penalty method is a rubber band that attracts the state to the manifold $g(\underline{x}) = 0$. The rubber band gets stronger as time increases, so that the state approaches $g(\underline{x}) = 0$. The penalty method adds a quadratic optimization term that penalizes violations of constraints [Hestenes]. Thus, the constrained minimization problem (2.7) is converted to the following unconstrained minimization problem (see figure 2.4):

$$\min \mathcal{E}_{\text{penalty}}(\underline{x}) = f(\underline{x}) + c(g(\underline{x}))^2. \tag{2.8}$$

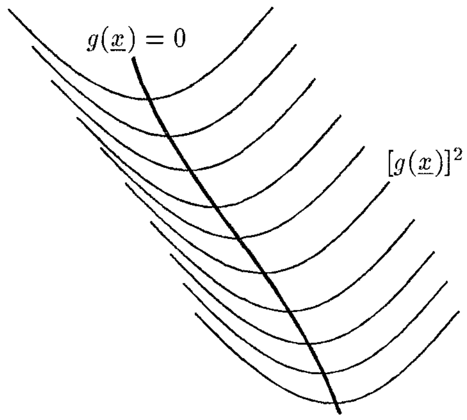


Figure 2.4. The penalty method makes a trough in state space

In this thesis, the penalty method is defined to be one minimization, with a fixed c. In other works, the penalty method is defined as a sequence of unconstrained minimization problems, where

$c \to \infty$ as the sequence progresses [Bertsekas]. The sequence of unconstrained minima gradually approaches the constrained minimum.

As an example, consider finding the closest point to the origin on the line $x + y = 1$. To use the penalty method, minimize

$$\mathcal{E} = x^2 + y^2 + c(x + y - 1)^2. \tag{2.9}$$

As $c \to \infty$, the minima approaches $(x, y) = (0.5, 0.5)$.

The penalty method can enforce inequality constraints by adding an exponential penalty as the inequality constraint is violated. Thus, if a minimum of $f(\underline{x})$ is constrained by $g(\underline{x}) > 0$, then find a minimum of

$$\mathcal{E} = f(\underline{x}) + \exp(-kg(\underline{x})). \tag{2.10}$$

Again, as $k \to \infty$, the inequality constraint will be fulfilled.

The penalty method can be extended to fulfill multiple constraints by using more than one rubber band. Namely, the constrained optimization problem

$$\min f(\underline{x}), \text{ subject to } g_\alpha(\underline{x}) = 0; \quad \alpha = 1, 2, \ldots, n, \tag{2.11}$$

is converted into unconstrained optimization problem (see figure 2.4)

$$\min \mathcal{E}_{\text{penalty}}(\underline{x}) = f(\underline{x}) + \sum_{\alpha=1}^{n} c_\alpha (g_\alpha(\underline{x}))^2. \tag{2.12}$$

The penalty method has a few convenient features.

- *Inexact Constraints* — There are situations in which it is not necessary to exactly fulfill constraints; sometimes it is desirable to compromise between constraints. Therefore, we can use a finite constraint strength and allow these compromises.
- *Ease of Use* — Adding a quadratic optimization term to an optimization function is simple and requires no extra differential equations.

However, the penalty method has number of disadvantages.

- *Inexact Constraints* — For finite constraint strengths c_α, the penalty method does not fulfill the constraints precisely. Under many circumstances, however, constraints should be fulfilled exactly. Using multiple rubber band constraints is like building a machine out of rubber bands; the machine would not hold together perfectly.
- *Many free parameters* — If there are many constraints, then the parameters that fulfill the constraints, yet find deep local minimum, are hard to choose.

- *Stiffness of Equations* — As the constraint strengths increase, the differential equations become *stiff*; that is, there are widely separated time constants. Most numerical methods must take time steps on the order of the fastest time constant, while most modelers are interested in the behavior at the slowest time constant. As a result of stiffness, the numerical differential equation solver takes very small time steps, using a large amount of computing time without getting much done.
- *Needs infinitely fast analog circuitry* — In order to implement the penalty method in analog circuitry, one usually implements a circuit that computes g. This circuit should be infinitely fast in order to compute the algebraic function. If the circuit is not infinitely fast, then the differential equation undergoes a singular perturbation and oscillatory behavior might result [Nayfeh].

2.6 Exact Penalty Method

The penalty function can be modified to render unnecessary a sequence of penalty strengths increasing to infinity. A differentiable penalty term has a flat region around the constraint manifold where the derivative is small. Therefore, the term must be multiplied by a large number to have an effect near the constraint manifold.

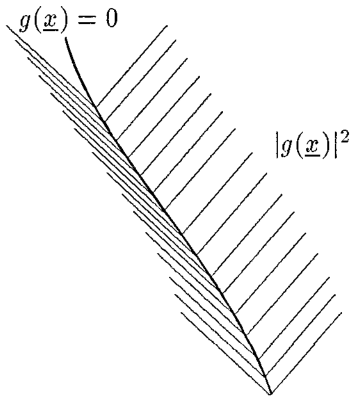


Figure 2.5. The exact penalty method

If we use a non-differentiable penalty term, the derivative may be large near the constraint manifold. Consider a penalty term of $|g|$ (see figure 2.5) [Gill, et al.].

$$\mathcal{E} = f(\underline{x}) + c|g(\underline{x})|, \tag{2.13}$$

$$\dot{x}_i = -\frac{\partial f}{\partial x_i} - c\,\mathrm{sgn}(g(\underline{x}))\frac{\partial g}{\partial x_i}. \tag{2.14}$$

For $c > c^*$, for some $c^* > 0$, the system exactly converges to the constraints, because if there is a non-zero force at the constrained minimum due to $\partial f/\partial x_i$, then the $c\,\mathrm{sgn}(g)$ term will overwhelm it for large enough c. Unfortunately, differential equation (2.14) does not have a Lipschitz bound. However, the signum function can be approximated with a high-gain tanh function, which is another amplifier. The accuracy of the constraint depends on the gain of the subsidiary amplifier.

2.7 Lagrange Multipliers

Lagrange multiplier methods, like the penalty method, convert constrained optimization problems into unconstrained extremization problems. Namely, a solution to the equation (2.7) is also a critical point of the function

$$\mathcal{E}_{\text{Lagrange}}(\underline{x}) = f(\underline{x}) + \lambda g(\underline{x}). \tag{2.15}$$

λ is called the Lagrange multiplier for the constraint $g(\underline{x}) = 0$ [Hestenes].

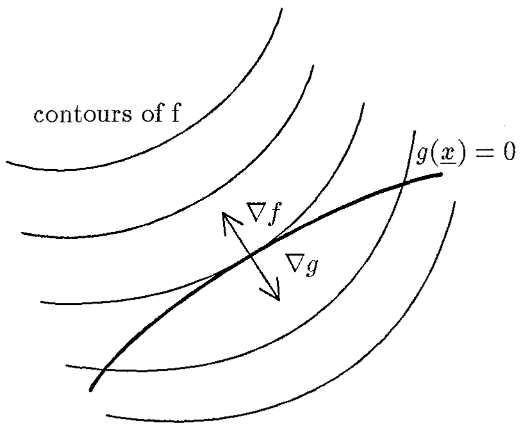


Figure 2.6. At the constrained minimum, $\nabla f = -\lambda \nabla g$

A direct consequence of equation (2.15) is that the gradient of f is collinear to the gradient of g at the constrained extrema (see figure 2.6). The constant of proportionality between ∇f and ∇g

is $-\lambda$:

$$\nabla \mathcal{E}_{\text{Lagrange}} = 0 = \nabla f + \lambda \nabla g. \tag{2.16}$$

A simple example shows that Lagrange multipliers provide the extra degrees of freedom necessary to solve constrained optimization problems. Again, consider the problem of finding a point (x, y) on the line $x + y = 1$ that is closest to the origin. Using Lagrange multiplier techniques,

$$\mathcal{E}_{\text{Lagrange}} = x^2 + y^2 + \lambda(x + y - 1). \tag{2.17}$$

Now, take the derivative with respect to all variables, x, y, and λ.

$$\begin{aligned}
\frac{\partial \mathcal{E}_{\text{Lagrange}}}{\partial x} &= 2x + \lambda = 0,\\
\frac{\partial \mathcal{E}_{\text{Lagrange}}}{\partial y} &= 2y + \lambda = 0,\\
\frac{\partial \mathcal{E}_{\text{Lagrange}}}{\partial \lambda} &= x + y - 1 = 0.
\end{aligned} \tag{2.18}$$

With the extra variable λ, there are now three equations in three unknowns. In addition, the last equation is precisely the constraint equation.

Applying gradient descent in equation (2.1) to the function in equation (2.15) yields

$$\begin{aligned}
\dot{x}_i &= -\frac{\partial \mathcal{E}_{\text{Lagrange}}}{\partial x_i} = -\frac{\partial f}{\partial x_i} - \lambda \frac{\partial g}{\partial x_i},\\
\dot{\lambda} &= -\frac{\partial \mathcal{E}_{\text{Lagrange}}}{\partial \lambda} = -g(\underline{x}).
\end{aligned} \tag{2.19}$$

Note that there is an auxiliary differential equation for λ, which is necessary to apply the constraint $g(\underline{x}) = 0$. Also, recall that when the system is at a constrained extremum, $\nabla f = -\lambda \nabla g$, hence, $\dot{x}_i = 0$.

Solutions to the constrained optimization problem (2.7) are saddle points of the function in equation (2.15), which has no lower bound [Arrow, et al.]. If the vector $\underline{x}$ is held fixed where $g(\underline{x}) \neq 0$, the optimization function can be decreased to $-\infty$ by sending λ to $+\infty$ or $-\infty$.

The gradient descent method does not work with Lagrange multipliers, because a critical point of the function in equation (2.15) need not be an attractor for equations (2.19). A stationary point must be a local minimum in order for gradient descent to converge.

2.8 Differential Multiplier Method

There is an alternative to differential gradient descent that estimates the Lagrange multipliers, so that the constrained minima are attractors of the differential equations, instead of "repellors." The differential equations that solve (2.7) are

$$\begin{aligned} \dot{x}_i &= -\frac{\partial f}{\partial x_i} - \lambda \frac{\partial g}{\partial x_i}, \\ \dot{\lambda} &= +g(\underline{x}). \end{aligned} \tag{2.20}$$

The algorithm described in equation (2.20) is called the basic differential multiplier method (BDMM).

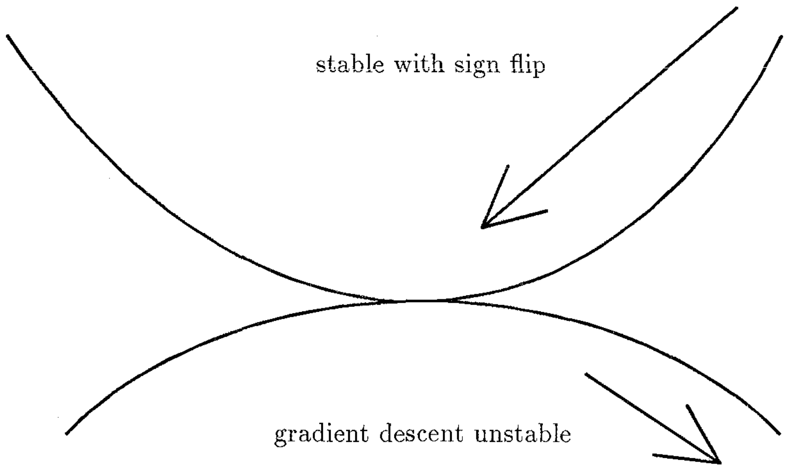


Figure 2.7. The sign flip from equation (2.19) to equation (2.20) makes the differential multiplier method stable

Equations (2.20) are similar to equations (2.19). As in equations (2.19), solutions to equation (2.7) are stationary points of equations (2.20). Notice, however, the sign inversion in the equations (2.20), as compared to equations (2.19). The equation (2.20) is performing gradient *ascent* on λ. The sign flip makes the method stable (see figure 2.7).

The system of differential equations (2.20) gradually fulfills the constraints. Notice that the function $g(\underline{x})$ can be replaced by $kg(\underline{x})$, without changing the location of the constrained minimum. As k is increased, the state begins to undergo damped oscillation about the constraint manifold $g(\underline{x}) = 0$. As k is increased further, the frequency of the oscillations increase, and the time to convergence increases (see figure 2.7).

One extension to equations (2.20) is an algorithm for constrained minimization with multiple constraints. Adding an extra differential equation for every equality constraint and summing all of the constraint forces creates the optimization function

$$\mathcal{E}_{\text{multiple}} = f(\underline{x}) + \sum_{\alpha} \lambda_\alpha g_\alpha(\underline{x}), \tag{2.21}$$

which yields differential equations

$$\begin{aligned} \dot{x}_i &= -\frac{\partial f}{\partial x_i} - \sum_{\alpha} \lambda_\alpha \frac{\partial g_\alpha}{\partial x_i}, \\ \dot{\lambda}_\alpha &= +g_\alpha(\underline{x}). \end{aligned} \tag{2.22}$$

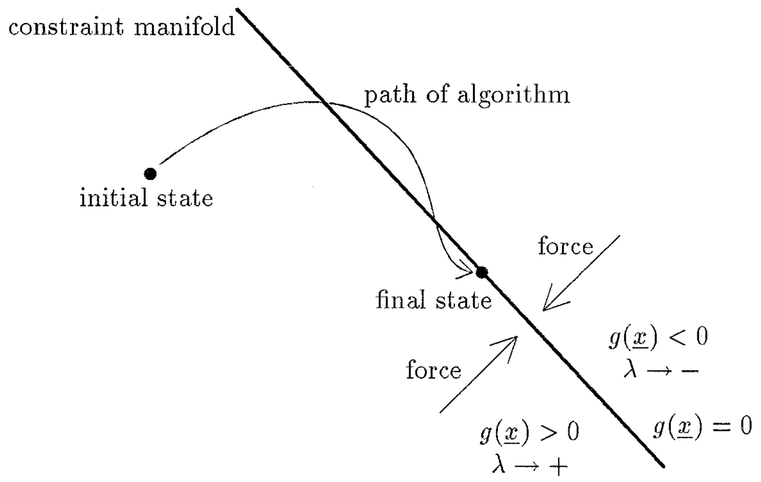


Figure 2.8. The state is attracted to the constraint manifold

Another extension is constrained minimization with inequality constraints. As in traditional optimization theory [Hestenes], one uses additional slack variables to convert inequality constraints into equality constraints. Namely, a constraint of the form $h(\underline{x}) \geq 0$ can be expressed as

$$g(\underline{x}) = h(\underline{x}) - z^2. \tag{2.23}$$

Since z^2 must always be positive, then $h(\underline{x})$ is constrained to be positive. The slack variable z is treated like a component of $\underline{x}$ in equation (2.20). An inequality constraint requires two extra differential equations, one for the slack variable z and one for the Lagrange multiplier λ.

Lagrange multipliers can be applied to systems of differential equations which do not precisely descend the gradient. For example, consider the unconstrained minimization in equation (2.4). The system of differential equations

$$\sum_j Q_{ij}\dot{x}_j = -\frac{\partial f}{\partial x_i} - \lambda\frac{\partial g}{\partial x_i} \qquad \dot{\lambda} = g(\underline{x}) \tag{2.24}$$

constrains the system in equation (2.4) to lie on the manifold $g = 0$. The stability of equations (2.24) is discussed in Appendix B. Therefore, the differential multiplier method is applicable to real circuits with mismatched time constants (see chapter 3).

Combining the basic differential multiplier method with the penalty method yields the modified differential multiplier method (MDMM). The MDMM has better convergence properties the BDMM. The BDMM is completely compatible with the penalty method. If one adds a penalty force to equation (2.20) that corresponds to a quadratic term

$$E_{\text{penalty}} = \frac{c}{2}(g(\underline{x}))^2, \tag{2.25}$$

then the set of differential equations for a MDMM is

$$\dot{x}_i = -\frac{\partial f}{\partial x_i} - \lambda\frac{\partial g}{\partial x_i} - cg\frac{\partial g}{\partial x_i}, \qquad \dot{\lambda} = g(\underline{x}). \tag{2.26}$$

The extra force from the penalty does *not* change the position of the stationary points of the differential equations, because the penalty force is zero when $g(\underline{x}) = 0$, independent of the value of c.

There is a minimum necessary penalty strength c required in some cases for the MDMM to converge. The minimum penalty strength in the MDMM is usually much less than the strength needed by the penalty method for an accurate solution [Bertsekas].

2.9 Rate-Controlled Constraints

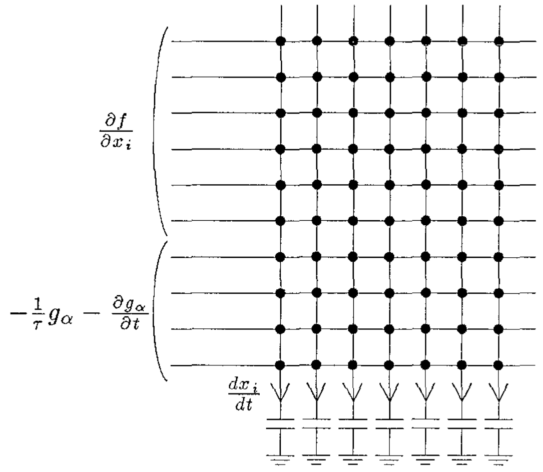
Analogous to the constraint stabilization method [Nikravesh], the Lagrange multipliers can be computed exactly, given that the constraint is fulfilled exponentially, with time constant τ. A constrained optimization problem with time varying constraints is converted into the extremization of

$$\mathcal{E} = f(\underline{x}) + \sum_\alpha \lambda_\alpha g_\alpha(\underline{x}, t). \tag{2.27}$$

Expressing the gradient descent as a differential-algebraic equation yields

$$\frac{dx_i}{dt} = \frac{\partial f}{\partial x_i} - \sum_\alpha \lambda_\alpha \frac{\partial g_\alpha}{\partial x_i}, \tag{2.28}$$

$$g_\alpha(\underline{x}, t) = 0. \tag{2.29}$$



$\frac{\partial f}{\partial x_i}$
$-\frac{1}{\tau}g_\alpha - \frac{\partial g_\alpha}{\partial t}$
$\frac{dx_i}{dt}$

Figure 2.9. Rate-Controlled Constraints add an extra synaptic matrix. The dots (•) are a synapse: they take the voltage on the horizontal wire, multiply it by a stored constant, then inject the result as a current on the vertical wires.

To construct a circuit, we need a pure differential equation. The algebraic constraint equations can be converted into differential equations that fulfill the constraints exponentially with time constant τ:

$$\frac{dg_\alpha}{dt} + \frac{1}{\tau} g_\alpha = 0. \tag{2.30}$$

Replacing equation (2.29) with equation (2.30) and combining with equation (2.28) yields

$$\begin{aligned} \frac{dx_i}{dt} + \sum_\alpha \lambda_\alpha \frac{\partial g_\alpha}{\partial x_i} &= -\frac{\partial f}{\partial x_i}, \\ \frac{dx_j}{dt}\frac{\partial g_\alpha}{\partial x_i} &= -\frac{1}{\tau} g_\alpha - \frac{\partial g_\alpha}{\partial t}. \end{aligned} \tag{2.31}$$

Equation (2.31) is a linear system in dx_i/dt and λ_α, which can be written in matrix form

$$\begin{bmatrix} \delta_{ij} & \frac{\partial g_\alpha}{\partial x_i} \\ \partial g_\alpha/\partial x_i & 0 \end{bmatrix} \begin{pmatrix} \frac{dx_i}{dt} \\ \lambda_\alpha \end{pmatrix} = \begin{pmatrix} -\frac{\partial f}{\partial x_i} \\ -\frac{1}{\tau} g_\alpha - \frac{\partial g_\alpha}{\partial t} \end{pmatrix}. \tag{2.32}$$

This system of linear equations can be solved for dx_i/dt and λ_α. If the $\partial g_\alpha/\partial x_i$ are constants, then the matrix is a constant and can be inverted once. The matrix inverse corresponds to a synaptic matrix as shown in figure 2.9.

If a subset of the neurons x_i are affected by only one constraint $g = 0$, then the Lagrange multiplier can be computed exactly for that one constraint, without resorting to solving a linear system.

$$\lambda = \frac{\frac{1}{\tau} g + \frac{\partial g}{\partial t} - \sum_i \frac{\partial f}{\partial x_i} \frac{\partial g}{\partial x_i}}{\sum_i \frac{\partial g}{\partial x_i} \frac{\partial g}{\partial x_i}}. \qquad (2.33)$$

2.10 Conclusions

This chapter reviews constrained optimization methods that are appropriate for constraining neural network differential equations, such as the penalty method, the differential multiplier method and rate-controlled constraints. The penalty method is a classical optimization method previously used to create neural networks [Hopfield & Tank]. The differential multiplier method and rate-controlled constraints are new to the field of neural networks.

The neural network differential equations can be converted into VLSI circuits in order to quickly perform the constrained optimization. Chapter 3 experimentally verifies the applicability of the methods of this chapter to building circuits. Chapters 4 and 5 explore the utility of constrained neural networks in solving hard problems and contrast the various methods described in this chapter.

Constrained Neural Networks: Applications

Chapter 3 : Constrained Circuits

3.1 Introduction: Analog Circuits for Constrained Optimization

Analog circuits can process large amounts of noisy data by implementing a differential equation that finds a minimum of some function [Sivilotti, et al.][Koch, et al.][Tanner]. Because of inaccuracies in the components, real circuits only approximately solve the desired differential equations. Dynamical system theory can often predict whether a small change in a differential equation affects the qualitative behavior of the differential equation [Guckenheimer & Holmes][Abrahams & Shaw]. Unfortunately, because of the high dimensionality of the system of differential equations associated with analog circuits, the stability of the behavior of the circuits against perturbations is hard to prove mathematically. Only by building the circuits do we actually demonstrate whether the differential equations are adequately approximated.

This chapter explores whether the differential constrained optimization algorithms described in chapter 2 can be adaquately approximated by circuits. Two example circuits are constructed: a circuit that solves quadratic programming and a circuit that constrains a flip-flop. The circuits are then shown to fulfill the correct constraints.

3.2 Quadratic Programming Circuit

This section describes a circuit that solves quadratic programming for two variables. The circuit is easily generalizable to quadratic programming for N variables. A quadratic programming circuit is interesting, because the basic differential multiplier method is guaranteed to find the constrained minimum (see appendix B). Also, quadratic programming is useful: it is frequently a sub-problem in a more complex task. A method of solving general nonlinear constrained optimization is sequential quadratic programming [Gill, et al.].

The quadratic programming problem for two variables is

$$\min A(x - x_0)^2 + B(y - y_0)^2, \tag{3.1}$$

subject to the constraint

$$Cx + Dy = E(t). \tag{3.2}$$

The basic differential multiplier method from chapter 2 converts the quadratic programming problem into a system of differential equations:

$$\begin{aligned} k_1\frac{dx}{dt} &= -2Ax + 2Ax_0 - C\lambda, \\ k_2\frac{dy}{dt} &= -2By + 2By_0 - D\lambda, \\ k_3\frac{d\lambda}{dt} &= Cx + Dy + E(t). \end{aligned} \tag{3.3}$$

The first two equations are implemented with a resistor and capacitor (with a follower for zero output impedance). The third is implemented with resistor summing into the negative input of a transconductance amplifier. The positive input of the amplifier is connected to $E(t)$.

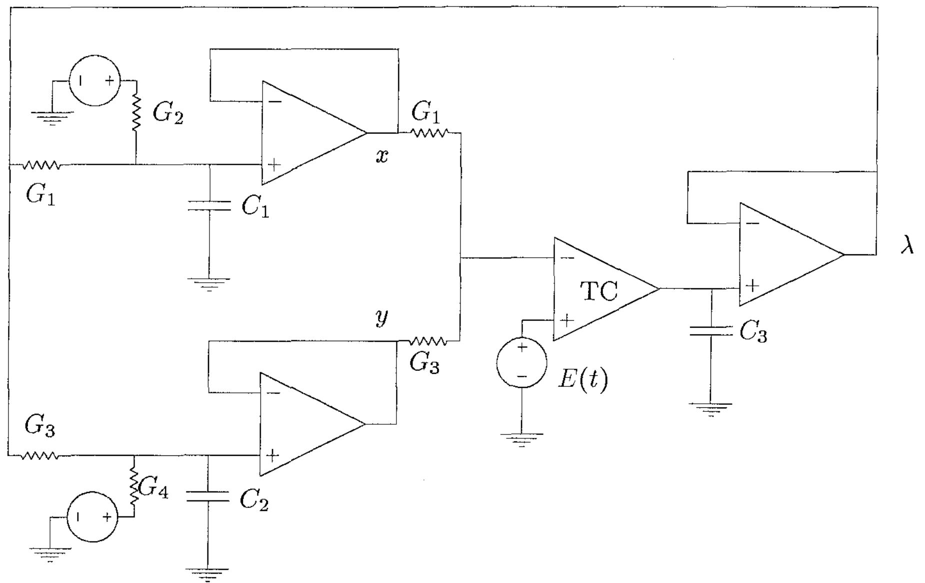


Figure 3.1. A circuit that implements quadratic programming. x, y, and λ are voltages.

The circuit in figure 3.1 implements the system of differential equations

$$\begin{aligned} C_1\frac{dx}{dt} &= G_1(\lambda - x) + G_2(V_x - x), \\ C_2\frac{dy}{dt} &= G_4(\lambda - y) + G_3(V_y - y), \\ C_3\frac{d\lambda}{dt} &= K\left(E(t) - \frac{G_1x + G_4y}{G_1 + G_4}\right), \end{aligned} \tag{3.4}$$

where K is the transconductance of the transconductance amplifier. The two systems of differential equations (3.3) and (3.4) are identical when

$$\begin{aligned} A &= G_1 + G_2, \\ B &= G_4 + G_5, \\ C &= -G_1, \\ D &= -G_4, \\ x_0 &= \frac{G_2 V_x}{G_1 + G_2}, \\ y_0 &= \frac{G_5 V_y}{G_4 + G_5}, \\ k_1 &= C_1, \\ k_2 &= C_2, \\ k_3 &= \frac{C_3(G_1 + G_4)}{K}. \end{aligned} \tag{3.5}$$

The circuit in figure 3.1 actually performs quadratic programming. The constraint is fulfilled when the voltages on the inputs of the transconductance amplifier are the same. The g function is a difference between these voltages. Figure 3.4 is a plot of the voltages as a function of time: they match reasonably well. The circuit in figure 3.1 therefore seems to successfully perform quadratic programming.

One can also plot the output variables x and y (see figure 3.5). As the constraint is smoothly changed by $E(t)$, the position of the constrained minimum also smoothly changes.

Decreasing the capacitance C_3 changes the Lyapunov function that governs the differential equation (see Appendix B). The forces that push the system towards the constraint manifold are increased without changing the damping. Therefore, the system becomes underdamped and the constraint is fulfilled with ringing (see figure 3.6).

3.3 Constrained Flip-flop

A flip-flop is two inverters hooked together in a ring. It is a bistable circuit: one inverter is on while the other inverter is off. A flip-flop can also be considered the simplest neural network: two neurons which inhibit each other.

If the inverters have infinite gain, then the flip-flop in figure 3.2 minimizes the function

$$\mathcal{E}_{\text{flip-flop}} = G_2 V_1 U_2 + G_4 V_2 U_1 - G_1 I_1 U_1 - G_3 I_2 U_2 + \frac{G_1 + G_2}{2} U_1^2 + \frac{G_3 + G_4}{2} U_2^2. \tag{3.6}$$

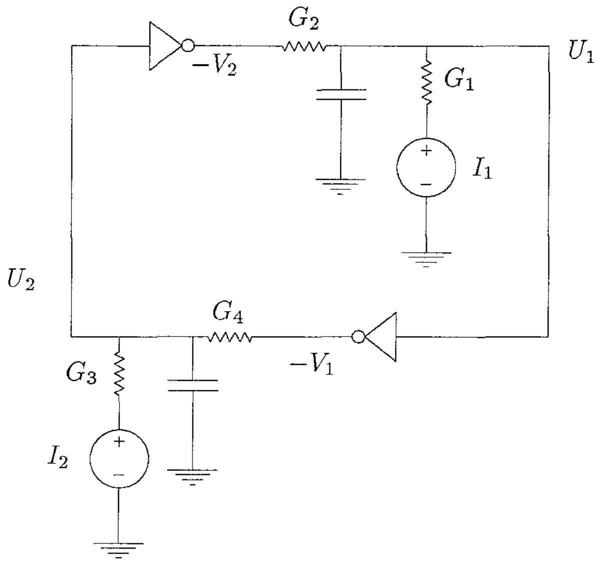


Figure 3.2. A flip-flop. U_1 and U_2 are voltages.

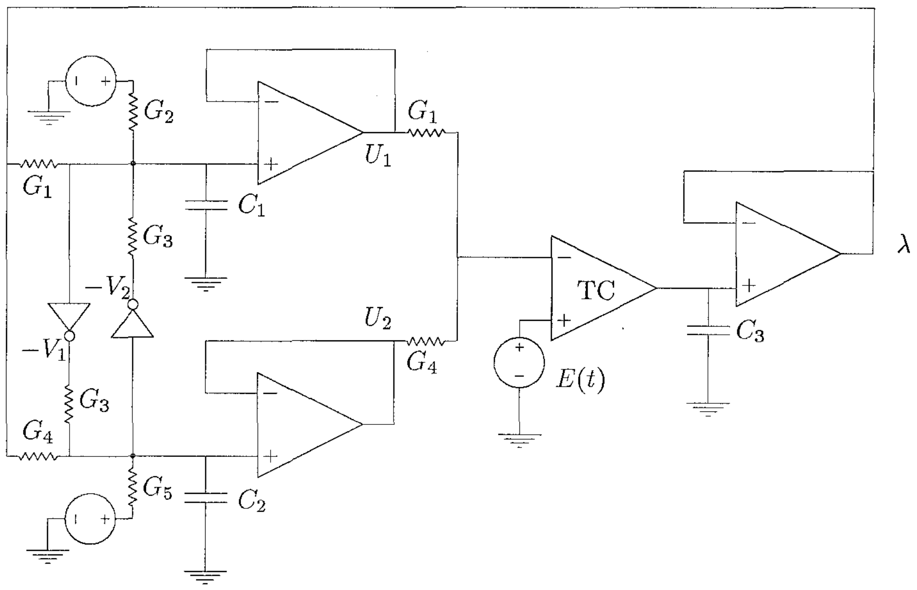


Figure 3.3. A circuit for constraining a flip-flop. U_1, U_2, and λ are voltages.

Now, we can construct a circuit that minimizes the function in equation (3.6), subject to some linear constraint $Cx + Dy = E(t)$, where x and y are the inputs to the inverters. The circuit diagram is shown in figure 3.3. Notice that this circuit is very similar to the quadratic programming circuit. Now, the x and y circuits are linked with a flip-flop, which adds non-linear terms to the optimization function.

A circuit was built with

$$C = 2/7, \quad D = 5/7. \tag{3.7}$$

The voltages $Cx + Dy$ and $E(t)$ for this circuit are plotted in figure 3.7. For most of the time, $Cx + Dy$ is close to the externally applied voltage $E(t)$. However, because $C \neq D$, the flip-flop moves from one minima to the other and the constraint is temporarily violated. But, the circuitry gradually enforces the constraint again. The temporary constraint violation can be seen in figure 3.7.

The plot of the U_i voltages in figure 3.8 shows that the flip-flop does change state as the constraint is varies sinusoidally over the optimization terrain. Between sudden changes, the voltages smoothly follow the constraint.

3.4 Conclusions

This chapter examines real circuits that have been constrained with the differential multiplier method. The differential multiplier method seems to work, even when the underlying circuit is non-linear, as in the case of the constrained flip-flop. Chapters 4 and 5 examine applications of constrained neural networks. These applications could be built with the same parallel analog hardware discussed in this chapter.

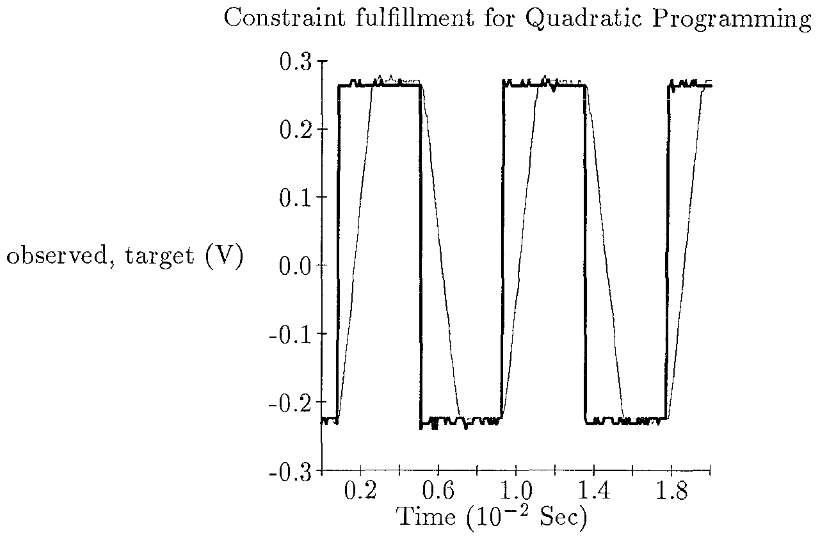


Figure 3.4. Plot of two input voltages of transconductance amplifier. The constraint depends on time: the voltage $E(t)$ is a square wave. The linear constraint is fulfilled when the two voltages are the same. The unusually shaped noise is caused by digitization by the oscilloscope.

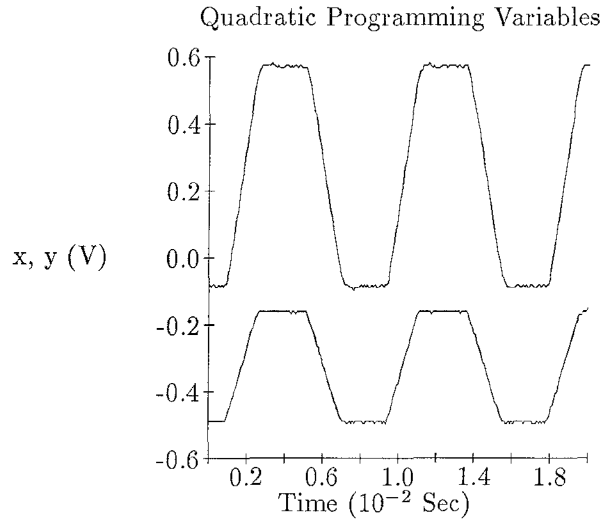


Figure 3.5. Plot of two output voltages x and y: the voltages are the result of the quadratic programming

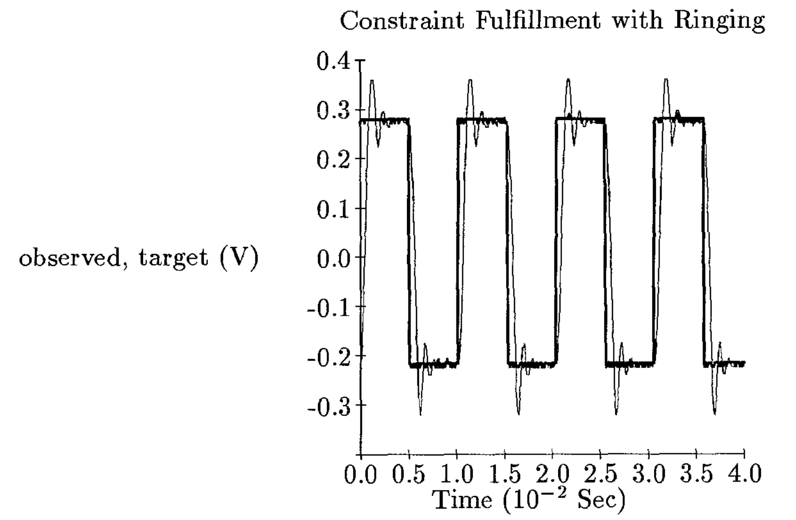


Figure 3.6. Plot of two input voltages of transconductance amplifier: the constraint forces are increased, which causes the system to undergo damped oscillations around the constraint manifold.

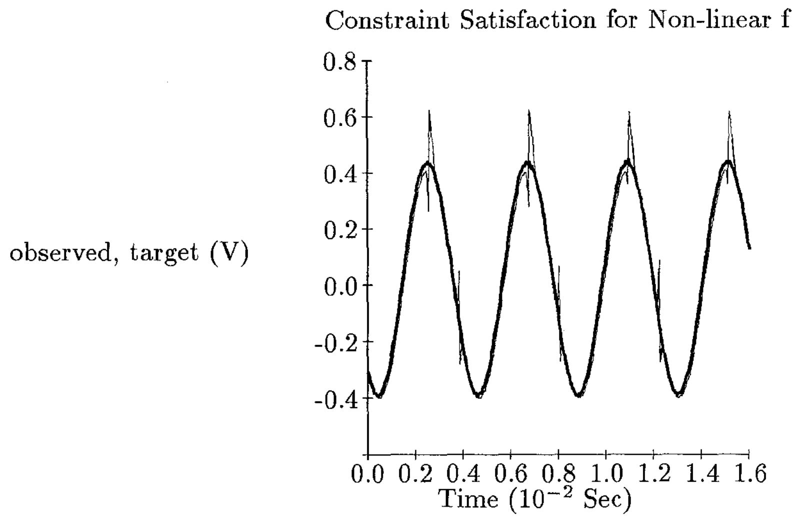


Figure 3.7. Constraint fulfillment for a non-linear optimization function. The plot consists of the two input voltages of the transconductance amplifier. The constraint is fulfilled when the two voltages are the same. As the constraint changes with time, the constrained minimum changes abruptly. After the abrupt change, the constraint is temporarily not fulfilled. However, the circuit quickly fulfills the constraint. The temporary violation of the constraint causes the transient spikes in the figure.

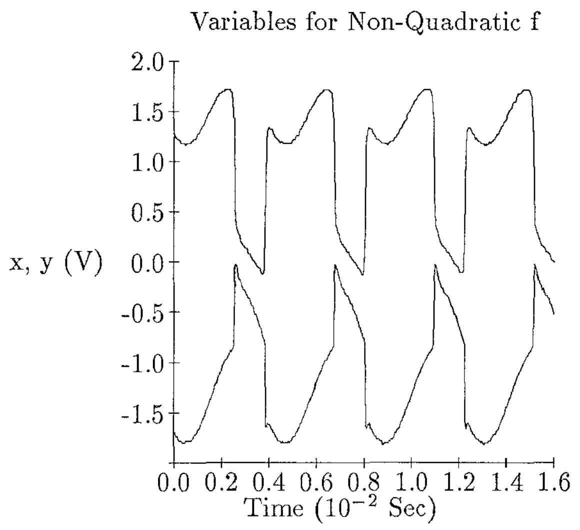


Figure 3.8. Plot of the two input voltages U_1 and U_2 of the inverters. When the output voltages V_i change state, the input voltages change abruptly.

Chapter 4 : Analog Decoding Using Neural Networks

4.1 Introduction: Error Correcting Codes

When information is to be transmitted or stored under noisy circumstances, error correction codes provide a means to retain the information faithfully. Error correction codes add redundancy to information so that the information survives the noise.

This chapter describes some error correction codes which can be effectively decoded by a relatively simple constrained optimization [Platt & Hopfield]. These codes are used to compare the differential multiplier method to the penalty method in the context of a real problem.

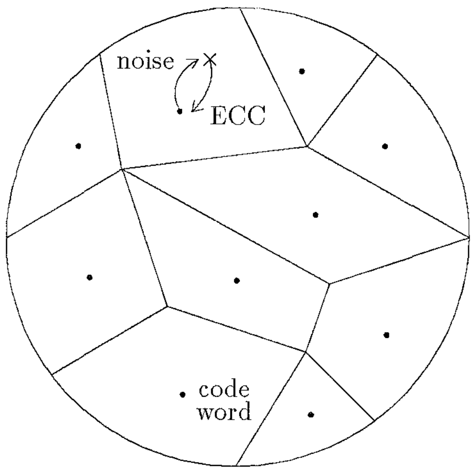


Figure 4.1. An error-correcting code (ECC) converts received codewords into correctly sent codewords

The general idea of simple error correction schemes can be most easily seen in the digital case. We consider a space of N-bit binary words. The states of this space are the corners of an N-dimensional hypercube. The Hamming distance between two words in this space (the number of bits in which they are different) will be taken as a metric of closeness of two states. A set of $C = 2^M$ codewords are chosen in this space, with $M < N$ (M need not be an integer) such that no two codewords are very close together. The entire N-dimensional space is now mapped onto these codewords by associating a particular point in the space with the codeword which is nearest to it. This mapping divides the space into C regions. (see figure 4.1).

Information storage or transmission is accomplished by storing or transmitting a selected codeword. The true information in such a codeword is M bits. If in transmission or recall a few bit errors are generated, the resulting word will still lie near the codeword from which it was transmuted, and be in the region assigned to that codeword. Thus, decoding is the process where words near a codeword are mapped to that codeword. While the complexities of good codes and coding schemes are almost unbounded, this simple idea is at the heart of all error correction.

A desirable code is one which suppresses errors as well as possible when codes are finite, does not require an impossible quantity of storage, and can be decoded with a reasonable amount of computation. If the system is such that either a bit error is made or it is not (e.g., a soft fail in ROM), this is the only coding problem. In the case of the transmission of weak signals in the presence of noise, however, the whole problem is truly an analog problem. A transmitted 1 or 0 actually corresponds to the level of an analog signal being at the level of a nominal 1 or a nominal zero. Without noise, 1's and 0's could be accurately recovered by a threshold circuit with its threshold level set halfway between the voltage level of a nominal 1 and a nominal 0. With additive Gaussian noise, however, there is a probability of the actual voltage when a 0 is sent being greater than threshold, or of a 1 being less than threshold. Thus, errors may be made in the identification of 1's and 0's.

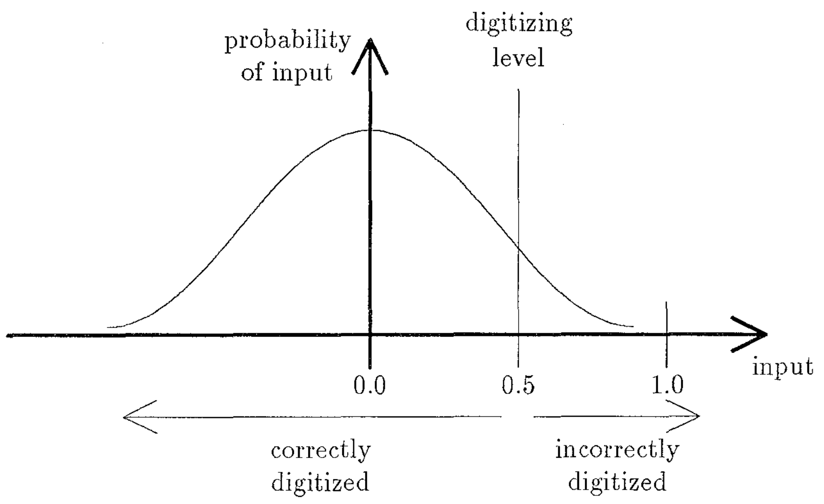


Figure 4.2. Immediate digitization adds errors to the signal

The simplest way to deal with the situation is to digitize the bits as they arrive, then use a binary error-correcting code. Unfortunately, this method of approach is far from optimal. Considering the figure above, we see that an input which is above the discrimination level should almost certainly correspond to a 1, while an input very near but above the threshold level has a sizable probability of corresponding to a 0, though it is slightly more likely to be a 1. If the signal is immediately digitized at the receiver, the distinction between almost certain bits and bits which are "iffy" is lost (see figure 4.2). With a Gaussian channel, this loss of information corresponds to a loss of approximately 2 dB, or 40 percent, of the signal power, compared to a scheme which keeps this information [McEliece]. Real decoders used in deep-space communications use "soft digitization," that is, instead of digitizing to either zero or 1, the input signal is represented by a few bits. The code used in deep-space communications uses 3 bits of digitization [Yuen].

To make optimal use of the available signal energy, it is necessary to use a code which permits initial analog decoding, effectively making a decision about an entire analog code word at a time. This paper describes some such codes which can be initially decoded by "neural" networks. Any real encoding-decoding system would use a standard digital encoding as well, and would follow the analog decoding by a further digital decoding procedure [Yuen].

This problem is of interest for two reasons. First, it exemplifies the design of a network of N neurons with fixed connections, and having far more than N stable states, to solve a real problem. (Having more than N stable states violates no information theorem, since the actual stable states are highly correlated, and thus contain much less than N bits of information per state.) Though the codes described are not as good as existing codes, better codes may also be decodable by "neural" networks.

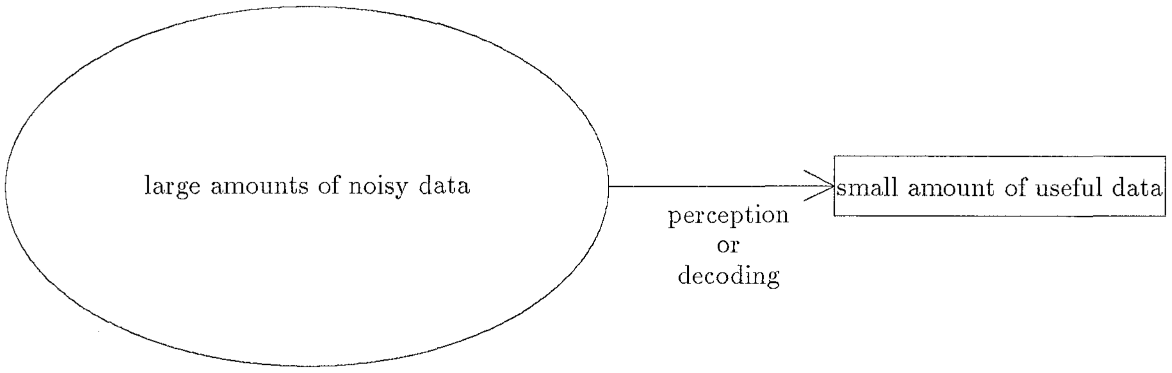


Figure 4.3. Decoding is like perception

Second, decoding is a stylized example of the typical problem in perception (see figure 4.3).

Consider the problem of pronouncing isolated words. To represent actual sound of speech in a word takes roughly 10^5 bits. However, a person typically knows tens of thousands of words, which can be represented as approximately 14 bits. Thus, speech recognition is really the distillation of 14 bits out of 10^5 bits. The 14 bits of information are corrupted by a noisy channel because no one pronounces words in exactly the same way and there is always background noise. So, a robust speech recognition system may use algorithms similar to analog decoding.

The process of perception can be viewed as a contraction of a huge state space down into the much smaller space of essential information. While we do not do the decoding in this extremely difficult case, vision clearly is "merely" a decoding problem, and seeing how to decode even in highly stylized and abstract codes should be of some use in thinking about the computational problem of perception.

4.2 Error-Correcting Codes for Neural Networks

4.2.1 1 in N Code

A 1 in N code is an example of a code that can be decoded by a neural network.

$$\begin{array}{cccccl}
0.2 & 0.4 & 0.8 & 0.7 & 0.1 & \text{Received noisy word} \\
\Downarrow & \Downarrow & \Downarrow & \Downarrow & \Downarrow & \\
0 & 0 & 1 & 0 & 0 & \text{Corrected codeword}
\end{array}$$

Figure 4.4. 1 in N code

We consider using frequency-shift-keying (FSK), where the sender puts energy into N closely spaced frequencies. For a 1 in N code, the sender only puts energy into one of the frequencies at any time. In the decoding process, a "1" is assigned to the one of the N frequencies in which the largest signal is found (see figure 4.4).

This decoding process can be done by a "neural" circuit made as an N-flop, with the input from frequency j connected to neuron j. Thus, the input for the decoder is fed to the neural network in parallel. The neurons in the N-flop compete, the neuron with the largest input wins, and shuts off all of the other neurons.

4.2.2 Permutation Matrix Code

An interesting and more complex code can be based on a 1 in N position coding. Consider a codeword that is N symbols long. The symbols are chosen from an alphabet that has N symbols. Now, let each symbol in the alphabet appear in the codeword exactly once. Thus, an N by N permutation matrix describes which symbol goes into which position in the codeword. In this matrix, there is only one 1 per row, and one 1 per column. Thus, the permutation matrix code can be viewed as an extension of a 1 in N code, where N old codewords are grouped into one new codeword.

$$\underset{\text{Noisy received word}}{\begin{pmatrix} 0.7 & 0.3 & 0.1 & 0.5 \\ 0.4 & 0.8 & 0.3 & 0.2 \\ 0.2 & 0.4 & 0.2 & 1.1 \\ 0.5 & 0.2 & 0.9 & 0.4 \end{pmatrix}} \Longrightarrow \underset{\text{Corrected codeword}}{\begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{pmatrix}}$$

Figure 4.5. Permutation matrix code

There are $N!$ possible N by N permutation matrices, so the information in such a matrix is $\log_2 N!$ bits. Without the restriction to be a permutation matrix, the information would be N^2 bits.

Again, FSK is a practical implementation of the permutation matrix code. Using FSK, each symbol in the alphabet is represented by a different frequency. N corresponding bandpass filters are listening to the channel. Thus, the output of bandpass filter i at time j can formed into a matrix I_{ij}. This matrix nominally looks like a permutation matrix V_{ij}, and the decoding procedure is to find the permutation matrix closest to the input.

The decoding can be performed by extending the neural network that decoded the 1 in N code. To enforce one on per row and one on per column, there should be simultaneous mutual inhibition along the rows and columns.

Consider a typical code word: the identity matrix. If, because of noise, one of the off-diagonal inputs is large, two neurons on the diagonal will cooperate to shut off the neuron corresponding to the spurious input. Thus, the code should be immune to single symbol errors.

4.3 Two Neural Networks that Perform Decoding

Let us examine neural networks for decoding the permutation matrix code. Let I_{ij} be the input for the neuron in the ith row and the jth column. Let V_{ij} be the state of that neuron. The neural network must find the closest permutation matrix V_{ij} to I_{ij}.

Let us express the problem of finding the closest permutation matrix as a constrained optimization problem. Minimize a measure of how close V_{ij} is to I_{ij},

$$\mathcal{E} = -\sum_{i,j} V_{ij} I_{ij}, \tag{4.1}$$

given the constraint that the final output of the decoder should be a digital permutation matrix, i. e.,

$$\sum_i V_{ij} = 1, \qquad \sum_j V_{ij} = 1, \qquad V_{ij}(1 - V_{ij}) = 0. \tag{4.2}$$

Equations (4.1) and (4.2) comprise an integer programming problem. Integer programming is usually quite difficult. However, the permutation matrix coding problem is equivalent to weighted bipartite graph matching (WBGM). The task in WBGM is to assign "workers" i to tasks j. Every worker should be assigned exactly one task. Also, there is an efficiency I_{ij} that each worker does a task. WBGM consists of finding a digital permutation matrix V_{ij} that assigns workers to tasks, such that the function in equation (4.1) is minimized.

WBGM can be shown to be solved with the following constraints.

$$\sum_i V_{ij} = 1, \qquad \sum_j V_{ij} = 1, \qquad 0 \leq V_{ij} \leq 1. \tag{4.3}$$

Since the quantity that is to be minimized is linear and the constraints are linear, WBGM can be solved by linear programming. The solution to the WBGM linear constraints can be shown to always fulfill $V_{ij} \in \{0, 1\}$ [Papadimiriou & Steiglitz].

Thus, to decode the permutation matrix code, one can use the simplex algorithm [Gill, et al.]. To decode the code quickly, one can develop analog hardware governed by neural network differential equations.

4.3.1 Penalty Method

The penalty method constructs the first neural network that solves the constrained optimization problem described in equations (4.1) and (4.2). Using quadratic penalty terms, the constrained optimization is converted into an unconstrained optimization of

$$\mathcal{E} = -\sum i, j V_{ij} I_{ij} + \frac{c_1}{2}\left(\sum_i V_{ij} - 1\right)^2 + \frac{c_2}{2}\left(\sum_j V_{ij} - 1\right)^2 + \frac{c_3}{2}\left[V_{ij}(1 - V_{ij})\right]^2. \tag{4.4}$$

The network slides down this optimization function. The first term is merely the quantity that needs to be minimized. The penalty terms enforce the constraints by creating wells in the state space. As the penalty strengths get larger, the system performs better, but infinite c_i are not physically realizable. In the results described below, $c_i = 10$

The equation of motion corresponding to optimization function in equation (4.4) is

$$\tau \dot{V}_{ij} = I_{ij} - c_1 \left(\sum_i V_{ij} - 1 \right) - c_2 \left(\sum_j V_{ij} - 1 \right) - c_3(1 - 2V_{ij})V_{ij}(1 - V_{ij}). \tag{4.5}$$

Notice that the terms corresponding to the permutation matrix constraints are merely row and column inhibition.

4.3.2 Differential Multiplier Method

The second neural network is constructed with the differential multiplier method. Using the constrained optimization described in equations (4.1) and (4.2), the MDMM from chapter 2 creates the differential equations for the neural network:

$$\begin{aligned}
\frac{dV_{ij}}{dt} =& I_{ij} - \lambda_j - c\left(\sum_i V_{ij} - 1 \right) \\
& - \sigma_i - c\left(\sum_j V_{ij} - 1 \right) \\
& - \left[\omega_{ij} + kV_{ij}(1 - V_{ij})\right](1 - 2V_{ij}), \\
\frac{d\lambda_j}{dt} =& \sum_i V_{ij} - 1, \\
\frac{d\sigma_i}{dt} =& \sum_j V_{ij} - 1, \\
\frac{d\omega_{ij}}{dt} =& V_{ij}(1 - V_{ij}).
\end{aligned} \tag{4.6}$$

In the results described below, the constants c and k were set to 0.2. A wide range of c and k work well, however.

4.4 Results

The optimization in equation (4.1) is not quadratic, it is linear. In addition, the last constraint in equation (4.2) is non-linear. Using the BDMM to solve equations (4.1) and (4.2) results in undamped oscillations. To converge onto a constrained minimum, the MDMM is used. For both a 5×5 and a 20×20 system, a $c = 0.2$ is adequate for damping the oscillations. The choice of c seems to be reasonably insensitive to the size of the system, and a wide range of c, from 0.02 to 2.0, damps the oscillations.

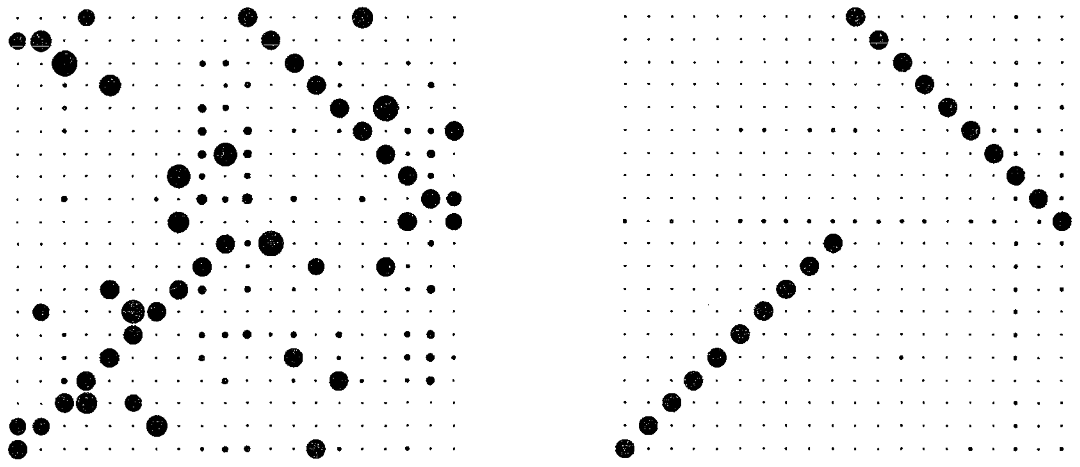


Figure 4.6. The decoder finds the nearest permutation matrix

In a test of the MDMM, a signal matrix which is a permutation matrix plus some noise, with a signal-to-noise ratio of 4, is supplied to the network. In the left half of figure 4.6, the system starts with the correct neurons and many incorrect neurons turned on. In the right half of figure 4.6, the constraints start to be applied, and eventually the system reaches a permutation matrix. The differential equations do not need to be reset. If a new signal matrix is applied to the network, the neural state will move towards the new solution.

The permutation matrix code was simulated with a Gaussian channel. Gaussian noise was added to the input of each neuron, and the various decoder implementations were simulated. The performance of the decoder is shown in a graph of the probability of decoding to the wrong code word versus the signal-to-noise power ratio.

The bit signal-to-noise ratio, E_b/N_0, is commonly used for signal-to-noise comparisons. E_b is the energy sent per information bit and N_0 is the noise power [McEliece]. Now,

$$E_b = P/R, \tag{4.7}$$

where P is the average signal power and R is the information rate of the code, in bits per time. If the signal sent down the channel is S_{ij}, then the average signal power is

$$P = \frac{1}{T}\sum_{i,j} S_{ij}^2, \tag{4.8}$$

where T is the total amount of time it takes to send a code word. The information rate of the code is

$$R = \frac{1}{T} n, \tag{4.9}$$

where n is the number of information bits in a code word. Therefore,

$$E_b = \frac{1}{n} \sum_{i,j} S_{ij}^2. \tag{4.10}$$

The noise power is related to the standard deviation σ of the Gaussian noise in the channel:

$$N_0 = 2\sigma^2. \tag{4.11}$$

Therefore, the bit signal-to-noise ratio is

$$\frac{E_b}{N_0} = \frac{\sum_{i,j} S_{ij}^2}{2\sigma^2 n}. \tag{4.12}$$

For an 8×8 permutation matrix code, $n = 15$, and

$$\frac{E_b}{N_0} = \frac{8}{30\sigma^2}. \tag{4.13}$$

Shannon's Source-Channel Coding Theorem states that one can transmit over a wideband Gaussian channel with arbitrarily low error probability, as long as $E_b/N_0 > \ln 2$.

The probabilities of incorrectly decoding a code word versus E_b/N_0 are plotted for various decoder implementations in figure 4.7.

As can be seen in figure 4.7, the penalty method performs poorly. Not only is the error rate high, but often the system does not even settle in a permutation matrix. The error rate of the differential multiplier method is close to the theoretical limit of the code. Furthermore, when the c_i in equation (4.4) are below 5, the penalty method does not yield any sensible permutation matrices. In order for the penalty method to work at all, the differential equations become stiff and take four times as much computer time than the differential multiplier method. The differential multiplier method is the better method for analog decoding.

4.5 Conclusions

The analog decoding of error-correcting codes is a direct application of constrained differential optimization discussed in the last chapter. The performance of a decoding algorithm can be measured quantitatively, yet decoding is a stylized version of perception. The quantitative superiority of the differential multiplier method over the penalty method might carry over to other tasks dealing with large amounts of inaccurate data.

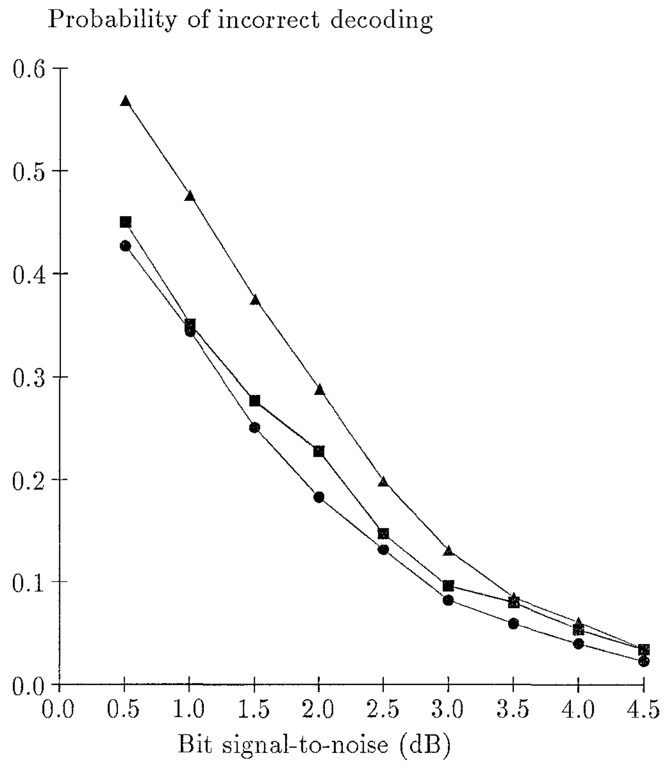


Figure 4.7. Performance of various neural decoders on the permutation matrix code: The penalty method performance is shown with triangles, the differential multiplier method performance is shown with squares, and the theoretical limit of the code is shown with circles.

Chapter 5 : Constrained Optimizing Splines

5.1 Introduction: What are Constrained Optimizing Splines?

In order for neural networks to be easily built in analog VLSI, the connections between neurons need to be fairly limited. Neural network chips with every neuron connected to every other, with arbitrary connections, currently have fewer than 100 neurons on a chip [Sivilotti, et al. 86]. However, when neurons are locally connected with constant strength connections, hundreds, or even thousands, of neurons can be placed on a chip [Sivilotti, et al. 87][Mead]. This chapter explores constrained neural networks whose optimization functions can be computed locally.

One example of an optimization that can be locally computed is an *optimizing spline.* A spline is a smooth manifold that is embedded in a space that smooths or approximates data in that space. An optimizing spline minimizes a measure of the smoothness of the manifold. The measure is a functional, which takes a manifold as a parameter and yields a number.

Splines are useful for many different tasks. [Kohonen] uses a spline representation to learn an input probability density. Optimizing splines can be used to remove noise in perceptual problems [Poggio & Torre]. Splines can also be used to represent geometry. Optimizing splines find a shortest path in the traveling salesman problem [Durbin & Willshaw]. Optimizing two-dimensional splines can perform surface reconstruction [Koch, et al.][Terzopoulos, et al.].

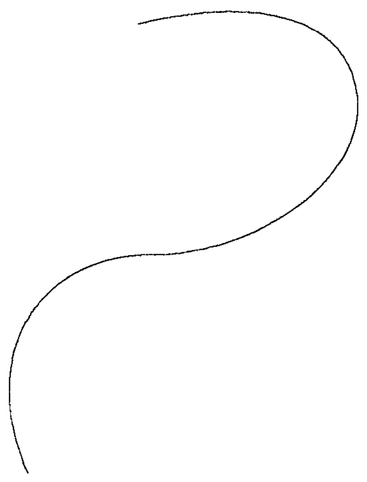


Figure 5.1. A snake is a one-dimensional manifold

The simplest example of an optimizing spline is a *snake* [Kass, et al.], which is a one-dimensional manifold that is embedded in a plane (see figure 5.1). The position of the snake is described by two functions $x(s)$ and $y(s)$, where s is a parameter that increases along the snake.

There are two popular functionals that are used to govern the behavior of a snake. One, called a membrane functional, measures the arc length of the snake [Courant & Hilbert]:

$$M(x(s), y(s)) = \int_a^b (x'(s))^2 + (y'(s))^2 . ds \tag{5.1}$$

The functional in equation (5.1) is called a membrane functional because it governs the behavior of a rubber membrane whose rest shape is a point of zero size. Another, called a thin plate functional, measures the roughness of the snake [Courant & Hilbert]:

$$P(x(s), y(s)) = \int_a^b (x''(s))^2 + (y''(s))^2 . ds \tag{5.2}$$

The functional in equation (5.2) is called a thin plate functional because it is an approximation to the energy that governs a thin plate of stiff material whose rest shape is uncurved. These functionals are linearizations of the functionals that govern elastic objects (see chapter 6).

The behavior of a snake can be governed with a neural network by discretizing the governing functional of the snake [Kass, et al.][Burr]. The snake is then an array of neural outputs (x_i, y_i) which represent the position of the snake. When the functional in equation (5.1) is discretized, the resulting optimization function is

$$\mathcal{E}_{\text{membrane}} = \sum_{i=0}^{N-1} (x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2 . \tag{5.3}$$

When the functional in equation (5.2) is discretized, the resulting optimization function is

$$\mathcal{E}_{\text{thin plate}} = \sum_{i=1}^{N-1} (x_{i+1} - 2x_i + x_{i-1})^2 + (y_{i+1} - 2y_i + y_{i-1})^2 . \tag{5.4}$$

Additional constraints may be placed on the optimization function by either the penalty method for approximate constraints, or by the differential multiplier method for exact constraints.

5.2 Constrained Snakes for Computer Vision

An example of constrained optimizing splines is a set of snakes that find a symmetric pair of edges in an image. In real images, the edges of objects or textured regions are frequently obscured by noise or other objects. There are many possible edges which might yield a particular noisy image. Following [Poggio & Torre], we want to regularize this ill-posed problem. In other words, we want to find the smoothest edge that is consistent with the image data [Kass, et al.].

Since we don't know the exact location of the edge in the image, we use the penalty method to constrain the snake to lie on the edge. The optimization function that encourages a snake to lie on edge is

$$\mathcal{E}_{\text{edge detection}} = \sum_{i=1}^{N-1} (x_{i+1} - 2x_i + x_{i-1})^2 + (y_{i+1} - 2y_i + y_{i-1})^2 - \sum_{i=0}^{N} ||\nabla I(x_i, y_i) * G||^2, \qquad (5.5)$$

where $I(x,y)$ is the image intensity at location (x,y), and G is a Gaussian of some suitable width.

Frequently, we have extra information about the objects in the image that allows us to find edges more effectively. One example is that objects in images frequently have axes of symmetry, especially when they are objects used in manufacturing. Therefore, we can look for symmetric edges in an image, which would be a stronger indication that we have found the edges of an object, instead of random discontinuities in the image.

A triplet of snakes is needed to find symmetric edges in an image: two to find edges in the image, and one to form a smooth axis of symmetry. The three snakes are defined by (x_i^r, y_i^r), (x_i^c, y_i^c), (x_i^l, y_i^l). There are thin plate terms governing each of the three snakes. In addition, there are edge penalty terms applied to the r snake and the l snake. The three snakes are linked with a penalty term which encourages the central snake c to lie halfway between the r snake and the l snake:

$$\mathcal{E}_{\text{symmetry}} = \sum_i (x_i^r + x_i^l - 2x_i^c)^2 + (y_i^r + y_i^l - 2y_i^c)^2. \qquad (5.6)$$

The symmetry is enforced with a penalty term because the symmetry might not be exact.

An example of a symmetric snake is shown in figure 5.2. The symmetry-seeking snake inspired a three-dimensional version of a symmetry-seeking optimizing manifold described in [Terzopoulos, et al.].

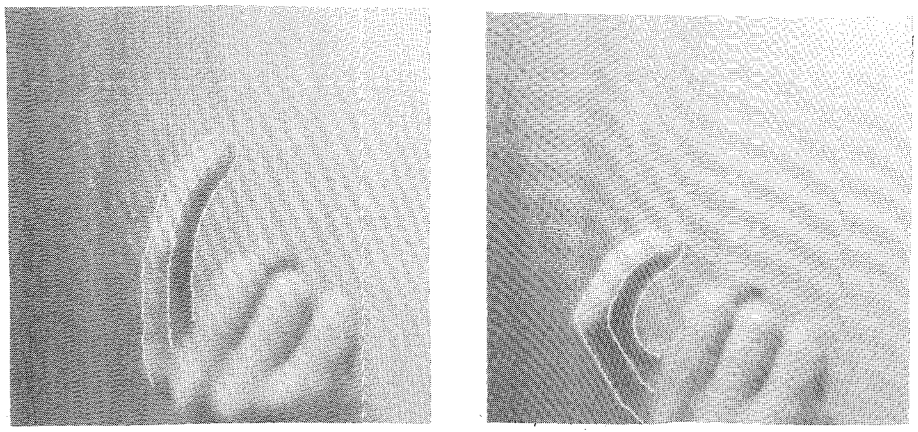


Figure 5.2. A symmetry-seeking snake found a finger

5.3 The Traveling Salesman Problem

A second example of a constrained optimizing spline is a snake that finds an approximate solution to the traveling salesman problem (TSP). The traveling salesman problem is an example of a difficult combinatorial optimization problem. The solutions to a combinatorial optimization problem are restricted to a discrete set of values [Papadimitriou & Steiglitz]. Combinatorial optimization problems arise in many design situations, such as printed circuit board layout. Finding the best answer to a combinatorial optimization problem is often very difficult. For example, the traveling salesman problem is NP-complete [Johnson & Papadimitriou]. However, researchers have come up with heuristics that approximately solve particular combinatorial optimization problems. Frequently, finding the global optimum of a combinatorial optimization problem is unnecessary: a very good answer suffices.

Neural networks are supposed to be good at approximate reasoning and heuristics. [Hopfield & Tank] used neural networks to approximately solve the traveling salesman problem. The traveling salesman problem consists of finding the shortest path that goes through each of N cities exactly once. Notice that an answer to the TSP has two properties: it is necessary that the path goes through all of the cities and it is desirable that the path be short. When the TSP is solved approximately, the path must still go through all of the cities, but the path length can be low, instead of globally minimum. The combination of necessary and desirable properties indicate that constrained neural networks should be good at approximately solving the TSP.

Following [Durbin & Willshaw] we use an optimizing snake to approximately solve the planar traveling salesman problem, where all of the cities lie on a plane. The snake represents the tour that the salesman travels. Since the tour should be as short as possible, the snake minimizes its length

$$\mathcal{E}_{\text{snake}} = \sum_i (x_{i+1} - x_i)^2 - (y_{i+1} - y_i)^2. \tag{5.7}$$

Since the tour must visit each city exactly once, the snake should be constrained to lie on all of the cities. More precisely, each city should force the nearest point on the snake to lie on the city:

$$k(x^* - x_c) = 0, \qquad k(y^* - y_c) = 0, \tag{5.8}$$

where (x^*, y^*) are city coordinates, (x_c, y_c) is the closest snake point to the city, and k is the constraint strength.

The minimization in equation (5.7) is quadratic and the constraints in equation (5.8) are piecewise linear, corresponding to a C^0 continuous Lyapunov function from Appendix B. Thus, the damping is positive definite, and the system converges to a state in which the constraints are fulfilled.

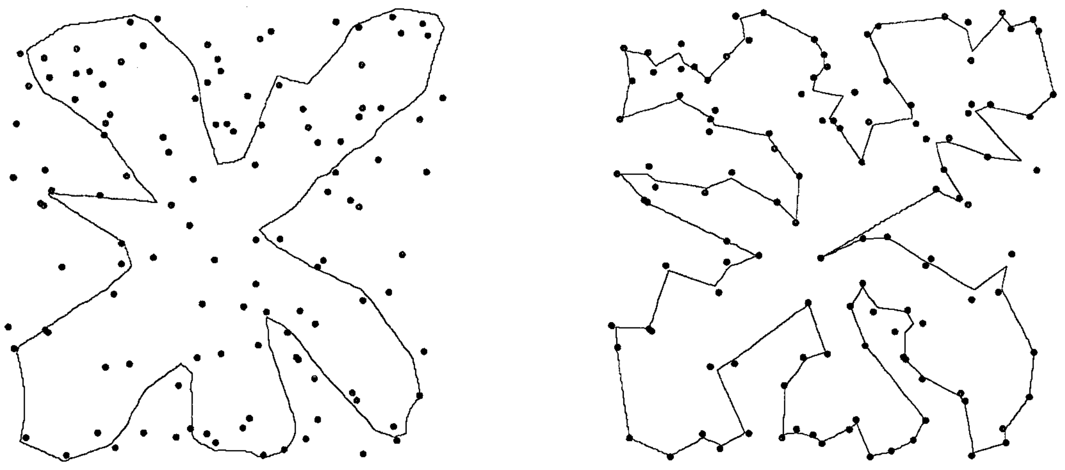


Figure 5.3. The snake is attaching to the cities

In practice, the snake starts out as a square surrounding the cities. Groups of cities grab onto the snake, deforming it. As the snake gets close to groups of cities, it grabs onto a specific ordering of cities that locally minimize its length (see figure 5.3).

The system of differential equations that solve equations (5.7) and (5.8) are piecewise linear. The differential equations for x_i and y_i are solved with implicit Euler's method, using tridiagonal LU decomposition and the Woodbury formula to solve the linear system [Press, et al.]. The points of the snake are sorted into bins that partition the plane, so that the computation of finding the nearest point is simplified.

The constrained minimization in equations (5.7) and (5.8) is a reasonably effective method for approximately solving the TSP. For 120 cities distributed in the unit square, and 600 snake points, a numerical step size of 100 time units, and a constraint strength of 5×10^{-3}, the tour lengths are $5\% \pm 2\%$ longer than that yielded by simulated annealing [Kirkpatrick, et al.]. This performance is competitive with other one-pass greedy algorithms that are commonly used to solve the Planar TSP [Golden & Stewart]. Empirically, for 30 to 240 cities, the time needed to compute the final city ordering scales as $N^{1.6}$, as compared to the Kernighan-Lin method [Lin & Kernighan] which scales roughly as $N^{2.2}$. The computation can be sped up by waiting until the city ordering has settled, instead of waiting until the snake has fully attached to the cities.

The constraint strength is usable for both a 30 city problem and a 240 city problem. Although changing the constraint strength affects the performance, the snake attaches to the cities for any non-zero constraint strength. Parameter adjustment does not seem to be an issue as the number of cities increases, unlike the penalty method.

5.4 Conclusions

This chapter described constrained optimizing splines which perform useful computational tasks, such as finding symmetric edges in an image or finding a good approximate solution to the traveling salesman problem. The constraints on optimizing splines can be enforced with either the penalty method or with the differential multiplier method.

Constrained Computer Graphics: Theory

Chapter 6 : Deformable Physically-Based Models

6.1 Introduction: Dynamic vs. Kinematic Models

Methods to formulate and represent instantaneous shapes of objects are central to computer graphics modeling. Such methods have been particularly successful for modeling purely geometric rigid objects whose shapes do not change over time. This thesis develops an approach to modeling which incorporates the physically-based dynamics of flexible materials into the purely geometric models which have been used traditionally. Chapter 6 proposes models based on elasticity theory which conveniently represent the shape and motion of deformable materials. Models based on elasticity can interact easily with other physically-based computer graphics model. These models are developed for easy-to-create computer graphics animation.

Most traditional methods for computer graphics modeling are kinematic; that is, the shapes are compositions of geometrically or algebraically defined primitives. Kinematic models are passive because they do not interact with each other or with external forces. The models are either stationary or are subjected to motion according to prescribed trajectories. Expertise is required to use kinematic models to create natural and pleasing dynamics.

In addition, kinematic models are difficult to control and constrain. Because the shape and motion of kinematic models are described with algebraic equations, the constraint methods must solve algebraic equations, which can be either under-constrained or over-constrained. Constraints on kinematic models might not look physically realistic, because the constraints are not enforced with forces.

Dynamic models are an alternative to kinematic models in computer graphics. Dynamic models are based on principles of mathematical physics [Courant & Hilbert]. They react to applied forces (such as gravity), to constraints (such as linkages), to ambient media (such as viscous fluids), or to impenetrable obstacles (such as supporting surfaces) as one might expect real, physical objects to react. Dynamic models can be active by generating forces by themselves, in order to look more "alive".

This chapter develops models of deformable solids which are based on elasticity theory. The static shapes of a wide range of deformable objects, including string, rubber, cloth, paper, and flexible metals can be created by simulating physical properties such as tension and rigidity. Furthermore, the dynamics of these objects can be simulated by including physical properties such as mass and damping. The simulation involves numerically solving the partial differential equations that govern the evolving shape of the deformable model and its motion through space.

The dynamic behavior inherent to deformable models in principle significantly simplifies the animation of complex objects. Consider the graphical representation of a flag. The traditional approach has been to represent the instantaneous shape of the flag as a mesh assembly of bicubic spline patches or polygons. Making the flag move plausibly with this representation is a nontrivial task. In contrast, deformable models can provide a physical representation of the flag which exhibits natural dynamics as it is subjected to external forces and constraints.

The dynamics of deformable models can be simulated reasonably quickly on a digital computer. The computation of the motion of the Jell-o shown later in this chapter took only six seconds per frame on the Symbolics LISP Machine. With either faster digital or analog hardware on the horizon, we envision deformable models running in real time as part of a synthetic reality.

Physically-based models are compatible with and complementary to the constraint-based modeling approach for rigid primitives proposed by [Barzel & Barr] as well as with the dynamics-based approaches of [Wilhelms & Barsky] and [Armstrong & Green], which animate articulated rigid bodies.

6.1.1 Outline

The remainder of the chapter develops as follows: Section 6.2 gives differential equations of motion describing the dynamic behavior of deformable models under the influence of external forces. Section 6.3 contains an analysis of deformation and defines deformation energies for solid models. Section 6.4 lists various external forces that can be applied to deformable models to produce animation. Section 6.5 describes a spring/mass approximation of deformable models. Section 6.6 presents some simulations illustrating the deformable models.

6.1.2 Notation

The notation used in chapters 6 and 7 is slightly nonstandard. A vector is denoted by a symbol with an underscore: $\underline{A}$. A matrix is denoted with two underscores: $\underline{\underline{A}}$.

6.1.3 Coordinate Systems

We must define the coordinate system that will be used throughout chapters 6 through 8.

Let $\underline{a}$ be the intrinsic or material coordinates of a point in a body Ω. For a solid body, $\underline{a}$

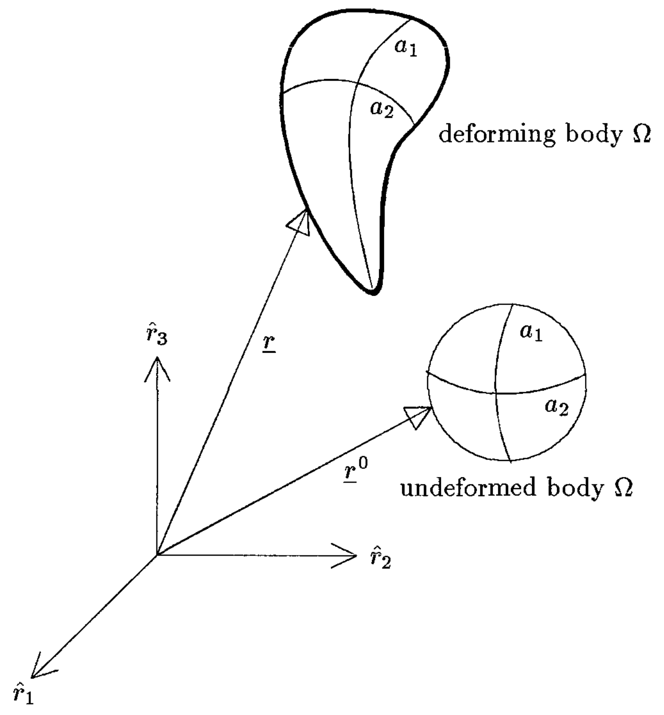


Figure 6.1. Coordinate systems used in this chapter.

has three components: $[a_1, a_2, a_3]$. Similarly, for a surface $\underline{a} = [a_1, a_2]$, and a curve $\underline{a} = [a_1]$. The Euclidean 3-space positions of points in the body are given by a time-varying vector valued function of the material coordinates $\underline{r}(\underline{a}, t) = [r_1(\underline{a}, t), r_2(\underline{a}, t), r_3(\underline{a}, t)]$. The body in its natural rest state (see figure 6.1) is specified by $\underline{r}^0(\underline{a}) = [r_1^0(\underline{a}), r_2^0(\underline{a}), r_3^0(\underline{a})]$.

6.2 Dynamics of Deformable Models

This section explains the equations of motion governing the dynamics of deformable models under the influence of applied forces. The equations of motion are obtained from Newtonian mechanics and balance the externally applied forces with the forces due to the deformable model. The equations of motion of a system can be derived from Newton's second law:

$$\underline{F} = m\underline{a}, \tag{6.1}$$

where $\underline{F}$ is the net force on a mass point with mass m, and $\underline{a}$ is the acceleration of the point.

The dynamics of a deformable model can be understood as a generalization of the dynamics of a spring-mass system.

6.2.1 The Dynamics of a Spring-Mass System

Consider the position $x(t)$ of a mass attached to a spring, a dashpot, and a time-dependent external force. Equation (6.1) turns into

$$m\frac{d^2x}{dt^2} = F_{\text{spring}} + F_{\text{dashpot}} + F_{\text{external}}. \tag{6.2}$$

If the spring exerts a linear force restoring the mass to $x = 0$ and the dashpot exerts a force proportional to the velocity, then the equation of motion is

$$m\frac{d^2x}{dt^2} = -kx - \gamma\frac{dx}{dt} + F_{\text{external}}(t), \tag{6.3}$$

or

$$m\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} + kx = F_{\text{external}}(t). \tag{6.4}$$

Where does the kx force term come from? There is a potential energy associated with the spring:

$$U_{\text{spring}} = \frac{1}{2}kx^2. \tag{6.5}$$

Thus, the spring is in a low energy state ("happy") when x is near 0, and is in a high energy state ("unhappy") when x is far away from zero. F_{spring} will try to make the system be in a low energy state. In fact, $F_{\text{spring}}(x)$ is related to $U_{\text{spring}}(x)$ through

$$F_{\text{spring}}(x) = -\frac{dU_{\text{spring}}}{dx}(x). \tag{6.6}$$

Thus, a more general equation which balances all of the forces on $x(t)$ is

$$m\frac{d^2x}{dt^2} + \gamma\frac{dx}{dt} + \frac{dU_{\text{spring}}}{dx}(x) = F_{\text{external}}(t). \tag{6.7}$$

6.2.2 The Dynamics of an Elastic Model

Now, the equations of motion for an elastic model are just a generalization for the equation for a spring. Each point in the model will obey Newton's laws. However, the equation for the potential energy is more complicated. Even in the more complex case, each term corresponds to a force. The acceleration is proportional to the sum of the forces. The equations governing a deformable model's motion can be written in Lagrange's form [Goldstein] as follows:

$$\frac{\partial}{\partial t}\left(\rho\frac{\partial \underline{r}}{\partial t}\right) + \frac{\delta\mathcal{D}(d\underline{r}/dt)}{\delta(d\underline{r}/dt)} + \frac{\delta\mathcal{E}(\underline{r})}{\delta\underline{r}} = \underline{f}(\underline{r},t), \tag{6.8}$$

where $\underline{r}(\underline{a},t)$ is the position of the particle $\underline{a}$ at time t, $\rho(\underline{a})$ is the mass density of the body at $\underline{a}$, and $\underline{f}(\underline{r},t)$ represents the net externally applied forces. $\mathcal{E}(\underline{r})$ is the net instantaneous potential energy of the elastic deformation of the body. $\mathcal{D}(d\underline{r}/dt)$ is the net instantaneous dissipation of the body.

In the elastic model, the internal elastic potential energy $\mathcal{E}(\underline{r}(\underline{a}))$ is an integral of a function over the entire body. The energy takes the shape of the body (a function $\underline{r}(\underline{a})$) and returns a number (the energy). Thus, $\mathcal{E}(\underline{r})$ is a *functional.* The force is the derivative of this functional, $\delta\mathcal{E}(\underline{r})/\delta\underline{r}$. Similarly, $\mathcal{D}(d\underline{r}/dt)$ is a functional, and the force $\frac{\delta\mathcal{D}(d\underline{r}/dt)}{\delta(d\underline{r}/dt)}$ is a variational derivative. However, standard calculus can only take derivatives of functions. In order to take derivatives of functionals, one must use the *Calculus of Variations* [Courant & Hilbert][Gelfand & Fomin] (see Appendix C).

The external forces are balanced against the force terms on the left hand side of (6.8) due to the deformable model. The first term is the inertial force due to the model's distributed mass. The second term is the damping force due to dissipation. The third term is the elastic force due to the deformation of the model away from its natural shape.

6.3 Elastic Materials

This section develops potential energies of deformation $\mathcal{E}(\underline{r})$ associated with the elastically deformable models. These energies are functionals that define the internal elastic forces of the models.

6.3.1 Analysis of Deformation

Elasticity theory involves the analysis of deformation [Landau & Lifschitz][Fung]. We will define measures of deformation using concepts from differential geometry[Docarmo]. One requirement of our present approach is that the measures should be insensitive to rigid body motion since it imparts no deformation.

The shape of a body is determined by the Euclidean distances between nearby points. The *metric tensor* relates distances in material coordinate to these Euclidean distances. As the body deforms, the metric tensor changes as a function of time.

Let $\underline{a}$ and $\underline{a} + d\underline{a}$ be the material coordinates of two nearby points in the body. The distance between these points in the deformed body in Euclidean 3-space is given by

$$dl = \sum_{i,j} G_{ij}\, da_i da_j, \tag{6.9}$$

where the symmetric matrix

$$G_{ij}\left(\underline{r}(\underline{a})\right) = \frac{\partial \underline{r}}{\partial a_i} \cdot \frac{\partial \underline{r}}{\partial a_j} \tag{6.10}$$

is the metric tensor or the first fundamental form [Faux & Pratt] (the dot indicates the scalar product of two vectors).

Two three-dimensional solids have the same shape (differ only by a rigid body motion) if their 3×3 metric tensors are identical functions of $\underline{a} = [a_1, a_2, a_3]$. However, this no longer need be true when the body becomes infinitesimally thin in one or more of its dimensions.

In the rest of the chapter, the fundamental forms associated with the natural shapes of deformable bodies will be denoted by the superscript 0. Thus,

$$G^0_{ij} = \frac{\partial \underline{r}^0}{\partial a_i} \cdot \frac{\partial \underline{r}^0}{\partial a_j}. \tag{6.11}$$

The *strain tensor* measures the amount of deformation of a body. The strain tensor is the continuum version of the displacement of a spring. We can define a strain tensor E_{ij} which describes the difference between the current rest tensor of the deformed body G_{ij} and a metric tensor of the natural, undeformed body, G^0_{ij}. In general, the strain tensor is a function $E(G, G^0)$. For Hookean elasticity, however, we use

$$E_{ij} = G_{ij} - G^0_{ij}.$$

A strain energy of a deformable solid is a norm of the strain tensor integrated over the body. When the strain tensor is small, then the strain energy is small. Similarly, when the strain is large in some or all of the body, then the strain energy is large (see figure 6.2). We use a strain energy for a deformable solid that is

$$\mathcal{E}_{\text{solid}}(\underline{r}) = \int_{\Omega} ||\underline{\underline{E}}||^2_{\alpha}\, da_1 da_2 da_3. \tag{6.12}$$

The deformation energy (6.12) is zero for rigid motions, and includes the fewest partial derivatives necessary to restore the natural shapes of solids. However, higher-order derivatives can be included to further constrain the smoothness of the admissible deformations of these bodies [Terzopoulos, et al.]. To simulate a deformable surface or curve, simply simulate the non-zero thickness of the curve or surface.

The form of matrix norm used in the energies above defines the behavior of the model. For example, Hooke's law states that the force on a object is linear in the displacement from a rest state.

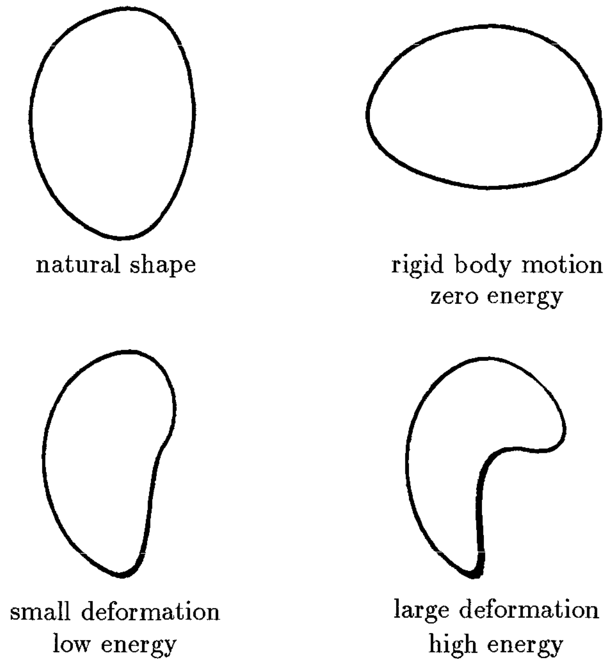


Figure 6.2. Energy of deformation depends on amount of deformation

A body which obeys Hooke's law would have an energy of the form

$$\mathcal{E}_{\text{Hookean}}(\underline{r}) = \int_{\Omega} \sum_{i,j,k,l} C_{ijkl} E_{ij} E_{kl} \, da_1 da_2 da_3. \tag{6.13}$$

Notice that in this case, as in the spring, the energy is quadratic in the displacement. The tensor C_{ijkl} specifies the material property of the body Ω. Since any anti-symmetric component of C_{ijkl} does not contribute to the energy, C_{ijkl} is symmetric by convention. Also, since E_{ij} is a symmetric tensor, the number of independent parameters in C_{ijkl} is further reduced to 21 [Fung].

When the elastic properties of a material are the same in all directions, the number of independent parameters in C_{ijkl} is reduced to two: the *modulus of rigidity* μ and *Poisson's ratio* ν [Fung]. The modulus of rigidity measures the resistance of the material to stretching or squashing. If Poisson's ratio is positive, then when the material is stretched in one direction, it will shrink in the other directions. C_{ijkl} can also be parameterized by the *Lamé constants*, λ and μ. [Fung]. λ is related to μ and ν by

$$\lambda = \frac{2\mu\nu}{1 - 2\nu}. \tag{6.14}$$

Notice that the Lamé constant λ is not the same as a Lagrange multiplier from chapter 2. A physically realistic material will have

$$\mu > 0, \qquad \lambda > -\frac{2}{3}\mu. \tag{6.15}$$

The energy of an isotropic Hookean elastic solid is

$$\mathcal{E}_{\text{isotropic}}(\underline{r}) = \mathcal{E}_\mu + \mathcal{E}_\lambda = \int_\Omega \mu \sum_{i,j} (E_{ij})^2 + \frac{1}{2}\lambda \left(\sum_i E_{ii} \right)^2 d\underline{a}. \tag{6.16}$$

6.3.2 Elastic Force for an Isotropic Solid

The elastic force for an isotropic solid can be derived by taking the variational derivative of (6.16). Consider the first term on the right-hand side of (6.16). Substituting the definition of the strain tensor E_{ij} and the metric tensor G_{ij} yields

$$\mathcal{E}_\mu(\underline{r}) = \mu \sum_{i,j} (\frac{\partial \underline{r}}{\partial a_i} \cdot \frac{\partial \underline{r}}{\partial a_j} - G_{ij}^0)^2. \tag{6.17}$$

The Calculus of Variation and the symmetry of the rest-state metric tensor G_{ij}^0 yields

$$\frac{\delta \mathcal{E}_\mu(\underline{r})}{\delta \underline{r}} = -4\mu \sum_i \frac{\partial}{\partial a_i} \left[\sum_j (\frac{\partial \underline{r}}{\partial a_i} \cdot \frac{\partial \underline{r}}{\partial a_j} - G_{ij}^0) \frac{\partial \underline{r}}{\partial a_j} \right]. \tag{6.18}$$

Equation (6.18) is the force that keeps a model from stretching. If the model is discretized into a grid of mass points, then the terms when $i = j$ encourage the points to be a fixed distance from their neighbors. The $i \neq j$ terms encourage the angles between the grid lines to be roughly fixed.

The second term in (6.16) is

$$\mathcal{E}_\lambda(\underline{r}) = \frac{1}{2}\lambda \left[\sum_i (\frac{\partial \underline{r}}{\partial a_i} \cdot \frac{\partial \underline{r}}{\partial a_i} - G_{ii}^0) \right]^2. \tag{6.19}$$

Again, applying the Calculus of Variation yields

$$\frac{\delta \mathcal{E}_\lambda(\underline{r})}{\delta \underline{r}} = -2\lambda \sum_i \frac{\partial}{\partial a_i} \left[\sum_k (\frac{\partial \underline{r}}{\partial a_k} \cdot \frac{\partial \underline{r}}{\partial a_k} - G_{kk}^0) \frac{\partial \underline{r}}{\partial a_i} \right]. \tag{6.20}$$

The elastic force term for an isotropic solid is merely the sum of equation (6.16) and (6.20). Remember that the true force is the negative of the variational derivative (see equation (6.8)). Also, for a surface and a curve, this force only constrains the metric tensor — it does not constrain the other fundamental forms (i.e, curvature, torsion).

6.4 Applied Forces on Deformable Models

Applying external forces to elastic models yields realistic dynamics. This section shows examples of external forces, such as the effects of gravity and fluids. The net external force $\underline{f}(\underline{r}, t)$ in equation (6.8) is the sum of the individual external forces.

A gravitational force acting on the deformable body is given by

$$\underline{f}_{\text{gravity}} = \rho(\underline{a})\underline{g}, \tag{6.21}$$

where $\rho(\underline{a})$ is the mass density and $\underline{g}$ is the gravitational field.

A force that connects a material point $\underline{a}_0$ to a point in world coordinates $\underline{r}_0 = [x_0, y_0, z_0]$ by an ideal Hookean spring is

$$\underline{f}_{\text{spring}} = k(\underline{r}_0 - \underline{r}(\underline{a}_0)), \tag{6.22}$$

where k is the spring constant.

The force on the surface of a body due to a viscous fluid is

$$\underline{f}_{\text{viscous}} = c(\underline{n} \cdot \underline{v})\underline{n}, \tag{6.23}$$

where c is the strength of the fluid force, $\underline{n}(\underline{a})$ is the unit normal on the surface, and $\underline{v}$ is the velocity of the surface relative to some constant stream velocity $\underline{u}$:

$$\underline{v}(\underline{a}, t) = \underline{u} - \frac{\partial \underline{r}(\underline{a}, t)}{\partial t}. \tag{6.24}$$

The force is a flow field projected onto the normal of the surface, is linear in the velocity, and models a viscous medium [Barr].

A force due to a non-viscous slow wind force is [Haumann]

$$\underline{F}_{\text{wind}} = c(\underline{n} \cdot \underline{v})\underline{v}. \tag{6.25}$$

Notice that this force is nonlinear in $\underline{v}$, but still goes to 0 when the velocity is tangential to $\underline{n}$.

The vibrations in a flexible model eventually damp out due to viscous forces. Viscous forces can be derived from *a Rayleigh dissipation function* (see glossary) which discourages changes in the metric tensor

$$\mathcal{D} = \int \frac{c}{2} \sum_{i,j} \frac{dG_{ij}}{dt} \frac{dG_{ij}}{dt} dt.$$

The first variation of the Rayleigh dissipation function with respect to the velocity is the viscous force

$$F_k^{\text{viscous}} = -c \sum_{i,j,l} \frac{\partial G_{ij}}{\partial r_k} \frac{\partial G_{ij}}{\partial r_l} \frac{dr_l}{dt}. \tag{6.26}$$

Using the force described above is equivalent to the Voigt model of viscoelasticity, where the rest shape of the model remains unchanged and the viscous force resists changes in the current shape of the model[Fung].

6.5 Approximations of Deformable Models

The variable $\underline{r}(\underline{a}, t)$ in the equation (6.8) is a continuous function of both material coordinates and time. In order to make computer animation of flexible models, the material coordinates and time are usually discretized, because there usually is not an analytic solution to the partial differential equation (6.8). When the material coordinates are discretized, a flexible model becomes a set of mass points or mass elements. When time is discretized, the animation is computed at sequential time frames.

Numerical analysis supplies numerous discretization algorithms for general use [White]. Appendix A describes some discretization algorithms that are applicable to the animation of flexible models.

One simple approximation to the differential equation (6.8) involves connecting mass points with linear springs. The model is discretized into a grid of mass points. A pattern of springs connects these mass points and encourages the model to attain its rest shape.

An examination of the metric tensor of a shape indicates how springs should be used. First, consider the G_{00} component of the metric tensor:

$$G_{00} = \frac{\partial \underline{r}}{\partial a_0} \cdot \frac{\partial \underline{r}}{\partial a_0}. \tag{6.27}$$

If the model is discretized into a set of mass points, the derivatives in equation (6.27) can be expressed as differences:

$$G_{00} \approx \frac{\Delta \underline{r}_0}{\Delta a_0} \cdot \frac{\Delta \underline{r}_0}{\Delta a_0}. \tag{6.28}$$

where $\Delta \underline{r}_0$ is the vector between two mass points in lab coordinates, and Δa_0 is the difference in the first material coordinate between the two mass points.

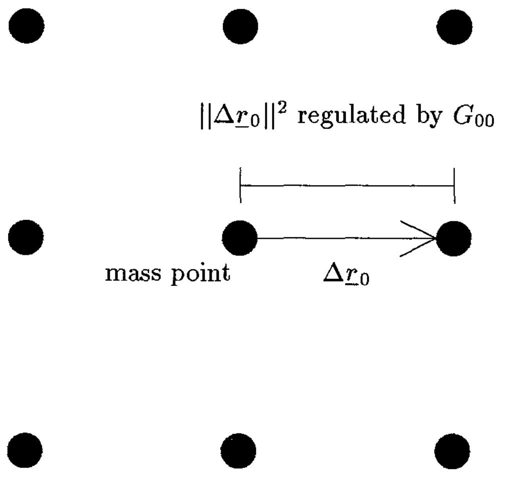


Figure 6.3. Diagonal components of the metric tensor regulate the length of the vector $\Delta \underline{r}_0$ that points from one mass point to another

The geometric meaning of equation (6.28) is explained in figure 6.3. The metric tensor component G_{00} is proportional to the square of the distance between the two mass points. Since the strain is merely the difference $E_{00} = G_{00} - G^0_{00}$, then the elastic forces should encourage the distance between mass points in the deformed model to be the same as the model at rest.

Now, consider the G_{01} component of the metric tensor:

$$G_{01} = \frac{\partial \underline{r}}{\partial a_0} \cdot \frac{\partial \underline{r}}{\partial a_1} \approx \frac{\Delta \underline{r}_0}{\Delta a_0} \cdot \frac{\Delta \underline{r}_1}{\Delta a_1}. \tag{6.29}$$

where $\Delta \underline{r}_0$ connect two mass points that differ in the first material coordinate by Δa_0, and $\Delta \underline{r}_1$ connect two mass points that differ in the second material coordinate by Δa_1.

The geometric meaning of equation (6.29) is explained in figure 6.4. The metric tensor component G_{01} is proportional to the dot product of the two vectors $\Delta \underline{r}_0$ and $\Delta \underline{r}_1$. Since the distance of the vectors are regulated by the diagonal terms, the elastic forces due to the off-diagonal term should encourage the angles between mass points to be the same in the deformed model and in the model at rest.

We can use springs to create an approximation to the energy in equation (6.12). Each spring has an associated quadratic energy. If springs are connected between the mass point in particular patterns, then they form a quadratic energy which encourages distances and angles to lie near those values in the rest shape.

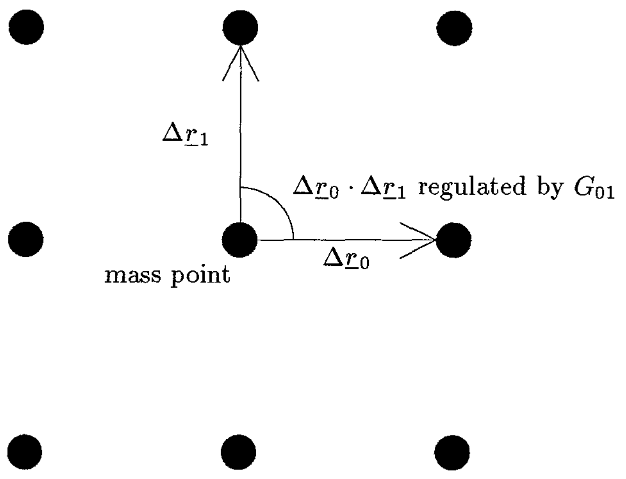

$\Delta \underline{r}_1$
$\Delta \underline{r}_0 \cdot \Delta \underline{r}_1$ regulated by G_{01}
mass point
$\Delta \underline{r}_0$

Figure 6.4. Off-diagonal components of the metric tensor regulate the angle between mass points by regulating the dot product of the vectors $\underline{\Delta} r_0$ and $\underline{\Delta} r_1$

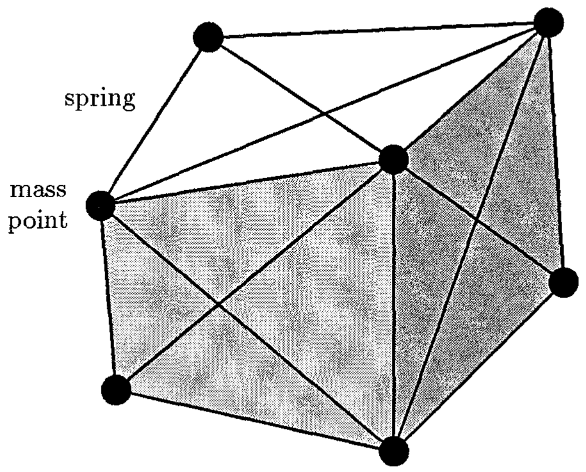


Figure 6.5. Springs in three faces of a three-dimensional hexahedral elastic solid element

A solid model is discretized into hexahedra. In each hexahedron, create a spring along each edge and two springs diagonally on each face (see figure 6.5). The model requires at least one diagonal per face, or the hexahedron may collapse while maintaining the same elastic energy. A surface model is discretized into a mesh of grid points. Springs are connected as shown in figure 6.6. The pattern in figure 6.6 not only regulates distances and angles, but also regulates the curvature of the surface.

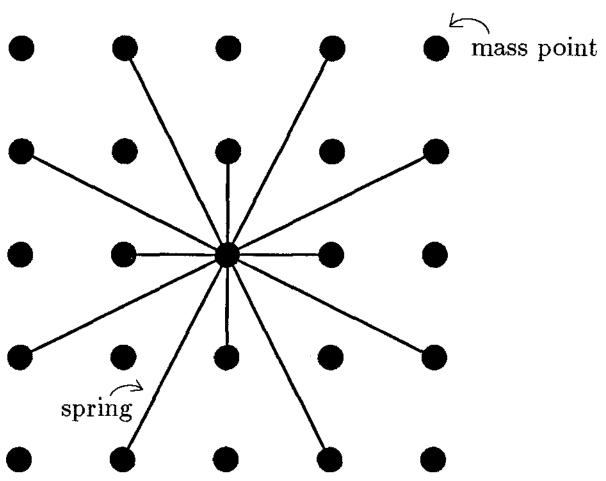


Figure 6.6. A pattern of springs connecting mass points in an elastic surface element

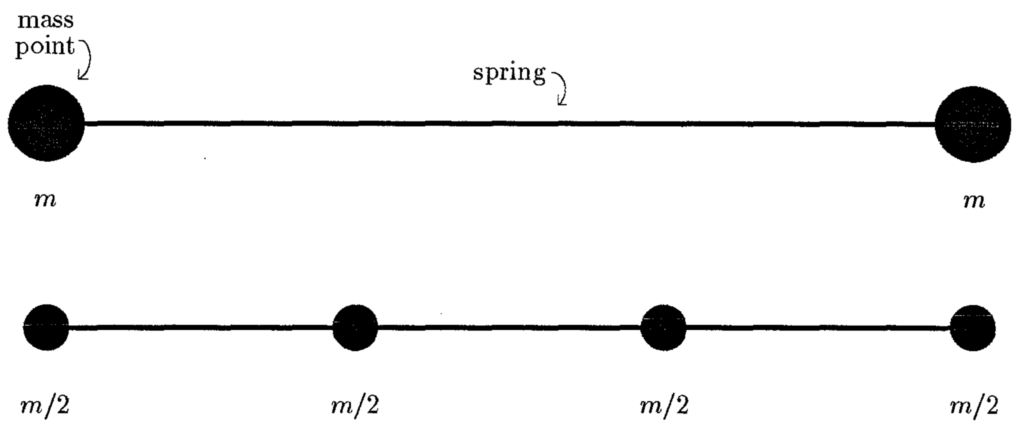


Figure 6.7. Different discretizations of a spring-mass system

The strengths of the springs must be scaled correctly, in order for the models to be automatically discretized with different degrees of fineness. Consider two mass points with a spring between them. Let the rest distance of the spring be l_0 and consider the potential energy of the system when the spring is stretched to l:

$$\mathcal{E}_1 = c_1(l - l_0)^2. \tag{6.30}$$

where c_1 is the spring constant. Now, discretize the system into N mass points and $N-1$ springs (see figure 6.7). The potential energy is now

$$\mathcal{E}_N = c_N \sum_{i=1}^{N} (\frac{l}{N} - \frac{l_0}{N})^2. \tag{6.31}$$

In order for the energy in equation (6.30) to be the same as the energy in equation (6.31), the scaling of the spring constants should be

$$c_N = c_1 N. \tag{6.32}$$

In other words, the spring strength should be inversely proportional to the rest length of the spring. In other words, the energy for a spring should be

$$\mathcal{E}_{\text{spring}} = \frac{c}{l_0}(l - l_0)^2. \tag{6.33}$$

Using the spring energy in the equation (6.33) allows spring-mass models to be discretized at any desired scale. The force corresponding to the energy in equation (6.33) is

$$\underline{F}_{\text{spring}} = \frac{c(l-l_0)}{l l_0}\underline{r} = c\left(\frac{1}{l_0} - \frac{1}{l}\right)\underline{r}, \tag{6.34}$$

where $\underline{r}$ is the vector from one mass point to the other.

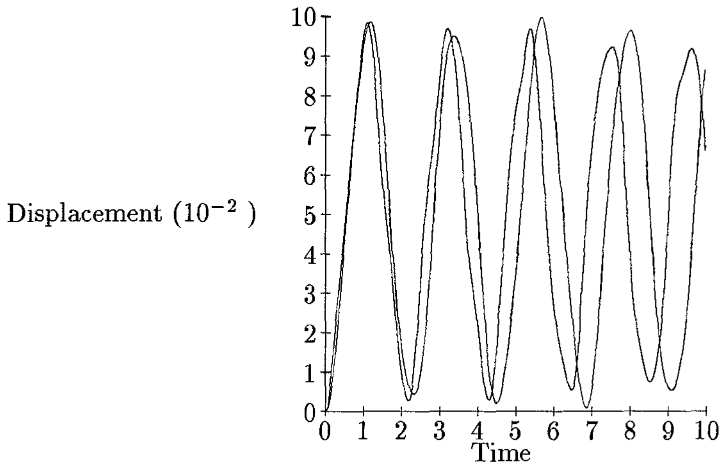


Figure 6.8. Position of endpoints of two differently discretized models

To demonstrate that the scaling of equation (6.33) works, two differently discretized spring mass systems were simulated: one with five mass points and the other with ten. All of the mass points were connected in a row. The total mass of each model is identical, and the models were stretched identically. The position of the endpoints of both models are plotted in figure 6.8. The amplitude and the frequency of the oscillation match within a few percent. The difference is caused by the extra modes that are in the ten-mass system. However, the scaling works well enough so that the discretization of a model can be changed without having to adjust the spring constraints.

6.6 Simulation Tests of Deformable Models

The following simulations have been selected to convey the broad scope of elastically deformable models. The figures appear at the end of the chapter.

Figures 6.9 through 6.11 illustrate a simulation of a flag waving in the wind. The metric tensor of the flag material is regulated. The flag tends to return to its rectangular rest shape. The flag is immersed in a constant wind vector field. The effect of the wind on the flag is modeled by the viscous force (6.23). The flag is fixed to a rigid flagpole along one of its edges by imposing a fixed-position (Dirichlet) boundary condition [Lapidus]. The animation sequence was made by Kurt Fleischer, using elastically deformable models that I helped to create in [Terzopoulos, et al.].

Figures 6.12 through 6.14 shows a simulation of a carpet falling onto two rigid objects. Algorithms for colliding flexible models with rigid models are described in chapters 7 and 8. The carpet animation is an example of simulation using differential equations that are first-order in time. These differential equations can be used to place models in desired positions, but they cannot simulate the natural oscillations of flexible objects. The animation sequence was also made by Kurt Fleischer.

Figures 6.15 and 6.16 show a cube of jello vibrating on a table. These figures show a full simulation of a flexible solid, using the potential energy from equation (6.16). More examples of simulated flexible solids are shown in chapter 8.

6.7 Conclusion

This chapter has proposed a class of elastically deformable models for use in computer graphics. Our goal has been to create models that inherit the essential features of elastic materials, while still maintaining computational tractability. Because our models are physically-based, they are dynamic: they respond to external forces and interact with other objects in a natural way. Our models yield realistic dynamics in addition to realistic statics; they unify the description of shape and motion. We therefore believe that physically-based modeling will prove to be an increasingly powerful technique for computer graphics animation.

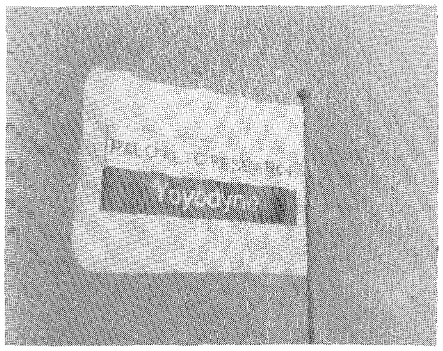


Figure 6.9. A simulation of a flag flapping in the wind

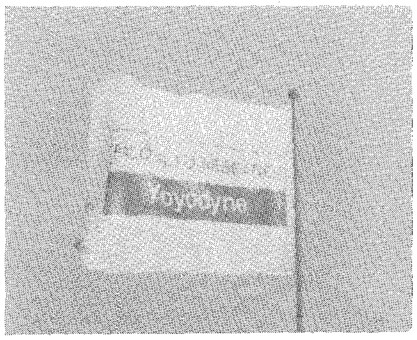


Figure 6.10. The flag later in time

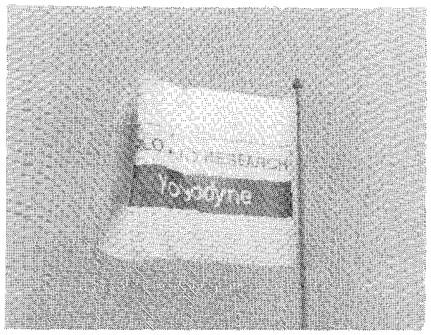


Figure 6.11. The flag even later in time

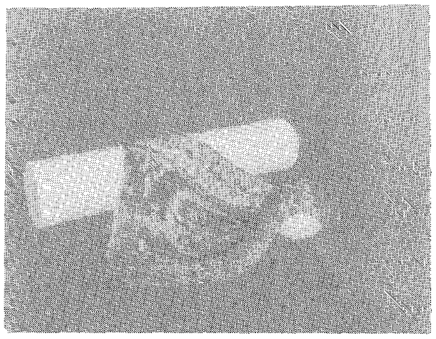


Figure 6.12. A simulation of a carpet falling on a sphere and a cylinder

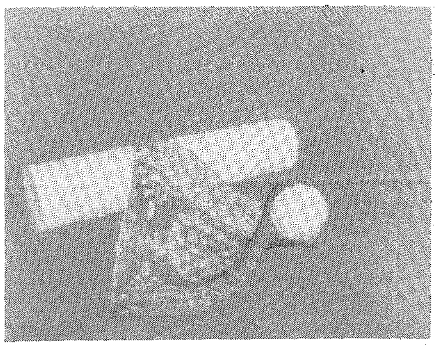


Figure 6.13. The carpet changes shape due to external forces

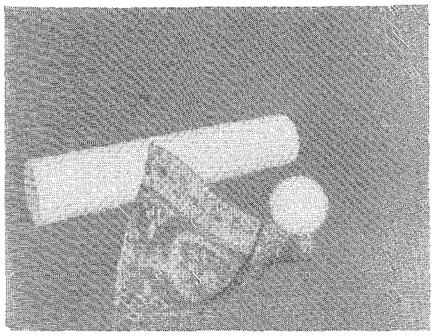


Figure 6.14. The carpet eventually slips between the other objects

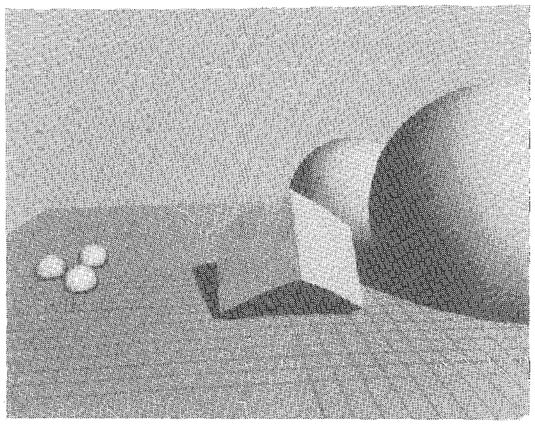


Figure 6.15. A simulation of jello vibrating on the table

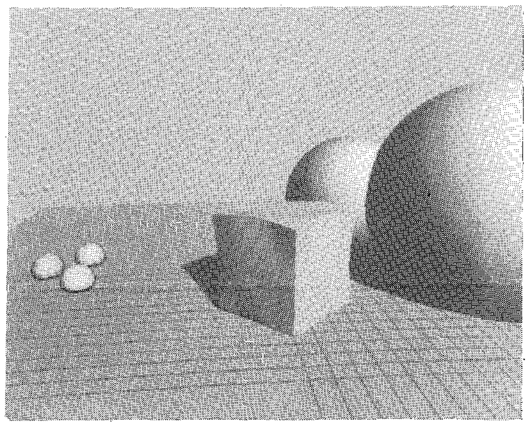


Figure 6.16. The jello eventually returns to its rest shape: a cube

Chapter 7 : Constraint Methods for Physical Systems

7.1 Introduction: Teleological Modeling

Chapter 7 discusses methods that cause physically-based models to fulfill user-specified goals. Teleological modeling, from the Greek word "telos" meaning "goal", is modeling using a new level of representation: modeling based on a set of user-designed goals [Barr].

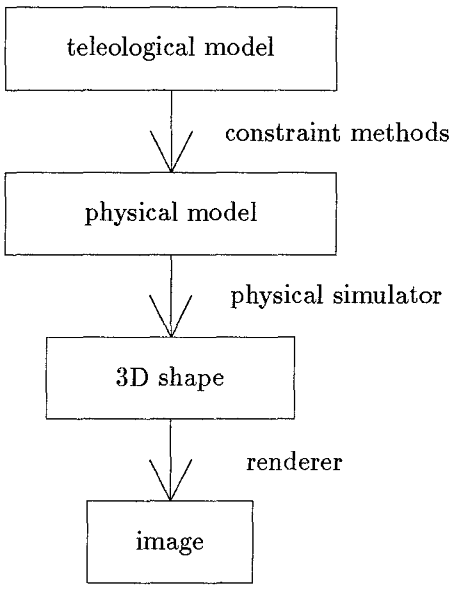


Figure 7.1. Levels of representation of an object

Teleological modeling adds an additional level of representation to physically-based modeling. Figure 7.1 shows various level of representation of an object. The simplest representation is that of a two-dimensional image of a model. This representation is often used in real-time applications, such as graphics editors or video games. The problem with a two-dimensional image is that realistic animation is extraordinarily difficult. A computer can easily move the image around on a screen, but the model cannot appear to rotate or deform.

The next level of representation of an object is a three-dimensional shape. Given a three-dimensional shape, an object can be viewed from all angles, and can appear as if it were moving freely in space. A renderer converts the three-dimensional shape back into an image. However, the motion of the three-dimensional shape must be completely specified laboriously, especially if the motion must be physically realistic.

The third level of representation of an object is a physical model. A physical model has mass, momentum, forces, and torques. A physical simulator takes a set of models in an initial configuration and simulates the laws of physics, automatically producing a time sequence of three-dimensional shapes. Many physicists and engineers use this level of representation. However, many computer applications need motions that fulfill user-specified goals.

The fourth level of representation of an object is a teleological model. A teleological model has a set of goals which constraint methods convert into the desired results, frequently using forces and impulses. These constraint methods are similar to constrained optimization methods, except that the differential equations that describe mechanical systems are second-order in time, instead of the first-order differential equations that describe gradient descent in optimization theory. Both the penalty method and Lagrange multipliers can be used to constrain mechanical systems. In fact, the concept of Lagrange multipliers was first developed to constrain physical systems.

We add force-based constraint methods to our physical simulator in order to cause models to fulfill goals. Force-based constraint methods that add external forces and impulses to physical systems yield physically realistic motion and allow simulation with simple, commercially available, differential equation solvers. Force-based constraint methods create models that are naturally guided by invisible forces or hands. Other constraint methods which arbitrarily manipulate positions and velocities tend to yield unattractive unphysical motion.

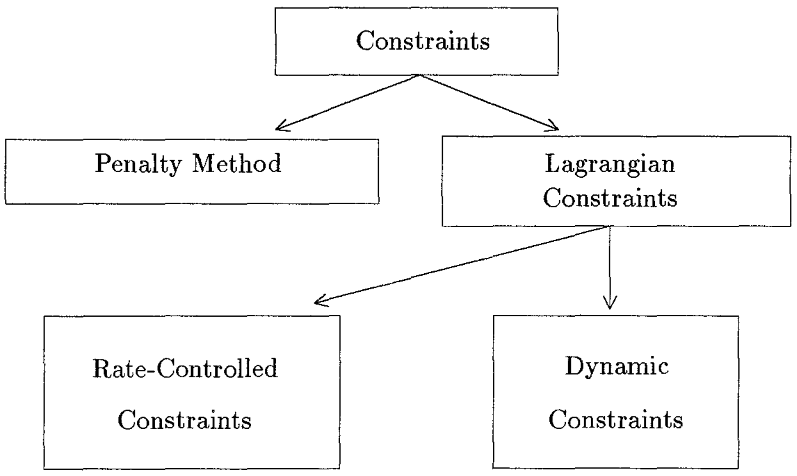


Figure 7.2. A hierarchy of force-based constraint methods

This chapter discusses several force-based constraint methods that allow the creation of flexible models with desirable properties (see figure 7.2).

- *The penalty method* changes the potential energy of a physical system to penalize incorrect behavior.
- *Lagrangian constraints* add additional forces to fulfill constraints exactly. Lagrangian constraints compute Lagrange multipliers either exactly or approximately.

There are at least two Lagrangian constraint methods that are applicable to constraining physically-based computer graphics models.

- *Rate-controlled constraints* computes Lagrange multipliers which force the system to undergo critically damped motion that fulfill the constraints. Rate-controlled constraints are new to the field of computer graphics.
- *Dynamic constraints* are a specialization of rate-controlled constraints. Dynamic constraints also fulfill constraints with critically damped motion, but add forces that are not necessarily in the direction of the constraint. Dynamic constraints are described in [Barzel & Barr87] and are not discussed in this chapter.

7.1.1 Previous Work

The penalty method is a technique from optimization theory [Hestenes] [Gill, et al.]. [Witkin, et al.] and [Terzopoulos, et al.] used the penalty method to assist in the design of computer graphics models.

Classical physics expresses constrained systems using Lagrangian formulations [Goldstein]. Mechanical engineers have used Lagrange multipliers to constrain systems, and, in recent years, have added damped motion towards the constraints in order to stabilize the constraint systems [Nikravesh] [Baumgarte] [Wittenburg] [Boland, et al.]. Chapter 7 exploits the damped motion in order to assemble computer graphics models and to simulate collisions between the computer graphics models.

Mechanical engineers have treated the collision of rigid bodies using impulses [Wittenburg] [Barzel & Barr89]. Chapter 7 applies impulses to the collision of flexible models.

7.2 The Penalty Method

This section discusses applying the penalty method to physical systems. The penalty method is equivalent to adding a rubber band to a mechanical system that attract the physical state to the subspace $g(\underline{x}) = 0$:

$$F_{\text{penalty}} = -2cg(\underline{x})\frac{\partial g}{\partial x_i}, \tag{7.1}$$

where c is the strength of the rubber band. The force in equation (7.1) creates a force field that eventually points towards the manifold $g = 0$, as long as $\partial g/\partial x_i$ is not zero. The system moves in the direction of $g = 0$, but the penalty method is not guaranteed to fulfill $g = 0$.

The penalty method is also equivalent to adding a quadratic energy term to the potential energy of the system that penalizes violations of the constraint [Hestenes]. Analogously to constrained optimization with the penalty method, the potential energy is composed of the unconstrained physical system's potential energy plus the potential energy of the constraint spring:

$$U_{\text{penalty}}(\underline{x}) = U(\underline{x}) + c(g(\underline{x}))^2. \tag{7.2}$$

The differential equations for a physical system constrained by the penalty method is

$$\sum_j m_{ij}\frac{dx_j}{dt} = F(\underline{x}) - 2cg(\underline{x})\frac{\partial g}{\partial x_i}, \tag{7.3}$$

where m_{ij} is the *generalized mass matrix* and $F(\underline{x})$ is the *generalized force* on the system.

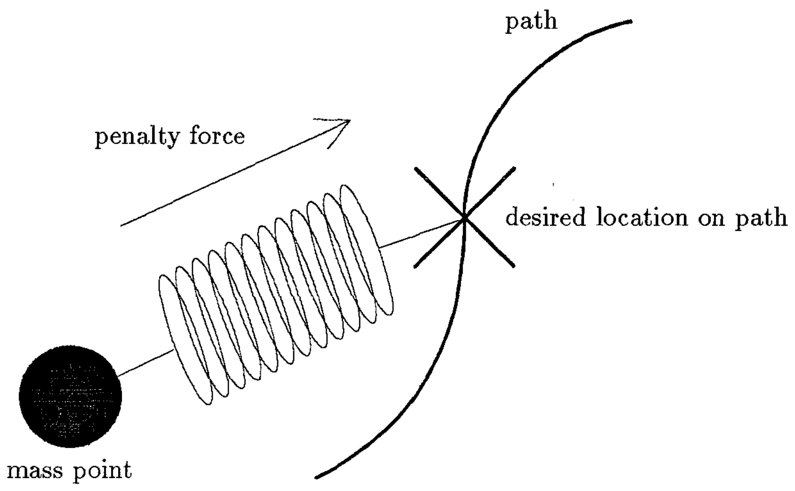


Figure 7.3. A mass point is attracted by a penalty force to a desired location on a path

The penalty method can be applied to inequality constraints of the form $g > 0$ by using a rubber band force that turns on when $g < 0$ (see figure 7.4). The potential energy for the inequality constrained system is

$$U_{\text{inequality}}(\underline{x}) = U(\underline{x}) + \exp(-kg(\underline{x})). \tag{7.4}$$

The corresponding differential equation is

$$\sum_j m_{ij} \frac{dx_j}{dt} = F(\underline{x}) + k \exp(-kg(\underline{x})) \frac{\partial g}{\partial x_i}. \tag{7.5}$$

The penalty method can be extended to fulfill multiple constraints by using more than one rubber band. The potential energy of the system with multiple constraints is

$$U_{\text{penalty}}(\underline{x}) = U(\underline{x}) + \sum_\alpha c_\alpha (g_\alpha(\underline{x}))^2. \tag{7.6}$$

The corresponding differential equation is

$$\sum_j m_{ij} \frac{dx_j}{dt} = F(\underline{x}) - 2 \sum_\alpha c_\alpha g_\alpha(\underline{x}) \frac{\partial g_\alpha}{\partial x_i}, \tag{7.7}$$

where c_α is the spring strength of the αth constraint.

An example of the penalty method is to force a point to follow a path (see figure 7.3). When the spring is connected from the mass point to the desired location, the mass point should move roughly towards that location.

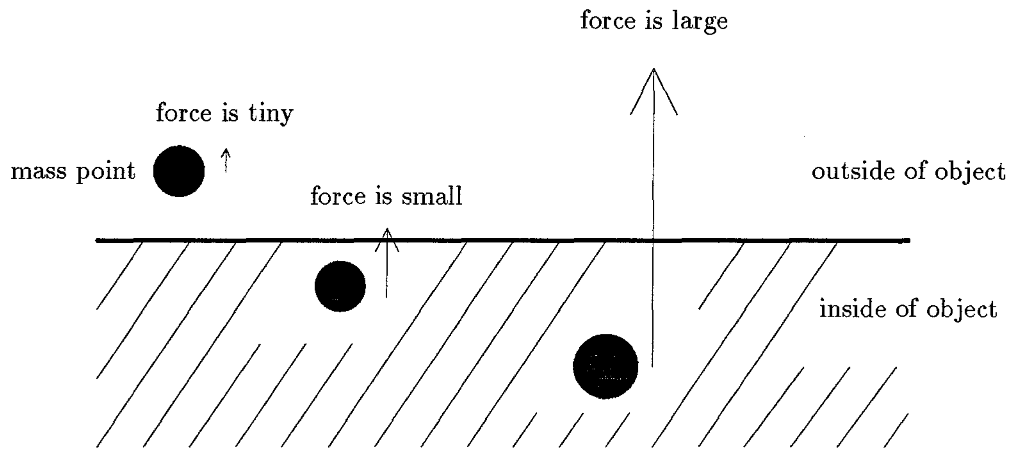


Figure 7.4. A mass point is expelled by a penalty force from the inside of an object

Another example of the penalty method is to expel a mass point from the inside of another object (see figure 7.4). As the mass point penetrates the object, a force starts to expel the mass point. As the mass point penetrates even further into the object, the force increases exponentially.

In summary, let us examine the features of the penalty method. The penalty method has a few convenient features.

- *Inexact Constraints* — There are situations in which it is not necessary to fulfill constraints exactly; sometimes it is desirable to compromise between constraints.
- *Ease of Use* — Adding a rubber band to a physical system is simple and requires no extra differential equations.

However, the penalty method has a number of disadvantages.

- *Inexact Constraints* — For finite constraint strengths c_α, the penalty method does not fulfill the constraints precisely. Under many circumstances, however, constraints should be fulfilled exactly. Using multiple rubber band constraints is like building a machine out of rubber bands; the machine would not hold together perfectly.
- *Stiffness of Equations* — Second, as the penalty strengths increase, the differential equations become *stiff*; that is, there are widely separated time constants. Most numerical methods must take time steps on the order of the fastest time constant, while most modelers are interested in the behavior at the slowest time constant. As a result of stiffness, the numerical differential equation solver takes very small time steps, using a large amount of computing time without getting much done. Also, it is unclear how to increase the penalty strengths as the simulation is proceeding.

7.3 Lagrangian Constraints

Flexible models should fulfill constraints quickly and exactly. As discussed in the last section, the penalty method does not swiftly fulfill precise constraints.

Lagrangian constraints are methods that retain many of the advantages of the penalty method while avoiding many of the disadvantages. The penalty method tries to balance the constraint force with the other forces in the physical system. In the penalty method, the constraint force is a function of system state, therefore the penalty method causes the system to find a particular state that balances the forces. In Lagrangian constraints, forces that would cause the system to violate the constraints are canceled independently of system state, and a force is substituted that gradually makes the system fulfill the constraints. Because of the Lagrangian formulation of physics, Lagrangian constraints are applicable to both mass points and finite elements. Very general constraints can be fulfilled with Lagrangian constraints.

Constraints can be integrated into the variational framework that describes the behavior of physical systems. If the functional

$$\mathcal{L} = \int \left(\frac{1}{2} \sum_{i,j} m_{ij} \frac{dx_i}{dt} \frac{dx_j}{dt} - U(\underline{x}) \right) dt, \tag{7.8}$$

is extremized by a physical system with generalized mass matrix m_{ij} and potential energy $U(\underline{x})$, then an additional term can be added to force the system to fulfill a constraint $g(\underline{x}) = 0$:

$$\mathcal{L} = \int \left(\frac{1}{2} \sum_{i,j} m_{ij} \frac{dx_i}{dt} \frac{dx_j}{dt} - U(\underline{x}) - \lambda g(\underline{x}) \right) dt, \tag{7.9}$$

The variable λ is the *Lagrange multiplier.*

Extremizing the functional in equation (7.9) yields a set of equations that describe a constrained physical system (see Appendix C). Taking the variational derivative with respect to $\underline{x}$, and setting to zero yields the Euler-Lagrange equations

$$\sum_j m_{ij} \frac{d^2 x_j}{dt^2} + \frac{\partial U}{\partial x_i} + \lambda \frac{\partial g}{\partial x_i} = 0. \tag{7.10}$$

Taking the variational derivative with respect to λ yields

$$g(\underline{x}) = 0, \tag{7.11}$$

which is precisely the constraint equation.

The third term in equation (7.10) is an additional force in the direction of $dg/d\underline{x}$, just like the penalty term in the last section. However, the strength of the force is proportional to the Lagrange multiplier λ, which is computed to fulfill the constraint $g(\underline{x}) = 0$ exactly.

Algorithmically, a Lagrangian constraint processes the net force at a point, $\underline{F}_{\text{input}}$ created by physics or other constraint techniques, to yield a constrained force at a point $\underline{F}_{\text{output}}$, needed to fulfill a particular constraint. The Lagrange multiplier first projects out undesirable components of $\underline{F}_{\text{input}}$ to yield $\underline{F}_{\text{unconstrained}}$. Next, $\underline{F}_{\text{constrained}}$ is computed to yield motion that fulfills the constraint. Finally, the control force $\underline{F}_{\text{output}}$ is the sum of the constrained and unconstrained forces:

$$\underline{F}_{\text{output}} = \underline{F}_{\text{constrained}} + \underline{F}_{\text{unconstrained}}. \tag{7.12}$$

An example of a Lagrangian constraint is shown in figures 7.5 and 7.6. The constraint goal in figures 7.5 and 7.6 is to place the mass point on the line. In figure 7.5, there is a force on the mass point pushing it up and to the right. The force pushing up is undesirable, but any horizontal force

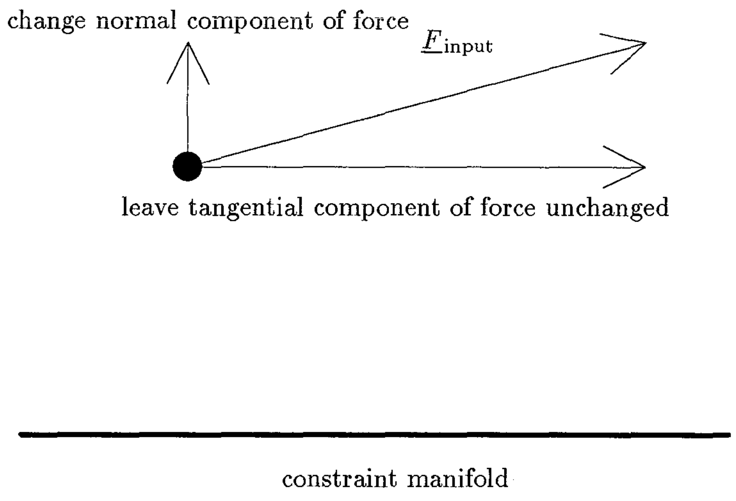

change normal component of force
$\underline{F}_{\text{input}}$
leave tangential component of force unchanged
constraint manifold

Figure 7.5. Lagrangian constraints change the forces which are normal to the constraint manifolds

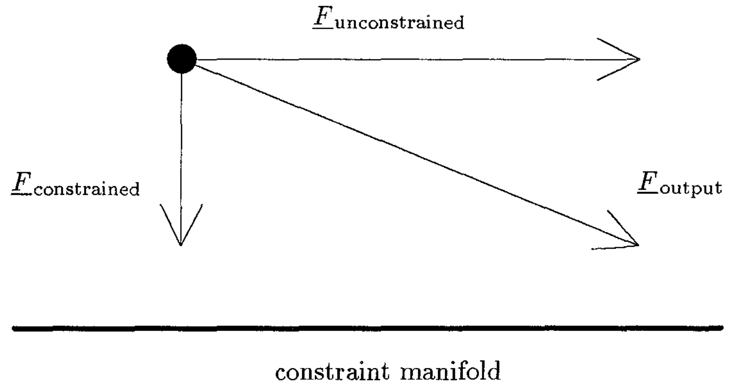

$\underline{F}_{\text{unconstrained}}$
$\underline{F}_{\text{constrained}}$
$\underline{F}_{\text{output}}$
constraint manifold

Figure 7.6. Lagrangian constraints also create forces that fulfill constraints gradually

is acceptable. Figure 7.6 shows a Lagrangian constraint creating a downwards force that is designed to bring the mass point to the constraint manifold. The final force is the sum of the constrained forces and the unconstrained forces. The mass point gradually fulfills the constraint, but it is still free to slide back and forth.

Again, Lagrangian constraints can be extended to multiple constraints by summing additional Lagrange multiplier terms:

$$\mathcal{L} = \int \left(\sum_j m_{ij} \frac{dx_i}{dt}\frac{dx_j}{dt} - U(\underline{x}) - \lambda_\alpha g_\alpha(\underline{x}) \right) dt. \tag{7.13}$$

Setting the variational derivative of $\mathcal{L}$ with respect to x_i to zero yields

$$\sum_j m_{ij} \frac{d^2 x_j}{dt^2} + \frac{\partial U}{\partial x_i} + \lambda_\alpha \frac{\partial g_\alpha}{\partial x_i} = 0. \tag{7.14}$$

Setting the variational derivative of $\mathcal{L}$ with respect to λ_α to zero yields

$$g_\alpha(\underline{x}) = 0. \tag{7.15}$$

Lagrangian constraints can also be applied to inequality constraints. An inequality constraint can be converted into an equality constraint by the use of a slack variable. Let $h(\underline{x}) \geq 0$ be the desired constraint. Introduce a new variable z, to make an equality constraint

$$g(\underline{x}, z) = h(\underline{x}) - z^2 = 0. \tag{7.16}$$

Since z^2 must always be non-negative, then $h(\underline{x}) \geq 0$. The term in the Lagrangian looks like

$$\mathcal{L}_{\text{inequality}} = -\lambda(h(\underline{x}) - z^2). \tag{7.17}$$

Taking the variational derivative with respect to z yields

$$2\lambda z = 0, \tag{7.18}$$

which means that at least one of z and λ must be zero.

Consider the case when $h(\underline{x}) \geq 0$. The constraint can be fulfilled when $z = \sqrt{h}$. Thus, $\lambda = 0$. When $h(\underline{x}) < 0$, then h must be forced to zero. Thus $z = 0$ and the Lagrangian term becomes

$$\mathcal{L}_{\text{inequality}} = -\lambda h(\underline{x}). \tag{7.19}$$

The slack variable does not need to be computed; an equality constraint $h(\underline{x}) = 0$ should be turned on whenever $h(\underline{x}) < 0$.

7.3.1 Rate-Controlled Constraints

The equations (7.14) and (7.15) need to be solved for the Lagrange multipliers. Combining equations (7.14) and (7.15) yields a differential-algebraic equation, which is difficult to solve. In addition, if a physical system starts away from a constraint surface $g(\underline{x}) = 0$, then the system must jump instantly onto the constraint surface. An instantaneous jump is unphysical and looks peculiar when animated.

The constraint can be fulfilled with any pre-determined functional form. To simplify the method, we choose to have g_α approach zero with critically damped motion:

$$\frac{d^2 g_\alpha}{dt^2} + \frac{2}{\tau}\frac{dg_\alpha}{dt} + \frac{1}{\tau^2} g_\alpha = 0. \tag{7.20}$$

Critically damped motion approaches $g_\alpha = 0$, yet only crosses $g_\alpha = 0$ at most one time for each α. If the system starts at rest, then the system effectively reaches $g_\alpha = 0$ in time 5τ. Because we can choose τ, we can choose the rate at which the constraint is fulfilled, hence the name "Rate-Controlled Constraints." Expressing equation (7.20) in terms of $\underline{x}$ yields

$$\sum_i \frac{\partial g_\alpha}{\partial x_i}\frac{d^2 x_i}{dt^2} = -\sum_{i,j} \frac{\partial^2 g_\alpha}{\partial x_i \partial x_j}\frac{dx_i}{dt}\frac{dx_j}{dt} - \frac{2}{\tau}\sum_i \frac{\partial g_\alpha}{\partial x_i}\frac{dx_i}{dt} - \frac{1}{\tau^2} g_\alpha = h\left(\underline{x}, \frac{d\underline{x}}{dt}\right). \tag{7.21}$$

The critically damped constraint equation may be combined with the equations of motion for the physical system, which can be even more general that those in equation (7.14). Constraints can be enforced on physical systems that are dissipative: the forces need not be a single variational derivative of a functional. The general differential equation for a physical system is

$$\sum_j m_{ij}\frac{d^2 x_j}{dt^2} + \sum_\alpha \lambda_\alpha \frac{\partial g_\alpha}{\partial x_i} = -F_i, \tag{7.22}$$

where F_i are the forces on the system, which need not be a variational derivative of one functional.

Combining equations (7.22) and (7.21) yield $N+M$ linear equations in $N+M$ unknowns, where N is the number of moving variables x_i and M is the number of Lagrange multipliers λ_α. Illustrated in matrix form, the linear system is shown in figure 7.7.

$$\begin{pmatrix} m_{ij} & \frac{\partial g}{\partial x_i} \\ \partial g/\partial x_i & 0 \end{pmatrix} \begin{pmatrix} \frac{d^2 x_j}{dt^2} \\ \lambda \end{pmatrix} = \begin{pmatrix} F_i \\ h(\underline{x}, d\underline{x}/dt) \end{pmatrix}. \tag{7.23}$$

Figure 7.7. Rate-controlled constraints solve a linear system to find Lagrange multipliers

The matrices m_{ij} and $g_{\alpha,i}$ are sparse, which implies that the matrix in equation (7.23) is both sparse and symmetric. Appendix A describes methods of solving sparse linear systems.

Time-dependent constraints can be expressed as fulfilling

$$g_\alpha(\underline{x},t) = 0. \tag{7.24}$$

The analysis in the last section can be applied to time varying constraints. The same matrix can be used, but the right-hand side of equation (7.21) is modified because the equation for critically damped motion is modified.

$$\sum_i \frac{\partial g_\alpha}{\partial x_i}\frac{d^2 x_i}{dt^2} = -\sum_{i,j}\frac{\partial^2 g_\alpha}{\partial x_i \partial x_j}\frac{dx_i}{dt}\frac{dx_j}{dt} - \sum_i \frac{\partial^2 g}{\partial x_i \partial t}\frac{dx_i}{dt} - \frac{\partial^2 g}{\partial t^2} - \frac{2}{\tau}\left(\sum_i \frac{\partial g_\alpha}{\partial x_i}\frac{dx_i}{dt} + \frac{dg}{dt}\right) - \frac{1}{\tau^2}g_\alpha. \tag{7.25}$$

7.3.2 Simplifications of Rate-Controlled Constraints

The Lagrange multipliers for certain special cases may be solved for directly, without numerically solving a linear system. If the mass matrix $m_{ij} = m\delta_{ij}$, that is, if the physical system consists of N mass points with identical mass and if for a certain constraint, the affected mass points have no other constraints, then the Lagrange multipliers may be solved for exactly.

The differential equation for a system of identical mass points is

$$m\frac{d^2 x_i}{dt^2} = -F_i - \lambda\frac{\partial g}{\partial x_i}. \tag{7.26}$$

Combining equation (7.21) with (7.26) yields

$$mh(\underline{x}, \frac{d\underline{x}}{dt}) = -\sum_i F_i \frac{\partial g_i}{\partial x_i} - \lambda \sum_i \frac{\partial g}{\partial x_i}\frac{\partial g}{\partial x_i} \tag{7.27}$$

which can easily be solved for the Lagrange multiplier λ:

$$\lambda = \frac{-mh - \sum_i F_i \frac{\partial g}{\partial x_i}}{\sum_i \frac{\partial g}{\partial x_i}\frac{\partial g}{\partial x_i}}. \tag{7.28}$$

The special case where equation (7.28) is applicable arises when a discretized flexible model interacts with any number of other fixed objects.

7.3.3 Inequality Constraints and Lagrange Multipliers

Frequently inequality constraints represent a barrier in position space. For example, a mass point being constrained to lie outside a plane can be represented with an inequality constraint.

When an physical object collides with another physical object, they tend to bounce. The bounce is caused by deformations of the physical objects. For nearly rigid bodies, the deformations are not visible and are wasteful to compute. Thus, the traditional way of computing collisions is by applying impulses to the rigid bodies.

Impulses create discontinuous jumps in the velocity of physical bodies. Therefore, the numerical solver of the differential equation must be able to cope with the velocity discontinuities. In the simulations, we use a piecewise ODE solver, described in [Barzel & Barr89]. A piecewise ODE is defined by three functions $\underline{F}(\underline{y},t)$, $\underline{G}(\underline{y},t)$, and $\underline{H}(\underline{y},t)$ such that the differential equation

$$\frac{d\underline{y}}{dt} = \underline{F}(\underline{y},t) \tag{7.29}$$

may change under the influence of the function g,

$$\min G_i(\underline{y},t) \begin{cases} > 0 & F(\underline{y},t) \text{ is continuous and bounded, } \underline{y} \text{ is a legal state} \\ = 0 & F(\underline{y},t) \text{ may change discontinuously, } \underline{y} \text{ is a legal state} \\ < 0 & \underline{y} \text{ is an illegal state} \end{cases} \tag{7.30}$$

and when $G_i = 0$, the function H_i changes the state vector $\underline{y}$:

$$\underline{y}^+ = \underline{y}^- + H_i(\underline{y},t) \tag{7.31}$$

where $\underline{y}^-$ is the state before the discontinuity and $\underline{y}^+$ is the state after the discontinuity.

For an inequality constraint $g \geq 0$, the system is in one of two states: either the system is fulfilling $g > 0$, or the system is at $g = 0$. When the system is at $g > 0$, the system does not need to activate the constraint. When the system is at $g = 0$, a constraint must be turned on in order to prevent the system from attaining the illegal state $g < 0$. The constraint must only prevent the system from entering the illegal region $g < 0$. If the forces on the system would make the state attain $g > 0$, then the constraint should shut off.

The inequality constraint can be properly implemented if G_α is different when $g_\alpha(\underline{x}) = 0$ and when $g_\alpha(\underline{x}) > 0$:

$$G_\alpha = \begin{cases} g_\alpha(\underline{x}), & \text{when } g_\alpha(\underline{x}) > 0; \\ -\lambda_\alpha, & \text{when } g_\alpha(\underline{x}) = 0. \end{cases} \tag{7.32}$$

When $\lambda_\alpha < 0$, the constraint force, $-\lambda \partial g/\partial x_i$, points towards positive g. When $\lambda_\alpha = 0$, the constraint force should shut off, because it no longer repels the system from $g_\alpha < 0$ (see figure 7.8).

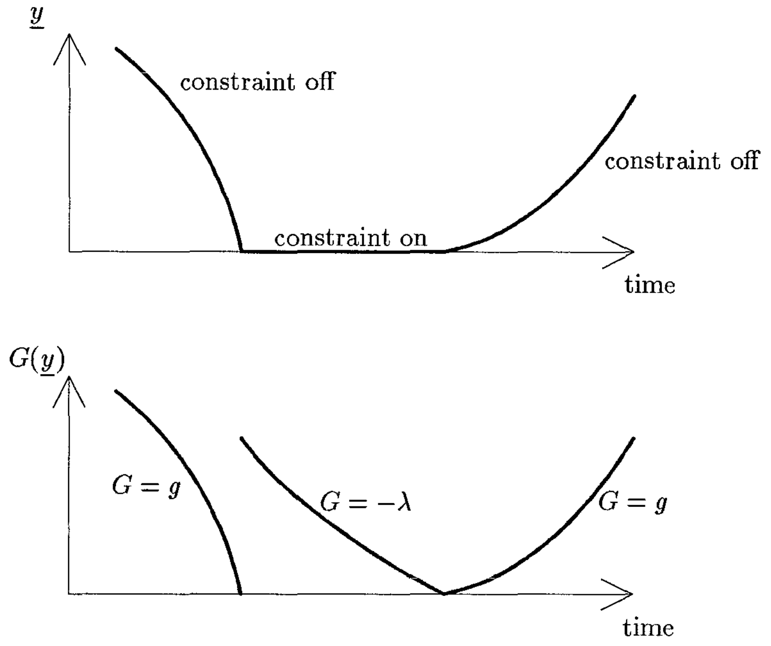


Figure 7.8. $\underline{y}$ and $G(\underline{y})$ for a mass point hitting a plane and sticking briefly

The inequality constraint can be made sticky by modifying the G function:

$$G_\alpha = \begin{cases} g_\alpha(\underline{x}), & \text{when } g_\alpha(\underline{x}) > 0; \\ k - \lambda_\alpha, & \text{when } g_\alpha(\underline{x}) = 0. \end{cases} \tag{7.33}$$

The constraint only shuts off when $\lambda_\alpha = k$, instead of $\lambda_\alpha = 0$. It takes a force of k to release the constraint. Therefore, the manifold $g = 0$ becomes sticky.

As the constraint is turned on, and $g = 0$ becomes true, the mechanical system should no longer penetrate the manifold $g = 0$. An *impulsive force* must therefore applied to the system. We can integrate the governing differential equation and find the discontinuous jump of the velocity of the system:

$$\int_{t_0^-}^{t_0^+} \sum_j m_{ij} \frac{d^2 x_j}{dt^2} + f_i + \sum_\alpha \delta(t - t_0)\lambda_\alpha \frac{\partial g_\alpha}{\partial x_i}\, dt = 0, \tag{7.34}$$

where t_0 is the time of the discontinuity. Evaluating the integral in equation (7.34) yields

$$\sum_j m_{ij} \Delta v_j + \sum_\alpha \Lambda_\alpha \frac{\partial g_\alpha}{\partial x_i} = 0, \tag{7.35}$$

where Δv_j is the jump in the velocity of the system and Λ_α is the strength of impulse. The mechanical system can bounce off of the $g = 0$ manifold with some coefficient of restitution ϵ. Collisions with flexible models usually use $\epsilon = 0$, because a non-zero ϵ creates small vibrations that do not change the animation and require a large amount of computation.

The change in the time derivative of g should be

$$\Delta\dot{g}_\alpha = -(1+\epsilon)\dot{g}_\alpha \tag{7.36}$$

Rewriting equation (7.36) in terms of Δv_i yields

$$\sum_i \frac{\partial g_\alpha}{\partial x_i}\Delta v_i = -(1+\epsilon)\sum_i \frac{\partial g_\alpha}{\partial x_i} v_i. \tag{7.37}$$

Equations (7.35) and (7.37) can be combined into one linear system:

$$\begin{pmatrix} m_{ij} & \frac{\partial g}{\partial x_i} \\ \partial g/\partial x_i & 0 \end{pmatrix}\begin{pmatrix} \Delta v_i \\ \Lambda \end{pmatrix} = \begin{pmatrix} 0 \\ -(1+\epsilon)\dot{g}_\alpha \end{pmatrix}. \tag{7.38}$$

Equation (7.38) uses the same matrix as (7.23), which simplifies the computer code that implements the mechanical system.

In summary, when the system obeys $g_\alpha > 0$, the constraint should remain inactive until $G_\alpha = g_\alpha = 0$. Then, a velocity jump described by equation (7.38) is applied to the system, the constraint $G_\alpha = 0$ is turned on, and G_α is set to be $-\lambda_\alpha$. The constraint is active until $G_\alpha \leq 0$, when the constraint is deactivated and G_α is set to be g_α.

7.4 Simulation Examples

Figures 7.9 through 7.11 show an animation using the penalty method. In figure 7.9, you see a rigid sphere falling onto a flexible sheet. The flexible sheet is glued to the cylinders to form a trampoline. The sphere should drop onto the trampoline, then bounce. The repulsion of the sphere from the trampoline is calculated with the penalty method. However, in figure 7.10, the sphere has penetrated the trampoline. There is a repulsive force between the trampoline and the sphere, but the force is not strong enough to support the weight of the sphere. The sphere drops through the trampoline in figure 7.11.

Figures 7.12 through 7.15 show an animation using Rate-Controlled Constraints. Again, in figure 7.12, you see a rigid sphere falling onto a flexible sheet. The flexible sheet is glued to the cylinders by Rate-Controlled Constraints to form a trampoline. The repulsion of the sphere from the trampoline is also done with Rate-Controlled Constraints. As you can see, in figure 7.13, the sphere does not penetrate then trampoline. In fact, the sphere bounces repeatedly (see figure 7.14) and finally comes to rest (see figure 7.15).

7.5 Conclusions

This chapter discusses two constraint methods for physical systems: the penalty method and rate-controlled constraints. The penalty method does not use any subsidiary equations, but it satisfies constraints only approximately. Rate-controlled constraints satisfy constraints exactly by computing the force necessary to fulfill the constraint with critically damped motion.

Rate-controlled constraints are new to the field of computer graphics. We extend the constraint stabilization method [Baumgarte] by creating models that fulfill goals, in addition to continuing to fulfill goals already met. The goal fulfillment is part of the object definition.

Rate-contolled constraints are examples of a new level of representation for computer graphics model: a teleological model. A designer gives a teleological model a list of goals, and the model fulfills these goals while undergoing realistic physical motion.

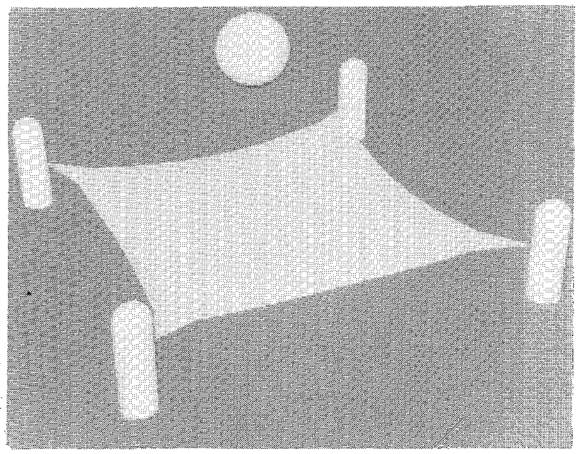


Figure 7.9. A sphere falling onto a trampoline using the penalty method

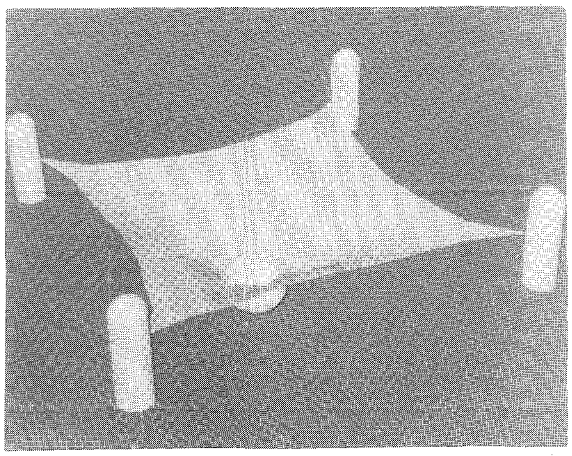


Figure 7.10. The penalty method fails to keep the sphere above the trampoline

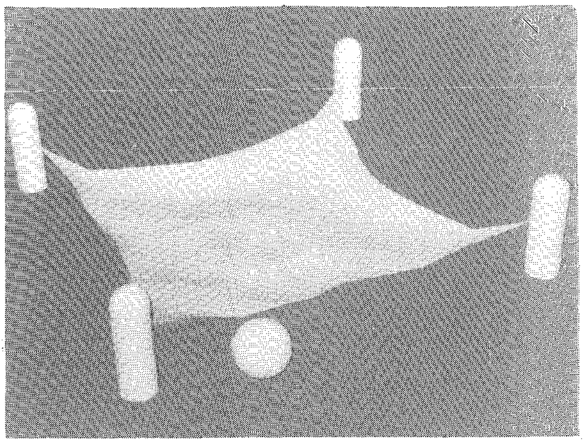


Figure 7.11. Using the penalty method, the sphere eventually falls through the trampoline

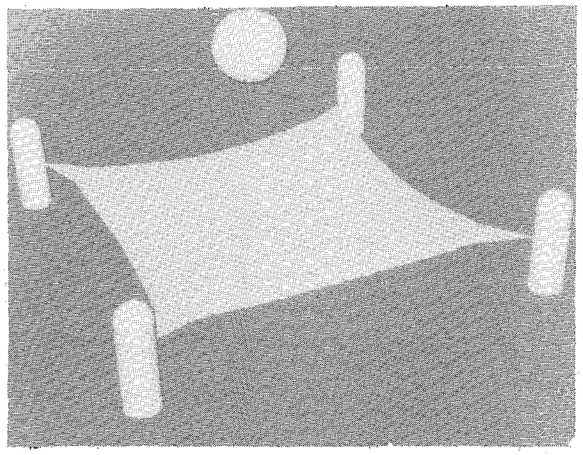


Figure 7.12. A sphere falling onto a trampoline using a rate-controlled constraint

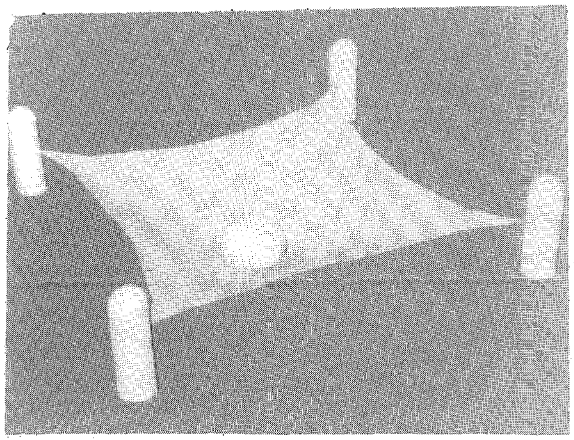


Figure 7.13. Rate-controlled constraints keep the sphere above the trampoline

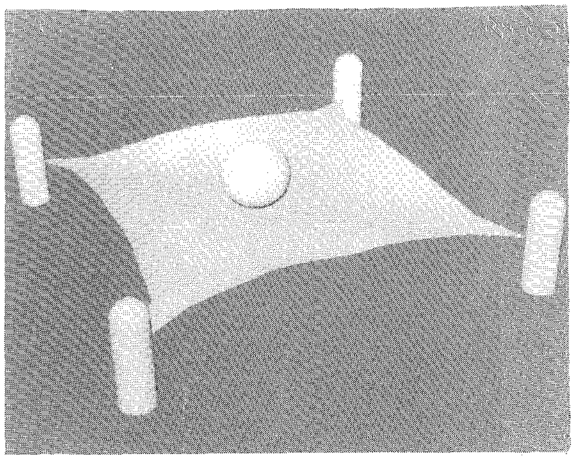


Figure 7.14. The sphere bounces because of the rate-controlled constraint

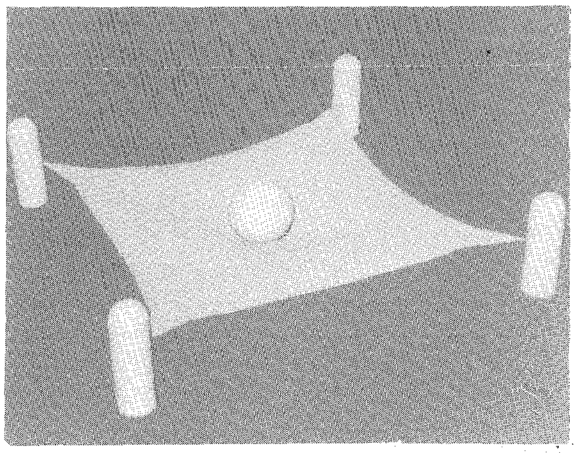


Figure 7.15. The sphere eventually comes to rest above the trampoline

Constrained Computer Graphics: Applications

Chapter 8 : Constraints for the Animation of Flexible Models

8.1 Introduction: Constraints are Useful for Creating Animation

This chapter discusses constraint methods that can be used to create fairly realistic computer animation. The animations use physically-based flexible models described in chapter 6 in order to represent complicated shapes and motions. The animations use the constraint methods described in Chapter 7 to cause the models to obey the wishes of an animator.

Creating realistic animation is useful for several reasons. First, realistic animation is useful for education, as can be seen in the Mechanical Universe project [Blinn]. Second, realistic animation allows better visualization of physical processes for scientific research. Third, realistic computer animation creates a new tool for computer artists to create art. Fourth, intuitive motion of computer graphics models open up a new world of computer-user interfaces.

Currently, the creation of realistic animation is a tricky and time-consuming task. If splines are naively used to guide the motion of models, the models have obviously fake motion. Talented experts, such as John Lasseter [Lasseter], can create beautiful motion with an immense amount of effort. Ideally, physically-based animation should automate the creation of animation: for example, a model ball held above a surface falls down and bounces. Physically-based animation is even more beneficial for the animation of flexible models, where there are hundreds or thousands variables that need to be specified.

However, some animators want controlled physically-based animation. An animator should be able to specify the position of a ball and have unseen hands or forces yank on the ball. Controlled physically-based animation is truly a animator's tool.

Combining physically-based flexible models with constraint methods creates a testbed where it may be possible to create "ultimate" animation: the animation towards which computer graphics has been building. For example, Jello bouncing on a table looks interesting, as does lava flowing over rocks and objects bouncing on a trampoline. Using physically-based plastic models, we can simulate clay and have a sculptor mold a model. We can try to animate creatures by simulating muscles that move a skeleton.

This chapter describes the mathematical formulation of constraints that are useful for creating animation of flexible bodies. There are numerous constraints that can be placed on models by animators. Animators can guide parts of models while allowing other parts to wriggle freely. Animators can allow objects to pass through each other, or they can force objects to bounce off each other or stick together. Furthermore, by specifying constraints on the material properties of the physically-based models, animators can easily create well-known animation effects, such as squash-and-stretch [Lasseter].

8.2 Path-following Constraints

In constraining flexible models, we frequently want to constrain a mass point to follow a specified spatial path parameterized by time, without speeding up or slowing down (see figure 8.1). The predefined path is a useful constraint in animation, where flexible models need to be picked up and moved around. If only a few mass points of the flexible models are constrained, then the rest of the model is free to wriggle in a physically realistic manner.

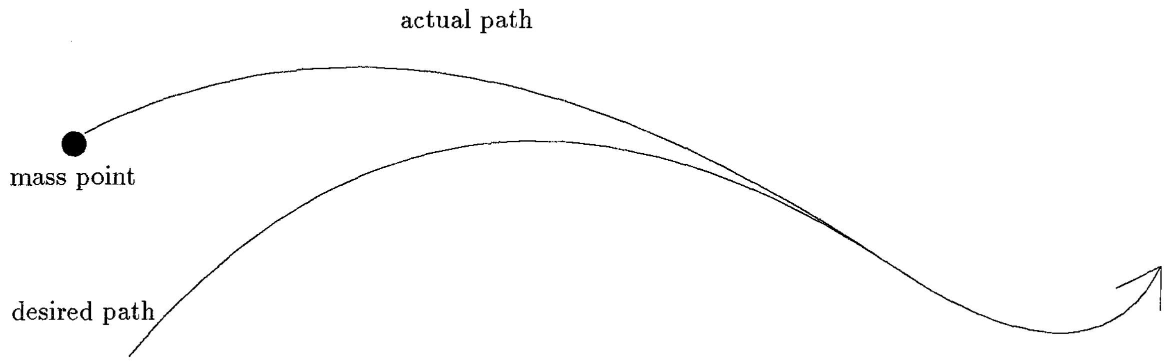


Figure 8.1. The actual path of the mass point gradually matches the desired path.

The constraint function $g_\alpha(\underline{x})$ is a vector between the mass point is and where it should be on the path at that time. When all of the components of g_α are zero, then the constraint is fulfilled. Thus, three Lagrange multipliers, λ_α need to be computed. Let $\underline{x}(t)$ be the current position of the mass point and $\underline{x}^*(t)$ be the desired position of the mass point. Then,

$$\underline{g} = \underline{x}^*(t) - \underline{x}(t). \tag{8.1}$$

Three linear constraints are used instead of one non-linear constraint because $\partial g/\partial x_i$ must be non-zero when the constraint is fulfilled. Also, $\partial g/\partial x_i$ should not change violently near where the constraint is fulfilled.

8.3 Planar Attraction/Repulsion Constraints

Another useful constraint is to force a mass point to lie on a plane. A mass point inside of a polygonal model can be forced outside of the polygonal model by using a planar reaction constraint (see figure 8.2).

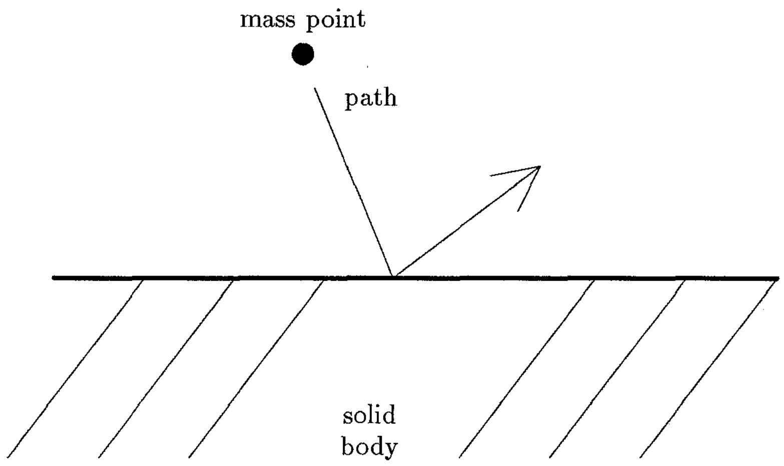


Figure 8.2. Impulses and rate-controlled constraints keep mass points on one side of a plane

Let the physical system have generalized coordinates $\underline{q}$. Let the position of the mass point be the 3-vector (q_i, q_{i+1}, q_{i+2}). Then, the distance of a mass point to a plane is described by the plane equation

$$g(\underline{q}) = A(\underline{q})q_i + B(\underline{q})q_{i+1} + C(\underline{q})q_{i+2}. \tag{8.2}$$

The plane may be part of a larger object, whose position and orientation may depend on a subset of the generalized coordinates $\underline{q}$.

Friction effects lend a greater degree of realism to the animation of physically-based models [Moore & Wilhelms]. A simple treatment of friction involves adding a force which opposes the tangential velocity of a particle. More realistically, however, a particle will stick to a surface until the force on the particle exceeds a threshold known as the static friction.

Consider a particle in contact with a polygon and experiencing a net force $\underline{f}$ (before modification by rate-controlled constraints). The normal force is $\underline{f}_{\mathrm{N}} = (\underline{f} \cdot \underline{\hat{n}})\underline{\hat{n}}$. The tangential force prior to applying friction is

$$\underline{f}_{\mathrm{T}} = \underline{f} - \underline{f}_{\mathrm{N}}. \tag{8.3}$$

If the tangential force is less than some threshold, known as the static friction, then the particle begins to stick and quickly comes to a halt ($\underline{v}_{\mathrm{T}} = \underline{f}_{\mathrm{T}} = \underline{0}$), otherwise a kinetic frictional force acts tangentially to the surface to retard sliding. For greater realism, the static and kinetic frictions should be proportional to the magnitude of the normal force into the surface. The coefficient of static friction ζ is always larger than the coefficient of kinetic friction κ [Feynman, et al.].

$$\underline{f}_T \leftarrow \begin{cases} -m\underline{v}_T/\tau, & \text{if } ||\underline{f}_T|| < \zeta; \\ \underline{f}_T - \kappa\underline{v}_T, & \text{otherwise,} \end{cases} \tag{8.4}$$

where τ is the time constant for the velocity to go to zero when the mass point is stuck. The modification of the tangential force occurs before the rate-controlled constraints are computed.

8.4 Repulsion Constraints from a Implicit Algebraic Model

Repulsion of a mass point from an implicit algebraic model is similar to repulsion from a plane, except that the normal to the implicit algebraic surface depends on the position of the point. The constraint function is general: $g = f(\underline{q})$. The normal vector is

$$\underline{N} = (\frac{\partial g}{\partial q_i}, \frac{\partial g}{\partial q_{i+1}}, \frac{\partial g}{\partial q_{i+2}}). \tag{8.5}$$

The friction algorithm described in the last section is applicable to the collision with an implicit algebraically defined model.

8.5 Constraints that Release

Lagrangian constraint methods can be used to constrain two mass points to occupy the same location (see figure 8.3). Let the position of the first mass point be $\underline{x}^1$ and the position of the second be $\underline{x}^2$. Then, the constraint function is

$$\underline{g} = \underline{x}^1 - \underline{x}^2 \tag{8.6}$$

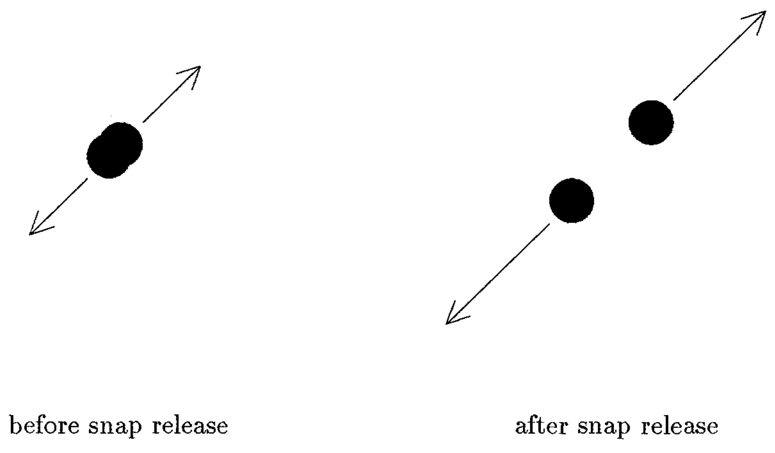


Figure 8.3. A snap constraint releases when the force separating two mass points gets too large.

Snaps can be made to release if the forces on the two mass points become too different. The constraint force is proportional to the vector of Lagrange multipliers $\underline{\lambda}$. Therefore, if the Lagrange multiplier vector exceeds a threshold, then the constraint should be shut off, similar to the algorithm described in [Terzopoulos].

8.6 Moldable Materials

Many materials, such as taffy and putty, are moldable. Moldable materials do not return to their rest shape after being strongly deformed (see figure 8.4). The rest metric of the material becomes a variable in extra set of differential equations. The material is then said to be "plastic" [Fung].

When a moldable material is strongly deformed, the rest state of the material should be constrained to be near the current state of the system:

$$\sum_{i,j}(G_{ij} - G^0_{ij})(G_{ij} - G^0_{ij}) < k. \tag{8.7}$$

The constraint in equation (8.7) limits the amount of strain on the material. In order to look realistic, the rest state of the moldable material can be incompressible, so that strong deformations do not seem to create or destroy material. An incompressibility constraint is described in the next section.

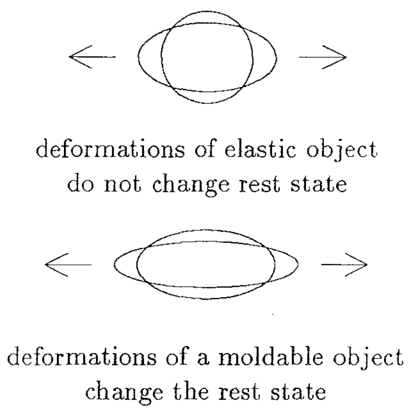

deformations of elastic object
do not change rest state
deformations of a moldable object
change the rest state

Figure 8.4. The rest shape of plastic materials changes after strong deformation.

8.7 Incompressibility Constraints

Hookean elasticity does not fully describe the range of materials that are desirable to animate. For example, a Hookean elastic model can be easily compressed. If an elastic model undergoes violent deformation, as is common in computer graphics, then it will behave more like a sponge than like gelatin. If an incompressible material is desired (see figure 8.5), then a constraint is added to the equations for an elastic element.

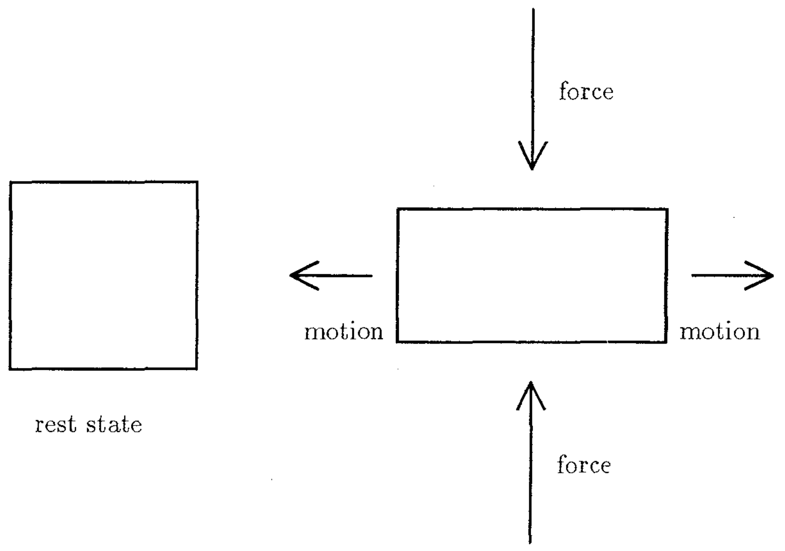


Figure 8.5. Incompressibility preserves the volume of an element

The volume squared of one element is the determinant of the metric tensor G_{ij} of that element [doCarmo]. To constrain the volume of an element to be a constant V_0, we apply the constraint

$$g = \det G_{ij} - V_0^2 = 0. \tag{8.8}$$

An analagous constraint on the rest metric of an element is

$$g = \det G_{ij}^0 - V_R^2 = 0. \tag{8.9}$$

The derivative of the constraint g with respect to the discretized spatial variables $\underline{r}_i$ is needed for the implementation of the constraint. Let C_{ij} be the matrix of cofactors of G_{ij}. Then, the derivative is

$$\frac{\partial g}{\partial \underline{r}_l} = \sum_{i,j} C_{ij} \frac{\partial G_{ij}}{\partial \underline{r}_l}. \tag{8.10}$$

8.8 Results

We have simulated many of the constraints discussed in this chapter using a piecewise differential equation solver based on a predictor-correct method [Barzel & Barr]. Since differential equations are simulated over a time interval, the results are in the form of animation. The figures in this section are individual frames from a sequence.

Figures 8.6 through 8.9 show an animation of a jello cube. The jello is being picked up by a path-following constraint in figures 8.6 and 8.7. Notice how the corner of the jello deforms. Then, the path-following constraint is shut off and the jello falls, hits the table, and becomes strongly deformed (see figure 8.8). Notice that since the cube is compressible, its volume can vary through the course of the animation. The table is implemented with a planar constraint that keeps the jello above the table. Eventually, the jello loses energy and comes to rest (see figure 8.9).

Figure 8.10 shows a compressible seat cushion being squashed with a sphere. The sphere is a physical model with mass. An Lagrangian constraint prevents the sphere from penetrating the cushion.

Figures 8.11 through 8.13 show an incompressible jello cube striking a surface. The cube deforms when it hits the table and then it bounces off. The incompressibility constraint is enforced by an augmented lagrangian constraint, which is described in [Platt & Barr] and is similar to the differential multiplier method.

Figure 8.14 shows an incompressible moldable cube striking a surface. Instead of bouncing off the surface, the moldable cube sticks to the surface, with its sides near the surface bulging out.

Incompressiblity forces the sides to bulge, and the moldability updates the rest shape so that the shape is no longer a cube. The floor constraint is enforced with the methods described in chapter 7. The incompressibility constraint and the moldability constraint are enforced with an augmented lagrangian constraint.

Figures 8.16 through 8.19 illustrate the moldability of the models. A sphere squashes the model in figure 8.16; but the elastic model bounces back to its rest shape in figure 8.17. In figure 8.18, a moldable model starts with the same rest shape, and is squashed by the sphere; but in figure 8.19, the moldable model has a dented edge. The sphere is approximate with a rate-controlled planar constraint and the moldability with an augmented lagrangian constraint.

Figures 8.20 through 8.22 show the friction algorithm described in equation (8.4). An elastic solid is falling onto a funnel in figure 8.20. In figure 8.21, the solid is stuck on the left side of the funnel by static friction. In figure 8.22, the solid comes to rest, held up by static friction on both sides of the funnel.

8.9 Conclusions

In the past, researchers have made models that simulate the behavior of flexible materials. These models automatically move in a physically realistic way, without specifying the exact positions and velocities of the model at all times. The "hands-off" nature of the physically-based models, however, made them hard for an animator to control.

By adding physical modeling constraints to the elastic models, a compromise can be reached between completely specifying the motion of a model and allowing a simulation package to run freely. Constraint methods are useful for controlling the flexible models, while retaining the physically realistic motion created by the physics.

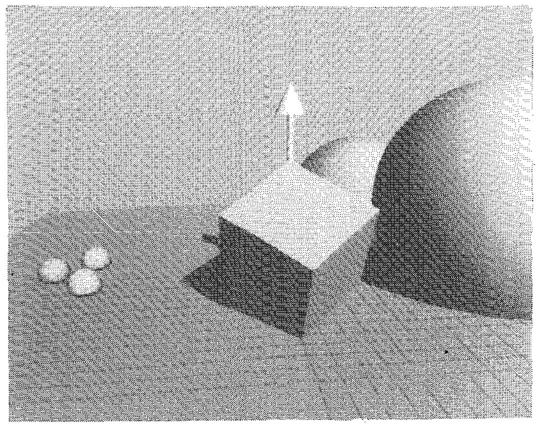


Figure 8.6. A compressible cube of jello is picked up by a path-following constraint.

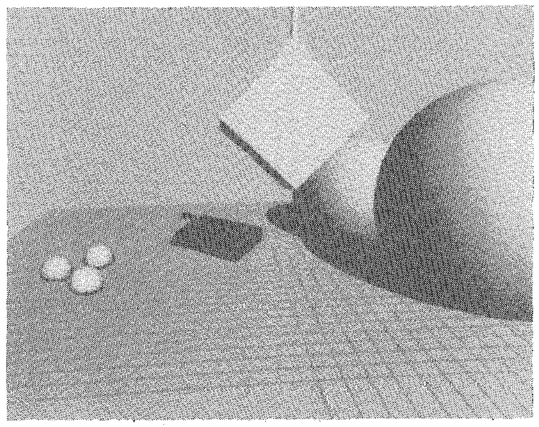


Figure 8.7. The corner of the cube continues to follow a predetermined path.

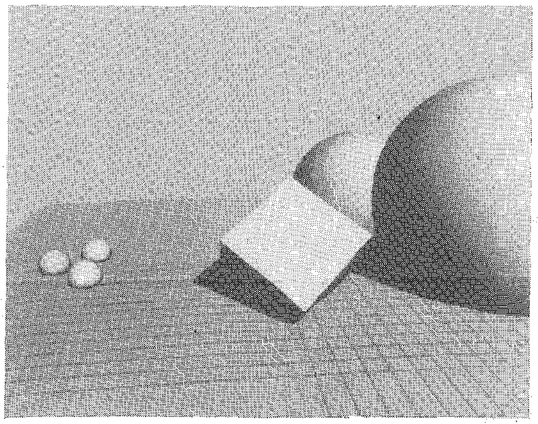


Figure 8.8. The compressible jello hits the table.

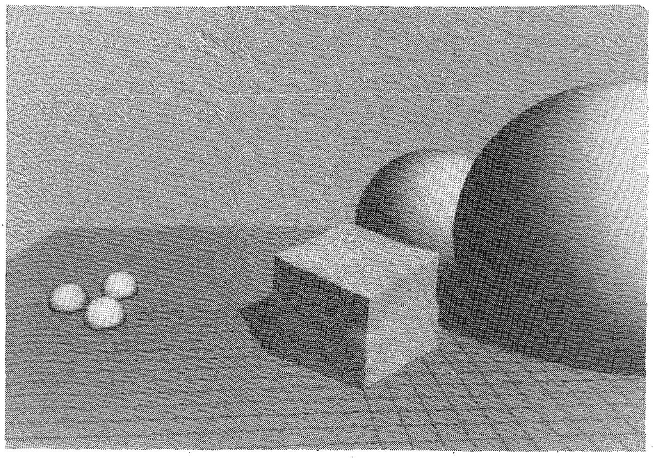


Figure 8.9. The jello wriggles on the table.

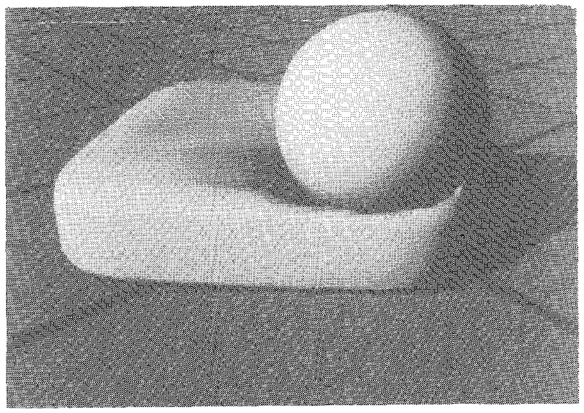


Figure 8.10. A sphere squashes a seat cushion.

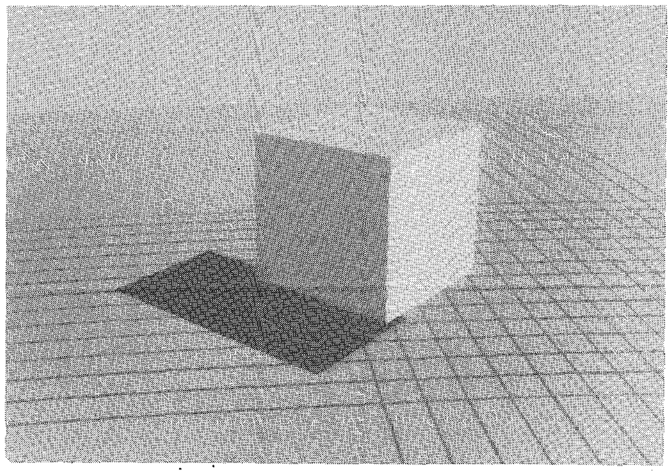


Figure 8.11. A cube of incompressible jello above a table.

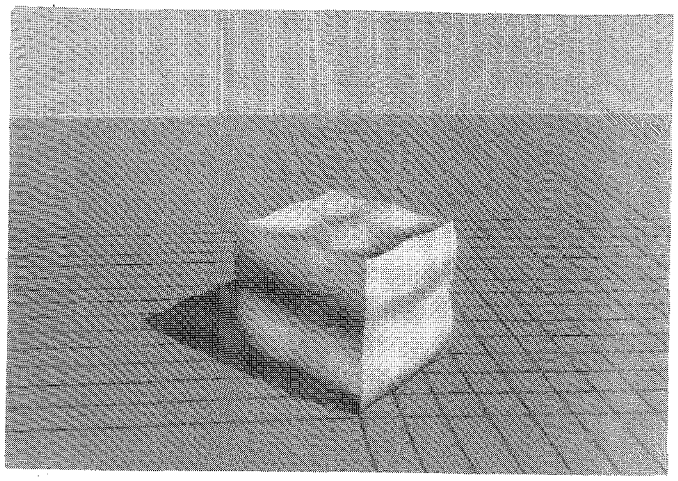


Figure 8.12. The jello hits the table.

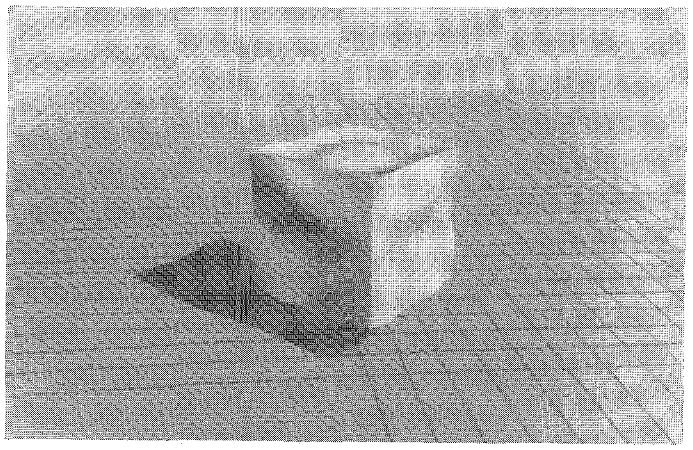


Figure 8.13. The jello quickly bounces off of the table.

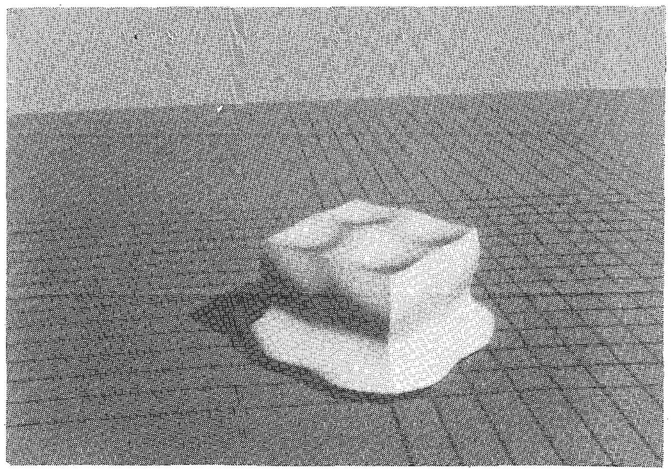


Figure 8.14. A lump of moldable incompressible clay sticks to the table.

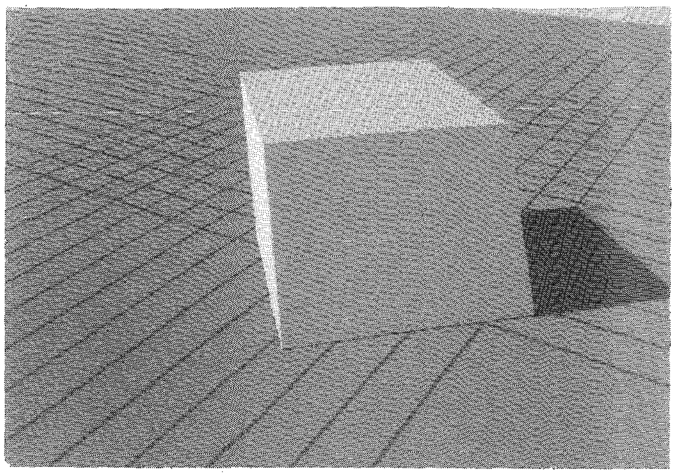


Figure 8.15. The initial rest shape of the models in figures 8.16 through 8.19.

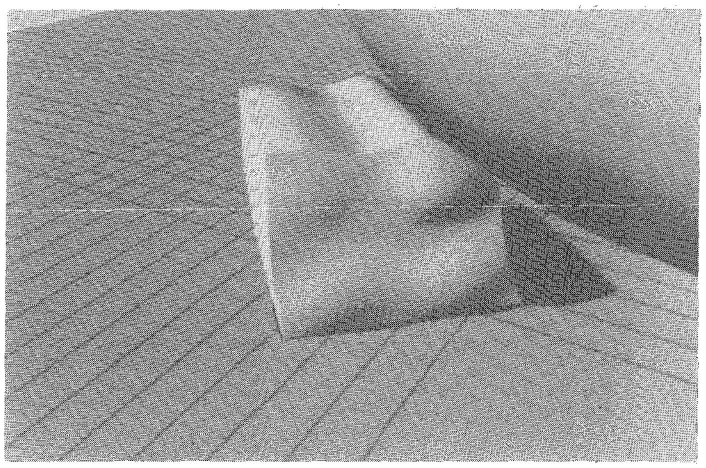


Figure 8.16. An elastic model is squashed.

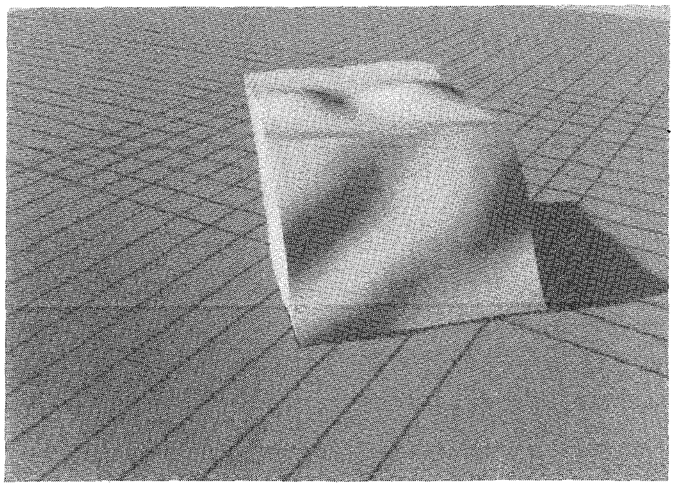


Figure 8.17. An elastic model returns to its rest shape.

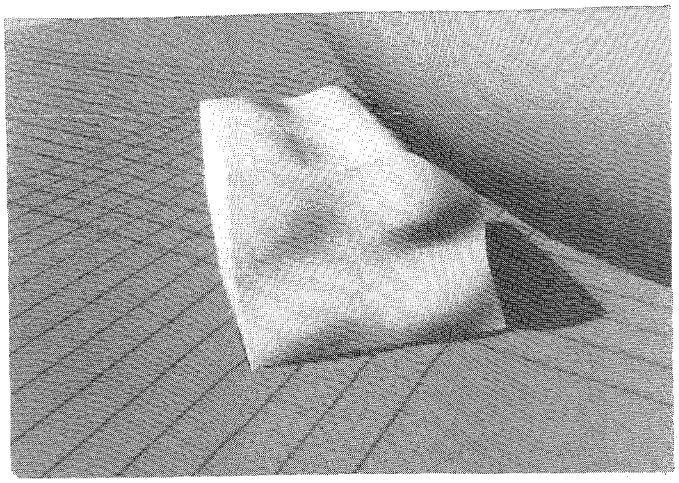


Figure 8.18. A moldable model is squashed.

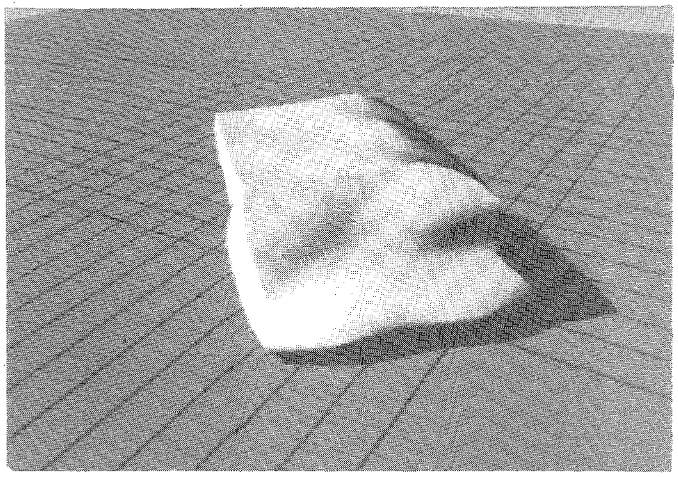


Figure 8.19. A moldable model assumes a new rest shape after strong deformation.

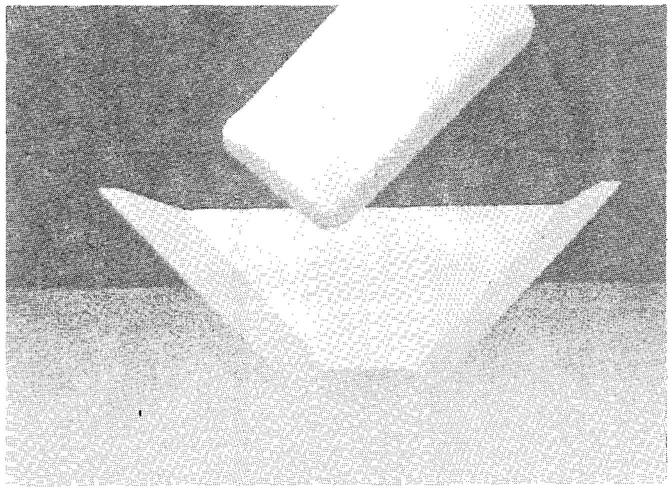


Figure 8.20. An elastic solid is falling onto a funnel.

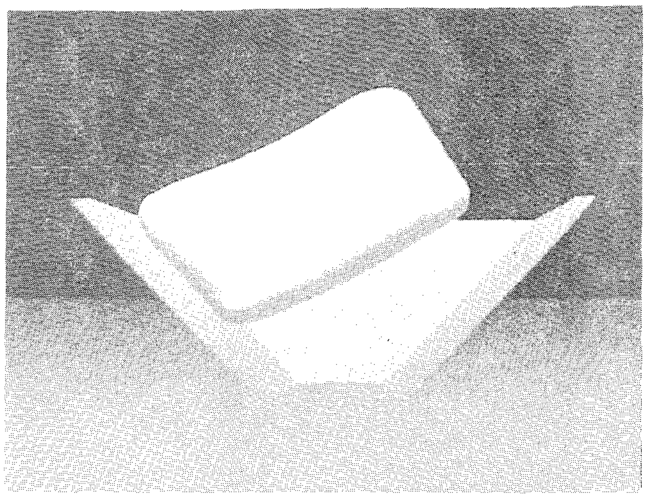


Figure 8.21. The solid is held up on the left side of the funnel by static friction.

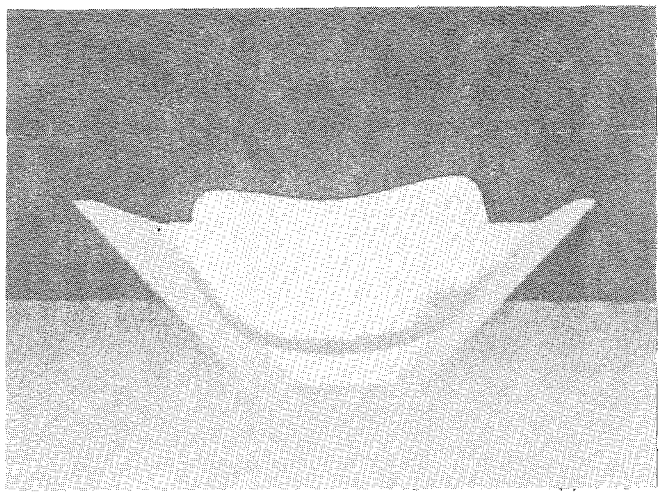


Figure 8.22. The solid is held up on both sides by static friction.

Conclusions and Appendices

Chapter 9 : Conclusions

9.1 Summary

Constraint methods assist designers to control complex models by separating designer-specified behavior from automatic behavior produced by the model. For example, constraints cause neural networks to produce results that lie on a designer-specified manifold, while the optimization function that governs the neural network picks a particular result on the manifold. Constraints cause the motion of computer graphics models to fulfill designer-specified goals, while the unconstrained motion is determined automatically by a physical simulator.

The first half of this thesis uses constrained optimization algorithms to constrain neural networks. Constrained neural network models generate differential equations which can be converted into VLSI circuits that might solve very difficult perceptual or cognitive problems. This thesis describes three constraint methods for constrained neural networks:

- the *penalty method*, which adds terms to the optimization function in order to penalize violation of constraints;
- the *differential multiplier method*, which adds subsidiary differential equations to the neural network to eventually enforce constraints exactly;
- *rate-controlled constraints*, which add extra terms to the differential equations in order to fulfill constraints exactly at some rate.

In order to check if the neural network constraints really work, the constraint methods were applied to solve a number of problems:

- *constrained circuits*,
- *analog decoding*,
- *symmetric edge detection*,
- the *traveling salesman problem*.

The second half of this thesis describes how to constrain physically-based computer graphics models by two force-based constraint methods:

- the *penalty method*, which adds springs to the physically-based models to encourage the model to fulfill the constraints;

- *rate-controlled constraints*, uses inverse dynamics to compute the forces needed to fulfill the constraints with critically-damped motion. These force-based constraint methods can be used to help design complex computer animation.

9.2 Future Work

Topics in this thesis may serve as starting points for further research. The main follow-up to the first part of the thesis would be to actually build VLSI circuits to perform constrained differential optimization:

- Analog circuits that perform constrained differential optimization can be compared to digital implementation of standard constrained optimization methods.
- VLSI circuits may be built that perform analog decoding or compute constrained optimizing splines. These circuits should substantially speed up such computations.
- Constrained neural networks may be used to decode a powerful error-correcting code, such as a convolutional code. Analog VLSI might substantially speed up the decoding of such codes.
- Constrained splines may be used as part of a learning system. Kohonen has described a locally connected neural system which classifies and orders inputs [Kohonen]. Perhaps constrained splines can be used to create a circuit that continually classifies inputs.
- Constrained splines might be useful in other combinatorial optimization problems, such as PC board layout. A circuit that computes constrained splines might be substantially faster than simulated annealing.

The second part of the thesis serves as a stimulus to further computer graphics research:

- New algorithms might speed up the simulation of physically-based flexible models, so that these models can be simulated in real-time. Real-time simulation would allow the creation of artificial reality in the computer, which is a qualitative jump in the usefulness of computer/user interface. The new algorithms might be based on multigrid methods [Hackbusch] or based on the decomposition of models into rigid and linear deformable motions [Terzopoulos & Witkin].
- Constrained physically-based models might be used to create new user-interfaces which are active and respond to user's wishes. New types of 2D and 3D models and new types of constraints may be used in these new user interfaces.
- Constrained physically-based models may be very useful in the creation of traditional 2D animation. Constrained models might help the in-betweening process and might automate the

deformation of cartoon characters.

9.3 Conclusions

This thesis allows neural network and computer graphics researchers to control their physically-based models. Once the models can be controlled, the models can be used to create new and better circuits and animations.

Appendix A : Numerical Techniques

A.1 Finite Elements for Elasticity

Following [Terzopoulos, et al.], there is a potential energy for each flexible element that encourages the metric tensor to be near the rest metric:

$$U = s \sum_{i,j} (G_{ij} - R_{ij})^2, \tag{A.1}$$

where s is the stiffness of the material. The energy in equation (A.1) describes an isotropic material with a Poisson ratio of zero. The force on the the points that make up the element is the derivative of the potential energy [Goldstein]:

$$\underline{F}_k^{\text{elastic}} = s \sum_{i,j} (G_{ij} - R_{ij}) \frac{\partial G_{ij}}{\partial \underline{r}_k}, \tag{A.2}$$

where $\underline{r}_k$ is the position of the kth corner. In addition, there is a viscous damping force that resists changes in the metric tensor:

$$\underline{F}_k^{\text{viscous}} = l \sum_{i,j} \dot{G}_{ij} \frac{\partial G_{ij}}{\partial \underline{r}_k} = l \sum_{i,j,m} \frac{\partial G_{ij}}{\partial \underline{r}_m} \cdot \underline{v}_m \cdot \frac{\partial G_{ij}}{\partial \underline{r}_k}, \tag{A.3}$$

where $\underline{v}_m$ is the velocity of the mth corner, and l is the viscous damping of the element. If $s \gg l$, then the material acts like a solid. If $l \gg s$, then the material acts like a fluid [Truesdell]. Using Newton's Second Law, the differential equations for an unconstrained viscoelastic element is

$$\begin{aligned} \frac{d\underline{r}_i}{dt} &= \underline{v}_i, \\ m_i \frac{d\underline{v}_i}{dt} &= \underline{F}_i^{\text{elastic}} + \underline{F}_i^{\text{viscous}}. \end{aligned} \tag{A.4}$$

where m_{ij} is a diagonal mass matrix.

The viscoelastic forces and the constraint force depend on G_{ij}. Following the finite element method, the G_{ij} in each element is assumed to be the integrated average of G_{ij} over the entire element [Zienkiewicz]. Let $\underline{a}$ be the material coordinates of a point in the element and let $\underline{r}(\underline{a})$ be the position of the points $\underline{a}$. Then, from the definition of metric tensor,

$$G_{ij} = \int \frac{\partial \underline{r}}{\partial a_i} \cdot \frac{\partial \underline{r}}{\partial a_j} \, dV. \tag{A.5}$$

Assuming a position in the element is a linear interpolation of the positions of the corners of the element, the average G_{ij} can be analytically computed from the positions of the corners. To compute G_{ij}, estimates of the spatial derivatives are required:

$$\begin{gathered}\underline{\alpha}_i = \underline{r}_{2i} - \underline{r}_{2i-1}, \quad i = 1,2,3,4 \\ \underline{\beta}_1 = \underline{r}_3 - \underline{r}_1, \quad \underline{\beta}_2 = \underline{r}_4 - \underline{r}_2, \quad \underline{\beta}_3 = \underline{r}_7 - \underline{r}_5, \quad \underline{\beta}_4 = \underline{r}_8 - \underline{r}_6,. \\ \underline{\gamma}_i = \underline{r}_{i+4} - \underline{r}_i, \quad i = 1,2,3,4\end{gathered} \tag{A.6}$$

Averages of the spatial derivatives are also required:

$$\underline{a} = \sum_{i=1}^{4} \alpha_i, \quad \underline{b} = \sum_{i=1}^{4} \beta_i, \quad \underline{c} = \sum_{i=1}^{4} \gamma_i. \tag{A.7}$$

Finally, the various components of G_{ij} can be computed, assuming the element has unit length, width, and height in material coordinates.

$$\begin{aligned} G_{00} &= \frac{1}{18}(2\underline{a}\cdot\underline{a} - \underline{\alpha}_1\cdot\underline{\alpha}_4 - \underline{\alpha}_2\cdot\underline{\alpha}_3), \\ G_{11} &= \frac{1}{18}(2\underline{b}\cdot\underline{b} - \underline{\beta}_1\cdot\underline{\beta}_4 - \underline{\beta}_2\cdot\underline{\beta}_3), \\ G_{22} &= \frac{1}{18}(2\underline{c}\cdot\underline{c} - \underline{\gamma}_1\cdot\underline{\gamma}_4 - \underline{\gamma}_2\cdot\underline{\gamma}_3), \\ G_{01} = G_{10} &= \frac{1}{24}[\underline{a}\cdot\underline{b} - (\underline{\alpha}_1+\underline{\alpha}_2)\cdot(\underline{\beta}_1+\underline{\beta}_2) \\ &\quad + (\underline{\alpha}_3+\underline{\alpha}_4)\cdot(\underline{\beta}_3+\underline{\beta}_4)], \\ G_{02} = G_{20} &= \frac{1}{24}[\underline{a}\cdot\underline{c} - (\underline{\alpha}_1+\underline{\alpha}_3)\cdot(\underline{\gamma}_1+\underline{\gamma}_2) \\ &\quad + (\underline{\alpha}_2+\underline{\alpha}_4)\cdot(\underline{\gamma}_3+\underline{\gamma}_4)], \\ G_{12} = G_{21} &= \frac{1}{24}[\underline{b}\cdot\underline{c} - (\underline{\beta}_1+\underline{\beta}_3)\cdot(\underline{\gamma}_1+\underline{\gamma}_3) \\ &\quad + (\underline{\beta}_2+\underline{\beta}_4)\cdot(\underline{\gamma}_2+\underline{\beta}_4)]. \end{aligned} \tag{A.8}$$

As in the continuous case, the diagonal terms of the metric tensor G_{ij} in equation (A.8) depend on various distances in the cube, while the off-diagonal terms depend on angles. Also, the G_{ij} are quadratic functions of the $\underline{r}_k$. Thus, $\partial G_{ij}/\partial r_k$ are complicated, although linear, functions of r_k.

A few examples of $\partial G_{ij}/\partial r_k$ are

$$\begin{aligned} \frac{\partial G_{00}}{\partial r_0} &= \frac{1}{18}(-4\underline{a} + \underline{\alpha}_4), \\ \frac{\partial G_{01}}{\partial r_2} &= \frac{1}{24}(-\underline{b} + \underline{a} - \underline{\beta}_1 - \underline{\beta}_2 + \underline{\alpha}_1 + \underline{\alpha}_2). \end{aligned} \tag{A.9}$$

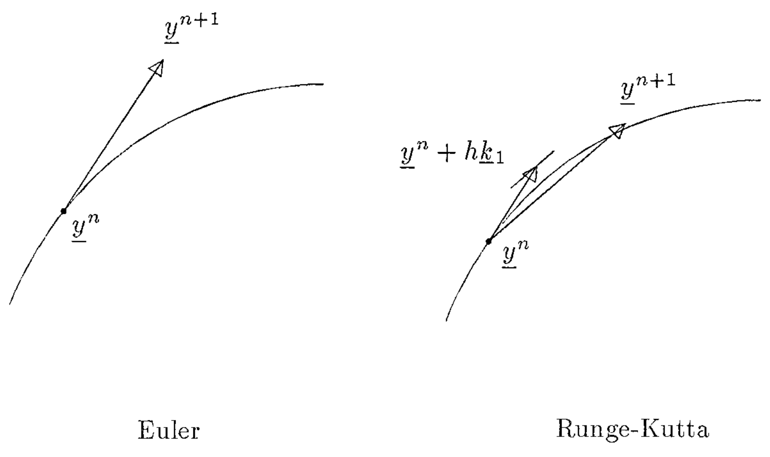


Figure A.1. Euler and Runge-Kutta

A.2 How to Numerically Solve Differential Equations

Now, let us consider the solution of a system of ODEs, either derived from PDEs, or from some other source. Again, we must discretize the derivatives. Very frequently, the derivative in a system of ODEs is time. However, time is different than space. Time flows forward: we usually do not want "non-causal" effects. Thus, we run the system forward in time, recording the state of the system as it simulates. Running time forward is known as an initial value problem, since the system is started in some initial value and then is simulated forward in time.

A set of linked ODEs can usually be expressed as the system

$$\frac{d\underline{y}}{dt} = \underline{f}(t, \underline{y}). \tag{A.10}$$

Higher order derivatives can always be broken down into first-order derivatives and extra variables.

The simplest time discretization of the equation (A.10) is

$$\underline{y}^{n+1} = \underline{y}^{n} + h\underline{f}(t^{n}, \underline{y}^{n}), \tag{A.11}$$

where the superscripts denote the discretized time, and h is the time step of the algorithm. This algorithm is called the Euler method, and it doesn't work very well. The derivative over a time step is estimated to be constant. The accuracy of the Euler method is proportional to the time step h. In addition, when the system has many different intrinsic time scales (this is known as a *stiff* system), the algorithm will be stable only at the smallest time scale, which will force you to wait for a long time for useful answers.

A differencing scheme which is $O(h^2)$ accurate is the second-order Runge-Kutta method, which tries to approximate the derivative by the derivative in between $\underline{y}^n$ and $\underline{y}^{n+1}$. This is equivalent to fitting a parabola to the function over the interval (see figure A.1). Namely,

$$\begin{aligned} \underline{k}_1 &= \underline{f}(t^n, \underline{y}^n), \\ \underline{k}_2 &= \underline{f}(t^n + \frac{h}{2}, \underline{y}^n + \frac{h}{2}\underline{k}_1), \\ \underline{y}^{n+1} &= \underline{y}^n + h\underline{k}_2. \end{aligned} \tag{A.12}$$

There are higher-order Runge-Kutta methods: consult [Press, et al.] or [Dahlquist & Bjorck] for more details. [Press, et al.] also contains code for adaptively controlling the step size of the Runge-Kutta solver.

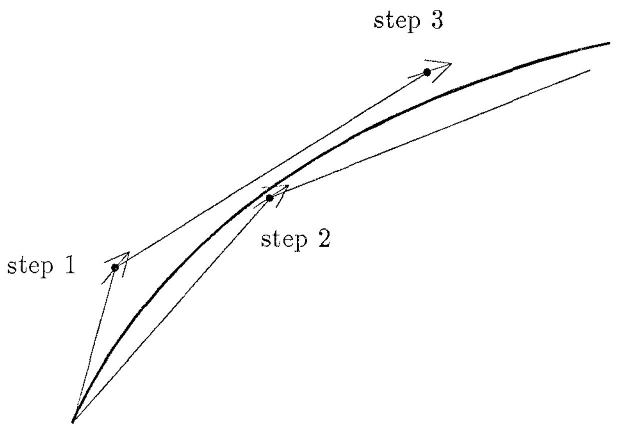


Figure A.2. The modified midpoint method

Frequently, higher-order Runge-Kutta solvers are slow, because they use many derivative evaluations per step. In the differential equations that typically arise in computer graphics, the derivative evaluations are very time consuming. One way of getting higher order accuracy without so many derivative evaluations is to use the *modified midpoint method* [Press, et al.]. A large step H is divided into n equal small steps $h = H/n$. Then, a second-order accurate method is used for each small step h:

$$\underline{z}^{m+1} = \underline{z}^{m-1} + 2h\underline{f}(t + mh, \underline{z}^m) \tag{A.13}$$

where $\underline{z}$ are the intermediate results. The method is started off with an Euler step:

$$\underline{z}^0 = \underline{y}(t), \quad \underline{z}^1 = \underline{z}^0 + h\underline{f}(t, \underline{z}^0), \tag{A.14}$$

and finally, the $\underline{y}(t+H)$ is computed via

$$\underline{y}(t+H) = \frac{1}{2}\left[\underline{z}^n + \underline{z}^{n-1} + h\underline{f}(x+H, \underline{z}^n)\right] , \tag{A.15}$$

(see figure A.2). The modified midpoint method asymptotically uses only one derivative evaluation per intermediate step.

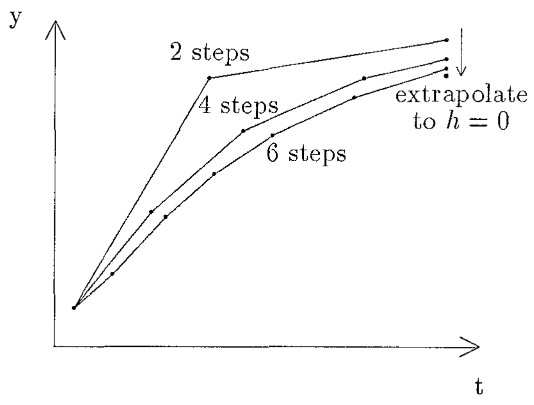


Figure A.3. The Bulirsch-Stoer method

The modified midpoint method is the basis of the powerful Bulirsch-Stoer method [Gear][Stoer & Bulirsch]. Bulirsch-Stoer incorporates a trick: it tries to perform the modified midpoint method for numbers of intermediate steps n. Then, it uses rational polynomial extrapolation to predict what the result $\underline{y}(t+H)$ would be for an *infinite* number of intermediate steps ($h = 0$) (see figure A.3). The accuracy of the method is quite good. Code for Bulirsch-Stoer is given in [Press, et al.]. Bulirsch-Stoer works extremely well, and doesn't used many derivative evaluations.

Runge-Kutta and Bulirsch-Stoer are both *explicit* methods. That is, the derivative at a point depends on information prior to that point. Explicit methods have the problem mentioned above for Euler method: they must operate at a step size shorter than the fastest time scale of the problem (see figure A.4). An *implicit* scheme is stable for large time steps. In an implicit method, a derivative at a point can depend on the information at that point. For example, an implicit Euler method (also called the *backwards Euler method*) is

$$\underline{y}^{n+1} = \underline{y}^n + h\underline{f}(t^{n+1}, \underline{y}^{n+1}) \tag{A.16}$$

Compare (A.16) to equation (A.11): the derivative is evaluated at the end of the step, instead of the beginning. Implicit techniques use the derivative information from the end of a time step to

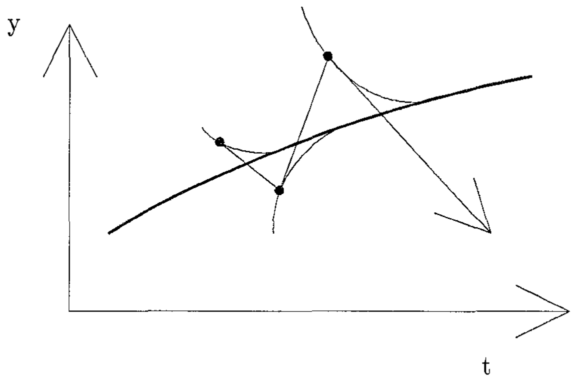


Figure A.4. Explicit methods are sometimes unstable

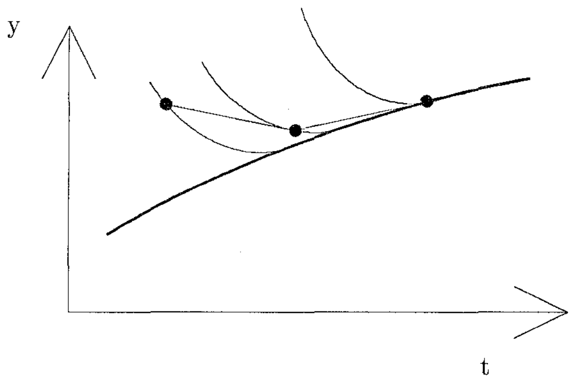


Figure A.5. Implicit methods are stable

make the method more stable (see figure A.5). A more accurate implicit method is an implicit second-order Runge-Kutta method:

$$\underline{y}^{n+1} = \underline{y}^{n} + \frac{h}{2}\left(\underline{f}(t^{n}, \underline{y}^{n}) + \underline{f}(t^{n+1}, \underline{y}^{n+1})\right) \tag{A.17}$$

How does one use these implicit methods? If $\underline{f}$ happens to be linear in $\underline{y}$, then these methods reduce the ODEs into a set of linear algebraic equations. Otherwise, these methods reduce the ODEs to a set of non-linear algebraic equations. Solutions of algebraic equations are discussed below. Implicit methods are very stable, but take a long time to compute. If the equations are stiff enough, then the increase in step size you can take makes up for the added amount of computation per step.

A related equation to an ODE is a Differential Algebraic Equation (DAE):

$$\underline{g}(\underline{y}, \frac{d\underline{y}}{dt}, t) = 0. \tag{A.18}$$

DAEs can also be solved using implicit techniques. Substituting the difference formula for $d\underline{y}/dt$ yields

$$\underline{g}(\underline{y}^{n+1}, \frac{1}{h}(\underline{y}^{n+1} - \underline{y}^{n}), t) = 0, \tag{A.19}$$

which is again a set of algebraic equations, possibly non-linear [Petzold].

A.3 How to Solve Non-Linear Algebraic Equations

Large systems of non-linear algebraic equations are hard to solve. However, non-linear equations derived from ODEs have a nice property: we can use the solution from the last time step as an initial guess for an iterative procedure to find the solution for the present time step. A reasonable non-linear equation solving technique is *Newton's method.* Newton's method tries to find the zero of a non-linear function $\underline{h}(\underline{y})$ by truncating the Taylor's series after two terms. Thus, an iteration procedure is defined:

$$0 = h_i(\underline{y}^k) + \left.\frac{\partial h_i}{\partial y_j}\right|_{\underline{y}^k} (\underline{y}^{k+1} - \underline{y}^k). \tag{A.20}$$

This can be re-written as

$$\sum_{i,j} \alpha_{ij}\delta_j = \beta_i,$$

where $\alpha_{ij} = \partial h_i/\partial y_j$, $\beta_i = -h_i(\underline{y}^k)$, and we try to solve for $\delta_j = y_j^{k+1} - y_j^k$. Thus, a linear algebraic system of equation must be solved at each step of this iteration procedure. Notice that Newton's method requires the matrix α_{ij}. Sometimes this matrix is sparse and it usually can be computed analytically. Thus, one can write code for the exact formula, $\partial h_i/\partial y_j$, or one can compute this derivative by using finite differences. Gear observes that the iteration procedure converges even when the matrix is only approximately correct [Gear]. Gear uses this fact to reduce the number of function evaluations in his ODE code.

Other non-linear equation solvers, such as quasi-Newton methods, are discussed in [Gill, et al.].

A.4 How to Solve Sparse Linear Algebraic Equations

There are a plethora of linear equation solvers. Which one you choose depends on the type of linear problem you are working with.

A method which yields a solution to a linear problem in one step is called a *direct* method. The prime example of a direct method is LU decomposition. A method which yields a sequence of solutions that gradually approach the solution to a linear problem is known as an *iterative* method. Examples of iterative methods are the Gauss-Seidel method, conjugate gradient [Golub & Van Loan], and Lanczos iteration [Golub & Van Loan]. Gauss-Seidel is particularly easy to describe and is described below. Pre-conditioned conjugate gradient and Lanczos iteration are very effective and described in [Golub & Van Loan].

Consider

$$\sum_{j=1}^{N} a_{ij} x_j = b_i. \tag{A.21}$$

Gauss-Seidel wants to update $\underline{x}$ one element at a time. Consider x_k:

$$a_{ik} x_k + \sum_{j=1}^{k-1} a_{ij} x_j + \sum_{j=k+1}^{N} a_{ij} x_j = b_i. \tag{A.22}$$

However, (A.22) is a system of N equations, which overdetermines x_k. Thus, consider one equation:

$$a_{kk} x_k + \sum_{j=1}^{k-1} a_{kj} x_j + \sum_{j=k+1}^{N} a_{kj} x_j = b_k. \tag{A.23}$$

We can sweep through k, "solving" one row of the matrix at a time. Thus, with superscripts denoting iteration number,

loop for $k = 1$ to N

$$x_k^{n+1} = -\frac{1}{a_{kk}} \left(\sum_{j=1}^{k-1} a_{kj} x_j^{n+1} + \sum_{j=k+1}^{N} a_{kj} x_j^{n} - b_k \right). \tag{A.24}$$

Gauss-Seidel does not need very much storage. As the loop proceeds, the new x_k are stored in place and are used in further computation of the vector. If the matrix $\underline{\underline{a}}$ is sparse, the iteration proceeds quickly. However, the convergence properties of the Gauss-Seidel method are somewhat poor. It will converge if $\underline{\underline{a}}$ is positive definite [Golub & Van Loan]

Appendix B : Convergence of the Differential Multiplier Method

To solve the constrained optimization problem

$$\text{Find minimum of } f(\underline{x}) \text{ subject to } g_\alpha(\underline{x}) = 0, \tag{B.1}$$

Chapter 2 discusses the basic differential multiplier method (BDMM): a set of differential equations

$$\begin{aligned} \frac{dx_i}{dt} &= -\frac{\partial f}{\partial x_i} - \sum_\alpha \lambda_\alpha \frac{\partial g}{\partial x_i} \\ \frac{d\lambda}{dt} &= g(\underline{x}). \end{aligned} \tag{B.2}$$

The damped oscillations of the BDMM can be explained by differentiating the first equation in equation (B.2) and then substituting:

$$\ddot{x}_i + \sum_j \left(\frac{\partial^2 f}{\partial x_i \partial x_j} + \sum_\alpha \lambda_\alpha \frac{\partial^2 g_\alpha}{\partial x_i \partial x_j} \right) \dot{x}_j + \sum_\alpha g_\alpha \frac{\partial g_\alpha}{\partial x_i} = 0. \tag{B.3}$$

Equation (B.3) is the equation for a damped mass system, with an inertia term, $\ddot{x}_i$, a damping matrix,

$$A_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} + \sum_\alpha \lambda \frac{\partial^2 g_\alpha}{\partial x_i \partial x_j}, \tag{B.4}$$

and an internal force, $\sum_\alpha g_\alpha \partial g_\alpha / \partial x_i$, which is the derivative of the internal energy,

$$U = \frac{1}{2} \sum_\alpha (g_\alpha(\underline{x}))^2. \tag{B.5}$$

If the system is damped and the state remains bounded, the state falls into a constrained minimum.

As in physics, we can construct a total energy of the system, which is the sum of the kinetic and potential energies [Luenberger].

$$E = T + U = \frac{1}{2} \sum_i (\dot{x}_i)^2 + \frac{1}{2} \sum_\alpha (g_\alpha(\underline{x}))^2. \tag{B.6}$$

If the total energy is decreasing with time, and the state remains bounded, then the system will dissipate any extra energy, and will settle down into the state where

$$\begin{aligned} g_\alpha(\underline{x}) &= 0, \\ \dot{x}_i &= \frac{\partial f}{\partial x_i} + \sum_\alpha \lambda \frac{\partial g_\alpha}{\partial x_i} = 0, \end{aligned} \tag{B.7}$$

which is a constrained extremum of the original problem in equation (B.1).

The time derivative of the total energy in equation (B.6) is

$$\dot{E} = \sum_i \ddot{x}_i \dot{x}_i + \sum_{i,\alpha} g_\alpha(\underline{x}) \frac{\partial g_\alpha}{\partial x_i} \dot{x}_i = -\sum_{i,j} \dot{x}_i A_{ij} \dot{x}_j . \tag{B.8}$$

If damping matrix A_{ij} is positive definite, the system converges to fulfill the constraints [Arrow, et al.].

MDMM always converges for quadratic programming, a special case of constrained optimization. A quadratic programming problem has a quadratic function $f(\underline{x})$ and piecewise linear continuous functions $g_\alpha(\underline{x})$, such that

$$\frac{\partial^2 f}{\partial x_i \partial x_j} \text{ is positive definite and } \frac{\partial^2 g_\alpha}{\partial x_i \partial x_j} = 0. \tag{B.9}$$

Under these circumstances, the damping matrix A_{ij} is positive definite for all $\underline{x}$ and λ, so that the system converges to the constraints.

It is possible, however, to pose a problem that has contradictory constraints. For example,

$$g_1(x) = x = 0 \text{ and } g_2(x) = x - 1 = 0. \tag{B.10}$$

In the case of conflicting constraints, the MDMM compromises, trying to make each constraint g_α as small as possible. However, the Lagrange multipliers λ_α go to $\pm\infty$ as the constraints oppose each other. It is possible, however, to arbitrarily limit the λ_α at some large absolute value.

Minimizations can converge without necessarily performing pure gradient descent and ascent. Consider the equations

$$\begin{aligned} \sum_j Q_{ij} \dot{x}_j &= -\frac{\partial f}{\partial x_i} - \sum_\alpha \lambda_\alpha \frac{\partial g_\alpha}{\partial x_i}, \\ \dot{\lambda}_\alpha &= \sum_\beta M_{\alpha\beta} g_\beta(\underline{x}). \end{aligned} \tag{B.11}$$

where Q_{ij} and $M_{\alpha\beta}$ are symmetric, positive definite, constant matrices. An energy analogous to that in equation (B.6) is

$$E = \sum_{i,j} \dot{x}_i Q_{ij} \dot{x}_j + \sum_\alpha g_\alpha M_{\alpha\beta} g_\beta . \tag{B.12}$$

The time derivative of the energy in equation (B.12) is

$$\dot{E} = -2 \sum_{i,j} \dot{x}_i \left(\frac{\partial^2 f}{\partial x_i \partial x_j} + \lambda_\alpha \frac{\partial^2 g_\alpha}{\partial x_i \partial x_j} \right) \dot{x}_j , \tag{B.13}$$

just as in the case with gradient descent. As long as both Q_{ij} and $M_{\alpha\beta}$ are positive definite, then the non-gradient descent equations converge to the same points as the gradient descent equations.

For a given constrained optimization problem, it is frequently necessary to alter the BDMM to have a region of positive damping surrounding the constrained minima. [Arrow, et al.] combine the multiplier method with the penalty method to yield a modified multiplier method that is locally convergent around constrained minima.

The damping matrix is modified by the penalty force to be

$$A_{ij} = \frac{\partial^2 f}{\partial x_i \partial x_j} + \sum_\alpha \lambda_\alpha \frac{\partial^2 g_\alpha}{\partial x_i \partial x_j} + c_\alpha \frac{\partial g_\alpha}{\partial x_i}\frac{\partial g_\alpha}{\partial x_j} + cg\frac{\partial^2 g_\alpha}{\partial x_i \partial x_j}. \tag{B.14}$$

[Arrow, et al.] prove a theorem that states that there exists a $c^* > 0$, such that if $c > c^*$, the damping matrix in equation (B.14) is positive definite at regular constrained minima. Using continuity, the damping matrix is positive definite in a region R surrounding each regular constrained minimum.

Appendix C : Calculus of Variations

This appendix will explain how to take *variational derivatives* of functionals. The derivation follows [Gelfand & Fomin].

Consider the function $y(x)$, where x and y are scalars, and the functional

$$J[y] = \int_a^b F(x, y, y')dx. \tag{C.1}$$

The easiest way to take a derivative of this functional is to consider the integral to be a sum of n terms, and let $n \to \infty$. The functional in equation (C.1) is representative of the functionals encountered in elasticity theory and other branches of physics.

Thus, divide the interval [a,b] into n equal segments, the endpoints of which are labeled by x_i. Let $h = x_{i+1} - x_i$. Now, approximate the function $y(x)$ by a polygonal line with the vertices

$$(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n), (x_{n+1}, y_{n+1}). \tag{C.2}$$

Now, equation (C.1) can be approximated by using finite differences to be

$$J(y_0, y_1, \ldots, y_n) = h \sum_i F\left(x_i, y_i, \frac{y_{i+1} - y_i}{h}\right). \tag{C.3}$$

Let us compute the partial derivatives of J:

$$\begin{aligned} \frac{\partial J(y_0, y_1, \ldots, y_n)}{\partial y_i} =& hF_y\left(x_i, y_i, \frac{y_{i+1} - y_i}{h}\right) \\ &+ F_{y'}\left(x_{i-1}, y_{i-1}, \frac{y_i - y_{i-1}}{h}\right) - F_{y'}\left(x_i, y_i, \frac{y_{i+1} - y_i}{h}\right). \end{aligned} \tag{C.4}$$

where F_y is the function $\partial F/\partial y$ and $F_{y'}$ is the function $\partial F/\partial y'$. From functional analysis [Gelfand & Fomin],

$$\left.\frac{\delta J[y]}{\delta y}\right|_{y_i} = \lim_{h \to 0} \frac{1}{h} \frac{\partial J(y_0, y_1, \ldots, y_n)}{\partial y_i}. \tag{C.5}$$

The extra factor of h allows the limit to exist, since the partial derivatives scale as h. Thus,

$$\begin{aligned} \left.\frac{\delta J[y]}{\delta y}\right|_{y_i} = \lim_{h \to 0} \Bigg[& F_y\left(x_i, y_i, \frac{y_{i+1} - y_i}{h}\right) \\ & - \frac{1}{h}\left(F_{y'}\left(x_i, y_i, \frac{y_{i+1} - y_i}{h}\right) - F_{y'}\left(x_{i-1}, y_{i-1}, \frac{y_i - y_{i-1}}{h}\right)\right)\Bigg]. \end{aligned} \tag{C.6}$$

Passing to the continuum limit yields the variational derivative

$$\frac{\delta J[y]}{\delta y} = \frac{\partial F}{\partial y} - \frac{d}{dx}\left(\frac{\partial F}{\partial y'}\right). \tag{C.7}$$

More generally, if a functional is of the form

$$J[y] = \int_a^b F(x, y, y', \ldots, y^{(N)})dx \tag{C.8}$$

then the variational derivative is

$$\frac{\delta J[y]}{\delta y} = \sum_{n=0}^{N} (-1)^n \frac{d^n}{dx^n} \left(\frac{\partial F}{\partial y^{(n)}} \right). \tag{C.9}$$

where $y^{(n)}$ means the nth derivative of y.

An even more general formula is when the functional is of the form

$$J[y] = \int_\Omega F(\underline{x}, y, y')d\underline{x} \tag{C.10}$$

where the integral is now a multiple integral over $\underline{x} \in \Omega$. In this case, the variational derivative is

$$\frac{\delta J[y]}{\delta y} = F_y - \sum_i \frac{\partial}{\partial x_i} F_{y_{x_i}} \tag{C.11}$$

Appendix D : An Introduction to Circuit Theory

This appendix is a very simple introduction to circuit theory. Circuit elements, such as resistors, capacitors, or amplifiers, can be composed to form more complex circuits. The mathematics that describe the circuit elements can also be composed to describe the complex circuits. The state of a circuit can be described by voltages of nodes and currents flowing into and out of nodes.

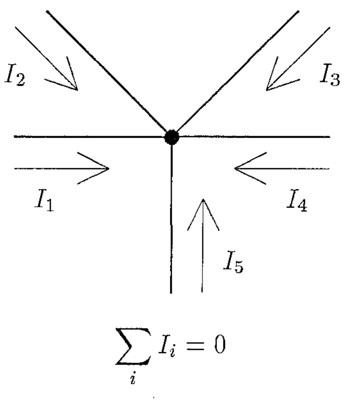


Figure D.1. Kirchoff's current law

The composition of circuit elements is regulated by Kirchoff's current law [Horowitz & Hill]. Kirchoff's current law states that the sum of the currents into a node sum to zero:

$$\sum_{\mathrm{i}} I_i = 0 \tag{D.1}$$

Many electronic components have current which depends on the current state of the circuit.

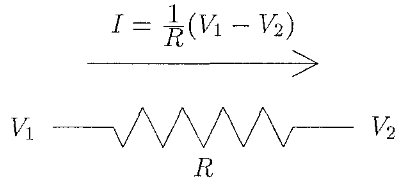


Figure D.2. A resistor

For example, a resistor has a current proportional to the difference in voltage across it (see figure D.2):

$$I_{\text{across resistor}} = \frac{1}{R}(V_1 - V_2) \tag{D.2}$$

A resistor tends to equalize the voltages across it.

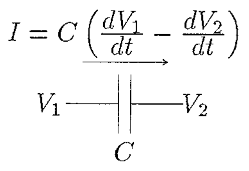


Figure D.3. A capacitor

A capacitor has a current proportional to the difference in the derivatives of the voltages (see figure D.3):

$$I_{\text{across capacitor}} = C\left(\frac{dV_1}{dt} - \frac{dV_2}{dt}\right). \tag{D.3}$$

A capacitor is generally used to integrate currents over time.

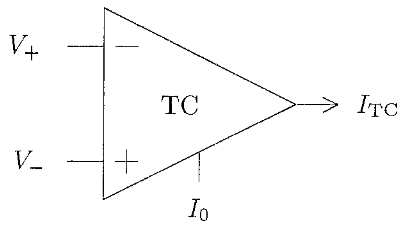


Figure D.4. A transconductance amplifier

A transconductance amplifier is often modeled as generating a current on its output that depends on the inputs to the amplifier (see figure D.4). The current generated is

$$I_{\text{TC}} = I_0 \tanh \alpha(V_+ - V_-), \tag{D.4}$$

where I_0 is determined by an external input to the amplifier. If V_- and V_+ are near each other, then the output of the amplifier can be linearized to be [Mead]

$$I_{\text{TC}} = I_0\alpha(V_+ - V_-). \tag{D.5}$$

Other circuit components do not generate a simple current that depends on voltage. Rather, they are modeled as being so powerful that the voltage on their output node is completely determined by the circuit component.

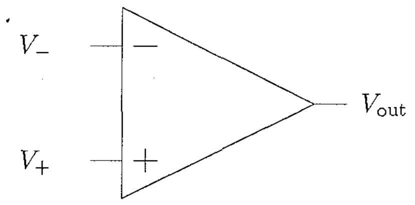


Figure D.5. An operational amplifier

An example is an operational amplifier, whose output is a steep non-linear function of its two inputs:

$$V_{\text{out}} = V_{\text{pow}} \tanh \beta(V_+ - V_-) + V_{\text{inv}} \tag{D.6}$$

where V_{pow} is half the difference of the power supply voltages, V_{inv} is the average of the power supply voltages, and β is an extremely large number, usually around 10,000. The gain of the operational amplifier is so large that it makes feedback circuits particularly easy to analyze.

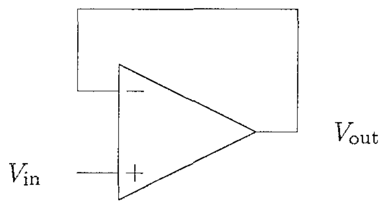


Figure D.6. A voltage follower created from an operational amplifier

An example of a feedback circuit is a voltage follower, shown in figure D.6. In order for the feedback system to be consistent, the output voltage automatically lies near the input voltage:

$$V_{\text{out}} \approx V_{\text{in}} \tag{D.7}$$

Another type of amplifier is an inverter (see figure D.7), which has a transfer function similar to that in equation (D.6):

$$V_{\text{out}} = V_{\text{pow}} \tanh \beta(V_{\text{in}} - V_{\text{inv}}) + V_{\text{inv}} \tag{D.8}$$

except that β is negative, usually around -10.

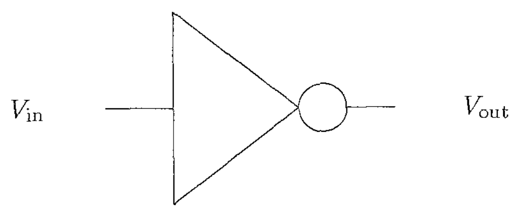


Figure D.7. An inverter

Appendix E : References

Chapter 1: Introduction

Armstrong, W.W., Green, M., [1985], "The Dynamics of Articulated Rigid Bodies for Purposes of Animation," *Proc. Graphics Interface '85*, Montreal, Canada, 407–415.

Arrow, K., Hurwicz, L., Uzawa, H., [1958], *Studies in Linear Nonlinear Programming*, Stanford University Press, Stanford, CA.

Badler, N.I., [1986], "The Design of a Human Movement Representation Incorporating Dynamics," *Advances in Computer Graphics I*, G. Enderle, M. Grave, F. Lillehagen (eds.), Springer–Verlag, 499-512.

Barr, A., [1984], "Global and Local Deformations of Solid Primitives," *Computer Graphics*, **18**, 3, (Proc. SIGGRAPH) 21-29.

Barr, A., [1988], "Teleological Modeling," *Developments in Physically Based Modeling, 1988 ACM SIGGRAPH Tutorial 27 Notes*, ACM SIGGRAPH.

Barzel, R., Barr, A., [1987], "Modeling with Dynamic Constraints," *Topics in Physically Based Modeling, 1987 ACM SIGGRAPH Tutorial 17 Notes*, ACM SIGGRAPH.

Borning, A.H., [1979], Thinglab – A Constraint-oriented Simulation Laboratory, Xerox PARC, SSL-79-3.

Cohen, M.A., Grossberg, S., [1983], "Absolute Stability of Global Pattern Formation and Parallel Memory Storage by Competitive Neural Networks," *IEEE Trans. Systems, Man, Cybernetics*, **SMC-13**, 5, 815–826.

Durbin, R., Willshaw, D., [1987], "An Analogue Approach to the Travelling Salesman Problem Using an Elastic Net Method," *Nature*, **326**, 689–691.

Feynman, C.R., [1986], Modeling the Appearance of Cloth, MSc thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA.

Fournier, A., Fussel, D., Carpenter, L., [1982], "Computer Rendering of Stochastic Models," *CACM*, **25**, 6, 371–384.

Fournier, A., Reeves, W.T., [1986], "A Simple Model for Ocean Waves," *Computer Graphics*, **20**, 4, 75–84.

Fung, Y.C., [1965], *Foundations of Solid Mechanics*, Prentice–Hall, Englewood Cliffs, NJ.

Gill, P.E., Murray, W., Wright, M.H., [1981], *Practical Optimization*, Academic Press, London.

Girard, M., Maciejewski, A., [1985], "Computational Modelling for the Computer Animation of Legged Figures," *Computer Graphics*, **19**, 4.

Grossberg, S., [1973], "Contour Enhancement, Short Term Memory, and Constancies in Reverberating Neural Networks," *Studies in Applied Mathematics*, **LII**, 213–257.

Hopfield, J.J., [1984], "Neurons with Graded Response Have Collective Computational Properties like those of Two-State Neurons," *PNAS*, **81**, 3088–3092.

Hopfield, J.J., Tank, D.W., [1985], "'Neural' Computation of Decisions in Optimization Problems," *Biol. Cyber.*, **52**, 141–152.

Horn, B.K.P., [1975], "Obtaining Shape from Shading Information," *The Psychology of Computer Vision*, P.H. Winston (ed.), McGraw-Hill.

Isaacs, P., Cohen, M., [1987], "Controlling Dynamic Simulation with Kinematic Constraints, Behavior Functions and Inverse Dynamics," *Computer Graphics*, **21**, 4, (Proc. SIGGRAPH) 215–224.

Kajiya, J.T., [1986], "The Rendering Equation," *Computer Graphics*, **20**, 4, (Proc. SIGGRAPH) 143–150.

Kass, M., Witkin, A., Terzopoulos, D., [1987], "Snakes: Active Contour Models," *Proc. 1st International Conf. on Computer Vision.*

Koch, C., Luo, J., Mead, C., Hutchinson, J., [1986], "Computing Motion Using Resistive Elements," *IEEE Conf. Neural Information Processing Systems* , Denver.

LaSalle, J., [1976], *The Stability of Dynamical Systems*, SIAM, Philadelphia.

Luenberger, D.G., [1973], *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Menlo Park, California.

Lundin, D., [1987], "Ruminations of a Model Maker," *IEEE Computer Graphics and Applications*, **7**, 5, 3–5.

Mead, C.A., Conway, L., [1980], *Introduction to VLSI Systems*, Addison-Wesley, Reading, MA.

Mead, C.A., [1989], *Analog VLSI and Neural Systems*, Addison-Wesley, Reading, MA.

Nikravesh, P.E., [1988], *Computer-Aided Analysis of Mechanical Systems*, Prentice-Hall, Englewood Cliffs, New Jersey.

Platt, J.C., Barr, A., [1987], "Constrained Differential Optimization," *1987 Neural Infor-*

mation and Processing Systems Conference.

Platt, J.C., Barr, A., [1988], "Constraint Methods for Flexible Models," *Computer Graphics*, **22**, 4, 279–288.

Platt, J.C., Hopfield, J.J., [1986], "Analog Decoding with Neural Networks," *Neural Networks for Computing, AIP Conf. Proc. 151*, Snowbird, UT, 364–369.

Poggio, T., Torre, V., [1984], "Ill-posed Problems and Regularization Analysis in Early Vision," *Proc. DARPA Image Understanding Workshop*, New Orleans, L.S. Baumann (ed.), 257–263.

Reeves, W.T., Blau, R., "Approximate and Probabilistic Algorithms for Shading and Rendering Structured Particle Systems," *Computer Graphics*, **19**, 3, 313–322.

Rumelhart, D.E., Hinton, G.E., Williams, R.J., [1986], "Learning Internal Representations by Error Propagation," *Parallel Distributed Processing*, Vol. 1, 318–362.

Sederberg, T.W., Parry, S.R., [1986], "Free-form Deformation of Solid Geometric Models," *Computer Graphics*, **20**, 4, (Proc. SIGGRAPH), 151–160.

Seltzer, D., [1984], Representation and Control of Three Dimensional Computer Animated Figures, Department of Computer and Information Science, Ohio State University.

Tanner, J.E., [1986], Integrated Optical Motion Detection, Ph.D. thesis, Department of Computer Science, California Institute of Technology.

Tank, D.W., Hopfield, J.J., [1986], "Simple Optimization Networks: An A/D Converter and a Linear Programming Circuit," *IEEE Trans. Cir. & Syst.*, **CAS-33**, 5, 533-541.

Terzopoulos, D., [1983], "Multilevel Computational Processes for Visual Surface Reconstruction," *Computer Vision, Graphics, and Image Processing*, **24**, 52–96.

Terzopoulos, D., [1987], "On Matching Deformable Models to Images: Direct and Iterative Solutions," *Topical Meeting on Machine Vision, Technical Digest Series, Vol. 12., Optical Society of America*, Washington, DC, 160–167.

Terzopoulos, D., Platt, J.C., Barr, A., Fleischer, K., [1987], "Elastically Deformable Models," *Computer Graphics*, **21**, 4, 205–214.

Truesdell, C., [1965], "The Non-Linear Field Theory of Mechanics," *Encyclopedia of Physics*, Vol. III/3, S. Flügge (ed.), Springer-Verlag, Berlin.

Ullman, S., [1979], *The Interpretation of Visual Motion*, MIT Press, Cambridge, MA.

Weil, J., [1986], "The Synthesis of Cloth Objects," *Computer Graphics*, **20**, 4, (Proc. SIGGRAPH), 49–54.

Wilhelms, J., Barsky, B.A., [1985], "Using Dynamic Analysis to Animate Articulated Bodies such as Humans and Robots," *Proc. Graphics Interface '85*, Montreal, Canada, 97–104.

Wilson, H.R., Cowan, J.D., [1972], "Excitatory and Inhibitory Interaction in Localized Populations of Model Neurons," *Biophysical Journal*, **12**, 1.

Witkin, A., Fleischer, K., Barr, A., [1987], "Energy Constraints on Parametrized Models," *Computer Graphics*, **21**, 4, 225-232.

Chapter 2: Constrained Optimization Methods

Arrow, K., Hurwicz, L., Uzawa H., [1958], *Studies in Linear Nonlinear Programming*, Stanford University Press, Stanford, CA.

Bertsekas, D., [1976], "Multiplier Methods: a Survey," *Automatica*, **12**, 133–145.

Foulds, L.R., [1981], *Optimization Techniques*, Springer-Verlag, New York.

Gill, P.E., Murray, W., Wright, M.H., [1981], *Practical Optimization*, Academic Press, London.

Hestenes, M., [1975], *Optimization Theory*, Wiley & Sons, New York.

Hopfield, J.J., Tank, D.W., [1985], "'Neural' Computation of Decisions in Optimization Problems," *Biol. Cyber.*, **52**, 141–152.

Koch, C., Luo, J., Mead, C., Hutchinson, J., [1986], "Computing Motion Using Resistive Elements," *IEEE Conf. Neural Information Processing Systems*, Denver.

Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.G., [1985], *The Traveling Salesman Problem*, John Wiley & Sons, Chichester, England.

Nayfeh, A.H., [1981], *Introduction to Perturbation Techniques*, John Wiley & Sons, New York.

Nikravesh, P.E., [1988], *Computer-Aided Analysis of Mechanical Systems*, Prentice-Hall, Englewood Cliffs, New Jersey.

Platt, J.C., Barr, A., [1987], "Constrained Differential Optimization," *1987 Neural Information and Processing Systems Conference.*

Platt, J.C., Hopfield, J.J., [1986], "Analog Decoding with Neural Networks," *Neural Networks for Computing, AIP Conf. Proc. 151,* Snowbird, UT, 364–369.

Poggio, T., Torre, V., [1984], "Ill-posed Problems and Regularization Analysis in Early Vision," *Proc. DARPA Image Understanding Workshop*, New Orleans, L.S. Baumann (ed.), 257–263.

Press, W., Flannery, B., Teukolsky, S., Vetterling W., [1986], *Numerical Recipes*, Cambridge University Press, Cambridge.

Chapter 3: Constrained Circuits

Ambrahams, R.H., Shaw, C.D., [1984], *Dynamics — The Geometry of Behavior. Part 3: Global Behavior*, Aerial Press, Santa Cruz, CA.

Gill, P.E., Murray, W., Wright, M.H., [1981], *Practical Optimization*, Academic Press, London.

Guckenheimer, J., Holmes, P., [1983], *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields*, Springer-Verlag, New York.

Koch, C., Luo, J., Mead, C., Hutchinson, J., [1986], "Computing Motion Using Resistive Elements," *IEEE Conf. Neural Information Processing Systems*, Denver.

Sivilotti, M.A., Emerling, M.R., Mead, C.A., [1986], "VLSI Architectures for Implementation of Neural Networks," *Neural Networks for Computing, AIP Conf. Proc. 151*, Snowbird, UT.

Tanner, J.E., [1986], Integrated Optical Motion Detection, Ph.D. thesis, Department of Computer Science, California Institute of Technology.

Chapter 4: Analog Decoding

Gill, P.E., Murray, W., Wright, M.H., [1981], *Practical Optimization*, Academic Press, London.

McEliece, R.J., [1977], *The Theory of Information and Coding*, Addison-Wesley, London, 103–116.

Papadimiriou, C.H., Steiglitz, K., [1982], *Combinatorial Optimization: Algorithms and Complexity*, Prentice–Hall, Englewood Cliffs, NJ, 248.

Platt, J.C., Hopfield, J.J., [1986], "Analog Decoding with Neural Networks," *Neural Networks for Computing, AIP Conf. Proc. 151*, Snowbird, UT, 364–369.

Yuen, J.H., [1982], Deep Space Telecommunications Systems Engineering, JPL, Publication 82-76.

Chapter 5: Constrained Active Splines

Burr, D., [1981], "Elastic Matching of Line Drawings," *IEEE Trans. PAMI*, **PAMI-3**, 6, 708–713.

Courant, R., Hilbert, D., [1953], *Methods of Mathematical Physics*, Vol. I, Interscience, London.

Durbin, R., Willshaw, D., [1987], "An Analogue Approach to the Travelling Salesman Problem using an Elastic Net Method," *Nature*, **326**, 689–691.

Golden, B.L., Stewart, W.R., [1985], "Empirical Analysis of Heuristics," *The Traveling Salesman Problem*, Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.), Wiley-Interscience, Chichester.

Hopfield, J.J., Tank, D.W., [1985], "'Neural' Computation of Decisions in Optimization Problems," *Biol. Cyber.*, **52**, 141–152.

Johnson, D.S., Papadimitriou, C.H., [1985], "Performance Guarantees for Heuristics," *The Traveling Salesman Problem*, Lawler, E.L., Lenstra, J.K., Rinnooy Kan, A.H.G., Shmoys, D.B. (eds.), Wiley-Interscience, Chichester.

Kass, M., Witkin, A., Terzopoulos, D., [1987], "Snakes: Active Contour Models," *Proc. 1st International Conf. on Computer Vision.*

Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P., [1983], "Optimization by Simulated Annealing," *Science*, **220**, 671–680.

Koch, C., Luo, J., Mead, C., Hutchinson, J., [1986], "Computing Motion Using Resistive Elements," *IEEE Conf. Neural Information Processing Systems*, Denver.

Kohonen, T., [1987], *Self-Organization and Associative Memory; Second Edition*, Springer-Verlag, Berlin.

Lin, S., Kernighan, B.W., "An Effective Heuristic Algorithm for the Traveling-Salesman Problem," *Operations Research*, **11**, 972–989.

Mead, C.A., [1989], *Analog VLSI and Neural Systems*, Addison-Wesley, Reading, MA.

Papadimiriou, C.H., Steiglitz, K., [1982], *Combinatorial Optimization: Algorithms and Complexity*, Prentice–Hall, Englewood Cliffs, NJ, 248.

Press, W., Flannery, B., Teukolsky, S., Vetterling W., [1986], *Numerical Recipes*, Cambridge University Press, Cambridge.

Sivilotti, M.A., Emerling, M.R., Mead, C.A., [1986], "VLSI Architectures for Implementation of Neural Networks," *Neural Networks for Computing, AIP Conf. Proc. 151*, Snowbird, UT.

Sivilotti, M.A., Mahowald, M.A., Mead, C.A., [1987], "Real-time Visual Computations using Analog CMOS Processing Arrays," *Advanced Research in VLSI: Proceedings of the 1987 Stanford Conference*, P. Losleben (ed.), 295–312.

Terzopoulos, D., Witkin, A., Kass, M., [1988], "Constraints on Deformable Models: Recovering 3D Shape and Non-rigid Motion," *Artificial Intelligence*, **36**, 91–123.

Chapter 6: Deformable Physically-based Models

Armstrong, W.W., Green, M., [1985], "The Dynamics of Articulated Rigid Bodies for Purposes of Animation," *Proc. Graphics Interface '85*, Montreal, Canada, 407–415.

Barr, A., [1983], Geometric Modeling and Fluid Dynamic Analysis of Swimming Spermatozoa, Ph.D. thesis, Department of Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY.

Barzel, R., Barr, A., [1987], "Modeling with Dynamic Constraints," *Topics in Physically Based Modeling, 1987 ACM SIGGRAPH Tutorial 17 Notes*, ACM SIGGRAPH.

Courant, R., Hilbert, D., [1953], *Methods of Mathematical Physics*, Vol. I, Interscience, London.

do Carmo, M.P., [1974], *Differential Geometry of Curves and Surfaces*, Prentice–Hall, Englewood Cliffs, NJ.

Faux, J.D., Pratt, M.J., [1981], *Computational Geometry for Design and Manufacture*, Halstead Press, Horwood, NY.

Fung, Y.C., [1965], *Foundations of Solid Mechanics*, Prentice–Hall, Englewood Cliffs, NJ.

Gelfand, I.M., Fomin, S.V., [1963], *Calculus of Variations*, Prentice–Hall, Englewood Cliffs, NJ.

Goldstein, H., [1950], *Classical Mechanics*, Addison-Wesley, Reading, MA.

Haumann, D., [1988], "Animating Using Behavioral Simulation," *1988 National Computer Graphics Association*, Vol. 3, 672–680.

Landau, L.D., Lifshitz, E.M., [1959], *Theory of Elasticity*, Pergamon Press, London, UK.

Lapidus, L., Pinder, G.F., [1982], *Numerical Solution of Partial Differential Equations in Science and Engineering*, Wiley, New York, NY.

Terzopoulos, D., Platt, J.C., Barr, A., Fleischer, K., [1987], "Elastically Deformable Models," *Computer Graphics*, **21**, 4, 205–214.

White, R., [1985], *An Introduction to the Finite Element Method with Applications to Nonlinear Problems*, John Wiley & Sons, New York.

Wilhelms, J., Barsky, B.A., [1985], "Using Dynamic Analysis to Animate Articulated Bodies such as Humans and Robots," *Proc. Graphics Interface '85*, Montreal, Canada, 97–104.

Chapter 7: Constraint Methods for Physical Systems

Barr, A., [1988], "Teleological Modeling," *Developments in Physically Based Modeling, 1988 ACM SIGGRAPH Tutorial 27 Notes*, ACM SIGGRAPH.

Barzel, R., Barr, A., [1987], "Modeling with Dynamic Constraints," *Topics in Physically Based Modeling, 1987 ACM SIGGRAPH Tutorial 17 Notes*, ACM SIGGRAPH.

Barzel, R., Barr, A., [1989], Solving Piecewise-Continuous Ordinary Differential Equations, to appear.

Baumgarte, J., [1972], "Stabilization of Constraints and Integrals of Motion," *Computational Methods Apple. Mech. Eng.*, **1**, 1–16.

Boland, P., Samin, J.C., Willems, P.Y., [1974], "On the Stability of Interconnected Deformable Bodies in a Topological Tree," *AIAA J.*, **12**, 1025–1030.

Gill, P.E., Murray, W., Wright, M.H., [1981], *Practical Optimization*, Academic Press, London.

Goldstein, H., [1950], *Classical Mechanics*, Addison-Wesley, Reading, MA.

Hestenes, M., [1975], *Optimization Theory*, Wiley & Sons, New York.

Terzopoulos, D., Platt, J.C., Barr, A., Fleischer, K., [1987], "Elastically Deformable Models," *Computer Graphics*, **21**, 4, 205–214.

Witkin, A., Fleischer, K., Barr, A., [1987], "Energy Constraints on Parametrized Models," *Computer Graphics*, **21**, 4, 225-232.

Wittenburg, J., [1977], *Dynamics of Systems of Rigid Bodies*, Teubner, Stuttgart.

Chapter 8: Constraints for the Animation of Flexible Models

Barzel, R., Barr, A., [1989], Solving Piecewise-Continuous Ordinary Differential Equations, to appear.

Blinn, J., "The Mechanical Universe: An Integrated View of a Large-Scale Animation Project," *1987 ACM SIGGRAPH Tutorial 6 Course Notes*, ACM SIGGRAPH.

do Carmo, M.P., [1974], *Differential Geometry of Curves and Surfaces*, Prentice–Hall, Englewood Cliffs, NJ.

Feynman, R.P., Leighton, R.B., Sands, M., [1963], *Feynman Lectures on Physics*, Addison-Wesley, Readming, MA.

Fung, Y.C., [1965], *Foundations of Solid Mechanics*, Prentice–Hall, Englewood Cliffs, NJ.

Lasseter, J., "Principles of Traditional Animation Applied to 3D Computer Animation," *Computer Graphics*, **21**, 4, 35–44.

Moore, M., Wilhelms, J., "Collision Detection and Response for Computer Animation," *Computer Graphics*, **22**, 4, 289–298.

Platt, J.C., Barr, A., [1988], "Constraint Methods for Flexible Models," *Computer Graphics*, **22**, 4, 279-288.

Terzopoulos, D., [1986], "Regularization of Inverse Visual Problems Involving Discontinuities," *IEEE Trans. Pattern Analysis and Machine Intelligence*, **PAMI-8**, 413–424.

Chapter 9: Conclusions

Hackbusch, W., [1985], *Multi-Grid Methods and Applications*, Springer-Verlag, Berlin.

Kohonen, T., [1987], *Self-Organization and Associative Memory; Second Edition*, Springer-Verlag, Berlin.

Terzopoulos, D., Witkin, A., [1988], "Physically-based Models with Rigid and Deformable Components," *IEEE Computer Graphics & Applications*, **8**, 6, 41–51.

Appendix A: Numerical Techniques

Dahlquist, G., Bjorck, A., [1974], *Numerical Methods*, Prentice–Hall, Englewood Cliffs, NJ.

Gear, C.W., [1971], *Numerical Initial Value Problems in Ordinary Differential Equations*, Prentice–Hall, Englewood Cliffs, NJ.

Gill, P.E., Murray, W., Wright, M.H., [1981], *Practical Optimization*, Academic Press, London.

Goldstein, H., [1950], *Classical Mechanics*, Addison-Wesley, Reading, MA.

Petzold, L., [1982], A Description of DASSL: A Differential/Algebraic System Solver, Sandia National Laboratories, Livermore, CA, SAND82-8637.

Press, W., Flannery, B., Teukolsky, S., Vetterling W., [1986], *Numerical Recipes*, Cambridge University Press, Cambridge.

Stoer, J., Bulirsch, R., [1980], *Introduction to Numerical Analysis*, Springer-Verlag, New York.

Terzopoulos, D., Platt, J.C., Barr, A., Fleischer, K., [1987], "Elastically Deformable Models," *Computer Graphics*, **21**, 4, 205–214.

Truesdell, C., [1965], "The Non-Linear Field Theory of Mechanics," *Encyclopedia of Physics*, Vol. III/3, S. Flügge (ed.), Springer-Verlag, Berlin.

Zienkiewicz, O., [1977], *The Finite Element Method*, McGraw-Hill, London.

Appendix B: Convergence of the Differential Multiplier Method

Arrow, K., Hurwicz, L., Uzawa H., [1958], *Studies in Linear Nonlinear Programming*, Stanford University Press, Stanford, CA.

Luenberger, D.G., [1973], *Introduction to Linear and Nonlinear Programming*, Addison-Wesley, Menlo Park, California.

Appendix C: Calculus of Variations

Gelfand, I.M., Fomin, S.V., [1963], *Calculus of Variations*, Prentice–Hall, Englewood Cliffs, NJ.

Appendix D: An Introduction to Circuit Theory

Horowitz, H., Hill, W., [1980], *The Art of Electronics*, Cambridge University Press, Cambridge, England.

Mead, C.A., [1989], *Analog VLSI and Neural Systems*, Addison-Wesley, Reading, MA.