3. Physically Based Modeling

Physically based modeling uses physical simulation as a modeling and animating tool. Since they are based on the fundamental Newtonian laws of mechanics, and simulated by solving differential equations, physically-based models can generate a rich variety of realistic or surrealistic behavior which can be controlled by the animator through intuitive physical parameters and constraint techniques. With the latest high-speed workstations, they can be simulated in real time, allowing interactive control. Examples of physically-based models in the computer graphics literature can be broadly classified into three types: particle systems, continuous systems and rigid body systems.

3.1. Introduction

Physically-based modeling is a relatively new field within computer graphics. It was introduced into the literature in the late 1980's to apply techniques commonly used in physics and engineering disciplines to the subject of computer graphics modeling and animation. Although the techniques themselves are fairly well established in physics and engineering simulation and robotics, what is new about physically-based modeling is the way in which they are applied. For the remainder of the chapter, we will give a detailed review of physically-based modeling with particular attention as to how it can best be applied to 3D character animation.

3.1.1. Definition

Physically-based modeling in computer graphics is the simulation of physical systems for the purposes of creating (usually animated) computer graphics images. As such, the physics is just a tool, another trick for creating interesting or appealing computer graphics. It does not particularly matter if the physical model is "correct" for the given situation, as long as it gives desirable results. In particular, it does not particularly matter if the physics is simulated accurately, as long as it looks OK and is reproducible. physically based modeling is therefore somewhat of a hybrid field: part physics, part numerical programming and part computer graphics. By combining these various elements in the right ways, realistic-*looking* or (surrealistic-looking) computer graphics animation can be achieved.

Physically-based modeling is not physical simulation. In physical simulation, which is a separate discipline in itself, scientists try to gain more insight into the physical world. The stress is on accuracy, prediction of the real world and the discovery of new phenomena, for example, simulating the collision of two galaxies, fluid flow, or the fracture of a metal plate. Another type of physical simulation is used for engineering purposes to simulate the behavior of machines such as airplanes or cars, and is closer to physically based modeling because it often uses computer graphics and requires real-time solution. What matters in physically-based modeling is whether it looks good, not whether the physical model is correct or numerically accurate. Neither is physically based modeling scientific visualization, in which experimental or simulation data is rendered in various ways to try to gain more physical insight.

The term physically based modeling, as used within the computer graphics community, is not generally applied to a variety of rendering techniques which nonetheless use physical simulation of light to achieve a natural appearance. Ray-tracing and radiosity methods fall into this category, and in a broad sense they should be considered a physically based modeling technique, but historically the term usually applies to classical mechanics, not optics or radiation.

3.1.2. Physics

The "physics" in physically based modeling is, therefore, usually Newtonian classical mechanics. The objects to be displayed are themselves usually collections of masses, either free-moving or confined in rigid or flexible bodies. Although the masses themselves follow fairly simple, Newtonian rules of behavior, the forces acting between them can be of wide-ranging complexity, resulting from various types of fundamental and phenomenological force, depending on the type of physical model.

In particular, these force laws may not represent any fundamental forces of nature, but rather phenomenological forces, that is, approximations to the actual forces resulting from very complex physical processes. We can see already that we are beginning to deal not with exact mathematical representations of physical reality but rather with approximations. This is of course true for most physical and engineering calculations, but it is especially true in physically based modeling because the only real arbiter of physical accuracy in this case is our eyes. So we are, in a way, really in the business of creating sort of physical Potemkin villages, which give the appearance of physical reality only from the restricted viewpoint of the virtual computer graphics camera.

We are helped in this endeavor by a curious fact in the study of physics: physical laws and behavior from widely divergent fields and varying scales are often very similar. It is therefore often possible to make a very simple physical model which displays qualitative behavior similar to a much more complicated one. In physically based modeling we are very often not modeling what we think we are seeing. What looks like exquisitely detailed flowing water may only be a handful of mass particles rendered using implicit surfaces.

From the physically based modeling point of view, therefore, physical simulation is little more than just another algorithm for generating interesting computer graphics and animation behavior. Because the algorithm is rooted in physical law, there is a better chance that we will get visually appealing results. Perhaps just as important, however, is that the parameters by which we control the model are often, given our human experience with the real world, more intuitive and natural than for non-physically-based models. By pushing these parameters to their extremes, or choosing deliberately unrealistic ones, we can generate behavior that caricatures natural behavior, something I call surrealistic behavior. This is a very common element of traditional animation and one which physically-based modeling can support very well.

The principal disadvantage of physically based modeling is one of control. Since the model tends to behave naturally, it tends to do what it wants, not what you want. One of the most important issues in the field of physically based modeling is therefore how to control the models themselves. At a basic level, the animator can usually just tune the models raw parameters, adjusting values of friction, mass, force fields, for example, until some global desired behavior is achieved. Ultimately, however, the animator will want to make certain objects follow predetermined paths, perhaps interactively, and reach certain positions at prearranged times. There is, in other words, still a need for some elements of kinematic control.

This control is accomplished through use of constraints, which force the physically based modeling to conform to certain geometries or paths or to perform a certain task, while still behaving in a physical way. The animator might, for instance, force one point on the model to follow a spline path in space so that all the other points attached to it will follow along in a physically natural way. The result is a computer animation that does what the animator specifies and, where the animator doesn't specify, does what is physically realistic. This goal of natural looking, controlled behavior is an ideal. For the rest of this chapter we will discuss the theory and techniques by which we try to attain this goal.

3.1.3. Numerical Programming

Theoretical physical models are the basis of physically based modeling, but how to simulate them on a computer is another subject, which falls under the domain of numerical programming. This field, one of the oldest in computer science, is concerned with finding numerical solutions to pure mathematical problems, using the imperfect medium of floating point numbers in a computer. Although numerical programming covers many topics, such as solving linear systems, finding roots of equations, and evaluating transcendental functions, the most important one for physically based modeling is the solutions of differential equations. Since most physical laws, and our models, can be expressed in the form of a differential equation, implementing a physically based modeling requires solving this differential equation numerically.

Traditionally, the emphasis in numerical differential equation solving has been on solving highly specific problems in science and engineering. The stress has been on accuracy and trying to predict actual behavior as closely as possible. In the early days of computing, CPU power was a precious resource and, since programs were run as batch jobs, there was rarely any chance to interact with a simulation once it was started.

For physically based modeling, the emphasis is less on accuracy and more on speed. Since CPUs are now cheap, an individual user can afford to sit at a single-user workstation and interact with a numerical solution as it happens. By observing the results and deciding what "looks good", he can adjust the parameters of the solution to obtain a proper compromise between speed and accuracy. Furthermore, a user can actually construct his model interactively, using the physical behavior as a modeling tool. Since computer graphics programmers themselves do not usually have the time to develop complex numerical programming strategies, the techniques need to be straightforward and easy to comprehend, or at least readily available in well-documented "black box" routine packages. Since the goal is to create a variety of different animation with the same software, the solution techniques need to be fairly general and applicable over a wide range of model parameters.

3.1.4. Computer Graphics

The third, and perhaps most important component of physically based modeling is computer graphics. This field provides not only the means of rendering our models, but also the techniques for interacting with them as they evolve. Computer graphics also provides the sorts of software techniques and data structures which allow complex physically-based *models* (rather than simply the numerical techniques) to be constructed.

physically based modeling can, in fact, be thought of as a natural evolution of computer graphics modeling techniques. Just as rendering has progressed over the last twenty years from wireframe representations to hidden-surface polygons to shaded polygons to ray-tracing and radiosity methods, so have graphics modeling methods evolved from rigid geometrical models to kinematically animated to dynamic, physically-based models to constraint-based models. This last category is referred to by Al Barr as *teleological* modeling [Barzel 92], in which higher-level goals, as opposed to physical values, are specified by the animator.

3.2. Classical Mechanics

The "physics" in physically based modeling is based on classical mechanics. Fortunately, this is a highly developed subject, much of the groundwork having been done in the 17th and 18th centuries by scientists such as Newton, Euler, and Laplace. Since physically based modeling usually tries to simulate objects with which we are familiar in our daily experience, there is little need for using modern quantum or relativistic physics. There can be some use for other areas of classical physics such as thermodynamics or electromagnetism, but they can generally be treated in our models as phenomenological external forces. We briefly review the basic elements of classical mechanics for purposes of physically based modeling. For a more complete and insightful introduction to the subject, see the excellent text by Richard Feynman [Feynman 63].

3.2.1. Definitions

Unlike computer programs and mathematics, which deal only in pure numerical quantities, most measurements in physics are made with respect to fundamental quantities or units. We start with definitions of a few fundamental physical quantities which we define as axioms, not unlike the axioms of mathematics, from which we can derive all other physical quantities. Like the axioms of mathematics, the choice of which quantities are fundamental and which are derived is somewhat arbitrary. Unlike mathematics, these axioms are based on real experience and have stood the test of experimental verification. Physical quantities may be constructs of the human mind, but they are based on something real that exists "out there".

We choose as our fundamental quantities a unit of time, a unit of space or distance, and a unit of matter or mass. These can be summarized as follows:

$$
\begin{aligned}
m &= \text{mass} \\
\vec{x} &= \text{position} \\
t &= \text{time} \\
\vec{v} &= \frac{d\vec{x}}{dt} = \text{velocity} \\
\vec{a} &= \frac{d\vec{v}}{dt} = \frac{d^2\vec{x}}{dt^2} = \text{acceleration} \\
\vec{f} &= \text{force}
\end{aligned}
\tag{3.1}
$$

The first of these quantities, distance, is measured in meters and in its most general form is a three-dimensional vector in Cartesian space which we will usually denote r. Models which contain only distances, without any reference to other physical quantities, can be considered geometric models, and the study of such objects can be considered the subject of geometry. Geometric modeling can be considered the first step in the evolution of computer graphics modeling described by Barr.

3.2.2. Kinematics

By adding the quantity time, measured in seconds, to our geometric analysis, we enter the subject of kinematics. Kinematics is the study of pure motion in space, without regard to its physical causes. With kinematics we define new derived quantities, velocity and acceleration. Later, when we discuss rigid body motion, we will define additional kinematic quantities. Kinematic modeling forms the second step in the evolution of computer graphics models and is still the basis for most computer animation techniques.

3.2.3. Newton's Laws: Dynamics

By adding mass, measured in kilograms, to kinematics, we form the subject of dynamics. Dynamics describes why physical systems move in certain ways, not just how, and allows us to predict the behavior of a physical system. The things that make objects move, according to dynamics, are forces, and the law which relates force to motion is Newton's famous second law:

$$\vec{f} = m\vec{a}$$

This states, that a thing called force causes an object to accelerate and the amount of this acceleration is proportional to the force and inversely proportional to the mass. This law defines a new type of derived quantity, force, with units of kilogram-meters per second squared. From a certain point of view, this equation is no more than an additional definition: force is simply the product of mass times acceleration, just as acceleration is the derivative of position with respect to time. What elevates this to a law, and not merely a definition is the fact that forces seem to have some objective existence , in other words, force seems to be a fundamental quantity in the real world. This is important not only because we have an intuitive sense of what a force is (it is something we can exert with our muscles, for example) but also because forces are the principle mechanism by which the components of physical systems interact with each other, and therefore form a kind of common currency in constructing the components of our physically based models.

Newton, who used his theories to explain the motions of the planets, described his forces in terms of point masses: particles whose mass was concentrated at an infinitesimal point. Although this is a considerable idealization of the real world, it turned out to work quite well for celestial mechanics. It also turned out, once the atomic theory of matter became accepted, to work well for modeling the microscopic world under certain circumstances. Under other circumstances, however, Newtonian physics does not work at all, and it eventually became apparent that a new theory, quantum theory was necessary. From the point of view of physically based modeling, however, we will find that a Newtonian-based atomic theory of matter can work quite well in building our models, and point masses are the starting point for most of our physically-based models. Newton therefore described his forces as acting on and between point masses, and in his third law of motion he states that "for every action there is an equal and opposite reaction." In other words, forces come in pairs. Whenever two particles interact, the force exerted by the first particle on the second is exactly the negative of the force exerted by the second on the first. If one particle is much more massive than the other (e.g. the sun vs. the earth), however, the force acting on the massive particle will cause virtually no acceleration compared to the first, and can usually be ignored.

In this last case, and particularly when there are many or an infinite number of particles involved, it can be more convenient to describe forces in terms of fields. Rather than considering a force to be between two points, we break up the force calculation into two stages. First, we say that a single particle or group of particles induces a *force field* in a given region of space. This field, which is a vector function of three-dimensional space, is then said to exert a force on any mass point within it. Whether this field has any objective reality or not is somewhat of a murky point, but it can be of considerable conceptual and practical utility since, rather than doing individual calculations of every mass point with each other, a single field can be calculated for the entire system, from which the force acting on each particle can then be calculated.

3.2.4. Conservation Laws

The description of classical mechanics by means of space, time, mass and forces is actually complete and it is in most cases sufficient for doing physically based modeling because our equations are usually force-based and solved numerically. There are other ways of approaching the problem, however, which the physicists have developed to gain more physical insight into the subject and to be able to make more general statements about a physical system then simply how it behaves over time.

One of these other approaches is through conservation laws. In particular, it is possible to define other derived quantities, besides force, which seem to be fundamental in character. They are so fundamental in fact that they seem to represent a finite quantity of something whose amount stays fixed at all times. To be more precise, given an isolated system which has no interaction with the outside world, certain quantities can be measured of the system as a whole which will not change, regardless of the behavior of the system. Such quantities are said to be *conserved*.

The first of these conserved quantities is momentum, which is defined to be the product of mass times velocity. By taking the derivative of this definition and comparing it to Newton's second law, we can see that force is the derivative of momentum with respect to time. Forces therefore have the effect of adding or removing momentum from a system and can be thought of as *flow of momentum*. By reexamining Newton's third law, we can see that, since forces always come in equal and opposite pairs, momentum may shift around from particle to particle, but *momentum will always be conserved*. Momentum therefore seems to have a very real existence, almost like a fluid, which can flow around from point to point but which is never created or destroyed.

$$\vec{p} = m\vec{v} = \text{momentum}$$
$$\vec{f} = \frac{d\vec{p}}{dt} = \text{flow of momentum}$$
$$\sum \vec{p}_i = \text{constant} \tag{3.2}$$

The second conserved quantity we will discuss is energy. For a moving particle, energy is defined as one half the mass times the velocity squared and is measured in joules, or kilogram meters squared per second squared. However, unlike momentum, energy can come in many forms and the energy associated with a particle's motion is only one form, called kinetic energy. Other forms include heat energy, caused by friction, and potential energy, which is created when a particle moves against a conservative force. The change in potential energy is defined as the line integral of the force over distance. When a particle moves within a conservative force field, and there are no dissipative (i.e. frictional) forces, the sum of the kinetic and potential energy will always be a constant. Any loss in kinetic energy will be exactly counteracted by a gain in potential energy and vice versa. In fact, even with friction, it can be shown that the energy gained in heat is exactly equal to the lost kinetic energy.

Energy is therefore a conserved quantity, like momentum. Unlike momentum, it is a scalar, not a vector, quantity and it has the ability to disappear into obscure forms outside the realm of classical mechanics, and therefore is not always conserved in mechanical physically-based models However, as a bookkeeping device, it is often useful to monitor the total energy of our models as they are simulated over time, to check for errors and if it is not constant, at least to see how rapidly it is being dissipated.

The concept of energy also has an important use in deriving force laws, and is in fact the basis for the Lagrangian formulation of Newton's laws, and D'Alembert's principle. Since potential energy is defined as a line integral over a conservative force, and since it can be show that for a conservative force, the line integral between any two points is independent of path, every point in such a force field can be associated with a scalar potential energy value. By the line integral theorem of calculus, the force field is equal to the gradient of the potential function, so in fact, a conservative vector force field can be described in terms of a scalar potential field. This can simplify the calculation of force fields in physically based modeling and we will see that internal elastic forces can also be described more easily in terms of energy than forces.

$$
\begin{aligned}
&E = \frac{1}{2} m\vec{v}^2 = \text{ kinetic energy} \\
&E = \int \vec{f} \cdot d\vec{x} = \text{ potential energy} \\
&\textstyle\sum_i E_i = \text{constant}
\end{aligned}
\tag{3.3}
$$

Just as the position of a point in space can be completely described by its Cartesian coordinates, the dynamical "position" or state of a point mass particle can be completely described by its position and momentum vectors. We can combine these two to form a single (normally six-dimensional) state vector, and the evolution of the particle over time can then be thought of as a path in state space which is a parametric function of time.

3.3. Differential Equations

"The universe is a differential equation, God is the initial conditions"

This saying reflects the rather intimate relationship that has existed between physics and differential equations. In fact, differential equations were invented originally to solve certain physics problems, and the search for a unified field theory in physics can sometimes be characterized as "trying to find the differential equation of the universe".

3.3.1. Modeling Using Differential Equations

From the point of view of physically based modeling, differential equations are the standard mathematical technique by which a given physical model can be simulated. Generally, the behavior of a given physical system can be described by a differential equation, which can be build up directly from the model based on first principles. Normally this is done be examining the system and making certain arguments about how it should behave differentially, based on Newton's laws of motion. There are also more formal "cookbook" approaches based on Lagrange's generalized coordinates, the principle of virtual work and D'Alembert's principle. The result is normally a differential equation of forces expressed in terms of differentials of the state variables with respect to time or space.

Solutions to this differential equation are therefore state vector functions of time which describes the possible behaviors of the system. If we can find a proper solution, than we can use it to animate our physically-based model. Traditionally, physicists have only studied the behavior of limited special case models, in which the differential equations have been solvable symbolically, using analytical techniques. However, the vast majority of physical problems are not solvable in this way, and can only be solved using numerical integration techniques. Furthermore, if we want to be able to control our model interactively, for example with arbitrary force data entered by the user, we are obliged to use numerical techniques because our input data is numerical.

Exactly which solution we find depends on the initial conditions of the state variable, which can be chosen arbitrarily, and can greatly affect the outcome of the solution. One way to effect the solution, if we do not like the way it is turning out, is to add our own arbitrary forces into the equation, either interactively or according to some algorithm. In this way we can "drive" the solution of the equation over time.

3.3.2. Newtonian Differential Equation

The simplest example of a Newtonian physics-based differential equation is simply Newton's second law for a single mass particle:

$$m\frac{d^2\vec{x}}{dt^2} = \vec{f}(t) \tag{3.4}$$

This can be thought of as equating two forces: the internal forces experienced by the particle because of its acceleration on the left-hand-side, and the external forces driving the particle on the right-hand-side. The left-hand-side is a consequence of the internal structure of the model, the right-hand-side can be determined by any combination of external phenomenological forces, constraint forces, or interactive input forces.

3.4. Constraints and Forces

Starting with this simple Newtonian differential equation, more complex models can be built up by increasing the number of particles, by adding more sophisticated internal force laws, or by adding more complicated external forces.

3.4.1. Adding Internal Forces

The previous equation describes an ideal free point mass particle traveling in space without friction. This is useful for modeling certain types of phenomena, but most everyday things in our environment are not free moving and frictionless. We can attempt to simulate this behavior by adding internal forces to the model.

Frictional or damping forces are non-conservative forces which dissipate kinetic energy from the system. They are in reality very complicated forces which convert kinetic energy into heat, but they can be modeled fairly well using simple phenomenological forces. The simplest type of damping mathematically is viscous damping, which adds a force proportional to the particle's velocity:

$$m\frac{d^2\vec{x}}{dt^2}+\gamma\frac{d\vec{x}}{dt}=\vec{f}(t) \qquad (3.5)$$

This approximates the effect of putting the particle in some sort of fluid, the viscosity of which is controlled by the damping factor, γ. The damping factor controls the rate at which energy is dissipated from the system. A certain amount of damping can not only add realism to a physically-based model, but also it can help to control the stability of the model, as we shall see later.

Many objects are neither free-floating nor fixed, but rather attached to other objects elastically. Typically this will result in some type of oscillatory behavior. A simple way to model elastic behavior in a particle is to attach an ideal Hookian spring to it. A Hookian spring exerts a force proportional to its displacement from rest position. In our case, we attach one end of a zero-rest-length spring to the particle and fix the other end to the origin in space, resulting in the following term in our differential equation:

$$m\frac{d^2\vec{x}}{dt^2}+\gamma\frac{d\vec{x}}{dt}+k\vec{x}=\vec{f}(t) \qquad (3.6)$$

where k is the spring constant which controls the stiffness of the spring. Since this is a purely position-dependent force, it is conservative and therefore doesn't dissipate energy. If the γ term and the right-hand-side are set to zero, this will result in a purely sinusoidal motion. Adding damping will result in a decaying sinusoidal motion. This can be proved because the differential equation can in fact be solved analytically, resulting in the following equation:

$$x=e^{-\gamma t/2m}(Ae^{i\omega t}+A^{*}e^{-i\omega t}) \qquad (3.8)$$

where

$$\omega = \sqrt{\frac{k}{m} - \frac{\gamma^2}{4m^2}} \tag{3.9}$$

so that k controls the rate of oscillation while γ controls the rate of decay. When the value under the square root is greater than one, the system is said to be underdamped. When it is less than one, it is said to be overdamped, and when it is zero, it is said to be critically damped.

This equation describes a canonical linear system, and is used to model many other linear physical phenomena such as electrical circuits. An analytical solution to this equation only exists, however, if the elastic force is a linear function of x. Since we are solving the differential equation numerically, however, there is no reason that it must be linear, and we can in general make the elastic force any function of the particle's position.

$$m\frac{d^2\vec{x}}{dt^2} + \gamma\frac{d\vec{x}}{dt} + F(\vec{x}) = \vec{f}(t,\vec{x}) \tag{3.10}$$

We can therefore model certain complicated types of elastic behavior with very little additional computational cost.

3.4.2. Modeling External Forces

In contrast to the internal forces, which can be thought of as being tied to the individual particles, external forces are considered to be either global environmental forces or the results of arbitrary constraints or user input. External forces can be functions of time, such wind, or of position, such as a fluid flow or a planetary gravitational field, or even of other model parameters such as temperature. They can represent pseudo-forces (e.g. centrifugal and coriolis forces) and can be controlled interactively by the user or even by another physically-based model.

3.4.3. Constraint Techniques

In addition to environmental external forces, we can add constraint forces to the right-hand side of the differential equation. Constraint forces are non-physical external forces which force the model to meet some kind of geometrical or other type of constraint. They are used by the animator or model builder to force the physical system to behave in a specific way. Constraints are the means by which an animator maintains active control over a physically-based model and makes it do what he wants. Constraint methods were developed originally for engineering and physical simulation of systems which by their structure are forced to behave in a constrained way. Variations on these techniques have been proposed specifically for use in physically based modeling.

3.4.4. Penalty Constraints

Penalty or energy constraints were proposed by Witken et al for imposing certain types of geometric constraints on rigid models [Witkin 87]. They are implemented by creating a potential energy function for each constraint such that the energy level is lowest when the constraint is met. These can be used to build self-assembling models which stay attached even when moved by other forces. The chief advantage of this type of constraint is that the scalar potential energy fields can be added together to form multiple constraints. Not all constraints are guaranteed to me met, however, so penalty constraints are not necessarily exact.

3.4.5. Dynamic Constraints

Dynamic constraints were proposed by Barzel et al [Barzel 88]. In this type of constraint, forces are calculated in order to drive the model towards the constraint path with critically-damped motion. For example, a particle can be constrained so that, eventually, it will follow a curve in space exactly. How rapidly it will conform to this constraint curve depends on the time constants of the constraint mechanism.

3.4.6. Reaction Constraints

Reaction constraints (RCs) are an extension of dynamic constraints and can be useful for constraining objects to surfaces [Plat 88]. RCs project out the component of the existing force on an object which violates the constraint and replaces it with a dynamic constraint. This allows objects to be constrained to a surface, for example, but still be allowed to slide along the surface in a physically natural way. RCs will fulfill their constraints exactly, but only one RC can be applied to a single particle.

3.4.7. Augmented Lagrangian Constraints

Platt pointed out that a physical system can be thought of as an optimization procedure and that a constrained physical system can be treated as a constrained optimization problem. Specifically, physical systems tend to minimize their total energy over time. A constrained minimization problem can be expressed as:

$$\text{minimize } f(\mathbf{x}) \text{ such that } g(\mathbf{x}) = 0 \tag{3.11}$$

were $g(\mathbf{x})$ is a scalar function of $\mathbf{x}$ which is zero when the constraint is met. There is a standard mathematical technique for solving these kinds of problems called *Lagrange multipliers* in which an extra variable, λ, is added to the minimization function:

$$L(\mathbf{x}) = f(\mathbf{x}) + \lambda g(\mathbf{x}) \tag{3.12}$$

A solution to the constrained minimization problem can then be found when the gradient of L vanishes, at which point:

$$\nabla f(\mathbf{x}) = -\lambda \nabla g(\mathbf{x}) \tag{3.13}$$

Platt added penalty constraints to Lagrange multipliers to create augmented Lagrangian constraints, or ALCs, which will evolve to fulfill their constraints exactly, even, in the presence of multiple constraints.

3.4.8. Space-Time Constraints

All the constraint techniques mentioned so far cause physically-based models to evolve gradually towards their constraint surfaces as time progresses. Witken and Kass developed a method in which the entire path of an object, in time as well as in space, is optimized to fit a set of constraints [Witkin 88]. This requires discretizing time over the entire motion sequence, placing the object in key positions and then performing a relaxation algorithm to minimize the power exerted on the object to force it to meet the constraints.

3.5. Numerical Solution

One of the first applications of computer technology, when machines were very expensive, was in the field of numerical solution of differential equations. This illustrates the power and usefulness of numerical techniques and underscores the essential fact that the overwhelming majority of physical problems have no other solution. To quote Feynman [Feynman 63]:

> *"Mathematical analysis is not the grand thing it is said to be; it solves only the simplest possible equations. As soon as the equations get a little more complicated, just a shade--they cannot be solved analytically. But the numerical method...can take care of any equation of physical interest."*

Solving those equations accurately in real-time, however, is not always possible. Nonetheless, the computing power of medium-priced workstations is now such that a number of problems of significant complexity can be solved interactively.

Numerical solution of differential equations tends to break down into two categories: initial value or time-evolution problems, in which the state variables are evolved forward in time from initial conditions, and boundary value or relaxation problems, in which a discrete set of values is iteratively refined towards a solution. This classification can be applied to both ordinary and partial differential equations, and has a strong association with the type of underlying physical problem.

3.5.1. Convert to First Order Equations

An ordinary differential equation of any order can always be written as a set of related first order equations, by expressing the derivatives as separate variables. For example, the mass-spring system described in equation (3.6) can be rewritten as two related equations:

$$\begin{aligned} \frac{d\vec{x}}{dt} &= \vec{v} \\ \frac{d\vec{v}}{dt} &= \frac{1}{m}\left(f - \gamma\vec{v} - k\vec{x}\right) \end{aligned} \tag{3.14}$$

where the velocity vector $\vec{v}$ is the new intermediate variable. This representation of the problem gives us a formula for the first derivatives of each variable, and these two equations can then be solved simultaneously using first-order solution techniques

3.5.2. Initial value problems

Initial value problems are generally associated with time evolution. We take a system in its initial state, and then step it through time in increments of Δt, where Δt is sufficiently small. The solution therefore mirrors the actual evolution of the physical system in time. The simplest and most intuitive method for solving initial value problems is the Euler method, which consists of linearly extrapolating the value at the next time step using the current derivative:

$$x_{t+\Delta t} = x_t + x'(t, x_t)\Delta t \tag{3.15}$$

where

$$x'(t, x_t) = \frac{dx}{dt} \tag{3.16}$$

The value of dx/dt is determined from the derivative formula for the problem, and can be a function of other intermediate variables such as velocity.

The Euler method is neither very accurate nor very stable unless very small time steps are used, and therefore several different methods have been developed which give better accuracy and stability for the same time step. Runge-Kutta methods, for example, use the Euler method to take several trial steps and then use this information to extrapolate a higher order Taylor expansion to the solution. Second order Runge-Kutta is also known as the midpoint method and takes the following form:

$$\begin{aligned} k_1 &= x'(t, x_t)\Delta t \\ k_2 &= x'\left(t + \tfrac{1}{2}\Delta t, x_t + \tfrac{1}{2}k_1\right)\Delta t \\ x_{t+\Delta t} &= x_t + k_2 + O\left(\Delta t^2\right) \end{aligned} \tag{3.17}$$

This same method can be extended to fourth-order Runge-Kutta, which is very popular and is considered by many to be the best general method for solving initial value problems. These kinds of methods can be enhanced by using adaptive stepsize techniques in which the value of Δt is varied according to some error criterion. Other more sophisticated methods can be applied, such as Richardson extrapolation or Bulirsch-Stoer methods, and predictor-corrector methods and more details can be found about them in [Press 92].

The problem of stiffness arises when solving sets of differential equations in which the time scales of the different equations are very different. The time step in this case must be limited to the requirements of the most rapidly varying equation to prevent instability. To restore stability, although not necessarily accuracy, a standard technique is to use implicit methods. The Euler method described in equation (3.15) is sometimes called the explicit or forward Euler method because it extrapolates based on the current value of x, x_t. The implicit or backward Euler method uses the value of the derivative at the next time step, $x_{t+\Delta t}$:

$$x_{t+\Delta t} = x_t + x'(t+\Delta t)\Delta t \tag{3.18}$$

Another method, the Euler-Cromer method [Tonnesen 91], averages the two derivative values:

$$x_{t+\Delta t} = x_t + \tfrac{1}{2}\left(x'_t + x'_{t+\Delta t}\right)\Delta t \tag{3.19}$$

For example, the system of equations (3.14) could be solved using the Euler-Cromer method as follows:

$$\begin{aligned} \vec{v}_{t+\Delta t} &= \vec{v}_t + \frac{1}{m}\left(\vec{f}_t - \gamma\vec{v}_t - k\vec{x}_t\right)\Delta t \\ \vec{x}_{t+\Delta t} &= \vec{x}_t + \frac{\vec{v}_{t+\Delta t} + \vec{v}_t}{2}\Delta t \end{aligned} \tag{3.20}$$

3.5.3. Boundary value problems

Boundary value problems, on the other hand, are associated with functions of space. The spatial domain is discretized in one or more dimensions and the values of all points are determined simultaneously, usually using an iterative relaxation method. The canonical example of a boundary value problem is the Poisson equation (here shown in two dimensions):

$$\nabla^2\psi(x,y) = \frac{\partial^2\psi}{\partial x^2} + \frac{\partial^2\psi}{\partial y^2} = \rho(x,y) \tag{3.21}$$

which basically says that the second spatial derivative, or Laplacian, of some scalar function of space (for example, electrical potential) should be locally equal to $\rho(x,y)$ at every point, subject to some boundary conditions. This can be solved numerically by breaking space up into a rectangular grid indexed by *i* and *j*, and with a grid spacing *h* and using the substitution:

$$\begin{aligned} \left.\frac{\partial\psi}{\partial x}\right|_{i,j} &= \frac{\psi_{i+1,j} - \psi_{i,j}}{h} \\ \left.\frac{\partial\psi}{\partial y}\right|_{i,j} &= \frac{\psi_{i,j+1} - \psi_{i,j}}{h} \end{aligned} \tag{3.22}$$

which converts the differential equation into the following difference equation:

$$\frac{\psi_{i+1,j} + \psi_{i-1,j} + \psi_{i,j+1} + \psi_{i,j-1} - 4\psi_{i,j}}{h^2} = \rho_{i,j} \tag{3.23}$$

The expression on the left can be thought of as a difference operator expressing the difference relationships between adjacent points on the grid, and can be represented as a difference operator stencil:

$$\frac{1}{h^2}\begin{Bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{Bmatrix}\psi = \rho \qquad (3.24)$$

When the value of ρ is everywhere zero, the Poisson equation reduces to the Laplace equation and we can show that this is equivalent to saying that the value of ψ at every point should be equal to the average of its neighboring points. The finite difference equation therefore specifies a certain relationship between local points in space. Given a set of boundary conditions, this local condition can be used to find a global solution to the equation.

The standard approach to solving these problems is the relaxation method. We start out by setting the value of ψ to some initial value, such as zero. Then we pass over each point in the grid and adjust it to fulfill the difference operator according to its neighbors, making sure to maintain the boundary conditions. This process is repeated continually until a solution within the desired accuracy is reached. Another way to describe this process is in matrix terms. If we consider the difference operator applied successively to each point on the MxN mesh, we obtain a system of MN linear equations, which can be expressed as an MNxMN matrix. Although this is normally a very large matrix, it is sparse and banded which make it straightforward to solve using iterative methods. The simplest iterative method is called Gauss-Siedel, which is identical to the relaxation method. More sophisticated methods include successive overrelaxation (SOR) and multi-grid methods.

Because the differential equation is only a description of local relationships, the solution is very much affected by the boundary conditions. Two types of boundary condition are classified as Dirichlet conditions, which are specified as fixed values at the boundary, and Neumann conditions, which are specified as fixed gradient values at the boundary. Free boundary conditions allow the boundary to float freely.

3.6. Building a Physically Based Model

3.6.1. Modeling Methodology

There is no generally accepted methodology for implementing a particular type of physically based modeling. Ronen Barzel proposes a Conceptual-Mathematical-Numeric (CMN) approach for breaking up the problem [Barzel 92]. In this breakdown, the C layer represents a higher-level data structure which encodes the conceptual model. In practical terms, this is usually an object oriented data model in which instances correspond to the physical entities and their physical properties, while references between them represent their physical interrelationships. The M layer embodies the purely mathematical representation of the problem, which is usually programmed in a declarative or functional style. It rests above the N layer, which solves the posed problems through traditional types of numerical programming routines.

This type of breakdown has several advantages. It .allows different programming styles (e.g. object-oriented, declarative, imperative) to be applied where they are most appropriate, and encourages modularity. One should, for example, be able to plug in different types of numerical solution routine without affecting the higher-level design of the program. However, practical difficulties often get in the way of such a clean conceptual division. In particular, rendering and efficiency considerations may demand that each layer make assumptions about how the others are implemented.

Witken [Witken 91] describes a less formal sort of division into an object-oriented layer and a numerical layer. Like Barzel, he discusses the interface between these layers as the gathering and dispersal of values between the objects and the arrays for numerical processing.

3.6.2. Create a Higher-Level Model

The first step in implementing a physically based modeling is to create the higher-level model. This can often be done quite well using an object-oriented design in which instances represent individual physical objects and their various attributes. This can be a straightforward extension to the traditional kinematic modeling hierarchy, although care must be taken in dealing with the different coordinate systems.

3.6.3. Representation as a Force Differential Equation

Once the higher-level model has been constructed, the behavior of the model must be defined in the form of a differential equation, using our intuition about the laws of physics, or following some sort of methodology. This differential equation can basically be thought of as a mathematical encoding of the behavior of the model.

Given a differential equation, and the appropriate initial conditions, simulated physical behavior can be generated by solving the differential equation numerically over time, subject to whatever constraints may be in place. If the model is rendered on the computer screen at the same time it is being solved, the user can view the animation as it is calculated and possibly interact with it by altering parameters of the differential equation or of the constraints. He also may be able to control the time step size in order to maintain stability or increase speed of solution.

3.6.4. Different Types of Physically Based Models

There are a large variety of physically based modeling techniques that have been proposed for computer animation purposes. Hégron and Arnaldi have surveyed these comprehensively in [Hégron 92]. These techniques can be hard to classify because they derive from a very large body of physics, engineering and numerical programming literature. However, it is possible to identify four broad classes of technique within the computer graphics community: free-particle systems, mass-spring systems, continuous systems, and rigid body systems.

Free particle systems are the simplest extension of single particle mechanics. The particles may be completely independent of each other, responding only to global environmental forces, or have some forces of interaction between themselves. However, there is no fixed topology and any particle can interact potentially with any other. Free particle systems are particularly good for modeling fluids such as wind or water, fire, powders such as sand, and malleable materials such as clay. However, the techniques have been extended to model such materials as cloth.

Mass-spring systems limit the connectivity of the mass particles to nearby particles in a one, two or three-dimensional lattice making them much more efficient to solve. They can be thought of as collections of mass points connected to their neighbors by massless springs. They are conceptually simple, easy to implement, and good for modeling springy solid materials or surfaces where the deformation is not too large, or for modeling some kinds of mechanical system made up of flexible parts.

Continuous systems are not based on finite particles at all, but rather on the physical theory of continuous materials known as continuum mechanics. To solve these equations numerically, we are nonetheless obliged to discretize our continuous materials so that the models can resemble mass-springs systems. The mathematical theory underlying the models, however, is more sophisticated. This makes them more difficult to program, but also generates more realistic results, especially for large deformations. Continuous system models are good for modeling elastic solids or surfaces where large deformations are present such as flags, clothing or skin.

Rigid body systems model objects which have appreciable size, but do not deform. They can be represented as mass points with the additional properties of orientation and moment of inertia. They are good for modeling a variety of everyday objects that are rigid or almost rigid such as cars, spaceships or flying logos, and can be combined to model mechanical assemblies, robots or rigid articulated figures.

3.7. Particle Systems

Particle systems are the most straightforward extension of single particle dynamics. they were first introduced into the computer graphics literature by Reeves, who used them to model fire in the Genesis Sequence in the film Star Trek II [Reeves 83]. By animating large numbers of dimensionless points of light, he was able to create a new type of "fuzzy" computer graphics primitive. These particles were modeled dynamically and responded to environmental forces but did not interact with each other. This type of model was extended by Sims [Sims 90] and implemented on a massively parallel computer. Particle systems were later enhanced to allow molecular-like interaction between particles by Miller, who used globular dynamics to model viscous fluids [Miller 89], Terzopoulos, who used models of visco-elastic behavior to model plastic materials

[Terzopoulos 87] and Tonnesen, who modeled particle systems with thermal properties [Tonnesen 91].

Particle systems have recently been used to model more structured types of materials such as cloth [Breen 91]. House pointed out [House 92] that particle systems are most appropriate to model materials in which the macroscopic behavior is determined mainly by highly non-linear interactions within the material's microstructure, because force laws can be made non-linear without increasing the complexity of the solution.

3.7.1. Free Particle Systems

Particle systems can be implemented easily by extending the single mass particle solution to solve for many particles simultaneously. This can be represented mathematically simply by converting the single particle equation to a vector equation, where the vector dimension is the number of particles. These are "vectors" of three-dimensional vectors, such that the arrow represents the Gibbs notation for a three-dimensional space vector, and the boldface represents an *n*-dimensional vector where *n* is the number of particles.

$$\mathbf{M}\frac{d^2\vec{\mathbf{x}}}{dt^2} + \mathbf{C}\frac{d\vec{\mathbf{x}}}{dt} + \vec{\mathbf{f}}_{\text{int}}(\vec{\mathbf{x}}) = \vec{\mathbf{f}}_{\text{ext}}(t) \tag{3.25}$$

Normally, these equations can be solved as initial values problems, as there are no spatial derivatives present. However, the time step must be sufficiently small that the particles do not pass through each other in a single iteration.

As before, there is an internal force term on the left and an external force term on the right. For systems of non-interacting particles, the internal force term does not exist or is a simple function of each particle, resulting in an O(n) complexity. For systems with interacting particles, the internal force term is a function of the entire vector **x**, which means that in the worst case every particle can affect every other particle, resulting in O(n^2) complexity. By contrast, the external force term is normally a function of space and time only and is therefore O(n) complexity.

The macroscopic behavior as well as the speed of solution of the particle system are therefore mainly dependent on the nature of the interparticle forces. If there is no fixed topology, then every particle can potentially interact with every other particle, resulting in the behavior of fluids or powders. This also represents a potential combinatorial explosion in complexity unless some sort of spatial division techniques are used to exclude distant particles from interacting with each other. Such techniques have been proposed, for example by House in [House 92].

3.7.2. Free Particle Force Laws

As long as the interparticle forces fall off to zero at reasonable distances, therefore, the exact force laws themselves can be arbitrarily complex without affecting the speed of solution unreasonably, and this kind of flexibility in allowing microscopic non-linearities is a powerful feature of particle systems. Varying these force laws can have a considerable effect on the overall particle system behavior. Inverse square laws such as gravity will result in celestial mechanical types of behavior such as in star clusters, for example. Normally, the goal in free particle systems models is to model fluids or plastic materials, so the interparticle force laws tend to be modeled after those of molecules and atoms, which are repulsive at short distances, attractive at medium distances, and fall off to zero at large distances.

An example of this kind of force law, expressed as a potential energy function, is the Lennard-Jones potential, which represents the binding energy of two molecules [Tonnesen 91]. It is expressed as a function of interparticle distance, *r*:

$$\phi(r) = \frac{-\varepsilon}{n-m}\left(m\left(\frac{r_0}{r}\right)^n - n\left(\frac{r_0}{r}\right)^m \right), \quad 3 < m < n \tag{3.26}$$

where r_0 is the equilibrium separation between particles, ε is the dissociation energy, which is the energy required to separate the two particles, and m and n are integers which can be chosen to control the falloff rate of the potential. This potential function has an energy well at the fixed distance r_0 between the two particles around which they will tend to oscillate, eventually coming to rest if dissipation is present. The particles will therefore naturally tend to form solid materials of hexagonal lattices at this spacing. The concept of temperature can be added to the particles in such a way that the ε or r_0 parameters increase with it, and that heat energy can flow from particle to particle. Heating up the particles sufficiently can cause them to "melt" and start to behave like a fluid.

3.7.3. Mass-Spring Systems

Mass-spring systems are essentially particle systems with a fixed topology so that each particle is attached to a finite number of neighboring particles. Normally, the interparticle force laws are modeled as massless, ideal Hookian springs. This restriction has two consequences: first it reduces the complexity of problem to *O(n)* since only the neighboring particles must be checked for force contributions, and second it limits the behavior of the resulting models to that of solids or surfaces. Mass-spring systems can, in fact be used to model continuous materials and therefore have many things in common with continuous physically-based models.

Mass-spring systems are useful for modeling reasonably flexible types of material such as jello and elastic surfaces. As the springs become stiffer, however, so do the equations, and a mass-spring model will become unstable as it approaches a rigid body. Furthermore, for large deformations, the separation between mass points can become unreasonably large and the linear Hookian spring force law no longer valid, resulting in unrealistic looking behavior.

3.8. Continuous Systems

Physically-Based models of continuous systems were developed for computer graphics by Terzopoulos et al, who used them to model deformable surfaces and solids [Terzopoulos 87, 88a]. Terzopoulos and Witken later developed a hybrid continuous model which contained both rigid-body and elastic components [Terzopoulos 88b]. Pentland et al have developed continuous deformable models based on modal dynamics [Pentland 89]. Continuous models have also been used by Carignan [Carignan 92] and Aono [Aono 90] to model cloth.

Continuous system models are based on continuum mechanics, which studies elastic, deformable and fluid materials. This branch of physics takes the basic laws of mechanics and applies them to objects whose state varies continuously in space. This means that quantities which were once represented as single values (for a particle) or as an array of values (for a particle system) now become scalar, vector or tensor fields. Mass, m, for example, becomes mass density, $\rho(x,y,z)$ Nonetheless, to solve these equations on a computer, it is necessary to sample these field values discretely, so that if we use a finite difference discretization our mass density field becomes again a collection of mass points. The question naturally arises if continuous physically based models are any different from mass-spring systems. In fact, by choosing certain types of non-linear springs, a mass-spring system can be made equivalent to a continuous model.

However, there are several advantages to using a continuous system model, as Barzel points out in [Barzel 92]. Continuous systems model the true non-linear behavior of elastic materials inherently, while mass-spring systems need to be specifically tailored to do this. Rendering can be done independently of the numerical discretization and penetration tests with other objects can be made on the smooth surface instead of with mass points. Also, the solution technique can be decoupled from the model so that, for example, the grid density of the discretization can be chosen automatically by the numerical solver or even varied adaptively. Alternatively, the equations can be discretized using a finite element method, which breaks the material into triangular or rectangular regions across which the solution is modeled using basis functions. The main disadvantage of continuous system models is the increased complexity of designing and debugging the models and numerical solutions. Continuous surface models, in particular a continuous elastic surface model due to Terzopoulos, will be discussed in greater detail in Chapter 5.

3.9. Rigid Body Mechanics

As continuous systems or particle systems become less and less flexible, they approach a mathematical abstraction called a rigid body. The study of rigid body mechanics has been highly developed in the fields of physics, engineering, robotics, and in automotive and aerospace simulation. Adding rigid body physics is a natural extension of standard kinematic motion control in traditional computer animation systems, and is in its simplest form quite easy to do mathematically. To animate rigid bodies in an interesting way, however, requires them to be constrained and to interact with each other, which can rapidly become a complex problem.

The basic theory of rigid body motion can be found in standard physics texts, for example [Feynman 63] and has been presented for use in computer animation by, for example, Hahn [Hahn 88]. Dynamic formulations of articulated rigid bodies were developed for computer graphics by Armstrong and Green [Armstrong 85], and Wilhelms and Barsky [Wilhelms 85]. General constraint mechanisms for rigid bodies were proposed by Barzel et al [Barzel 88], Witken et al [Witken 87, 88] and Schröder et al [Schröder 90]. Rigid body collision response techniques have been developed by Moore et al [Moore 88] and Baraff [Baraff 89]. Unified models, which can combine multiple techniques, have been proposed by Hégron [Hégron 89], Arnaldi et al [Arnaldi 89] and Luciani [Luciani 91].

3.9.1. Generalization of Multi-Particle Systems

A rigid body is defined as a finite or infinite collection of mass points which are constrained to maintain a fixed distance from each other. In true life, all objects are somewhat flexible, but the rigid body approximation is a very good one for modeling a variety of everyday objects. Since the rigid body cannot bend, it cannot store potential energy. However, it can rotate, and this requires the introduction of new derived physical quantities, which can be thought of as rotational extensions to the translational quantities of a mass point. These rotational quantities are: orientation, angular velocity, angular acceleration, moment of inertia, torque and angular momentum. These correspond one-to-one with the translational quantities as follows:

$$\begin{array}{ll}
m = \text{mass} & \mathbf{I} = \text{moment of inertia} \\
\vec{x} = \text{position} & q = \text{orientation} \\
\vec{v} = \dfrac{d\vec{x}}{dt} = \text{velocity} & \vec{\omega} = \dfrac{d\vec{\theta}}{dt} = \text{angular velocity} \\
\vec{a} = \dfrac{d\vec{v}}{dt} = \text{acceleration} & \vec{\alpha} = \dfrac{d\vec{\omega}}{dt} = \text{angular acceleration} \\
\vec{f} = \text{force} & \vec{\tau} = \text{torque} \\
\vec{f} = m\vec{a} & \vec{\tau} = \mathbf{I}\vec{\alpha}
\end{array} \tag{3.27}$$

Since we are dealing with orientation space, orientation quantities are represented by quaternions. Fortunately, all of the other quantities except for the moment of inertia involve derivatives of rotations and therefore can be represented as three-dimensional vectors, although they should not be regarded as ordinary Cartesian vectors and are sometimes called pseudovectors.

3.9.2. Moment of Inertia Matrix

The major point where the analogy to translational dynamics breaks down is with the inertia matrix. Unlike mass, which is a simple scalar quantity, the inertia matrix has a preferred direction in space. Since it is a symmetric matrix, it can be diagonalized so that in its preferred coordinate system there are only three components, one in each of the x, y, and z directions. For the special case of a sphere, these are all equal so the moment of inertia reverts in this case to a scalar. In general this is not the case, and since the preferred coordinate system of the rigid body rotates with it, the simple laws of motion stated above apply only to the local rotating system, not to the global inertial coordinate system. Since it is much more convenient to work in the local coordinate system, thereby avoiding having to recalculate the inertia matrix at every instant, it is necessary to add in a component to the torque law to account for the "fictitious" centrifugal and coriolis forces generated by the rotating coordinate system.

$$\check{\tau} = \mathbf{I}\check{\alpha} + \check{\omega} \times (\mathbf{I}\check{\omega}) \tag{3.28}$$

The rigid body motion can then be simulated in a manner analogous to that for a single particle, with the additional orientation and angular velocity parameters in the state vector, using the rotational equations of motion.

3.9.3. Multiple Rigid Body Systems

Single rigid body mechanics can be extended in several ways. One way this can be done is to have multiple free rigid bodies colliding with each other and obstacles in the environment. Since rigid bodies can collide in a variety of ways, involving friction and rotation, the mathematics can become very complicated. A second type of multiple rigid body system is a mechanical assembly if rigid bodies fastened together by holonomic constraints. Since the holonomic constraints do not add or remove energy from the system, they have the effect of reducing the number of degrees of freedom of the system. These rigid body systems are the basis for mechanical simulation and dynamic models of articulated figures.

3.10. Layered Elastic Models

In Chapter 3 we reviewed the technique of layered character models containing a skeleton and outer skin layer. This division into two layers is conceptually simple and is also practical, since the skin envelope can be sculpted using standard surface modeling techniques, or scanned directly from a sculpture or a living creature. Simple observation of human or animal skin in motion, however, reveals that the deformation of the outer skin envelope results from many other factors besides the position of the skeleton. In fact, the skin reveals as much as it conceals details of the underlying anatomy in motion. More advanced layered models take into account some aspects of these intermediate anatomical layers between the skeleton and skin.

Some 3D character models, such as the baby in *Tin Toy* and more recently the dinosaurs in *Jurassic Park*, have used geometric techniques to simulate underlying muscle and bone layers. One basic limitation of these layered models, as well as non-layered character models, is the absence of any physical basis. Both the skeleton and the surface envelope, as well as any intermediate layers, are purely geometric. To improve realism, and to create more interesting deformations, layered elastic models are used which add physically based elastic components to some or all of the layers.

3.10.1. Goop

A simple example of this type of approach is Pacific Data Images' *Goop* system, in which a mass and spring with damping are attached to each vertex of a polygonal model [Walters 89]. The spring is free to move in three dimensions and has a zero initial length. The mass, spring constant and damping factor of each vertex can be varied locally. Moving the model is accomplished by moving the anchor points of the springs, causing the vertex points to oscillate about the new position until the motion is damped. This causes a jello-like effect which can be controlled by the animator. The surface points are not attached to each other, so the skin is in fact a free particle system. It therefore has no surface-like physical behavior. However, within a range of small deformations this can be acceptable. This technique was used to animate *Waldo*, Jim Henson's computer generated Muppet character. The character motion was input in real-time by a professional puppeteer using a Waldo motion capture device. The puppeteer was able to monitor a rigid body display of the character interactively, while the Goop technique was applied as a step in the rendering of the final animation.

Figure 3.1: Waldo C. Graphic, the Computer Muppet [Graves 89]

3.10.2. Snakes and Worms

Gavin Miller took this idea a step further by attaching the vertex points together to form a mass-spring lattice. This was arranged in the form of a tube to model dynamic snakes and worms [Miller 88]. By actively varying the rest-lengths of the springs, he was able to simulate muscles, resulting in worm-like motion. This model did not have any internal skeleton, however, which limits its application to certain types of 3D characters for which skeleton control techniques are not applicable.

3.10.3. The Critter System

A more sophisticated examples of layered elastic construction for animated characters is found in the Critter system [Chadwick 89] in which a regular network of connected springs and masses is used to create a control point lattice for free-form deformations of the geometric surface. Some of the control points are bound to links of the underlying skeleton so that, when the skeleton is animated, the unattached mass points are influenced to move through the spring lattice. Since the lattice is controlling the space deformation and not the actual model, it can be relatively coarse with few enough mass points that the physical simulation can be conducted at interactive speeds. This can be quite effective in producing large-scale deformations of the character, but the resolution of the deformation is limited by the size of the mesh, and although the mass-spring lattice allows for shape control over the body deformation, the skin is still fundamentally a geometric surface model, not a model of a physical skin.

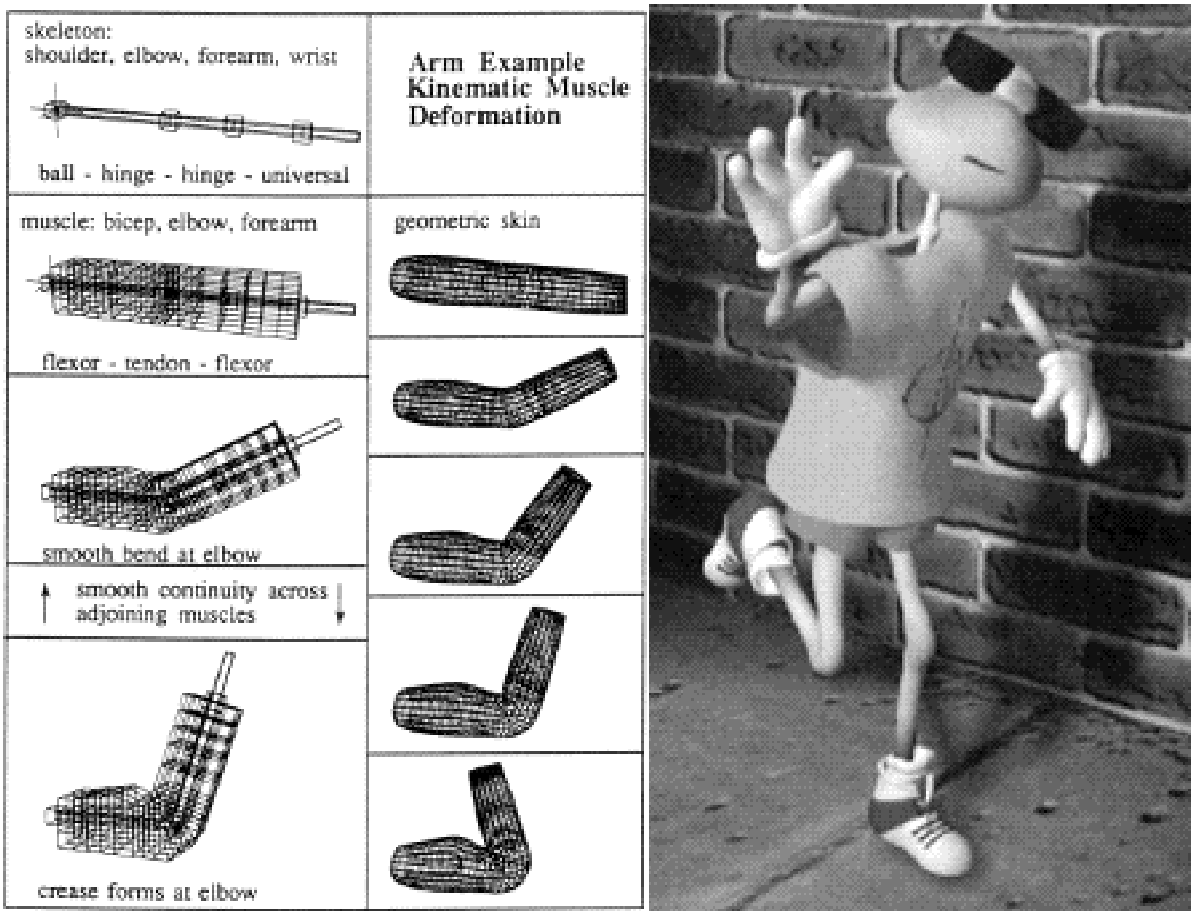

Figure 3.2: The Critter System [Chadwick 89]

3.10.4. Layered Elastic Facial Animation

A sophisticated example of a layered elastic model is used by Terzopoulos to implement facial animation [Terzopoulos 91]. In this model, an elastic solid simulation, consisting of a mass-spring lattice of depth three, is attached to a human skull model and deformed by muscles which take the form of force constraints between points on the skin surface and the underlying bone. The springs are biphasic to emulate the non-linear behavior of real skin, that is, they have a low spring constant for small displacements which shifts to a high spring constant for large displacements. Volume preserving constraints are used to simulate the effects of incompressible fatty tissue. This type of model is highly specialized for the human face, and is derived from real biomechanical information. It is not clear how well the three-layer mass-spring lattice, which is bound to the bone layer, would model other parts of the body such as joints where the skin is looser. Nonetheless, the visual results for facial animation are quite effective, and the model represents a promising approach to layered construction whose more general application needs to be further explored.

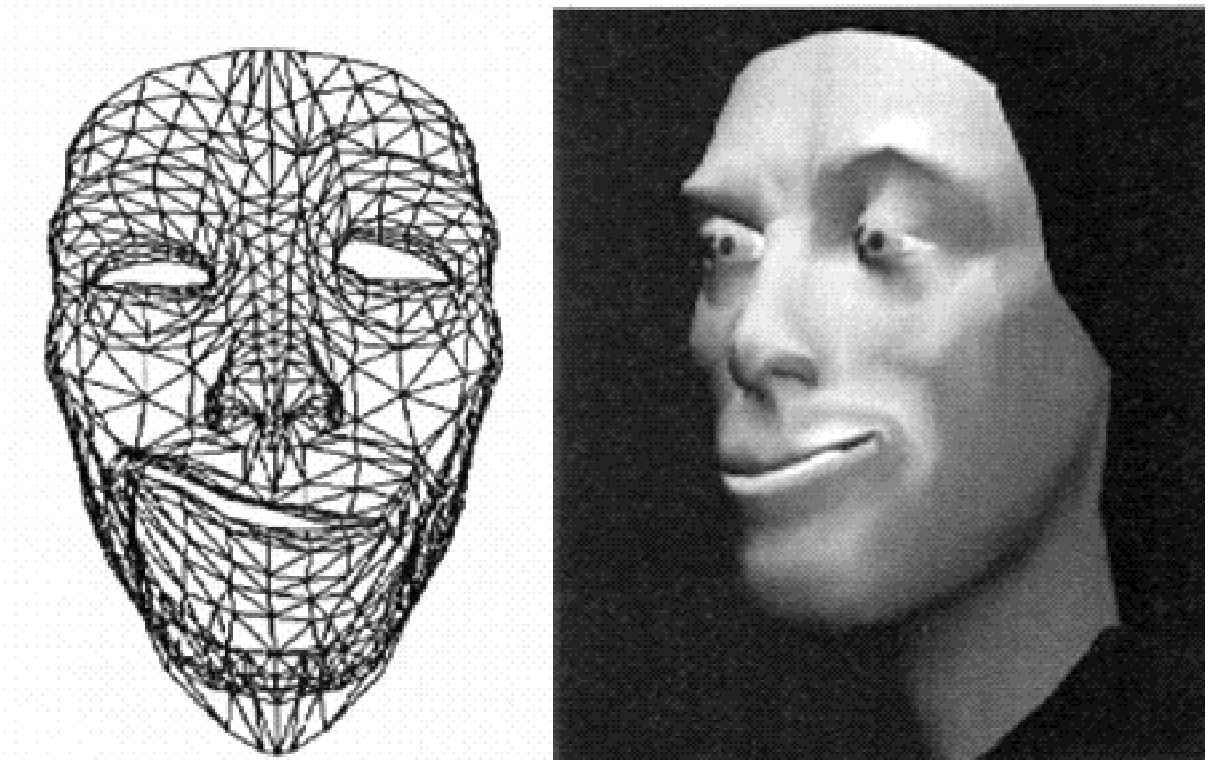

Figure 3.3: Physically-Based Facial Animation [Terzopoulos 91]

3.10.5. Other Physically-Based Models

An elastic layered model for complex deformable bodies was proposed by Gascuel et al [Gascuel 91] in which the control points for an interpolating spline surface are bound to a rigid bone layer by stiff springs. The control points are connected to each other to form a surface using a non-physically-based geometric technique for propagating deformations. This model was used to animate an octopus with free-swinging arms. Another type of elastic layered model in two dimensions was developed by Overveld [Overveld 90]. The finite element method was used by Gourret et al [Gourret 89], who describe a human hand modeled as a volume element mesh surrounding bones. This was used to simulate a hand grasping a rubber ball. Chen et al [Chen 92], also used the finite element method to develop a biomechanically-based model of muscles on bone based on biomechanical data from a frog's leg. No attempt was made to model the skin.

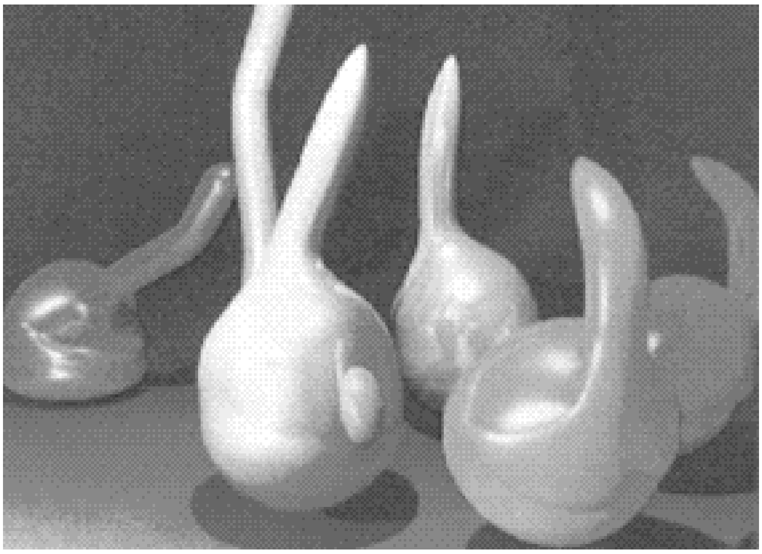

Figure 3.4: Complex Deformable Bodies [Gascuel 91]