Realistic Simulation of Viscoelastic Bodies

Rogério L. W. Liesenfeld and Jorge Stolfi
Institute of Computing, University of Campinas

Relatório Técnico IC–97-12

Setembro de 1997

Realistic Simulation of Viscoelastic Bodies

Rogério L. W. Liesenfeld*and Jorge Stolfi†
Institute of Computing, University of Campinas

September 28, 1997

Abstract

We describe an animation system that simulates the dynamics of viscoelastic bodies subject to equality and inequality constraints. We show how Lagrange's method can be used to derive the equations of motion of such bodies from general formulas for the elastic and kinetic energy, the viscous power loss, and mechanical constraints, in terms of generalized coordinates.

We also describe a convenient two-parameter non-linear model for the elastic forces, that agrees with Hooke's law for small deformations, but does not allow the material to be compressed to zero or negative volume. In particular, we derive the equations of motion for elastic bodies modeled by tetrahedral finite elements with affine deformations. Finally, we show how collisions between such bodies can be efficiently and accurately detected by combining Hermite interpolation of the non-penetration constraints with Lin and Manocha's bounding box tests.

1 Introduction

The kinematic techniques still used in most commercial animation systems leave to the animator the task of estimating the object motions according to the laws of physics. Physically-based simulation offers a promising alternative.

We describe here an animation system that simulates the dynamic behavior of elastic bodies, according to the laws of Newtonian mechanics. Each body is modeled by a collection of tetrahedral elements, glued together by their faces. Each element is allowed to deform only by affine transformations, so that its shape remains tetrahedral along the simulation.

In order to make the paper as self-contained as possible, we show how Lagrange's method can be used to derive the equations of motion of a mechanical system from general formulas for the elastic and kinetic energy, and the viscous power loss. This exposition is quite general, allowing the position and deformation of the bodies to be specified by an arbitrary collection of shape and position coordinates.

The Lagrangian approach allows us to model general mechanical constraints on the positions of the bodies, expressed as algebraic equalities on their coordinates. These constraints

*FAPESP grant 94/4132-6.
†CNPq grant 301016/92-5.

can be used to control the animation: to keep a body fixed, to force it to follow a predefined trajectory, to establish a mechanical linkage between two bodies, and so on.

We also describe a convenient two-parameter mathematical model for the elastic properties of simulated isotropic materials. The model is non-linear in the deformation measures, and was designed to allow large deformations without allowing the bodies to be compressed to zero or negative volume. This model, nevertheless, reduces to Hooke's linear model for small deformations. The two parameters are then identified with the two elastic moduli that define the material's resistance to static compression and shearing.

A similar two-parameter model is given for the viscous forces that resist changes in the material's deformation. This model too reduces to the standard Newton model for slow deformations, its two parameters then defining the material's resistance to dynamic compression and shearing.

The independent control of these four physical parameters allows for the realistic simulation of a wide range of materials, such as solid rubber, plastic foam, putty, protoplasm — and, of course, Jello$^{\mathrm{TM}}$.

An essential feature of a practical dynamic animation system is the ability to automatically detect and handle collisions between the simulated bodies (or different parts of the same body). We show how the timing of such events can be accurately computed, at relatively low cost, by using Hermite interpolation on the formulas that define the non-penetration constraints. Finally, we show how to drastically reduce the number of such tests by a bounding-box technique due to Lin and Manocha [1].

1.1 Related work

The earliest dynamic animation systems for elastic bodies, such as the one described by Terzopoulos and others [2, 3], used a model consisting of point masses laid down in a rectangular grid. The local deformation was computed by finite differences. Platt and Barr [4] extended that model to include general algebraic constraints (such as incompressibility) and plastic deformations. They also used cubical finite elements instead of point masses. Witkin, Gleicher and Welch [5] further developed this model for the specific case of affine deformations.

In all these works, however, the elastic deformation energy was modeled as a quadratic function of the amount of deformation, thus allowing in principle a finite force to compress the material down to zero volume. Moreover, the stiffness and viscosity of the material were controlled by a single parameter each.

The finite element approach is rather expensive: one needs a large number of elements to obtain acceptably smooth deformations. Alternative approaches, which restrict the deformations in order to reduce the simulation cost, have been proposed by Pentland and Williams [6], and Witkin and Welch [7].

Techniques for fast collision detection in dynamic animation were proposed by Moore and Wilhelms [8], Baraff [9], Ponamgi, Manocha, and Lin [10], and many others. However, these methods are typically optimized for rigid polyhedral bodies: they often start by decomposing each body into convex parts, whose collisions are relatively easy to detect (typically $O(1)$ operations for each pair of bodies in each time step). This approach cannot be used for deformable bodies, which may change from convex to concave, and even collide with themselves. One must then view each surface element as a separate object. The bounding-box methods of Lin and Manocha [1], which we describe in section 6.4, are well-suited for such model.

Once a collision has been detected, it must be handled in some way. A simple approach, described by Moore and Wilhelms [8] and other authors, is to introduce a stiff virtual spring between the colliding bodies, that pushes them apart and gets removed when the bodies begin separate. This approach runs into problems when there are multiple collisions or sliding contacts. The modeling of collisions by impulses (infinitely strong forces of infinitesimal duration) is conceptually more consistent, but efficiently computing the required impulses and contact forces is still an open problem; a detailed analysis and some partial solutions were given by Baraff [11, 12]. A newer approach, still under development, is Mirtich and Canny's microimpulse model [13, 14].

1.2 Notation

We denote a row vector by $[u_1, \ldots, u_m]$, and a column vector by $[u_1, \ldots, u_m]^\top$. Vectors are columns unless said otherwise.

If f is a scalar function of the m-vector $u = [u_1, \ldots, u_m]^\top$, we denote by $\partial f / \partial u$ the vector $[\partial f / \partial u_1, \ldots, \partial f / \partial u_m]^\top$; and by $\partial^2 f / \partial u^2$ the $m \times m$ matrix whose element in row i and column j is $\partial^2 f / \partial u_i \partial u_j$, for $i, j \in \{1, \ldots, m\}$.

If v is any property of the system that changes with time, we denote by $v(t)$ its value at instant t and by v' and v'' its first and second derivatives with respect to time.

2 Equations of motion

For the purposes of this section, a *dynamic system* is a collection of point-like material particles that move in space in response to internal and external forces, each according with Newton's law $F = ma$.

2.1 Generalized coordinates

The *configuration* of a dynamic system at a given instant consists of the positions $\chi_1, \chi_2, \ldots, \chi_m$ of all its material particles in $\mathbf{R}^3$. The *state* of the system consists of the current positions and velocities of those particles.

Material objects contain an astronomical number of particles, and it is obviously impossible to simulate all their individual motions. In dynamic animation we must necessarily work with a highly simplified model of the system, where only the most important degrees of freedom in the motion of the particles are represented. For example, to simulate the motion of a rigid object, we would keep track only of the position of its center of mass, and the orientation of an orthogonal frame fixed on the body; and assume that every particle has a fixed position in this moving coordinate system.

So, suppose we have a simplified model of the system, and $q_1, \ldots, q_n$ are n real-valued parameters whose values at any instant t completely determine the model's configuration at t; i. e., the particle positions $\chi_1, \ldots, \chi_m$ can be written as functions of $q_1, \ldots, q_n$. The particle velocities $\chi'_1, \ldots, \chi'_m$ at t are then completely determined by the quantities $q_1, \ldots, q_n$ and their time derivatives $v_1, \ldots, v_n = q'_1, \ldots, q'_n$. One says that the former are a set of *generalized coordinates* for the system, and the latter are the corresponding *generalized velocities.*

We stretch the language a bit and say that the column n-vector $q = [q_1, \ldots, q_n]^\top$ *is* the configuration of the system; and the pair (q, v), where $v = q' = [v_1, \ldots, v_n]^\top$, *is* its state. Note that both q and v are functions of time.

Given a collection of forces $f_1, \ldots, f_m$ acting on the m particles of the system, one defines the corresponding *generalized force* E_i acting on each generalized coordinate q_i, such that the work done by those forces when the coordinates q change by an infinitesimal vector δ will be $E^\top \delta$. It can be shown that each E_i is a linear function of the particle forces f_j:

$$E_i = \sum_{j=1}^{m} f_j \cdot \frac{\partial \chi_j}{\partial q_i} \tag{1}$$

2.2 Lagrange's equation

It follows from the laws of classical mechanics that the evolution of a dynamic system is completely determined by its initial state and the forces applied by the environment over time. *Lagrange's equation* [15] is a general differential formula that determines the system's evolution, in terms of an arbitrary system of generalized coordinates, from the formulas that express the energy of the system in those coordinates.

In many dynamic systems, there are physical processes that give rise to *conservative forces*—forces that depend only on the position of all particles in the system. We can view those forces as storing a certain amount of *potential energy*, that can be converted into mechanical work or into the *kinetic energy* of the system's particles. The *internal energy* of the system is the sum of these two terms; it may change due to *external forces* applied by the environment on the particles, or by *dissipative* (*friction*) *forces* that resist the motion of the particles, and depend only on their position and velocity.

The potential energy of the system depends on the particle's positions alone, so it is a function P of the coordinate vector q. The kinetic energy depends only on the particle velocities, so it can be written as a function K of q and v. The rate of energy loss due to internal friction depends on the positions and velocities of the particles, and therefore it can be expressed as a function W of q and v.

Lagrange's equation, which is ultimately derived from Newton's law, states that the evolution of the state (q, v) satisfies

$$\frac{d}{dt}\left(\frac{\partial K}{\partial v}\right) - \frac{\partial K}{\partial q} + \frac{1}{2}\frac{\partial W}{\partial v} + \frac{\partial P}{\partial q} = E \tag{2}$$

where $E = [E_1, \ldots, E_n]^\top$ is the vector of generalized forces applied on the system by the environment.

Expanding the derivatives of (2) and rearranging the terms, we get the matrix form of Lagrange's equation,

$$\mathrm{M} q'' = F \tag{3}$$

where M is the *generalized mass matrix*, and F is the *generalized total force vector*, defined as

$$\mathrm{M}_{ij} = \frac{\partial^2 K}{\partial v_i \partial v_j} \tag{4}$$

$$F_i = E_i - \sum_{j=1}^{n} \frac{\partial^2 K}{\partial v_i \partial q_j} v_j + \frac{\partial K}{\partial q_i} - \frac{1}{2}\frac{\partial W}{\partial v_i} - \frac{\partial P}{\partial q_i} \tag{5}$$

Equations (3–5) allow us to compute the accelerations q'' for a state (q, q'), given the external force vector E. The system's evolution can be determined by integrating the second-order differential equation $q'' = \mathrm{M}^{-1}F$, where F is computed from q, q', and E.

There are certain non-mechanical internal processes that give rise to forces that are neither conservative nor dissipative. Examples include thermal expansion, chemical reactions, state changes, electromagnetic phenomena, and so on. For the purposes of mechanical simulation, any such forces should be included in the vector E_i. (In fact, all forces could be handled this way. The separate handling of conservative and dissipative forces is justified by modeling convenience: usually, it is much easier to define the scalar functions P and W, than the corresponding generalized forces $F_i - E_i$.)

3 Constraints

Lagrange's formula (2) can be used only when the generalized coordinates q_i are independent and non-redundant; that is, when the set of allowed configurations for the system has dimension exactly n.

In many situations, however, this requirement is too restrictive. In practice, we generally use a *redundant* set of generalized coordinates, together with one or more *constraints* that restrict them to some lower-dimensional manifold of *valid configurations*. For example, in the case of a particle restricted to move on the unit sphere $\mathbf{S}^2$, we could let the q_i be the Cartesian coordinates (x, y, z) of the particle, together with the constraint $x^2+y^2+z^2-1 = 0$.

Constraints typically arise in systems that consist of several solid bodies in contact or connected by mechanical joints, whether among themselves or to the external environment. As a rule, the only practical way to model such systems is to model each part independently, concatenate the coordinate vectors of all parts, and subject the resulting vector to the equations implied by the additional constraints.

In the context of computer animation, constraints can be used also to keep an object fixed in space, or drag it along a prescribed trajectory.

In general, suppose we have a set of constraints on the system's configuration that can be expressed by equations $\Phi_r(q, t) = 0$, for $r = 1, \ldots, k$. In order to keep these equations satisfied, the constraining processes must apply appropriate *constraint forces* on the system. The corresponding generalized forces must be added to the right-hand side of Lagrange's equation (2).

3.1 Constraints as springs

There are two main approaches for computing these forces. The simplest, and perhaps most intuitive, is to model each constraint $\Phi_r(q,t) = 0$ by a spring whose stretching energy is $K\Phi_r(q,t)^2$, for some constant $K > 0$. This approach allows the constraint to be slightly violated, but the spring will automatically provide a force that tends to restore the constraint. By increasing the spring stiffness K, the magnitude of the violations can be made as small as desired. The drawback of this method is that the presence of stiff springs makes the differential equation unstable.

3.2 Exact constraint forces

Another approach consists of directly computing the generalized constraint force vector C that is needed to exactly fulfill the constraints at each instant. Given a constraint equation $\Phi_r(q,t) = 0$, let h_r be a function of time that records the value of the left-hand side throughout the evolution of the system. Satisfying the constraint means ensuring that $h_r = 0$ at all times. If we start from a valid state, we must have $h_r = 0$ and $h'_r = 0$ at the initial moment. To maintain these conditions, we need only to ensure that the acceleration q'', at every instant, is such that h''_r is always zero. This requirement contributes one linear equation relating q'' to known quantities, namely

$$\frac{\partial \Phi_r}{\partial q}(q,t)^\top q'' = \psi_r \tag{6}$$

where

$$\psi_r = -q'^\top \frac{\partial^2 \Phi_r}{\partial q^2}(q,t)\, q' - 2\frac{\partial^2 \Phi_r}{\partial q \partial t}(q,t)^\top q' - \frac{\partial^2 \Phi_r}{\partial t^2}(q,t) \tag{7}$$

The matrix formulation (3) is then replaced by

$$\left\{ \begin{array}{rcl} \mathrm{M}q'' &=& F + C \\ \mathrm{N}q'' &=& \psi \end{array} \right. \tag{8}$$

where C is a column n-vector of unknown constraint forces, N is a $k \times n$ matrix given by

$$\mathrm{N}_{rj}(q,t) = \frac{\partial \Phi_r}{\partial q_j}(q,t) \tag{9}$$

for row $r = 1, \ldots, k$ and column $j = 1, \ldots, n$, and ψ is the k-vector defined by formula (7).

3.3 Lagrange multipliers

In general, equations (7–9) do not determine the constraint forces completely. Fortunately, for most kinds of mechanical constraints (including contacts and mechanical joints), we can easily determine the direction of the associated constraint force; only the magnitude remains to be determined. For example, for a particle sliding on a fixed plane, the constraint force will be some multiple of $u - \mu v$, where u is the plane's unit normal, v is the particle's velocity, and μ is the dynamic friction coefficient.

If we know the directions $d_1, \ldots, d_k$ of the constraint force vectors for all the equations $\Phi_1, \ldots, \Phi_k$, the total force vector C is some linear combination $C = \lambda_1 d_1 + \ldots + \lambda_k d_k$. We can determine the multipliers $\lambda_1, \ldots, \lambda_k$ by solving equation (8). This is the so-called *method of Lagrange multipliers.*

Let λ be a vector of Lagrange multipliers. By writing $C = -\mathrm{G}\lambda$, for the $n \times k$ matrix G whose columns are the directions $d_1, \ldots, d_k$, we can solve (8) for λ, obtaining

$$\mathrm{N}\mathrm{M}^{-1}\mathrm{G}\lambda = \mathrm{N}\mathrm{M}^{-1}F - \psi$$

This equation allows us to compute λ, and hence C, from known quantities.

For constraints that describe frictionless joints, linkages, or sliding parts, the constraint force is usually directed along the gradient $\partial\Phi_r/\partial q$ of the associated equation Φ_r. If all constraints are of this type, then matrix G is just $\mathrm{N}^\top$.

4 Continuous model for elastic bodies

In a solid body, neighboring particles remain close to each other. Therefore, we can model a solid body as a piece of a continuous medium moving through $\mathbf{R}^3$, and subject to continuous deformations.

More precisely, we model the system as a closed and finite region U of $\mathbf{R}^3$, its *reference configuration.* (Note that U does not have to be connected, so the system may consist of two or more separate bodies.) A *configuration* of the system is then a continuous function f that maps each point $u = [u_x, u_y, u_z]^\top \in U$ of the medium to a position $f(u) = [f_x(u), f_y(u), f_z(u)]^\top$ in $\mathbf{R}^3$. Since we don't want the bodies to interpenetrate, we require that f be one-to-one when restricted to the interior U^+ of U.

4.1 Potential energy of deformation

For an elastically deformable body, a significant part of the potential energy is stored in the elastic deformation of the material. At the microscopic level, this energy is due to the displacement of each particle relative to its neighbors.

Let u be a point of U^+. The relative displacement of particles in the neighborhood of u, for a given configuration f, is determined to first order by the Jacobian matrix of f at u,

$$\mathbf{J}f(u) = \begin{bmatrix} \partial f_x/\partial u_x & \partial f_x/\partial u_y & \partial f_x/\partial u_z \\ \partial f_y/\partial u_x & \partial f_y/\partial u_y & \partial f_y/\partial u_z \\ \partial f_z/\partial u_x & \partial f_z/\partial u_y & \partial f_z/\partial u_z \end{bmatrix} \tag{10}$$

Specifically, a particle that is located at $v = u + \varepsilon$ in the reference configuration, for any infinitesimal vector ε, will be located at $f(v) = f(u) + (\mathbf{J}f(u))\varepsilon + O(|\varepsilon|^2)$ in the configuration f. For ordinary materials, it turns out that the second-order terms have negligible effect on the elastic energy. Therefore, the density of elastic energy $\phi_f(u)$ in the neighborhood of point u can be computed from the Jacobian $\mathbf{J}f(u)$ alone.

Let g be a configuration of the body for which the density $\phi_g(u)$ is zero. (We say that g is *locally relaxed* at u.) Then $\phi_f(u)$ depends on the local deformation determined by f, relative to that determined by g. That is, ϕ_f must be expressible as $\Phi(\mathrm{C})$, where Φ is a function that depends on the material, and

$$\mathrm{C} = \mathbf{J}(f \circ g^{-1})(u) = (\mathbf{J}f(u))(\mathbf{J}g(u))^{-1} \tag{11}$$

The matrix C is called the *strain tensor* of the configuration f at u.

Moreover, the elastic energy should not be affected by rotating the body as a rigid whole. In that case, it can be shown that $\Phi(\mathrm{C})$ must depend only on the (symmetric) matrix $\mathrm{D} = \mathrm{C}^\top\mathrm{C}$, the *metric tensor* of f at u.

Also, if the material is isotropic (meaning that its mechanical properties are the same in all directions), the energy should remain unaffected by rotation of the locally relaxed configuration around point $g(u)$. In that case, it can be shown that $\Phi(\mathrm{C})$ must depend only on the coefficients d_0, d_1, d_2 of the characteristic polynomial of D,

$$\chi(\lambda) = \det(\mathrm{D} - \lambda\mathrm{I}) = \lambda^3 + d_2\lambda^2 + d_1\lambda + d_0 \tag{12}$$

Expanding the determinant shows that

$$d_0 = \det \mathrm{D} = (\det \mathrm{C})^2 \tag{13}$$
$$d_1 = \textstyle\sum_i \mathrm{D}^{(2)}_{ii} = \sum_{ij} (\mathrm{C}^{(2)}_{ij})^2 \tag{14}$$
$$d_2 = \textstyle\sum_i \mathrm{D}_{ii} = \sum_{ij} \mathrm{C}^2_{ij} \tag{15}$$

where $\mathrm{M}^{(2)}$ denotes the matrix of 2×2 cofactors of M.

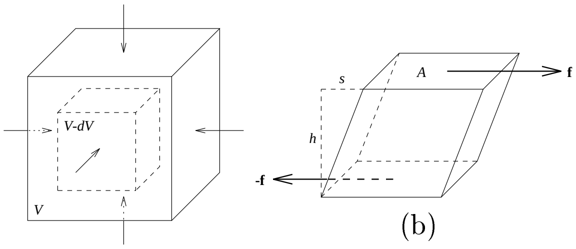
In conclusion, the energy function $\Phi(\mathrm{C})$ is a symmetric function of the coefficients d_0, d_1, d_2 that is minimum (zero) when D is the identity matrix, i.e. when $d_0 = 1$ and $d_1 = d_2 = 3$. These conditions still allow infinitely many functions Φ. Partly for reasons of computational efficiency, we have chosen

$$\phi_f = \frac{\alpha}{32}\left(d_0^2 + \frac{1}{d_0^2} - 2\right) + \frac{\beta}{6}\left(d_2^2 - 3d_1\right) \tag{16}$$

For small deformations, the coefficients α and β that appear in formula (16) are precisely the two *elastic moduli* of the material, that express its resistance to changes in volume and in shape, respectively.

Specifically, suppose that application of a uniform pressure p causes a sample of the material to shrink from volume V to volume $V - \delta V$. See figure 1(a). For small deformations, the relative shrinkage $\delta V/V$ is proportional to p; the ratio $p/(\delta V/V)$ is the *bulk elasticity modulus*, which coincides with α in formula (16).

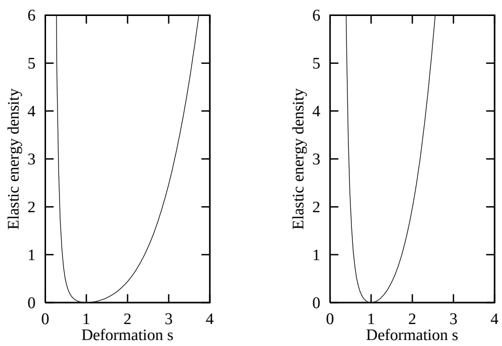
On the other hand, suppose a sample of the material with cross-section of area A is attached between two horizontal plates, constrained to lie a fixed distance h apart; and that a horizontal force f is applied to one of the plates. See figure 1(b). For small deformations, the resulting plate displacement s is found to be proportional to h and $|f|$, and inversely proportional to A. The ratio $|f|/(sA/h)$ is the *rigidity* or *shear modulus*, which is the β of formula (16).



(a) (b)

Figure 1: Elasticity coefficients: (a) bulk, (b) shear.

One advantage of formula (16), compared to the simpler quadratic forms that have been used in other works, is that the first term tends to infinity as the volume approaches zero. See figure 2. Therefore, the elastic forces computed by this formula will automatically prevent the tetrahedra from collapsing and turning “inside out”, even under extreme forces.



(a) (b)

Figure 2: The elastic energy for simple deformations. (a) A uniform scaling that changes the volume by a factor s; (b) A constant-volume deformation that simultaneously scales one axis by s and another by $1/s$. The moduli are $\alpha = \beta = 1$.

In this analysis, we have assumed that f is differentiable to first order almost everywhere in the set U; in other words, the set U^* of points where f is not differentiable has zero volume. For ordinary materials, the energy contents of a zero-volume set is zero, and therefore the total elastic energy can be computed as the integral of $\phi_f(u)$ over the set $U \setminus U^*$.

4.2 Viscosity losses

A real deformable body generally shows viscous friction, a loss of kinetic energy due to forces that tend to oppose the relative motion of neighboring particles inside the body. Experiments show that, for most materials, the power loss $\omega_{f,f'}(u)$ per unit volume around a particle u depends only on the first-order spatial variation of the particle velocities near u.

The function that gives the current velocity of a particle in terms of its current position is $h = f' \circ f^{-1}$. The relative motion of particles in a small neighborhood is therefore summarized, to first order, by the spatial derivatives of this function, namely by the matrix $\mathrm{C}' = \mathbf{J}h = \mathbf{J}(f' \circ f^{-1}) = (\mathbf{J}f')(\mathbf{J}f)^{-1}$.

Rigid rotational and translational motions entail no viscous losses. It follows that the viscous loss density $\omega_{f,f'}$ depends only on the time derivative of the metric tensor $\mathrm{D}' = (\mathrm{C}^\top \mathrm{C})' = \mathrm{C}'^\top \mathrm{C} + \mathrm{C}^\top \mathrm{C}'$. Furthermore, in an isotropic material, a static rotation of the reference configuration does not affect the viscous forces. That is, $\omega_{f,f'}$ should be the same for D' or $\mathrm{S}^{-1}\mathrm{D}'\mathrm{S}$, where S is any static rotation matrix. Therefore, $\omega_{f,f'}$ is some function Ω of the three eigenvalues ξ_1, ξ_2, ξ_3 of D'.

The function Ω is further constrained by noting that the viscous losses are always non-negative, are zero when D' is the null matrix, and must be symmetric on the three eigenvalues. It follows that, for gradual deformations (meaning small $||\mathrm{D}'||$), Ω must be a quadratic, homogeneous, symmetric function of ξ_1, ξ_2, ξ_3. Any such function can be written in the form

$$\Omega(\xi_1, \xi_2, \xi_3) \;=\; \eta_1\, \mathcal{H} + \eta_2\, \mathcal{J} \tag{17}$$

where

$$\begin{aligned} \mathcal{H} &= \frac{1}{2}(\xi_1 + \xi_2 + \xi_3)^2 \\ \mathcal{J} &= \frac{1}{3}(\xi_1^2 + \xi_2^2 + \xi_3^2 - \xi_1\xi_2 - \xi_1\xi_3 - \xi_2\xi_3) \end{aligned}$$

The coefficients η_1 and η_2 have physical significance: η_1 measures the resistance of the material to uniform slow expansion or contraction, while η_2 does the same for slow shearing flows. They are called the *viscosity coefficients* of the material.

We use formula (17) for arbitrary flows, not just for slow ones. It is not necessary to compute the eigenvalues explicitly, since the terms $\mathcal{H}$ and $\mathcal{J}$ can be computed directly from the matrix D', by the formulas

$$\mathcal{H} \;=\; \frac{1}{2}(d_2')^2 \tag{18}$$

$$\mathcal{J} \;=\; \frac{2}{3}\left((d_2')^2 - 3d_1'\right) \tag{19}$$

where d_0', d_1', d_2' are the coefficients of the characteristic polynomial of D', given by

$$d_0' \;=\; \det \mathrm{D}' \tag{20}$$

$$d_1' \;=\; \textstyle\sum_i \mathrm{D}'^{(2)}_{ii} \tag{21}$$

$$d_2' \;=\; \textstyle\sum_i \mathrm{D}'_{ii} \tag{22}$$

That is,

$$\omega_{f,f'} = \frac{\eta_1}{2}(d_2')^2 + \frac{2\eta_2}{3}\left((d_2')^2 - 3d_1'\right) \tag{23}$$

5 Finite element model

We describe the configuration of an elastic body by a simple finite element model. Namely, we approximate its shape by the union of tetrahedra (*elements*), with pairwise disjoint interiors, glued by their faces. Note that the kinetic energy, elastic energy, and dissipated power of the model are simply the sums of the corresponding terms for each element.

We restrict the deformations of the body so that, within each element, the configuration function is always an affine map of $\mathbf{R}^3$ to $\mathbf{R}^3$. We assume that the elastic moduli and viscosity coefficients of the material are equal for all points in the interior of each tetrahedron, and do not change with time. Finally, we assume that the total mass of each element is constant over time, and uniformly distributed within it. (Note that the *density* will vary as the element gets deformed).

5.1 Barycentric coordinates

Let T be a tetrahedron in U, with vertices $u_1, \ldots, u_4$. If u is a point in T, we can write it as a convex linear combination of the vertices u_i,

$$u = \alpha_1 u_1 + \alpha_2 u_2 + \alpha_3 u_3 + \alpha_4 u_4$$

where $0 \leq \alpha_i \leq 1$, for $i = 1, \ldots, 4$, and $\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 = 1$. The α_i's are called the *barycentric coordinates of u in T*. If the vertices $u_1, \ldots, u_4$ are mapped to the points $p_1, \ldots, p_4$ by some configuration f, the current position $p = f(u)$ of point u will be

$$p = \alpha_1 p_1 + \alpha_2 p_2 + \alpha_3 p_3 + \alpha_4 p_4 \tag{24}$$

Analogous interpolation formulas give the current velocity and acceleration of point u, in terms of the current velocities and accelerations of the vertices of T.

5.2 Kinetic energy

Let T be a tetrahedron of mass μ and vertex velocities $v_1, \ldots, v_4$. The kinetic energy of T is the integral

$$K_T = \int_0^1 \int_0^{1-\alpha_3} \int_0^{1-\alpha_3-\alpha_2} \frac{v^2}{2}(6\mu)d\alpha_1\, d\alpha_2\, d\alpha_3$$

where $v = \alpha_1 v_1 + \alpha_2 v_2 + \alpha_3 v_3 + (1 - \alpha_1 - \alpha_2 - \alpha_3)v_4$. This integral evaluates to

$$K_T = \frac{\mu}{20}\left(\sum_{i=1}^{4} v_i^2 + \sum_{i=1}^{4}\sum_{j=1}^{i-1} v_i \cdot v_j\right) \tag{25}$$

Observe that the kinetic energy formula (25) for a tetrahedron T depends only on the vertex velocities of T, and not on their positions. It follows that the mass matrix M, given by formula (4), is constant, depending only on the element masses and their adjacency relationships. Therefore, M needs to be computed only once, at the beginning of the simulation.

For the same reason, the derivatives of K in formula (5) are zero, so the total force F can be computed by

$$F_i = E_i - \frac{1}{2}\frac{\partial W}{\partial v_i} - \frac{\partial P}{\partial q_i} \tag{26}$$

The derivatives of P and W with respect to the state coordinates are computed efficiently with the technique of Baur and Strassen [16], which is basically a sistematic application of the chain derivation rule.

Moreover, it can be shown that, in this case, M is symmetric and positive-definite. Therefore, it admits a *Choleski decomposition* $\mathrm{M} = \mathrm{LDL}^\top$ where L is a lower triangular matrix with unit diagonal and D is a diagonal matrix with positive values [17]. Besides, M is quite sparse: it has exactly $3n_v + 6n_e$ non-zero entries, for a model with n_v vertices and n_e edges. For typical models, the factor matrix L is also sparse. Thus, by storing only the non-zero elements of L, the cost of evaluating the accelerations becomes practically linear in n.

5.3 Elastic energy

We will assume that for each tetrahedron T there is a configuration g_T of the body where T is *relaxed*, i.e. has zero elastic energy. If the current configuration f maps the vertices of T to points $p_i = [x_i, y_i, z_i]^\top$, the strain tensor $\mathrm{C} = \mathbf{J}(f \circ {g_T}^{-1})$ at any point u in the interior of T can be computed as $\mathrm{C} = \mathrm{B}\,\mathrm{A}^{-1}$, where

$$\mathrm{B} = \begin{bmatrix} x_2 - x_1 & y_2 - y_1 & z_2 - z_1 \\ x_3 - x_1 & y_3 - y_1 & z_3 - z_1 \\ x_4 - x_1 & y_4 - y_1 & z_4 - z_1 \end{bmatrix}$$

and A is computed in a similar way from the vertices of T in its relaxed configuration.

From the strain tensor, we compute the elastic energy density $\phi_f(u)$ by formula (16). Its integral over the tetrahedron T is the product of $\phi_f(u)$ by the tetrahedron's volume V_T in its relaxed configuration.

5.4 Viscosity losses

Similarly, the loss of energy due to viscous friction inside a tetrahedron is obtained by multiplying the energy loss rate per unit volume ω by the current volume of the tetrahedron.

Since $\mathrm{C} = \mathrm{BA}^{-1}$ and A is constant, C' reduces to $\mathrm{B}'\,\mathrm{A}^{-1}$. We compute D' from C and C', its characteristic coefficients d_0', d_1', d_2' by formulas (20–22), and the viscous power loss density ω by formula (23).

6 Inequality conditions and discrete events

In general, integration of the differential equations cannot continue forever. Besides the equality-type constraints, there are other inequality-type *conditions* that must be satisfied. In general, we consider conditions that can be expressed by inequalities $\Gamma_s(q, q', \lambda, t) > 0$ for $i = 1, \ldots, l$, where the Γ_s are continuous functions, and λ is a vector consisting of the Lagrange multipliers associated to the constraints. We call a triplet (q, q', λ) an *extended state* of the system.

Inequality conditions arise, for example, when we want to avoid interpenetration between bodies. This condition can be translated into a combination of algebraic inequalities applied to the vertex coordinates. Also, when a body is resting or sliding on another, the contact force must always push the bodies apart, rather than against each other. This condition can be written as $\lambda_r > 0$, where λ_r is the Lagrange multiplier associated with the contact force.

6.1 Discrete event detection

While integrating the equations of motion, we must stop the integration whenever we reach a state where one of the Γ_s is about to become negative. In order to continue the simulation past that moment, it will be necessary to change the system's state, the equations of motion, or the set of conditions that need to be satisfied. In any case, we'll then say that a *discrete event* has occured at that moment.

Suppose the integration took us from a valid extended state s_a at time t_a to a proposed extended state s_b at time t_b. We must check whether all functions Γ_s remained of the same sign during that interval. If they didn't, we must estimate the first instant after t_a, say t_e, at which one of the functions Γ_s becomes zero, and redo the integration from t_a to t_e. Since the path computed for the interval $[t_a, t_e]$ may deviate from that computed for $[t_a, t_b]$, we must redo the discrete event detection, too. (In order to prevent an infinite loop here, we must round t_e down, so that either $t_e = t_a$, or $|t_b - t_e|$ is bounded away from zero.)

We can suppose in practice that at most one function Γ_s becomes zero at any given instant. Simultaneous sign changes (which might correspond, for example, to bodies colliding in two or more points at the same time), have probability zero in general, and are meaningless anyway in the presence of numerical errors.

6.2 Detection by Hermite interpolation

Let g be a function of time that describes the evolution of a condition function $\Gamma(q, q', \lambda, t)$ along the system's trajectory. In order to check whether the condition $g(t) > 0$ was satisfied throughout the integration step, we use Hermite interpolation of order k for g in that interval; that is, a polynomial of degree $2k+1$ whose values and derivatives to order k agree with those of g at t_a and t_b.

If the function Γ depends only on q, and possibly on t (which is true, for example, for the non-penetration conditions), we use a Hermite interpolant with first-order continuity, that is, the cubic polynomial whose values and first derivatives agree with those of g at t_a

and t_b. These parameters are

$$
\begin{aligned}
g(t_a) &= \Gamma(q_a, t_a) \\
g'(t_a) &= \frac{\partial \Gamma}{\partial q}(q_a, t_a)^\top q'_a + \frac{\partial \Gamma}{\partial t}(q_a, t_a)
\end{aligned}
\tag{27}
$$

and similarly for t_b.

In order to check whether the cubic polynomial is positive throughout the interval, we rewrite it in terms of the Bernstein-Bézier basis [18], that is

$$
g(t) = P_1(1-u)^3 + P_2\, u(1-u)^2 + P_3\, u^2(1-u) + P_4\, u^3 \tag{28}
$$

where $u = (t - t_a)/dt$, $dt = t_b - t_a$, and

$$
\begin{aligned}
P_1 &= g(t_a) & P_4 &= g(t_b) \\
P_2 &= g(t_a) + \frac{dt}{3} g'(t_a) & P_3 &= g(t_b) - \frac{dt}{3} g'(t_b)
\end{aligned}
\tag{29}
$$

The advantage of this representation is that the value of $g(t)$ for any $t \in [t_a, t_b]$ is a convex combination of the coefficients $P_1, \ldots P_4$. Therefore, if these coefficients are all positive or all negative, the same can be said of $g(t)$ throughout the interval. If they have mixed signs, we bissect the interval with DeCasteljau's algorithm [18], and repeat the test in each half, recursively.

We cannot use this cubic approximation when the condition Γ depends on λ or on q', because we would need the derivatives λ' and q'', which are not available in general. In such cases, we use a straight-line aproximation (Hermite interpolant of order zero) between the values of $g(t_a)$ and $g(t_b)$.

6.3 Collision detection

Realistic animation requires the detection and handling of collisions between the objects. Since a flexible body can bend and collide with itself, we must view each exposed element of a flexible object as a separate body, and watch for collisions between all pairs of such elements. In fact, we only need to watch for collision between a vertex and a non-adjacent face, or two non-adjacent edges; all other combinations are "coincidences" that occur with probability zero.

In either case, the collision can be detected by watching the signs of certain *indicator functions* $g_0, g_1, \ldots$, that depend on the coordinates of the four vertices involved. Specifically, for collisions between a vertex v and a face $u_0u_1u_2$, we use the functions

$$
\begin{aligned}
g_i &= (v - u_i) \cdot r_i \quad (i = 0, 1, 2) \\
g_3 &= n \cdot (v - u_0)
\end{aligned}
\tag{30}
$$

where $n = (u_1 - u_0) \times (u_2 - u_0)$ is the face normal, and $r_i = \vec{n} \times (u_{i+1} - u_i)$ is a vector parallel to the face and orthogonal to the edge $u_{i+1} - u_i$. (The indices of u are modulo 3.) A collision occurs when g_3 changes from positive to negative, provided g_0, g_1, g_2 are positive.

For collision between two edges $a = u_0 u_1$ and $b = v_0 v_1$, belonging to tetrahedra T_a and T_b, we use the five functions

$$\begin{array}{rclcrcl} g_0 & = & (v_0 - p) \cdot r & \qquad & g_1 & = & (p - v_1) \cdot r \\ g_2 & = & (u_0 - q) \cdot s & & g_3 & = & (q - u_1) \cdot s \\ & & & g_4 = \det(u_0, u_1, v_0, v_1) & & & \end{array} \tag{31}$$

where: p and q are the midpoints of the edges of T_a and T_b opposite to a and b, respectively; $r = (u_0 - p) \times (u_1 - p)$; and $s = (v_0 - q) \times (v_1 - q)$. A collision occurs when either g_4 becomes negative while $g_0, \ldots, g_3$ are all positive, or g_4 becomes positive while $g_0, \ldots, g_3$ are all negative.

6.4 Accelerated collision detection

Checking for collisions among all possible vertex-face and edge-edge pairs would cost $\Theta(m^2)$ operations, where m is the number of exposed triangles. For typical models, where m is in the hundreds or thousands, this cost would be excessive.

Fortunately, most of these pairs are widely separated in space. To take advantage of this fact, we compute an axis-aligned bounding box for each exposed vertex, edge, and face, over the current integration interval, and consider only pairs of elements whose bounding boxes intersect.

The bounding box of a surface vertex is computed by applying Hermite interpolation and Bernstein-Bézier range estimation to each coordinate, as in section 6.2. The bounding box for an edge is then obtained by enclosing the bounding boxes of its endpoints; and similarly for each face.

In order to quickly find the pairs of boxes that intersect, we exploit the spatial and temporal coherence of the scene, by a technique due to Lin and Manocha [1]. We store the minimum and maximum coordinates of all boxes in three sorted lists L^0, L^1, L^2, for the x, y, and z axes, respectively. We also keep three boolean matrices S^0_{ij}, S^1_{ij}, S^2_{ij}, that record whether the bounding boxes of elements i and j overlap when projected on each axis. Finally, we maintain a set $\mathcal{S}$ of all element pairs (i, j) whose boxes overlap on all three axes.

At each integration step, we recompute the bounding boxes for the time interval in question, and reorder each list L^c, with the insertion sort algorithm. During the sort, whenever we swap two list entries we update the corresponding overlap flag S^c_{ij}. If this update causes S^0_{ij}, S^1_{ij}, and S^2_{ij} to become all **true**, we add the pair (i, j) to the set $\mathcal{S}$.

After reordering all three lists, we scan the set $\mathcal{S}$, and delete from it any element pairs (i, j) whose boxes do not intersect. The collision tests of section 6.3 are applied only to the pairs that remain in $\mathcal{S}$ after this scan.

The cost of reordering the lists L^c and updating the flags S^c_{ij} is proportional to the the number of swaps performed. If the tetrahedral elements are not too thin relative to their diameter δ, and their displacement in the integration step is small compared to δ, it is not hard to show that the expected number of swaps is only $O(m)$. The cost of scanning the set $\mathcal{S}$, deleting the non-overlapping pairs, and performing the accurate collision tests is proportional to the size of that set; if the tetrahedra are not too thin, each bounding box will intersect a constant number of other boxes, and hence the size of $\mathcal{S}$ will be $O(m)$, too. We conclude that the total cost of collision detection at each integration step, by this method, grows only linearly with the number of exposed faces.

7 Handling collisions

When two solid bodies collide, the material around the points of contact must deform, in order to prevent their interpenetration. The deformation gives rise to a contact force that tends to push the two bodies away from each other. The force disappears if and when the two bodies move apart.

For elastic bodies, the deformations are macroscopic, and therefore the contact forces are finite and have nonzero duration. Therefore, we have chosen to use a simple spring-based model for the contact forces. Specifically, when we detect a collision between two surface points a and b, we attach a virtual spring between them, and allow the integration to continue from the same state. The polyhedra representing the two bodies will then interpenetrate to some extent, but the spring will eventually stop and possibly reverse this motion. We remove automatically the spring if and when it starts pulling the two bodies towards each other, instead of pushing them apart.

The virtual springs have linear force and zero rest length, i.e. their stored energy is simply KL^2 where L is the current distance between the endpoints. The constant K must be chosen with some care: if too small, the bodies may push right through each other, and perhaps become tangled in a multitude of springs. If too strong, the integrator will have to use a small time step in order to follow the spring motion.

We have implemented only "sticky" (non-sliding) collisions. That is, a contact spring remains tied throughout its life to the same surface points (relative to the vertices involved). The sticky contact model still allows soft objects to roll against each other: springs get added at the front edge of the contact region, and removed from the back edge. Unfortunately, sliding contacts, which would be necessary for the realistic animation of slippery soft objects, seem to be very hard to implement, because of the many new kinds of discrete events that must be considered [11].

8 Experimental results

Our dynamic simulator consists of about 10000 lines of code. It was written in Modula-3 [19], a modern language for systems programming that is freely available for many platforms. To integrate the differential equation $q'' = \mathrm{M}^{-1}F$, we use the 4^{th} order adaptive method of Runge-Kutta-Fehlberg [17].

The simulator takes a description of the finite-element model, the initial state vector (the positions and velocities of all vertices), and a few other parameters; and outputs a sequence of state vectors, for specified frame times. The animation may then be played in "real time," with a simple viewer for animated triangles, or converted to high-quality images with standard rendering tools, such as POV-Ray [20].

8.1 Falling torus

Figure 3 shows a simple animation produced by our system. The model is a soft rubber torus, consisting of 480 tetrahedra and 180 vertices, with 320 exposed triangles. The model, relaxed and at rest, was initially positioned in an almost vertical plane, and allowed to falling under its own weight. Its motion was unconstrained except for one internal vertex which was fixed at the origin. Simulating its motion for 10 seconds after release took 3 hours and 36 minutes on a SPARC 1000 (1297 times slower than real time). In this animation, the collision detection machinery was turned off—the torus never got deformed to the point of colliding with itself.

The 24 frames shown cover about 3.83 seconds of simulated time. The accompanying graph (figure 4) shows how the various components of the system's energy changed along the 10 simulated seconds. The fact that the total energy remained practically constant is an indication that the integration was quite accurate.

8.2 Snapping torus

In the next example (figure 5), the dynamically animated parts of the scene are two vertical rubber cylinders, a rubber torus, and a rigid quadrangular platform. The posts and domes in the background were added after the simulation, to help visualize the camera motion. Together, those objects have 906 tetrahedra, 353 vertices (of which 34 are fixed), and 945 exposed faces.

Both cylinders have their base fixed to the platform. The left cylinder also has its top face magically fixed in space. Initially, the torus is floating in space, relaxed and at rest, around the left cylinder. An internal vertex of the torus is then forcibly dragged (by means of time-dependent positional constraints) along a path that starts halfway between the two cilinders, goes over the right cylinder, and then down behind it. At that moment (about 5 seconds into the animation), the vertex is released, and the objects are allowed to move on their own.

The original intent was to thread the ring around the second cylinder. However, the cylinder wasn't rigid enough, and got squashed as the ring came down on top of it. When the "handle" vertex was released, the ring snapped back, and went flying off.

9 Conclusions

Two main original features of our simulator are the provision of four independent elasticity and viscosity parameters with physical meaning, and the non-linear elastic forces that automatically preserve the topological consistency of the tetrahedral mesh. We have found that these features allow realistic simulation of soft objects under fairly large deformations.

Another feature of our simulator is the collision detection mechanism, that combines Lin and Manocha's incremental bounding box tests with Hermite interpolation of the indicator functions for the non-penetration conditions. We have found that, for small models, the cost of collision detection is comparable to that of computing the internal forces; and we expect it to grow like $O(n^{2/3})$ as the number of elements n increases.

The main problem we have encountered is that the contact springs often fail to get removed when they should. For instance, the contact springs sometimes allow a vertex to enter through a face and exit through an adjacent one; and our spring removal code isn't smart enough to detect such events. Thus, a collision between soft bodies, entailing dozens of simultaneous contacts, often results in them sticking permanently to each other by a tangle of unremoved contact springs. Unfortunately, handling the separation events in a reliable manner seems almost as hard as implementing sliding contacts. Clearly, this is a problem that must be solved before the dynamic simulation of deformable bodies becomes a practical animation tool.

Figure 3: Falling torus with one fixed vertex.

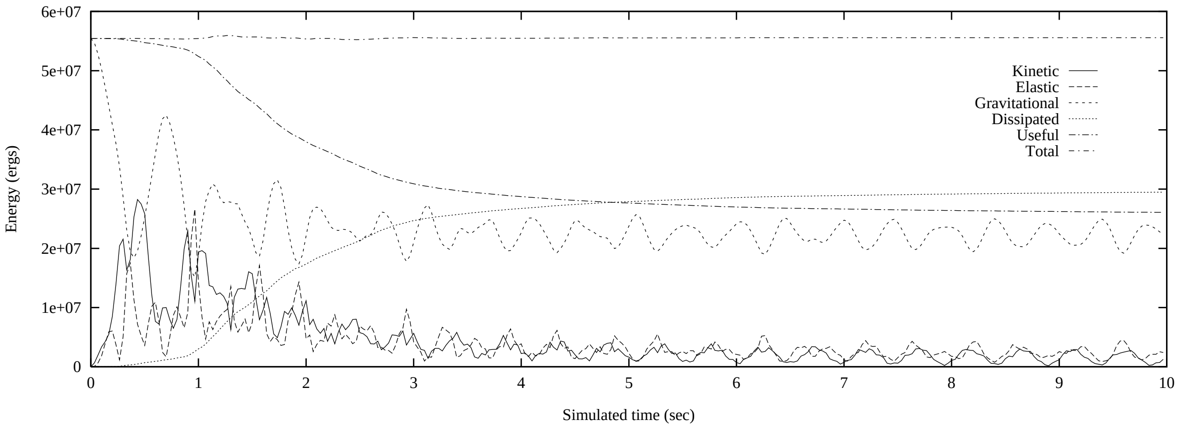


Figure 4: Energy evolution for the falling torus example.

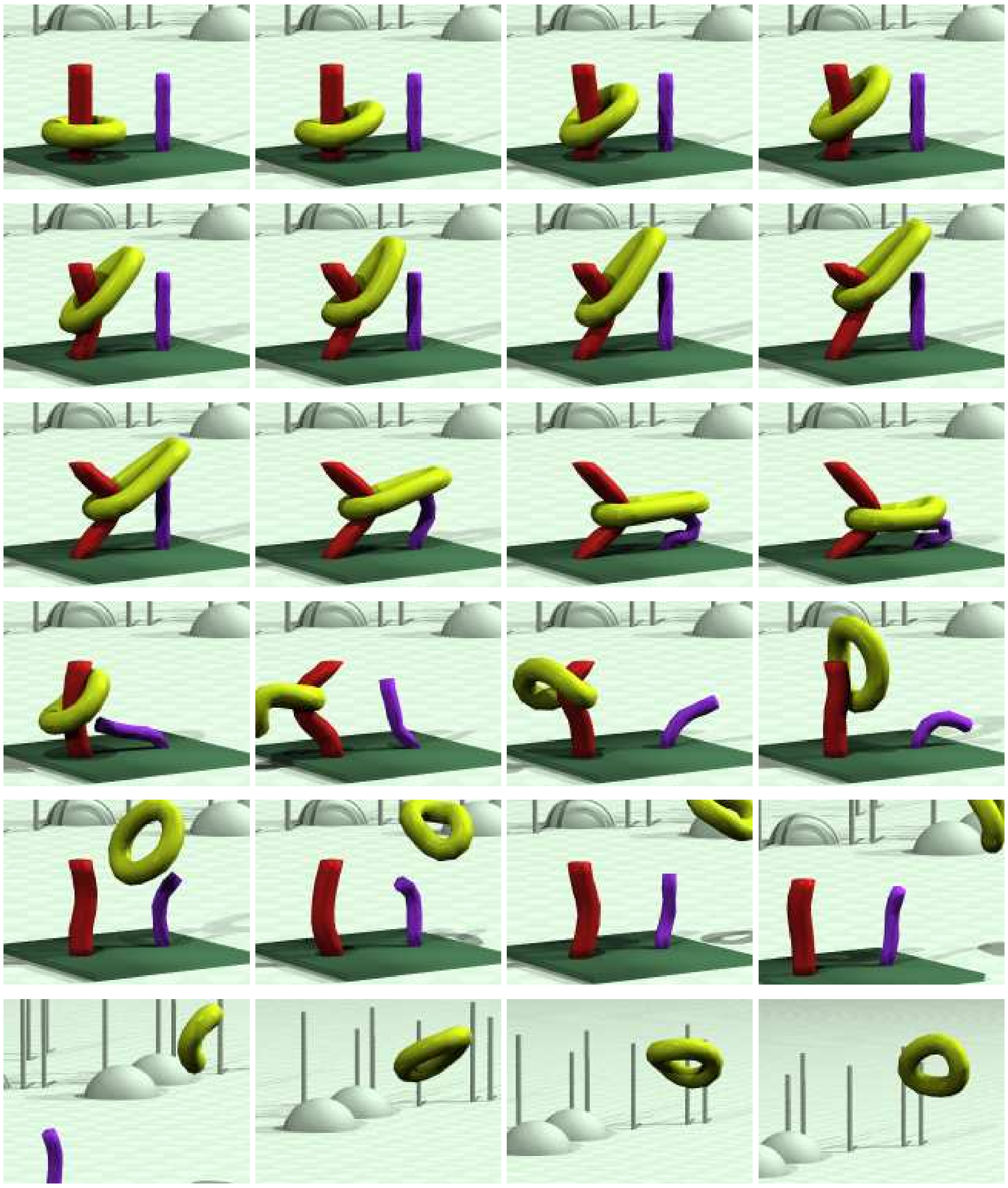


Figure 5: The snapping torus.

References

[1] M. C. Lin and D. Manocha, "Efficient contact determination between geometric models", tech. rep., University of North Carolina, Chapel Hill, NC, (1994).

[2] D. Terzopoulos, J. Platt, A. H. Barr, and K. Fleischer, "Elastically deformable models", *Computer Graphics (Proc. SIGGRAPH)*, **21**(4), pp. 205–214 (1987).

[3] D. Terzopoulos and K. Fleischer, "Modeling inelastic deformation: viscoelasticity, plasticity, fracture", *Computer Graphics (Proc. SIGGRAPH)*, **22**(4), pp. 269–278 (1988).

[4] J. Platt and A. H. Barr, "Constraint methods for flexible bodies", *Computer Graphics (Proc. SIGGRAPH)*, **22**(4), pp. 279–288 (1988).

[5] A. Witkin, M. Gleicher, and W. Welch, "Interactive dynamics", *Computer Graphics*, **24**(2), pp. 11–22 (1990).

[6] A. Pentland and J. Williams, "Good vibrations: modal dynamics for graphics and animation", *Computer Graphics (Proc. SIGGRAPH)*, **23**(3), pp. 215–222 (1989).

[7] A. Witkin and W. Welch, "Fast animation and control of nonrigid structures", *Computer Graphics (Proc. SIGGRAPH)*, **24**(4), pp. 243–250 (1990).

[8] M. Moore and J. Wilhelms, "Collision detection and response for computer animation", *Computer Graphics (Proc. SIGGRAPH)*, **22**(4), pp. 289–298 (1988).

[9] D. Baraff, "Curved surfaces and coherence for non-penetrating rigid body simulation", *Computer Graphics (Proc. SIGGRAPH)*, **24**(4), pp. 19–28 (1990).

[10] M. K. Ponamgi, D. Manocha, and M. C. Lin, "Incremental algorithms for collision detection between solid models", in *Proceedings of ACM/SIGGRAPH symposium on solid modeling*, pp. 293–304, (1995).

[11] D. Baraff, "Issues in computing contact forces for non-penetrating rigid bodies", *Algorithmica*, **10**, pp. 292–352 (1993).

[12] D. Baraff, "Fast contact force computation for non-penetrating rigid bodies", *Computer Graphics (Proc. SIGGRAPH)*, **28**(3), pp. 23–34 (1994).

[13] B. Mirtich and J. Canny, "Impulse-based dynamic simulation", tech. rep., Department of Computer Science,University of California, Berkeley, CA, (1994).

[14] B. Mirtich, "Hybrid simulation: combining constraints and impulses", tech. rep., Department of Computer Science, University of California, Berkeley, (1996).

[15] H. Goldstein, *Classical Mechanics.* Addison-Wesley, 2nd ed., (1980).

[16] W. Baur and V. Strassen, "The complexity of partial derivatives", *Theoretical Computer Science*, **22**, pp. 317–330 (1983).

[17] R. Burden and J. Faires, *Numerical Analysis.* PWS-Kent, 4th ed., (1989).

[18] J. D. Foley, A. van Dam, S. K. Feiner, and J. F. Hughes, *Computer graphics/Principles and practice.* Addison-Wesley, 2th ed., (1990).

[19] G. Nelson, ed., *Systems programming with Modula-3.* Prentice Hall Series in Innovative Technology, Prentice Hall, (1991).

[20] P.-R. Team, *Persistence of Vision Ray Tracer (POV-Ray) User's Documentation*, (1996).