Crashing Waves, Awesome Explosions, Turbulent Smoke, and Beyond:

Applied Mathematics and Scientific Computing in the Visual Effects Industry

Aleka McAdams, Stanley Osher, and Joseph Teran

Whether it's an exploding fireball in *Star Wars: Episode 3* or a swirling maelstrom in *Pirates of the Caribbean: At World's End*, special effects leveraging numerical simulations can be seen in a wide range of Hollywood blockbusters. Although previously considered too involved and prohibitively expensive for applications like movie special effects, as computers get faster and architectures evolve, simulation of such phenomena is now much more practical. Moreover, as the bar has been raised for increasingly realistic effects and even for computer-generated imagery to blend seamlessly with live performances, physically based simulation has become not only tractable but an invaluable tool for creating realistic virtual worlds in movies and video games.

Some of the physical phenomena commonly simulated in movie and video game special effects include water, fire, smoke, explosions, rigid body dynamics, and the deformation of elastic bodies. The governing equations for these processes are most often in the form of a system of partial differential equations. The development of algorithms for solving such equations with the computer is one of the cornerstones of applied mathematics and scientific computing. In fact, some techniques (most notably, level set methods) have completely revolutionized the industry and have even been honored with technical Academy Awards.

In this article we will describe some of the most compelling applications of applied math and scientific computing in the visual effects industry. Specifically we will talk about the techniques used to simulate every imaginable component of the digital environment, as well as the interactions between these components within a scene. Furthermore, we will discuss some of the ways in which physical simulation techniques for

Aleka McAdams is a graduate student in applied mathematics at UCLA. Her email address is amcadams@math.ucla.edu.

Stanley Osher is professor of mathematics and director of applied mathematics at UCLA. His email address is sjo@math.ucla.edu.

Joseph Teran is assistant professor of mathematics at UCLA. His email address is jteran@math.ucla.edu.

special effects differ from those developed for more classical applications in physics and engineering. Particularly, there are many cases in which the artistic vision of a scene requires a high level of controllability in the outcome of a simulation. To this end, special effects simulation tools, while physically based, must be able to be dynamically controlled in an intuitive manner in order to ensure both believability and the quality of the effect. We will highlight techniques from computational fluid dynamics, computational solid dynamics, rigid body simulation, and collision detection and resolution.

Computational Fluid Dynamics

Computational fluid dynamics (CFD) techniques are used to simulate a number of phenomena. Obviously, crashing waves and oceans can leverage CFD, but explosions, fireballs, and smoke effects all make use of CFD nowadays as well. Before the use of CFD, computer generated (CG) special effects such as explosions were driven by force fields applied to passive unconnected particles, producing less than realistic results. However, with the combination of improved hardware and faster algorithms, realistic CFD-based special effects for smoke, fire, water, and other fluids have become much more prevalent.

The governing equations for fluid dynamics can be derived from the principle of the conservation of mass and momentum. Although compressible fluid models have been used for some special effect applications (e.g., explosions and shock waves), these models are more difficult to solve numerically. As such, practitioners tend to use incompressible fluid models whenever possible, so we will overview their derivation here.

From an Eulerian frame of reference, i.e., a reference frame which is independent of the fluid's motion, an incompressible fluid's velocity is governed by the Navier-Stokes equations

$$\rho\left(\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v}\cdot\nabla)\mathbf{v}\right) = -\nabla p + \mu\nabla^2\mathbf{v} + \mathbf{f} \qquad \nabla\cdot\mathbf{v} = 0, \tag{1}$$

where ρ is density, p is pressure, μ is viscosity, and $\mathbf{f}$ represents outside forces.

The first equation can be derived directly from the conservation of momentum (with some simplifications from the conservation of mass and incompressibility), but perhaps a more intuitive derivation begins with Newton's Second Law, $F = ma$. The left side of the equation can be thought of as the "mass times acceleration" term with ρ corresponding to mass and $\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v}\cdot\nabla)\mathbf{v}$, or the material derivative of $\mathbf{v}$, corresponding to acceleration. The forces acting on the fluid can be divided into two types: those from internal stresses and those acting on the fluid from outside (e.g., gravity). For an incompressible fluid, the internal forces can be described by $-\nabla p + \mu\nabla^2\mathbf{v}$.

The second equation can be derived from the conservation of mass and the assumption that the fluid is incompressible. By the conservation of mass, the material derivative of density, i.e., the change in density of a parcel of advected fluid through time, is zero:

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0.$$

However, for an incompressible fluid, ρ is a positive constant, so this simplifies to $\nabla\cdot\mathbf{v} = 0$.

A number of numerical schemes are used in CFD to solve these equations. We can classify many methods commonly used in computer graphics into two categories: splitting or projection methods typically solved on a grid, and gridless methods typically solved using a smoothed particle hydrodynamics (SPH) algorithm.

Smoothed particle hydrodynamics (SPH) were initially developed for simulation of astrophysical problems. Müller et al. introduced the notion of using SPH to simulate liquids to computer graphics [25]. Particle methods such as SPH are good at simulating very active fluids (e.g., splashing water, smoke) in real time, making them excellent candidates for video game simulations, and more complex and high resolution particle simulations can be used to create realistic effects for motion pictures as well. Furthermore, this particle representation of the fluid makes it easy to determine the boundary between the simulated fluid and air.

As the name suggests, SPH represents the fluid as particles whose attributes (e.g., velocity, pressure, density) are spatially smoothed using a radially symmetric smoothing kernel $W(\mathbf{r})$. Any scalar quantity A is thus interpolated to a location $\mathbf{x}$ to produce the smoothed field

$$A_s(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} W(|\mathbf{x} - \mathbf{x}_j|),$$

where j iterates over all of the particles, m_j is the mass of the j^{th} particle, $\mathbf{x}_j$ is its location, ρ_j the density, and A_j the field value at $\mathbf{x}_j$. Derivatives on A can then be computed by differentiating W, e.g.,

$$\nabla A_s(\mathbf{x}) = \sum_j m_j \frac{A_j}{\rho_j} \nabla W(|\mathbf{x} - \mathbf{x}_j|).$$

Applying SPH to the Navier-Stokes equations then is a matter of representing the quantities in (1) with these weights. Furthermore, since $\frac{\partial \mathbf{v}}{\partial t} + (\mathbf{v}\cdot\nabla)\mathbf{v}$ in fact represents the time derivative of a fluid particle's velocity in an Eulerian frame, we can simplify this expression when using a particle representation with the material derivative $\frac{D\mathbf{v}}{Dt}$. Some complications arise when using SPH, particularly, the force terms (i.e., ∇p and $\nabla^2\mathbf{v}$)

are not symmetric, but these can be alleviated by using different approximations. (See [25] for one such approximation.)

Many grid-based methods use finite differencing on a staggered grid where the pressure is defined on cell centers and velocity components are defined on cell faces. The timestepping scheme then depends on the Helmholtz-Hodge decomposition, which states that any vector field $\mathbf{v}^*$ can be decomposed into $\mathbf{v}^* = \mathbf{v} + \nabla p$ where $\mathbf{v}$ is divergence-free and p is a scalar field. With this, the problem can be broken into two basic steps. In the first step, the velocity is advected (i.e., it is moved along the velocity field) and forces and diffusion are applied by solving

$$\rho\left(\frac{\partial \mathbf{v}^*}{\partial t} + (\mathbf{v}^* \cdot \nabla)\mathbf{v}^*\right) = \mu\nabla^2\mathbf{v}^* + \mathbf{f}$$

with initial data

$$\mathbf{v}^*|_{t=t^n} = \mathbf{v}^n.$$

This candidate velocity $\mathbf{v}^*$ is then made divergence-free by solving

$$\nabla^2 p = \nabla \cdot \mathbf{v}^* \tag{2}$$
$$\mathbf{v} = \mathbf{v}^* - \nabla p$$

for $\mathbf{v}$, the projection of $\mathbf{v}^*$ onto the divergence-free subspace. (See [34] for a frequently used discretization to these equations.)

While the Navier-Stokes equations and these discretizations are enough to determine much of the behavior of a fluid, additional quantities must be solved for and specialized treatment is needed to better capture the behavior of different types of fluids. We will look into grid-based methods for smoke, fire, and water in more detail.

Smoke

Made up of fine particles suspended in heated air, smoke can be modeled as an inviscid (i.e., $\mu = 0$) incompressible fluid. In addition to (1), the smoke density ρ_s (which is different from the constant air density) and the gas temperature T are evolved through the velocity field via

$$\frac{\partial \rho_s}{\partial t} + (\mathbf{v} \cdot \nabla)\rho_s = 0 \tag{3}$$

and

$$\frac{\partial T}{\partial t} + (\mathbf{v} \cdot \nabla)T = 0, \tag{4}$$

respectively. The force $\mathbf{f}$ is typically defined as a function of T and ρ_s and is used to capture the effects of buoyancy.

The finite difference scheme described in [34] can produce numerical dissipation and diffusion (i.e., modeled quantities such as velocity can lose energy and be smoothed out due to interpolation), especially on low-resolution grids. In the case of smoke, this can damp out the pluming and other interesting effects one would wish to capture in smoke. While a higher-resolution grid may be able to capture some of these effects, this would be computationally expensive. Instead, Fedkiw et al. introduced vorticity confinement techniques (created by Steinhoff for extremely turbulent flow fields about helicopters [35]) for smoke which are able to counteract this dissipation even on low-resolution grids [11]. Additional techniques to add even more turbulence or speed up computations have also been explored [31, 23]. Examples of smoke CFD special effects can be seen in *Terminator 3* [14] and *Star Wars: Episode 3* [15].

Fire

In many cases, fire can be modeled as an incompressible fluid in a way very similar to smoke. Again, vorticity confinement can be used to produce turbulent fireballs and swirling flames, and additional combustion particles can be added to the simulation to spark new flames [15].

While these combustion particles can produce gas expansion effects for fireballs and exploding spaceships, additional tools are necessary to simulate true fuel-based combustion such as a burning building or a gas-powered blowtorch. Nguyen et al. introduced the idea of modeling the gaseous fuel and the flaming gaseous products as separate (but coupled) incompressible fluids. The reaction front where fuel turns into fire is defined by a level set function which evolves according to the fuel velocity [26] (more about level set functions and their evolution can be found in the section "The Level Set Method"). These techniques were used to create the dragon's flaming breath in *Harry Potter and the Goblet of Fire*.

Water

Of course, one of the more obvious applications of CFD to special effects is the simulation of water-based phenomena. The state-of-the-art in water simulation techniques used in the movies are by and large based on the particle level set method introduced by Foster and Fedkiw [12], further refined by Enright et al. [10] and for which Fedkiw shared an Academy Award for Technical Achievement. Just as the reaction front for fire is defined by a level set function, the surface of the water can be defined in the same way. However, numerical dissipation and diffusion can cause the level set defined surface to lose volume in high curvature regions. The particle level set method provides a means to retain this volume by defining the water surface as a combination of a level set function and particles.

Several improvements and artist controls to this method have been introduced, and a number of films feature particle level set-based special effects, including the turbulent seas and flooding

Figure 1. (top) CG ocean simulated using the particle level set method in *Poseidon* [13]. (bottom) The surface of the tarmonster in *Scooby Doo 2* is modeled as a level set function and is evolved dynamically using CFD [40].

waters of *Poseidon* [13], the storm surge in *The Day After Tomorrow* [19], and the maelstrom in *Pirates of the Caribbean: At World's End*.

In Hollywood, CFD techniques aren't limited to modeling run-of-the-mill gases and fluids. The melting Terminatrix in *Terminator 3* [36] and the mud, beer, and other fluids in *Shrek* were simulated with particle level set methods. Additionally, CFD-inspired techniques have been used to simulate nonfluids such as hair (see the subsection "Cloth and Hair").

The Level Set Method

First introduced by Osher and Sethian in 1988 [29], the level set method provides a dynamic implicit representation of surfaces (see also [8] and [9] for interesting precursors). In this method, a closed curve (or set of curves) in $\mathbb{R}^2$ or closed surface(s) in $\mathbb{R}^3$ can be implicitly defined by the zero isocontour of a "level set function" which is defined throughout space. The implicitly defined surface can then be evolved by an underlying flow, handling topological changes such as merging and break-up automatically.

The most commonly used level set function is the "signed distance function". As the name suggests, the function is defined by the signed distance to the implicit surface where the sign is negative inside the surface and positive outside. For example, the function $\phi(x, y) = \sqrt{x^2 + y^2} - 1$ is the signed distance function for the unit circle. Signed distance functions have a number of properties which make them very useful for defining object surfaces in physical simulations. The fast-marching method of Tsitsiklis [28] makes initializing a signed distance function from an explicit representation (such as a triangulated surface) computationally inexpensive. By checking the sign of the function evaluated at a point, one can quickly determine if the point is inside or outside of the object. Furthermore, the surface normal (useful in collision handling) is defined by $\mathbf{n} = \frac{\nabla\phi}{|\nabla\phi|}$, and the curvature of the surface is simply $\kappa = \nabla \cdot \frac{\nabla\phi}{|\nabla\phi|}$.

The power of the level set method for two phase incompressible flows was originally shown in the computational mechanics literature by Sussman et al. [37]. Here, the idea of using an Eikonal equation-based reinitialization at every time step was first introduced, along with accurate advection schemes for the level set evolution. The method allowed for natural topology changes associated with pinching and merging in contrast with explicit representations of the boundary (e.g., by parameterized surfaces), where topology changes are more tricky.

One widely used application of the level set method in special effects is the particle level set method for fluid simulation (see the section "Water"). This method uses a combination of particles and a level set to define the water surface. The level set function, ϕ, can be evolved with the fluid velocity field, $\mathbf{v}$, by integrating $\phi_t + \mathbf{v} \cdot \nabla\phi = 0$. Changes in the fluid surface's topology (as occurs in the presence of splashing or air pocket formation/destruction) are automatically defined by the implicit surface. Numerical dissipation and diffusion can cause volume loss in high-curvature regions of the fluid surface, producing visible artifacts such as disappearing droplets. Here, the particle representation can compensate for this loss, and fast marching allows for easy surface reconstruction. Conversely, since the level set function is defined throughout space, the implicit surface can fill in low-curvature regions where the particle number is too low to properly resolve the surface. The convenient definition of curvature for level sets makes determining high/low curvature regions automatic. Additionally, this significantly simplifies the inclusion of effects such as surface tension which depend directly on the surface curvature.

Solids

Between complicated meshing problems and physics-based solids simulations, applied math

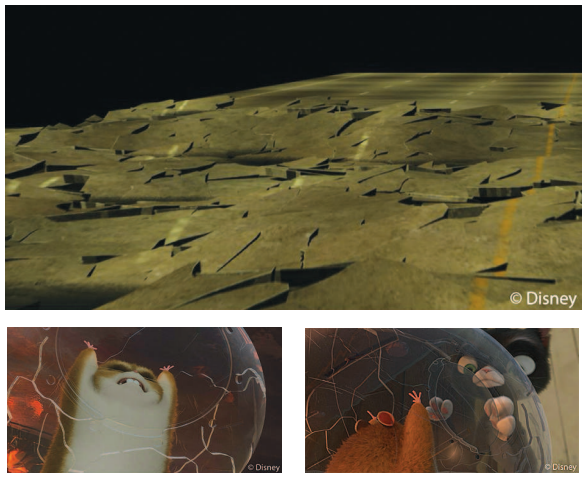

Figure 2. Geometric fracturing of Rhino's ball (bottom left, bottom right) and a roadway (top) in *BOLT* [18].

plays an integral role in CG sequences involving more than just fluids. Whether it's your favorite Pixar animated character or the apocalyptic cityscapes of *The Day After Tomorrow*, virtually every CG solid has an explicit representation as a meshed surface or volume. Mesh generation, deformation, and topological changes are all mathematically complex challenges faced by CG engineers. Artists turn to physics-based simulations in scenes where physical effects such as inertia can provide more realism or the complexity of the scene would be difficult to control by hand. Flesh simulations can endow CG characters with realistically bulging muscles and rippling fat. Hair simulators can tackle the complexity of thousands of interacting hairs, and rigid body simulations can control the dynamics of an exploding spacecraft.

Geometry, Meshing, and Fracture

Mesh generation is the first problem encountered when creating a new CG object. Artists typically work with surface meshes, which are created either by combining and deforming primitives or by converting laser-scanned data from a model into a triangulated or quadrilateral surface. One commonly used algorithm for building a triangulated surface and level set function from scanned data was introduced by Curless and Levoy [7]. First, the data from each scan is converted to a level set function defined over a discrete cubic lattice whose individual cells we will refer to as voxels. Voxels near the surface then take on signed distance function values, and voxels away from the surface are either marked as "empty" (between the surface and the sensor) or "unseen". The level set functions from each scan are then combined to form a single level set function. A surface extraction method such as marching cubes [22] is used to construct the triangulated surface at the zero-isocontour of the level set function. The surface is also extracted on boundaries between empty and unseen regions, which effectively fills any holes in the scan (Figure 3).

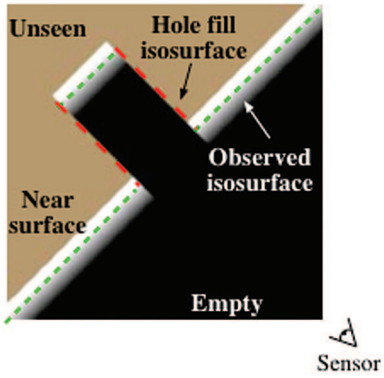

Figure 3. Extracting a surface from laser-scanned data [7].

Whereas artist versions of most CG objects are in the form of triangulated surfaces, tetrahedralized volume forms of the object are needed for most simulations. The first step of many algorithms is to define a body-centered cubic (BCC) or face-centered cubic (FCC) lattice on a uniform or octree grid. This initial lattice tetrahedralizes space with well-conditioned elements. The tetrahedralized volume is then modified to conform to the triangulated surface. The isosurface stuffing algorithm by Labelle and Shewchuk produces a well-conditioned mesh by snapping some vertices to the triangulated surface and refining others using a precomputed stencil [21].

Additionally, in scenes where an elaborately shaped triangulated surface cracks or shatters, it is necessary to somehow create new fragment meshes defined on the broken pieces. For the movie *BOLT*, Hellrung et al. [18] developed an algorithm which resolves every fragment exactly into a separate triangulated surface mesh and allows straightforward transfer of texture and look properties of the unfractured model. This method takes advantage of a cutting algorithm by Sifakis et al. [32] which cuts tetrahedralized volumes by duplicating cut elements and embedding the cut surface in these elements. In the embedding process, the cut elements are divided into "material" and "void" sections, defining regions inside and outside of the object, respectively [32]. In the first step of the triangulated surface algorithm, they generate a tetrahedral mesh that fully covers the object to be fractured. The triangulated surface of the unfractured object itself is used as the first cut in an application of the cutting algorithm,

Figure 4. Davy Jones rigid simulation rig from *Pirates of the Caribbean: Dead Man's Chest* [6].

effectively sectioning the background tetrahedral volume into "material" and "void" regions. The fracture surface is then applied as the second cut, resulting in the separation of the material volume into separate fragments. The cutting algorithm computes the triangulated boundary of every volumetric fragment in such a way that every triangle of a fragment is contained inside a triangle either of the uncut object or of the fracture surface.

Rigid Body Simulations

The simplest objects to simulate are rigid bodies. As the name suggests, these objects can only follow rigid motions (i.e., rotations and translations). Rigid bodies can be divided into two types: unconstrained rigid bodies, such as the pieces from a breaking up fighter plane in *Pearl Harbor*, and rigid bodies connected by joints which constrain their degrees of freedom (known as articulated rigid bodies), such as the battle droids in *Star Wars: Episode I* or Davy Jones' beard in *Pirates of the Caribbean: Dead Man's Chest* [6].

Each rigid body can be described by a state vector $X(t)$ consisting of its position $x(t)$, orientation $R(t)$ (or more typically a unit quaternion representing orientation), momentum $p(t)$, and angular momentum $L(t)$. By differentiating $X(t)$, we get to the known quantities that drive the simulation, force $F(t)$ and torque $\tau(t)$:

$$\frac{d}{dt}X(t) = \begin{pmatrix} v(t) \\ \omega(t)R(t) \\ F(t) \\ \tau(t) \end{pmatrix}.$$

Here, the velocity v is related to momentum by $mv(t) = p(t)$, where m is the mass of the body, and angular velocity ω is related to angular momentum by $I(t)\omega(t) = L(t)$, where $I(t)$ is the inertia tensor. Using an ordinary differential equations solver to numerically integrate $X'(t)$, the state vector (and thus the rigid body's position and orientation) can be updated at each timestep.

Figure 5. Hair's high geometric complexity presents a challenge to researchers. (left) A close-up of real hair. (right) Simulated hair [24].

The motion of articulated rigid bodies is constrained by the joints that connect the pieces. Various solution techniques exist for enforcing these constraints. One class of methods uses generalized coordinates to reduce the degrees of freedom, disallowing motions not permitted by the joint. By handling the constraints locally, these methods are fast but can break down in the presence of contact and collisions with other objects. These constraints can be enforced in a more global manner by using Lagrange multipliers [4] or impulses [39].

Deformable Object Simulations

Artists turn to deformable object simulations to handle complex systems such as hair, fur, and cloth or to give realistic responses to a wide range of objects, from CG food in *Ratatouille* to flesh simulations for characters in *Van Helsing*. Since the deformable object can stretch and bend, partial differential equations are needed to model the material dynamics. The object's behavior is usually described in terms of the relationship between its undeformed (or material) configuration B_0 and its deformed configuration at time t, B_t. This relationship is defined by $x(t) = \phi(X, t)$, where ϕ is a function which maps points $X \in B_0$ to points $x(t) \in B_t$.

The governing equations for these simulations come from continuum mechanics. The first, which is derived from Newton's second law, is the equation of motion, $\rho\ddot{\phi} = \nabla \cdot \mathbf{P} + \mathbf{f}$. Here, ρ is the density, $\mathbf{P}$ is the first Piola-Kirchoff stress tensor, and $\mathbf{f}$ represents external forces. Depending on the model, another stress tensor may be used in the place of $\mathbf{P}$. The second governing equation depends on the constitutive model which defines the stress-strain relationship for the material; in other words, it defines how the material deforms when stressed.

The linear elasticity model is the simplest constitutive model and is based on Hooke's law of elasticity. In this model, there is a linear relationship between stress and strain, typically denoted by $\sigma = \mathbf{C} : \varepsilon$, where σ is the Cauchy stress tensor,

ε is the infinitesimal strain tensor, and $\mathbf{C}$ is the fourth-order tensor of material stiffness. When the material is isotropic and homogeneous, this relationship can be simplified to $\sigma = 2\mu\varepsilon + \lambda tr(\varepsilon)I$, where μ and λ are the Lamé parameters and I is the identity matrix. Valid for small deformations, while it does not accurately model large deformations, it (and its rotation-invariant analogue) is still useful for a lot of graphics applications due to its easy computation. The neo-Hookean constitutive model is a generalization of linear elasticity for large deformations, and other even more general models exist for modeling soft rubber-like objects.

Just as important as the constitutive model to a simulation is the choice of discretization. Typically the simulated object will already have an explicitly meshed form (see the section "Geometry, Meshing, and Fracture"). The finite element method (FEM) [27] or finite volume method (FVM) [38] can then be applied to this mesh (or a triangulated/tetrahedralized version of this mesh) to numerically integrate the governing equations. Since high-order accuracy is usually not needed, linear interpolating functions are used for FEM, and σ and $\mathbf{f}$ are assumed to be constant on each element, so all integrands in the FEM or FVM formulations are constant and easy to compute.

Collision Detection and Handling

Rarely is an object simulated in isolation, so additional work is necessary to detect and handle contact and collisions with other objects. This problem can be tricky even in the simple scenario of two convex rigid bodies interacting. In highly complex collision scenarios (e.g., hair simulations), this problem can become intractable if standard geometric collision algorithms are used, so additional machinery is needed. In this section, we will describe the framework for a standard geometric collision detection and handling algorithm and highlight some recent work tackling the complexity of this problem with cloth and hair.

Geometric Collision Detection. Collisions can be detected either continuously or at the end of each simulation timestep. Since results only need to be physically plausible, not necessarily physically accurate, continuous collision detection and resolution are not typically used. For volumetric objects, collisions are detected by checking for any intersections at the end of the timestep. This type of detection can miss collisions where objects completely pass through each other in one step, but it is usually assumed that the simulation timesteps are small enough to avoid these situations.

Rigid body collisions are most easily checked by using level set representations of the objects. Intersections between two implicitly defined surfaces can then be found by testing sample points

Figure 6. (top) Penny's hair from *BOLT* is simulated using a clumping technique. Only a few hundred hairs were simulated, and the rest are added at render time [disneyanimation.com]. (bottom) All hairs are simulated in this hybrid volumetric/geometric collision algorithm [24].

(e.g., vertices of its triangulated surface representation) on one object with the level set function on the other. Of course, this check is not sufficient for edge-face collisions since all vertices are outside of the implicit surface. On high-resolution meshes this case can be ignored, but additional checks are necessary for low-resolution meshes.

This implicit surface-detection method cannot detect self-collisions for deformable objects or be used for objects without a level set representation (e.g., open surfaces and curves in 3D). In these cases, point-face and edge-edge collisions can be detected by considering the linear interpolations between time n and time $n+1$ configurations. Define a tetrahedron by the four vertices in a point-face or edge-edge pair. A collision is detected if at any point between time n and time $n+1$, the linearly interpolated positions of the four points become coplanar, i.e., the volume of the tetrahedron is 0. The time at which intersection occurs can then be determined by solving a cubic equation in time [5].

In both cases, acceleration structures are used to reduce the number of intersection tests that need to be performed. Structures such as bounding box hierarchies are used to eliminate element pairs

from the detection list which are too far away from each other.

Geometric Collision Handling. Although continuous collision resolution, in which collisions are resolved in the order in which they occur, can produce more accurate and stable results, it can also be prohibitively expensive when a large number of collisions need to be processed. Instead, less accurate (though physically plausible) solutions are found by simultaneously resolving all collisions detected in the timestep between time n and $n+1$.

The first step in collision resolution is to determine the normal direction between two colliding elements, $\mathbf{n}$, and the relative velocities of the two objects at the collision point, u_{rel}. When $u_{rel} \cdot \mathbf{n} < 0$, the objects are colliding, and something must be done to prevent interpenetration. In the absence of friction, the collision is then resolved by applying impulses to the colliding objects in the normal direction so that the new normal relative velocity is $u'_{rel} \cdot \mathbf{n} = -\epsilon u_{rel} \cdot \mathbf{n}$. Here, $\epsilon \in [0,1]$ is the coefficient of restitution and determines how "bouncy" the collision is. When $\epsilon = 1$, the collision is said to be perfectly elastic, and kinetic energy is conserved. When $\epsilon = 0$, the collision is perfectly inelastic, and all kinetic energy is lost. The choice of ϵ depends on the materials being modeled. If collisions are resolved in a pairwise fashion (which is common to many collision algorithms), new collisions between different pairs may be created by these impulses, so this algorithm must be iterated over more than once.

When $u_{rel} \cdot \mathbf{n} = 0$, the objects are considered to be in contact. In this case, the forces between the objects must be resolved to prevent the objects from accelerating toward each other. This case is much more difficult, as any force or impulse applied that is too strong can cause the objects to bounce apart, and the problem can be complicated when many objects are stacked on top of each other. One method, proposed by Baraff, analytically calculates the contact forces between objects [2, 3]. Another method by Guendelman et al, uses impulses similar to those for collisions; however, by partially ordering the contact pairs to work from the ground up, unwanted bouncing is avoided [16]. It is also possible to determine the necessary impulses to resolve all collisions and contact at once. See [20] for one such example. Friction can be handled in similar ways by applying impulses or modifying forces in the tangential direction, and the development of physically accurate models with nice numerical representations is an active area of research today.

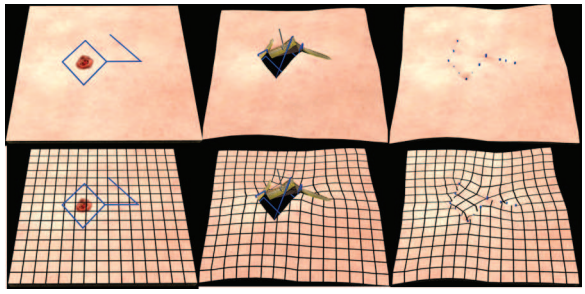

Figure 7. The images show a simulated malignant melanoma removal procedure using the framework of [33]. The bottom array of images uses an imposed surface texture to better visualize the post procedure equilibrium configuration of the tissue.

Cloth and Hair. Due to their complexity, CG cloth and hair/fur are almost always simulated rather than controlled by hand. However, this complexity also means that the standard geometric collision handling algorithms are not able to effectively resolve all collisions, and the cloth/hair can easily become tangled, producing visible artifacts.

For cloth, a modern approach is that of [5], which uses a three-stage process to ensure no collisions are missed. First, contact is preconditioned using penalty-based repulsions that are small enough to prevent visual artifacts. Then, in the second phase, collision impulses are iteratively applied, as in the subsection "Geometric Collision Handling". Third, rigid groups (where colliding objects are rigidly evolved) are used as a final safety net, postconditioning the collisions. One example of this algorithm used in practice is Yoda's robes in *Star Wars: Episode II*.

Hair simulation is even more complicated than cloth due to the massive number of hairs interacting and colliding. Instead of simulating each individual strand, the most common methods manage the complexity of many hairs interacting by simulating a smaller set of guide hairs (typically no more than several hundred) and interpolating a larger number of hairs for rendering. The use of sparse clumps often allows the use of inexpensive repulsion penalties as opposed to true geometric collision handling because guides are expected to have thickness. Examples of this technique include Penny's hair in *BOLT*. Alternatively, there have been several methods that treat every simulated hair as part of a fluid-like continuum volume [1, 17, 30]. These approaches naturally model hair interaction without explicit collisions.

More recently, McAdams et al. [24] explored a hybrid technique factoring hair computation into two parts: a coarse, highly coupled volumetric behavior, which is efficiently modeled by a continuum, and a finer, more locally coupled

Lagrangian particle simulation of hair. However, unlike previous continuum-based approaches that only simulate guide hairs that do not interact directly, this method simulates many hairs (several thousand) that are allowed to collide directly via the same geometric collision handling algorithm as used for cloth. The volumetric step works by projecting out any divergence in the hair's velocity field using the projection method as described in (2) for computational fluid dynamics. Geometric collision handling is only tractable because the volumetric step of the algorithm handles bulk collision behavior, leaving fewer collisions to be resolved.

Scientific Computing in Real Time

Simulations using numerical methods for partial differential equations are being used at an increasing rate due largely to improvements in processor speed and the availability of multicore processors. One very new example of this is real-time scientific computing—that is, simulations that run fast enough that a user can interact with them while the simulation is running. Perhaps the most obvious way to make real-time simulation possible is to reduce the complexity of the model by coarsening the representations of the simulated objects (e.g., using a coarser grid for fluid simulation or a lower resolution mesh for deformable object simulation). However, in many cases, the level of coarsening necessary for real-time application can produce unacceptably poor results. As such, one active area of research is in model reduction, where real-time low-resolution simulations make use of precomputed higher-resolution simulation information.

Such techniques will soon become commonplace in video games when creating scenes with natural phenomena. These techniques are also being used to simulate surgeries [33]. Figure 7 shows a finite element simulation of elastic soft tissues during a malignant melanoma procedure. The user makes incisions and deforms and sutures the tissue in real time while the simulator solves the governing equations.

Conclusion

With increased applicability of numerical methods in movies and video games, a new appreciation for applied math and scientific computing is emerging. This burgeoning aspect of the effects industry is serving as an exciting new frontier for applied mathematicians that uniquely combines the need for mathematical and computer science insights with the art of moviemaking. The nascent state of applied mathematics in the effects industry is driving research and development efforts to provide the algorithms and innovations needed to produce larger-scale and more impressive effects. That is, because scientific computing techniques are only recently becoming practical at a large scale in movies, their full efficacy and final place in the moviemaking process is far from determined. There will be efforts for years to come to fully realize the potential of applied math in the industry, and this can only be done by researchers with a strong background in mathematics, computer science, and physics.

Figure Credits

References

1. Yosuke Bando, Bing-Yu Chen, and Tomoyuki Nishita, Animating hair with loosely connected particles, *Computer Graphics Forum* **22**(3) (2003), 411–418.
2. D. Baraff, Analytical methods for dynamic simulation of non-penetrating rigid bodies, *Comput. Graph. (Proc. SIGGRAPH 89)* **23**(3) (1989), 223–232.
3. _____, Fast contact force computation for nonpenetrating rigid bodies, *Proc. SIGGRAPH 94*, 1994, pp. 23–34.
4. _____, Linear-time dynamics using Lagrange multipliers, *Proc. of ACM SIGGRAPH 1996*, 1996, pp. 137–146.
5. Robert Bridson, Ronald Fedkiw, and John Anderson, Robust treatment of collisions, contact and friction for cloth animation, *ACM TOG SIGGRAPH* **21** (2002), 594–603.
6. B. Criswell, K. Derlich, and D. Hatch, Davy Jones' beard: Rigid tentacle simulation, *SIGGRAPH 2006 Sketches*, 2006, p. 117.
7. B. Curless and M. Levoy, A volumetric method for building complex models from range images, *Comput. Graph. (SIGGRAPH Proc.)* (1996), 303–312.
8. A. Dervieux and F. Thomasset, A finite element method for the simulation of Rayleigh-Taylor instability, *Lecture Notes in Mathematics* **771** (1979), 145–159.
9. _____, Multifluid incompressible flows by a finite element method., *Lecture Notes in Physics* **141** (1980), 158–163.
10. D. Enright, Ronald Fedkiw, J. Ferziger, and I. Mitchell, A hybrid particle level set method for improved interface capturing, *Journal of Computational Physics* **183** (2002), 83–116.
11. Ronald Fedkiw, J. Stam, and H. Jensen, Visual simulation of smoke, *Proc. of SIGGRAPH 2001* (2001), pp. 15–22.
12. N. Foster and Ronald Fedkiw, Practical animation of liquids, *Proc. of SIGGRAPH 2001* (2001), pp. 23–30.

13. W. Geiger, M. Leo, N. Rasmussen, F. Losasso, and R. Fedkiw, So real it'll make you wet, *SIGGRAPH 2006 Sketches & Applications*, ACM Press, (2006).
14. W. Geiger, Nick Rasmussen, S. Hoon, and Ronald Fedkiw, Big bangs, *SIGGRAPH 2003 Sketches & Applications*, ACM Press, (2003).
15. ———, Space battle pyromania, *SIGGRAPH 2005 Sketches & Applications*, ACM Press, 2005.
16. E. Guendelman, R. Bridson, and R. Fedkiw, Nonconvex rigid bodies with stacking, *ACM Trans. Graph. (SIGGRAPH Proc.)* **22** (2003), no. 3, 871–878.
17. Sunil Hadap and Nadia Magnenat-Thalmann, Modeling dynamic hair as a continuum, *Computer Graphics Forum* **20** (2001), no. 3, 329–338.
18. J. Hellrung, A. Selle, E. Sifakis, and J. Teran, Geometric fracture modeling in *BOLT*, *SIGGRAPH 2009 Sketches & Applications*, ACM Press, 2009.
19. J. Iversen and R. Sakaguchi, Growing up with fluid simulation on "The Day After Tomorrow", *SIGGRAPH 2004 Sketches & Applications*, ACM Press, 2004.
20. Danny M. Kaufman, Timothy Edmunds, and Dinesh K. Pai, Fast frictional dynamics for rigid bodies, *SIGGRAPH '05: ACM SIGGRAPH 2005 Papers* New York, ACM, (2005), pp. 946–956.
21. Francois Labelle and Jonathan Richard Shewchuk, Isosurface stuffing: Fast tetrahedral meshes with good dihedral angles., *ACM Trans. Graph.* **26** (2007), no. 3, 57.
22. W. Lorensen and H. Cline, Marching cubes: A high-resolution 3D surface construction algorithm, *Comput. Graph. (SIGGRAPH Proc.)* **21** (1987), 163–169.
23. Frank Losasso, F. Gibou, and Ronald Fedkiw, Simulating water and smoke with an octree data structure, *ACM Trans. Graph. (SIGGRAPH Proc.)* **23** (2004), 457–462.
24. A. McAdams, A. Selle, K. Ward, E. Sifakis, and J. Teran, Detail preserving continuum simulation of straight hair, *ACM Trans. Graph. (SIGGRAPH Proc.)* (2009).
25. Matthias Muller, David Charypar, and Markus Gross, Particle-based fluid simulation for interactive applications, *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics Symposium on Computer animation* Eurographics Association, Aire-la-Ville, Switzerland, 2003, pp. 154–159.
26. D. Nguyen, Ronald Fedkiw, and H. Jensen, Physically based modeling and animation of fire, *ACM Trans. Graph. (SIGGRAPH Proc.)* **21** (2002), 721–728.
27. J. O'Brien and J. Hodgins, Graphical modeling and animation of brittle fracture, *Proc. SIGGRAPH 99*, vol. 18, 1999, pp. 137–146.
28. S. Osher and R. Fedkiw, *Level Set Methods and Dynamic Implicit Surfaces*, Springer-Verlag, New York, 2002.
29. S. Osher and J. Sethian, Fronts propagating with curvature-dependent speed: Algorithms based on Hamilton-Jacobi formulations, *J. Comput. Phys.* **79** (1988), 12–49.
30. Lena Petrovic, Mark Henne, and John Anderson, *Volumetric methods for simulation and rendering of hair*, Tech. Report 06-08, Pixar Animation Studios, 2005.
31. Andrew Selle, Nick Rasmussen, and Ronald P. Fedkiw, A vortex particle method for smoke, water, and explosions, *ACM Trans. Graph. (SIGGRAPH Proc.)* **24**(3) (2005), 910–914.
32. E. Sifakis, K. Der, and R. Fedkiw, Arbitrary cutting of deformable tetrahedralized objects, *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (in press), 2007.
33. E. Sifakis, J. Hellrung, J. Teran, A. Oliker, and C. Cutting, Local flaps: A real-time finite element based solution to the plastic surgery defect puzzle, *Studies in Health Technology and Informatics* **142** (2009), 313–8 (Eng).
34. J. Stam, Stable fluids, *Proc. of SIGGRAPH 1999*, ACM, 1999, pp. 121–128.
35. J. Steinhoff and D. Underhill, Modification of the Euler equations for "vorticity confinement": Application to the computation of interacting vortex rings, *Phys. of Fluids* **6**(8) (1994), 2738–2744.
36. N. Sumner, S. Hoon, W. Geiger, S. Marino, N. Rasmussen, and R. Fedkiw, Melting a terminatrix, *SIGGRAPH 2003 Sketches & Applications*, ACM Press, 2003.
37. M. Sussman, P. Smerka, and S. Osher, A level set approach for computing solutions to incompressible two-phase flow, *J. Comput. Phys.* **114**(1) (1994), 146–159.
38. J. Teran, S. Blemker, V. Ng, and R. Fedkiw, Finite volume methods for the simulation of skeletal muscle, *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, 2003, pp. 68–74.
39. R. Weinstein, J. Teran, and R. Fedkiw, Dynamic simulation of articulated rigid bodies with contact and collision, *IEEE Trans. on Vis. and Comput. Graph.* **12** (2006), no. 3, 365–374.
40. Mark Wiebe and Ben Houston, The tar monster: Creating a character with fluid simulation, *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, ACM, New York, 2004, p. 64.