Crashing waves, awesome explosions, turbulent smoke and beyond: applied mathematics and scientific computing in the visual effects industry

Stanley Osher and Joseph Teran

University of California, Los Angeles

1 Introduction

As computers get faster and architectures evolve, simulation of the dynamics of natural phenomena is becoming an increasingly indispensable tool for creating virtual worlds in movie special effects and video games. For example, nearly all companies involved in effects have a team dedicated to simulation-based dynamics of water, fire, smoke, explosions, rigid body dynamics and deformable/elastic bodies etc. 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, simulation of such phenomena is now much more practical on moderately powerful PCs. Also, the bar has been raised so high for realism in these industries that simulating the physics of such phenomena is necessary to produce effects at the state of the art. The governing equations for such phenomena come by and large from classical physics and 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. And although this is the geek version of the award, at least awardees get the chance to meet celebrity hosts like Jessica Alba!

In this article, we will describe 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 natural phenomena for every imaginable component of a digital environment (e.g. plants, clouds, clothing, hair, water etc). In addition to creating the look of the scene, it is essential to create realistic relative movement of all components of the scene. Failure to do so will be immediately perceptible and will cause the viewer to notice discrepancies with perceived real-life behavior. This can be desirable in some cases to create other worldly effects, but the degree to which the behavior deviates from real-life must be at least controllable. Furthermore, in the many movies that blend computer generated imagery with live performances (e.g. *Star Wars*, *Terminator*, *Pirates of the Carribean*), the effects need to blend seamlessly with real life materials and environments and so the ability to dynamically control the movement in addition to the look in a completely realistic manner can become critically important to ensuring the believability and quality of the effect. We will talk about the scientific computing techniques including computation fluid dynamics, computational solid dynamics, rigid body simulation, collision detection/resolution etc. used to make the viewing experience more realistic and controllable for the movie maker.

2 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.

Although compressible fluid models have been used for some special effect applications (e.g. explosions and shock waves), severe timestep restrictions for the associated numerical methods have led practitioners to use incompressible fluid models whenever possible. An incompressible fluid's velocity is governed by the Navier-Stokes equations

$$\begin{gathered}\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) = -\nabla p + \mu\nabla^2\mathbf{u} + \mathbf{f} \qquad (1)\\ \nabla\cdot\mathbf{u} = 0\end{gathered}$$

where ρ is density, p is pressure, μ is viscosity, and $\mathbf{f}$ represents outside forces. Depending on the application, additional variables and equations may be added.

A number of numerical schemes are used in CFD to solve these equations; a very popular technique is projection based methods with finite differencing on a staggered grid where the pressure is defined on cell centers and velocity components are defined on cell faces. This depends on the Helmholtz-Hodge decomposition of $\mathbf{u}$ and can be broken into two steps:

$$\rho\frac{\mathbf{u}^* - \mathbf{u}^n}{\Delta t} = -\rho\left(\mathbf{u}^n\cdot\nabla)\mathbf{u}^n\right) + \mu\nabla^2\mathbf{u}^n + \mathbf{f}$$

and

$$\begin{aligned}\nabla^2 p^{n+1} &= \frac{\rho}{\Delta t}\nabla\cdot\mathbf{u}^*\\ \mathbf{u}^{n+1} &= \mathbf{u}^* - \frac{\Delta t}{\rho}p^{n+1}\end{aligned}$$

While high-order accuracy is not needed (or even desired) for most special effects applications, additional application-dependent treatment is needed to produce the visually pleasing effects we see today.

2.1 Smoke

Made up of fine particles suspended in heated air, smoke can be modeled as an inviscid (i.e. $\mu = 0$) incompressible fluid with variable temperature and density. In addition to (1), the smoke density ρ and the gas temperature T are evolved through the velocity field via $\frac{\partial\rho}{\partial t} + (\mathbf{u}\cdot\nabla)\rho = 0$ and $\frac{\partial T}{\partial t} + (\mathbf{u}\cdot\nabla)T = 0$, respectively. The force $\mathbf{f}$ is typically defined as a function of T and ρ and is used to capture the effects of buoyancy.

The finite difference scheme described above can produce numerical dissipation, especially on low resolution grids. In the case of smoke, this dissipation 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 [32]) 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 [30, 23]. Examples of smoke CFD special effects can be seen in *Terminator 3* [14] and *Star Wars: Episode 3* [15].

2.2 Fire

In many cases, fire can be modeled as an incompressible fluid in a very similar way 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 blow torch. 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 is advected according to the fuel velocity [25] (more about level set functions can be found in Section 3). These techniques were used to create the dragon's flaming breath in *Harry Potter and the Goblet of Fire*.

2.3 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 received 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 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 waters of *Poseidon* [13], the storm surge in *The Day After Tomorrow* [20], 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*[33], tar monster in *Scooby Doo 2*, 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 non-fluids such as hair (see Section 4.4.3).

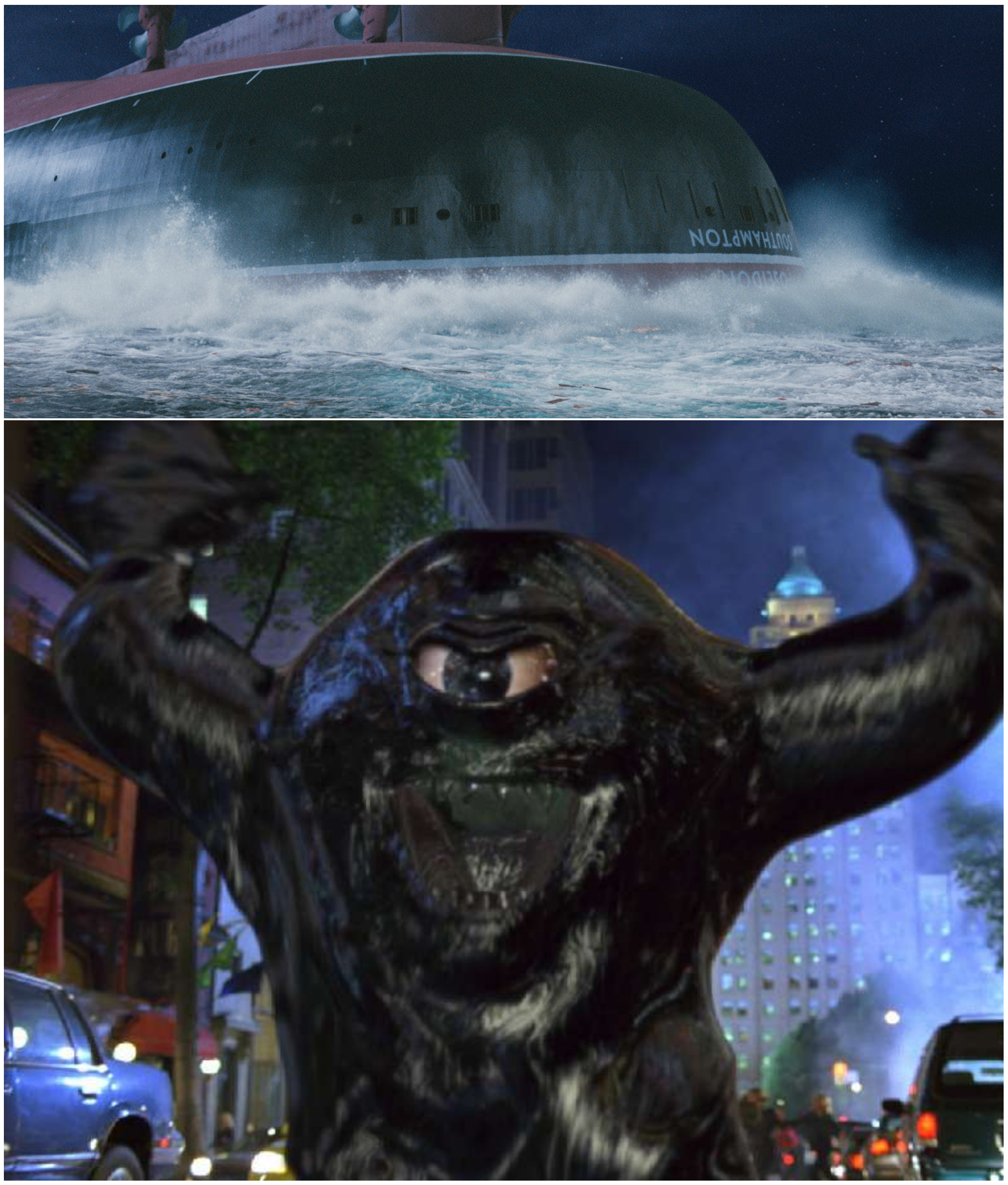

Figure 1: (top) CG ocean simulated using the particle level set method in *Poseidon*. (bottom) The surface of the tarmonster in *Scooby Doo 2* is modeled as a level set function and is evolved dynamically using CFD.

3 The level set method

First introduced by Osher and Sethian in 1988 [28], 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 advected 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 [27] 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 $\kappa = \nabla \cdot \frac{\nabla\phi}{|\nabla\phi|}$.

One widely used application of the level set method in special effects is the particle level set method for fluid simulation (see Section 2.3). This method uses a combination of particles and a level set to define the water surface. The level set function, ϕ can be advected with the fluid velocity field, $\mathbf{u}$, by integrating $\phi_t + \mathbf{u} \cdot \nabla\phi = 0$. 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 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 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 massively simplifies the inclusion of effects such as surface tension which depend directly on the surface curvature.

The power of the level set method for two phase incompressible flows was originally shown in the computational mechanics literature by Sussman et al [34]. 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 type phenomena without the emotional involvement usually associated with previous techniques.

4 Solids

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

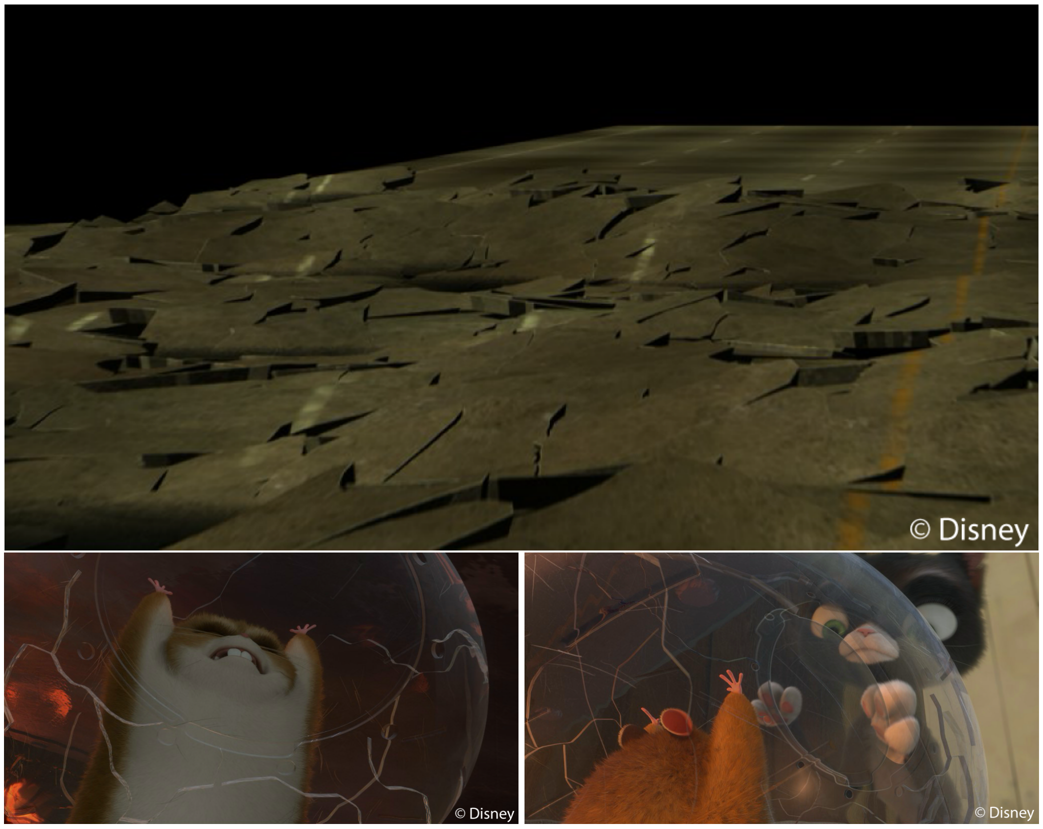

Figure 2: Geometric fracturing of Rhino's ball (bottom left, bottom right) and a roadway (top) in *BOLT*

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.

4.1 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 where voxels near the surface 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.

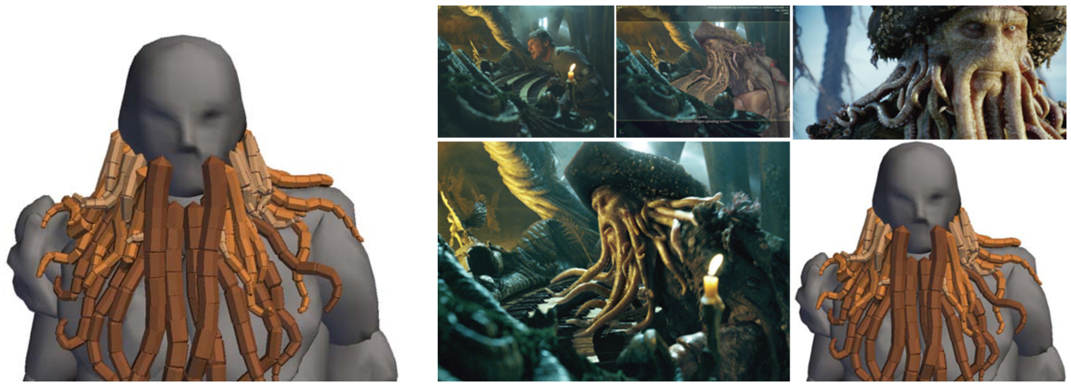

Figure 3: (left) Davy Jones rigid simulation rig from *Pirates of the Caribbean: Dead Man's Chest*.

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 un-fractured model. This method takes advantage of a cutting algorithm by Sifakis et al. [31] 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 [31]. 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 un-fractured object itself is used as the first cut in an application of the cutting algorithm, 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 a way that every triangle of a fragment is contained inside a triangle either of the uncut object, or of the fracture surface [18].

4.2 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].

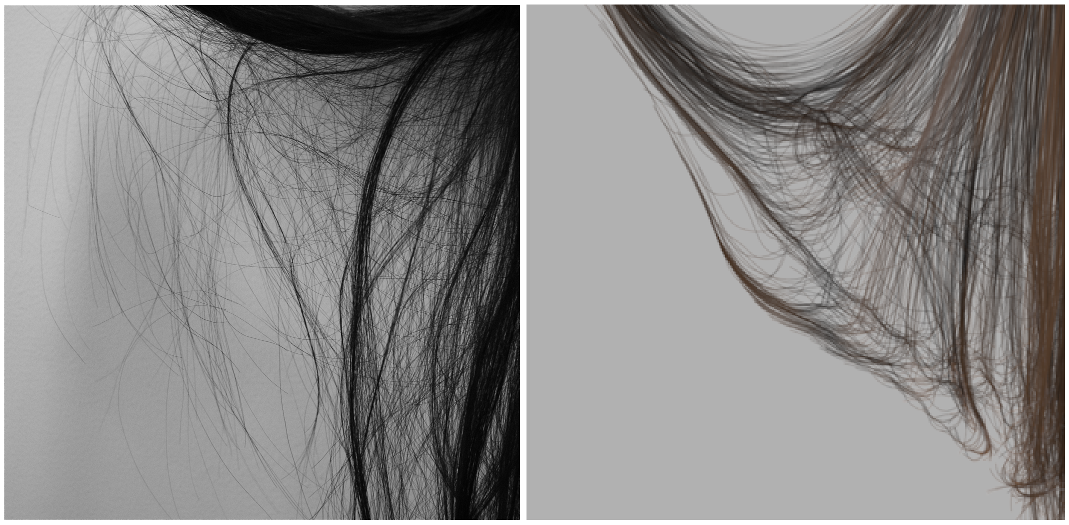

Figure 4: Hair's high geometric complexity presents a challenge to researchers. (left) A close-up of real hair. (right) Hair simulated by a new hybrid volume/geometric method.

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.

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 use 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 [36].

4.3 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 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 such as Mooney-Rivlin 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 Section 4.1). The finite element method (FEM) [26] or finite volume method (FVM) [35] 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.

4.4 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 collisions 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.

4.4.1 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 be found by testing sample points (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 prune element pairs from the detection list which are too far away from each other.

4.4.2 Geometric collision handling

Although continuous collision resolution, where 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, n, and the relative velocities of the two objects at the collision point, u_{rel}. When $u_{rel} \cdot n < 0$, 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 $u'_{rel} \cdot n = -\epsilon u_{rel} \cdot n$. 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. Since collisions are resolved in a pairwise fashion, new collisions between different pairs may be created by these impulses, so this algorithm must be iterated over more than once.

When $u_{rel} \cdot n = 0$, 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]. Friction can be handled in similar ways by applying impulses or modifying forces in the tangential direction.

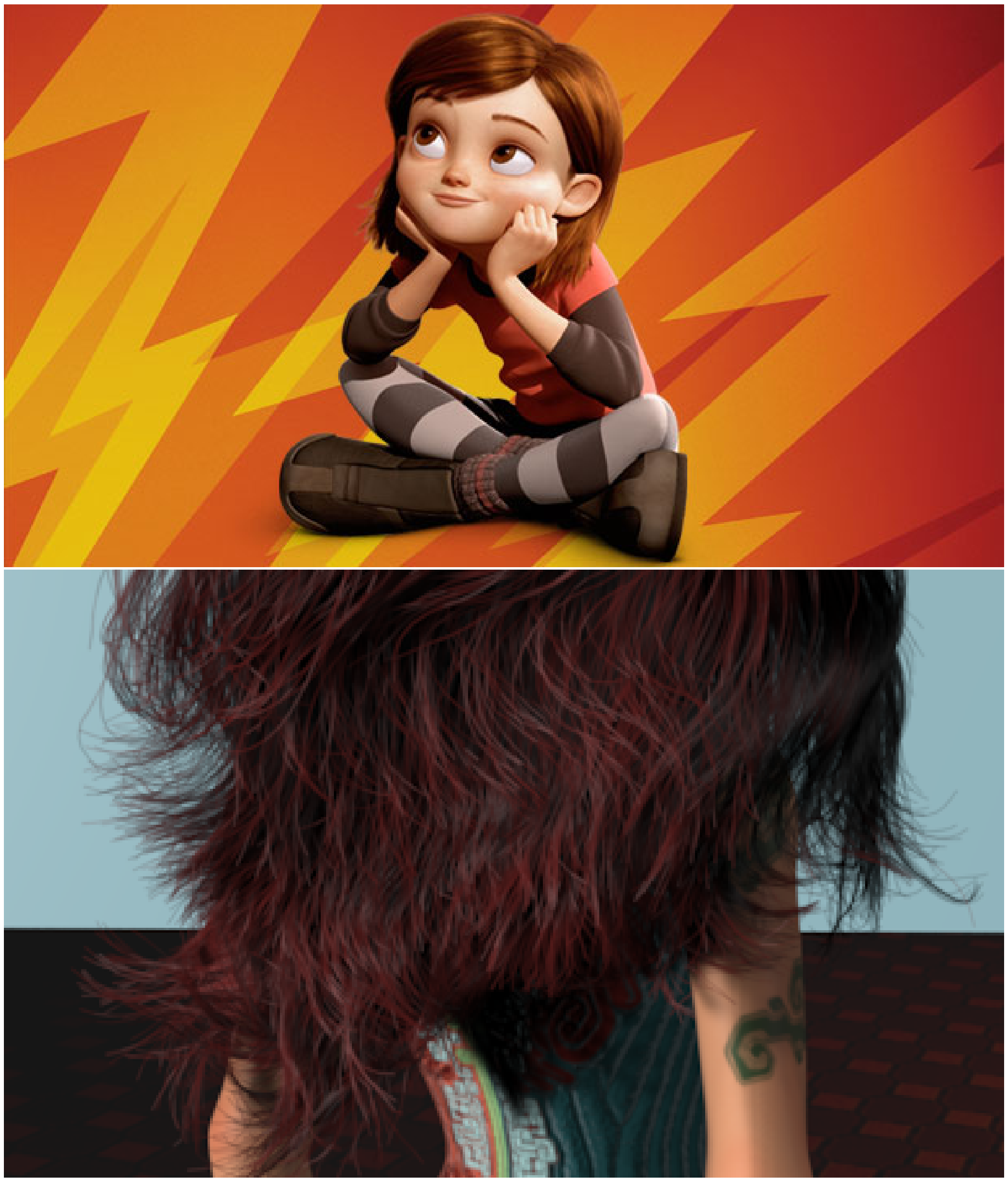

Figure 5: (top) Penny's hair from *BOLT* is simulated using a clumping technique. Only a few hundred hairs were simulated and the rest are added at render time. (bottom) All hairs are simulated in this hybrid volumetric/geometric collision algorithm [24].

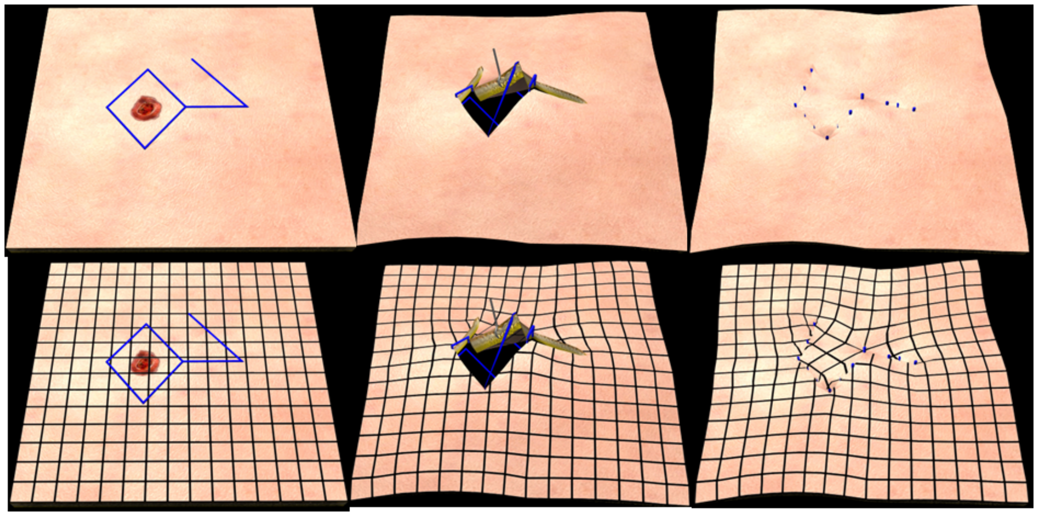

Figure 6: The images show a simulated malignant melanoma removal procedure using the framework of [19]. The bottom array of images uses an imposed surface texture to better visualized the post procedure equillibrium configuration of the tissue.

4.4.3 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 Section 4.4.2. Third, rigid groups (impact zones) are used as a final safety net, postconditioning the collisions. One example of this algorithm used in practice is Yoda's robes in *Star War: Episode I*.

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, 29]. 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 Section 2 on 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 [24].

5 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. 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 in while the simulation is running. Such techniques will soon become commonplace in video games when creating scenes with natural phenomena. This is also being used to simulate surgeries [19]. Figure 4.4.3 shows a finite element simulation of elastic soft tissues during a malignant melanoma procedure. The user makes incisions, deforms and sutures the tissue in real-time while the simulator solves the governing equations.

References

[1] Yosuke Bando, Bing-Yu chen, and Tomoyuki Nishita. Animating hair with loosely connected particles. In *Comp. Graph. Forum (Eurographics Proc.)*, pages 411–418, 2003.

[2] D. Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Comput. Graph. (Proc. SIGGRAPH 89)*, 23(3):223–232, 1989.

[3] D. Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proc. SIGGRAPH 94*, pages 23–34, 1994.

[4] D. Baraff. Linear-time dynamics using Lagrange multipliers. In *Proc. of ACM SIGGRAPH 1996*, pages 137–146, 1996.

[5] R. Bridson, R. Fedkiw, and J. Anderson. Robust treatment of collisions, contact and friction for cloth animation. *ACM Trans. Graph.*, 21(3):594–603, 2002.

[6] B. Criswell, K. Derlich, and D. Hatch. Davy jones' beard: rigid tentacle simulation. In *SIGGRAPH 2006 Sketches*, page 117, 2006.

[7] B. Curless and M. Levoy. A volumetric method for building complex models from range images. *Comput. Graph. (SIGGRAPH Proc.)*, pages 303–312, 1996.

[8] A. Dervieux and F. Thomasset. A finite element method for the simulation of rayleigh-taylor instability. *Lecture Notes in Mathematics*, 771, 1979.

[9] A. Dervieux and F. Thomasset. Multifluid incompressible flows by a finite element method. *Lecture Notes in Physics*, 141, 1980.

[10] D. Enright, R. Fedkiw, J. Ferziger, and I. Mitchell. A hybrid particle level set method for improved interface capturing. *J. Comput. Phys.*, 183:83–116, 2002.

[11] R. Fedkiw, J. Stam, and H. Jensen. Visual simulation of smoke. In *Proc. of ACM SIGGRAPH 2001*, pages 15–22, 2001.

[12] N. Foster and R. Fedkiw. Practical animation of liquids. In *Proc. of ACM SIGGRAPH 2001*, pages 23–30, 2001.

[13] W. Geiger, M. Leo, N. Rasmussen, F. Losasso, and R. Fedkiw. So real it'll make you wet. In *SIGGRAPH 2006 Sketches & Applications*. ACM Press, 2006.

[14] W. Geiger, N. Rasmussen, S. Hoon, and R. Fedkiw. Big bangs. In *SIGGRAPH 2003 Sketches & Applications*. ACM Press, 2003.

[15] W. Geiger, N. Rasmussen, S. Hoon, and R. Fedkiw. Space battle pyromania. In *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(3):871–878, 2003.

[17] S. Hadap and N. Magnenat-Thalmann. Modeling dynamic hair as a continuum. In *Comp. Graph. Forum (Eurographics Proc.)*, pages 329–338, 2001.

[18] J. Hellrung, A. Selle, E. Sifakis, and J. Teran. Geometric fracture modeling in bolt. In *SIGGRAPH 2009 Sketches & Applications*. ACM Press, 2009. to appear.

[19] J. Hellrung, E. Sifakis, J. Teran, A. Oliker, and C. Cutting. Local flaps: Surgical simulation and evaluation of excision strategies. *Studies in Health Technology and Informatics, Proceeding of Medicine Meets Virtual Reality 17*, 142:313–318, 2009.

[20] J. Iversen and R. Sakaguchi. Growing up with fluid simulation on "The Day After Tomorrow". In *SIGGRAPH 2004 Sketches & Applications*. ACM Press, 2004.

[21] Franois Labelle and Jonathan Richard Shewchuk. Isosurface stuffing: fast tetrahedral meshes with good dihedral angles. *ACM Trans. Graph.*, 26(3):57, 2007.

[22] W. Lorensen and H. Cline. Marching cubes: A high-resolution 3D surface construction algorithm. *Comput. Graph. (SIGGRAPH Proc.)*, 21:163–169, 1987.

[23] F. Losasso, F. Gibou, and R. Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 23:457–462, 2004.

[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. To appear.

[25] D. Nguyen, R. Fedkiw, and H. Jensen. Physically based modeling and animation of fire. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 21:721–728, 2002.

[26] J. Obrien and J. Hodgins. Graphical modeling and animation of brittle fracture. *Proc. ACM SIGGRAPH 99*, pages 137–146, 1999.

[27] S. Osher and R. Fedkiw. *Level Set Methods and Dynamic Implicit Surfaces*. Springer-Verlag, 2002. New York, NY.

[28] Stanley Osher and James A. Sethian. Fronts propagating with curvature dependent speed: Algorithms based on hamilton-jacobi formulations. *Journal of Computational Physics*, 79:12–49, 1988.

[29] L. Petrovic, M. Henne, and J. Anderson. Volumetric methods for simulation and rendering of hair. Technical report, Pixar Animation Studios, 2005.

[30] A. Selle, N. Rasmussen, and R. Fedkiw. A vortex particle method for smoke, water and explosions. *ACM Trans. Graph. (SIGGRAPH Proc.)*, 24(3):910–914, 2005.

[31] E. Sifakis, K. Der, and R. Fedkiw. Arbitrary cutting of deformable tetrahedralized objects. In *Proc. of ACM SIGGRAPH/Eurographics Symp. on Comput. Anim. (in press)*, 2007.

[32] 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):2738–2744, 1994.

[33] N. Sumner, S. Hoon, W. Geiger, S. Marino, N. Rasmussen, and R. Fedkiw. Melting a terminatrix. In *SIGGRAPH 2003 Sketches & Applications*. ACM Press, 2003.

[34] M. Sussman, P. Smerka, and S. Osher. A level set approach for computing solutions to incompressible two-phase flow. *J. Comput. Phys.*, 114(1):146–159, 1994.

[35] J. Teran, S. Blemker, V. Ng, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Proc. of the 2003 ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.*, pages 68–74, 2003.

[36] 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(3):365–374, 2006.