Eurographics/ ACM SIGGRAPH Symposium on Computer Animation (2009), pp. 1–10
E. Grinspun and J. Hodgins (Editors)

Energy Stability and Fracture for Frame Rate Rigid Body Simulations

Jonathan Su[†] Stanford University Intel Corporation
Craig Schroeder[†] Stanford University Pixar Animation Studios
Ronald Fedkiw[†] Stanford University Industrial Light + Magic

Abstract

Our goal is to design robust algorithms that can be used for building real-time systems, but rather than starting with overly simplistic particle-based methods, we aim to modify higher-end visual effects algorithms. A major stumbling block in utilizing these visual effects algorithms for real-time simulation is their computational intensity. Physics engines struggle to fully exploit available resources to handle high scene complexity due to their need to divide those resources among many smaller time steps, and thus to obtain the maximum spatial complexity we design our algorithms to take only one time step per frame. This requires addressing both accuracy and stability issues for collisions, contact, and evolution in a manner significantly different from a typical simulation in which one can rely on shrinking the time step to ameliorate accuracy and stability issues. In this paper we present a novel algorithm for conserving both energy and momentum when advancing rigid body orientations, as well as a novel technique for clamping energy gain during contact and collisions. We also introduce a technique for fast and realistic fracture of rigid bodies using a novel collision-centered prescoring algorithm.

Categories and Subject Descriptors (according to ACM CCS): Computer Graphics [I.3.5]: Computational Geometry and Object Modeling—, Physically Based Modeling

1. Introduction

Rigid body dynamics have been of great interest to the graphics community beginning with the early work of [Hah88, MW88, Bar89, Bar90, Bar91], and optimizing these algorithms to be more efficient was also of early interest, see e.g. [Bar94]. More recently, various authors have revisited rigid body simulation, focusing on a variety of phenomena [GBF03, KEP05, KSJP08]. There has also been interest in fracture [BHTF07] and magnetism [TGPS08]. Though all these rigid body techniques have been highly successful in the visual effects industry, the amount of computational power they require has prevented their use in real-time applications. However, when one examines today's real-time rendering systems it is clear that modern and next-generation hardware provides for the rendering of incredibly complex and detailed scenes in real-time, implying that significant computational resources are available. While real-time rendering systems must only compute once per frame—making all resources available for handling higher scene complexity—physics engines typically perform many small time steps per frame, trading spatial complexity for temporal complexity. Our stated goal is to simulate rigid bodies at the frame rate, taking one time step per frame so that we can simulate as detailed a scene as possible. While any basic simulation engine could be used as a starting point for this, we chose [GBF03], emphasizing that many of our ideas for robustly handling contacts, collisions, rigid body evolution, etc. could be adapted to other algorithmic frameworks with relatively little effort.

Numerical simulations are typically designed to converge to the correct solution as the time step goes to zero. Moreover, shrinking the time step improves stability, robustness, and accuracy. Therefore, constraining our simulations to go at the frame rate will pose problems for almost every part of the algorithm. [CR98] discusses some known flaws in state of the art collision models including common misconceptions, mentioning for example that the coefficient of restitution can be greater than one in frictional collisions. [ST00] discusses non-unique solutions and the impossibility of predicting which solution occurs in practice. We further illustrate that the standard evolution equations can lead to overall energy gain. Obviously, energy should be conserved if not dissipated due to damage and heat, and since this effect is highly exacerbated using large time steps (i.e. the 30 Hz frame rate), we propose a novel method for clamping collisions in order to guarantee that energy does not increase. We note there has been work on energy conservation with respect to collisions and contact in *deforming* bodies, as in [AP98]. Whereas one might debate the correct behavior,

[†] e-mail: {jonsu|cas43|fedkiw}@cs.stanford.edu

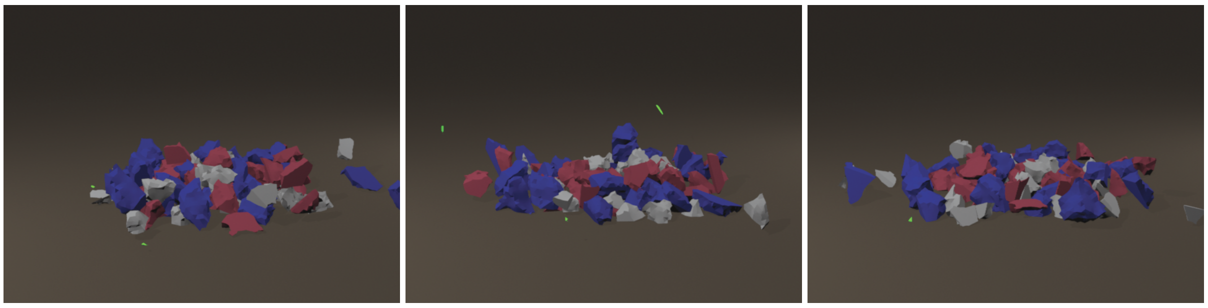

Figure 1: *The images above show a prescored pillar being dropped onto the ground with (left) a small time step, (center) a frame rate time step with no energy conservation, (right) a frame rate time step with energy conservation. During both the small time step simulation and the frame rate simulation with energy conservation, the small shards highlighted in green behave as expected. However, in the frame rate simulation with no energy conservation, the green shards bounce non-physically high into the air.*

it is pretty clear from the aforementioned references that this is not yet understood, and clamping the energy at least leads towards better plausibility in the sense of [BHW96]. Similar ideas are needed for processing contact.

Besides the known problems with contact and collision, rigid body simulations are prone to accuracy errors during evolution such as energy increase. These errors can be quite unacceptable for bodies that are rotating quickly or have ill-conditioned inertia tensors, such as those often introduced during simulations of fracturing bodies. These problems are well known and have been addressed by several authors, including [vZS07], who propose a rather complicated analytic solution to the problem which is still susceptible to instability due to round-off errors. [Vil08] instead focused on the conservation of the first integrals of the system. In the mechanics literature, there has been work on energy and momentum conserving rigid body dynamics, such as one of the algorithms in [SW91], but this algorithm requires the iterative solution of a fully nonlinear equation and therefore does not fit well with our goal of efficient simulation. In the same vein we propose a novel algorithm for conserving kinetic energy as rigid bodies rotate. In some sense this focus on energy conservation to provide better stability is analogous to variational integration, see e.g. [KYT*06, BRM09, KCD09].

Both virtual environments and video games have found that their users desire dynamic environments, and one of the simplest to provide is that governed by rigid body simulation. Whereas a number of systems provide basic rigid simulation in real time, much of the recent interest is focused on destruction scenarios. In fact, fracture has been of great interest to the graphics community for some time [NTB*91], and has regained recent popularity via [OH99, YOH00, OBH02]. Other notable works include the real time work of [MMDJ01], the virtual node algorithm for decomposing meshes [MBF04], and the fracturing of rigid bodies [BHTF07]. One can envision fracturing an object in two specific ways. The object can be prescored and subsequently broken apart based on a variety of rules, or one can compute the fracture dynamically using finite element analysis to determine internal stress. Prescoring the material is faster and thus desirable for real time simulations; however, a great deal of realism is lost because the object does not fracture based on the point of impact. We propose a novel method to *prescore all of space*, and then subsequently apply this prescoring in the specific location of impact to generate the fracture. This avoids the need for expensive algorithms that compute the stress and fracture dynamically while still breaking the objects based on the point of impact with the efficiency of prescoring. A further benefit of our approach is that it gives the artist more control over the fracture patterns without constraining where the fracture is initiated, i.e. where the collision takes place. (We refer the reader to a rather interesting paper on generating fracture patterns [IO06].)

2. Rigid Body Evolution

Simple translation through space is trivially integrated accurately at any time step, whereas orientation is non-linear and more complex. Typically one solves an ordinary differential equation using a scheme written to conserve angular momentum exactly, but kinetic energy is not necessarily conserved. In certain scenarios, a small time step is needed to guarantee accuracy, and a time step equal to the frame rate can cause an unacceptable gain in energy. For well-conditioned bodies with well-behaved inertia tensors and relatively slow rotations, the second order evolution suggested in [SSF08] adequately prevents energy growth. However, for poorly conditioned and rapidly rotating bodies evolved at the frame rate, we have observed energy increases of over an order of magnitude in a single time step even with higher order evolution. This can lead to significant and noticeable artifacts, such as those seen in Figure 1.

In order to properly conserve kinetic energy during evolution one must adjust the angular momentum or the orientation of the body. Since we desire to conserve momentum as well, we adjust the orientation. That is, the ordinary differential equation solver produces errors in kinetic energy and orientation, and our aim is to adjust those errors in orientation to conserve kinetic energy. We do this by analytically rotating the body to an orientation that has the same kinetic energy as the initial state. Since we do not change the angular momentum of the body, it is trivially conserved. Note that while this conserves angular momentum and energy, we do not properly resolve rotation, so the resulting orientations may still be inaccurate. If objects are moving very rapidly a large time step may cause collisions to be missed or resolved poorly. That said, the user is unlikely to notice that a collision occurred a frame too early or that the body had a slightly incorrect orientation when the collision normal was computed, nor is the user likely to notice that the body's orientation is inaccurate. Errors in momentum and energy, however, tend to produce noticeable artifacts.

2.1. Energy and Momentum Conserving Orientation

First, we show that there exist many nearby states that have the same kinetic energy and angular momentum. Consider the rotational kinetic energy of a body

$$KE = \frac{1}{2}\mathbf{L}^T\mathbf{I}^{-1}\mathbf{L}$$

where $\mathbf{L}$ is the angular momentum and $\mathbf{I}$ is the inertia tensor. If we change KE over time while keeping the angular momentum fixed, we have

$$\dot{KE} = \frac{1}{2}\mathbf{L}^T\,\dot{\mathbf{I}^{-1}}\,\mathbf{L} = -\frac{1}{2}\mathbf{L}^T\mathbf{I}^{-1}\,\dot{\mathbf{I}}\,\mathbf{I}^{-1}\mathbf{L} \tag{1}$$

Using the orientation $\mathbf{R}$ and object space inertia tensor $\mathbf{I}_0$, where $\mathbf{I} = \mathbf{R}\mathbf{I}_0\mathbf{R}^T$, we can expand $\dot{\mathbf{I}}$ to

$$\dot{\mathbf{I}} = \dot{\mathbf{R}}\,\mathbf{I}_0\mathbf{R}^T + \mathbf{R}\mathbf{I}_0\,\dot{\mathbf{R}}^{\,T} = \dot{\mathbf{R}}\,\mathbf{R}^T\mathbf{I} + \mathbf{I}\mathbf{R}\,\dot{\mathbf{R}}^{\,T} = \mathbf{u}^*\mathbf{I} + (\mathbf{u}^*\mathbf{I})^T$$

where we defined $\mathbf{u}^* = \dot{\mathbf{R}}\,\mathbf{R}^T$ as the cross product matrix for the vector $\mathbf{u}$ which represents our desired change in orientation. Substituting into (1),

$$\begin{aligned}\dot{KE} &= -\frac{1}{2}\mathbf{L}^T\mathbf{I}^{-1}(\mathbf{u}^*\mathbf{I} + (\mathbf{u}^*\mathbf{I})^T)\mathbf{I}^{-1}\mathbf{L}\\ &= -\mathbf{L}^T\mathbf{I}^{-1}(\mathbf{u}^*\mathbf{I})\mathbf{I}^{-1}\mathbf{L} = -\omega^T\mathbf{u}^*\mathbf{L} = -\mathbf{u}^T\mathbf{L}^*\omega\end{aligned}$$

Since KE is to be conserved, $\dot{KE} = 0$ and therefore $\mathbf{u}^T\mathbf{L}^*\omega = 0$. Thus $\mathbf{u}$ must lie in the space spanned by $\mathbf{L}$ and ω, or

$$\mathbf{u} = c_1\omega + c_2\mathbf{L},$$

where c_1 and c_2 are constants. Note that in the special case when $\mathbf{L}$ and ω are parallel, $\mathbf{u}$ can actually be any vector. In this case there will be no errors in the trivial integration of orientation. However, in general we have a two parameter-family of equations that conserve both angular momentum and kinetic energy. This suggests that there are many nearby states (i.e. orientations) that have both the same kinetic energy and angular momentum.

2.2. Computing the Energy Fix

Let KE be the kinetic energy obtained after evolution

$$KE = \frac{1}{2}\mathbf{L}^T\mathbf{I}^{-1}\mathbf{L} = \frac{1}{2}\mathbf{L}^T\mathbf{R}\mathbf{I}_0^{-1}\mathbf{R}^T\mathbf{L}$$

and let KE_0 be the kinetic energy that should have been obtained by energy conservation. We would like to adjust the orientation obtained from evolution to fix the kinetic energy by replacing $\mathbf{R}$ with $e^{\mathbf{u}^*}\mathbf{R}$. Let $\mathbf{u} = \varepsilon\mathbf{s}$, with the unit vector $\mathbf{s}$ being the axis of rotation and ε being the amount to rotate. We set the kinetic energy of the modified post-evolution state equal to KE_0

$$\begin{aligned}KE_0 &= \frac{1}{2}\mathbf{L}^T(e^{\mathbf{u}^*}\mathbf{R})\mathbf{I}_0^{-1}(e^{\mathbf{u}^*}\mathbf{R})^T\mathbf{L}\\ &= \frac{1}{2}((e^{\mathbf{u}^*})^T\mathbf{L})^T\mathbf{I}^{-1}((e^{\mathbf{u}^*})^T\mathbf{L})\end{aligned} \tag{2}$$

Expanding $e^{\mathbf{u}^*}$ where δ is the identity matrix, we get

$$\begin{aligned}e^{\mathbf{u}^*} &= e^{\varepsilon\mathbf{s}^*} = \delta\cos\varepsilon + \mathbf{s}^*\sin\varepsilon + \mathbf{s}\mathbf{s}^T(1-\cos\varepsilon)\\ (e^{\mathbf{u}^*})^T\mathbf{L} &= \mathbf{L}\cos\varepsilon - \mathbf{s}^*\mathbf{L}\sin\varepsilon + \mathbf{s}\mathbf{s}^T\mathbf{L}(1-\cos\varepsilon)\\ &= \mathbf{L}\cos\varepsilon - \mathbf{s}^*\mathbf{L}\sin\varepsilon\end{aligned}$$

which we can then substitute back into (2) to get

$$\begin{aligned}KE_0 &= \frac{1}{2}(\mathbf{L}\cos\varepsilon - \mathbf{s}^*\mathbf{L}\sin\varepsilon)^T\mathbf{I}^{-1}(\mathbf{L}\cos\varepsilon - \mathbf{s}^*\mathbf{L}\sin\varepsilon)\\ &= KE\cos^2\varepsilon - a\sin\varepsilon\cos\varepsilon + b\sin^2\varepsilon\end{aligned} \tag{3}$$

where we have defined $a = (\mathbf{s}^*\mathbf{L})^T\omega$ and $b = \frac{1}{2}(\mathbf{s}^*\mathbf{L})^T\mathbf{I}^{-1}\mathbf{s}^*\mathbf{L}$. Note that ε is never used directly but instead appears only as $\sin\varepsilon$ and $\cos\varepsilon$. Letting $x = \cos\varepsilon$ and $y = \sin\varepsilon$,

$$KE_0 = KEx^2 - axy + y^2b \tag{4}$$

Letting $c = KE - b$ and $k = KE - KE_0$, we obtain

$$\begin{aligned}axy &= k - cy^2\\ a^2(1-y^2)y^2 &= (k-cy^2)^2\\ 0 &= k^2 - (2kc + a^2)y^2 + r^2y^4\end{aligned} \tag{5}$$

where $r = \sqrt{c^2 + a^2}$. Solving for y^2 we get

$$\begin{aligned}y^2 &= \frac{2kc + a^2 \pm \sqrt{(2kc+a^2)^2 - 4r^2k^2}}{2r^2}\\ &= \frac{1}{r^2}\left(kc + \frac{1}{2}a^2 \pm a\sqrt{\frac{1}{4}a^2 + k(c-k)}\right)\\ &= \frac{1}{2r^2}\left(r^2 - cn \pm a\sqrt{r^2 - n^2}\right)\end{aligned}$$

where $n = c - 2k$. Finally, if we let $q = a\sqrt{r^2 - n^2}$, then

$$y = \frac{p_s}{r}\sqrt{\frac{r^2 - cn + p_bq}{2}} \qquad x = \frac{p_c}{r}\sqrt{\frac{r^2 + cn - p_bq}{2}} \tag{6}$$

where we have used p_s, p_c, and p_b (all equal to $\pm$ 1) to indicate signs that must be chosen.

From (4) we note that only the sign of xy, that is $p_s p_c$, is significant. For small angles of rotation, $\cos\varepsilon > 0$, so we choose $p_c = 1$. From (5) we choose $p_s = \mathrm{sgn}(k - cy^2)$. In the limit of very small errors in kinetic energy, the correction should also be very small, leading to the final sign choice $p_b = -\mathrm{sgn}(a)$.

2.3. Numerical Considerations

Direct use of (6) is not recommended. If $|k|$ is near floating point precision, then no significant error has been made and no correction is required. Similarly, if the kinetic energy is sufficiently small, then any energy increase that may have occurred is irrelevant to the simulation. Let $g = a^2 + 4k(c - k)$. If a/KE is very small (not robust), or $g < 0$ (infeasible search direction), then a fallback search direction should be used. Let $p = a^2 + g + 2|a|\sqrt{g}$.

$$x = \frac{\sqrt{p + (c+n)^2}}{2r} \qquad \hat{y} = \frac{2|k|}{\sqrt{p + (c-n)^2}}$$

where $y = \mathrm{sgn}(k - c\hat{y}^2)\hat{y}$.

2.4. Choosing a Rotation Axis

The previous section describes how to compute ε in a way that conserves the kinetic energy once a direction $\mathbf{s}$ is chosen. As discussed in section 2.1, if $\mathbf{u}$ is chosen as a linear combination of ω and $\mathbf{L}$, then kinetic energy is conserved. The locally optimal direction to choose is orthogonal to both, $\mathbf{s} = \mathbf{L}^*\omega/\|\mathbf{L}^*\omega\|$. This can be seen formally by noting that the Taylor series expansion of (3) is $KE_0 = KE - \mathbf{L}^T\mathbf{I}^{-1}\mathbf{s}^*\mathbf{L}\varepsilon = KE - \mathbf{s}^T(\mathbf{L}^*\omega)\varepsilon$. The magnitude of the linear term is optimized by choosing $\mathbf{s}$ to be in the direction of $\mathbf{L}^*\omega$.

One issue with choosing $\mathbf{s} = \mathbf{L}^*\omega$ is that only the angular momentum $\mathbf{L}$ remains constant as the object is rotated, and the angular velocity ω changes. Therefore, while $\mathbf{s} = \mathbf{L}^*\omega$ was a good choice for an instantaneous and small rotation, $\mathbf{L}^*\omega$ starts changing as the rotation is applied, and the best choice changes as well. Though locally optimal, this choice can fail for larger errors, in which case we simply choose a different $\mathbf{s}$.

At this point we fall back to using $\mathbf{s} = \mathbf{L}^*\mathbf{r}$, where $\mathbf{r}$ is either the maximum or minimum principal inertial direction, depending on the whether the current KE is greater than or less than KE_0. If $KE > KE_0$, then we want to rotate such as to align $\mathbf{L}$ and the largest principal inertial direction, $\mathbf{r}_{max}$. In this state, the body has the lowest KE it could possibly have, so $\mathbf{s} = \mathbf{L}^*\mathbf{r}_{max}$. If $KE < KE_0$, then we want to align $\mathbf{L}$ and the smallest principal inertial direction, $\mathbf{r}_{min}$, and so $\mathbf{s} = \mathbf{L}^*\mathbf{r}_{min}$. Note that since we conserve angular momentum exactly, and the maximum and minimum possible kinetic energy with this angular momentum is obtained by rotating in this way, this method of choosing directions is fail safe.

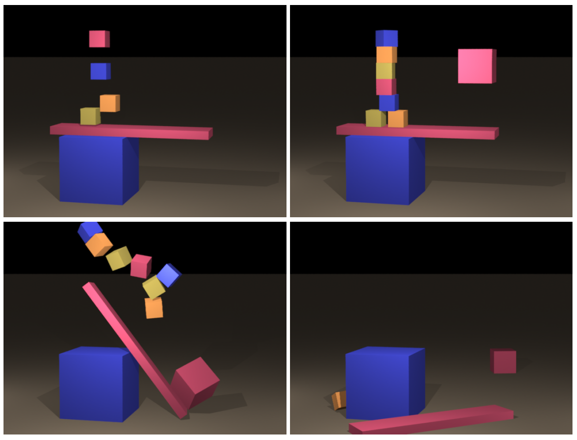

Figure 2: *Collisions and contact behave correctly using our energy clamping. In this example, we drop several boxes to form a stack. A larger box then disrupts this stack and scatters the objects.*

3. Collision and Contact Clamping

Contact and collision processing with Coulomb friction can also lead to energy gain, even in the otherwise very well-behaved example of a cube falling on the ground with a relatively small time step. Of course, taking a large time step exacerbates this problem. This limitation of the Coulomb friction model is also recognized and addressed in the literature, such as by replacing the Coulomb friction model with one that cannot increase energy [CR98]. However, this is insufficient in the case of collision or contact algorithms such as that of [GBF03], since position and velocity live at different points in time when the collision impulse is computed and applied.

3.1. Clamping in Split States

Many time integration schemes include states where velocities and positions live at different times, a condition we refer to as a split state. Clamping energy while in a split state can lead to very bad simulation artifacts. Consider the forward Euler scheme with gravity as the sole force, $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t\mathbf{v}^n$, followed by $\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t\mathbf{M}^{-1}\mathbf{g}$. Between the two updates lies a split state with position ahead of velocity. If the object is rising, so that $\mathbf{x}^{n+1} > \mathbf{x}^n$, one would calculate an increased potential energy and thus increased total energy in this split state. Clamping $\mathbf{x}^{n+1}$ to $\mathbf{x}^n$ removes this energy gain, but prevents the objects from rising under gravity. Similarly, if we consider the same forward Euler scheme computed in the reverse order, $\mathbf{v}^{n+1} = \mathbf{v}^n + \Delta t\mathbf{M}^{-1}\mathbf{g}$ followed by $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t\mathbf{v}^n$, we have a similar problem when falling under gravity. In the split state, we observe an increase in kinetic energy from gravity and clamp $\mathbf{v}^{n+1}$ to $\mathbf{v}^n$, preventing the object from falling correctly. From these examples, it is clear that conserving total energy is only meaningful when comparing non-split states, as momentary changes in

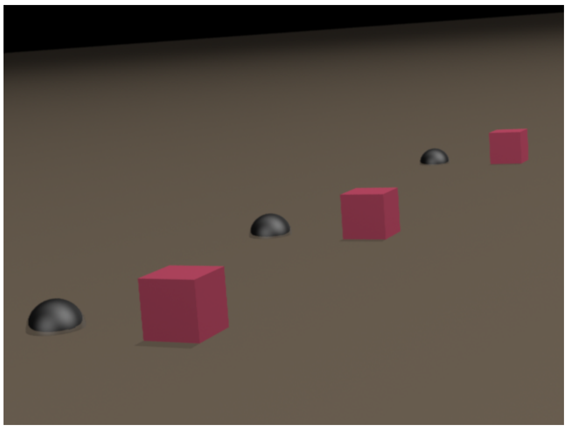
Figure 3: *A rigid cube sliding down an inclined plane and eventually stopping with friction, where the half sphere represents the analytic solution.*

total energy in split states are an inherent property of time integration schemes that utilize split states.

3.2. Clamping Impulses

In the case of both contact and collision processing, it is important to note that we do not modify positions. Therefore, potential energy is left unchanged, and preventing an increase in total energy is equivalent to preventing an increase in kinetic energy. First, we outline our general strategy for clamping in a split state, and subsequently explain how it is applied to both collisions and contact.

Consider two bodies A and B to which an impulse $\mathbf{j}$ and angular impulse $\mathbf{j}_\tau$ are to be applied to body A with the negations applied to body B to conserve momentum. We clamp the linear and angular impulses by scaling them by ε. Since kinetic energy is quadratic in velocity and the trivial solution $\varepsilon = 0$ results in an energy conserving impulse, there will always exist another real solution ε that can be found as follows. First, we write the kinetic energy of body A before and after the impulse is applied, KE_a and KE'_a respectively, in the non-split state.

$$\begin{aligned} KE_a =& \frac{1}{2}\mathbf{p}_a^T m_a^{-1}\mathbf{p}_a + \frac{1}{2}\mathbf{L}_a^T\mathbf{I}_a^{-1}\mathbf{L}_a \\ KE'_a =& \frac{1}{2}(\mathbf{p}_a+\varepsilon\mathbf{j})^T m_a^{-1}(\mathbf{p}_a+\varepsilon\mathbf{j}) + \\ & \frac{1}{2}(\mathbf{L}_a+\varepsilon\mathbf{j}_{\tau a})^T\mathbf{I}_a^{-1}(\mathbf{L}_a+\varepsilon\mathbf{j}_{\tau a}) \end{aligned}$$

where $\mathbf{p}$ is the linear momentum, m is the mass, and $\mathbf{j}_{\tau a} = \mathbf{r}_a^*\mathbf{j} + \mathbf{j}_\tau$. Note that both $\mathbf{j}$ and $\mathbf{j}_\tau$ will be computed in a split state, and in particular $\mathbf{r}_a$ will also be computed in that split state. This simply means that the impulses used to compute KE'_a will come from a split state computation, but as long as equal and opposite impulses are applied to each body we will conserve linear and angular momentum. That is, split state impulses have no effect on conservation of momentum and are fairly standard in the literature. However in our kinetic energy calculation, it is important to view every term in KE_a and KE'_a as existing at a non-split state.

The change in kinetic energy of body A is given by

$$\begin{aligned} KE'_a - KE_a =& \varepsilon(\mathbf{j}^T m_a^{-1}\mathbf{p}_a + \mathbf{j}_{\tau a}^T\mathbf{I}_a^{-1}\mathbf{L}_a) + \\ & \frac{1}{2}\varepsilon^2(\mathbf{j}^T m_a^{-1}\mathbf{j} + \mathbf{j}_{\tau a}^T\mathbf{I}_a^{-1}\mathbf{j}_{\tau a}) \end{aligned}$$

and the total change in kinetic energy including both bodies is

$$\begin{aligned} KE' - KE =& \varepsilon(\mathbf{j}^T m_a^{-1}\mathbf{p}_a + \mathbf{j}_{\tau a}^T\mathbf{I}_a^{-1}\mathbf{L}_a - \mathbf{j}^T m_b^{-1}\mathbf{p}_b - \mathbf{j}_{\tau b}^T\mathbf{I}_b^{-1}\mathbf{L}_b) \\ & + \frac{1}{2}\varepsilon^2(\mathbf{j}^T m_a^{-1}\mathbf{j} + \mathbf{j}_{\tau a}^T\mathbf{I}_a^{-1}\mathbf{j}_{\tau a} + \mathbf{j}^T m_b^{-1}\mathbf{j} + \mathbf{j}_{\tau b}^T\mathbf{I}_b^{-1}\mathbf{j}_{\tau b}) \end{aligned}$$

Setting $KE' - KE = 0$ to preserve the kinetic energy results in the trivial solution of $\varepsilon = 0$, as well as the nontrivial solution

$$\varepsilon = \frac{2(\mathbf{j}^T m_b^{-1}\mathbf{p}_b + \mathbf{j}_{\tau b}^T\mathbf{I}_b^{-1}\mathbf{L}_b - \mathbf{j}^T m_a^{-1}\mathbf{p}_a - \mathbf{j}_{\tau a}^T\mathbf{I}_a^{-1}\mathbf{L}_a)}{\mathbf{j}^T m_b^{-1}\mathbf{j} + \mathbf{j}_{\tau b}^T\mathbf{I}_b^{-1}\mathbf{j}_{\tau b} + \mathbf{j}^T m_a^{-1}\mathbf{j} + \mathbf{j}_{\tau a}^T\mathbf{I}_a^{-1}\mathbf{j}_{\tau a}}.$$

If $\varepsilon > 1$, then the impulse required to conserve kinetic energy is larger than the impulse we are applying. This simply means that Coulomb friction is reducing the kinetic energy and nothing need be done. If $0 < \varepsilon < 1$, this means that the proposed impulse is erroneously increasing the kinetic energy and thus we scale the impulses by epsilon in order to prevent any energy increase. Note that $\varepsilon < 0$ is impossible in a non-split state but can occur in a split state, representing an infeasible direction. In this case, we choose $\varepsilon = 0$, meaning that we discard the impulse altogether. Note that discarding the impulse leads to loss of friction, but in practice this case is very rare.

When one of the two bodies has infinite mass, such as the static ground plane or a moving kinematic body (consider for example an elevator), the above equations need a slight modification. For the case of a static body, all the terms pertaining to the body vanish as m^{-1} and $\mathbf{I}^{-1}$ are 0. One might assume the same applies to a kinematic body with non-trivial velocity, however this would preclude an object body from being lifted with an elevator. In particular, the terms relating to the infinite mass body only cancel in the sense of relative velocities. Thus, for a collision with a kinematic object we rewrite the velocities of the dynamic body in the relative frame of the kinematic object based on its pointwise velocity at the point of collision. Then the clamping can be done computing ε in the same manner as for a static body. After applying the impulse, the velocity of the dynamic body is scaled back, adding back the relative velocities. This preserves kinetic energy in the moving reference frame, giving the desired effect.

Collision Clamping The collision algorithm of [GBF03] applies collisions to the split state $\mathbf{x}^{n+1}$, $\mathbf{v}^n$. However, we have a valid non-split state to clamp against, since we can clamp the impulses applied to $\mathbf{v}^n$ in the state $\mathbf{x}^n$, $\mathbf{v}^n$. This

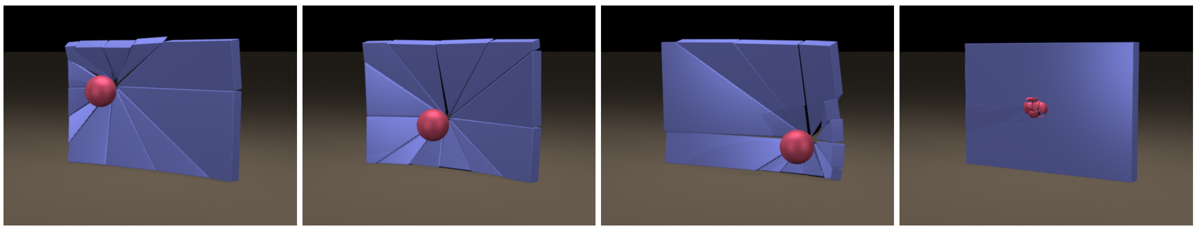

Figure 4: *Our algorithm for prescoring all of space allows us to center a fracture pattern around the point of impact. The above series of images shows a wall being fractured at 3 separate locations, with all the fractures being generated from the same fracture pattern. Finally, a sphere fractures as it hits the wall using the same fracture pattern. The level set resolution of all the walls used in our examples is 150x10x100.*

means that KE is computed using entirely time n quantities, as are all terms in KE' except those relating to $\mathbf{j}$ and $\mathbf{j}_\tau$. Instead, $\mathbf{j}$ and $\mathbf{j}_\tau$ are computed using the split state, just as in [GBF03] where in particular $\mathbf{r}_a$ (and $\mathbf{j}_{\tau a}$) are also computed using that split state.

Contact Clamping The contact algorithm of [GBF03] also involves a split state. We use a Newmark variant of this scheme introduced by [SSF08], which performs two slightly different contact steps, the first as part of a position update and the second as part of a velocity update, rather than one as in [GBF03]. The second contact step involves the split state $\mathbf{x}^{n+2}$, $\mathbf{v}^{n+1}$, but we observe that we can clamp against the non-split state $\mathbf{x}^{n+1}$, $\mathbf{v}^{n+1}$ using the method described above. This is identical to the treatment of collisions, where the split state was formed by updating the position one time step while leaving the velocity the same.

The first contact step involves the split state $\mathbf{x}^{n+1}$, $\mathbf{v}^{n+1/2}$. Since the only available non-split state at that point in the integration is at $\mathbf{x}^n$, $\mathbf{v}^n$, we tried clamping there. Unfortunately, clamping against that state leads to problems. For example, if bodies are initially at rest and in contact, any impulse will change $\mathbf{v}^n$ away from zero and increase energy, resulting in the impulse being clamped to zero. This prevents contacting bodies at rest from applying contact impulses and leads to jittering. Therefore, we do not clamp during the first contact step, and we have not observed any ill effects from this omission. This may be partially due to the fact that this first contact step is only used to update the position and its effects on the velocity are removed. Therefore, energy gain here may be less detrimental than during the second contact step or during the collision processing where the results are used to directly modify the velocity. However, in spite of our ability to obtain stable and accurate simulations at the frame rate, we do think that clamping the first contact step should be addressed as future work. That is, while the energy behavior of the scheme is highly improved by limiting any kinetic energy growth during evolution, collisions, and the contact phase of the velocity update, there may be scenarios where the kinetic energy growth during the contact phase of the position update could have adverse effects.

Observe that this approach to clamping does not rely on any of the details of the collision or contact algorithms being employed, beyond the relatively simple assumption that it will result in an impulse to be applied. It can be applied when the state is non-split, or when the state is split but a suitable non-split state is available. It is equally suitable for clamping the impulses obtained from any other process that affects the state only through the impulse computed.

4. Fracture

Fracture of rigid bodies has typically been achieved through prescoring methods or by dynamically computing the stresses of the object to determine how to fracture it. Prescoring is fast, but constrains the artist's control over the look of the fracture. The artist could make a realistic fracture pattern emanating from a particular point on the object and hope that the point of impact is close to the fracture center or make a somewhat uniform fracture pattern that is not centered about any point and is not very realistic. Computing fracture dynamically from the internal stresses of an object produces realistic, point-of-impact-centered patterns, but requires significantly more computation time and is therefore imprac-

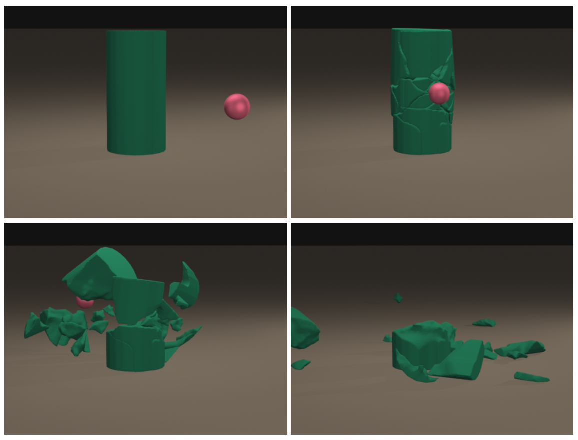

Figure 5: *A cylinder (level set resolution of 55x105x55) is shattered into 25 pieces by a sphere. This simulation averaged 22 frames/sec on a single 3.00GHz Xeon core using only algorithmic optimizations.*

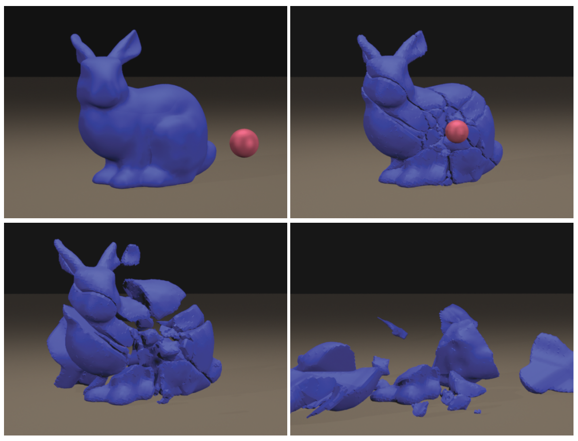

Figure 6: *A complex bunny model with a level set resolution of 70x60x45 is shattered into 35 pieces.*

tical for real-time applications. The goal of our method is to combine the speed of a prescoring technique with the point-of-impact-centered realism of stress-computing methods. We achieve this by *prescoring all of space* generating a fracture pattern that emanates from a fracture center which can then be aligned on-the-fly with the collision location on the object we wish to fracture. This approach provides artists with the ability to more precisely control how an object breaks around a collision point, while still being feasible for real-time applications.

4.1. Framework

Rigid Body Representation As we will be implementing our fracture algorithm in the context of [GBF03], it is important to note that for their implementation, they represented rigid bodies with a triangulated surface mesh and a volumetric level set. Contact and collisions are both detected and processed by computing level set data from one object at the positions of the surface particles of the other object. We note that these algorithms do not require the surface triangles but rather only the surface particles. Thus, each of our rigid bodies will contain a position, orientation, velocity, angular velocity, mass, inertia tensor, list of surface particles, and volumetric level set function. Any standard fast dual-contouring [JLSW02] or marching cubes method [LC87] can be used to triangulate the level set for rendering.

Fracture Patterns The goal is to fracture all of space, and this is easily accomplished by fracturing any piece of geometry that can subsequently be translated and rescaled to enclose the dynamically simulated object to be fractured. For our examples we simply used the bounding box of the object and fractured this box into a number of regions. A single point in this cube is defined as the point of impact for the fracture, and a coordinate system is attached to that point to represent the collision normal. When an object fractures, this box will be translated and rotated to align with the collision, and scaled to enclose the object, both important for the artist to keep in mind when sculpting the pieces. Each of the fracture regions represents a fragment of the fractured space and stores a list of surface particles, a volumetric level set function, and the location of the fracture center with respect to the region.

4.2. Fracturing a Rigid Body

Each rigid body in our simulation is assigned a fracture threshold, which represents the magnitude of the collision impulse the body must feel before it will fracture. When a pair of rigid bodies is processed for collision, we compute the impulse that would be applied due to the collision, and if it is above the fracture threshold of the body, we proceed to fracture the body. The result of this fracture will be a set of fragment bodies, each of which is the result of intersecting the original body, which we refer to as the parent body, with one of the regions of the fracture pattern.

Level Set To generate the level set of a fragment, we will merge the level sets of the fracture region and parent body that intersect to form the fragment. The level set of the fragment is computed at each point as the maximum of the fracture region's and parent body's level set values at that point. This algorithm will produce a level set function that represents the correct signed distance value within the fragment body but may underestimate the signed distance outside the fragment body. This does not cause problems, since we still correctly answer inside/outside queries, and such queries are the only usage of level set values *outside* a body by the collision and contact algorithms of [GBF03]. The level set of the intersection could be disconnected, so we flood fill the level set, giving each connected component a unique color and a separate fragment.

Besides this basic algorithm, we propose an aggressive optimization. If one disregards the notion of a collision normal when aligning the prescored fracture pattern and the parent body and allows up to a half-grid cell perturbation of the point of collision when aligning the parent body and the fracture pattern, it is possible to overlay the prescored level set and the parent body's level set in the material space of the parent body. This means that one can avoid all interpolation

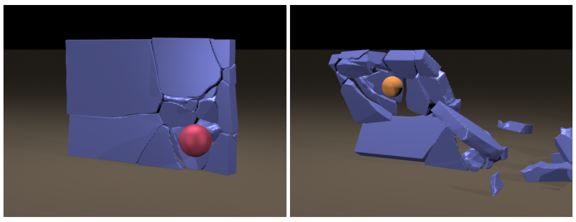

Figure 7: *A wall is fractured in the lower right corner with a sphere, creating large pieces away from the fracture location (left). Then, a second sphere fractures one of the large fragments (right).*

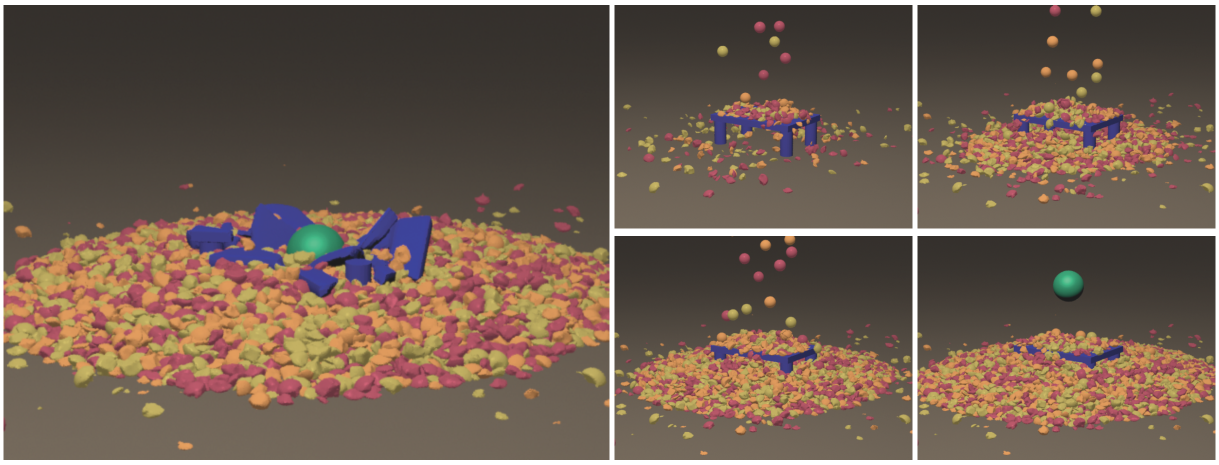

Figure 8: *1000 small spheres are dropped onto a table and then a larger sphere is dropped to fracture the table, resulting in 8992 rigid body fragments. The small spheres have a level set resolution of 55x55x55, and the table top has a resolution of 150x10x100.*

when computing level set values and simply visit each grid point, comparing the two level set values that exist there. In fact we use this optimization in all of our examples.

Surface Particles When constructing a new dynamic fragment of the subsequently fractured parent body, one must find all the particles on the fragment's surface. Those particles come from the surfaces of the fracture region in question and the parent body we are fracturing, and we just need to determine which particles are retained and which particles are discarded. The most straightforward way to test a particle from one object against the other object is to compute its level set value on the other object and see if it inside or outside that object. Particles outside the other object are discarded, and those inside the other object are will help form the new surface and are retained. After doing this for each object against the other, one obtains the final set of all surface particles. However if a single fracture region results in two or more separate disconnected pieces when intersected with the parent body, then we have not properly divided the particles between these pieces. This is easily remedied during the flood fill stage that identifies these connected components as the particles lie closer to the zero contour of one level set than the other, and they are simply assigned to the fragment to which they are closer, i.e. the one that has a smaller level set value at the location.

When using the optimization mentioned above that was used to exactly align a fracture pattern's level set with that of the parent body's in material space, a further optimization can be done for assigning particles to each region. One can simply assign particles to voxels, both for the parent rigid body and for each fracture region. Then when the flood fill process determines which fragment a voxel belongs to, the particles contained in that voxel are simply assigned to it.

Inertial Properties Once we have the level set and particles for a fragment, we need to compute the inertial properties of the fragment. Since we do not store a surface mesh for the fragments, we instead observe that we can compute the desired inertial properties from the level set. The volume of the parent V_p can be computed from its level set region $\mathbb{P}$ as $V_p = \int_{\mathbb{P}} d\mathbf{x}$. The density of the parent is given by $\rho_p = m_p/V_p$. The density of a subsequent fragment ρ_f is set equal to that of the parent. Then, we compute a few more quantities from the fragment's level set region $\mathbb{F}$,

$$V_f = \int_{\mathbb{F}} d\mathbf{x}, \quad \mathbf{C}_f = V_f^{-1} \int_{\mathbb{F}} \mathbf{x}\, d\mathbf{x}, \quad \Sigma_f = \int_{\mathbb{F}} \mathbf{x}\mathbf{x}^T\, d\mathbf{x},$$

where V_f and $\mathbf{C}_f$ are the volume and center of mass of the fragment. Finally, the mass is $m_f = \rho_f V_f$, and the inertia tensor is $\mathbf{I}_f = \rho_f(\mathrm{tr}(\Sigma_f)\delta - \Sigma_f)$. We can simplify these computations by treating a level set cell as entirely inside or outside the fragment based on the sign of its center.

Position and Orientation We choose the position of the fragment to be its center of mass, and we diagonalize the inertia tensor to obtain the orientation of the fragment. We are careful to transform the surface particles of the fragment into the new coordinate system, as well as storing this same transform along with the level set.

Velocity and Angular Velocity We compute velocity and angular velocity to conserve both momentum and angular momentum, as in [MMDJ01]. That is, $\omega_f = \omega_p$ and $\mathbf{v}_f = \mathbf{v}_p + \omega_p \times \mathbf{r}_f$, where $\mathbf{r}_f$ points from the center of mass of the parent to the center of mass of the fragment.

4.3. Generating Fracture Patterns

We do not propose any particular method for fracture pattern generation, though we explore a few possibilities for doing so. The most straightforward is for an artist to design the

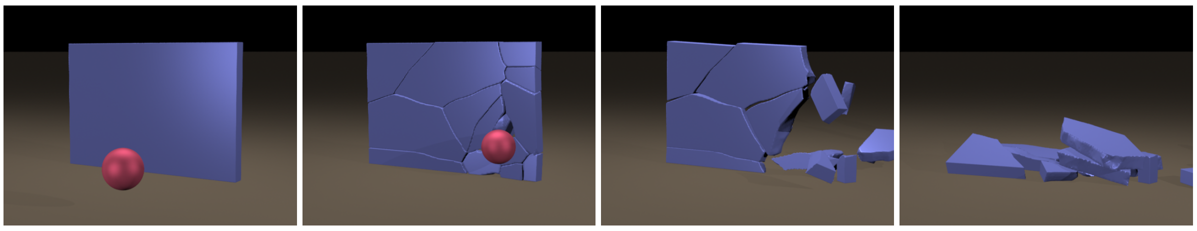

Figure 9: *A wall is fractured using a realistic fracture pattern.*

fracture regions in the form of surface meshes for those regions using existing tools in much the same way that they prescore models today; the main difference is that the artist would prescore a large cube rather than prescoring individual models. Alternatively, these surfaces may be computed using offline finite element analysis in which some portion of space is fractured. From the surface meshes, the level sets can be calculated using a standard algorithm such as the fast marching method. The surface particles might be the vertices of the surface mesh, vertices of a refined surface mesh, or even simply particles sampled on the mesh depending on the quality and refinement of the sculpted mesh.

4.4. Alternatives to Level Sets

We chose to use level sets for our fracturing implementation because we need level sets for our collision algorithms already, and implementing the proposed fracturing algorithm is particularly straightforward to do using level sets. It should be noted, however, that level sets have a number of limitations. Level sets tend to be memory intensive, and it is difficult to properly resolve thin geometry or shards at a reasonably coarse resolution. Due to the level set based collisions, the fractured pieces do not fit snugly together after fracture, but since fracturing tends to be energetic, we have not found this to be a problem. The same high level fracturing algorithm can be performed using only surface meshes by performing Boolean operations of the meshes against the fracture patterns. We note that while such Boolean operations are complicated to implement robustly, implementations are available. In this case, the inertial properties of the fragments should be computed from their surface meshes.

5. Examples

We ran all of our simulations on a single 3.00GHz Xeon core using only the algorithmic optimizations proposed in this paper, i.e. no aggressive hardcoding, special libraries, or other hardware tricks were used. In Figure 4, we show a sphere fracturing a wall at different locations, all using a single radial fracture pattern. We then lower the fracture threshold of the ball and increase the threshold of the wall, allowing the ball to shatter while the wall does not. These examples show that our method can use a single fracture pattern to generate point-of-impact-centered fracture of any object in the scene. Figure 9 shows the same scenario, but with a more realistic fracture pattern. All of these simulations ran at 15 frames/second or faster.

In Figure 5, we show that we can handle the fracture of other objects, simulating a sphere fracturing a tall cylinder at 22 frames/second. Figure 6 illustrates that we can handle more complicated geometry, simulating a sphere fracturing a bunny into 35 fragments at 4 frames/second. Figure 7 illustrates that the fragments produced by our fracture algorithm are just normal rigid bodies, and can themselves be fractured.

Finally, Figure 8 shows that our algorithms scale to a large number of rigid bodies, shattering 1000 spheres on a table top and finally shattering the table itself. After the table top fractures, there are 8992 rigid bodies in the scene. We also ran the same example with 25 and 100 spheres. For the 25 sphere drop, the early frames averaged 25 frames/second while the later frames averaged 3 frames/second. For the 100 sphere drop, the early frames averaged 25 frames/second while the later frames averaged .26 frames/second. For the 1000 sphere drop, the early frames averaged 25 frames/second while the later frames averaged 5 minutes/frame with 8992 fragment bodies in contact on the ground.

6. Conclusion

We present a method for conserving both kinetic energy and angular momentum during rigid body evolution as well as a method for clamping kinetic energy during contact and collision processing. These methods allow for stable rigid body simulations at the frame rate. Moreover, we introduce a method for *prescoring all of space*, allowing realistic point-of-impact-centered fracture of rigid bodies that runs efficiently on modern machines.

Though our methods for mitigating energy gain allow us to achieve stable simulations while taking time steps at the frame rate, there may still be situations in which energy can increase, as we do not clamp kinetic energy during the contact phase of the position update. Although we did not notice any visual artifacts, we believe that this should be addressed as future work.

Acknowledgements

Research supported in part by a Packard Foundation Fellowship, an Okawa Foundation Research Grant, ONR N0014-06-1-0393, ONR N00014-05-1-0479 for a computing cluster, NIH U54-GM072970, and NSF CCF-0541148. J.S. was supported in part by an NSF Graduate Research Fellowship and C.S. was supported in part by a Stanford Graduate Fellowship. We would like to thank Bill Dally and Christos Kozyrakis as well as Charbel Farhat for computing resources, and Jacob Leverich for helping us use those resources.

References

[AP98] ARMERO F., PETOCZ E.: Formulation and analysis of conserving algorithms for frictionless dynamic contact/impact problems. *Computer methods in applied mechanics and engineering 158*, 3 (1998), 269–300.

[Bar89] BARAFF D.: Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Comput. Graph. (Proc. SIGGRAPH 89) 23*, 3 (1989), 223–232.

[Bar90] BARAFF D.: Curved surfaces and coherence for non-penetrating rigid body simulation. *Comput. Graph. (Proc. SIGGRAPH 90) 24*, 4 (1990), 19–28.

[Bar91] BARAFF D.: Coping with friction for non-penetrating rigid body simulation. *Comput. Graph. (Proc. SIGGRAPH 91) 25*, 4 (1991), 31–40.

[Bar94] BARAFF D.: Fast contact force computation for nonpenetrating rigid bodies. In *Proc. SIGGRAPH 94* (1994), pp. 23–34.

[BHTF07] BAO Z., HONG J., TERAN J., FEDKIW R.: Fracturing rigid materials. *IEEE Trans. Viz. Comput. Graph. 13* (2007), 370–378.

[BHW96] BARZEL R., HUGHES J., WOOD D.: Plausible motion simulation for computer graphics animation. In *Comput. Anim. and Sim. '96* (1996), Proc. Eurographics Wrkshp., pp. 183–197.

[BRM09] BOU-RABEE N., MARSDEN J.: Hamilton-pontryagin integrators on lie groups part i: Introduction and structure-preserving properties. *Foundations of Computational Mathematics 9*, 2 (2009), 1615–3375.

[CR98] CHATTERJEE A., RUINA A.: A new algebraic rigid body collision law based on impulse space considerations. *ASME J. Appl. Mech. 65*, 4 (1998), 939–951.

[GBF03] GUENDELMAN E., BRIDSON R., FEDKIW R.: Non-convex rigid bodies with stacking. *ACM Trans. Graph. (SIGGRAPH Proc.) 22*, 3 (2003), 871–878.

[Hah88] HAHN J.: Realistic animation of rigid bodies. *Comput. Graph. (Proc. SIGGRAPH 88) 22*, 4 (1988), 299–308.

[IO06] IBEN H. N., O'BRIEN J. F.: Generating surface crack patterns. 177–185.

[JLSW02] JU T., LOSASSO F., SCHAEFER S., WARREN J.: Dual contouring of Hermite data. *ACM Trans. Graph. (SIGGRAPH Proc.) 21*, 3 (2002), 339–346.

[KCD09] KOBILAROV M., CRANE K., DESBRUN M.: Lie group integrators for animation and control of vehicles. *ACM Transactions on Graphics 28*, 2 (Apr. 2009).

[KEP05] KAUFMAN D., EDMUNDS T., PAI D.: Fast frictional dynamics for rigid bodies. *ACM Trans. Graph. (SIGGRAPH Proc.) 24*, 3 (2005), 946–956.

[KSJP08] KAUFMAN D., SUEDA S., JAMES D., PAI D.: Staggered projections for frictional contact in multibody systems. *ACM Transactions on Graphics (SIGGRAPH Asia 2008) 27*, 5 (2008), 164:1–164:11.

[KYT*06] KHAREVYCH L., YANG W., TONG Y., KANSO E., MARSDEN J., SCHRÖDER P., DESBRUN M.: Geometric variational integrators for computer animation. *ACM SIGGRAPH/Eurographics Symp. on Comput. Anim.* (2006), 43–51.

[LC87] LORENSEN W., CLINE H.: Marching cubes: A high-resolution 3D surface construction algorithm. *Comput. Graph. (SIGGRAPH Proc.) 21* (1987), 163–169.

[MBF04] MOLINO N., BAO Z., FEDKIW R.: A virtual node algorithm for changing mesh topology during simulation. *ACM Trans. Graph. (SIGGRAPH Proc.) 23* (2004), 385–392.

[MMDJ01] MÜLLER M., MCMILLAN L., DORSEY J., JAGNOW R.: Real-time simulation of deformation and fracture of stiff materials. In *Comput. Anim. and Sim. '01* (2001), Proc. Eurographics Wrkshp., Eurographics Assoc., pp. 99–111.

[MW88] MOORE M., WILHELMS J.: Collision detection and response for computer animation. *Comput. Graph. (Proc. SIGGRAPH 88) 22*, 4 (1988), 289–298.

[NTB*91] NORTON A., TURK G., BACON B., GERTH J., SWEENEY P.: Animation of fracture by physical modeling. *Vis. Comput. 7*, 4 (1991), 210–219.

[OBH02] O'BRIEN J., BARGTEIL A., HODGINS J.: Graphical modeling of ductile fracture. *ACM Trans. Graph. (SIGGRAPH Proc.) 21* (2002), 291–294.

[OH99] O'BRIEN J., HODGINS J.: Graphical modeling and animation of brittle fracture. In *Proc. of SIGGRAPH 1999* (1999), pp. 137–146.

[SSF08] SHINAR T., SCHROEDER C., FEDKIW R.: Two-way coupling of rigid and deformable bodies. In *SCA '08: Proceedings of the 2008 ACM SIGGRAPH/Eurographics symposium on Computer animation* (2008), pp. 95–103.

[ST00] STEWART D. E., TRINKLE J. C.: An implicit time-stepping scheme for rigid body dynamics with Coulomb friction. In *IEEE Int. Conf. on Robotics and Automation* (2000), pp. 162–169.

[SW91] SIMO J., WONG K.: Unconditionally stable algorithms for rigid body dynamics that exactly preserve energy and momentum. *International Journal for Numerical Methods in Engineering 31*, 1 (1991), 19–52.

[TGPS08] THOMASZEWSKI B., GUMANN A., PABST S., STRASSER W.: Magnets in motion. *ACM Trans. Graphics (Proc. SIGGRAPH Asia) 27*, 5 (2008), 162:1–162:9.

[Vil08] VILMART G.: Reducing round-off errors in rigid body dynamics. *J. Comput. Phys. 227* (2008), 7083–7088.

[vZS07] VAN ZON R., SCHOFIELD J.: Numerical implementation of the exact dynamics of free rigid bodies. *J. Comput. Phys. 225* (2007), 145–164.

[YOH00] YNGVE G., O'BRIEN J., HODGINS J.: Animating explosions. In *Proc. SIGGRAPH 2000* (2000), vol. 19, pp. 29–36.