Interpenetration Free Simulation of Thin Shell Rigid Bodies

R. Elliot English, Michael Lentine, and Ron Fedkiw

Abstract—We propose a new algorithm for rigid body simulation that guarantees each body is in an interpenetration free state, both increasing the accuracy and robustness of the simulation as well as alleviating the need for ad hoc methods to separate bodies for subsequent simulation and rendering. We cleanly separate collision and contact resolution such that objects move and collide in the first step, with resting contact handled in the second step. The first step of our algorithm guarantees that each time step produces geometry that does not intersect or overlap by using an approximation to the continuous collision detection (and response) problem and, thus, is amenable to thin shells and degenerately flat objects moving at high speeds. In addition, we introduce a novel fail-safe that allows us to resolve all interpenetration without iterating to convergence. Since the first step guarantees a noninterfering state for the geometry, in the second step we propose a contact model for handling thin shells in proximity considering only the instantaneous locations at the ends of the time step.

Index Terms—Computer graphics, rigid bodies, thin shells

1 Introduction

Rigid body simulation has been popular in computer graphics for over two decades, dating back to [1], [2], [3], [4], [5]. Later, authors focused on efficiency [6], [7] and explored the concept of plausible motion [8], [9]. These authors asserted that collisions do not necessarily have to be handled in consecutive order to achieve plausible results. More recently authors have focused on a variety of topics such as magnetism [10], two-way coupling with deformable objects [11], sampling rigid body behaviors [12], energy conservation [13], and synthesizing sounds from fracturing bodies [14]. There are also a number of commercially available and open source software packages such as Bullet, ODE, Havok, PhysX, and so on. Although these techniques have proven to work well for many applications, they do not guarantee an interpenetration free state, and thus cannot handle complex arbitrarily thin geometry (e.g., thin shells—see Fig. 1). To achieve this goal, we have developed robust algorithms for both collision detection and response.

Rigid body simulation involving volumetric bodies has been extensively studied in the literature. Most prior methods use interference detection to compute collisions and contacts, such as [15], [16], [17], [18], [19], [20]. While these methods are reasonably efficient for relatively thick bodies, when the thickness approaches zero, collisions and contacts can be easily missed as bodies can move large distances through one another in a single time step. Many methods, such as [20], require a convex decomposition to compute distances and normals. However, thin shells cannot be broken in convex components other than into groups of exactly coplanar faces. Furthermore, when computing a response to the detected collisions and contacts, it becomes unclear as to which direction the bodies should be moving once they have partially passed through one another. To avoid this scenario, a time-step restriction can be introduced, which prevents bodies from moving more than half the thickness in a single time step. This leads to a tradeoff between accuracy and efficiency, where highly thickened bodies can be simulated using fewer time steps but incur a large error, while only slightly thickened bodies require many time steps increasing computational cost.

Redon et al. [21] introduced continuous collision detection for rigid bodies to resolve the issues with interference testing. However, their response algorithm [22] neither handles friction nor guarantees an interpenetration free state. Otaduy et al. [23] presented an alternative continuous collision response algorithm that implicitly computes contact forces in conjunction with deformable body constitutive forces. This method uses a mixed linear complementarity problem (MLCP) that can be solved efficiently for deformable bodies due to the local nature of the resulting forces. However, for rigid bodies, contact forces are generally nonlocal, and thus, this method becomes infeasible for even moderately sized stacks of bodies. In fact, using a Gauss-Seidel approach as in [23] and other methods (see, e.g., [15]) requires an intractable number of iterations to converge when the constraints need to be exactly satisfied as is required to prevent interpenetration. The authors of [24] have also addressed the continuous collision response problem, choosing to apply penalty forces; however, they do not guarantee an interpenetration free state. As a result, it remains an open problem to find an algorithm for computing a set of impulses that both terminates in a fixed number of iterations and plausibly resolves all collisions within a single time step for large scale stacking problems.

- *R.E. English is with the Computer Science Department, Stanford University, Gates Computer Science Building, 353 Serra Mall, Room 206, Stanford, CA 94305-9025. E-mail: eenglish@gmail.com.*
- *M. Lentine is with Lucas Arts, San Francisco, CA and the Computer Science Department, Stanford University, Gates Computer Science Building, 353 Serra Mall, Room 206, Stanford, CA 94305-9025. E-mail: mlentine@stanford.edu.*
- *R. Fedkiw is with Industrial Light + Magic, San Francisco, CA and the Computer Science Department, Stanford University, Gates Computer Science Building, 353 Serra Mall, Room 207, Stanford, CA 94305-9025. E-mail: fedkiw@cs.stanford.edu.*

Manuscript received 16 Oct. 2011; revised 29 Jan. 2012; accepted 26 Aug. 2012; published online 4 Sept. 2012.
Recommended for acceptance by J. Keyser.
For information on obtaining reprints of this article, please send e-mail to: tvcg@computer.org, and reference IEEECS Log Number TVCG-2011-10-0254.
Digital Object Identifier no. 10.1109/TVCG.2012.179.

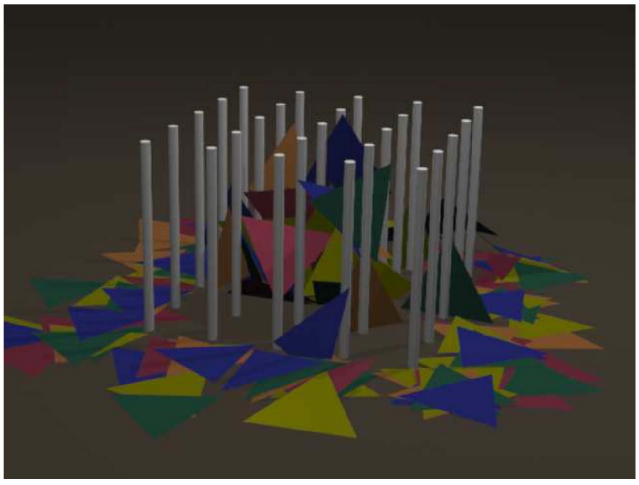


Fig. 1. Degeneratly thin triangles are dropped on several fixed pegs and allowed to stack both on the ground and lying against the pegs.

The authors from [25] considered similar problems, but because they let their objects deform, they were unable to guarantee a final interpenetration free state for the undeformed geometry. This leads to issues with collision response, contact modeling, and prolonged unresolved interpenetrations. Allowing the geometry to deform requires either the rendering of the deformed copies of the geometry or accepting severe visual artifacts due to potentially large interpenetrations—note that this is exacerbated by large velocities and rotational motion. Furthermore, subsequent simulation of the rigid bodies is difficult when large deformations occur, as it is unclear that the retargeting of the rigid body velocities will lead to plausible motion.

To handle thin shells efficiently, our method uses a linearized continuous collision detection algorithm based upon the continuous collisions detection algorithms typically used for cloth and deformable simulation [26], [27], [28]. To handle these collisions and contacts, we follow [15] and cleanly separate the processing into two steps. We process collisions by applying a sequential impulse approach derived from [15]. To process contact, we apply two separate phases to handle static and dynamic contact. In static contact, we consider the bodies in an instantaneous configuration and generate contact points by finding nearby feature pairs on objects in this configuration. We then solve for contact forces at these contacts using a modified version of the projected Gauss Seidel method and shock propagation schemes introduced in [29]. In dynamic contact, we consider the motion of bodies over the time step by modifying and reusing the procedure from our collision processing routine to apply inelastic contact forces instead of elastic contact forces. However, unlike collision processing, it is critical to resolve all contacts to prevent interpenetration. This unfortunately leads to an iterative problem that can take a prohibitive number of iterations to converge. We note that our use of dynamic and static when differentiating these two types of contact refers to whether we detect contacts using an instantaneous configuration of bodies or by using their swept geometry. In both steps, we apply static and dynamic friction.

To prevent our algorithm from iterating indefinitely, we limit the number of collision and dynamic contact iterations and then apply a novel fail-safe that clusters together bodies to resolve any remaining interpenetrations. In contrast to [27], we cluster bodies using a kinematic rigidification that preserves their relative motion. While they do not implement one, [23] ultimately says a fail-safe is necessary to eliminate all collisions in complex enough cases. Harmon et al. [30] also propose to resolve all collisions by implicitly formulating a linear system by sequentially adding each remaining collision as an equality constraint until no further collisions occur. However, for rigid bodies these rigidifying fail-safes tend to fully rigidify large systems of bodies, particularly in the case of stacks of bodies, producing nonphysically plausible results. Furthermore, in the case of [30] it would also take a significant amount of time to compute the necessary SVD due to the system's singularity.

Finally, after combining our method with [13], we no longer have a time-step restriction to achieve stable and physically plausible simulation of rigid bodies. Note that while this can be achieved without using our method by adaptively thickening bodies as discussed above, our method is necessary to guarantee an interpenetration free state when bodies have a fixed or zero thickness.

2 Time Evolution

The equations governing the motion of a rigid body are as follows:

$$\frac{d\mathbf{x}}{dt} = \mathbf{v}, \quad \frac{d\mathbf{q}}{dt} = \omega^* \mathbf{q} \tag{1}$$

$$\frac{d\mathbf{v}}{dt} = m^{-1}\mathbf{f}, \quad \frac{d(\mathbf{M}_a\omega)}{dt} = \tau, \tag{2}$$

where $\mathbf{x}$ and $\mathbf{q}$ are the position and rotation (to simplify the exposition, we use a rotation matrix instead of a unit quaternion), $\mathbf{v}$ and ω are the linear and angular velocity, $\mathbf{f}$ and τ are the linear force and torque, and m and $\mathbf{M}_a$ are the mass and world space rotational inertia of the rigid body. For convenience, we later refer to generalized positions, velocities, and masses of bodies using $\mathbf{X}$ (a rigid body frame as a transformation matrix), $\mathbf{V}$ (the linear and angular velocity vectors concatenated), and $\mathbf{M}$ (the linear and angular inertia blocks as a block diagonal matrix) with the appropriate subscripts.

To numerically integrate the position, we use the explicit second order accurate approximation from [11], which we give below for completeness

$$\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v} \tag{3}$$

$$\mathbf{q}^{n+1} = \mathbf{R}(\Delta t\omega + 0.5\Delta t^2 \mathbf{M}_a^{-1}(\mathbf{M}_a\omega)^*\omega)\mathbf{q}^n, \tag{4}$$

where $\mathbf{R}(\cdot)$ returns a rotation matrix given an angle scaled axis vector. This position integration scheme is applied multiple times throughout our overall integration scheme, e.g., when handling unconstrained motion and when reevolving bodies during collisions and dynamic contact. Note that this second order accurate scheme is used to increase the accuracy at a small cost. However, the scheme could also easily be replaced by a first order accurate method.

In this section, we present the time integration details of our rigid body solver. We refer the interested reader to related works [11], [15], [31], [32] for other approaches. The basic time integration algorithm we use proceeds using a variant of [15], which first handles collisions in Step 1.

Steps 2-5 compute temporary velocities that are processed by a first contact step, which is subsequently used to produce interpenetration free updated positions and is then thrown out. Steps 6 and 7 integrate explicit and body forces, and then apply a second contact step to produce the final velocity.

The velocity from Steps 2-5 is discarded and this proceeds as follows:

1. Modify $\boldsymbol{v}^n$ to $\hat{\boldsymbol{v}}^n$ with collisions.
2. $\boldsymbol{v}^{n+1/2} = \hat{\boldsymbol{v}}^n + \frac{\Delta t}{2}\boldsymbol{a}(t^{n+1/2}, \boldsymbol{x}^n, \boldsymbol{v}^{n+1/2})$.
3. Modify $\boldsymbol{v}^{n+1/2}$ with static contact.
4. $\hat{\boldsymbol{x}}^{n+1} = \boldsymbol{x}^n + \Delta t \boldsymbol{v}^{n+1/2}$.
5. Modify $\hat{\boldsymbol{x}}^{n+1}$ to $\boldsymbol{x}^{n+1}$ with dynamic contact and fail-safe.
6. $\boldsymbol{v}^{n+1} = \hat{\boldsymbol{v}}^n + \Delta t \boldsymbol{a}(t^{n+1/2}, \boldsymbol{x}^{n+1}, \boldsymbol{v}^{n+1/2})$.
7. Modify $\boldsymbol{v}^{n+1}$ with static contact,

where $a(\cdot)$ returns the acceleration due to body forces and explicit forces, such as gravity.

In Step 1, the $\hat{\boldsymbol{v}}^n$ velocities are initially set to be equal to $\boldsymbol{v}^n$ and are iteratively updated using our collision processing algorithm (see Section 3) by finding and handling collisions considering the motion of bodies between their time n and temporary time $n+1$ positions that are found by explicitly integrating the time n positions using the most recent update of $\hat{\boldsymbol{v}}^n$. Step 2 adds explicit forces to find temporary time $n+1/2$ velocities. Step 3 applies static contact (see Section 4.3) to the temporary time $n+1/2$ velocities using contacts found with the bodies in their time n positions. This static contact step is important to the efficiency of our algorithm because it lifts the computational burden from the following dynamic contact processing step by preventing the majority of interpenetration. Step 4 integrates the positions using the time $n+1/2$ velocities. Step 5 applies dynamic contact (see Section 4.1) and the fail-safe (see Section 4.2) to the positions from Step 4 to produce the final time $n+1$ positions. Step 6 integrates the post collision velocities by adding explicit and body forces to produce candidate time $n+1$ velocities. Step 7 applies static contact to the candidate time $n+1$ velocities using contacts found with the bodies in their time $n+1$ positions (and only between pairs of bodies processed during collision response in Step 1 to prevent collisions being missed) to produce the final time $n+1$ velocities.

Note that typically, collision and contact detection would be performed once per time step. However, to guarantee an interpenetration free state, we must recompute the pairs of colliding bodies and their collision and contact points after the positions have been updated to reflect the impulses applied due to previously resolved collisions and contacts. This is because it is possible that using a collision unaware position integration scheme to update the positions can lead to a new configuration with interpenetrations. It is important to emphasize that processing additional pairs of interacting bodies in dynamic contact and the first static contact step does not lead to missed elastic collisions.

3 Collisions

There have been a number of papers on the plausible simulation of rigid bodies [8], [33], [34], [35]. Similar to these methods, we do not try to treat every collision in chronological order to obtain the exact analytic solution. Instead, we work toward a plausible solution that shifts momentum among objects by applying conservative elastic and partially elastic impulses. To handle collisions, we use an interwoven detection and response scheme similar to that of [15] with modifications to handle thin shells using continuous collision detection.

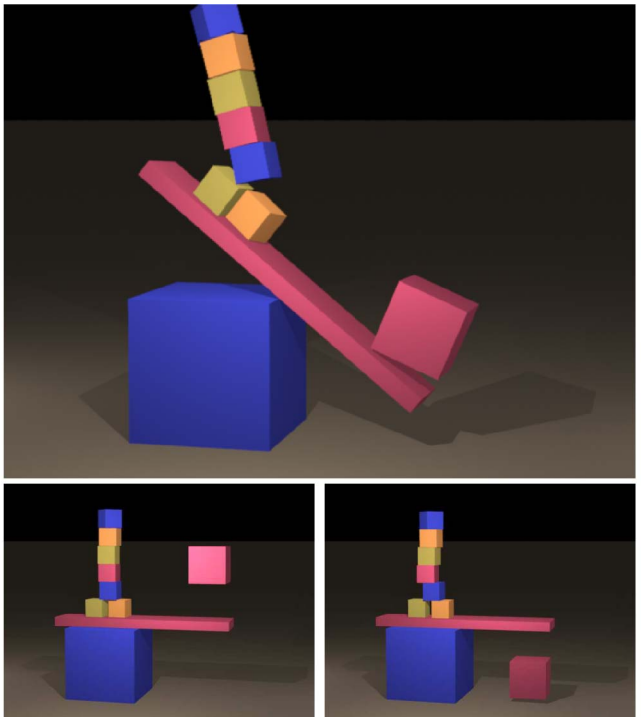


Fig. 2. A stable stack of blocks are launched by a catapult hit by a fast moving block. (Bottom right) [15] allows the block to pass through the catapult. (Top) Our algorithm catches the collision.

To find pairs of interacting bodies, we use a Cartesian grid-based spatial partition. To initialize the spatial partition, we rasterize the swept bounding box of each rigid body onto the background grid. We subsequently compile a list of interacting body pairs by querying the background grid for the swept bounding box of each body. As bodies are reevolved during collision handling and dynamic contact, we update the background grid with their updated swept bounding boxes. After a body has been moved, newly colliding pairs that need processing are found by requerying the background grid.

Many volumetric rigid body solvers, such as [15], use an interference detection approach. The method of [15] used a discrete level set representation of the rigid body for efficient depth queries. Between two potentially colliding bodies, the points (vertices) on the surface of one body would be queried against the level set of the other body to determine if the two bodies were colliding and then for the deepest point, a response impulse would be applied and then the bodies are reevolved. While this approach does not prevent bodies from passing through each other, one could thicken the bodies by a collision threshold distance, and then place a time-step restriction that requires that each point on every body move at most this distance in a single time step. As a result, every collision would be detected regardless of the speed of objects, and if one were careful when applying collision responses to ensure that the bodies at their time $n+1$ positions were separated by a minimum collision threshold distance, then an interpenetration free algorithm could be produced. Unfortunately, while this method can handle thin and exactly zero thickness bodies, the method can become impractical by

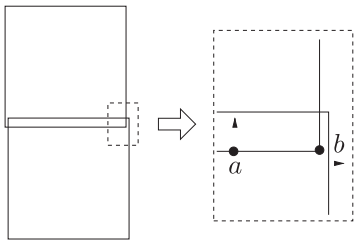


Fig. 3. Two contact points and their normals found by using level set depth queries for two boxes in resting contact on top of one another. The contact normal for point a is correct because the nearest face on the lower box is on the top side. The contact normal for point b is incorrect because the nearest face on the lower box is the right side. This contact point and normal combination would erroneously prevent the boxes from sliding against one another.

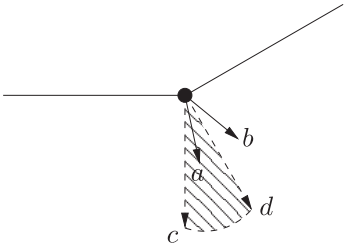


Fig. 4. For a vertex, where the area above the line is the interior of the body, the admissible set of collision and contact normals is defined by the conical combination of the incident face normals. In this case, vectors c and d are the face normals with the shaded area being the admissible set. Vector a is an admissible normal, while vector b is not.

requiring very small time steps for fast moving bodies, or by overly thickening level sets (see Fig. 2 for a simple example when an extremely small time step is required to catch a high-speed collision).

Another major issue when using the type of interference detection in [15] is that it is difficult to produce accurate collision/contact normals. For example, when a penetrating point is near two or more faces, the normal could be taken from any of the faces and may not be representative of the desired contact behavior (see Fig. 3). One solution to this issue is to remove contact points with normals that are not in the admissible region as defined by the conical combination of the normals of the faces meeting at the feature (e.g., a single vector for a plane, a wedge for an edge, and generally an n-sided pyramid for a vertex). Fig. 4 demonstrates this characterization. Note that in tests, we also encountered this issue when using a discretized level set despite the fact that it creates a smoothed representation of the original geometry. We also note that this issue is often avoided in [15] due to increased point sampling, epsilon scaling, and only applying impulses to the deepest sample point in their iterative scheme. However, if one wished to construct a set of contact points without iterating to construct a monolithic system, such as a linear complementarity problem (LCP) or nonlinear complementarity problem (NCP), to improve the convergence and stability of the simulation, these normals can produce highly inaccurate results. Instead, we use continuous collision detection to find collisions between thin shells as detailed below. We can also eliminate the time-step restriction on volumetric bodies by treating them as thin shells.

3.1 Continuous Collision Detection

We base our approach on the linearly swept vertex/triangle and edge/edge (generally referred to as feature pairs in our exposition) method of [27]. In our case, we linearize the problem by assuming each vertex moves linearly at a constant velocity between the global position of the vertex at times n and $n+1$. We similarly extend this to edges and triangles by linearizing the motion of their corner vertices. This implies that each vertex on the edge and/or triangle also moves along a constant velocity linear trajectory (See Fig. 5). Using this linearization we find the times of coplanarity between the feature pairs in the current time step. For each time of coplanarity, we check whether the simplices are actually coincident (using a small nonzero tolerance on the order of 10^{-4} for robustness) and, thus, actually collide at that time, to find the earliest colliding time for the pair.

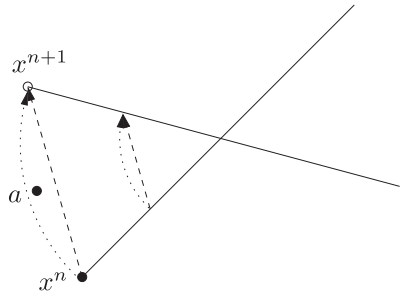


Fig. 5. An edge rotates in 2D, colliding with a particle a according to the rigid trajectories (dotted lines), and missing it according to the linearized trajectories (dashed lines).

To efficiently find candidate feature pairs between a pair of bodies, we intersect the local bounding sphere hierarchies of each body using swept bounding sphere intersection checks. We then find the earliest collision time among all candidate feature pairs.

If a collision has not occurred between the swept geometry, we next check whether the time $n+1$ geometry of the pair is in close proximity using the scheme from Section 4.3.1. If the pair is found to be within the contact rest distance and with an approaching point velocity at one of the contact points, we treat the body pair as colliding and process the nearest pair of features as the collision feature pair.

Another approach to continuous collision detection for rigid bodies is given by Redon et al. [21]. They approximate the motion of the geometry during the time step as a constant screwing motion by fixing the angular velocity. While this scheme could be used instead of our collision detection algorithm, we chose to apply a cloth-based continuous collision detection algorithm for its simplicity, robustness, efficiency, and to exploit existing cloth CCD frameworks. While there are also a number of other collision detection (see, e.g., [36], [37], [38]) methods that could be used for thin shells, they generally suffer from issues of robustness, time-step restrictions, and computational cost. We note that by linearizing the motion of the rigid body geometry, the geometry can distort and even become completely colinear during the time step, which could result in missed collisions. One solution to this problem could be to subdivide the time step and perform our linearized continuous collision detection over each interval to find the earliest time of impact over the original time step. Note that this would not change the simulation time step, but simply improve the approximation of collision detection. However, during our tests we never

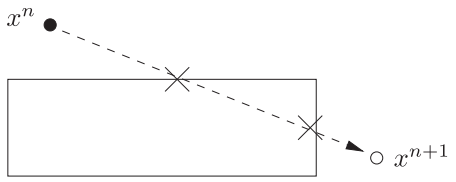


Fig. 6. If the second collision was handled first, the particle would appear to ricochet off the box to the left rather than colliding with the top of the box and ricocheting upwards.

encountered any problems due to this linearization. Even in cases when issues might arise due to this limitation, the method still provides a "bullet-proof" collision handling scheme that prevents objects from interpenetrating along linear trajectories.

3.2 Collision Resolution

Similar to [15], our collision resolution algorithm is integrated with our collision detection algorithm by iteratively handling collisions immediately after they are found. Our algorithm proceeds by iterating over pairs of potentially colliding bodies. For each pair, we attempt to resolve all collisions between the two bodies by iteratively finding the earliest collision, computing and applying a response impulse, and then reevolving the bodies, stopping after a fixed number of iterations. To compute the response impulse, we use the same method as described in [15] with slight modifications to use the information from the linearized collision. In addition, we clamp the resulting impulse to prevent energy increases using the method described in [13]. For an outline of our collision handling scheme see Algorithm 1.

Algorithm 1. Collision Handling
for $i = 1 \rightarrow maxIterations$ **do**
  for all $b \in allBodies$ **do**
    $candidates \leftarrow$ all bodies potentially colliding with or in proximity to body b
    for all $c \in candidates$ **do**
      update bodies b and c to resolve elastic or partially collisions and proximities between bodies b and c using the impulse computation from § 3.2
    end for
  end for
end for

Noteably, it is important to use the normal and position from the linearized geometry at the collision time, as well as the linear velocities of the vertices of the involved simplices. It is necessary to use these so-called effective velocities to find the relative pointwise velocity to ensure that the relative collision velocity is approaching (not separating) so that a valid response impulse is computed. We found that in certain cases, due to the linearization, the exact pointwise rigid velocities could be tangential, even separating, when a collision was occurring between the geometry. To prevent spurious rotations when objects are moving quickly, we use the moment arms from the time of impact.

Recall in Section 3.1, the case where the swept geometry does not collide but rather it is in close proximity at the end of the time step. In this case, we use the normal and location as computed in Section 4.3.1, and the relative pointwise velocity directly computed from both the rigid velocities and the moment arms at the end of the step.

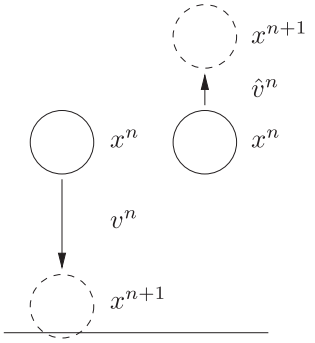


Fig. 7. (Left) A ball is in collision with the ground before any response impulse is applied. (Right) After applying an elastic collision impulse and reevolving the ball, it is even further off the ground. (Right) After applying an inelastic contact impulse and reevoling the ball, it is exactly at the rest distance.

While we do not globally handle the earliest collision first, we do handle all the collisions between a given *pair* of bodies in consecutive order to prevent several common cases that can occur with an arbitrarily small time step. In these cases, the collision ordering has a significant effect on the solution. For example, see Fig. 6.

Note that our collision handling scheme can prevent bodies from ever touching due to the application of the response impulse to the velocities and subsequent reevolution from the time n positions (e.g., see Fig. 7). This error largely depends upon the phase of the collision within the time step, and vanishes as the time step size goes to 0. One way to reduce the visibility of such temporal aliasing in final simulation and rendering is to use motion blur—although this requires some storage of the intermediate state at the time of collision.

For completeness, we now give the equations to compute the frictional response impulse as given in [15]. The new velocities for object i are $\mathbf{v}_i' = \mathbf{v}_i + \boldsymbol{l}/m_i$ and $\omega_i' = \omega_i + \mathbf{M}_{a,i}^{-1}(\mathbf{r}_i^* \boldsymbol{l})$, where $\mathbf{r}_i^*$ is the cross product matrix of the moment arm, and $\boldsymbol{l}$ is the collision response impulse. The update equations are the same for the second body j with the impulse terms negated. The change in pointwise velocity for object i can be found by multiplying the impulse by $\mathbf{K}_i = \mathbf{I}/m_i + \mathbf{r}_i^{*T}\mathbf{M}_{a,i}^{-1}\mathbf{r}_i^*$. Similarly, multiplying the impulse by $\mathbf{K} = \mathbf{K}_i + \mathbf{K}_j$ gives the change in relative pointwise velocity. Let $\mathbf{v}_{rel}$ be the relative pointwise collision velocity and $\mathbf{v}_{rel,n} = \mathbf{n}\mathbf{n}^T\mathbf{v}_{rel}$ be the relative normal pointwise velocity, where $\mathbf{n}$ is the collision normal. Then, we let $\mathbf{v}_{rel,t} = \mathbf{v}_{rel} - \mathbf{v}_{rel,n}$ be the relative tangential pointwise velocity. We now solve for a set of impulses that give the new relative normal velocity $\mathbf{v}_{rel,n}' = -\epsilon\mathbf{v}_{rel,n}$, where ϵ is the coefficient of restitution.

We handle friction as follows: Assuming static friction, set the new tangential relative velocity to zero, $\mathbf{v}_{rel,t}' = 0$, and the impulse we apply at the point is then found by substituting $\mathbf{v}_{rel}' = \mathbf{v}_{rel} + \mathbf{K}\boldsymbol{l}$ (the relative velocity update equation) into the combined equations for $\mathbf{v}_{rel,n}'$ and $\mathbf{v}_{rel,t}'$. If $\|\boldsymbol{l} - (\boldsymbol{l}^T\mathbf{n})\mathbf{n}\| \leq \mu\boldsymbol{l}^T\mathbf{n}$, where μ is the coefficient of friction,

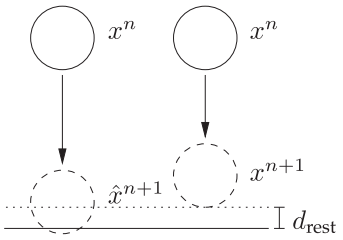


Fig. 8. (Left) A ball is interpenetrating with the ground before any response impulse is applied. (Right) After applying an inelastic contact impulse and reevoling the ball, it is exactly at the rest distance.

then the frictional impulse is correct and the object is sticking. Otherwise, we define the tangential direction as $\mathbf{t} = \mathbf{v}_{rel,t}/\|\mathbf{v}_{rel,t}\|$ and set $\boldsymbol{l} = \mathbf{n}\lambda_n - \mu\mathbf{t}\lambda_n$, where λ_n is the normal impulse. We substitute this equation into the relative velocity update equation that is then substituted into the equation for $\mathbf{v}'_{rel,n}$ and solved to give the final collision impulse.

4 Contact

Once high-speed collisions between bodies have been handled and explicit forces have been integrated into the velocities, we need to generate contact forces to prevent objects in rest from interpenetrating one another. In our time integration scheme, we perform both dynamic and static contact. In dynamic contact, we attempt to find contact forces that resolve all interpenetrations that occur during the time step by considering the motion of the bodies from their time n to time $n+1$ positions. This is done by sequentially finding contacts between each pair of bodies and attempting to resolve all of them by updating both the positions and velocities (only temporarily as they are thrown out before the real velocity is updated) before moving onto the next pair of bodies. In static contact, we only consider the bodies at their time $n+1$ positions and compute forces to alter their time $n+1$ velocities during the contact resolution step.

4.1 Dynamic Contact

To detect and handle contacts between moving objects, we modify the collision detection and handling algorithm from Section 3 to apply relaxed inelastic contact impulses that allow the bodies to come into exact noninterpenetrating contact.

As in [15] we handle each contact by computing a response impulse, applying it to the bodies in contact, and then re-evolving their positions. In contrast to [15], which requires the normal relative velocity to be 0, we allow the bodies to come exactly into resting contact at the end of the time step by modifying the impulse computation procedure from Section 3.2 to solve for the new normal relative velocity $\mathbf{v}'_{rel,n} = (d_{\text{rest}} - d_{\text{current}})/\Delta t$, where d_{rest} is the rest distance and d_{current} is the contact distance at the beginning of the time step. This modification allows bodies to more stably reach contact without stopping prematurely and reaccelerating when contacts are missed in subsequent steps as illustrated in Fig. 8. When processing a pair of bodies, if after a fixed number of iterations the contacts cannot be resolved, we rigidify the pair using the approach from Section 4.2.1. The remainder of the dynamic contact handling algorithm proceeds iteratively in the same manner as the collision handling algorithm and is outlined in Algorithm 2.

Algorithm 2. Dynamic Contact Handling
```
for i = 1 → maxIterations do
  for all b ∈ allBodies do
    candidates ← all bodies (or clusters) potentially
    colliding with or in proximity to body b
    for all c ∈ candidates do
      iteratively update bodies (or clusters) b and c
      using the impulse computation method from
      §3.2 and §4.1, updating any child bodies
      using the appropriate set of Equations 9-10
      §3.2 and §4.1
      if unable to resolve all contacts then
        use §4.2.1 to fully rigifidy the motion of
        bodies b and c
      end if
    end for
  end for
end for
```

4.2 Fail-Safe

While dynamic contact usually resolves any interpenetrations during the time step, it can take a very large number of iterations to converge, if at all. To avoid this, we impose an iteration limit to prevent infinite loops from occurring. If there are outstanding interpenetrations once this limit is reached, we need to resolve them using a more robust scheme that guarantees a noninterpenetrating state in a reasonable amount of time. In cloth simulation, fail-safes, such as in [26], [27], [30], have long been used to guarantee there are no interpenetrations during and at the end of the time step. In the fail-safe used in [27], when two pieces of cloth are interpenetrating after the main collision processing step, they are combined into a single cluster (referred to as impact zone in previous work) and rigidly evolved over the time step. While this is certainly nonphysical, it does preserve momentum and guarantees that the cloth geometry has followed a nonpenetrating trajectory. This process of merging cloth elements continues until either all the cloth elements have been merged into a single cluster, or when no interpenetrations remain. We note that this process is guaranteed to terminate in a fixed number of iterations because at least one pair is merged per fail-safe iteration and the maximum number of pairs that can be merged over the entire procedure is at most the number of bodies minus one. This holds true for our fail-safe as well.

While for cloth, complete rigidification often suffices because of the local effect of applying contact impulses; for rigid bodies, the instant propagation of impulses can cause large numbers of bodies to be rigidified. We propose a modification to this scheme that mitigates the visual artifacts of being completely rigidified. We do this by preserving relative motion between bodies that have been clustered together. See Fig. 9 for a comparison between a fully rigidifying fail-safe and ours.

Our fail-safe proceeds in the same manner as our dynamic contact algorithm. For each body in sequence we

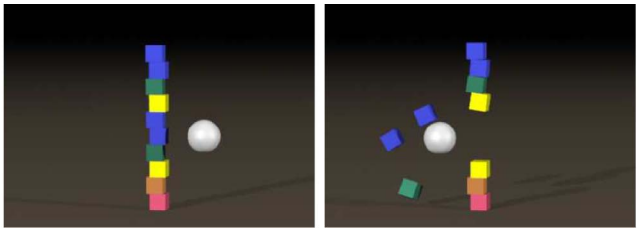


Fig. 9. A stack of blocks is hit by a sphere moving from the right. Both images are after 13 time steps. (Left) Using only rigidification in the fail-safe causes the stack and the ball to have zero velocity. (Right) Using our kinematic rigidification allows the boxes to separate while guaranteeing an interpenetration free state.

compute all potentially colliding pairs of bodies and then sequentially process each pair of bodies attempting to resolve all interpenetrations between a single pair of bodies. After processing a pair of bodies, if all interpenetrations are resolved, then the pair is kinematically rigidified into a single cluster as described in Section 4.2.2. If there are interpenetrations remaining between a pair of bodies, then the pair is fully rigidified as described in Section 4.2.1. We note that when iterating through pair of bodies we discard any pairs that have already been clustered, and also that when processing pairs, where one or both bodies are clusters, we require that all intercluster interpenetrations are resolved to kinematically rigidifying the pair. See Algorithm 3 for an outline of our fail-safe procedure.

Algorithm 3. Failsafe

```
needAnotherIteration ← true
while needAnotherIteration do
  needAnotherIteration ← false
  for all b ∈ allBodies do
    candidates ← all bodies (or clusters) potentially
    colliding with or in proximity to body b
    for all c ∈ candidates do
      iteratively update bodies (or clusters) b and c
      using the impulse computation method from
      §3.2 and §4.1, updating any child bodies using
      the appropriate set of Equations 9-10 or 13-14
      if any contacts found between bodies b and c
      then
        needAnotherIteration ← true
        if able to resolve all contacts then
          use Equations 5-7,11 and 12 to compute
          the kinematic cluster properties for the
          aggregate of the bodies (or clusters) b and c
        else
          use Equations 5-10 to compute the cluster
          properties and fully rigifidy the bodies
          (or clusters) b and c
        end if
      end if
    end for
  end for
end while
```

One problem that arises when applying kinematically rigidification is that bodies can pinched by kinematic clusters such that not all interpenetrations can be resolved. As a result, the pair will be fully rigidified in their time n interpenetration free states and subsequently rigidly evolved. Hence, the work done to preserve any relative motion within the kinematic cluster will be undone. In tests, we found this case occurs in regions of high-speed impact and in the interior regions of stacks, where bodies were not visible. Exterior bodies tended to be remain uncluster or be kinematically rigidified in later iterations, hiding the full rigidification on the interior. We note that this is highly sensitive to the order in which pairs are processed and is an avenue for future investigation.

4.2.1 Rigidification

When the interpenetrations between a group of bodies cannot be resolved within the iteration limit, they must be rigidified to prevent interpenetration. To do this, the bodies are clustered together at the beginning of the time step and rigidly evolved together. We now give the procedure for rigidifying a cluster of rigid bodies.

We first compute the combined properties of the clustered bodies as follows:

$$m_c = \sum_{i \in \mathbf{C}} m_i \tag{5}$$

$$\mathbf{x}_c^n = 1/m_c \sum_{i \in \mathbf{C}} m_i \mathbf{x}_i^n \tag{6}$$

$$\mathbf{M}_{a,c} = \sum_{i \in \mathbf{C}} \left(\mathbf{M}_{a,i} + m_i \left(\mathbf{x}_i^n - \mathbf{x}_c^n \right)^* \left(\mathbf{x}_i^n - \mathbf{x}_c^n \right)^{*T} \right), \tag{7}$$

where $\mathbf{C}$ is the set of child bodies within the cluster. Equation (5) defines the cluster's mass, m_c, where m_i is child body i's mass. Equation (6) defines the time n position of the cluster, $\mathbf{x}_c$, as the center of mass for the bodies, where $\mathbf{x}_i$ is child body i's position. Equation (7) defines the cluster's rotational inertia, $\mathbf{M}_{a,c}$, where $\mathbf{M}_{a,i}$ is child body i's rotational inertia.

Next we compute the cluster's new velocity as follows:

$$\mathbf{V}_c = \mathbf{M}_c^{-1} \sum_{i \in \mathbf{C}} \begin{pmatrix} m_i \mathbf{I} & \mathbf{0} \\ m_i \left(\mathbf{x}_i^n - \mathbf{x}_c^n \right)^* & \mathbf{M}_{a,i} \end{pmatrix} \mathbf{V}_i, \tag{8}$$

where $\mathbf{V}_i$ is child body i's generalized velocity and $\mathbf{M}_c$ is the cluster's generalized inertia. Equation (8) finds the new generalized velocity of the cluster by computing the combined momentum of each child body about the cluster's center of mass and then multiplying it by the inverse of the cluster's generalized inertia.

We use this new generalized velocity to integrate the generalized cluster position $\mathbf{X}^n$ (a transformation matrix, where the initial time n orientation is the identity matrix) to a new generalized position, $\mathbf{X}^{new}$, using the rigid body integration scheme defined earlier in (3) and (4). Finally, we update the positions and velocities of the child bodies as follows:

$$\mathbf{X}_i^{new} = \mathbf{X}_c^{new} \left(\mathbf{X}_c^n \right)^{-1} \mathbf{X}_i^n \tag{9}$$

$$\mathbf{V}_i^{new} = \begin{pmatrix} \mathbf{I} & -\left(\mathbf{x}_i^n - \mathbf{x}_c^n \right)^* \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \mathbf{V}_c, \tag{10}$$

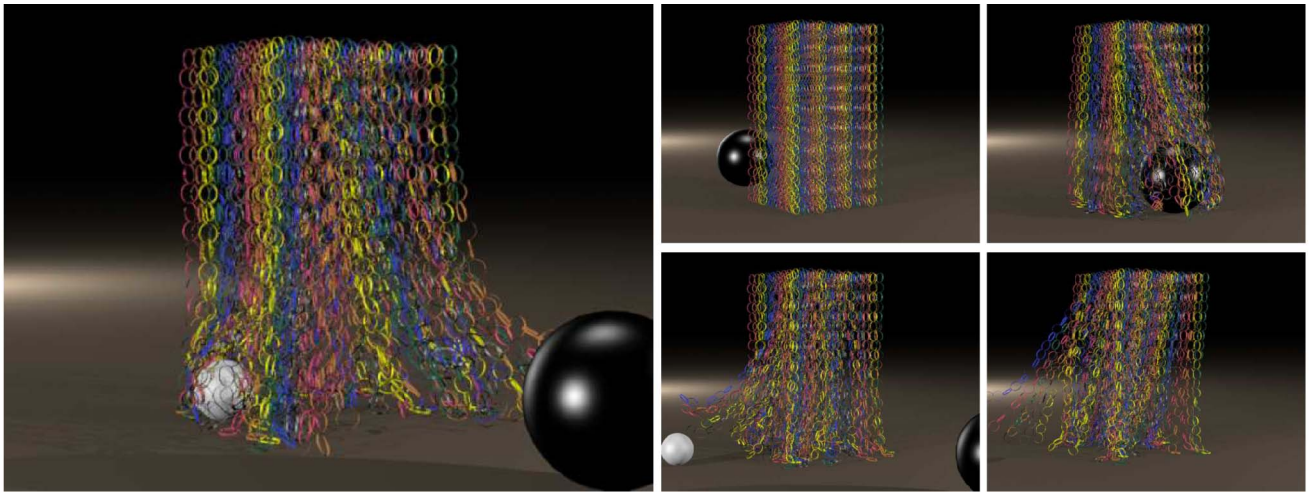


Fig. 10. Several hanging thin shell ring chains are hit by two balls rolling along the ground. This demonstrates the ability of our method to handle large quantities of thin shell objects.

where $\mathbf{X}_i^n$ is the time n generalized position for child body i. Equation (9) computes the new position of each child body by finding the relative position of the body within the cluster at time n, and then multiplying this relative position by the new position of the cluster. Equation (10) computes the new linear velocity of each child body as the pointwise velocity of the cluster while assigning the same angular velocity of the cluster to the child body.

When the cluster subsequently collides with other bodies, child body positions and velocities are updated again using (9) and (10).

4.2.2 Kinematic Rigidification

Before giving the details, we start with a simple motivating example. Suppose one has three bodies and can resolve the interpenetrations between two of these bodies. However, as a result of resolving these interpenetrations, the third body now impacts with the first two bodies causing them to again collide in subsequent iterations. Removing the interpenetrations between the first two bodies can again cause interpenetration with the third body and so on and so forth. Instead, after handling all the interpenetrations between the first two bodies, we cluster them together such that their relative velocities and positions at both the beginning and end of the step are preserved. We then collide the cluster with the third body. As a result, all the bodies will be interpenetration free and the first two bodies will still move relative to one another unlike the fully rigidifying case.

First note that we cannot form a kinematic cluster between two bodies unless we can remove all interference between them through contact detection and response iterations. Once we have noninterpenetrating trajectories for the bodies, we write the new time $n+1$ position of the cluster as follows:

$$\mathbf{x}_c = 1/m_c \sum_{i \in \mathbf{C}} m_i \mathbf{x}_i, \tag{11}$$

where $\mathbf{x}_i$ is the current time $n+1$ position of child body i. Equation (11) is the position of the center of mass of the cluster at the end of the time step if all the child bodies were allowed to freely evolve. The orientation for the generalized position is the identity matrix.

We also compute an effective velocity for the cluster:

$$\mathbf{v}_c = \frac{\mathbf{x}_c - \mathbf{x}_c^n}{\Delta t}, \tag{12}$$

where $\mathbf{x}_c^n$ is defined in (6). We also form a generalized velocity for the cluster, $\mathbf{V}_c$, noting that the angular velocity is zero in contrast to (8) because the initial rotation of child bodies over the time step is handled through the kinematic relative motion of the child bodies themselves.

Once a contact impulse has been computed and applied to the effective cluster velocity, $\mathbf{V}_c$, to get a new cluster velocity, $\mathbf{V}_c^{new}$, we reintegrate the old cluster position, $\mathbf{X}_c^n$, using the rigid body integration scheme defined in (3) and (4), to arrive at a new cluster time $n+1$ position $\mathbf{X}_c^{new}$. We then update the positions and velocities of the child bodies as shown below:

$$\mathbf{X}_i^{new} = \mathbf{X}_c^{new} \mathbf{X}_c^{-1} \mathbf{X}_i \tag{13}$$

$$\mathbf{V}_i^{new} = \mathbf{V}_i + \begin{pmatrix} \mathbf{I} & -\left(\mathbf{x}_i^n - \mathbf{x}_c^n\right)^* \\ \mathbf{0} & \mathbf{I} \end{pmatrix} \left(\mathbf{V}_c^{new} - \mathbf{V}_c\right). \tag{14}$$

Equation (13) updates the position of each child body to reflect the change in the time $n+1$ position of the cluster due to the impulse. Equation (14) updates the velocity of each child body by propagating the difference in velocity from the cluster. Note that the relative positions and velocities of the child bodies are preserved.

When comparing (13) and (14) to (9) and (10), note that instead of $(\mathbf{X}_c^n)^{-1}\mathbf{X}_i^n$ we have $\mathbf{X}_c^{-1}\mathbf{X}_i$ where both $\mathbf{X}_c$ and $\mathbf{X}_i$ are time $n+1$ quantities. The latter expression could be substituted into (9) to replace the prior expression, obtaining the same answer as long as one is careful to use the post clustering velocities of the individual objects in computing the time $n+1$ state. However, this adds extra complexity to (9), which is not needed. Similar statements hold for (10) and (14). The point is that the equations for kinematic clusters are more general, but do properly reduce to those for normal rigid clusters under appropriate assumptions.

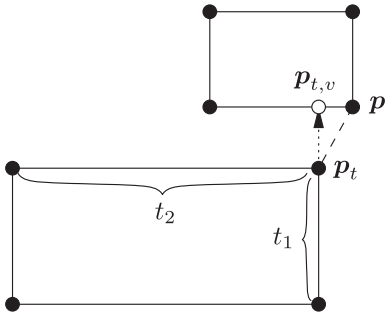


Fig. 11. Although vertex p is in proximity to faces t_1 and t_2, to prevent the boxes from catching when they are in sliding contact (since the contact normal would be in the direction of $\overrightarrow{pp_t}$) we do not create contacts between the vertex and either face. To illustrate our pruning procedure, consider the proximity between vertex p and face t_1. We first project the location of vertex p onto face t_1 to find point p_t. We next project point p_t onto the upper box to find point $p_{t,v}$. We then prune the proximity between vertex p and face t_1 because the angle between $\overrightarrow{pp_t}$ and $\overrightarrow{p_{t,v}p_t}$ is greater than $\theta_{\text{proximity}}$. The same process is used to prune the proximity with face t_2.

4.3 Static Contact

Typically, algorithms such as in [15] use level set depth testing to determine contacts between pairs of bodies. However, as mentioned above, this method does not work for thin shell processing without thickening the geometry. Instead, we present a feature pair proximity contact detection algorithm that allows for more accurate and stable stacking.

4.3.1 Contact Detection

Since we are guaranteed an interpenetration free configuration after the dynamic contact and the fail-safe steps of our integration scheme, it is necessary to use proximity detection to find contact points. We proceed by finding nearest points on vertex/triangle and edge/edge pairs that are nearer than a contact proximity, $d_{\text{proximity}}$. We generally use the direction between pairs of nearest points as the contact normal and one of the points as the contact location. However, these normals are not guaranteed to be within the admissible region for both contact locations on each body. In fact, even in simple cases such as a box sliding on a flat surface, contacts are likely to occur with inaccurate normals as illustrated in Fig. 11. Note that the same scenario can also occur away from sharp corners, such as when vertices are near edges between colinear faces. While the classification from Section 3 could be used to eliminate contacts with inadmissible normals, a robust implementation handling nonmanifold geometry and vertices with concave incident edges would require considering many cases. Instead, we apply a simple heuristic to filter contacts.

When computing vertex/triangle contacts, we only consider vertex/triangle pairs, where the triangle is the nearest face to the vertex. For each of these pairs, we project the vertex p onto the triangle to find the point p_t. We next project p_t onto the vertex body (the body containing the vertex) to find the point $p_{t,v}$. We then cull the vertex/triangle proximity, if the angle between $\overrightarrow{pp_t}$ and $\overrightarrow{p_{t,v}p_t}$ is greater than $\theta_{\text{proximity}}$. Otherwise, we create a contact using $\overrightarrow{pp_t}/|\overrightarrow{pp_t}|$ as the contact normal, p as the contact location, and $|\overrightarrow{pp_t}|$ as the contact distance. See Fig. 11 for an illustration of this process. For edge/edge proximities, we first find the nearest points in proximity, p_1 and p_2. We next project these points onto their respective opposing bodies giving points p_{1,e_2} and p_{2,e_1}. We then cull the proximity if the angle between either $\overrightarrow{p_1p_{1,e_2}}$ and $\overrightarrow{p_1p_2}$ or $\overrightarrow{p_2p_{2,e_1}}$ and $\overrightarrow{p_2p_1}$ exceeds $\theta_{\text{proximity}}$. Otherwise, we create a contact using $\overrightarrow{p_1p_2}/|\overrightarrow{p_1p_2}|$ as the contact normal, p_1 as the contact location, and $|\overrightarrow{p_1p_2}|$ as the contact distance. Furthermore, in both vertex/triangle and edge/edge cases, to prevent contacts being created between feature pairs, which are occluded we cast a ray between the points and reject the proximity if an intersection is found.

4.3.2 Contact Resolution

Once the set of contact points has been found, a set of contact impulses resolving any approaching velocities is computed. Guendelman et al. [15] proceed by iterating over each contact, computing an impulse and then immediately applying that impulse before moving on to the next contact. Unfortunately, this can result in excessive forces being applied at certain contacts, causing bodies to separate rather

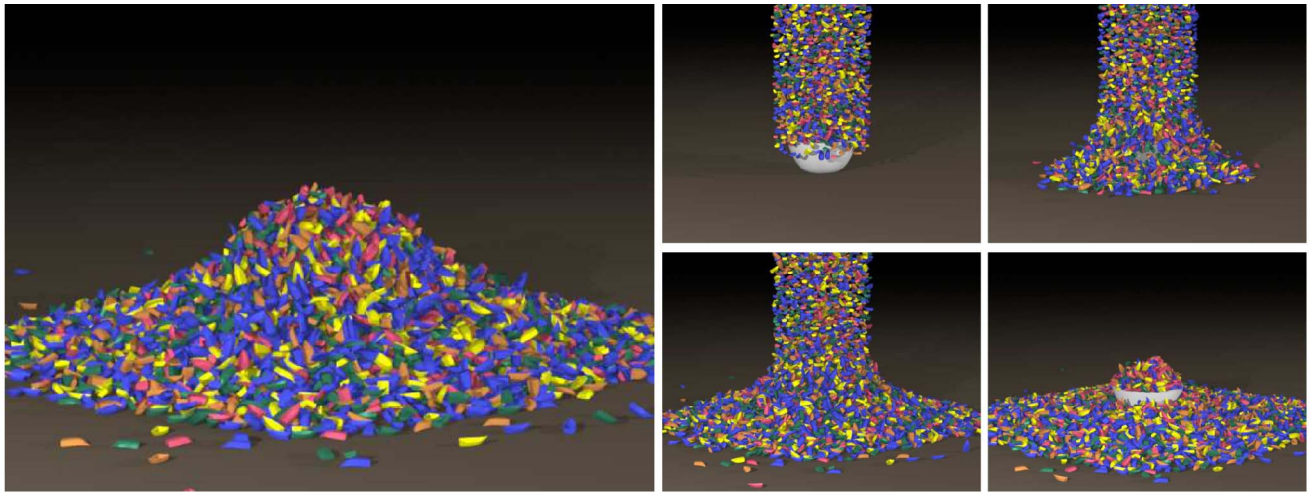


Fig. 12. Ten thousand thin shell boats, made of triangles with no interior, are dropped into a bowl demonstrating the scalability of our method even in the presence of high-velocity objects.

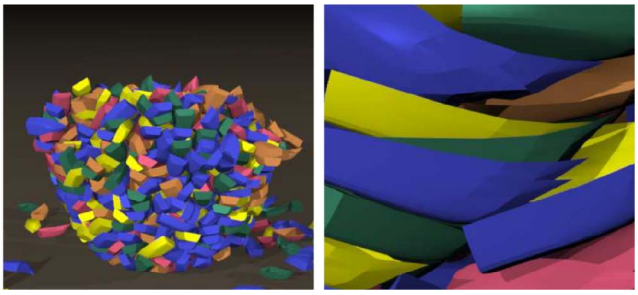


Fig. 13. A simulation of 1,000 boats dropped into a bowl (bowl not rendered). Note in (right) that the boats are tightly packed within the bowl, but remain interpenetration free.

than stay in contact. Guendelman et al. [15] mitigated this by scaling the impulse applied at each contact by a small epsilon parameter. This allowed the effect of impulses to be at least partly propagated to other contacts in further iterations to prevent excessive overshooting. The immediate consequence of this approach is that the convergence rate can severely deteriorate, depending upon the epsilon parameter chosen.

Instead, we use a modified projected Gauss Seidel approach (see [29], [39] for a reference to other projected Gauss Seidel implementations) to handle contact resolution. In general, projected Gauss Seidel converges much more quickly than the sequential impulse approach of [15]. Projected Gauss Seidel does not need to apply epsilon scaling to achieve stable results because it works toward the solution of the NCP and, when converged, does not introduce a bias in the solution dependent upon the order in which contacts are handled. The order in which the constraints are evaluated and impulses are applied does not change the solution in the limit.

We base our contact model on that described in [29], and modify it to target a nonzero relative velocity in the normal direction allowing bodies in contact to exactly come to rest. Similar to dynamic contact, we compute the desired relative velocity at each contact as $(d_{\text{current}} - d_{\text{rest}})/\Delta t$ that we use as the right-hand side in the nonpenetration constraint. We now state the modified equations from [29]. For the kth contact, we define the relative velocity as

$$\boldsymbol{v}_{k,\text{rel}} = \underbrace{\begin{pmatrix} -\mathbf{J}_{k,i} & \mathbf{J}_{k,j} \end{pmatrix}}_{\mathbf{J}_k} \underbrace{\begin{pmatrix} \mathbf{V}_i^{n+1} \\ \mathbf{V}_j^{n+1} \end{pmatrix}}_{\mathbf{V}^{n+1}} = \mathbf{J}_k \mathbf{V}^{n+1}, \tag{15}$$

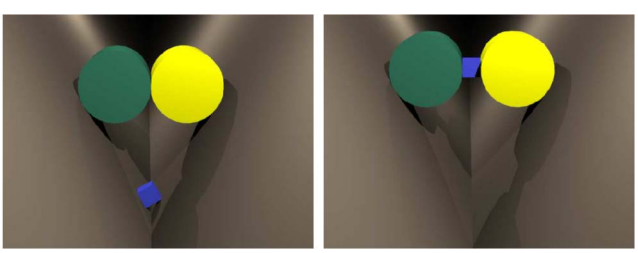


Fig. 14. A block is suspended between two cylinders being compressed by two planes under gravity. (Left) Bias in the solution given by the sequential impulses method allows the block to slip. (Right) Our scheme computes the correct frictional impulses to statically suspend the block.

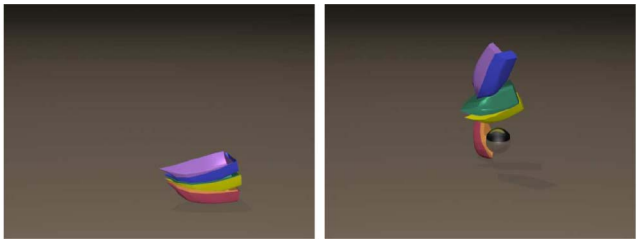


Fig. 15. A stack of boats at rest is hit by a ball. The ball is moving fast enough such that the collision between the balls and the stack would be missed without continuous collision detection at larger time steps.

where $\mathbf{J}_{k,i} = (\mathbf{I} - \mathbf{r}_i^*)$ and $\mathbf{J}_{k,j} = (\mathbf{I} - \mathbf{r}_j^*)$ are Jacobians mapping the rigid velocities of the bodies in contact (indexed by i and j) to the pointwise velocity at contact k. The modified nonpenetration constraint is then as follows:

$$\boldsymbol{n}_k^T \boldsymbol{v}_{k,\text{rel}} \geq \frac{d_{\text{rest}} - d_{k,\text{current}}}{\Delta t}, \tag{16}$$

where $\boldsymbol{n}_k$ is the contact normal and $d_{k,\text{current}}$ is the current distance between the bodies in contact. The right-hand side of (16) allows bodies to settle into exact contact at the rest distance in a single time step. This both improves the stability of our simulation as well as reduces the subsequent number of collision and contact iterations. Note that it is important when implementing this constraint that the rest distance, d_{rest}, is less than the $d_{\text{proximity}}$, so that objects in contact will be processed in the collision step and subsequently processed in the second static contact step to prevent jittering. Similar formulations, such as the post-stabilization methods of [40], [41], [42], also account for relative motion. However, they do not necessarily intend to anticipate contacts between bodies not already in contact, rather using it to prevent objects from penetrating too deeply or jittering. The complementarity condition for this constraint then becomes

$$\lambda_{k,n} \geq 0 \perp \left(\boldsymbol{n}_k^T \boldsymbol{v}_{k,\text{rel}} - \frac{d_{\text{rest}} - d_{k,\text{current}}}{\Delta t} \right) \geq 0, \tag{17}$$

where $\lambda_{k,n}$ is the contact impulse in the normal direction. The friction conditions and contact impulse definition

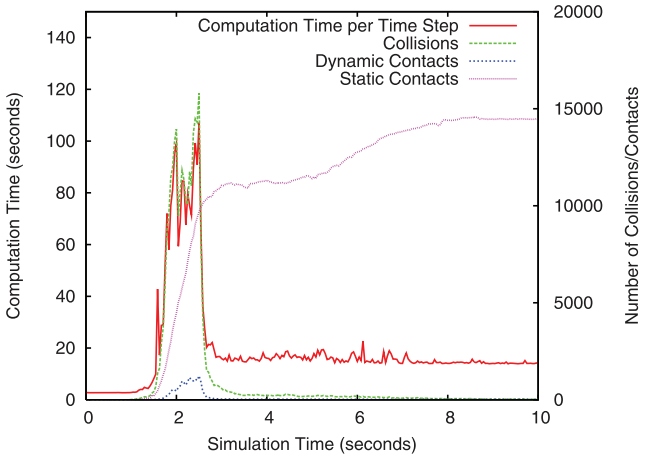


Fig. 16. The computation time and number of collisions/contacts per frame plotted against simulation time for the first 10 seconds of the example with 1,000 boats dropped into a bowl.

Example	Number of Bodies	Time per Step	Collisions/Dynamic Contact		Static Contact	
			Detection	Response	Detection	Response
Boats Falling into Bowl	102	$1.7s$	$1.3s$	$0.4ms$	$0.20s$	$0.18s$
	252	$4.8s$	$3.9s$	$1.1ms$	$0.46s$	$0.52s$
	502	$9.6s$	$8.0s$	$2.5ms$	$0.84s$	$0.80s$
	1002	$20.9s$	$17.7s$	$6.0ms$	$1.45s$	$1.61s$
Hanging Rings	2003	$19.1s$	$15.7s$	$9.8ms$	$0.30s$	$1.93s$
Triangles Falling onto Pegs	186	$0.45s$	$0.34s$	$0.6ms$	$0.02s$	$0.08s$

Fig. 17. Timing information per time step for the first 200 steps of several simulations. In these tests, the memory usage was less than 10 MB due to geometry instancing and was greatly outweighed by the overhead of our framework.

remain unchanged from [29] and for completeness are included. The four sided friction pyramid is defined as

$$\lambda_{k,s} = -\mu\lambda_{k,n} \Rightarrow \boldsymbol{t}_{k,s}^T \boldsymbol{v}_{k,\text{rel}} \geq 0, \quad s = 1, 2 \tag{18}$$

$$\lambda_{k,s} = \mu\lambda_{k,n} \Rightarrow \boldsymbol{t}_{k,s}^T \boldsymbol{v}_{k,\text{rel}} \leq 0, \quad s = 1, 2 \tag{19}$$

$$-\mu\lambda_{k,n} < \lambda_{k,s} < \mu\lambda_{k,n} \Rightarrow \boldsymbol{t}_{k,s}^T \boldsymbol{v}_{k,\text{rel}} = 0, \quad s = 1, 2, \tag{20}$$

where $\boldsymbol{t}_{k,1}$ and $\boldsymbol{t}_{k,2}$ are the tangent vectors for contact k, $\lambda_{k,1}$ and $\lambda_{k,2}$ are the contact impulses in the tangent directions, and μ is the coefficient of friction. The impulse due to contact k is then defined as

$$\boldsymbol{l}_k = \boldsymbol{n}_k\lambda_n + \boldsymbol{t}_{k,1}\lambda_{k,1} + \boldsymbol{t}_{k,2}\lambda_{k,2}. \tag{21}$$

To compute updated time $n+1$ velocities, we explicitly integrate the contact impulse defined in (21) by multiplying it by $\mathbf{J}_k^T$ to compute the linear and angular impulses for each body and then using the momentum update equation

$$\mathbf{V}^{n+1} = \hat{\mathbf{V}}^n + \mathbf{M}^{-1}\boldsymbol{l}_k, \tag{22}$$

where $\hat{\mathbf{V}}$ is the precontact generalized velocity, $\mathbf{a}$ is the acceleration due to explicit forces, and $\mathbf{M}$ is the generalized mass matrix for the bodies in contact. When assembling a system for multiple contacts, the momentum update equation for each body is modified to contain the sum of the impulse terms from each contact is involved in.

Once the full system has been built, we solve it using the projected Gauss Seidel iteration as described in [29]. To check for convergence, we use the L_∞ norm of the constraint violations, stopping when it is less than a user specified tolerance. In addition, we also use an iteration limit and once it is reached we apply the shock propagation scheme from [29]. This enables us to efficiently handle large stacks. We would also like to note that other iterative approaches such as the one in [40] could be used to implement a more accurate friction cone. However, by randomly choosing the tangent vectors, any bias in large piles of objects due to the friction cone approximation was imperceptible.

Time Step Size	Time per Step
$1/96s$	$43.7ms$
$1/48s$	$73.6ms$
$1/24s$	$48.0ms$
$1/12s$	$114.7ms$
$1/6s$	$215.2ms$

Fig. 18. A comparison of the computation times for the example shown in Fig. 15 when run with different time steps.

5 Examples

Unless otherwise noted all simulations were run with the parameters listed in Fig. 20. For timing information on several examples see Fig. 17.

Figs. 2 and 15 show examples where high-speed collisions are handled by our method. In Fig. 2, the interference testing used by Guendelman et al. [15] allows the block to pass through the plank entirely missing the collision, while our method detects the collision and produces a plausible response. We explore the computational cost for the example in Fig. 15 with varying time steps in Fig. 18 and note that we do see a reduced overall computational cost when taking larger time steps; however, the improvement diminishes for larger time steps. One example where [15] breaks down, even in the volumetric case, is shown in Fig. 14, where a box is pinched between two cylinders in a funnel. Since the contact algorithm in [15] is biased, the box eventually slips. Instead, by applying a static contact step, we accurately find the sticking solution, allowing the box to remain suspended by friction alone.

Fig. 9 demonstrates the ability of our fail-safe to plausibly resolve collisions. In this case, a stack of boxes is hit in the middle by a ball. We limit the number of collision and contact iterations such that the fail-safe is forced to rigidify the entire stack. Without kinematic rigidification, the stack is rigidified statically to the ground and the ball freezes once it collides with the stack. Even if the algorithm was modified to not rigidify with the ground, the impact of the ball would be distributed evenly about the entire stack. The stack would then tip over as a single cluster until subsequent steps when the remaining collisions are resolved before the fail-safe is applied. With kinematic

Collisions/Dynamic Contact Iterations	Static Contact Iterations	Time per Step
10	1000	$38.9s$
20		$19.8s$
50		$22.3s$
100		$20.9s$
100	20	$96.5s$
	50	$41.6s$
	100	$31.2s$
	250	$23.3s$
	1000	$20.9s$

Fig. 19. A comparison of the computation times for the 1,000 boats dropping into a bowl example as, shown in Fig. 13, when run with different iteration limits.

Parameter	Value
Time Step (Fixed)	$1/24s$
Collision Iterations	100
Collision Pair Iterations	4
Dynamic Contact Iterations	100
Dynamic Contact Pair Iterations	20
Static Contact Iterations	1000
Static Contact Tolerance	10^{-6}
Gravity Constant	$9.8m/s^2$
Contact Proximity ($d_{\text{proximity}}$)	$0.02m$
Contact Angle Threshold ($\theta_{\text{proximity}}$)	3°
Rest Distance (d_{rest})	$0.01m$
Coefficient of Friction (μ)	0.1
Coefficient of Resitution (ϵ)	0.1

Fig. 20. List of parameters that were used for the examples in this paper. We note that $d_{\text{proximity}}$ and d_{rest} are set relative to the size of the objects in our tests that were on the order of 1 to 10 m in size.

rigidification, the impacted box is allowed to slide such that the stack breaks up immediately.

Fig. 1 shows a set of degenerate triangles (thin shells with their geometry lying in a single plane) stacking both flat on the ground and leaning up against the fixed pegs. The close-up views in Fig. 13 demonstrate the ability of our algorithm to tightly pack concave objects. We extend this example to 10,000 falling bodies in Fig. 12.

In Fig. 19, we give the timings for our 1,000 boats falling into a bowl example, run with varying numbers of iterations. We found that it was necessary to take a minimum of 50 collision/dynamic contact iterations to avoid excessive rigidification in more complicated examples such as in Figs. 13 and 10. We found that simulation times were overall faster when taking more iterations due to fail-safe iterations being more expensive in our implementation, particularly in stacks, where rigidifying bodies resulted in large numbers of new collisions. Furthermore, since we discard the velocities from dynamic contact and the fail-safe, collisions not resolved during processing in one time step were necessarily handled in subsequent time steps still requiring the same work. While we applied shock propagation and achieved excellent results for free standing stacks, for cases where there is no single shock direction such as in Figs. 13 and 10, it was necessary to take several hundred iterations to allow objects to settle stably. We found that it was very important to accurately solve the system in the static contact step to achieve reasonable scaling by only relying upon dynamic contact to handle collisions and contacts due to nonlinear motion or high speeds. This also further improved performance since static contact iterations are much less expensive than our collision or dynamic contact iterations. Fig. 16 shows the computation time and number of collisions and contacts per time step plotted against time. Note that the majority of the computational expense occurs during the initial impact and stacking of the boats in the bowl and that once they come to rest the number of contacts stabilize and the cost decreases.

6 Conclusions

We have addressed several unresolved problems within rigid body simulation to enable the interpenetration free simulation of rigid bodies with polygon soup type geometric representations at framerate time steps. We accomplished this by integrating a linearized continuous collision detection method into an iterative collision response algorithm and coupling it with a more accurate fail-safe. Specifically, this has allowed the unthickened simulation of completely thin shells, even degenerately thin bodies where all of the geometry lies within a single plane. In addition to enabling interpenetration free simulation, we have proposed a proximity based contact detection and iterative contact response algorithm. This contact algorithm both avoids the inside/outside problem for finding contacts between thin shell bodies as well as improves the performance of our overall method by handling the majority of contacts and collisions without expensive continuous collision detection. One caveat with our fail-safe is that, as with most interpenetration free methods, when multiple infinite mass bodies are moving with different velocities, finding a solution, even if one exists, is a problem. For example, consider the scenario when a body is being pushed or pulled along the ground by a second infinite mass object. Note that although our contact algorithm scales well due to the use of a contact graph in shock propagation, parallelization to a large number of processors and GPUs is complex and an avenue for future research.

Acknowledgments

The authors would like to acknowledge Craig Schroeder for suggesting the example in Fig. 14 and Robert Bridson for ideas leading us to develop our improved fail-safe. R. Elliot English was supported in part by a Stanford Graduate Fellowship and NSERC Postgraduate Scholarship. Michael Lentine was supported in part by an Intel Graduate Fellowship. Research was supported in part by National Science Foundation (NSF) IIS-1048573, ONR N00014-09-1-0101, ONR N-00014-11-1-0027, ONR N00014-06-1-0505, ARL AHPCRC W911NF-07-0027, the Intel Science and Technology Center for Visual Computing, and ONR N00014-05-1-0479 for a computing cluster.

References

[1] J. Hahn, "Realistic Animation of Rigid Bodies," *ACM SIGGRAPH Computer Graphics*, vol. 22, no. 4, pp. 299-308, 1988.
[2] M. Moore and J. Wilhelms, "Collision Detection and Response for Computer Animation," *ACM SIGGRAPH Computer Graphics*, vol. 22, no. 4, pp. 289-298, 1988.
[3] D. Baraff, "Analytical Methods for Dynamic Simulation of Non-Penetrating Rigid Bodies," *ACM SIGGRAPH Computer Graphics*, vol. 23, no. 3, pp. 223-232, 1989.
[4] D. Baraff, "Curved Surfaces and Coherence for Non-Penetrating Rigid Body Simulation," *ACM SIGGRAPH Computer Graphics*, vol. 24, no. 4, pp. 19-28, 1990.
[5] D. Baraff, "Coping with Friction for Non-Penetrating Rigid Body Simulation," *ACM SIGGRAPH Computer Graphics*, vol. 25, no. 4, pp. 31-40, 1991.
[6] D. Baraff, "Fast Contact Force Computation for Nonpenetrating Rigid Bodies," *Proc. ACM SIGGRAPH '94*, pp. 23-34, 1994,
[7] D. Baraff, "Interactive Simulation of Solid Rigid Bodies," *IEEE Computer Graphics and Applications*, vol. 15, no. 3, pp. 63-75, May 1995.
[8] R. Barzel, J. Hughes, and D. Wood, "Plausible Motion Simulation for Computer Graphics Animation," *Proc. Eurographics Workshop Computer Animation and Simulation '96*, pp. 183-197, 1996.

[9] A. Chatterjee, "On the Realism of Complementarity Conditions in Rigid Body Collisions," *Nonlinear Dynamics,* vol. 20, no. 2, pp. 159-168, 1999.

[10] B. Thomaszewski, A. Gumann, S. Pabst, and W. Strasser, "Magnets in Motion," *ACM Trans. Graphics,* vol. 27, no. 5, pp. 162:1-162:9, 2008.

[11] T. Shinar, C. Schroeder, and R. Fedkiw, "Two-Way Coupling of Rigid and Deformable Bodies," *Proc. ACM SIGGRAPH/ Eurographics Symp. Computer Animation (SCA '08),* pp. 95-103, 2008.

[12] S.-W. Hsu and J. Keyser, "Statistical Simulation of Rigid Bodies," *Proc. ACM SIGGRAPH/Eurographics Symp. Computer Animation (SCA '09),* 2009.

[13] J. Su, C. Schroeder, and R. Fedkiw, "Energy Stability and Fracture for Frame Rate Rigid Body Simulations," *Proc. ACM SIGGRAPH/ Eurographics Symp. Computer Animation,* pp. 155-164. 2009,

[14] C. Zheng and D.L. James, "Rigid-Body Fracture Sound with Precomputed Soundbanks," *ACM Trans. Graphics,* vol. 29, no. 3, article 69, http://www.cs.cornell.edu/projects/fracturesound/, July 2010.

[15] E. Guendelman, R. Bridson, and R. Fedkiw, "Nonconvex Rigid Bodies with Stacking," *ACM Trans. Graphics* vol. 22, no. 3, pp. 871-878, 2003.

[16] M.K. Ponamgi, D. Manocha, and M.C. Lin, "Incremental Algorithms for Collision Detection between Solid Models," *Proc. ACM Symp. Solid Modeling and Applications,* pp. 293-304. 1995,

[17] M. Lin and S. Gottschalk, "Collision Detection between Geometric Models: A Survey," *Proc. IMA Conf. Math. of Surfaces,* pp. 37-56, 1998.

[18] Y. Kim, M. Otaduy, M. Lin, and D. Manocha, "Fast Penetration Depth Computation for Physically-Based Animation," *Proc. ACM Symp. Computer Animation,* 2002.

[19] S. Redon and M.C. Lin, "A Fast Method for Local Penetration Depth Computation," *J. Graphics Tools,* vol. 11, pp. 37-50, 2006.

[20] C.J. Ong and E. Gilbert, "The Gilbert-Johnson-Keerthi Distance Algorithm: A Fast Version for Incremental Motions," *Proc. IEEE Int'l Conf. Robotics and Automation,* vol. 2, pp. 1183-1189, Apr. 1997.

[21] S. Redon, A. Kheddar, and S. Coquillart, "Fast Continuous Collision Detection between Rigid Bodies," *Computer Graphics Forum,* vol. 21, no. 3, pp. 279-288, 2002.

[22] S. Redon, A. Kheddar, and S. Coquillart, "Gauss' Least Constraints Principle and Rigid Body Simulations," *Proc. IEEE Int'l Conf. Robotics and Automation (ICRA '02),* vol. 1, pp. 517-522, 2002.

[23] M. Otaduy, R. Tamstorf, D. Steinemann, and M. Gross, "Implicit Contact Handling for Deformable Objects," *Computer Graphics Forum,* vol. 28, no. 2, pp. 559-568, 2009.

[24] M. Tang, D. Manocha, M.A. Otaduy, and R. Tong, "Continuous Penalty Forces," *ACM Trans. Graphics,* vol. 31, no. 4, pp. 107:1-107:9, http://doi.acm.org/10.1145/2185520.2185603, July 2012.

[25] Z. Bao, J. Hong, J. Teran, and R. Fedkiw, "Fracturing Rigid Materials," *IEEE Trans. Visualization Computer Graphics,* vol. 13, no. 2, pp. 370-378, Mar./Apr. 2007.

[26] X. Provot, "Collision and Self-Collision Handling in Cloth Model Dedicated to Design Garment," *Proc. Graphics Interface,* pp. 177-189, 1997.

[27] R. Bridson, R. Fedkiw, and J. Anderson, "Robust Treatment of Collisions, Contact and Friction for Cloth Animation," *ACM Trans. Graphics,* vol. 21, no. 3, pp. 594-603, 2002.

[28] M. Tang, D. Manocha, S.-E. Yoon, P. Du, J.-P. Heo, and R.-F. Tong, "VoICCD: Fast Continuous Collision Culling between Deforming Volume Meshes," *ACM Trans. Graphics,* vol. 30, no. 5, pp. 111:1-111:15, http://doi.acm.org/10.1145/2019627.2019630, Oct. 2011.

[29] K. Erleben, "Velocity-Based Shock Propagation for Multibody Dynamics Animation," *ACM Trans. Graphics,* vol. 26, article 12, http://doi.acm.org/10.1145/1243980.1243986, June 2007.

[30] D. Harmon, E. Vouga, R. Tamstorf, and E. Grinspun, "Robust Treatment of Simultaneous Collisions," *Proc. ACM SIGGRAPH '08 Papers,* pp. 23:1-23:4, http://doi.acm.org/10.1145/1399504.1360622, 2008.

[31] D. Kaufman, T. Edmunds, and D. Pai, "Fast Frictional Dynamics for Rigid Bodies," *ACM Trans. Graphics,* vol. 24, no. 3, pp. 946-956, 2005.

[32] D. Kaufman, S. Sueda, D. James, and D. Pai, "Staggered Projections for Frictional Contact in Multibody Systems," *ACM Trans. Graphics,* vol. 27, no. 5, pp. 164:1-164:11, 2008.

[33] C.D. Twigg and D.L. James, "Many-Worlds Browsing for Control of Multibody Dynamics," *Proc. ACM SIGGRAPH '07 Papers,* http://doi.acm.org/10.1145/1275808.1276395, 2007.

[34] C.D. Twigg and D.L. James, "Backward Steps in Rigid Body Simulation," *Proc. ACM SIGGRAPH '08 Papers,* pp. 25:1-25:10, http://doi.acm.org/10.1145/1399504.1360624, 2008.

[35] T.Y. Yeh, G. Reinman, S.J. Patel, and P. Faloutsos, "Fool Me Twice: Exploring and Exploiting Error Tolerance in Physics-Based Animation," *ACM Trans. Graphics,* vol. 29, pp. 5:1-5:11, http://doi.acm.org/10.1145/1640443.1640448, Dec. 2009.

[36] G. Grabner and A. Kecskeméthy, "An Integrated Runge-Kutta Root Finding Method for Reliable Collision Detection in Multibody Systems," *Multibody System Dynamics,* vol. 14, pp. 301-316, http://dx.doi.org/10.1007/s11044-005-2640-6, 2005, doi: 10.1007/s11044-005-2640-6.

[37] B. Kim and J. Rossignac, "Collision Prediction for Polyhedra under Screw Motions," *Proc. Eighth ACM Symp. Solid Modeling and Applications (SM '03),* pp. 4-10, http://doi.acm.org/10.1145/781606.781612, 2003.

[38] E. Schömer, J. Sellen, and M. Welsch, "Exact Geometric Collision Detection," *Proc. Seventh Canadian Conf. Computational Geometry,* pp. 211-216, 1995.

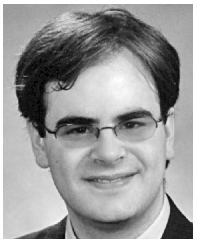
[39] M. Silcowitz-Hansen, S. Niebe, and K. Erleben, "A Nonsmooth Nonlinear Conjugate Gradient Method for Interactive Contact Force Problems," *Visual Computer,* vol. 26, pp. 893-901, http://dx.doi.org/10.1007/s00371-010-0502-6, June 2010.

[40] M. Anitescu and A. Tasora, "An Iterative Approach for Cone Complementarity Problems for Nonsmooth Dynamics," *Computational Optimization and Applications,* vol. 47, no. 2, pp. 207-235, http://dx.doi.org/10.1007/s10589-008-9223-4, Oct. 2010.

[41] A. Tasora and M. Anitescu, "A Matrix-Free Cone Complementarity Approach for Solving Large-Scale, Nonsmooth, Rigid Body Dynamics," *Computer Methods in Applied Mechanics and Eng.,* vol. 200, pp. 439-453, 2011.

[42] N. Chakraborty, S. Berard, S. Akella, and J. Trinkle, "A Fully Implicit Time-Stepping Method for Multibody Systems with Intermittent Contact," *Robotics: Science and Systems,* http://www.cs.rpi.edu/trink/Papers/CBATrss07.pdf, June 2007.

R. Elliot English received the BS degree in computer science from the University of British Columbia in 2008, and is currently working toward the PhD degree at Stanford University under the support of both the Stanford Graduate Fellowship and an NSERC Postgraduate Doctoral Scholarship. While at UBC, he worked on several research projects involving the numerical simulation of coupled rigid and deformable bodies, and isometrically deforming cloth.



Michael Lentine received the BS degree in computer science from Carnegie Mellon University in 2007, and is currently working toward the PhD degree at Stanford University, where he received the Intel Graduate Fellowship. While there, he was an undergraduate research assistant working on computer animation research. He has also been a consultant for Industrial Light + Magic for the last two years working in the research and development group on physical simulation.

Ron Fedkiw received the PhD degree in mathematics from the University of California, Los Angeles, in 1996 and the postdoctoral studies both at UCLA in mathematics and at Caltech in aeronautics before joining the Stanford Computer Science Department. He received an Academy Award from The Academy of Motion Picture Arts and Sciences, the National Academy of Science Award for Initiatives in Research, a Packard Foundation Fellowship, a Presidential Early Career Award for Scientists and Engineers (PECASE), a Sloan Research Fellowship, the ACM Siggraph Significant New Researcher Award, an Office of Naval Research Young Investigator Program Award (ONR YIP), the Okawa Foundation Research Grant, the Robert Bosch Faculty Scholarship, the Robert N. Noyce Family Faculty Scholarship, two distinguished teaching awards, and so on. Currently, he is on the editorial board of the *Journal of Computational Physics*, *Journal of Scientific Computing*, *SIAM Journal on Imaging Sciences*, and *Communications in Mathematical Sciences*, and he participates in the reviewing process of a number of journals and funding agencies. He has published more than 90 research papers in computational physics, computer graphics and vision, as well as a book on level set methods. For the past seven years, he has been a consultant with Industrial Light + Magic. He received screen credits for his work on "Terminator 3: Rise of the Machines," "Star Wars: Episode III - Revenge of the Sith," "Poseidon," and "Evan Almighty."