DOI:10.1145/2133806.2133828

Asynchronous Contact Mechanics

By David Harmon,* Etienne Vouga, Breannan Smith, Rasmus Tamstorf, and Eitan Grinspun

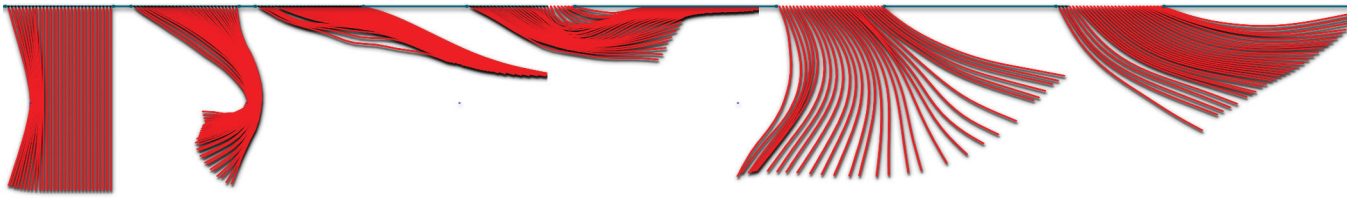


Figure 1. A prescribed particle slowly moves through a set of curtains, then impulsively shifts to a very high velocity. The slow and fast phases highlight the method's ability to handle smooth resting and sliding with deep stacking, and arbitrarily fast penetration-free movements in which collisions are treated when (as opposed to well before or after) they occur. The curtains continue to swing for a long time, even as controlled internal dissipation damps high frequencies.

1. MOTIVATION

Physicists have long observed physical phenomena, such as the motion of fluids and the interaction of galaxies, and developed mathematical models to describe these systems. More recently, the advent of computers has allowed us to implement these models as software in a computational environment, launching the field of physical simulation. On a computer we are able to recreate and study physical phenomena within a controlled setting both for descriptive as well as exploratory purposes, leading to advancements in design, engineering, and entertainment.

However, even as computer hardware benefits from Moore's Law, our ability to program, debug, and maintain software advances at a slower pace. This observation shapes our priorities as we develop physical simulation tools for computer graphics. While making choices that yield up-front simplicity and blazing performance is important today, we prefer that these choices do not obstruct our long-term goals of extending functionality and improving physical realism. Laying aside ad hoc models in favor of physical approaches might require a deeper initial investment, but it promises to pay off handsomely in predictability, controllability, and extensibility.

1.1. Safety, correctness, progress

One particularly difficult aspect of simulation is the modeling of complex collisions. A collision occurs when two objects attempt to occupy the same point in space at the same time. Even simple scenarios, like a crumpled shirt, contain an extraordinary number of these contacting points that arise and disappear through the course of a simulation. Robust simulation of complex contact scenarios is critical to applications spanning graphics (training, virtual worlds, entertainment) and engineering (product design, safety analysis, experimental validation). The presence of frequent and plentiful collisions (Figure 1), interactions involving sharp boundaries, resting and sliding contact, and all combinations thereof make it challenging to simulate contact reliably. The inability to handle these difficult situations results in interpenetration, visual artifacts where objects intersect one another—a clearly unphysical configuration. Useful resolution of these scenarios requires consideration of the fundamental issues of geometric *safety*, physical *correctness*, and computational *progress*. These have the respective meanings that (a) for well-posed problems the simulation does not enter an invalid (interpenetrating) state, (b) collision response obeys physical laws of causality and conservation (of mass, momentum, energy, etc.), and (c) the algorithm completes a simulation in finite, preferably short, time. An ideal algorithm offers provable guarantees of safety, correctness, and progress that hold even in the discrete setting of a computer. A safety guarantee eliminates the need to iterate through the animation-design process because of unsightly penetration artifacts; such a guarantee should not fall on a user overburdened with tunable parameters. Respecting discrete conservation laws allows for the development of controllable dissipation without artificial numerical damping. Respect for causality is critical to capturing chain reactions and phenomena such as wave propagation and stacking. If, however, these two guarantees are not accompanied by guaranteed progress, the simulation may never complete, no matter how fast or parallel the hardware.

1.2. Shortcomings of synchrony

Dynamic simulations progress by integrating differential equations, such as Newton's familiar second law, over small steps in time. Most of these integration methods are synchronous, moving the entire configuration forward in lock-step from one instant in time to the next. Such synchrony is fundamentally at odds with safety, correctness, and progress: the first two goals are assured by attending to collisions in order of causality, which, since collisions may propagate at unbounded speed, can require arbitrarily small time steps. The number of possible impact events in a single

The original version of this paper was published in *Proceedings of SIGGRAPH '09*, July 2009, ACM.

*Now at New York University. Work was done while at Columbia University.

"reasonable" time step can be enormous: in their analysis of contact, Cirak and West[5] present a counting argument and conclude that synchronous "contact simulation algorithms cannot attempt to exactly compute the sequence and timing of all impacts," as this would preclude reasonable progress.

The graphics community's prevailing emphasis on *progress* has motivated many efforts to find, *retroactively*, a physically plausible resolution to a given set of collisions that occurred over a preceding time interval.[4, 20] Such methods typically have adjustable parameters that must be carefully chosen to balance safety and progress; other methods discard causality in favor of progress.[19] The principled, faithful simulation of complex collisions for deformable objects, such as cloth and other flexible materials, remains an open, challenging, and important problem.

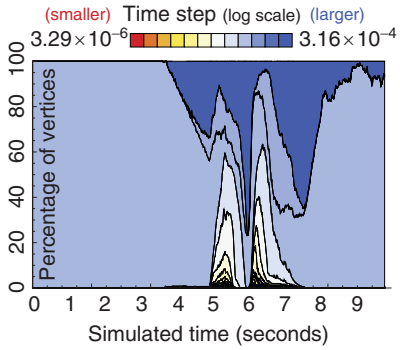
1.3. Asynchrony

We propose to place safety and correctness on an equal footing with progress. To overcome the fundamental opposition between these requirements, we turn to *asynchronous integration*, which integrates each geometric element of a discrete shape (e.g., the stretching resistance of cloth defined across a triangle) at its own pace, *not* in lockstep with the entire object. Asynchrony offers compelling long-term advantages for simulations of deformable objects in complex contact—advantages that remain unexplored, in particular, in terms of safety, correctness, and progress. For scenarios involving sharp boundaries or dispersed points of contact, such as crumpled clothing, asynchrony renders noninterpenetration and momentum conservation tractable. Because elements advance at their own pace, those not entangled in collisions can proceed at large time steps. As shown in Figure 2, the median time step of an asynchronous method can be moderate even when high-impact collisions force some elements to proceed at small time steps.

1.4. Asynchronous integration

As a point of departure we consider *asynchronous variational integrators* (AVIs),[16] which belong to a larger class of integrators that exactly conserve both momentum and symplecticity (loosely related to preservation of areas in phase space); such integrators are highly regarded because of their provable approximate conservation of energy over long spans of simulated time. However, a correct contact model remains unexplored.

Figure 2. Asynchrony in the curtain simulation, depicted by the time-evolving distribution of vertex time step sizes, enables adaptive allocation of computational resources in space time.



1.5. Asynchronous collision detection

To ensure safety, we require an equally principled approach to collision detection. With every object able to collide with any other object, collision detection is fundamentally a quadratic problem. Thus, efficient collision detection algorithms are necessary to prune the non-intersecting pairs. Furthermore, we must reliably find those elements which are proximate rather than actually intersecting, so that we may counteract the impending penetration. This is a heavily studied problem; alas, the many reported successes are specific to the synchronous context, and as a group current methods can be intractably slow if naïvely applied after each local asynchronous step. This motivates our interest in *kinetic data structures* (KDSs)[3]: a KDS algorithm maintains a data structure governed by formal invariants describing some discrete attribute (such as absence of collisions), in response to the continuous movement of geometric elements. Many existing collision detection methods can be reformulated from a KDS perspective. KDSs seem destined for asynchronous applications, because their focus on fast, minimal, "output-sensitive" data-structure updates makes them ideally suited for the small, local changes effected by each AVI step.

These observations motivate our interest in approaching contact mechanics for both graphics and mechanics applications from a new direction. In particular, (a) we formulate a contact model that is *safe* independent of user parameters, such as the stiffness and "bounciness" of collisions. (b) We *correctly* discretize time, using asynchrony to preserve the model's safety and to respect causality, and using a symplectic-momentum integrator to exactly conserve momentum and approximately conserve energy over long runtimes. Finally, (c) we lay out the basic foundations for the union of AVIs with KDSs, making the safe, correct integration of complex contact for highly deformable objects tractable.

2. ASYNCHRONOUS INTEGRATORS

Consider a physical system with a time-varying configuration $\mathbf{q}(t)$ in the space $\mathbf{Q}$ of all configurations; concretely, for a mesh with vertices $\mathbf{x}_1, \ldots, \mathbf{x}_n$ in 3D we represent $\mathbf{Q} = \mathbf{R}^{3n}$ by a vector of all the vertices' Cartesian coordinates. We use a dot to denote differentiation in time, so that $\dot{\mathbf{q}}(t)$ is the velocity of the system. Let M be the mass matrix, so that $\mathbf{p} = M\dot{\mathbf{q}}$ is the momentum. The Störmer–Verlet ("leapfrog") integrator evolves a sequence of positions $\mathbf{q}_0, \mathbf{q}_1, \mathbf{q}_2, \ldots$ and momenta $\mathbf{p}^{0+\frac{1}{2}}, \mathbf{p}^{1+\frac{1}{2}}, \mathbf{p}^{2+\frac{1}{2}}, \ldots$ via the update rules

$$\mathbf{q}_{k+1} - \mathbf{q}_k = hM^{-1}\mathbf{p}^{k+\frac{1}{2}},$$
$$\mathbf{p}^{k+\frac{1}{2}} - \mathbf{p}^{k-\frac{1}{2}} = hF(\mathbf{q}_k),$$
$$t_k - t_{k-1} = h,$$

where h is the time step and $F(\mathbf{q})$ is the force. The sub/superscripted indices remind us that positions and velocities are staggered in time, with t_k associated to $\mathbf{q}_k$, and (t_k, t_{k+1}) associated to $\mathbf{p}^{k+\frac{1}{2}}$. In effect, leapfrog first updates the position at t_k using the constant momentum associated to the preceding interval (t_{k-1}, t_k), and then impulsively "kicks," obtaining

a new momentum for the following interval (t_k, t_{k+1}), yielding a piecewise linear (p.l.) trajectory over the intervals (t_k, t_{k+1}) (Figure 3). Being a *geometric integrator*,[14] leapfrog tracks conservation laws (e.g., mass, momentum, energy) and adiabatic invariants (e.g., temperature) over long runtimes, and offers more consistency and qualitatively predictable behavior across a range of time step sizes.

AVIs naturally extend leapfrog. Each force receives an independent, regular (fixed-rate) clock, fixed a priori by stability requirements. While impulses of a force are regularly spaced in time, the superposition of forces yields events irregular in time. As with leapfrog, the trajectory is p.l., interrupted by "kicks." When their clocks are nested—as quarter notes are nested in half notes—AVIs reduce to an instance of multistepping methods[14]; our developments apply to this family of methods.

For example, Lew et al.[16] assign an elastic potential to each mesh element. Irregular meshes have spatially varying element shapes and corresponding time step stability restrictions; with AVIs each element advances at its own pace. Since an elemental potential depends only on a local mesh neighborhood, each integration event is *local*, affecting the position and velocity of a small number of *stencil* vertices.

To schedule the interrupts to the p.l. trajectory, AVIs use a priority queue, conceptually populated with all event times until eternity. In practice it suffices to schedule only the next tick for each clock, since that event can schedule the subsequent tock.

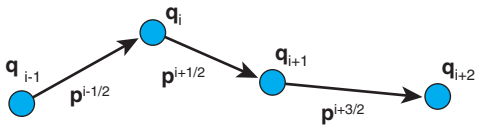
2.1. Ensuring correctness

A more complete analysis leading to the geometric and conservation properties of AVIs invokes ideas from discrete mechanics and variational integration.[16, 17] Here we stress a key outcome: Lew et al. conjecture that AVIs' remarkable properties are due to its *multisymplecticity*; the derivation requires each force to have a regular (constant-rate, ever-ticking) clock. Playing with this clock—accelerating or pausing—is strictly forbidden. Interrupting the p.l. trajectory with other mechanisms (e.g., interleaving a velocity filter) breaks multisymplecticity.

2.2. AVIs and contact

The conservation properties of AVIs rely on preservation of the multisymplectic form[17, 18] and are easily broken by naïvely incorporating existing contact-resolution methods. A principled treatment must consider a multisymplectic formulation of contact mechanics and an asynchronous computation of collision detection and response.

Figure 3. A piecewise linear trajectory where mid-step momenta $\mathbf{p}^{i+\frac{1}{2}}$ carry positions from $\mathbf{q}_i$ to $\mathbf{q}_{i+1}$.



3. DISCRETE PENALTY LAYERS

As a contact model, consider a simple penalty method that penalizes proximity between bodies. We will represent this penalty as a linear half-spring, which only counteracts compression from its designated rest length. Elongation is ignored, allowing separating bodies to move away freely.

For a given surface thickness η, the gap function

$$g_\eta(\mathbf{q}) = \|\mathbf{x}_b - \mathbf{x}_a\| - \eta$$

tracks signed proximity between moving points $\mathbf{x}_a$ and $\mathbf{x}_b$. When $g < 0$, the points are said to be *proximate*. We can express the penalty (half-spring) potential and force in terms of g

$$V_\eta^r(g(\mathbf{q})) = \begin{cases} \frac{1}{2} r g^2 & \text{if } g \le 0, \\ 0 & \text{if } g > 0, \end{cases} \qquad \mathbf{F} = \begin{cases} -rg\nabla g & \text{if } g \le 0, \\ 0 & \text{if } g > 0, \end{cases}$$

respectively, where r is the contact *stiffness*. Choosing a penalty stiffness is the most criticized problem of the penalty method.[1] For any fixed stiffness r, there exists a sufficiently large approach velocity such that the contact potential will be overcome by the momentum, allowing the configuration to tunnel illegally into a penetrating state.

The *barrier method* replaces the above contact potential by a function that grows unbounded as the configuration nears the boundary $g(\mathbf{q}) = 0$, eliminating the possibility of tunneling. However, such a function must also have unbounded second derivative, ruling out stable fixed-step time integration for *any* choice of step size.[14]

To alleviate these concerns, we propose a construction consisting of an infinite family of *nested potentials*

$$V_{\eta(l)}^{r(l)}, \qquad l = 1, 2, \ldots,$$

where $\eta(l)$ is a monotonically decreasing proximity (or "thickness") for the lth potential, and $r(l)$ is a monotonically increasing penalty stiffness. For these nested potentials to be a barrier, the cumulative energy of these potentials must diverge as the distance between two primitives vanishes:

$$\sum_l r(l)\eta(l)^2 \to \infty.$$

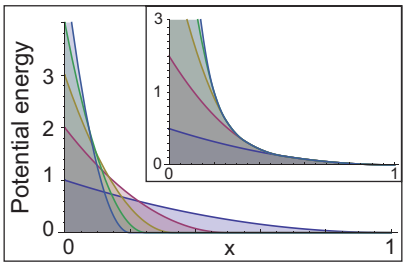
We use $r(l) = r(1)l^3$ and $\eta(l) = \eta(1)l^{-1/4}$, where $r(1)$ and $\eta(1)$ are a simulation-dependent base stiffness and thickness for the outermost layer.

We call the region $\eta(n+1) \le g(\mathbf{q}) \le \eta(n)$, where exactly n of the potentials are nonzero, the nth *discrete penalty layer* (see Figure 4).

The nested potentials' respective maximal stable time steps form a decaying sequence, and therefore this construction *requires* an adaptive or asynchronous time stepping algorithm. Each interaction potential has its own integration clock and has the opportunity to apply an impulsive change in trajectory when its clock ticks. The question is how to time step such an infinite sequence.

As we are about to see, the above construction transforms a seemingly intractable problem in Computational Mechanics—establishing a multisymplectic treatment of

Figure 4. *Discrete penalty layers*. Potential energy of layer *n* plotted against proximity. *Inset:* total potential energy contributed by all layers ″ *n*. The potential energy diverges as x_a approaches x_b, guaranteeing that constraint enforcement is robust.



contact mechanics with *guaranteed* absence of tunneling—into a challenging but addressable problem in Computer Science: efficient bookkeeping on a conceptually infinite set of interaction potentials.

3.1. Central observation

During any time interval, while conceptually the (infinite number of) clocks continues to tick, and the totality of the clock ticks is dense in time, only a *finite*, *sparse* set of clock ticks apply (nonzero) impulses. In particular, the index (l) of the discrete penalty layer (DPL) indicates the number of *active* potentials; the rest, while conceptually present, do not influence the trajectory and can be culled without approximation (Figure 5). What is needed is efficient bookkeeping to track which interaction potentials are active; each status change corresponds to a transition between penalty layers—a *discrete* change in state due to motion along a *continuous* trajectory. This is a problem that KDSs were born to solve.

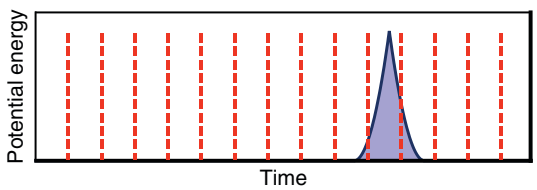
4. KINETIC DATA STRUCTURES

Guibas[12] gives an overview of kinetic data structures. Our culling of inactive forces uses an implementation of kinetic separating slabs for tracking proximity between primitives, closely related to those used by Guibas et al.[13] in the context of rigid polytopes.

4.1. Kinetic separating slabs

Proximity for triangle meshes can be written in terms of distances between vertex-triangle and edge–edge pairs. Thus, our algorithm tracks proximity between these primitives using *certificates*, a concept from the kinetic data structures literature. A certificate is a declaration of some invariant, in this case, that two primitives are separated by at least $\eta(l)$ for a penalty layer l.

Figure 5. Force evaluations (dashed lines) must be evenly spaced in time, yet only those where the potential is nonzero (blue region) must be explicitly evaluated.



To maintain the data structure, we must compute a *certificate failure*, which is the time, given current configuration and velocity, that a certificate ceases to be valid. Computing the time at which two primitives with piece-wise linear motion enter within some fixed proximity requires finding the roots of a degree-six polynomial. This is too expensive for our application, so we use the observation that a certificate failure time needs only to be conservative, not exact.

In this light, we introduce a kinetic separation slab, which we define as a plane in 3-space with constant velocity extruded by $\eta(l)$. Then, for each vertex $\mathbf{q}_i$ in the primitive pair, we can compute the time at which it enters this slab,

$$(\mathbf{q}_i \cdot \hat{\mathbf{n}} - \eta(l))/\mathbf{v},$$

where $\hat{\mathbf{n}}$ is the normal of the separating plane and $\mathbf{v}$ is the assigned constant velocity (we use the relative velocity between the closest point of the two primitives). The earliest of these times is selected as the certificate failure event time.

Because this time is conservative, at the time of a certificate failure event we must check that the primitives are indeed within proximity before creating an appropriate penalty layer event. See Section 5.1 for a walkthrough of the complete algorithm.

4.2. Broad phase

Our implementation begins with the simple separating slab KDS described above. We consider this the "narrow phase" of collision detection, the low-level processing required to track intersections between geometric elements.

While formally correct, the simple KDS used on its own will not scale efficiently to large scenes. Various sophisticated KDSs track proximity, offer better "broad-phase" scaling, and could be easily adapted to the bookkeeping of the DPL index.[6, 9, 11]

One common broad-phase algorithm in traditional (synchronous) simulations are bounding volume hierarchies (BVH).[7] For our implementation, we adopt the kinetic BVH described by Weller and Zachmann,[22] extending their axis-aligned bounding box based method to use k-Discrete Oriented Polytopes, or k-DOPs, which in general provide tighter bounds. For implementation and optimization details, we refer the reader to the full-length publication.

5. ALGORITHM

Kinetic data structures have existed for some time, but this is the first time they have been integrated with AVIs, despite their similar implementations. In this section, we walk the reader through a simple setup to reveal the logic of our algorithm. For simplicity of exposition, we will forego the existence of a k-DOP hierarchy and assume separating slabs are responsible for all proximity detection.

5.1. Walkthrough

Consider a single particle falling toward a fixed floor (Figure 6). Conceptually, the clock for the first penalty layer is always ticking; however, it is active (exerting a nonzero impulse)

only when the particle drops below height $\eta(1)$, say at time t. We must "activate the clock," no later than time t. Activating too late introduces error (misses impulses), while activating too early is correct, albeit overly conservative (some null events are not culled). The separation slab KDS is responsible for this activation.

We initialize a priority queue of events, sorted by time. Initially, this queue contains a gravity event with time step $\mathbf{g}$ (in general, internal forces will be added as well) and one certificate failure event representing the separation slab between the particle and floor. Simulation progresses by repeatedly popping events off the queue and processing them (updating velocities or rescheduling certificates). When a force event modifies velocities, all certificates which depend on that velocity must be rescheduled.

Initially, the particle's velocity is zero and the gravity event is at the front of the queue (Figure 6a). When processed, the particle is given some velocity downward (Figure 6b). The certificate must be rescheduled for the time the particle enters the separation slab, say time t_c.

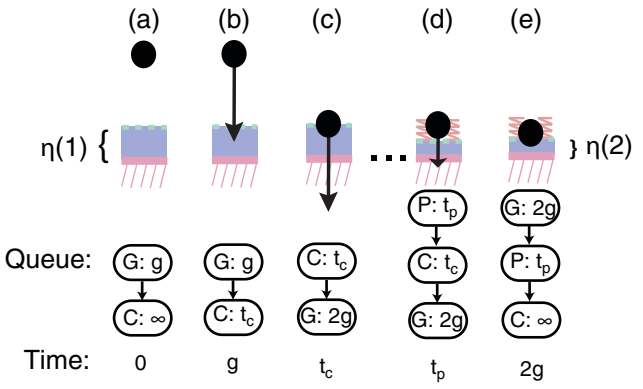
At time t_c the certificate event is popped off the queue (Figure 6c). We see that the particle is within proximity of the floor and add a layer 1 penalty event to the queue. The queue now contains two force events: gravity and a penalty layer 1 event. In general, penalty events are far more frequent than gravity events.

With the creation of a penalty layer 1 event, the certificate event switches to tracking $\eta(2)$ proximity. However, the penalty layer 1 event is still counteracting this motion to reduce further penetration (Figure 6d).

The simulation progresses with penalty force events and gravity events applying impulses in opposite directions. Eventually one of two things will happen: either the particle will enter layer 2 proximity and a second, stiffer penalty force will aid the first in counteracting gravity, or the layer 1 event's force will balance the downward force of gravity. In our illustration, the layer 1 force reaches equilibrium with gravity, a state called *resting contact* (Figure 6e).

For elastic contact where the interacting elements separate, we will need to deactivate penalty forces. The penalty layer force event serves as an opportunity to check whether the particle is transitioning to a shallower penalty layer: if (a) the penalty impulse is null, i.e., separation distance exceeds $\eta(l)$, and (b) the relative velocity is separating rather than approaching, then we deactivate the penalty force, transitioning to the next-shallower layer, and adjusting the certificates accordingly. This lazy approach to deactivation is safe by clause (a) alone; clause (b) aids in efficiency, avoiding rapid toggling of penalty layers.

Figure 6. This didactic example shows a particle falling towards a fixed floor. The queue shows the processing order of events as gravity pulls the particle downward (a)–(b), the certificate creates penalty forces (c), and the penalty force counteracts the penetration (d)–(e).



5.2. Stencils, supports, and scheduling dependencies

Consider the execution of an event at its scheduled time. The set of vertices whose velocities are altered by this event is the *stencil* of the event. The set of vertices whose trajectory was used to schedule this time is the *support* of that event. Building on the notions of stencil and support, an event *depends*, or is *contingent*, on another event if the support of the former overlaps the stencil of the latter; vice versa, an event *supports* another if the stencil of the former overlaps the support of the latter. Table 1 shows the support and stencils for a set of typical events.

KDSs were previously applied only to synchronous simulations, where the velocities of all primitives are updated at the same instant, i.e., the stencil of *the* force-integration event contains the set of *all* vertices. By contrast, in an AVI simulation, force-integration events typically bear small stencils.

Having executed a supporting event, we must reschedule all dependent events before proceeding. This is a problem of executing partially ordered instructions with dependencies, and it is thoroughly studied in the computer systems literature.[15]

Our implementation maintains a directed graph, where edges from events to vertices and vice versa denote stencil and support relations, respectively. When an event executes, the two-neighborhood of outgoing edges yields the set of events to reschedule. The graph abstraction reveals that events with large stencils, such as gravity, should cache a list of contingent events, while events with small stencils should construct the list of contingent events on-the-fly; refer to Figure 7a and b, respectively.

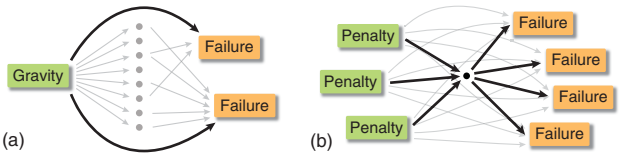
6. RESULTS

We turn our attention to challenging problems involving complex contact geometries, sharp features, and sliding during extremely tight contact.

Table 1. Events and their associated supports and stencils

Event	Supporting vertices	Stencil vertices
Gravity		Entire mesh
Stretching force[16]		Triangle
Bending force[10]		Hinge
Penalty force (Section 3)		Pair of primitives
Separation slab (Section 5.1)	Pair of primitives	
k-DOP overlap (Section 4.2)	Those in *k*-DOP	
Render frame		

Figure 7. Directed graphs depicting events (boxes), vertices (dots), and dependencies (directed edges). Integration events (left green boxes) alter vertex trajectories, forcing rescheduling of dependent events due to certificate failure (right orange boxes). (a) If an integration event has a large stencil, we store event–event dependencies. (b) If a vertex belongs to multiple stencil and support relations, we store event–vertex–event dependencies.



6.1. Knots

We simulate the tying of a ribbon into a reef knot (see Figures 8 and 9). The ribbon is modeled as a loose knot, assigned a material with stiff stretching and weak bending, and the ends are pulled by a prescribed force. The final configuration is faithful to the shape of actual "boyscout manual" knots.

This example demonstrates the strength of asynchrony in allocating resources to loci of tight contact. As the knot tightens, progressively finer time steps are used for the tightest areas of contact. If instead of prescribing reasonable forces we directly prescribe an outward motion of the two ends of the ribbon, the simulations execute to the point where the mesh resolution becomes the limiting reagent, i.e., a tighter knot cannot be tied without splitting triangles; past this point, the computation slows as penalty interactions burrow to deeper layers and the mean time step decays. This highlights both a feature and a potential artistic objection to the method: when presented with an impossible or nearly impossible situation (nonstretchy ribbon with prescribed diametrically opposing displacements at its ends) the method's safety guarantee induces Zeno's Paradox (Figure 9).

6.2. Trash compactor

We place triangle meshes of varying complexity into a virtual trash compactor consisting of a floor and four walls, and then prescribe the inward motion of opposing walls (see Figure 10 and incident image). The method is able to simulate the approach of the walls without ever allowing for seen or unseen penetrations. As with the knots, the overall rate of progress decays as the simulation approaches a limiting configuration.

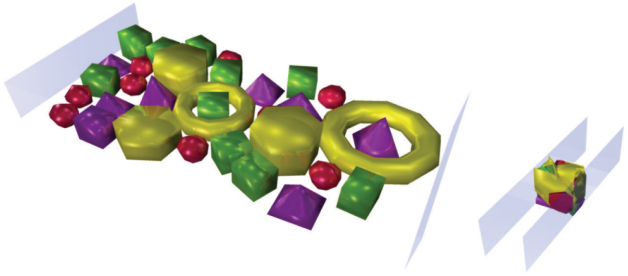
6.3. Bed of nails

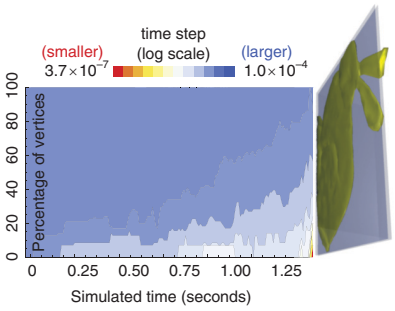
We crafted a problem to test the handling of isolated point contacts and sharp boundaries. Four sliver triangles are assembled into a nail, and many such nails are placed point-up on a flat bed. We drape two stacked fabrics over the bed of nails (see Figure 11), and observe that the simulated trajectory is both realistic and free of penetrations, oscillations, or any other artifacts typically associated to contact discontinuities. Next, we prescribe the motion of one end of the fabric, tugging on the draped configuration to demonstrate sliding over sharp features. We extend the bed of nails into a landing pad for various coarsely meshed projectiles. Variably sized to barely fit or not fit between the nails, and thrown with different initial velocities and angles, the projectiles exhibit a wide array of behaviors, including bouncing, rolling, simple stacking, ricocheting at high frequencies (this requires resolving each collision when it occurs, as resolving collisions over a fixed collision step size can cause aliasing that prevents the ricochet); sliding and getting stuck between nails (the sliding requires a deformable model and friction, since a perfectly rigid object would be constrained to a sudden stop by the distance between nails).

Figure 8. Simulated tying of ribbons into a reef knot.

Figure 9. A closeup of the reef knot.

Figure 10. Virtual trash compactor and assorted virtual trash.





6.4. Timing

We list computation time for the various examples, as executed on a single thread of a 3.06 GHz Intel Xeon with 4GB RAM. The bulk is allocated to the maintenance of the kinetic data structures used for collision detection.

As a more detailed study, consider that the reef knot simulation required 4.8% of total simulation time for integration of elastic forces and gravity, 0.09% for integration of penalty forces, 0.9% for processing and 1.0% for rescheduling

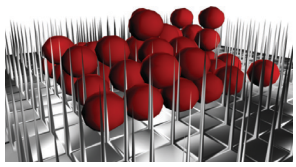
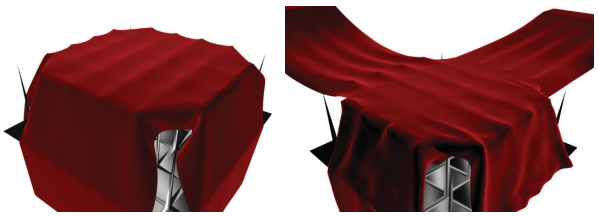


Figure 11. Experiments with a bed of nails highlight the method's ability to deal with sharp boundaries, isolated points of contact, sliver triangles, and localized points of high pressure between two nearly incident surfaces.



of separating plane events, respectively, 5.2% and 23.0% for processing and rescheduling of *k*-DOP events, respectively. The incident figure demonstrates how per frame runtime increases as the stress on the ribbons elevates.

6.5. Parameters

We list parameters for the various examples. Bending and stretching stiffness refers to the Discrete Shells[10] and common edge spring models. COR refers to coefficient of restitution, or the "bounciness" of the collisions.

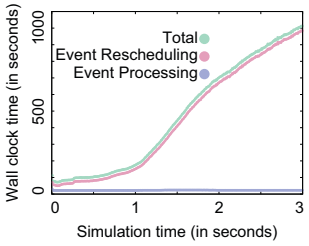
7. DISCUSSION

7.1. Parameters and the triad of safety, correctness, and progress

One of our driving goals is to investigate methods that ensure safety, correctness, and progress regardless of the choice of parameters. The method proposed here does expose some parameters to the user, such as the proximity η. These parameters affect performance, not the triad of guarantees. Our experience in running the problem

Examples	Verticles	Simulation seconds	Event processing (hours)	KDS event rescheduling (hours)	Total (hours)
Reef knot	10,642	2.00	1.5	16.7	18.5
Bowline knot	3,995	5.00	3.0	141.1	144.5
Trash compactor	714	3.08	0.5	53.0	53.6
Two sheets draped	15,982	3.95	4.5	260.8	265.5
Two sheets pulled	15,982	3.83	13.6	310.5	325.6

scenarios, therefore, was qualitatively different than when using other methods, in that we did not need to search for parameters to ensure a successful modeling of contact. On the other hand, our method does not address the spatial discretization of elasticity (stretching and bending models), which can also require user tuning.



Although in theory the nested penalty barrier has infinitely many penalty layers at its disposal, it is impractical to activate penalty layers whose stable time steps are too small, e.g., below the floating point epsilon. Simulations with thicknesses $\eta(1)$ too small, or velocities or masses too high, can thus fail to make progress (but remain safe). This limitation can be worked around by choosing a slow-shrinking layer distribution function, which is why we recommend $\eta(l) = \eta(l)l^{-1/4}$.

Example	Density	COR	r(1)	$\eta(1)$	Stretching stiffness	Stretching damping	Bending stiffness
Reef knot	0.1	0.0	1000.0	0.1	750.0	0.1	0.01
Bowline knot	0.01	0.0	1000.0	0.1	100.0	0.1	0.01
Bunny compactor	0.01	0.01	10000.0	0.05	1000.0	0.0	1000.0
Trash compactor	0.001	0.01	1000.0	0.05	1000.0	15.0	10.0
Two sheets draped	0.001	0.0	1000.0	0.1	1000.0	1.0	0.1
Reef knot untied	0.1	0.0	1000.0	0.1	1000.0	0.1	0.01
Two sheets pulled	0.001	0.0	1000.0	0.1	1000.0	1.0	0.1
Balls on nails	0.016	0.3	10000.0	0.1	50000.0	1.0	100000.0
2D sludge	–	0.0	1000.0	0.1	–	–	–

Multistepping methods such as AVIs are known to have *resonance instabilities*,[8, 14] particularly if the simulation contains adjacent mesh elements of very different size. However, we have not observed any such instabilities or artifacts that we can attribute to such instabilities in our use of the method.

7.2. Broader exploration

In this paper we were concerned with building the most robust contact implementation we could; therefore, we tied the knots as tight as possible, until each triangle was packed as tightly as possible into its neighbors. In the tightest configurations the spatial discretization becomes evident. It would therefore be interesting to introduce spatial adaptation, refining the mesh where curvature is high. Another alternative would be to improve the smoothness at render time, using for example the collision-aware subdivision of Bridson et al.[4]

Dissipation and friction are critical for expressing the widest possible range of scenarios in physical simulation. We have omitted their discussion in this extended abstract, but refer the reader to the original publication for simple models that fit this criteria. Nevertheless, future work might explore efficient algorithms to handle stacking and static friction while still fitting the multisymplectic treatment.

7.3. Immediate and future impact

In considering this method for immediate industrial use, we anticipate two important hurdles. From the standpoint of incorporation into animation systems the first hurdle is the method's insistence on safety even at the cost of artistic freedom. This effectively disallows all pinching,[2, 21] as well as commencing from invalid configurations. We believe that the method can be extended to permit shallow ("skimming") pinching, but handling extremely unphysical boundary conditions within this framework seems at least initially at odds

with the basic premise, and will require further research.

Second, the proposed method is not competitive in performance compared to existing methods, which do not attempt to make strong safety and correctness guarantees; if an artist is willing to search for parameters that provide non-penetrating good-looking results, they may become impatient with the method proposed here.

From the standpoint of long-term, curiosity-driven research, however, this method is appealing not just in its formalism but also in terms of performance, since it lays out a formal asynchronous framework from which one can investigate parallelization, optimization, and even approximation techniques that preserve guarantees of safety, correctness, and progress. To aid such future investigation, source code for our initial C++ implementation, along with data files needed to generate the examples shown in this paper, is available online [a]. C

References

1. Baraff, D. Analytical methods for dynamic simulation of non-penetrating rigid bodies. In *SIGGRAPH'89: Proceedings of the 16th Annual Conference on Computer Graphics and Interactive Techniques* (1989), ACM, New York, NY, USA, 223–232.
2. Baraff, D., Witkin, A., Kass, M. Untangling cloth. *ACM Trans. Graph. 22*(3) (2003), 862–870.
3. Basch, J., Guibas, L. J., Hershberger, J. Data structures for mobile data. *J. Algorithms 31* (1999), 1–28.
4. Bridson, R., Fedkiw, R., Anderson, J. Robust treatment of collisions, contact and friction for cloth animation. In *SIGGRAPH'02* (2002), 594–603.
5. Cirak, F., West, M. Decomposition-based contact response (DCR) for explicit finite element dynamics. *Int. J. Numer. Methods Eng., 64*(8) (2005), 1078–1110.
6. Erickson, J., Guibas, L. J., Stolfi, J., Zhang, L. Separation-sensitive collision detection for convex objects. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms* (1999), 102–111.
7. Ericson, C. *Real-Time Collision Detection (The Morgan Kaufmann Series in Interactive 3D Technology)*. Morgan Kaufmann, December 2004.
8. Fong, W., Darve, E., Lew, A. Stability of asynchronous variational integrators. *J. Comput. Phys., 227*(18) (2008), 8367–8394.
9. Gao, J., Guibas, L. Hershberger, J., Zhang, L., Zhu, A. Discrete mobile centers. *Discrete Comput. Geom. 30*(1):45–65, 2003.
10. Grinspun, E., Hirani, A., Desbrun, M., Schröder, P. Discrete shells. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (August 2003), 62–67.
11. Guibas, L., Xie, F., Zhang, L. Kinetic collision detection: Algorithms and experiments. In *Proceedings of the International Conference on Robotics and Automation* (2001), 2903–2910.
12. Guibas, L. J. Kinetic data structures—a state of the art report. In *Proceedings of the 3rd Workshop on Algorithmic Foundations of Robotics (WAFR)* (1998), 191–209.
13. Guibas, L. J., Xie, F., Zhang, L. Kinetic collision detection: Algorithms and experiments. In *ICRA* (2001), 2903–2910. http://www.cs.columbia.edu/cg/ACM/
14. Hairer, E., Lubich, C., Wanner, G. *Geometric Numerical Integration: Structure-Preserving Algorithms for Ordinary Differential Equations*. Springer, 2002.
15. Korneev, V., Kiselev, A. *Modern Microprocessors*. Charles River Media, 2004.
16. Lew, A., Marsden, J. E., Ortiz, M., West, M. Asynchronous variational integrators. *Arch. Rational Mech. Anal. 167* (2003), 85–146.
17. Marsden, J., Patrick, G., Shkoller, S. Multisymplectic geometry, variational integrators, and nonlinear PDEs. *Commun. Math. Phys. 199*(2) (1998), 351–395.
18. Marsden, J., Pekarsky, S., Shkoller, S., West, M. Variational methods, multisymplectic geometry and continuum mechanics. *J. Geom. Phys. 38*(3–4) (June 2001), 253–284.
19. Milenkovic, V. J., Schmidl, H. Optimization-based animation. In *SIGGRAPH'01: Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (2001), ACM, New York, NY, USA, 37–46.
20. Provot, X. Collision and self-collision handling in cloth model dedicated to design garments. In *Computer Animation and Simulation '97* (1997), Springer Verlag, Wien, 177–189.
21. Volino, P., Magnenat-Thalmann, N. Resolving surface collisions through intersection contour minimization. In *SIGGRAPH'06: ACM SIGGRAPH 2006 Papers* (2006), ACM, New York, NY, USA,1154–1159.
22. Weller, R. and Zachmann, G. Kinetic separation lists for continuous collision detection of deformable objects. In *Third Workshop in Virtual Reality Interactions and Physical Simulation (Vriphys)* (Madrid, Spain, 6–7 November 2006).

[a] http://www.cs.columbia.edu/cg/ACM

David Harmon Columbia University, New York, NY.

Etienne Vouga Columbia University, New York, NY.

Breannan Smith Columbia University, New York, NY.

Rasmus Tamstorf Walt Disney Animation Studios, Burbank, CA.

Eitan Grinspun Columbia University, New York, NY.