Velocity-Based Shock Propagation for Multibody Dynamics Animation

KENNY ERLEBEN
University of Copenhagen

Multibody dynamics are used in interactive and real-time applications, ranging from computer games to virtual prototyping, and engineering. All these areas strive towards faster and larger scale simulations. Particularly challenging are large-scale simulations with highly organized and structured stacking. We present a stable, robust, and versatile method for multibody dynamics simulation. Novel contributions include a new, explicit, fixed time-stepping scheme for velocity-based complementarity formulations using shock propagation with a simple reliable implementation strategy for an iterative complementarity problem solver specifically optimized for multibody dynamics.

Categories and Subject Descriptors: I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling—*Physically based modeling*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*

General Terms: Algorithms

Additional Key Words and Phrases: Multibody dynamics, constraint-based simulation, complementarity formulations, shock-propagation, stacking

ACM Reference Format:

Erleben, K. 2007. Velocity-based shock propagation for multibody dynamics animation. ACM Trans. Graph. 26, 2, Article 12 (June 2007), 20 pages. DOI = 10.1145/1243980.1243986 http://doi.acm.org/10.1145/1243980.1243986

1. MULTIBODY DYNAMICS IN COMPUTER GRAPHICS

A long-term goal in computer graphics is to increase realism and believability in computer generated animations and pictures. The general belief is that, as we get better and better at rendering images, the lack of physical realism and believability will be increasingly obvious and therefore increasingly annoying to the common observer. The main argument for achieving the goal of more realism has been to use physics to model the behavior and movement of computer models. Today these efforts have culminated in what is usually referred to as physics-based animation. Over the past decade physics-based animation has matured, and today there is a wealth of simulation methods solving many simulation problems. There are a vast amount of examples where physics-based animation is used, for example, rigid bodies stumbling around (The Hulk, Grand Turismo, Medal of Honor, Half-Life); skin and muscle deformations (Shrek, the Nutty Professor, Jurassic Park, the Mummy); water splashing (Shrek, Titanic, Finding Nemo); jelly blobs dancing around (Flopper); death-like animations (Hitman); hair blowing in the wind or bending due to motion of a character (Monsters Inc); cloth moving (Monsters Inc); and melting robots and cyborg parts of characters (Terminator 3, Treasure Island), just to mention a few.

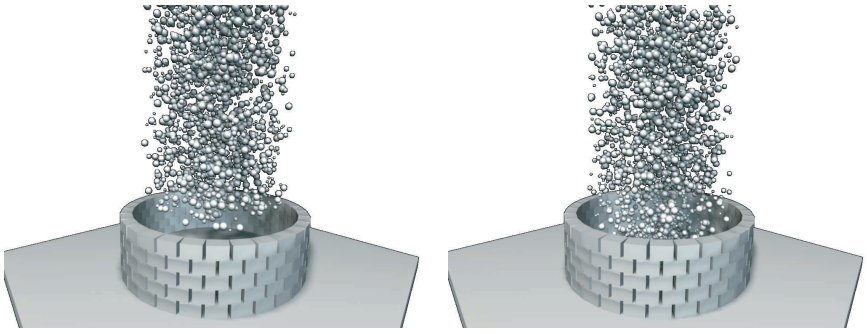
While it was a computationally heavy burden 10–20 years ago to kinetically animate a linked character consisting of no more than a handful of limbs, today this is considered a trivial task due to the large increase in computer power. The increase in computer power allows us to simulate increasingly complex scenarios in an apparently never-ending spiral, and it appears that there will always be a demand for faster methods with more details, larger scenes, and so on.

Current state-of-the-art middleware used in computer games run at real-time interactive rates but do not deliver the same motion quality as our method. The methods published in the computer graphics literature is not interactive but delivers better motion quality than computer game middleware. In this article, we present a method for computing high quality plausible motion of several hundreds rigid bodies at interactive rates.

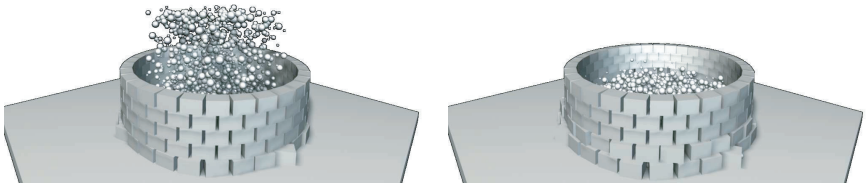
Multibody dynamics was introduced to the graphics community in the late 80's [Hahn 1988; Moore and Wilhelms 1988] using penalty-based and impulse-based approaches to describe the physical interactions. Penalty-based simulation lacks the ability to be easily adopted to different simulations without parameter-tuning. They are, therefore, not a good choice for versatile multibody dynamics animation. The impulse-based approach was extended and improved [Mirtich 1996] but suffered from creeping and stacking problems until recently when these problems were rectified [Guendelman et al. 2003]. Constraint-based simulation has received much attention as an alternative to the impulse-based approach [Baraff 1989, 1994]. With the new time-integration method and shock propagation presented in Guendelman et al. [2003] impulse-based simulation has become a serious competitor to constraint-based simulation.

Constraint-based simulation of multibody dynamics can be classified into two groups of algorithms: minimal coordinate methods,

Author's address: Department of Computer Science, University of Copenhagen, Universitetsparken 1, DK-2100, Denmark; email: kenny@diku.dk.



(a) frame 48 (b) frame 60



(c) frame 72 (d) frame 84

Fig. 1. Soli. A 3000-object simulation with a frame time of 1 second using our method. See supplementary movie.

which tend to be recursive in nature [Armstrong and Green 1985; Featherstone 1998], and maximal coordinate methods, which in computer animation are dominated by complementarity formulations [Baraff 1994]. The focus of this article is on complementarity formulations. There exist alternatives to complementarity formulations based on kinetic energy [Milenkovic and Schmidl 2001; Schmidl and Milenkovic 2004] and motion space [Redon et al. 2003]. However, the former solves a more general problem but is not attractive for performance reasons, and the latter is of limited use for realistic animation since it does not include friction. Recently Kaufman et al. [2005] presented a velocity-based method using projections onto convex subspaces of feasible velocities. The authors used a contact model, which is based on limit surfaces and maximum dissipation [Goyal et al. 1989], together with an ad-hoc model for bounciness and an approximation of momentum conservation.

Complementarity formulations come in two flavors: acceleration-based formulations [Baraff 1994, 1995; Trinkle et al. 2001] and velocity-based formulations [Stewart and Trinkle 1996]. Acceleration-based formulations cannot handle collisions, and one must stop at the point of collision and switch to a impulse-momentum law [Baraff 1989; Anitescu and Potra 1996; Pfeiffer and Wösle 1996; Chatterjee and Ruina 1998]. Further, acceleration-based formulations suffer from indeterminacy and inconsistency [Stewart 2000]. Although mostly overlooked in the computer graphics literature, the velocity-based formulation suffers from none of these drawbacks.

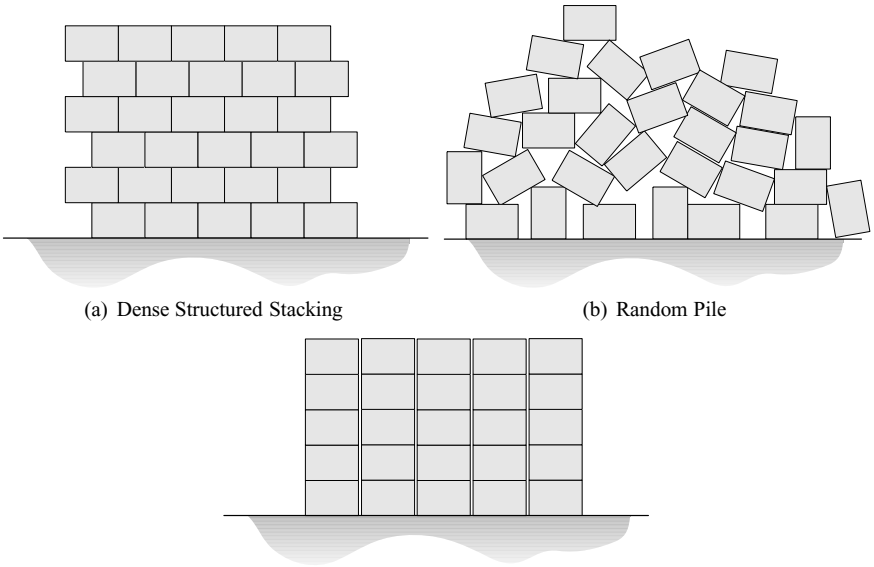
The focus in this article is on computing the dynamics for large-scale dense structured stacking at interactive frame rates. Other common configuration types are sparse structured stacking and random piles. The three types are shown in Figure 2.

In the past, sparse structured stacking has been attractive since it breaks down the all-pair dependence of the contact constraints at the design time of the configuration. This is an advantage in, for instance, computer games where game-level designers can build worlds with more predictable performance.

Recently random piles have received a lot of attention in computer graphics [Guendelman et al. 2003; Kaufman et al. 2005]. They appear very complex and difficult to simulate since the all-pair dependence makes them computationally more intractable. Nevertheless, the randomness helps hide penetration errors. In fact, as long as the topmost objects move in a plausible manner, an observer will not notice any errors deep inside the pile. Penetration errors result in plausible motion even in the case where objects inside the random pile may be observed. In fact the random pile in Figure 2 has penetration errors. We encourage the reader to find these by visual inspection.

Dense structured stacking has the same all-pair dependence as random piles. However, this type of configuration is more challenging since the nice alignment of structure is immediately destroyed if penetration errors are not kept under control. For instance, an observer would immediately notice any misalignment of any box in Figure 2(a). Further, most error correction methods tend to blow up a structured stack during the process of fixing errors in the bottom-most layers. When studying dense structured stacking, the natural object of interest is box geometry because it allows us to immediately observe misalignment and penetration errors by visual inspection, and therefore it is our choice of geometry for most test cases. The method presented in this article deals with all the problems of large-scale structured stacking with hundreds of objects as our examples demonstrate.

This article does not deal with the problem of collision detection. In fact, we use the built-in collision detection engine from OpenTissue , which only supports interference queries of primitives and signed distance maps. The focus of this article lies on solving the



(a) Dense Structured Stacking (b) Random Pile

(c) Sparse Structured Stacking

Fig. 2. The three types of stacking often encountered in computer graphics.

dynamics. Our results generalize to arbitrary polygonal shapes because the derived model holds for planar contacts. Thus using more complex geometries really only influences the narrow-phase collision detection algorithms used. Besides, for performance reasons, interactive applications rarely use polygonal objects for collision detection. These are frequently replaced with compounds of boxes, spheres etc., and polygonal models are used as in-place geometry.

Our article is organized as follows: we will derive the velocity-based complementarity formulation in Section 2. Hereafter, we present a simple iterative solver in Section 3. Section 4 extends the complementarity formulation with shock propagation, and Section 5 shows results of our velocity-based shock propagation method and compares our method to existing methods. Section 6 discusses various aspects of simulation relevant to our method.

2. VELOCITY-BASED COMPLEMENTARITY FORMULATION

The contact constraints are formulated using a Jacobian matrix defined as

$$\mathbf{J} = \left[\mathbf{J}_{\text{row}_1}^T \mathbf{J}_{\text{row}_2}^T \mathbf{J}_{\text{row}_3}^T\right]^T. \tag{1}$$

Looking at a point of contact between two rigid bodies i and j, with contact normal $\vec{n}$ and with vectors $\vec{r}_i$ and $\vec{r}_j$ from respective body centers to the point of contact, we can write the nonpenetration constraint as

$$\underbrace{\left[-\vec{n}^T \quad -(\mathbf{r_i}^\times \vec{n})^T \quad \vec{n}^T \quad (\mathbf{r_j}^\times \vec{n})^T\right]}_{\mathbf{J}_{\text{row}_1}} \underbrace{\begin{bmatrix}\vec{v}_i \\ \vec{\omega}_i \\ \vec{v}_j \\ \vec{\omega}_j\end{bmatrix}}_{\vec{u}} = \mathbf{J}_{\text{row}_1}\vec{u} \geq 0. \tag{2}$$

Here $\vec{v}_i$ and $\vec{v}_j$ are the linear velocities of the bodies, and $\vec{\omega}_i$ and $\vec{\omega}_j$ are the angular velocities. The vector $\vec{u}$ is referred to as the generalized velocity vector, and $\mathbf{r}^\times \in \mathbb{R}^{3\times 3}$ is the skew-symmetric matrix given by

$$\mathbf{r}^\times = \begin{bmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{bmatrix}, \tag{3}$$

such that $\mathbf{r}^\times \vec{n} = \vec{r} \times \vec{n}$. The principle of virtual work requires the constraint force to be orthogonal to the constraint, which means that

$$\vec{f}_1 = \mathbf{J}_{\text{row}_1}^T \lambda_1, \tag{4}$$

where λ_1 is a Lagrange multiplier which we need to solve for, and where $\vec{f}_1$ is the normal force. Note that $\vec{n}$ is assumed to be a unit vector, meaning that λ_1 is the magnitude of the normal force. Physics requires the normal force to be repulsive and zero at separation, which yields the complementarity constraint

$$\mathbf{J}_{\text{row}_1}\vec{u} \geq 0 \quad \text{compl.} \quad \lambda_1 \geq 0. \tag{5}$$

Applying a friction pyramid as in Baraff [1994] and given the unit contact tangent plane vectors $\vec{t}_1$ and $\vec{t}_2$, the tangential velocities can be written as

$$\begin{bmatrix}\mathbf{J}_{\text{row}_2} \\ \mathbf{J}_{\text{row}_3}\end{bmatrix}\vec{u} = \begin{bmatrix}\left[-\vec{t}_1^{\,T} \quad -\left(\mathbf{r_i}^\times \vec{t}_1\right)^T \quad \vec{t}_1^{\,T} \quad \left(\mathbf{r_j}^\times \vec{t}_1\right)^T\right] \\ \left[-\vec{t}_2^{\,T} \quad -\left(\mathbf{r_i}^\times \vec{t}_2\right)^T \quad \vec{t}_2^{\,T} \quad \left(\mathbf{r_j}^\times \vec{t}_2\right)^T\right]\end{bmatrix}\vec{u}. \tag{6}$$

The constraint forces, that is, the friction forces $\vec{f}_2$ and $\vec{f}_3$, can thus be written as

$$\vec{f}_2 = \mathbf{J}_{\text{row}_2}^T \lambda_2, \qquad \text{and} \qquad \vec{f}_3 = \mathbf{J}_{\text{row}_3}^T \lambda_3, \tag{7}$$

where λ_2 and λ_3 are unknown Lagrange multipliers we need to solve for. According to Coulomb's friction model, dynamic friction occurs when the tangential velocity is nonzero. At that point, the friction force attains its maximum value and a direction opposite the motion.

This means,

$$\mathbf{J}_{\text{row}_2}\vec{u} > 0 \Rightarrow \lambda_2 = -\mu\lambda_1, \tag{8a}$$
$$\mathbf{J}_{\text{row}_2}\vec{u} < 0 \Rightarrow \lambda_2 = \mu\lambda_1, \tag{8b}$$
$$\mathbf{J}_{\text{row}_2}\vec{u} = 0 \Rightarrow \lambda_2 <\mid \mu\lambda_1 \mid, \tag{8c}$$

where μ is the coefficient of friction. The last constraint is the case of static friction. Similar constraints hold for $\mathbf{J}_{\text{row}_3}\vec{u}$ and λ_3. Equations (8) are in fact a general complementarity condition called a box-constraint, whereas (5) is a corner constraint.

If we define $\lambda_{\text{lo}} = [0, (-\mu\lambda_1), (-\mu\lambda_1)]^T$ and $\lambda_{\text{hi}} = [\infty, (\mu\lambda_1), (\mu\lambda_1)]^T$, then all constraints, $i = 1$, 2, and 3, can be rewritten in the same unified notation as

$$\lambda_i = \vec{\lambda}_{\text{lo}_i} \Rightarrow (\mathbf{J}\vec{u})_i \geq 0, \tag{9a}$$
$$\lambda_i = \vec{\lambda}_{\text{hi}_i} \Rightarrow (\mathbf{J}\vec{u})_i \leq 0, \tag{9b}$$
$$\vec{\lambda}_{\text{lo}_i} < \lambda_i < \vec{\lambda}_{\text{hi}_i} \Rightarrow (\mathbf{J}\vec{u})_i = 0. \tag{9c}$$

For N bodies and a set of K contact points, the method is easily extended as follows. Let the generalized velocity vector $\vec{u} \in \mathbb{R}^{6N}$ be

$$\vec{u} = \left[\vec{v}_1, \vec{\omega}_1, \vec{v}_2, \vec{\omega}_2, \ldots, \vec{v}_N, \vec{\omega}_N\right]^T. \tag{10}$$

Define the Jacobian of the k'th contact to be

$$\mathbf{J}_k = \begin{bmatrix}\mathbf{J}^i_{\text{lin}_k} & \mathbf{J}^i_{\text{ang}_k} & \mathbf{J}^j_{\text{lin}_k} & \mathbf{J}^j_{\text{ang}_k}\end{bmatrix}, \tag{11}$$

where

$$\mathbf{J}^i_{\text{lin}_k} = \begin{bmatrix}-\vec{n}_k^T \\ -\vec{t}_{k_1}^T \\ -\vec{t}_{k_2}^T\end{bmatrix}, \qquad \mathbf{J}^j_{\text{lin}} = \begin{bmatrix}\vec{n}_k^T \\ \vec{t}_{k_1}^T \\ \vec{t}_{k_2}^T\end{bmatrix}, \tag{12a}$$

$$\mathbf{J}^i_{\text{ang}_k} = \begin{bmatrix}-\left(\mathbf{r}_{\mathbf{k_i}} \times \vec{n}_k\right)^T \\ -\left(\mathbf{r}_{\mathbf{k_i}} \times \vec{t}_{k_1}\right)^T \\ -\left(\mathbf{r}_{\mathbf{k_i}} \times \vec{t}_{k_2}\right)^T\end{bmatrix}, \qquad \mathbf{J}^j_{\text{ang}} = \begin{bmatrix}\left(\mathbf{r}_{\mathbf{k_j}} \times \vec{n}_k\right)^T \\ \left(\mathbf{r}_{\mathbf{k_j}} \times \vec{t}_{k_1}\right)^T \\ \left(\mathbf{r}_{\mathbf{k_j}} \times \vec{t}_{k_2}\right)^T\end{bmatrix}. \tag{12b}$$

Then a system Jacobian $\mathbf{J} \in \mathbb{R}^{3K \times 6N}$ is assembled by filling out a 3×3 block structure. That is, for the k'th contact point between bodies i and j, all blocks of the k'th row of $\mathbf{J}$ is set to zero except for the subblocks:

$$\mathbf{J}_{k,2i} = \mathbf{J}^i_{\text{lin}_k}, \qquad \mathbf{J}_{k,2i+1} = \mathbf{J}^i_{\text{ang}_k}, \tag{13a}$$
$$\mathbf{J}_{k,2j} = \mathbf{J}^j_{\text{lin}_k}, \qquad \mathbf{J}_{k,2j+1} = \mathbf{J}^j_{\text{ang}_k}. \tag{13b}$$

A similar setup is used for all other contact points. Notice that a sparse or compressed matrix data structure should be used for efficiency. Now define

$$\vec{w} = \mathbf{J}\vec{u}, \tag{14}$$

then the i'th constraint $i \in [1..3K]$ is given by

$$\lambda_i = \vec{\lambda}_{\text{lo}_i} \Rightarrow \vec{w}_i \geq 0, \tag{15a}$$
$$\lambda_i = \vec{\lambda}_{\text{hi}_i} \Rightarrow \vec{w}_i \leq 0, \tag{15b}$$
$$\vec{\lambda}_{\text{lo}_i} < \lambda_i < \vec{\lambda}_{\text{hi}_i} \Rightarrow \vec{w}_i = 0. \tag{15c}$$

The equations of motion can be written as,

$$\mathbf{M}\dot{\vec{u}} = \mathbf{J}^T\vec{\lambda} + \vec{f}_{\text{ext}}, \tag{16}$$

where $\mathbf{M} \in \mathbb{R}^{6N \times 6N}$ is the generalized mass matrix given by

$$\mathbf{M} = \begin{bmatrix} m_1\mathbf{1} & & & \mathbf{0} \\ & \mathbf{I}_1 & & \\ & & \vdots & \\ & & & m_N\mathbf{1} \\ \mathbf{0} & & & \mathbf{I}_N \end{bmatrix}, \tag{17}$$

with m_i as the mass of the i'th body, $\mathbf{1}$ as the identity matrix, $\mathbf{I}_i$ as the corresponding inertia tensor, and where the external and velocity-dependent forces are given by $\vec{f}_{\text{ext}} \in \mathbb{R}^{6N}$,

$$\vec{f}_{\text{ext}} = \left[\vec{f}_1^{\text{ext}}, \vec{\tau}_1^{\text{ext}} - \vec{\omega}_1 \times I_1\vec{\omega}_1, \cdots, \vec{f}_n^{\text{ext}}, \vec{\tau}_n^{\text{ext}} - \vec{\omega}_n \times I_n\vec{\omega}_n\right]^T, \tag{18}$$

with $\vec{f}_i^{\text{ext}}$ as the total linear force acting on the center of mass of body i, and $\vec{\tau}_i^{\text{ext}}$ the total external torque with respect to the center of mass of body i. Performing an explicit Euler step on (16) and isolating the next generalized velocity vector yields,

$$\vec{u}^{\,t+1} = \vec{u}^{\,t} + \mathbf{M}^{-1}\mathbf{J}^T\Delta t\vec{\lambda} + \Delta t\mathbf{M}^{-1}\vec{f}_{\text{ext}}. \tag{19}$$

This equation is known as the velocity update. Inserting it into (14) yields

$$\vec{w} = \underbrace{\mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T}_{\mathbf{A}}\Delta t\vec{\lambda} + \underbrace{\mathbf{J}(\vec{u}^{\,t} + \Delta t\mathbf{M}^{-1}\vec{f}_{\text{ext}})}_{\vec{b}} = \mathbf{A}\vec{\lambda} + \vec{b}. \tag{20}$$

In the last step we have simply moved the timestep Δt into the Lagrange multiplier vector, which means that we solve for magnitudes of the impulses rather than magnitudes of the forces. Bouncing can be added to the formulation by using Newton's Impact Law, that is, given a coefficient of restitution, $0 \leq \varepsilon_k \leq 1$, for the k'th contact point equation in (2), implies that

$$\mathbf{J}_{k,\text{row}_1}\vec{u}_k^{\,t+1} \geq -\underbrace{\varepsilon_k\mathbf{J}_{k,\text{row}_1}\vec{u}^{\,t}}_{b_k}, \tag{21}$$

which is equivalent to adding the vector $\vec{b}_{\text{bounce}} \in \mathbb{R}^{3K}$,

$$\vec{b}_{\text{bounce}} = [b_1 \;\; 0 \;\; 0 \;\; \cdots \;\; b_K \;\; 0 \;\; 0]^T, \tag{22}$$

to the $\vec{b}$-vector in (20). Equation (20) and the constraints in (15) form a complementarity problem. Solving for $\vec{\lambda}$ yields the constraint impulses needed to perform a velocity update. Defining a generalized position vector $\vec{s} \in \mathbb{R}^{7N}$ as,

$$\vec{s} = [\vec{r}_1, q_1, \vec{r}_2, q_2, \ldots, \vec{r}_N, q_N]^T, \tag{23}$$

where $\vec{r}_i$ is the center of mass position of body i, and $q_i = [s_i, x_i, y_i, z_i]^T \in \mathbb{R}^4$ is the orientation of body i represented as a quaternion, and defining

$$\mathbf{Q}_i = \frac{1}{2}\begin{bmatrix}-x_i & -y_i & -z_i \\ s_i & z_i & -y_i \\ -z_i & s_i & x_i \\ y_i & -x_i & s_i\end{bmatrix}, \tag{24}$$

and $\mathbf{S} \in \mathbb{R}^{7N \times 6N}$ as

$$\mathbf{S} = \begin{bmatrix}\mathbf{1} & & & \mathbf{0} \\ & \mathbf{Q}_1 & & \\ & & \ddots & \\ & & & \mathbf{1} \\ \mathbf{0} & & & \mathbf{Q}_N\end{bmatrix}, \tag{25}$$

then the position update can be written as,

$$\vec{s}^{\,t+\Delta t} = \vec{s}^{\,t} + \Delta t \mathbf{S}\vec{u}^{\,t+\Delta t}, \tag{26}$$

which follows from taking an explicit Euler step of $\dot{\vec{s}} = \mathbf{S}\vec{u}$. Notice that we are using the constrained velocities $\vec{u}^{\,t+\Delta t}$ in the position update as obtained from the velocity update. This adds a degree of implicitness to the explicit-time-stepping scheme, which is given by first solving for $\vec{\lambda}$, and then doing a velocity update followed by a position update.

3. ITERATIVE COMPLEMENTARITY PROBLEM SOLVER

Equations (15) and (20) result in the complementarity formulation,

$$\mathbf{A}\vec{\lambda} + \vec{b} \geq \vec{0}, \quad \text{and} \quad \vec{\lambda}_{\text{lo}} \leq \vec{\lambda} \leq \vec{\lambda}_{\text{hi}}. \tag{27}$$

In Cottle et al. [1992] splitting methods are described for solving (27) which are similar to well-known matrix solvers: Jacobi, Gauss-Seidel, and successive overrelaxation (SOR) methods, followed by a projection step to enforce the complementarity constraints. It is possible to use methods like Conjugate Gradient in a similar manner [Renouf et al. 2005] or to use an active set method [Murty 1988]. The conjugate projected gradient method has an erratic convergence rate when friction is included, but a quadratic convergence rate for the frictionless case [Renouf and Alart 2004]. Our initial test indicated that the Jacobi method had terrible convergence and that SOR suffered from energy gain. The Conjugate Gradient method has been reported to be computationally intractable due to frequent changes of the active set [Moravanszky 2004]. Thus, we were left with Gauss-Seidel as our practical choice. Gauss-Seidel solvers have previously been applied for multibody dynamics [Moreau 1999; Jean 1999] although in a blocked version.

Two other types of iterative methods are Newton methods and Interior Point methods. The theoretical convergence rate of Newton's method is quadratic, which is a clear improvement over Gauss-Seidel, although Lacoursiere [2003] reported linear convergence in experiments. The amount of work done per-iteration in a Newton method is $O(n^3)$ because a linear subsystem is solved. Thus, even if a low number of Newton steps is used, the total complexity will be $O(n^3)$. Interior Point Methods also have quadratic convergence but, like Newton methods, they, too, must solve a linear subsystem, yielding the same kind of complexity. Often 8–14 outer steps are needed to produce acceptable results for game animation with Newton Methods and Interior Point Methods. The subsystems can be solved with incomplete Cholesky preconditioned Conjugate Gradient methods, reducing the complexity per-iteration to $O(n)$ and experience indicates that on the order of 12–14 iterations is needed for acceptable results. The computational work of such approaches compare to $14 \cdot 14 \approx 200$ brute force Gauss-Seidel iterations. In this article, we present a simulation method that only requires about 30 Gauss-Seidel iterations.

For direct methods, the amount of work per-iteration is roughly on the order of $O(n^2)$ using incremental matrix factorization. In Lacoursiere [2003], it is reported that Lemke and Keller methods need n iterations, which indicates a total complexity of $O(n^3)$.

To describe the iterative Gauss-Seidel solver, we introduce the splitting $\mathbf{A} = \mathbf{L} + \mathbf{D} + \mathbf{U}$, where $\mathbf{D}$, $\mathbf{L}$, and $\mathbf{U}$ are the diagonal, the strictly lower, and strictly upper parts of $\mathbf{A}$. A single iteration of the iterative Gauss-Seidel method, equivalent to solving the matrix equation $\mathbf{A}\vec{\lambda} + \vec{b} = \vec{0}$, consists of looping over all variables $i \in [1..3K]$, and updating their values

$$\vec{\lambda}_i^{\,k+1} = \frac{\left(-\sum_{j=0}^{i-1} \mathbf{L}_{i,j}\vec{\lambda}_j^{\,k+1} - \sum_{j=i+1}^{n-1} \mathbf{U}_{i,j}\vec{\lambda}_j^{\,k} - \vec{b}_i\right)}{\mathbf{D}_{i,i}}, \tag{28}$$

where superscript is the iteration number. If the i'th variable is a friction constraint, then the upper and lower limits are updated,

$$(r = i \bmod 3) \neq 0 \quad \Rightarrow \quad \vec{\lambda}_{\text{hi}_i} = \mu\vec{\lambda}_{i-r}, \vec{\lambda}_{\text{lo}_i} = -\vec{\lambda}_{\text{hi}_i}. \tag{29}$$

Next a projection step is performed,

$$\vec{\lambda}_i^{\,k+1} = \min\left(\max\left(\vec{\lambda}_{\text{lo}_i}, \vec{\lambda}_i^{\,k+1}\right), \vec{\lambda}_{\text{hi}_i}\right). \tag{30}$$

Note that this coupling between normal and tangential directions yield a nonlinear complementarity formulation (NCP). It is possible to extend the projection to limit surfaces as done in Goyal et al. [1989]; our choice corresponds to a square limit surface.

Computing $\mathbf{A}$ and $\vec{b}$ takes linear time with respects to the number of constraints when using a sparse matrix representation for $\mathbf{M}$, $\mathbf{J}$, and $\mathbf{A}$ because each row of the Jacobian has exactly 12 nonzero elements, and $\mathbf{M}$ is a 3-by-3 block diagonal matrix. Some optimizations can be done by rewriting equation (28) as

$$\vec{\lambda}_i = \frac{-b_i - \mathbf{A}_{\text{row}_i}\vec{\lambda} + \mathbf{A}_{i,i}\vec{\lambda}_i}{\mathbf{D}_{i,i}} = \vec{\lambda}_i - \frac{b_i + \mathbf{A}_{\text{row}_i}\vec{\lambda}}{\mathbf{D}_{i,i}}. \tag{31}$$

Observing that we need $\mathbf{M}^{-1}\mathbf{J}^T\vec{\lambda}$ in the velocity update in Equation (19), we set $\Delta\vec{V} = \mathbf{M}^{-1}\mathbf{J}^T\vec{\lambda}$, such that

$$\vec{\lambda}_i = \vec{\lambda}_i - b_i' + \mathbf{J}'_{\text{row}_i}\Delta\vec{V}, \tag{32}$$

where the divisions are avoided by precomputing $\vec{b}_i' = \vec{b}_i/\mathbf{D}_{i,i}$ and $\mathbf{J}'_{\text{row}_i} = \mathbf{J}_{\text{row}_i}/\mathbf{D}_{i,i}$. After the projection step, we let $\Delta\vec{\lambda}_i = \vec{\lambda}_i^{\,k+1} - \vec{\lambda}_i^{\,k}$, and, by precomputing $\mathbf{T} = \mathbf{M}^{-1}\mathbf{J}^T$, the update of $\Delta\vec{V}$ is

$$\Delta\vec{V} = \Delta\vec{V} + \mathbf{T}_{\text{col}_i}\Delta\vec{\lambda}_i, \tag{33}$$

which can be implemented extremely efficiently, since $\mathbf{T}_{\text{col}_i}\Delta\vec{\lambda}_i$ only has 12 nonzero entries. After solving the NCP problem, the value $\Delta\vec{V}$ is returned and used directly in the velocity update.

3.1 Convergence Rate

In this section, we will present results on the iterative solver. If the iterative method is rephrased as a matrix equation, then we have

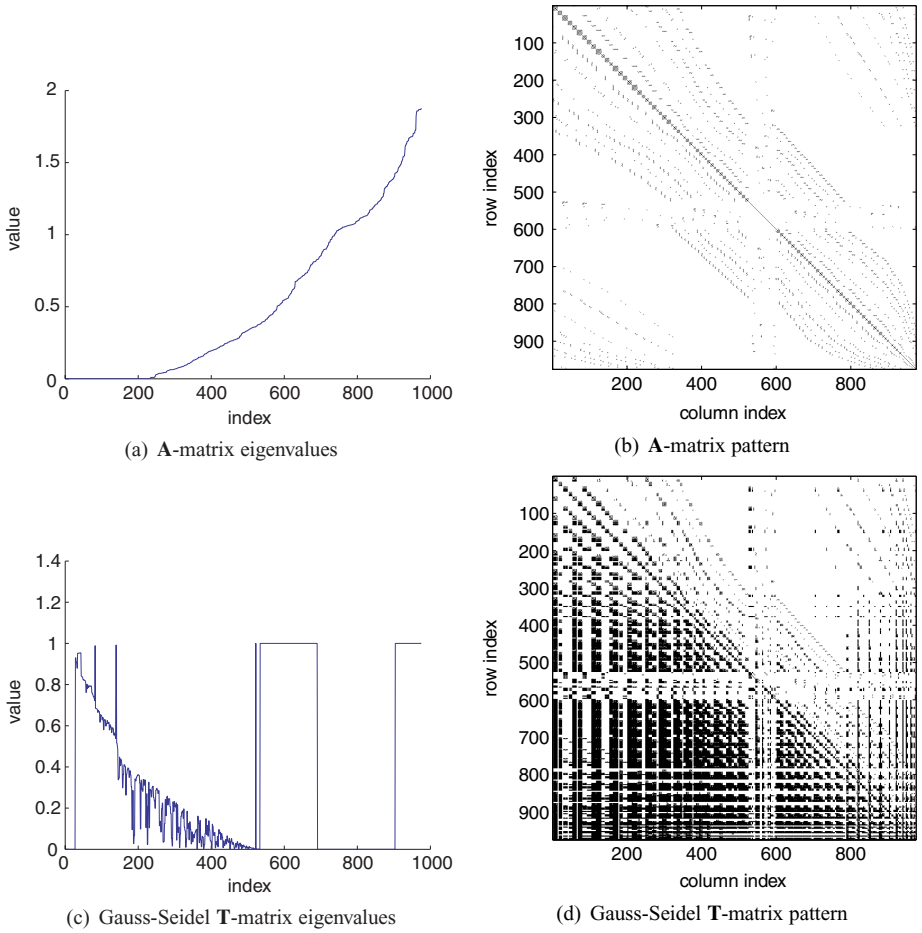
$$\vec{\lambda} = \mathbf{T}\vec{\lambda} + \vec{c}, \tag{34}$$

with

$$\mathbf{T} = (\mathbf{D} + \mathbf{L})^{-1}\mathbf{U}, \tag{35}$$

$$\vec{c} = -(\mathbf{D} + \mathbf{L})^{-1}\vec{b}. \tag{36}$$

For such a fix-point matrix equation to converge at all, the spectral radius of $\mathbf{T}$ must be less than 1. Generally speaking, the smaller the spectral radius, the faster the convergence. Large mass ratios seem to affect the magnitude of the eigenvalues of the system matrix. Thus the spectral radius of the $\mathbf{T}$-matrix is affected. Figures 3 and 4 show results of performing a matrix analysis of the system matrices $\mathbf{A}$ and the $\mathbf{T}$-matrices from a ball grid and a wall animation similar to Figure 10 and 13. From the figures, it is clear that the system matrices, $\mathbf{A}$, are extremely sparse, symmetric, and blocked. Also they have large null spaces, which is indicated from the eigenvalue plots. This means that the system is highly overdetermined. It is



(a) **A**-matrix eigenvalues

(b) **A**-matrix pattern

(c) Gauss-Seidel **T**-matrix eigenvalues

(d) Gauss-Seidel **T**-matrix pattern

Fig. 3. Analysis of matrix system for the Ball grid configuration from Figure 13.

interesting to look at the eigenvalue plots of the **T**-matrices. Here, many eigenvalues have the value 1, and the **T**-matrices appear to be more of a lower diagonal. Generally speaking, the best results are obtained for those configurations where we have fewer multiple eigenvalues of **T** equal to 1, and where **T** is closer to being a lower diagonal.

In Figure 5, we have plotted the convergence rate as a function of the number of iterations. We used a nonsmooth reformulation of the boxed complementarity formulation

$$\vec{H}(\vec{\lambda}) = \min(\vec{\lambda} - \vec{\lambda}_{\text{lo}}, \max(\vec{\lambda} - \vec{\lambda}_{\text{hi}}, \vec{w})), \tag{37}$$

and plotted the value of the merit function

$$\theta(\vec{\lambda}) = \frac{\|\vec{H}(\vec{\lambda})\|^2}{2} = \frac{\vec{H}(\vec{\lambda})^T \vec{H}(\vec{\lambda})}{2}. \tag{38}$$

Thus, if $\theta = 0$, then we have found a solution to the boxed complementarity problem.

The log-plot of the merit function in Figure 5 clearly shows, a linear convergence rate of the iterative Gauss-Seidel solver. This implies an exponentially slow convergence, and, after some fixed threshold, the effort spent on Gauss-Seidel iterations is wasted. Besides, each configuration behaves differently. Thus we cannot find a global reasonably low iteration limit that will work for all configurations. In Section 4, we will introduce shock propagation which overcomes all of these problems with the iterative Gauss-Seidel solver.

3.2 Handling Large-Mass Ratios

Large-mass ratios often lead to convergence problems for matrix splitting methods. In this section, we will discuss the techniques often applied to this problem.

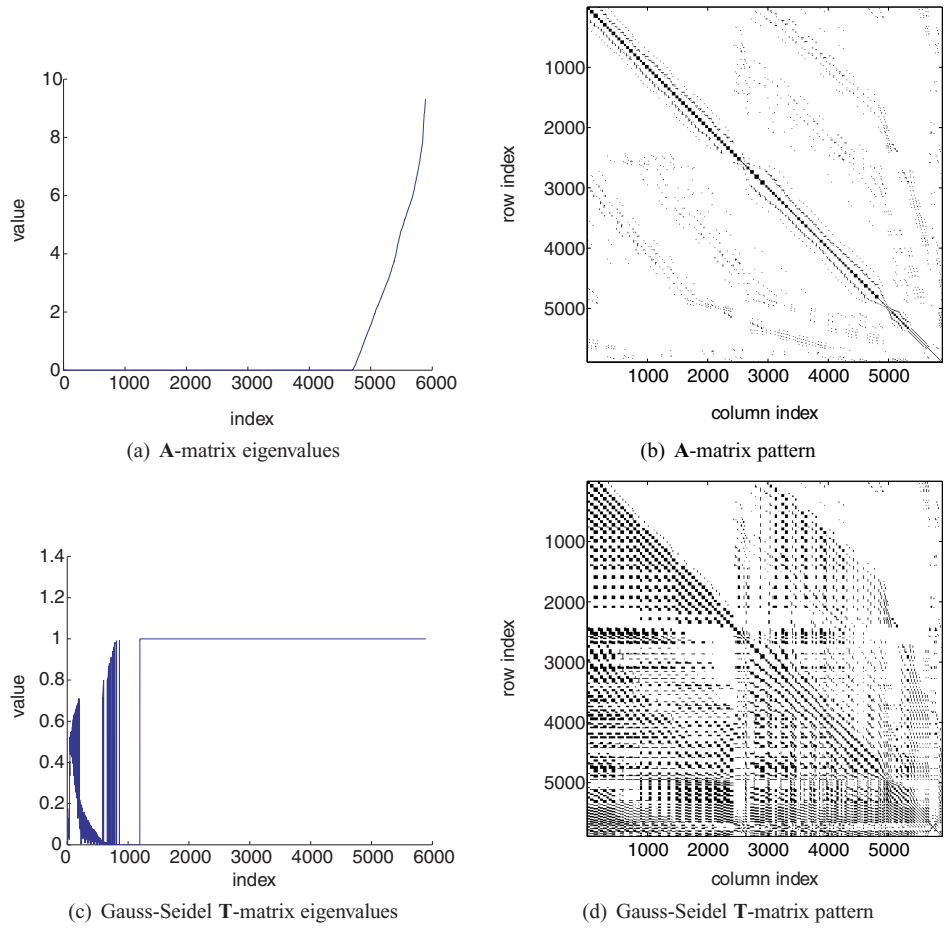
Preconditioning and diagonal scaling (diagonal scaling is a left-right preconditioning) can be used to change the spectrum of the eigenvalues. Generally speaking, the new preconditioned system looks like

$$\mathbf{A}' = (\mathbf{C}^{-1}\mathbf{A}\mathbf{C}^{-T})\mathbf{C}^T, \tag{39}$$

$$\vec{b}' = \mathbf{C}^{-1}\vec{b}. \tag{40}$$

For diagonal scaling $\mathbf{C}^2 = 1/diag(A)$, when comparing the plot of the error of the preconditioned system with the nonpreconditioned system, a small displacement is observed. Diagonal scaling lowers the convergence rate by a constant but does not change the asymptotic behavior.

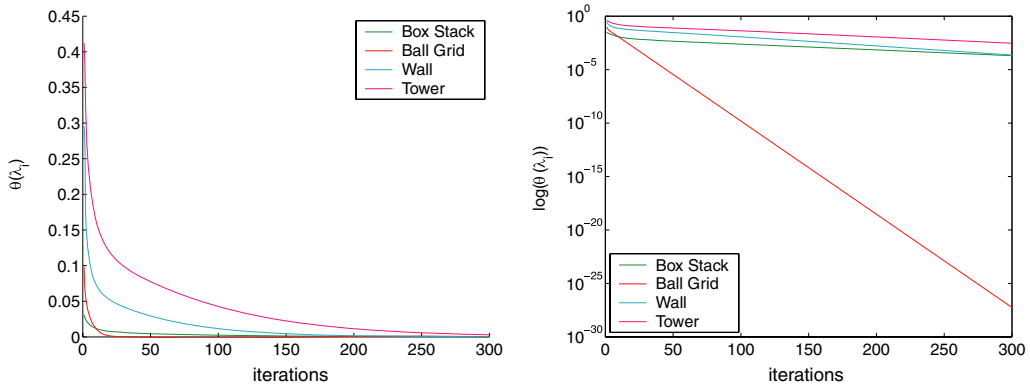
Preconditioning works best for a **C**-matrix which is a good approximation to $\mathbf{A}^{-1}$. If this can be achieved, then one would have $\mathbf{A}' \approx \mathbf{I}$. Using diagonal scaling **C** is not a really good approximation to the inverse of **A** since diagonal scaling seems to work best for



(a) **A**-matrix eigenvalues (b) **A**-matrix pattern

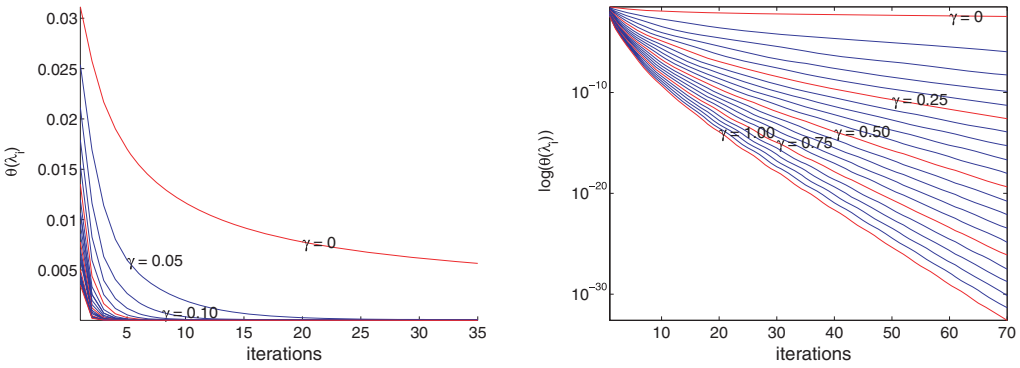
(c) Gauss-Seidel **T**-matrix eigenvalues (d) Gauss-Seidel **T**-matrix pattern

Fig. 4. Analysis of matrix system for the wall configuration from Figure 10.



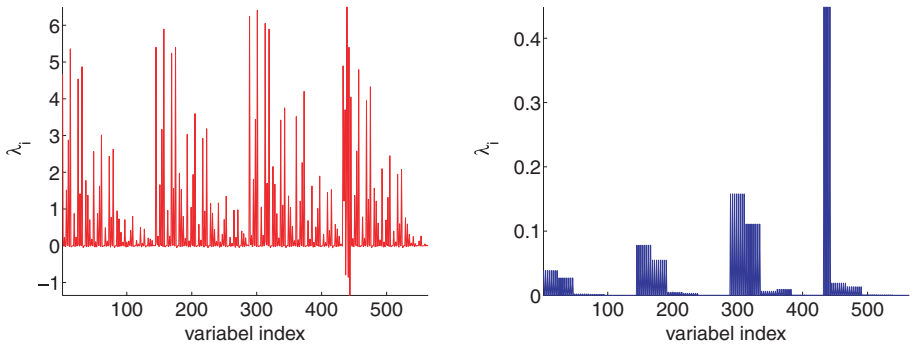
(a) Convergence (b) Log Convergence

Fig. 5. Convergence rate plots of a few selected configurations.



(a) Convergence Damped (b) Log Convergence Damped

Fig. 6. Convergence rate plots for different relaxation values.



(a) Undamped solution (b) Damped solution,$\gamma = 0.1$

Fig. 7. Comparison of a slightly damped solution with an undamped solution of the box-stack configuration from Figure 9.

diagonal-dominant systems, and multibody dynamics problems are not diagonal-dominant. In general, matrix splitting methods seem to work best on diagonal-dominant systems. Just consider the contrived case of using Jacobi on a diagonal system. Here $\mathbf{T}$ will be the identity matrix, and the solution will be reached in a single iteration.

Preconditioning for multibody dynamics problems is not well described in the computer graphics literature. The reason may be that only a limited fixed amount of time is set aside for a dynamics update. Applying the preconditioner is computationally more expensive than throwing more brute force iterations at the problem.

Relaxation or damping has also been applied. That means for nonnegative γ, the system matrix is changed as follows

$$\mathbf{A}' = \mathbf{A} + \gamma\mathbf{I}. \tag{41}$$

As $k \to \infty$, we have $\gamma \to 0$. This is termed constraint force mixing in the Open Dynamics Engine (ODE) community, although they use a constant γ. Consider the equivalent quadractic programming (QP) problem formulation, that is, minimize

$$f(\vec{\lambda}) = \vec{\lambda}^T\vec{b} + \vec{\lambda}^T\mathbf{A}\vec{\lambda} + \gamma\vec{\lambda}^T\vec{\lambda}, \tag{42}$$

subject to

$$\vec{\lambda} \geq 0. \tag{43}$$

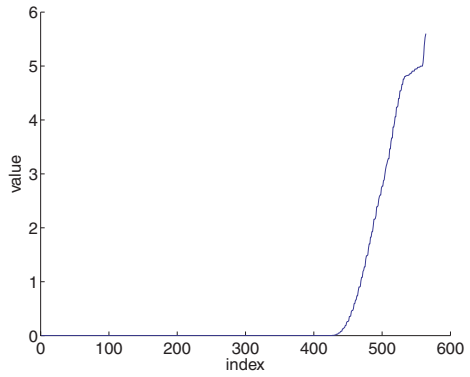
Using Karush-Kuhn-Tucker (KKT) conditions leads to an LCP with the previous $\mathbf{A}'$ matrix. Looking closely at f, one notice that the magnitude of $\vec{\lambda}$ is minimized by the last term. Also note that $\vec{\lambda}$ always will be dampened by this scheme. This leads to weaker constraint forces as already pointed out by Gleicher [1994]. Although $\mathbf{A}$ is symmetric-positive semidefinite, $\mathbf{A}'$ is symmetric-positive definite, which from a numerics viewpoint is tractable. This technique can be used to model soft constraints.

Relaxation seems to change the slope of the convergence rate. The convergence rate p is defined by

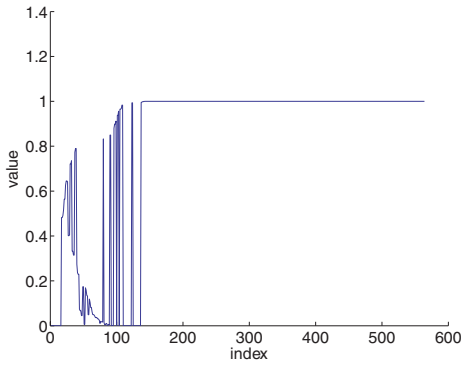
$$lim_{k\to\infty}\frac{|e^k|}{|e^{k-1}|^p} = c, \tag{44}$$

where c is the convergence constant. Larger γ means lower c. For Gauss-Seidel type solvers, $p = 1$, which results in very slow convergence. Figure 6 shows how the convergence rate for the Gauss-Seidel type solver changes for different values of γ when simulating the box-stack configuration from Figure 9. Notice that even a small value of γ has a large impact on the convergence. Further, the damping on the solution is just as significant as shown in Figure 7.

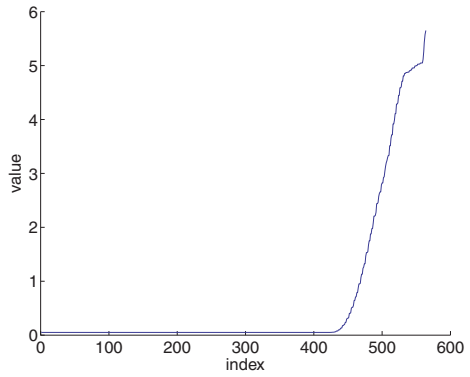
In Figure 8, we have shown results of a matrix analysis. It is particularly interesting to compare the eigenvalue plots with the undamped case. From the figure, it is clear that the large number multiple eigenvalues with value 1 have disappeared.



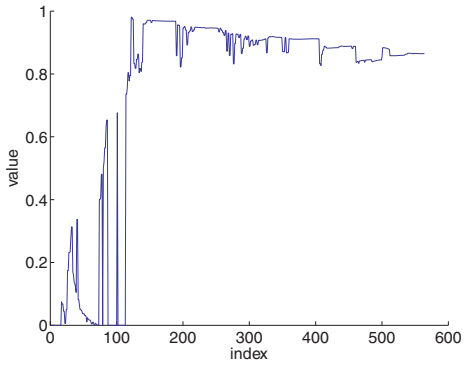
(a) Eigenvalues of undamped box stack **A**-matrix



(b) Eigenvalues of undamped box-stack Gauss-Seidel **T**-matrix



(c) Eigenvalues of damped box-stack **A**-matrix



(d) Eigenvalues of damped box-stack Gauss-Seidel **T**-matrix

Fig. 8. Matrix analysis of undamped and damped box-stack configuration from Figure 9.

4. VELOCITY-BASED SHOCK-PROPAGATION

Equations (45) summarizes the explicit timestepping scheme outlined in Section 2.

$$\vec{b} = \mathbf{J}(\vec{u}^{\,t} + \Delta t \mathbf{M}^{-1}\vec{f}_{\text{ext}}), \tag{45a}$$

$$\mathbf{A} = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T, \tag{45b}$$

$$\vec{\lambda} = \mathbf{ncp}\,(\mathbf{A}, \vec{b}), \tag{45c}$$

$$\vec{u}^{\,t+1} = \vec{u}^{\,t} + \mathbf{M}^{-1}\mathbf{J}^T\vec{\lambda} + \Delta t \mathbf{M}^{-1}\vec{F}_{\text{ext}}, \tag{45d}$$

$$\vec{s}^{\,t+1} = \vec{s}^{\,t} + \Delta t \mathbf{S}\vec{u}^{\,t+1}. \tag{45e}$$

For ease of notation, let us denote a timestep according to these equations by *dynamics*(Δt). Applying *dynamics*(Δt) using the iterative solver in Section 3 could result in slow convergence, as shown in Figure 9. Experiments [Erleben 2005] indicate that not even 100,000 iterations will work for the box-stack. Figures 10 and 11 show acceptable visual results with 100 iterations, but the computational cost is high.

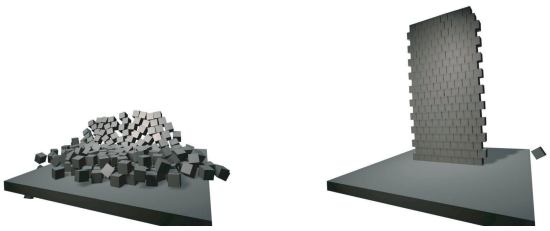
Adopting shock propagation [Guendelman et al. 2003] to the velocity-based complementarity formulation will yield a significantly lower number of iterations and solve the slow convergence problem of the box-stack in Figure 9. In all our simulations, we never used more than 10 iterations when performing *dynamics*(·),

(a) Initial position. (b) 100 iterations after 4 secs.

Fig. 9. Box-Stack: A stack of 25 resting boxes with a poor convergence rate for the iterative solver.

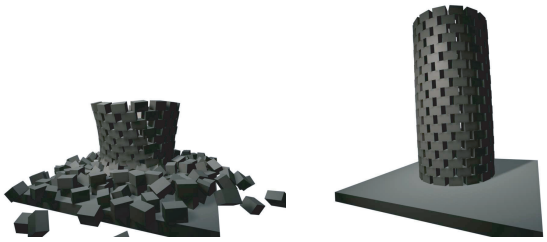
and only 5 iterations when doing error correction which will be discussed in the following.

Each body in a configuration is assigned a stack-height number, which indicates the number of bodies in immediate contact that needs to be visited in order to reach the closest fixed body. Free-floating bodies are simply assigned the maximum stack-height. A stack layer i is defined as all bodies with stack-height i and $i+1$ and including all contact points between these bodies. Keeping a contact graph data structure, the stack-layers are easily analyzed and constructed by performing a single breadth-first traversal starting at all the fixed bodies in the configuration. This approach is



(a) 10 iterations after 4 secs. (b) 100 iterations after 4 secs.

Fig. 10. Wall: A wall of 200 bricks with an acceptable convergence rate of the iterative solver.



(a) 10 iterations after 4 secs. (b) 100 iterations after 4 secs.

Fig. 11. Tower: A tower of 320 bricks with an acceptable convergence rate of the iterative solver.

```
algorithm shock-propagation(algorithm A)
  compute contact graph
  for each stack layer in bottom up order
    fixate bottom-most objects of layer
    apply algorithm A to layer
    un-fixate bottom-most objects of layer
  next layer
end algorithm
```

Fig. 12. Pseudocode version of the general shock-propagation algorithm.

a little different from Guendelman et al. [2003] because we want both to minimize the usage of the collision detection and to keep it strictly separated from the dynamics. This also implies that we do not reevaluate contact points during the shock propagation. Figure 12 shows a pseudocode version of the shock-propagation algorithm we use, and which we will denote *shock propagation*() in the following. The initial intention of shock-propagation is to fix simulation errors, thus we can apply the projection error correction [Baraff 1995] to each stack-layer in a bottom-up fashion. This has the advantage of being able to completely fix penetration errors. During error correction, all contact tangent plane constraints are dropped, and the right-hand side vector $\vec{b}$ is replaced with a $\vec{d}_{\text{penetration}}$-vector of penetration depths in the contact normal direction. This results in the timestepping scheme,

$$\mathbf{A} = \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T, \tag{46a}$$

$$\vec{\lambda} = \mathbf{ncp}\,(\mathbf{A}, \vec{d}_{\text{penetration}}), \tag{46b}$$

$$\vec{s}^{\,t+1} = \vec{s}^{\,t} + \mathbf{S}\mathbf{M}^{-1}\mathbf{J}^T\vec{\lambda}, \tag{46c}$$

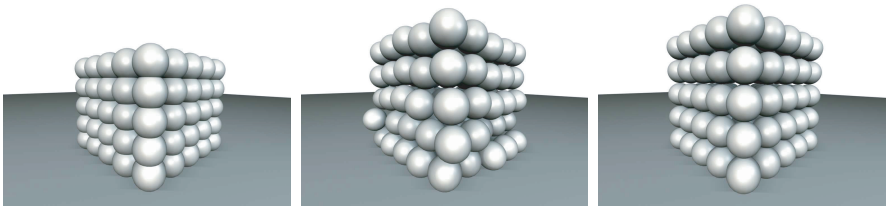
which we denote by *correction* () for convenience. Note that the NCP reduces to a linear complementarity problem (LCP) since the coupling between normal and tangential directions are dropped, but (46) is still solvable by the same iterative solver. Figure 13 shows the difference in using *correction* () and *shock propagation* (*correction* ()). Note that within 6 frames, shock-propagation converges correctly even with the low number of iterations, whereas the correction completely fails to converge without shock propagation. When using the velocity-based complementarity formulation with shock propagation, we propose the timestepping scheme shown in Figure 14. We have introduced a weighting of the dynamics versus the shock propagation given by a, where $0 \leq a \leq 1$. The weighting was necessitated by experiments which indicated that the weighting was directly related to the amount of simulation error. In order to fix simulation errors in the velocities during the shock propagation, we first perform a dynamics step, followed by an error-correction step to fix positional errors. Ideally a should be set to 1 in which case poor convergence of the iterative solver will dominate the simulation. Setting a equal to zero results in perfect behavior of the shock propagation, resulting in perfect error correction. Unfortunately, bodies initially at rest in configurations of in-equilibrium systems cannot feel the weight of each other. Therefore a should never be set to zero in practice. In most cases, we have used a-values in the range $0 < a \leq 0.01$.

The last step in the timestepping scheme consist of a final error-correction step, and it is needed because we do not reevaluate contact points during the shock propagation. The final error-correction step has the effect of smoothing the simulation errors by distributing them to nearby objects. The benefits is twofold: the amplification of simulation errors is avoided, and the cyclic dependency problem of shock propagation is remedied. Usually configurations such as Figure 17 suffer from the cyclic dependency problem where artifacts such as high-frequency oscillating bodies can be observed just below the top-most bodies. Our simulations do not suffer from such artifacts.

Shock propagation can be understood as a two-stage iterative scheme with a preconditioner: first the outer stage consists of dividing the problem into stack-layers. Each stack-layer corresponds to a single block. Then one iterates over the blocks in a bottom-up fashion. This is an unconventional mixture of Jacobi/Gauss-Seidel-like iterations: the diagonal blocking is similar to a Jacobi scheme, but layers must be processed in a certain order which makes the processing of blocks sequential in nature, that is, similar to Gauss-Seidel. The inner stage consists of solving each block, that is, each stack-layer, using a projected Gauss-Seidel method. For each of these blocks, the mass ratios are changed by fixating some of the bodies before solving the block. This is similar to preconditioning.

5. RESULTS FOR VELOCITY-BASED SHOCK PROPAGATION

Figure 1(a)–1(d), 15, 16, and Figure 17 show simulation results using the velocity-based shock propagation timestepping scheme. Computations were performed on a 1.7GHz CPU with 1GB system memory. During simulation, we measured the total frametime, the time used for collision detection, and the total number of generated contact points. Worst case frametime were: engraved letters, 0.18 seconds, cow pile, 1.2 seconds, falling roof, 0.25 seconds, and silo, 1.1 seconds. In all of our simulations, we have used a timestep of 0.01 seconds. The simulation in Guendelman et al. [2003] ranges from 500–1000 bodies and frametime go from 5 minutes to 7 minutes, whereas frametime for similar configurations using our simulator are more than a hundred times faster. The collision detection systems for the two simulators are not the same, and, in Section 5.5, we present a comparison with identical collision detection. Figure 18



(a) 1'st frame with severe penetrations (b) 6'th frame with out shock-propagation (c) 6'th with shock-propagation

Fig. 13. Ball-Grid: A grid stack of 125 balls. The iterative solver used only 5 iterations.

```
algorithm velocity-based-shock-propagation(a,dT)
  collision detection at time t
  dynamics(a*dT)
  shock-propagration(
    dynamics( (1-a)*dT ), correction()
    )
  collision detection at time t + dT
  correction()
  t = t + dT
end algorithm
```

Fig. 14. Pseudocode of velocity-based shock propagation (VSP) algorithm.

shows the actual frametime spent on computing the motion, that is, frametime minus time used on collision detection.

The plots show a linear dependence on the number of contact points, the different slopes are the result of the different stacking topology in the four configurations.

For completeness, we have also plotted the time spent on collision detection in Figure 19. The silo, engraved plane, and falling roof configurations all used box and sphere geometries, whereas the cow pile configuration used signed distance maps combined with sphere-trees. The plots appear to have linear or piecewise linear behavior: horizontal or negative slopes are a consequence of the caching schemes applied to the contact graph in the collision detection engine. The piecewise positive slopes indicate that the broad-phase collision detection quickly prunes unnecessary test such that time is only spent on doing collision detection for objects in close proximity. The scattering of the cow-pile plot indicate that the pruning capability of the sphere-trees may be improved.

To compare the velocity-based shock propagation (VSP) described in this article with other methods, we have selected a challenging test configuration. The test configuration consists of a 40-layers-high tower with 800 objects. The stacking causes large mass ratios, thus leading to a numerical badly conditioned system. Animating the tower standing still for a long time stresses the algorithm's ability to counter accumulation of numerical errors. During the animation, a large block falls down and impacts with the top of the tower. On impact, the large block will transfer large impulses down through the tower due to restitution, and the impilses are then transfered back up through the tower. After a while, it is expected that the heavy block will come to rest. Besides a badly conditioned problem due to the large mass of the large block at time of impact, there are also large penetration errors further stressing the stacking capabilities. Finally, a canonball is used to hit the tower, and the tower is expected to collapse in a physically plausible way.

We have compared VSP with the rigid body simulator in Maya6.5, the aggressive impulse-based shock propagation (ISP) method in Guendelman et al. [2003], Novodex (v2.1.2), and ODE (v0.30). All computations were performed on a 1.7GHz CPU with 1GB system memory. Timing results are presented in Section 5.5. We did not choose the method in Kaufman et al. [2005] because, even though it has been shown to work quite well for large scale dense random piles, it has not been verified to work on large-scale dense structured stacking. The add-hoc contact model for bounce in Kaufman et al. [2005] and the approximation of Newton's third law raise some concern about stability. In addition, the method tolerates penetration errors, which are visually displeasing for structured stacks. This method will not be discussed further in this article.

Figure 20, 21, and 22 show the three best simulation results of the tower-roof configuration from three different viewing angles.

5.1 Maya6.5

The simulator in Maya6.5 is, as far as we know, an acceleration-based complementarity formulation [Baraff 1994] using a solver based on a direct method (Dantzig pivoting) and a bisection root-searching algorithm for time control. Under the same initial conditions as VSP, this simulator crashed. We speculate that the problem is caused by objects initially in touching contact. Subsequently, any timestep regardless of the size will result in penetration, and the bisection algorithm will enter an infinite loop.

We did alter the initial setup slightly to ensure that all objects were separated. As soon as objects came in contact, the time complexity of the direct method affected the performance of the animation. After 2 hours, the simulation was manually stopped with only half of the 800 bricks in contact.

5.2 Impulse-Based Shock Propagation

A comparison has been made between our velocity-based shock propagation (VSP) and the aggressive impulse-based shock propagation (ISP) method in Guendelman et al. [2003]. We have implemented ISP in OpenTissue to make the comparison as fair as possible.

The timestepping of ISP has the same time complexity, $O(n)$, as VSP. The major differences are the following.

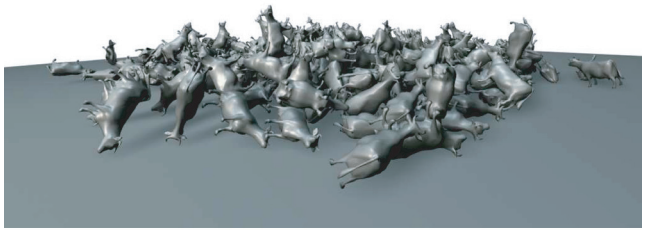
—ISP uses the equivalent of 15 collision queries per-frame, VSP uses only 2 queries.

—ISP uses both predicted positions and velocities, implying a more implicit stepping method than VSP.

—ISP cannot deal with penetration errors.

Both methods are impulse-based, but impulses in ISP are instantaneous, whereas VSP can integrate the effect over time. We

(a) frame 0



(a) frame 74

Fig. 15. Cow Pile: 250 cows falling into a pile simulated with our method. See supplementary movie.

believe this causes the difference in visual quality. When stressing large-scale structured stacking, like banging a big box on top of a large stack, objects tend to jitter and shake with the ISP method, whereas the VSP does not add unnecessary jittering as shown in Figure 23.

There is a clear difference in how material parameters such as friction and restitution behave. Simulations with ISP seem to be a little too lively compared to the physical meaning of these parameters. VSP seems to do a better job of simulating plausible physically correct behavior. When using complex geometries, the query times of the collision detection really slows ISP down compared to VSP.

5.3 Novodex (v2.1.2)

Novodex is current state-of-the-art in commodity software, and there is no way of really telling what kind of algorithms are being used. Based on the parameters available and the effect of tuning these, Novodex appears to use some sort of hybrid between a velocity-based complementarity formulation solved using a iterative Gauss-Seidel-like scheme and a penalty method.

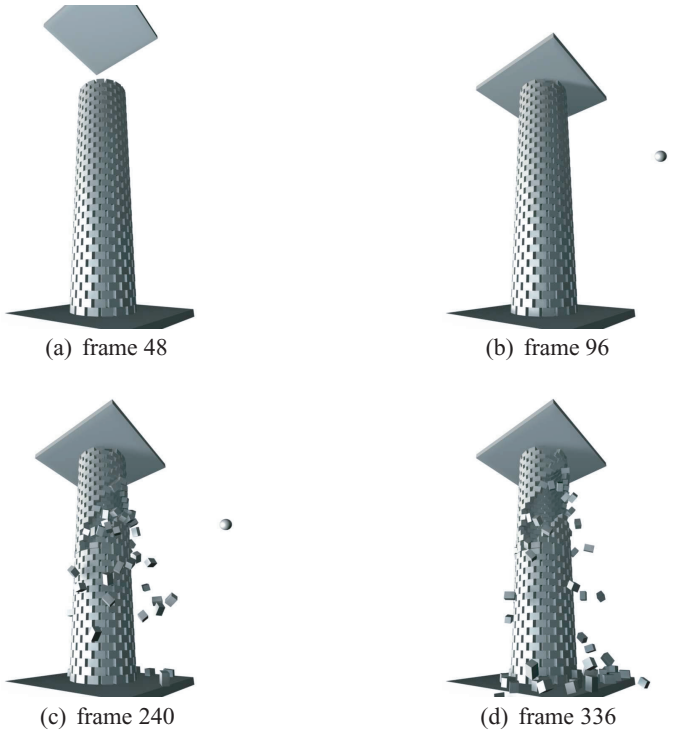
Novodex is for game animation. It is intended for game scenarios where exaggeration is acceptable and desired, which is observed from a simulation using default settings as shown in Figure 24. In game physics, physical properties are often set to make the simulation look plausible, while running extremely fast. This means low mass densities and low gravitation, typically density $= 10$ Kg/m^3 and gravity $= 0.5$ m^2/s. Novodex is multithreaded which makes it very difficult to perform timing comparisons of the algorithms used. We used the following API calls to make Novodex block its simulation until it was completed.

```
NxScene->setTiming(0.01,1,NX_TIMESTEP_FIXED);
NxScene->simulate(0.01);
Nxcene->flushStream();
NxScene->fetchResults(NX_RIGID_BODY_FINISHED,true );
```

Time duration was measured between calls to `simulate` and `fetchResults`. Novodex offers many parameter-tuning options, and those that appeared important for us are listed in Table I. It is difficult to make the tower appear rigid using the default settings for which the tower tends to be elastic and explode on large impact as shown in Figure 24. PSV the best tweaked setting we found, the tower behaved in a rigid manner, but penetration errors were quite large as shown in Figure 25, and the canonball impact did not behave as expected as shown in Figure 22.

5.4 Open Dynamics Engine (v.0.30)

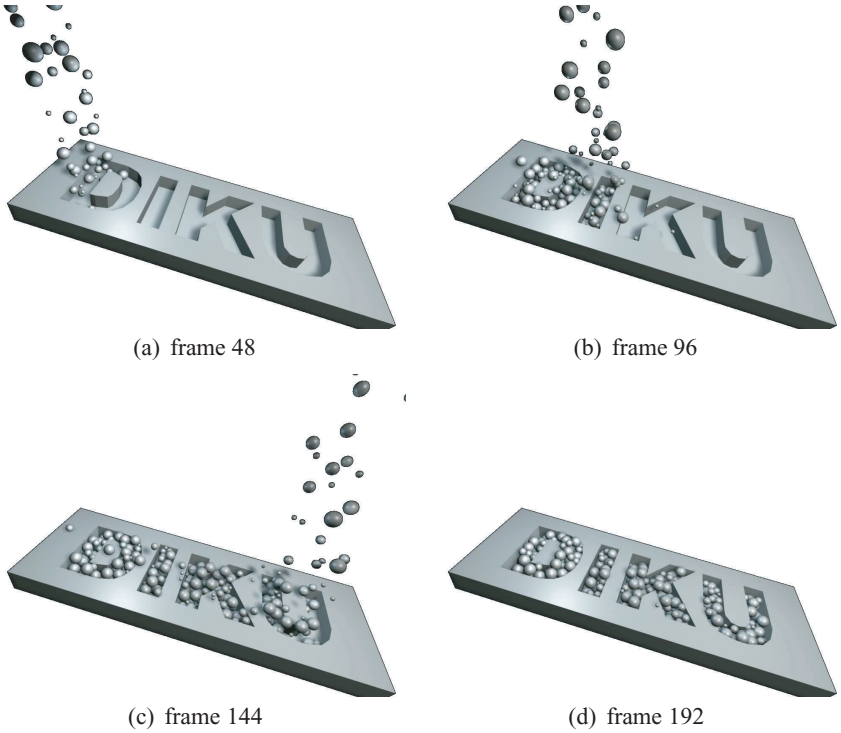
Open Dynamics Engine (ODE) is a velocity-based complementarity formulation. It uses a damped iterative solver similar to SOR. Further error correction is handled by stabilization. Like Novodex, this engine works best with game-friendly values, that is, low density and gravity values. Damping, together with stabilization, adds a soft/elastic impression to the objects. To achieve fast and error-tolerant simulation, these parameters cannot simply be turned off. Performance-wise the engine should have the same complexity as VSP and ISP. We were not able to simulate a tower larger than 6



(a) frame 48 (b) frame 96

(c) frame 240 (d) frame 336

Fig. 16. Falling roof: Roof falling onto a tower of 640 bricks followed by a canon-ball collision simulated using our method. See supplementary movie.



(a) frame 48 (b) frame 96

(c) frame 144 (d) frame 192

Fig. 17. DIKU: 1000 balls of various sizes falling onto an inclined plane with engraved letters simulated with our method. See supplementary movie.

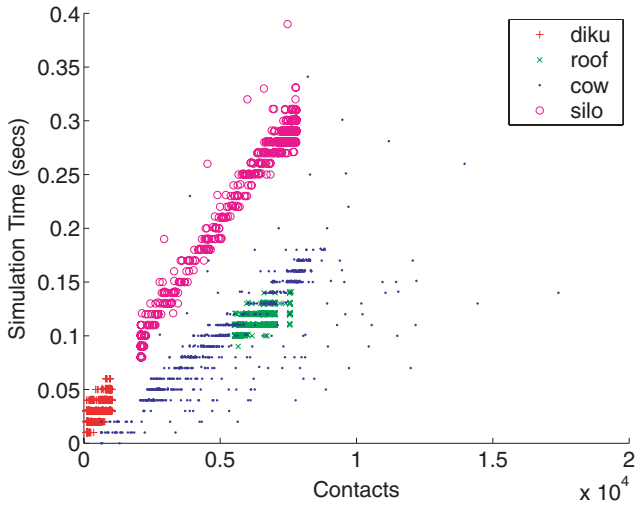


Fig. 18. Total timestepping computation time as a function of the number of contacts points showing linear complexity.

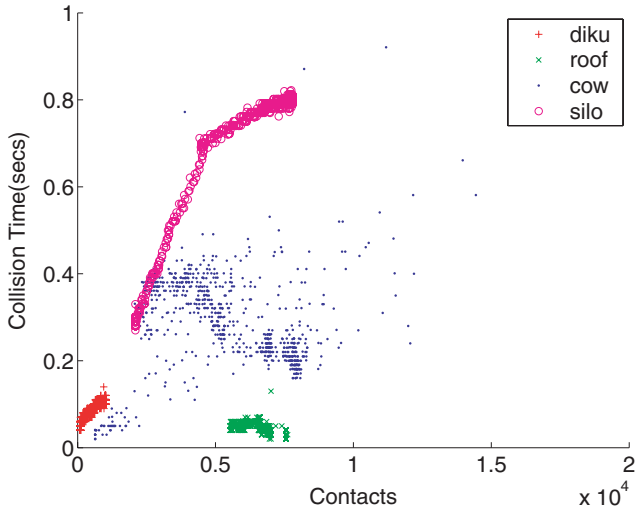


Fig. 19. Total collision detection computation time as function of the number of contacts points showing noworse than linear complexity.

layers without obtaining a memory error. The bricks appears quite squeezed, and upon impact with the heavy block, the bricks are knocked down onto the ground as shown in Figure 26.

5.5 Comparing Performance

In order to compare the performance of the different methods, we have measured the total wall-clock time of each update step, that is, the total time of both doing collision detection and computing the dynamics. Our results are shown in Figure 27. ODE is incapable of competing with either ISP, VSP, Novodex, or Maya since it breaks down due to memory errors for large-scale configurations. We have therefore omitted ODE from the timing comparisons. The performance of Maya was so slow that the simulation was not completed within reasonable time. We have therefore omitted Maya from the timing comparisons.

Novodex is extremely fast and works great in most cases, but for large-scale dense-structured stacking, this engine has trouble delivering the same kind of quality as ISP and VSP.

ISP is, to our knowledge, the state-of-the-art for stable stacking, and in comparison to VSP, ISP is slower. The difference in speed is caused by the difference in interaction with the collision detection engine. There is also a difference in the visual quality of the simulation. ISP tends to shake and rattle objects apart in large-scale dense-structured stacks, which is not observed with VSP.

Neither Novodex nor ODE were able to simulate the initial configuration without tweaking masses and gravity. ISP and Maya were able to simulate the configuration without altering masses and gravity, but Maya was not able to deal with initial touching objects.

6. DISCUSSION

In the following, we will discuss various aspects of numerical simulation and give some perspective on our method. We will make comments on visual artifacts and make suggestions for improvements.

6.1 Penetration Errors in Multibody Dynamics

Timestepping methods for multibody dynamics can lead to unavoidable penetrations. In the following, we discuss the causes that lead to this unwanted artifact.

(1) Doing physics-based simulation means that some approximation of the laws of physics is used. For instance, a contact manifold is often represented as a plane at the point of contact due to a myopic view of the world. Applying nonpenetration constraints to planar contacts means that objects are prevented from moving beyond the plane at the point of contact. Thus, if the surface is of higher order than a plane, then penetrations can occur no matter how small the timesteps that are used.

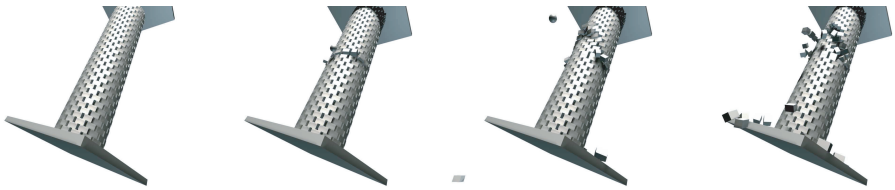
(2) If the order of the timestepping scheme is less than the order of the spatial representation, then one may overshoot. Often fixed timestepping schemes are used in real-time interactive simulation in order to deliver predictable performance. This is equivalent to some variant of a first-order Explicit/Modified/Implict Euler scheme.

(3) There is the problem of precision and round-off errors which are often dealt with using either thresholds or interval arithmetics.

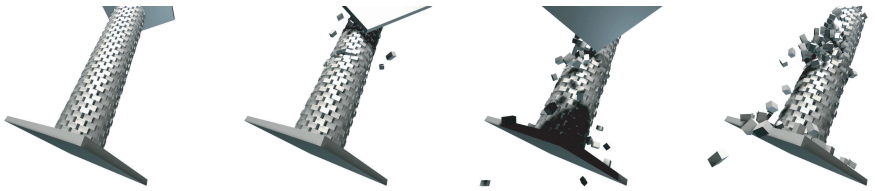
(4) Future undetected contacts may also lead to penetrations. This is often caused by the timestepping scheme and is an effect easily observed for explicit fixed timestepping schemes using large timesteps. It can lead to jittering/jumping if one uses error correction by projection, or explosive behavior if one corrects errors using stabilization. In extreme cases, one may even get tunneling and overshooting artifacts.

If the motion or surface representation is of higher order than any of the approximations in (1) or (2), then it is to be expected that the time-integration of the motion may lead to penetration. If higher-order approximations are used instead, and thereby gain more accuracy at a performance degradation, then the number of troublesome cases will be reduced.

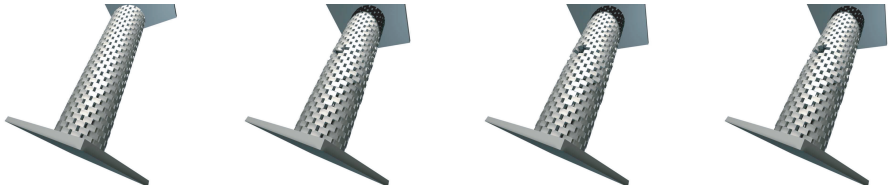
The cow configuration from Figure 15 is a very challenging configuration to animate. Each cow is basically an ellipsoidal object with long thin tentacles. The cows are dropped from several hundred meters, and, when a cow hits the surface, its speed will be approximately half its own length per-frame. This causes extreme penetrations between the surface and the cows. After the first bounce, the cows have gained a large angular velocity, causing them



(a) Velocity-based Shock-propagation (VSP)

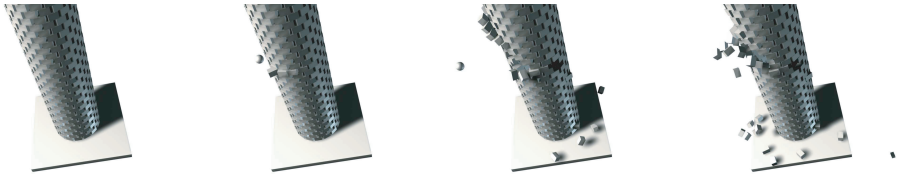


(b) Impulse-based Shock-propagation (ISP)

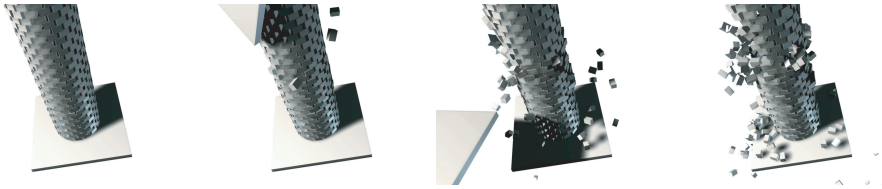


(c) Novodex using 30 iterations and low mass

Fig. 20. Results of roof-tower simulation corresponding to frame 60, 120, 180 and 240. See supplementary movies.



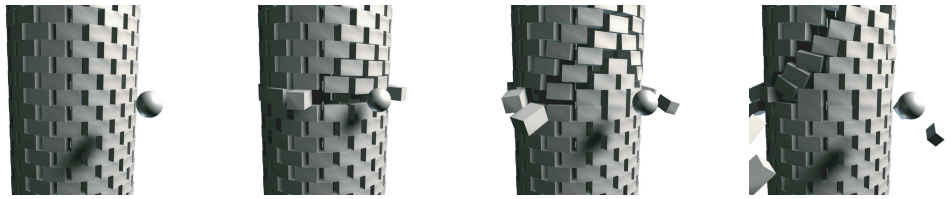
(a) Velocity-based Shock-propagation (VSP)



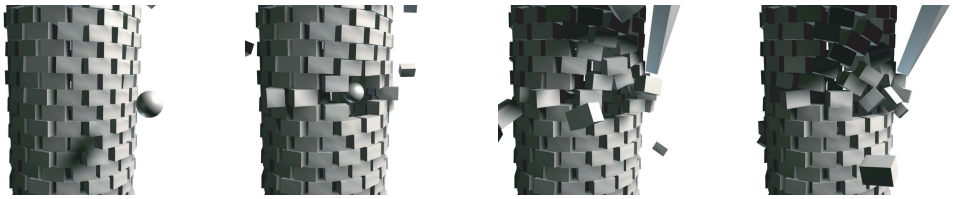
(b) Impulse-based Shock-propagation (ISP)

(c) Novodex using 30 iterations and low mass

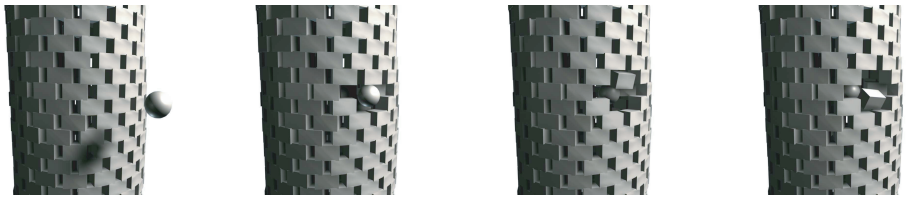
Fig. 21. Results of roof-tower simulation corresponding to frame 60, 120, 180 and 240. See supplementary movies.



(a) Velocity-based Shock-Propagation (VSP)

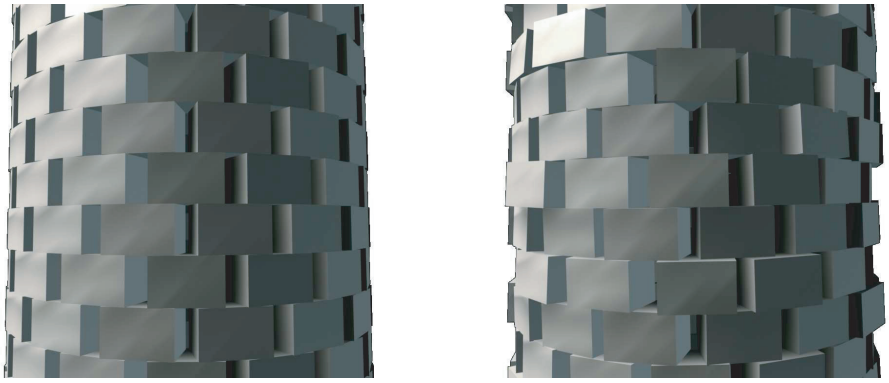


(b) Impulse-based Shock-Propagation (ISP)



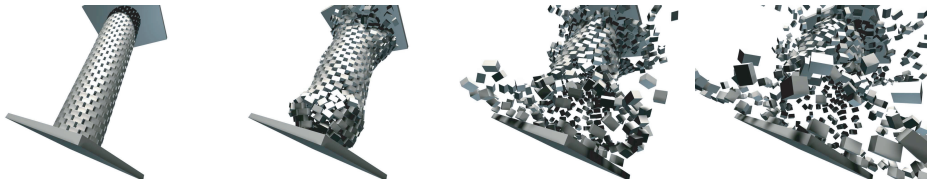
(c) Novodex using 30 iterations and low mass

Fig. 22. Closeup views of simulation results when a canonball hits the tower. Images from left-to-right correspond to frames 110, 120, 130, and 140. Notice the randomness look of ISP and the failure of Novodex to feel the impact of the canonball. VSP forms a nice crack in the wall on when the ball impacts it.



(a) Velocity-based Shock-Propagation (b) Impulse-based Shock-Propagation

Fig. 23. Closeup of frame 80, showing the shaking and jittering of impulse-based shock propagation compared with the more stable result of velocity-based shock propagation.

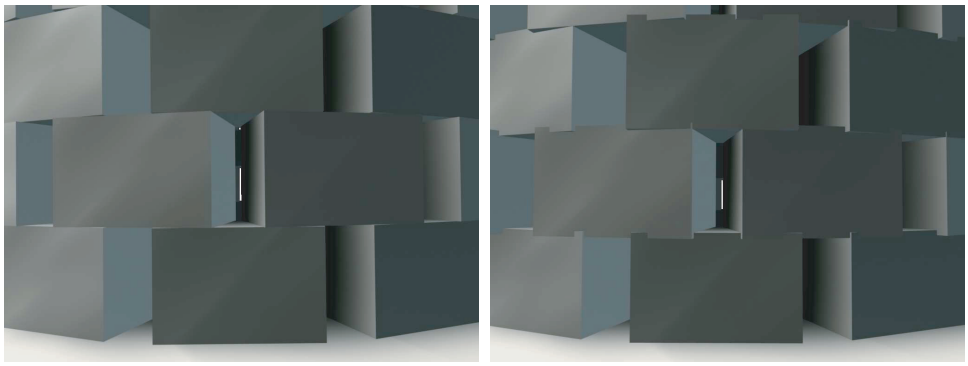


(a) frame 70 (b) frame 80 (c) frame 90 (d) frame 100

Fig. 24. Results of tower simulation with Novodex using default settings. Notice the unwanted explosion due to the impact. See supplementary movie.

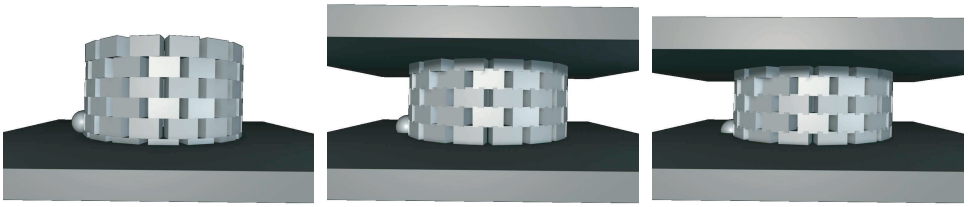
Table I. Novodex Options

Name	Description
`NxBodyDesc.solverIterationCount`	This seems to be directly related to some kind of Gauss-Seidel maximum iteration count. Tweaking it definitely has a similar effect. Default value is 4. Using 2000 iterations without tweaking other options seems to maintain a good rigid structure.
`NX_DEFAULT_SLEEP_LIN_VEL_SQUARED`, `NX_DEFAULT_SLEEP_ANG_VEL_SQUARED`	These parameters seem to control the sleepy policy. For a fair comparison, we turn this off by setting the values to zero.
`NX_MIN_SEPARATION_FOR_PENALTY`	Simulations are difficult to perform for values close to zero. The default value of -0.05 seems to work best even when iteration count are very high.
`NX_PENALTY_FORCE`	This controls the scale of the penalty forces. Default value is 0.6.



(a) Velocity-based Shock-Propagation (VSP) (b) Novodex using 30 iterations and low mass

Fig. 25. Closeup of frame 80, showing deep penetrations for Novodex and none for VSP.



(a) frame 330 (b) frame 420 (c) frame 450

Fig. 26. Results of a limited roof-tower simulation with ODE. Notice that boxes sink into the plane and large interpenetrations develop. Upon impact, the first box layer is completely knocked below the plane. See supplementary movie.

to spin violently. The thin legs and tail of one cow can therefor move completely inside another cow during a single timestep. Figure 28 illustrates the problem. The cow simulation shows that correction by projection is robust in extreme cases but results in a popcorn effect.

In the supplementary video (see cow pile video with timestep size of 0.01 seconds), the projection artifacts are prominent in an initial, very short period of time. However, a long-term popcorn effect can also be seen. If we study the energy plots of the small-size timestep simulation, we will notice a particular correspondence with the the long-term popcorn effects. In Figure 29, we have plotted the potential energy of two cows showing long-term popcorn effects. From the plots, it is evident that the potential energy slowly increases

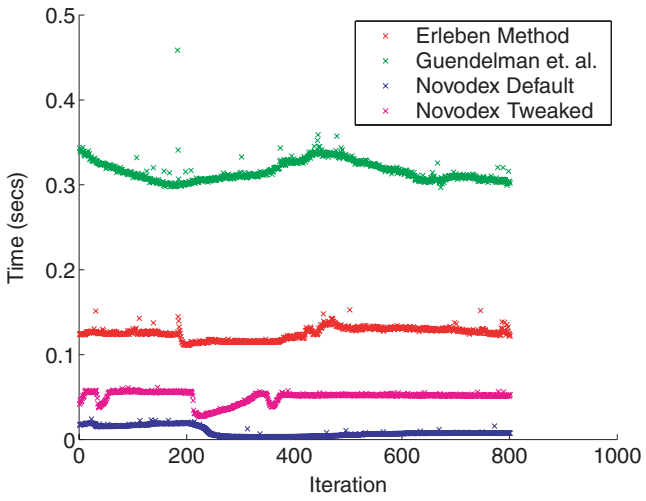


Fig. 27. Performance timings of comparisons.

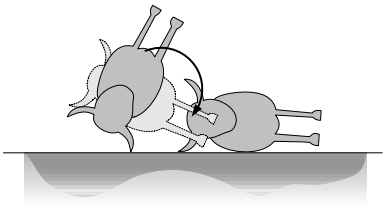


Fig. 28. Fast spinning objects with thin tentacles cause large penetration in explicit fixed timestepping schemes.

over time, and then almost instantaneously drop. The cause of this effect is not due to shock propagation but due to the Gauss-Seidel solver. The langrange multipliers found by the solver are not very accurate. The lagrange multipliers are slightly too large, causing bodies to be slowly pushed apart. Once bodies are separated by more than the size of the collision envelope, then the bodies will once again experience a free-fall under gravity, and thereby sink deep into the collision envelope, and the process will be repeated. This can also be verified in the cow pile video by noticing that with short-term popcorn effects, the cows are pushed apart, but in the long-term popcorn effects, cows are sucked down.

We tracked the potential energy of a single cow standing on a fixed ground to demonstrate that this long-term popcorn effect is caused by the Gauss Seidel solver. In this configuration our shock propagation scheme only has a single layer, and shock propagation thus has no effect on the simulation. The energy plots are shown in Figure 30. As shown from the figure, the saw-tooth pattern is diminished with decreasing timestep size. The reason for this is that the distance a cow falls under gravity during a single timestep is shortened. This is also visually verified by running the cow pile simulation with a lower timestep size of 0.001 second. The problem could also have been handled by using a more accurate NCP solver but that is out of the scope of this article, and we leave it for future research.

From a convergence point of view, all of the previous penetration problems can be made as small as wanted by setting the timestep size sufficiently low. This implying that, if one knows the maximum allowable error, then one can find the maximum velocities, and a timestep size for an explicit method can be computed. However, in real-time interactive applications which are the ones we have in mind, one cannot obtain a predictable performance.

Large penetrations are a natural consequence of explicit fixed timestepping schemes. These schemes can be combined with continuous collision detection to solve this problem as discussed in next section.

6.2 Continuous Collision Detection

All the artifacts described in Section 6.1 are unattractive in real-time animation where we do not have the luxury of decreasing the timestep size too much. Continuous collision detection may be a very interesting solution.

Ideally, continuous collision detection will not miss a collision during a timestep, and the contact time will be computed exactly. A less restrictive definition would be to ease the requirement of not missing any collisions. If we consider collision detection algorithms concerned with the time aspect, these then have existed in graphics for at least fifteen years. Early work includes space-time bounds [Cameron 1990] and hyperbolic approximations [Hubbard 1993]. The methods in these works tried to compute a time-of-impact (TOI) based on a four-dimensional computation. Later, sweeping volumes and TOI computations based on ballistic approximation of motion trajectories were introduced [Mirtich 1996]. Oriented bounding-box hierarchies have been extended to determine first point of intersection queries using linear and angular velocities to predict object motion [Eberly 2007a, 2007b]. In Milenkovic and Schmidl [2001], noninterpenetration positions were solved for by minimizing the kinetic energy. Their method required solving a large convex quadractic programming (QP) problem. Recently [Stephane Redon and Coquillart 2002; Redon 2004] introduced screw motion and extended it to articulated figures [Redon et al. 2004].

Overall methods for continuous collision detection can be divided into two groups:

—conservative advancement methods and

—interpolation methods.

Conservative advancement methods compute a smaller TOI, advance time to the TOI, perform collision resolving, and finally recompute TOIs before advancing once again. Interpolation methods take as input two configuration states at different times, and then try to find a penetration-free state in between these two states. The problem of interpolation methods is to decide what kind of interpolation is needed. If objects are penetrating at the starting and ending times, then it would be difficult to devise an interpolation of the objects that result in nonpenetration. The problem may sound contrived, but it is not. In game-like worlds, users are allowed to add and manipulate objects in unphysical ways, thus, creating a box halfway inside another box is quite common.

The continuous aspect has also been addressed in engineering using more complex timestepping schemes. An implicit timestepping scheme is capable of seeing future contacts, and these future contacts appear as an extra term which is added to the right-hand side of a velocity-based complementarity formulation [Sauer and Schömer 1998]. In other words, they act as a kind of TOI estimate. The implicit scheme will guarantee that none of these future contacts are violated.

In practice, objects appear to have a repel-envelope. If nonzero restitution is used, then objects will bounce at the future contacts, and their altered motion may lead to new penetrations at other undetected contacts. To remedy thus, one can either make a fully implicit scheme using a fixed-point timestepping scheme or use adaptive

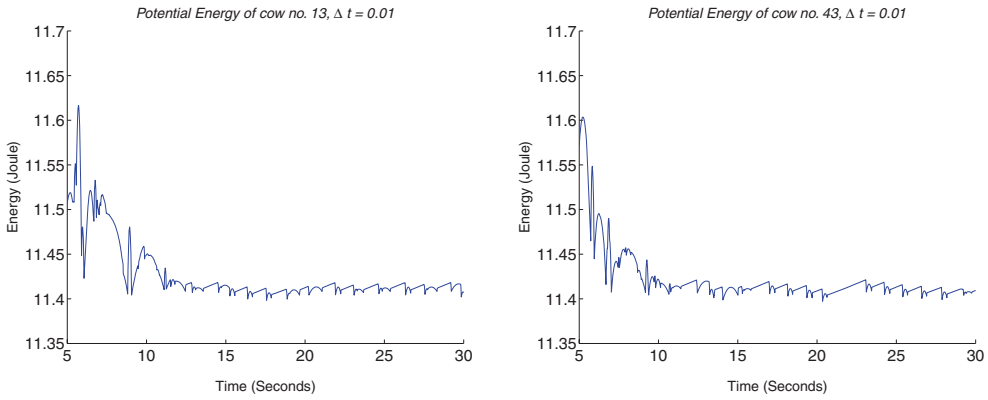


Fig. 29. Potential energy of two cows from the cow pile simulation with timestep size of 0.01 second. Both cows have noticeable long-term popcorn effects. Notice the remarkable saw-tooth shape of the energy plots.

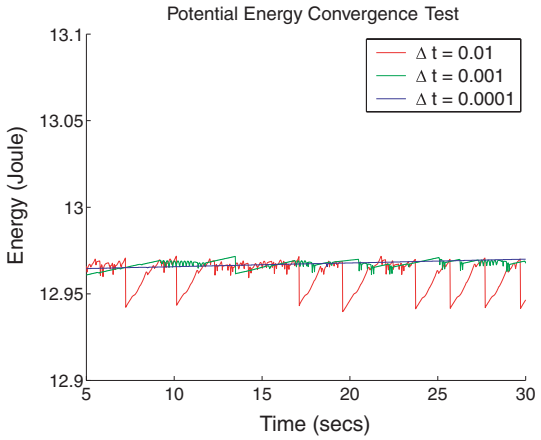


Fig. 30. Convergence plots of potential energy of a single cow standing on a fixed ground. Notice that the saw-tooth pattern is diminished when the timestep size is lowered.

timestepping. In real-time interactive animation, these choices are undesirable for performance reasons.

In game-engines, velocities and forces are typically clamped with an upper bound. The benefits are twofold: one can control the maximum possible penetration, and one can suppress some of the sources to numerical problems. Often several layers of envelopes are applied: a small envelope where penetrations are dealt with using stabilization (i.e., penalty forces), and a deeper and larger envelope where penetrations are dealt with using projection. Many engines run at an internally faster rate than the frame rate, typically 0.01 seconds, thus keeping the timestep size down and ensuring that bodies do not move too much in between a timestep.

6.3 Summary

We have presented an algorithm based on an iterative solver and shock propagation for velocity-based complementarity problems that yields substantial improvements in performance and quality over previous methods. It has been shown how to incorporate error-correction by projection into this scheme, which results in a simple solution to the problem of cyclic dependency of shock propagation. A publicly available implementation of the presented algorithms is available from OpenTissue.

The work presented achieves linear complexity in the number of contact points. One cannot do better than this since at best it would require linear time for the collision detection engine to detect the contact points in the first place.

It is our belief that better convergence of iterative methods could lower the number of iterations used. Thus, a fertile area for future research is to study and improve the convergence rate of iterative solvers applied to the specific case of multibody dynamics. Another interesting avenue of further work is to use continuous collision detection to avoid the artifacts of explicit timestepping with large timestep sizes.

REFERENCES

ANITESCU, M. AND POTRA, F. A. 1996. Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementary problems. Tech. Rep. No 93/1996, Department of Mathematics, The University of Iowa.

ARMSTRONG, W. W. AND GREEN, M. W. 1985. The dynamics of articulated rigid bodies for purposes of animation. *Visual Comput. 1*, 4, 231–240.

BARAFF, D. 1989. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *Comput. Graph. 23*, 3, 223–232.

BARAFF, D. 1994. Fast contact force computation for nonpenetrating rigid bodies. *Comput. Graph. 28*, 23–34.

BARAFF, D. 1995. Interactive simulation of solid rigid bodies. *IEEE Comput. Graph. Appl. 15*, 3 (May), 63–75.

CAMERON, S. 1990. Collision detection by four–dimensional intersection testing. *IEEE Trans. Robotics Automa. 6*, 3, 291–302.

CHATTERJEE, A. AND RUINA, A. 1998. A new algebraic rigid body collision law based on impulse space considerations. *J. Appl. Mechanics*.

COTTLE, R., PANG, J.-S., AND STONE, R. E. 1992. *The Linear Complementarity Problem*. Academic Press.

EBERLY, D. 2007a. Dynamic collision detection using oriented bounding boxes. Online Paper. Magic Software, Inc.

EBERLY, D. 2007b. Intersection of objects with linear and angular velocities using oriented bounding boxes. Online Paper. Magic Software, Inc.

ERLEBEN, K. 2005. Stable, robust, and versatile multibody dynamics animation. Ph.D. thesis, Department of Computer Science, University of Copenhagen, Denmark.

FEATHERSTONE, R. 1998. *Robot Dynamics Algorithms*, 2nd ed. Kluwer Academic Publishers.

GLEICHER, M. 1994. A differential approach to graphical manipulation. Ph.D. thesis, Carnegie Mellon University.

GOYAL, S., RUINA, A., AND PAPADOPOULOS, J. 1989. Limit surface and moment funktion descriptions of planar sliding. In *Proceedings of the IEEE International Conference on Robotics and Automation*. Scottsdale, AZ, 794–799.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. 2003. Nonconvex rigid bodies with stacking. *ACM Trans. Graph.* (July).

HAHN, J. K. 1988. Realistic animation of rigid bodies. *Comput. Graph. 22*, 299–308.

HUBBARD, P. M. 1993. Interactive collision detection. In *Proceedings of the IEEE Symposium on Research Frontiers in Virtual Reality*. 24–32.

JEAN, M. 1999. The non-smooth contact dynamics method. *Comput. Methods Appl. Mechanics Engin. 177*, 3–4 (July) 235–257.

KAUFMAN, D. M., EDMUNDS, T., AND PAI, D. K. 2005. Fast frictional dynamics for rigid bodies. *ACM Trans. Graph. 24*, 3, 946–956.

LACOURSIERE, C. 2003. Splitting methods for dry frictional contact problems in rigid multibody systems: Preliminary performance results. In *The Annual SIGRAD Conference*. M. Ollila, Ed, Vol. 10.

MILENKOVIC, V. J. AND SCHMIDL, H. 2001. Optimization-based animation. *SIGGRAPH Conference*.

MIRTICH, B. 1996. Impulse-based dynamic simulation of rigid body systems. Ph.D. thesis, University of California, Berkeley.

MOORE, M. AND WILHELMS, J. 1988. Collision detection and response for computer animation. In *Comput. Graph. 22*, 289–298.

MORAVANSZKY, A. 2004. A path to practical rigid body dynamics. Annual CISP Workshop, Copenhagen, Denmark.

MOREAU, J. 1999. Numerical aspects of the sweeping process. *Comput. Methods Appl. Mechanics Engin. 177*, 3–4 (July), 329–349.

MURTY, K. G. 1988. *Linear Complementarity, Linear and Nonlinear Programming*. Helderman-Verlag.

OPENTISSUE. Opensource Project, Physics-based Animation and Surgery Simulation, www.opentissue.org.

PFEIFFER, F. AND WÖSLE, M. 1996. Dynamics of multibody systems containing dependent unilateral constraints with friction. *J. Vibration Control 2*, 161–192.

REDON, S. 2004. Continuous collision detection for rigid and articulated bodies. *ACM SIGGRAPH Course Notes*. To appear.

REDON, S., KHEDDAR, A., AND COQUILLART, S. 2003. Gauss least constraints principle and rigid body simulations. In *Proceedings of IEEE International Conference on Robotics and Automation*.

REDON, S., KIM, Y. J., LIN, M. C., AND MANOCHA, D. 2004. Fast continuous collision detection for articulated models. In *Proceedings of ACM Symposium on Solid Modeling Applications*. To appear.

RENOUF, M., ACARY, V., AND DUMONT, G. 2005. Comparison of algorithms for collisions, contact and friction in view of real-time applications in multibody dynamics. In *Proceedings of the International Conference on Advances in Computational Multibody Dynamics (ECCOMAS Thematic Conference)*.

RENOUF, M. AND ALART, P. 2004. Gradient type algorithms for 2d/3d frictionless/frictional multicontact problems. In *ECCOMAS'04*.

SAUER, J. AND SCHÖMER, E. 1998. A constraint-based approach to rigid body dynamics for virtual reality applications. *ACM Symposium on Virtual Reality Software and Technology*, 153–161.

SCHMIDL, H. AND MILENKOVIC, V. J. 2004. A fast impulsive contact suite for rigid body simulation. *IEEE Trans. Visualiz. Comput. Graph. 10*, 2, 189–197.

STEPHANE REDON, A. K. AND COQUILLART, S. 2002. Fast continuous collision detection between rigid bodies. *Comput. Graph. Forum (Eurographics'02) 21*, 3.

STEWART, D. AND TRINKLE, J. 1996. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *Int. J. Numeric. Methods Engin.*

STEWART, D. E. 2000. Rigid-body dynamics with friction and impact. *SIAM Rev. 42*, 1, 3–39.

TRINKLE, J. C., TZITZOUTIS, J., AND PANG, J.-S. 2001. Dynamic multi-rigid-body systems with concurrent distributed contacts: Theory and examples. *Philosoph. Trans. Mathemat. Phys. Engin. Sci. 359*, 1789 (Dec.), 2575–2593.

Received February 2006; accepted December 2006