Optimization-Based Animation[*]

Victor J. Milenkovic[†] Harald Schmidl[‡]

University of Miami, Department of Computer Science
P.O. Box 248154, Coral Gables, FL 33124-4245
http://www.cs.miami.edu

Abstract

Current techniques for rigid body simulation run slowly on scenes with many bodies in close proximity. Each time two bodies collide or make or break a static contact, the simulator must interrupt the numerical integration of velocities and accelerations. Even for simple scenes, the number of discontinuities per frame time can rise to the millions. An efficient optimization-based animation (OBA) algorithm is presented which can simulate scenes with many convex three-dimensional bodies settling into stacks and other "crowded" arrangements. This algorithm simulates Newtonian (second order) physics and Coulomb friction, and it uses quadratic programming (QP) to calculate new positions, momenta and accelerations strictly at frame times. Contact points are synchronized at the end of each frame. The extremely small integration steps inherent to traditional simulation techniques are avoided. Non-convex bodies are simulated as unions of convex bodies. Links and joints are simulated successfully with bi-directional constraints. A hybrid of OBA and retroactive detection (RD) has been implemented as well. A review of existing work finds no other packages that can simulate similarly complex scenes in a practical amount of time.

CR Categories: G.1.6 [Numerical Analysis]: Optimization—*Linear Programming, Quadratic Programming Methods*; I.3.5 [Computer Graphics]: Computational Geometry and Object Modeling—*Physically Based Modeling* I.3.7 [Computer Graphics]: Three-Dimensional Graphics and Realism—*Animation* I.6.8 [Simulation and Modeling]: Types of Simulation—*Animation*

Keywords: Animation, Animation w/Constraints, Physically Based Animation, Physically Based Modeling, Scientific Visualization, Solid Modeling.

1 Introduction

The principles of rigid body simulation have been studied intensively, especially during the last decade when computing power became more available at affordable cost.

Recent research in this area has generated modern techniques, which have been implemented in commercial software packages.

[*]This work was supported by NSF grant NSF-CCR-97-12401
[†]vjm@cs.miami.edu
[‡]harald@cs.miami.edu

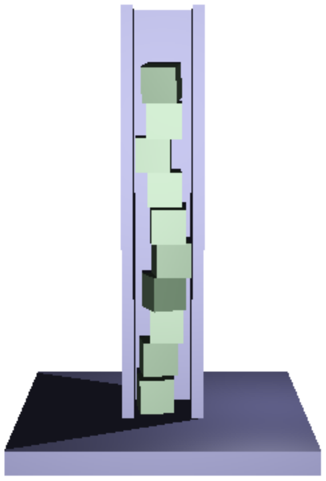

Figure 1: Simulating stacked bodies is known to be difficult.

Major areas of application are 3D tools for graphics designers and for the computer game industry. A very creative group of users is the movie industry. Computer graphics and animation have enabled film-makers to achieve unprecedented levels of realism in complex scenes for which conventional techniques of cinematography would be impossible or too expensive. Simulation techniques also help in manufacturing and other areas for which it is desirable to visualize an irreversible process before it is applied in real life.

The outstanding quality of some of today's animations can be seen as an experimental proof that existing techniques work. Yet there are aspects which suggest improvement. Closer examination reveals that performance problems arise in current algorithms when the number of bodies in a simulation grows. Stacks of bodies (Fig. 1) are the canonical example where many techniques "go down on their knees" computationally. This paper suggests a novel approach for rigid body simulation. It is able to handle scenes with several hundreds of concurrent contact points per frame and large numbers of solids in a single cluster.

Many researchers have developed excellent simulation techniques for various applications. Generally, these techniques must satisfy the need for physical realism and numerical stability, yet they should be computationally efficient. We feel that stability is an important feature of any simulation algorithm. Probabilistic reasoning suggests that in a scene with many interacting solids, almost all frames will have degeneracies in the contact geometry. One cannot ignore bad or hard to compute cases. Baraff has a series of papers on different issues found in rigid body simulation [2, 4, 3]. Barzel and Barr [6] propose a constraint-based solution. Mirtich describes a method of conservative advancement in an impulse-based simulator [19] and a parallelizable timewarp algorithm with which he was able to simulate an avalanche of hundreds of rocks [21]. Milenkovic's position-based physics simulates large numbers of highly interacting non-rotating spheres in an hourglass by minimizing gravitational energy using linear programming [18]. Stewart and Trinkle [25] and similarly Sauer and Schömer [24] show how a linear complementarity (LCP) approach can be employed to simulate realistic motion.

Some of these techniques are specialized for particular domains. Deeper analysis makes it apparent that there is no general algorithm to solve the forward simulation problem due to the difficulties that are involved. Impulse physics is stalled by small time steps in crowded scenes with many contact points. Even for simple scenes, the number of steps per frame time can rise to the millions. Constraint-based approaches occasionally fail due to solution non-existence. Position-based physics cannot simulate elastic collisions. LCP still has the notion of simulation integration steps that are smaller than the frame time. Although it is specifically designed to address the small time-step problem, timewarp cannot avoid it if all bodies form one large contact group. In general, efficiency suffers in these techniques due to small integration time steps and computations that are done between frames. Huge amounts of computations are used to simulate motion that the human eye does not even see.

It is true that the micro-steps taken in-between frames affect the final outcome of a simulation. However, in many cases, it is perfectly sufficient to simulate *plausible* motion. The OBA method presented here allows the use of a fixed time step, which we set equal to the frame time. Bodies follow their Newtonian trajectories (second order physics). Collisions are resolved by enforcing non-overlap constraints. We use optimization to compute new body positions. For all contact points, impulses and contact forces are computed with Coulomb friction. We have implemented a simulator, and the results are visually convincing. The method is well suited to simulate large stacks of bodies with a high number of static contacts.

The following section gives a brief introduction to rigid body dynamics and simulation methods. Section 3 describes the OBA method, and Section 4 presents experimental results and some implementation details.

2 Simulation Basics

In order to provide a good basis for the discussion of our work, this section briefly reviews simulation basics. Experts in the field may skim Section 2.1 to acquaint themselves with the notation in our paper.

2.1 General Rigid Body Dynamics

Any book on theoretical mechanics, e.g. Goldstein [11], provides the principles of three-dimensional rigid body motion. It is convenient to express some body properties in a fixed coordinate system that rests in the body's center of mass. Bodies have linear momentum $\mathbf{p}$ and angular momentum $\mathbf{l}$, with associated velocities $\mathbf{v}$ and ω. The relation between them is established by the mass m, $\mathbf{p} = m\mathbf{v}$, and the inertia tensor $\mathbf{I}$, $\mathbf{l} = \mathbf{I}\omega$. Similarly, we have linear acceleration $\mathbf{a}$ and angular acceleration α for the bodies. The body translation with respect to the world coordinate origin is denoted by $\mathbf{x}$, and the rotation matrix $\mathbf{R}$ gives the orientation of the body with respect to the body's fixed system. The net position $\mathbf{q}_{\text{world}}$ in world coordinates of a point $\mathbf{q}_{\text{body}}$ in body coordinates on the body is calculated according to $\mathbf{q}_{\text{world}} = \mathbf{R}\mathbf{q}_{\text{body}} + \mathbf{x}$. It is sometimes convenient to express the orientation as a three-dimensional rotation vector $\mathbf{r}$. The vector $\mathbf{r}$ points along the rotation axis and has magnitude $|\mathbf{r}|$ equal to the angle of rotation in radians counterclockwise about the rotation axis.

These are the basics of rigid body physics as they are applied in rigid body simulation. The body momenta determine the body trajectory. Body momenta change when impulses $\mathbf{j}$ or forces $\mathbf{f}$ act on the body. Impulses directly and instantaneously change the momenta, whereas forces and their rotational counterpart torques act over time. We adopt the terms *collisions* and *static contacts* for discontinuities. Collisions are contacts with negative normal velocity, i.e. interpenetration. They are resolved with impulses to instantaneously change the contact normal velocity to non-penetrating (receding). Forces act at static contacts with zero (or near zero in practice) normal velocity. Contact forces must prevent the static contacts from accelerating into each other, i.e. the contact normal acceleration must be equal to or greater than zero. The laws of physics are modeled and implemented by any physical simulator. The following section summarizes these simulators.

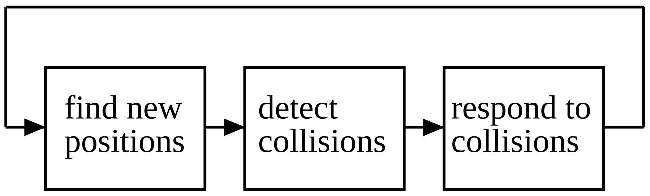

Figure 2: The simulation loop.

2.2 Simulation Methods

There are different types of simulators, which are sometimes specialized to simulate a certain kind of physical system, such as particles, solids, fluids, deformable objects like cloth, unique systems without rotation *etc*. [2, 4, 19, 18, 5, 23]. Most current simulators have in common a basic modular construction. One module calculates new positions as a body follows its trajectory according to Newtonian physics. Another module must detect if any pair of bodies overlaps at any point of the trajectory. A third module resolves these collisions.

The heart of the simulator is a loop (Fig. 2) that processes the bodies' states repeatedly to generate snapshots of individual frames. When played back, those frames constitute a computer-simulated movie that displays a system of bodies in motion. In the simulation loop, physical body states are integrated forward in time. If a collision between a pair of bodies occurs, i.e. their computed distance diminishes to zero, the simulation is stopped, and impulses and forces are applied to resolve collisions and static contacts. At each full frame time, a redraw command is issued and a snapshot of the new body states is displayed.

When the number of bodies in a scene becomes large or if the bodies are tightly confined, the number of collisions rises. More collisions require more iterations of the simulation loop to simulate the same duration of motion. Actually, a scene does not have to be very complex to have many collisions: put a single elastic solid into a container just σ larger. The number of collisions between frames is proportional to $1/\sigma$ times the frame time. As $\sigma \to 0$, the number of collisions between frames goes to infinity. Imagine a ping-pong paddle being brought down over a bouncing ball on a table, causing the ball to bounce increasingly faster. This "bad" case occurs even when only a few bodies settle into a stack: each middle body is caught in a shrinking "container" formed by the body above and the body below. Hence, the admissible time step goes to zero and thus slows the simulation. For example, an admissible time step of 10^{-6} sec. requires 33,333 iterations of the simulation loop to advance the system for just one frame of $1/30$ sec.

Mirtich gave a very comprehensive summary of the problems with traditional techniques in [21]. In the same paper, he presented a parallelizable timewarp algorithm that uses contact groups and desynchronizes the simulation loop for all bodies. Particularly well suited for multiprocessor implementation, the algorithm makes use of discrete properties in rigid body simulation to achieve a speed up. This has been demonstrated to work excellently for a large number of bodies as long as they form multiple contact groups. If all bodies are stacked and are therefore members of essentially one group, the performance gain can no longer be achieved, as the motion of the individual solids cannot be desynchronized. We will now introduce the structure of our method and its individual components.

3 OBA Simulation of Rigid Bodies

Our proposed paradigm, optimization-based animation (OBA), also follows the modular construction of other simulators. It consists of three stages to update positions, momenta, and forces. Position update replaces the first two modules in Figure 2. It finds new body positions but also provides the contact points at the new positions. Momentum and force computation resolve collisions and static contacts at the new positions. A Coulomb friction model is applied. Each of the stages is implemented as a quadratic programming problem.

OBA uses a fixed time step Δt, which we set equal to the frame time ($1/30$ sec.). For each step, each body has a *target* position and orientation, which is where its trajectory would take it in one time step in the absence of collisions. An optimization algorithm moves the bodies *as close as possible* to their target positions and orientations under the constraint that bodies cannot overlap. If bodies overlap at their targets, the optimization translates and rotates them the minimal amount to get rid of the overlap. The overall motion of bodies must be physical within limits where physical realism is actually visible.

In crowded scenes, traditional simulators take extremely many micro-steps, and thus they require huge amounts of computation and simulate motion that is not visible in the final video. Our OBA technique can compute plausible visible states at frame times without executing all the steps in-between. It tends to align neighboring solids so they have multiple contact points. In essence, all collisions and static contacts that would have occurred during a time step are *synchronized* at the end of each time step. On the other hand, body motion is *desynchronized* as each body can advance as much as possible until the next frame. Hence, it becomes tractable to simulate large numbers of bodies in close proximity.

OBA alters the positions of bodies and the times at which they collide. Consequently, it is not suitable for applications that require microscopically accurate motion. However, it can be employed in situations where a plausible realistic motion is sufficient. Chenney and Forsyth [8] have shown plausible motion to be useful for animation. Animating dancing beverage cans is different from simulating the lunar module landing on the moon. While the accuracy of the latter might affect a person's life span, any plausible motion of the former should be sufficient. For crowded scenes in which bodies bounce multiple times between frames, the human eye has no chance of accurately predicting where a body will be at the next frame time. In this case, a physically plausible solution is as reasonable as true physics. Fortunately, a physically plausible solution is computationally much less expensive than true physics.

3.1 Position Update

Milenkovic's *position-based physics* used linear programming to update the positions of bodies [18]. This work was a generalization of an algorithm for translational *compaction* of two-dimensional part layouts [14]. He has also generalized the compaction algorithm to allow rotations in two dimensions [17]. However, the position update algorithm we present here is not simply a three-dimensional generalization of the rotational compaction algorithm. In contrast to his position-based approach, our bodies elastically collide (bounce), are subject to friction, and follow Newtonian (parabolic) trajectories. We therefore achieve a higher level of physical realism. To do so, we do not minimize gravitational energy in the position update, but instead we set up an *artificial* energy potential that attracts solids to where they want to be at the next frame time: their *target* positions (and orientations). At time t_{cur}, all bodies are at their (non-overlapping) positions in the current frame. The *target* position and orientation of a body is the location where the body would be at time $t_{\text{tgt}} = t_{\text{cur}} + \Delta t$, where Δt is the frame time, if it followed its Newtonian trajectory in the absence of the other bodies. Therefore, two bodies $\mathbf{A}$ and $\mathbf{B}$ may be overlapping at the target time t_{tgt}, and this is what we need to prevent from happening. The next section describes a non-linear constraint to keep bodies separated. Section 3.1.2 describes possible positive-definite objectives to attract bodies to their target positions. Section 3.1.3 shows how to linearize the non-overlap constraint. Section 3.1.4 gives an algorithm using iterated quadratic programming to minimize the objective under the non-linear separation constraint.

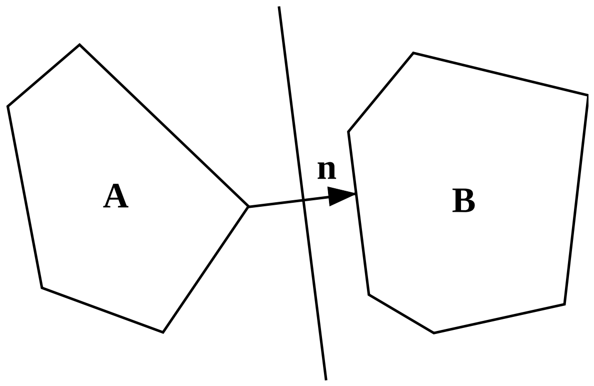

Figure 3: Separating plane between bodies $\mathbf{A}$ and $\mathbf{B}$ with normal vector $\mathbf{n}$.

3.1.1 Separating Plane Constraints

Suppose that the system has k convex polyhedral bodies. Two convex bodies do not overlap if and only if there exists a separating plane between them (Fig. 3).[1] Specifically, convex bodies $\mathbf{A}$ and $\mathbf{B}$ do not overlap if and only if there exists a unit vector $\mathbf{n}$ and scalar d such that,

$$\forall \mathbf{q}_a \in \mathbf{A}, \mathbf{n} \cdot \mathbf{q}_a \leq d \quad \text{and} \quad \forall \mathbf{q}_b \in \mathbf{B}, \mathbf{n} \cdot \mathbf{q}_b \geq d. \qquad (1)$$

Geometrically, $\mathbf{n}$ is the unit vector perpendicular to the separating plane, and d is the plane's distance from the origin. Since the bodies are polyhedral, it suffices to check Equation 1 for *vertices* $\mathbf{q}_a$ and $\mathbf{q}_b$ of $\mathbf{A}$ and $\mathbf{B}$, respectively. The separating plane constraint is non-linear because both $\mathbf{n}$ and $\mathbf{q}_a$ (or $\mathbf{q}_b$) are variables.

Each body starts at a position $\mathbf{x}^{\text{cur}}$ and orientation $\mathbf{R}^{\text{cur}}$ in the current frame. The vector $\mathbf{x}^{\text{tgt}}$ and orientation matrix $\mathbf{R}^{\text{tgt}}$ denote its target position: where the body wants to be in the next frame if overlaps were ignored. If no pair of bodies overlaps at the targets, we just set each position $\mathbf{x}$ and orientation $\mathbf{R}$ equal to $\mathbf{x}^{\text{tgt}}$ and $\mathbf{R}^{\text{tgt}}$. However, if some pairs are overlapping at the targets, this must be resolved. First, all such potentially overlapping pairs of bodies are detected. We can employ any efficient collision detection algorithm, such as I-COLLIDE [9], V-Clip [20] or SWIFT [10].

[1]Two-dimensional figures are sometimes used for simplicity.

Currently, we use bounding boxes for pre-sorting and an exact distance finding algorithm based on Lin-Canny [15]. It uses witnesses and Voronoi regions to exploit coherence. If the distance found is below a threshold and declared as overlapping, we insert the pair into a list of close pairs.

3.1.2 Objective

This section defines the objective used in the OBA position update. This objective is a positive-definite quadratic function of the bodies' current positions and orientations and their target positions and orientations. Since Equation 1 is non-linear, we will calculate a perturbed position $\mathbf{x}^{\text{per}}$ and orientation $\mathbf{R}^{\text{per}}$. Initially, $\mathbf{x} = \mathbf{x}^{\text{cur}}$ and $\mathbf{R} = \mathbf{R}^{\text{cur}}$. After each perturbation, we set $\mathbf{x} = \mathbf{x}^{\text{per}}$ and $\mathbf{R} = \mathbf{R}^{\text{per}}$. The variables $\Delta\mathbf{x}$ and $\Delta\mathbf{R}$ denote the perturbations, where

$$\Delta\mathbf{x} = \mathbf{x}^{\text{per}} - \mathbf{x} \quad \text{and} \quad \Delta\mathbf{R} = \mathbf{R}^{\text{per}}\mathbf{R}^{-1}. \tag{2}$$

The *target perturbations* are the perturbations which would move the body to its target:

$$\Delta\mathbf{x}^{\text{tgt}} = \mathbf{x}^{\text{tgt}} - \mathbf{x} \quad \text{and} \quad \Delta\mathbf{R}^{\text{tgt}} = \mathbf{R}^{\text{tgt}}\mathbf{R}^{-1}. \tag{3}$$

Let $\Delta\mathbf{r}$, $\Delta\mathbf{r}^{\text{per}}$, and $\Delta\mathbf{r}^{\text{tgt}}$ denote the vector representation for the orientation matrices $\Delta\mathbf{R}$, $\Delta\mathbf{R}^{\text{per}}$, and $\Delta\mathbf{R}^{\text{tgt}}$, respectively. Define the *target deficits* $\Delta\Delta\mathbf{x}$ and $\Delta\Delta\mathbf{r}$ as follows:

$$\Delta\Delta\mathbf{x} = \Delta\mathbf{x} - \Delta\mathbf{x}^{\text{tgt}} \quad \text{and} \quad \Delta\Delta\mathbf{r} = \Delta\mathbf{r} - \Delta\mathbf{r}^{\text{tgt}}. \tag{4}$$

These represent the distance from the current perturbations to the target perturbations. When they are zero (vectors), the body is at its target position.

We want to move each body as close as possible to its desired target position, i.e. minimize its distance to the target. The diameter D of a body is the largest distance between two vertices on the body. The objective for k bodies is

$$\sum_{i=1}^{k} \Delta\Delta\mathbf{x}_i \cdot \Delta\Delta\mathbf{x}_i + D_i^2 \Delta\Delta\mathbf{r}_i \cdot \Delta\Delta\mathbf{r}_i, \tag{5}$$

where the subscript i refers to that property of the ith body. We have to weight the rotational part by D so that it has the same units as the translational part. A rotation $\Delta\mathbf{r}$ moves a point $\mathbf{q}$ on the body by at most a distance $|\Delta\mathbf{r}|\,|\mathbf{q}|$, and D is an upper bound on $|\mathbf{q}|$.

We also tested the following objective:

$$\sum_{i=1}^{k} m_i \Delta\Delta\mathbf{x}_i \cdot \Delta\Delta\mathbf{x}_i + \Delta\Delta\mathbf{r}_i^T \mathbf{I}_i \Delta\Delta\mathbf{r}_i. \tag{6}$$

The linear and angular parts are weighted by the body mass and inertia respectively. This way, a heavier body pushes a lighter body out of the way. One might argue this makes more sense physically. Both methods work well in practice with slight differences in simulation stability in favor of (5). If the bodies in a scene vary greatly in size, shape, and density, (6) might be a better choice.

Note: we use a quadratic objective because a linear objective can perturb the bodies in a highly unrealistic manner. To see why, consider a simple one-dimensional scene with two unit line segment objects centered at $x = 0$. If x_1 and x_2 represent the translations of these objects, the non-overlap constraint is $x_2 - x_1 \geq 1$. Minimizing the quadratic objective $x_1^2 + x_2^2$ yields $(x_1, x_2) = (-0.5, 0.5)$: each object moves 0.5 unit. If we use a non-quadratic objective such as $|x_1| + |x_2|$, then $(x_1, x_2) = (-1, 0)$ or $(x_1, x_2) = (0, 1)$ are both equally valid solutions which minimize the objective. While $(x_1, x_2) = (-0.5, 0.5)$ is also a valid solution for this linear objective, the simplex method for solving linear programs (LP) will *not* generate it, because the simplex method always chooses an extreme (vertex) solution of the feasible solution space. Instead, the simplex method will generate an "unnatural" solution for which one object sits still and pushes the other one unit to the side, and the choice of the "winning" object might vary arbitrarily from frame to frame. The quadratic objective, corresponding to an artificial "energy" potential, yields positions which are more natural and stable.

3.1.3 Linearizing the Separation Constraints

This section describes how to linearize the separation constraints in Equation 1. Let $\mathbf{n}$ and d denote a separating plane for body $\mathbf{A}$ at $\mathbf{x}_a, \mathbf{R}_a$ and $\mathbf{B}$ at $\mathbf{x}_b, \mathbf{R}_b$. If $\mathbf{A}$ does not touch $\mathbf{B}$, then $\mathbf{n}$ points in the direction of a shortest line segment from $\mathbf{A}$ to $\mathbf{B}$. Let $\mathbf{n}$, $\mathbf{n}_x$, and $\mathbf{n}_y = \mathbf{n} \times \mathbf{n}_x$ form an orthonormal basis. Vector $\mathbf{n}^{\text{per}}$ is the perturbed version of $\mathbf{n}$ with perturbation,

$$\Delta\mathbf{n} = \mathbf{n}^{\text{per}} - \mathbf{n} \approx \delta_x \mathbf{n}_x + \delta_y \mathbf{n}_y, \tag{7}$$

where δ_x and δ_y are scalars. (For small perturbations of unit vector $\mathbf{n}$, $\Delta\mathbf{n}$ is almost perpendicular to $\mathbf{n}$.) Since d has no non-linear terms, it just appears as a variable.

If $\mathbf{q}_a^{\text{body}}$ is the body coordinates of a vertex $\mathbf{q}_a$ on body $\mathbf{A}$, then its unperturbed and perturbed world coordinates are

$$\mathbf{q}_a = \mathbf{R}_a \mathbf{q}_a^{\text{body}} + \mathbf{x}_a \quad \text{and} \quad \mathbf{q}_a^{\text{per}} = \mathbf{R}_a^{\text{per}} \mathbf{q}_a^{\text{body}} + \mathbf{x}_a^{\text{per}}.$$

Combining these with Equation 2 yields,

$$\mathbf{q}_a^{\text{per}} = \Delta\mathbf{R}_a(\mathbf{q}_a - \mathbf{x}_a) + \mathbf{x}_a^{\text{per}}. \tag{8}$$

We linearize this formula using the small-angle approximation $\Delta\mathbf{R}\mathbf{w} \approx \mathbf{w} + \Delta\mathbf{r} \times \mathbf{w}$:

$$\mathbf{q}_a^{\text{per}} \approx \mathbf{q}_a + \Delta\mathbf{x}_a + \Delta\mathbf{r}_a \times (\mathbf{q}_a - \mathbf{x}_a). \tag{9}$$

Plugging Equations 7 and 9 into Equation 1 ($\mathbf{n}^{\text{per}} \cdot \mathbf{q}^{\text{per}} \leq d$),

$$(\mathbf{n} + \delta_x \mathbf{n}_x + \delta_y \mathbf{n}_y) \cdot (\mathbf{q}_a + \Delta\mathbf{x}_a + \Delta\mathbf{r}_a \times (\mathbf{q}_a - \mathbf{x}_a)) \leq d. \tag{10}$$

In this inequality, the variables (unknowns) are $\Delta\mathbf{x}_a$, $\Delta\mathbf{r}_a$, δ_x, δ_y, and d. We carry out the multiplication, throw away quadratic terms, move all terms with variables to the left and terms with constants to the right, and apply the identity, $\mathbf{a} \cdot (\mathbf{b} \times \mathbf{c}) = -(\mathbf{a} \times \mathbf{c}) \cdot \mathbf{b}$. This yields the linearized constraint,

$$\begin{array}{lcl} \mathbf{n} \cdot \Delta\mathbf{x}_a - (\mathbf{n} \times (\mathbf{q}_a - \mathbf{x}_a)) \cdot \Delta\mathbf{r}_a & & \\ +(\mathbf{n}_x \cdot \mathbf{q}_a)\delta_x + (\mathbf{n}_y \cdot \mathbf{q}_a)\delta_y - d & \leq & -\mathbf{n} \cdot \mathbf{q}_a. \end{array} \tag{11}$$

Similarly, for a vertex $\mathbf{q}_b$ on body $\mathbf{B}$ from the second part of Equation 1,

$$\begin{array}{lcl} \mathbf{n} \cdot \Delta\mathbf{x}_b - (\mathbf{n} \times (\mathbf{q}_b - \mathbf{x}_b)) \cdot \Delta\mathbf{r}_b & & \\ +(\mathbf{n}_x \cdot \mathbf{q}_b)\delta_x + (\mathbf{n}_y \cdot \mathbf{q}_b)\delta_y - d & \geq & -\mathbf{n} \cdot \mathbf{q}_b. \end{array} \tag{12}$$

3.1.4 Iterative Update

For each body, the initial values of $\mathbf{x}$ and $\mathbf{R}$ are $\mathbf{x}^{\text{cur}}$ and $\mathbf{R}^{\text{cur}}$. After each perturbation, we set $\mathbf{x} = \mathbf{x} + \Delta\mathbf{x}$ and $\mathbf{R} = \Delta\mathbf{R}\,\mathbf{R}$, where $\Delta\mathbf{R}$ is the matrix format of $\Delta\mathbf{r}$. To calculate $\Delta\mathbf{x}$ and $\Delta\mathbf{r}$, we solve a QP. Each body has twelve variables, the components of $\Delta\mathbf{x}$, $\Delta\Delta\mathbf{x}$, $\Delta\mathbf{r}$, and $\Delta\Delta\mathbf{r}$. Each close pair has three variables, δ_x, δ_y, and d. Each body requires six linear equality constraints to enforce Equation 4. Each close pair has the two linear inequality constraints of Equations 11 and 12. The objective is given by Equation 5 or 6. **Note**: we do not use the computed values of δ_x and δ_y to update $\mathbf{n}$. Instead, the separating plane normal is calculated from the new positions and orientations of the bodies.

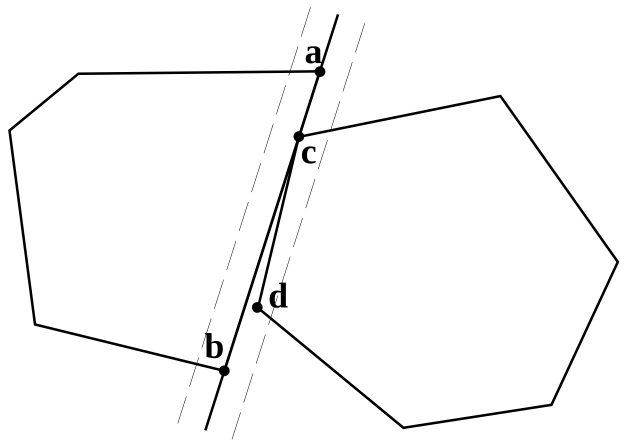

Figure 4: Explanation of critical vertices.

The objective is nicely quadratic and positive-definite, but the original non-overlap constraints are non-linear and non-convex. Solving this particular optimization is NP-hard in general. However, after linearizing our constraints, we can solve a series of QPs that converge to a local minimum of the objective. The original objective (3.1.2) and the linearized constraints (3.1.3) yield a QP, which when solved produces perturbations on the current positions that move the system towards a local minimum of the objective. When the system reaches a local minimum, the perturbation goes to zero (numerically), and the algorithm stops the iteration. Since the last perturbation is zero, the small-angle approximations and other linearizations become exact, and thus the system is non-overlapping at a local minimum.

3.1.5 Reckless Dynamic Update

In order to do the position update efficiently, we have to keep the size of each QP as small as possible. Each constraint in the QP refers to a vertex on a body and a separating plane with respect to another body. We cannot simply add constraints for every vertex with respect to every separating plane. Instead, we employ Milenkovic's reckless dynamic update approach [17]. For each close pair of bodies, we first find *critical vertices* and add constraints (Equation 11 or 12) only for these. Critical vertices are those which lie within a slab of the separating plane at the current positions. In Figure 4, only one true contact is present, but four vertices ($\mathbf{a}$, $\mathbf{b}$, $\mathbf{c}$ and $\mathbf{d}$) lie within a critical slab.

This approach is indeed reckless because a non-critical vertex might move past the separating plane and result in an overlap. However, this vertex is added as a critical vertex[2] for the next iteration, and thus we "learn" from our mistakes. Usually, the overlap is not too bad and solving the next QP will remove it. There is a chance, however, for the algorithm to paint itself into a corner. If it gets stuck in an overlapping configuration which it cannot resolve, it will produce an *infeasible* QP. If this happens, we roll back a perturbation (of *all* the bodies). In theory, it might be necessary to roll all the way back to the initial non-overlapping (and thus feasible) starting configuration of the current frame (although this never happens in practice). **Note**: even if the algorithm rolls back, it maintains the current set of close pairs and critical vertices; otherwise, it would keep repeating the same mistake. The pseudo-code in Table 1 summarizes the position update algorithm. In the algorithm, a close pair $P = \langle \mathbf{A}, \mathbf{B}, C \rangle$ consists of a body $\mathbf{A}$, a body $\mathbf{B}$, and a set C of critical vertices for that pair. Once a vertex of $\mathbf{A}$ or $\mathbf{B}$ is added to C, it is never removed. Similarly, close pairs are added to but never removed from a set S of close pairs.

[2] A vertex is critical if it is on the wrong side of the separating plane even if it is not in the slab.

```
Input:   current and target positions for each body
Output:  new current position for each body

$S \leftarrow \emptyset$
repeat
    find current close pairs and add to $S$
    for all close pairs $P = \langle \mathbf{A}, \mathbf{B}, C \rangle \in S$
        calculate the separating plane $\langle \mathbf{n}, d \rangle$ for $\mathbf{A}$ and $\mathbf{B}$
        find current critical vertices of $\mathbf{A}$ and $\mathbf{B}$ with respect
          to $\langle \mathbf{n}, d \rangle$ and add them to $C$
    solve QP
    if infeasible
        rollback all body positions
    else
        update the current positions
while QP was infeasible or
     critical vertices were added or
     objective was improved
```

Table 1: Position update algorithm.

3.1.6 Bounds

Because of the linearization, the solution to an individual QP may result in positions and orientations that slightly overlap the solids, even if all close pairs and critical vertices are known. However, each subsequent QP re-asserts the non-overlap constraints. If the iterated QPs converge, then they must converge to a non-overlapping configuration. Hence, the position update preserves the non-overlap constraint. **Note**: in the overlapping case, the "separating plane" normal $\mathbf{n}$ points in the direction that $\mathbf{B}$ could translate, to most swiftly eliminate the overlap (equivalently, $-\mathbf{n}$ points in the direction that $\mathbf{A}$ could translate, to most swiftly eliminate the overlap). The same formula for the distance d of the separating plane from the origin,

$$d = \frac{1}{2}\left(\max_{\mathbf{q}_a \in A} \mathbf{n}\cdot\mathbf{q}_a + \min_{\mathbf{q}_b \in B} \mathbf{n}\cdot\mathbf{q}_b\right), \tag{13}$$

works in all three cases: separated, touching, overlapping.

Due to the linearized constraints, it is possible that the solution to a QP would be so badly overlapped that the next QP would be infeasible. If all critical vertices are known, then even rolling back will not help. In practice, for each iteration we limit the coordinates of each body's $\Delta\mathbf{r}$, the perturbation of the orientation, to lie within the interval $[-0.1, +0.1]$ in radians. This has so far prevented an infeasible configuration and has ensured convergence to a local minimum of the objective.

3.2 Collisions and Static Contact Response

Section 3.1 described how to construct and use QPs to find feasible solutions for the positions of solids in animation. When all current positions are updated, resulting contact points are determined and resolved with appropriate bounces and forces. Our position update has the tendency to align bodies, and therefore we will have many simultaneous contact points. A feature (vertex, edge, or face) of $\mathbf{A}$ and a feature of $\mathbf{B}$ touch the separating plane, and these features intersect at a point, line segment, or convex polygon. That point, the endpoints of the line segment, or the vertices of that polygon are the contact points. Under this definition, two cubes can have up

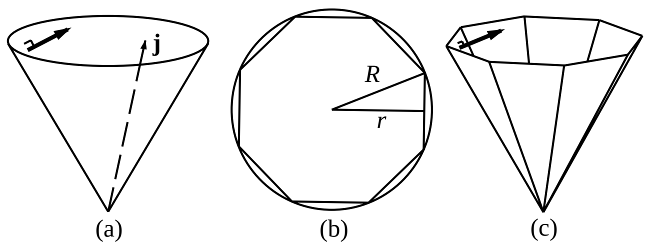

Figure 5: Friction cone and regular octagonal approximation. Ratio r/R of inner to outer radius is $\cos\pi/8$.

to eight contact points if one is rotated and stacked on top of the other. In practice, a tolerance must be used to determine if a feature lies "on" the separating plane, and the features must be projected onto the separating plane before intersection.

We can use the contact points as input to any analytic algorithm [12, 22, 7, 2, 3, 19] for calculating the new body velocities (momenta), and accelerations (forces). In this manner, the bodies in a scene will continue to follow their trajectories. However, this section presents new optimization-based (in particular, QP-based) algorithms to update velocities and accelerations of contacting bodies with friction. It is for most cases vital to model friction, because it is such a substantial part of reality. Without friction in the real world, every object would be even harder to pick up than a wet piece of soap.

3.2.1 Momentum Update

Although it is true that any analytic algorithm for momentum calculation would work, we feel that our QP-based solution for a simultaneous impact model with Coulomb friction is very well suited for our needs. A simultaneous model fits in well with the resulting contact geometry, since there will be so many simultaneous contact points due to the synchronization of collisions and static contacts after the position update. Our experiments with a propagation impact model were also satisfactory. We opted for simultaneous impulses to avoid the high computational effort for dealing with large numbers of collisions in a propagation model. As mentioned by Baraff [2], we also experienced that the computational effort for propagation models could be over 90% of the running time.

Among others, Brach [7] has investigated collisions with Coulomb friction. Let $\mathbf{q}$ be a contact point for bodies $\mathbf{A}$ and $\mathbf{B}$, and let $\mathbf{n}$ be the normal to the separating plane. The vectors $\mathbf{n}$, $\mathbf{n}_x$, and $\mathbf{n}_y$ (Section 3.1.3) form a right-handed *collision frame*. The normal $\mathbf{n}$ points up along the cone axis in Figure 5a, and c, and the coordinate origin is at the cone tip. The two parameters for our impulses are the friction coefficient μ and the coefficient of restitution ϵ. Both can be understood as material constants. Any collision impulse $\mathbf{j}$ must satisfy the Coulomb friction law, stating that the tangential frictional impulse at a contact point is (at most) μ times the non-negative (no "stickiness") normal component,

$$(\mathbf{n}_x \cdot \mathbf{j})^2 + (\mathbf{n}_y \cdot \mathbf{j})^2 \le \mu^2 (\mathbf{n} \cdot \mathbf{j})^2 \quad \text{and} \quad \mathbf{n} \cdot \mathbf{j} \ge 0. \tag{14}$$

This constraint is convex but non-linear. It confines $\mathbf{j}$ to a *friction cone* whose axis is $\mathbf{n}$ and which has slope $1/\mu$ (Fig. 5a).[3] To linearize, we approximate this cone by an eight-sided inscribed polygonal cone whose sides have slope $(\mu\cos\pi/8)^{-1} > \mu^{-1}$ (Fig. 5b and c). Inward pointing normal vectors to these sides are

$$\mathbf{u}_h = \mu\cos\frac{\pi}{8}\mathbf{n} + \cos\frac{\pi h}{4}\mathbf{n}_x + \sin\frac{\pi h}{4}\mathbf{n}_y, \tag{15}$$

[3]The right circular cone $z^2 = s^2(x^2 + y^2)$ has slope s.

for $h = 0, 1, 2, \ldots, 7$.[4] The impulse $\mathbf{j}$ lies inside the linearized cone if and only if it lies inside all these sides:

$$\mathbf{u}_h \cdot \mathbf{j} \ge 0, \qquad \text{for} \qquad h = 0, 1, 2, \ldots, 7. \tag{16}$$

We calculate the (scalar) relative contact normal velocity $v(\mathbf{q})$ of bodies $\mathbf{A}$ and $\mathbf{B}$ at contact $\mathbf{q}$ as [2],

$$v(\mathbf{q}) = \mathbf{n} \cdot ((\mathbf{v}_b + \omega_b \times (\mathbf{q} - \mathbf{x}_b)) - (\mathbf{v}_a + \omega_a \times (\mathbf{q} - \mathbf{x}_a))). \tag{17}$$

The impulse $\mathbf{j}$ is applied positively to $\mathbf{B}$ and negatively to $\mathbf{A}$. It acts on the bodies linearly by adding it to the bodies' linear momenta. In the same fashion, the resulting torsional impulse $(\mathbf{q} - \mathbf{x}) \times \mathbf{j}$ is added to the body's angular momenta. Hence, the impulse $\mathbf{j}$ at contact $\mathbf{q}$ between bodies $\mathbf{A}$ and $\mathbf{B}$ adds

$$m_b^{-1}\mathbf{j} \qquad \text{and} \qquad \mathbf{I}_b^{-1}((\mathbf{q} - \mathbf{x}_b) \times \mathbf{j}) \tag{18}$$

to $\mathbf{v}_b$ and ω_b and subtracts

$$m_a^{-1}\mathbf{j} \qquad \text{and} \qquad \mathbf{I}_a^{-1}((\mathbf{q} - \mathbf{x}_a) \times \mathbf{j}) \tag{19}$$

from $\mathbf{v}_a$ and ω_a.

Let v^- denote the contact normal velocity $v(\mathbf{q})$ (Equation 17) before any impulses are applied and let v^+ denote the contact normal velocity $v(\mathbf{q})$ after all the impulses are applied and $\mathbf{v}_a$, ω_a, $\mathbf{v}_b$, and ω_b have been updated. If the bodies were not really colliding ($v^- \ge 0$) before the collision, then they must still not be colliding after the collision, else they must semi-elastically "bounce" [2]:

$$\text{if } v^- \ge 0 \text{ then } v^+ \ge 0 \text{ else } v^+ \ge -\epsilon \cdot v^-. \tag{20}$$

Using quadratic programming, we simultaneously solve for each impulse $\mathbf{j}$ at each contact. The QP has six variables per body: the components of $\mathbf{v}$ and ω. It has four variables per contact: the components of $\mathbf{j}$ and the collision normal velocity v^+. It has six equality constraints per body to express how each body's $\mathbf{v}$ and ω change as a result of impulses according to Equations 18 and 19. It has one equality constraint per contact to relate v^+ to the updated body velocities according to Equation 17. It has eight inequalities per contact to constrain $\mathbf{j}$ to the linearized friction cone according to Equation 16. Finally, it has one more linear inequality per contact point to implement the bounce according to Equation 20.

The objective of the QP is the total kinetic energy, which is a positive definite quadratic function of the $\mathbf{v}$'s and ω's. Without this objective, the QP's linear constraints permit energy to *increase*. We assume that the physical system satisfies all the constraints and also converts as much kinetic energy to heat as possible. Minimizing the kinetic energy moves a slowly moving system to rest and thus results in a very stable simulation.

3.2.2 Force or Acceleration Calculation

After collisions are resolved with bounces, resulting static contacts ($v^+ = 0$) must have contact forces computed. As with the momentum update, any algorithm for contact force calculation can be employed, e.g. Baraff's pivoting scheme [3]. It has been stated [16] that problems with the mathematics of static contacts with friction make determining the right contact force sometimes hard. Problems can have more than one solution or can be unbounded.

Our initial goal was to develop a robust QP algorithm that avoids problems with un-boundedness, but also avoids heuristic pivoting strategies. Contact forces and resulting torques change a body's linear acceleration $\mathbf{a}$ and its angular acceleration α respectively. If we know these accelerations, we can calculate the contact acceleration similarly to the contact velocity (Equation 17). We again

[4]For convenience, $\mathbf{u}_h$ is not unit-length, but its $\mathbf{n}_x, \mathbf{n}_y$ component is.

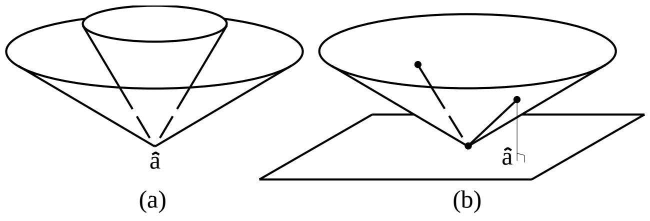

Figure 6: Acceleration cone vs. friction cone. If $\mathbf{a}$ lies on side of cone, new $\hat{\mathbf{a}}$ is projection to $\mathbf{n}_x, \mathbf{n}_y$ plane.

use the collision frame from Section 3.1.3. Let $\mathbf{a}(\mathbf{q})$ be the relative contact acceleration of bodies $\mathbf{A}$ and $\mathbf{B}$ at contact point $\mathbf{q}$:

$$\mathbf{a}(\mathbf{q}) = (\mathbf{a}_b + \alpha_b \times (\mathbf{q} - \mathbf{x}_b)) - (\mathbf{a}_a + \alpha_a \times (\mathbf{q} - \mathbf{x}_a)). \quad (21)$$

In the frictionless case, the only additional constraint is the bodies do not accelerate into each other,

$$\mathbf{n} \cdot \mathbf{a}(\mathbf{q}) \geq 0. \quad (22)$$

We formulate a QP. It has six variables per body: the components of $\mathbf{a}$ and α. It has three variables per contact point: the components of $\mathbf{a}(\mathbf{q})$. Equation 21 adds three linear equality constraints per contact point. Equation 22 adds one linear inequality constraint per contact point. We investigated the use of several quadratic objectives, and came up with the following. For each body, plug $\mathbf{a}$ in place of the $\mathbf{x}$ in the formula for potential energy. Plug $\mathbf{a}$ and α in place of $\mathbf{v}$ and ω in the formula for kinetic energy. For k bodies, take the sum,

$$\sum_{i=1}^{k} -m_i \mathbf{g} \cdot \mathbf{a} + \frac{1}{2} m_i \mathbf{a} \cdot \mathbf{a} + \frac{1}{2} \alpha_i^T \mathbf{I}_i \alpha_i. \quad (23)$$

Our experiments indicate that solving this QP gives a plausible set of accelerations for the frictionless case. **Note**: the contact forces are implicit and do not appear as variables.

Given our success with the frictionless case, we tried to come up with an artificial frictionless system which modeled the system with friction. We wanted a solution with implicit forces that satisfied Coulomb's law. This led us to the following. First, for each body, set $\mathbf{a} = -\Delta t^{-1} \mathbf{v}$ and $\alpha = -\Delta t^{-1} \omega$. This is the acceleration that will bring all bodies to a halt in one time-step $\Delta t = 1/30$ sec. For each contact $\mathbf{q}$, plug these values into Equation 21 to calculate $\mathbf{a}(\mathbf{q})$. Set $\hat{\mathbf{a}}(\mathbf{q}) = \mathbf{a}(\mathbf{q})$. Add to the system an *acceleration cone* constraint (Fig. 6a): find a solution for $\mathbf{a}(\mathbf{q})$ which lies in a cone of slope μ with axis $\mathbf{n}$ and tip $\hat{\mathbf{a}}(\mathbf{q})$. This acceleration cone (outer cone in Figure 6a) is perpendicular to the friction force cone (inner cone in Figure 6a).

Since this acceleration cone is curved, the acceleration cone constraint on each $\mathbf{a}(\mathbf{q})$ is non-linear, and we cannot solve the system using quadratic programming. Suppose for the moment that we can solve this system. Figure 6b indicates the the possible types of position for the contact acceleration $\mathbf{a}(\mathbf{q})$: at the tip, in the interior (Fig. 6b, left contact), or on the side (Fig. 6b, right contact). The tip case corresponds to a contact which feels a force satisfying Coulomb's law, although this force is implicit. The interior case corresponds to a contact which has broken free. The side case is non-physical: the contact has both positive normal force and acceleration. In the side case, we move $\hat{\mathbf{a}}(\mathbf{q})$ directly "underneath" $\mathbf{a}(\mathbf{q})$ (Fig. 6b), i.e. we project it straight down onto the $\mathbf{n}_x$, $\mathbf{n}_y$ plane:

$$\hat{\mathbf{a}}(\mathbf{q}) = \mathbf{a}(\mathbf{q}) - (\mathbf{n} \cdot \mathbf{a}(\mathbf{q}))\mathbf{n}. \quad (24)$$

Then we solve again. We repeatedly adjust $\hat{\mathbf{a}}(\mathbf{q})$ and solve until there are no side cases. Each time we move a cone, we give the

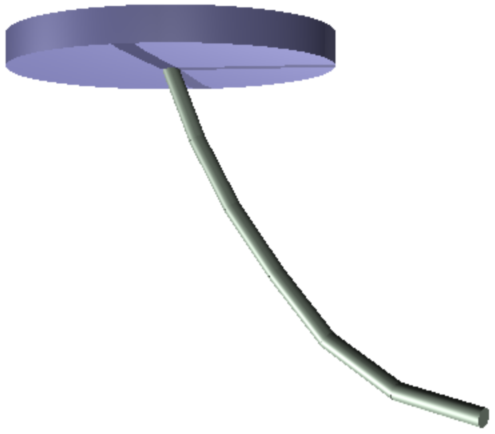

Figure 7: Simple multi-body pendulum made of six sticks.

system a bit more local freedom: $\mathbf{a}(\mathbf{q})$ was "pressed up against the side" and then the tip of the cone is moved beneath it in the direction of the "pressure". Therefore, each system will have a smaller value of the objective (23) and this iteration must converge.

In practice, we must linearize the acceleration cone constraint as in the impulse calculation: for $h = 0, 1, 2, \ldots, 7$,

$$\left(\mu^{-1} \cos \frac{\pi}{8} \mathbf{n} + \cos \frac{\pi h}{4} \mathbf{n}_x + \sin \frac{\pi h}{4} \mathbf{n}_y\right) \cdot (\mathbf{a}(\mathbf{q}) - \hat{\mathbf{a}}(\mathbf{q})) \geq 0. \quad (25)$$

We add to the QP these eight linear inequality constraints per contact and solve. For each resulting $\mathbf{a}(\mathbf{q})$, we want strict equality for each constraint in Equation 25 (all $= 0$) (tip case) or we want strict inequality for each constraint (all > 0) (interior case). For each contact $\mathbf{q}$ with mixed inequalities (some $= 0$ and some > 0) (side case), we move $\hat{\mathbf{a}}(\mathbf{q})$ directly "underneath" $\mathbf{a}(\mathbf{q})$ (Equation 24). Then we solve a new QP. We repeatedly adjust $\hat{\mathbf{a}}(\mathbf{q})$ and solve the resulting QP until there are no mixed inequalities.

Unfortunately, the method often requires many iterations to converge and is therefore not superior to pivoting as we had hoped. However, we found it to produce very good results if we used a limit of three iterations. Some of the contact forces/accelerations are non-physical but not visibly so.

After having now introduced the components of our OBA algorithm, we would like to summarize its overall structure in pseudocode:

```
for each frame
    update positions and find contact points
    apply impulses
    calculate new body accelerations
```

3.3 Joints

We handle joints (Fig. 7) or, more generally, links between bodies, as bi-directional constraints. There is a good introduction to multi-body physics in [1]. A joint connects two bodies in a way such that they can revolve around the joint, but they cannot stretch it out. In our algorithm, two linked bodies are not regarded as a close pair. Instead, we add a simple constraint to our position update algorithm to ensure that points on bodies where they are linked together are perturbed to the same coordinate in space:

$$\mathbf{q}_a + \Delta \mathbf{x}_a + \Delta \mathbf{r}_a \times (\mathbf{q}_a - \mathbf{x}_a) = \mathbf{q}_b + \Delta \mathbf{x}_b + \Delta \mathbf{r}_b \times (\mathbf{q}_b - \mathbf{x}_b), \quad (26)$$

where $\mathbf{q}_a = \mathbf{q}_b$ is the common link point of bodies $\mathbf{A}$ and $\mathbf{B}$. This way, they will stay connected. Attachment impulses are calculated as in [22] to simulate realistic motion of multi-bodies.

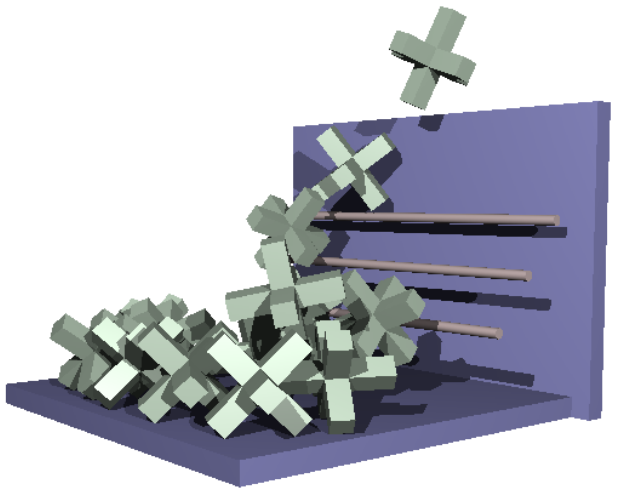

Figure 8: *Jacks*: an example of non-convexity.

3.4 Non-Convex Solids

Non-convex bodies (Fig. 8) are implemented as collections of convex parts. Any polyhedron can be expressed as the union of convex polyhedra. These can be rigidly attached to each other simply by giving them all the same position. All constraints must be constructed with respect to the convex components. The rest of the calculations can be made with respect to the non-convex solids.

3.5 Hybrid Algorithm

Pure OBA is not suited for extremely fast bodies moving more than their own width in a single frame time. When two solids approach each other at very high velocity, their target positions might be highly overlapping, or they may even go completely past each other. In this case, the OBA method chooses very unrealistic positions for the solids. We implemented a hybrid algorithm of OBA and RD to solve this problem.

The hybrid algorithm takes as input a maximum $c_{\max}$ on the number of times each pair of bodies may collide and a minimum collision velocity $v_{\min}$. The algorithm first runs the standard RD algorithm with the following modification. If a pair of bodies has collided $c_{\max}$ times or if the pair collides with velocity less than $v_{\min}$, then the modified algorithm does not enqueue the collision as an event. In other words, that pair of bodies no longer "sees" each other. This modified RD is run until the next frame time. The output is a set of possibly overlapping positions. The hybrid algorithm then runs the OBA position update step with these positions as targets.

First, note that the hybrid algorithm is the same as pure OBA if $c_{\max} = 0$ (or $v_{\min} = \infty$): bodies simply follow their Newtonian trajectories without regard to collisions. For $c_{\max} = \infty$, the hybrid algorithm is the same as pure RD. For finite but non-zero $c_{\max}$, the algorithm is a compromise between OBA and RD. The modified RD only executes the loop in Figure 2 a number of times equal to $c_{\max}$ times the number of colliding pairs.

By specifying $c_{\max}$ and $v_{\min}$, we can perform a simulation that is sufficiently realistic yet is as computationally efficient as possible. The more we allow a pair of bodies to bounce between frames, the higher the physical realism will be. Unfortunately also the running time will rise. Allowing fast bodies to bounce before OBA picks up will ensure that they do not just pass through each other. Instead, they will bounce "a few times" until they reach a point where our optimization method is realistic enough to generate a plausible motion. For all slow collisions, OBA is a good choice to start with. By selecting $v_{\min}$, we avoid any extra computation for these pairs of bodies.

4 Experiments and Results

Our experiments are set up to reflect the advantages of OBA. We examine the challenge in simulating stacks of bodies or other arrangements where bodies are very densely packed. When contacts become static and when all bodies in a simulation form essentially one contact group as in a stack, our approach becomes very useful.

The *stack* simulation simply shows how 10 cubes get stacked on top of each other in a vertical shaft until they come to a complete rest. *Cubejam* shows a large group of cubes in contact as they squeeze around obstacles. It resembles somewhat a winding river. In *wall*, we see how two blast waves cause a wall to collapse. The *jacks* simulation shows an example with non-convex bodies. Each jack is a collection of three convex boxes. The effective number of bodies in the scene is therefore three times the number of jacks. *Pendulum* is a simple example of linked bodies, and the *hybrid* scene shows a simulation where a mix of OBA and RD was used. The bodies here are tiny enough to pass through each other if the penetration velocity is large. However, with the hybrid method, we have a reliable way to prohibit just that. In this particular simulation, we bounced each pair of bodies one time until OBA takes over. Note how the cubes get wedged between the thin walls. The *hourglass* scene (actually "minute" glass) has 1000 spheres without friction. In our last scene, we combined traditionally animated bodies with dynamic simulation. *Robot* has an animated robot and a claw interacting with a pool of convex and non-convex bodies.

Table 2 gives information about the complexity of the scenes and provides insight into running time and efficiency related data. We achieve good performance: the time spent to calculate one frame stays reasonable even for scenes with many solids, collisions, and static contacts. The *pendulum* scene illustrates that OBA can handle links. Although the *pendulum* is not very complicated, the running time is 3x real time. This is due to the computational overhead in solving small QPs. Our experiments show that OBA has its strength in solving large and crowded systems rather than small systems with few collisions. As expected, the *hybrid* method has a higher running time than the otherwise similarly complex *cubejam*, since it makes partial use of RD. The *hourglass* scene illustrates that OBA runs fast even for 1000 bodies. It is frictionless but, unlike position-based physics [18], it has bouncing and parabolic trajectories. Our experiments show that our iterated position QP method does not require excessive numbers of QPs to be solved. Also the number of rollbacks due to infeasibilities which result from constraint linearization is mostly zero. *Robot* is an exception here. The robot and claw move on paths given by the animator. These paths are not necessarily physical. They were generated by a human and can potentially lead to infeasibilities. Therefore, the number of rollbacks is higher as is the number of QPs solved. The algorithm needs to do more optimization work to accommodate non-overlap constraints when animated bodies are present. All scenes are visually extremely stable. Stability is an important feature of any simulator.

The complexity of the algorithm is governed by the position update. It takes up over 50% of the running time. The momentum and force calculation each take up about 20-25% of the running time. The remainder goes into other tasks like maintaining data structures and general bookkeeping. Each of the three stages of our algorithm solves QPs, and the running time is determined by how fast we can solve these QPs. Typically, this time depends on the number of constraints m in a QP. Momentum and force updates have m perfectly proportional to the number of contacts per frame. In the position update, m is proportional to the number of close pairs. Finding the close pairs is minor as far as running time goes. Each solid can have up to a constant number of neighbors depending on the body geom-

	#solids	close pairs/fr.	collisions/fr.	contacts/fr.	#frames	sec./fr.	avg. #qp/fr.	rollbacks [%]
Stack	10	6.6	29	25.7	600	0.9	3	0
Cubejam	100	172	411.7	278.4	1500	22.2	4.8	0
Wall	90	129.2	404.3	220.6	600	12.3	4	0
Jacks	50(150)	123.3	167.1	94	1500	15.5	4.6	0.03
Pendulum	6	0	0	0	1000	0.1	2.1	0
Hybrid	100	159.1	518.7	497.6	1500	25.9	4.8	0.07
Hourglass	1000	1723.4	1673.6	1673.6	2000	13.5	2.6	0
Robot	306	308	507.7	507.7	580	75.5	6	2.7

Table 2: Complexity of scenes and efficiency issues.

etry. The physics of crystals provides theory and different models for such tightest packing problems.

It is therefore clear that the overall running time will depend on the particular scene, the shape of its bodies, and how crowded the bodies are. It is hard to compare the *stack* simulation with the *jacks*. Our *wall* can be simulated much faster than *cubejam*, although the number of solids is only moderately smaller. This is due to the difference in number of contacts and close pairs. During large parts of the *wall* simulation, the bricks are flying through the air, and interaction is rather dynamic. There are not so many contact points. The *jacks* have a lower number of contacts overall. This is at first surprising, but a closer look reveals that the jacks in our video end up in a very unordered fashion like tumbleweed. In many instances, a pair of jacks will have as few as three contact points. However, a pair of cubes has in most instances at least four and up to eight contact points. The two simulations *jacks* and *wall* still run in comparable time since there are multiple convex components in each jack. *Robot* has a relatively high number of close pairs, collisions, and contacts. These and the higher number of iterations in the position update are responsible for its higher simulation time per frame.

For these reasons, we ran a separate simulation to analyze the efficiency of OBA. We dropped 200 cubes in packages of nine at every ten frames into a vertical shaft from a small height. The cubes settle into a stable, tightly packed arrangement. We record the time that it takes to solve each QP, the number of constraints in the QP, the number of close pairs or contacts, and the number of iterations. We verified that the position update takes the majority of the running time. Since position update is done by iterating QPs, the time spent is number of iterations times the time for solving each individual QP. The number of iterations varies slightly with the number of close pairs, but not in a strictly monotone fashion: overall and on average, it seems to grow very slowly with the number of close pairs. The number of close pairs is proportional to the number of bodies, and therefore the number of constraints in a QP is also proportional to the number of bodies. The time to solve a QP depends heavily on the complexity of the problem at hand.

The theoretical time to solve a QP is a polynomial of high degree, if the objective is positive semi-definite, as it is in our case. However, CPLEX software uses a variety of techniques to obtain good running times in practice. In our case, we verified that the time to solve an individual QP using CPLEX is roughly $\mathrm{O}(m^2)$, with m the number of constraints. This running time has occasional outliers that take four to five times as long as predicted.

The number of iterations grows extremely slowly with the number of close pairs n. There is no clear pattern, but it seems safe to assume it is not more than $\mathrm{O}(\log n)$. The number n of close pairs is bounded, and dependent on the body geometry and complexity of the scene. We mentioned that there is proportionality between m and n, and thus also the number m of constraints is bounded. Essentially, we expect an overall running time of $\mathrm{O}(n^2)$. Figure 9 shows the total time spent in the position update versus the number of close pairs n. For larger numbers of close pairs, the graph exhibits some noise. This is due to somewhat unpredictable changes in the number of iterations. At any rate, the number of iterations stays reasonable. It is mostly as low as 2-4, and it goes only in some cases up to 8. Another more important reason for noise are the QPs which are outlying regarding their solution time.

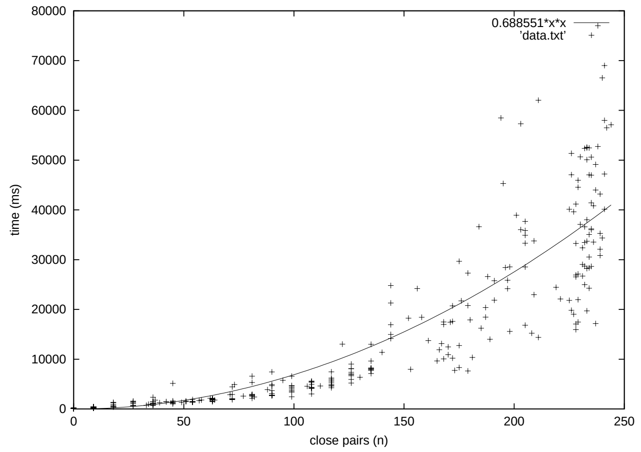

Figure 9: Solution time of one QP vs. number of close pairs for the position update.

There is a possibility to improve the running time by decomposing the position update. It has been investigated for two-dimensional polygonal compaction problems [13], and we intend to generalize this work to three dimensions. The decomposition of a large problem into smaller sub-problems can even be done for scenes where the bodies form a single cluster, and a speed-up can be achieved even in a sequential uni-processor environment. Distributing the sub-problems in a parallel fashion further improves running time.

We would like to provide some implementation details at this point. All simulations use the CPLEX 7.0 run-time library to solve quadratic programs. All examples except the *hourglass* were coded in Java on a 450MHz Pentium II processor running Windows NT. The *hourglass* was implemented in C++ on a 933MHz Pentium III running Redhat Linux 7.0. Java programs have been reported to be as fast or almost as fast for arithmetical operations as compiled languages. The major performance differences are to be found in output routines, such as for drawing. Our simulator uses these output routines very sparingly, and we do not see a significant performance disadvantage in using Java for some of our experiments. The main reason for the better simulation speed of *hourglass* can be seen in the absence of friction in this simulation, and in the particular nature of the contact geometry of spheres. Two spheres have only one contact point, whereas two cubes can have up to eight.

We used the usual value for gravity $|\mathbf{g}| = 10$ m/sec^2 throughout our experiments. The coefficient of restitution for the hourglass was $\epsilon = 0.7$ and for all other simulations was $\epsilon = 0.3$. Except for the frictionless *hourglass*, the Coulomb friction coefficient was $\mu = 0.3$. The bodies have density 0.8 g/cm^2. These values correspond to some "normal" types of wood, and changing them has no effect on the performance of the algorithm.

5 Conclusions and Future Work

We presented a new method for animation of large systems of convex bodies. OBA is efficient, very stable, and realistic where it matters within visual limits. It can be perfectly employed for scenes where *plausible* animations are adequate. Bodies follow Newtonian trajectories, and optimization makes it possible to simulate stacks of many bodies or otherwise "crowded" scenes. Stacks are the canonical example where traditional simulation techniques have problems, and we feel that our method is superior in this particular application. We have shown that we can handle links and non-convexity, and we have devised a hybrid method that allows a trade-off between speed and physical accuracy. The beauty of mathematical programming software is that it has been well researched, and is readily available. People who are not necessarily experts in the field of mathematical programming can make use of it in their programs as we have done. We feel that further development of OBA should point towards a parallel implementation.

Acknowledgments

The authors would like to thank Adam McMahon for his creative input, and expertise in traditional animation for generating the video.

References

[1] William W. Armstrong and Mark W. Green. The dynamics of articulated rigid bodies for purposes of animation. *The Visual Computer*, 1:231–240, 1985.

[2] David Baraff. Analytical methods for dynamic simulation of non-penetrating rigid bodies. *SIGGRAPH 89 Conference Proceedings*, pages 223–232, 1989.

[3] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. *SIGGRAPH 94 Conference Proceedings*, pages 23–34, 1994.

[4] David Baraff and Andrew Witkin. Dynamic simulation of non-penetrating flexible bodies. *SIGGRAPH 92 Conference Proceedings*, pages 303–308, 1992.

[5] David Baraff and Andrew Witkin. Large steps in cloth simulation. *SIGGRAPH 98 Conference Proceedings*, pages 43–52, 1998.

[6] Ronen Barzel and Alan H. Barr. A modeling system based on dynamic constraints. *SIGGRAPH 88 Conference Proceedings*, pages 179–187, 1988.

[7] Raymond M. Brach. Rigid body collisions. *Journal of Applied Mechanics*, 56:133–138, March 1989.

[8] Steven Chenney and D.A. Forsyth. Sampling plausible solutions to multi-body constraint problems. *SIGGRAPH 00 Conference Proceedings*, pages 219–228, 2000.

[9] Jonathan C. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav K. Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scaled environments. *Symposium on Interactive 3D Graphics*, pages 189–196, 1995.

[10] S. A. Ehmann and M. C. Lin. SWIFT: Accelerated proximity queries between convex polyhedra by multi-level voronoi marching. Technical report, Computer Science Department, University of North Carolina at Chapel Hill, 2000.

[11] Herbert Goldstein. *Klassische Mechanik*. AULA Verlag Wiesbaden, 11. Auflage, 1991.

[12] J.B. Keller. Impact with friction. *Journal of Applied Mechanics*, 53:1–4, March 1986.

[13] Iraklis Kourtidis. Distributed rotational polygon compaction. Master's thesis, University of Miami, May 2000.

[14] Z. Li and Victor Milenkovic. Compaction and separation algorithms for non-convex polygons and their applications. *European Journal of Operations Research*, 84:539–561, 1995.

[15] Ming Lin and John Canny. A fast algorithm for incremental distance calculation. *International Conference on Robotics and Automation*, pages 1008–1014, 1991.

[16] Per Lötstedt. Coulomb friction in two-dimensional rigid body systems. *Zeitschrift für angewandte Mathematik und Mechanik*, 61:605–615, 1981.

[17] V. J. Milenkovic. Rotational polygon overlap minimization and compaction. *Computational Geometry: Theory and Applications*, 10:305–318, 1998.

[18] Victor J. Milenkovic. Position-based physics: Simulating the motion of many highly interacting spheres and polyhedra. *SIGGRAPH 96 Conference Proceedings*, pages 129–136, 1996.

[19] Brian Mirtich. Impulse-based simulation of rigid bodies. *Proceedings of the 1995 Symposium on Interactive 3D Graphics*, pages 181–189, 1995.

[20] Brian Mirtich. V-Clip: Fast and robust polyhedral collision detection. *Transactions on Graphics*, 17(3):177–208, 1998.

[21] Brian Mirtich. Timewarp rigid body simulation. *SIGGRAPH 00 Conference Proceedings*, pages 193–200, 2000.

[22] Mathew Moore and Jane Wilhelms. Collision detection and response for computer animation. *SIGGRAPH 88 Conference Proceedings*, pages 289–297, 1988.

[23] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. *SIGGRAPH 99 Conference Proceedings*, pages 137–146, 1999.

[24] Jörg Sauer and Elmar Schömer. A constraint-based approach to rigid body dynamics for virtual reality applications. *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, pages 153–162, 1998.

[25] David Stewart and Jeff C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *In Zeitschrift für Angewandte Mathematik und Mechanik*, 77(4):267–279, 1997.